



The Agile Software Tester

Software Testing in the Agile World

KC Martin

KC Martin

The Agile Software Tester

Revision 7

June 2021

Copyright © KC Martin (2014-2021)

The right of KC Martin to be identified as the author of this work has been asserted by him in accordance with section 77 and 78 of the Copyright, Designs and Patents Act 1988.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior permission of the publishers.

Any person who commits any unauthorized act in relation to this publication may be liable to criminal prosecution and civil claims for damages.

Table of Contents

[Preface](#)

[About the Author](#)

[Introduction](#)

[Foundations of Testing](#)

[The Testers Mindset](#)

[Types of Testing](#)

[Black Box Testing](#)

[White Box Testing](#)

[Static Testing](#)

[Serverless Testing](#)

[The Path to Agile](#)

[The Agile Process Framework](#)

[The five agile meetings](#)

[The Agile QA Tester](#)

[The Agile Organisation](#)

[Moving to agile and what to avoid](#)

[User Stories](#)

[Gherkin and Cucumber](#)

[Burndown Charts](#)

[Test Driven Vs Behaviour Driven](#)

[Automated Testing with Selenium](#)

[Selenium and ASP.Net](#)

[The first Selenium test](#)

[The Selenium Command Set](#)

[A Complete C# Example](#)

[A Complete Ruby Example](#)

[A Complete Java Example](#)

[Automated API testing in C#](#)

[Load testing with Jmeter](#)

[API testing with Postman](#)

[Structured Query Language](#)

[Stored Procedures](#)

[Continuous Integration and Deployment](#)

[CI/CD with Azure](#)

Conclusion
Agile Myths
Glossary
Further Reading
Useful download URL's

Preface

In the world today many forward-thinking, fast-moving organisations have adopted the agile software development framework fully with a carefully planned strategy and 100% company commitment. This leap of faith now means they are hopefully reaping the benefits gained from this strategy, however, there are still plenty of software companies out there who have, for one reason or another, not engaged in the framework. These companies still ignore the agile framework methodology or they have simply placed a task board in the centre of the office and stated ‘there, now we are agile’.

While it is true that the agile methodology is not for everyone and not every software development project is suited to the framework it is, however, the way forward for the majority of companies who are involved in modern software development.

As agile has grown in popularity and usage over the decades the amount of literature about the subject has also grown tremendously. Sadly, however, most of the books currently available on the market focus on the project management or the code development areas of the software development life cycle, there is still very little for the aspiring agile software quality assurance (QA) tester to read. In the agile world. Testing and the software QA are just as important as any other function or person and that is why I have written this book. Hopefully experienced and new QA’s alike will find some useful pointers within these humble pages which will help them enhance their career and enjoyment of testing software.

Test professionals involvement in agile projects remains challenging because of the very different nature of the agile methodology compared to older methodologies such as waterfall and the V Modal. This is also not helped by a level of misunderstanding about the true nature of agile that persists in many companies and deep-rooted prejudices aimed at QA’s by a very small percentage of programmers and project managers (*they are nothing more than failed programmers being a common misconception*).

Although many test professionals are succeeding in agile projects, many others continue to struggle to succeed and achieve their true potential that their skills and dedication deserve. QA’s who have spent many years testing

outside of agile can also often struggle to make the jump across from the waterfall methodology. However with quality training, good management and self-belief this jump can be completed, this is where this book comes in. This is edition five of The Agile Tester, in this edition I have updated most chapters and corrected some typing mistakes which you the readers have kindly made me aware of. If you find anymore in this version please let me know.

So do you want to be an agile quality assurance software tester?

Have you got what it takes?

Time to find out, read on and see.

Enjoy.

What you will need for this book

Chrome web browsers

Chrome Driver

Visual Studio Community Edition and/or Ruby Mine

A basic understanding of software testing

Who this book is for?

This book is for existing software QA's who wish to extend their skillset into the world of agile and automated software testing. The book is also for anyone who is considering entering this exciting, rewarding and challenging line of work.

Reader feedback

Feedback from my growing collection of readers is always welcome. Please let me know what you think about this book—what you liked or may have disliked. Constructive reader feedback is important for me to develop this publication further so that you do get the most out of it. To send me general feedback, simply send an e-mail to feedback@Keysbox.com, and mention the book title inside the subject of your message.

Errata

Although I have taken every care to ensure the accuracy of the content, mistakes do happen; I am after all only human. If you find a mistake in this publication—for example, a mistake in the text or the code—I would be grateful if you would report this to me. By doing so, you can save many other readers from frustration and help me improve subsequent versions of this book. If you find any errata, please report them by sending an e-mail to feedback@Keysbox.com and include the details of the errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website (www.keysbox.com), or added to any list of existing errata, under the Errata section of that title.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At keysbox.com we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of my works, in any form, on the Internet, please provide me with the location address or website name immediately so that I can pursue a remedy.

Please contact us at copyright@Keysbox.com with a link to the suspected pirated material. I appreciate your help in protecting my work, and my ability to bring you valuable content.

Questions

You can contact me at questions@Keysbox.com if you are having a problem with any aspect of the book, and I will do our best to address it.

About the Author



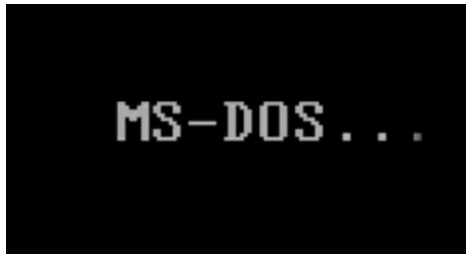
KC Martin is a professionally qualified Software Developer and Software QA with over twenty nine years' experience of agile, selenium, scrum, software testing, CI & CD, software design, support and installation. During his career in Information Technology, he has developed, maintained and tested applications using COBOL, C, C++, Visual Basic, C# and Java.

Kevin graduated from The University of Portsmouth in September 2009 with Master of Science in Software Engineering with Merit. More recently he has passed the Professional Scrum Master 1 examination and in October 2013 he also sat and passed the ISTQB Advanced Test Manager examination while in April 2014 he sat and passed the ISTQB Certified Agile Tester examinations.

Kevin started as a software programmer and has since moved into Quality Assurance and Software Testing. He has experienced the software development cycle from all angles and has used that experience to create this book which he hopes will help both new and experienced software QA's equally. Kevin's other interests also include running, keeping fit, rock music, sailing and cooking.

Introduction

Software development has changed dramatically over the past sixty or so years. In the early days of software development system memory was at a premium, coding was achieved in machine code and most programs were small and simple when compared to their modern counterparts. As a result, testing was considered a minor task for lesser mortals and a hindrance by most programmers.



Gradually as software development tools evolved, we entered the era of the DOS programmer. These programmers used development tools such as C, Ada and Pascal to write their beloved programs. They still considered testing a hindrance and an insult to their skills. Therefore, they tried to avoid it at any cost. It was, of course, a job for lesser mortals. Most software companies spent as little investment as possible on testing and most bugs tended to be reported by end-users as systems crashed and burned around the globe.



With the advent of Windows, a new level of complexity and confusion was added to the software mix. Suddenly not only did software development companies have to worry about their bugs but there were also the added concerns of just how stable the Windows operating systems was their clients were now using.

Windows and other GUI's have evolved tremendously over the past two decades, some versions have been much better than others, a prime example of a quickly dropped but never to be forgotten version will always be the infamous Windows ME. All these concerns have awoken most software houses to the real importance of high quality, technically adept, well-paid software testers (also known as QA's).

You see competent, well trained software QA's will also have a basic understanding of at least one programming language. They will also be confident with Structured Query Language (SQL) and the basics of computer architecture. They will be very methodical in their approach to work and the need for patience, concentration and tolerance is also very high. Let's face it, today good, high-quality, skilled programmers are everywhere, however, good talented software QA's are still relatively few and far between.

So important is software testing today that I will even go as far as to say that talented software QA's are a much more valuable asset than talented software developers. When I first made this statement, a few programmers refused to talk to me for a long time, some still do not.

Over the past twenty-five years, there has been a slow but sure change in software development strategies as users have come to accept web-based applications and now regard them as a normal part of their computing experience. No matter if it is leisure-based software such as Facebook or business related most modern applications are now either web based or have a web alternative to their more traditional windows-based versions.

The richness and complexity of web software are also increasing, however, the more complex a program is the more likely it has bugs somewhere in the tangle of code and the more important the testing role becomes. Some software developers get past the testing question by keeping their applications in a permanent Beta mode. This passes the testing mantle on to the actual end-user who, it would seem, is happy to test the product as they use it, usually because the software is very cheap or free.

For serious, business applications this is certainly not a viable option. Applications that offer business-related solutions such as job management and online ordering also need to offer a high level of reliability and integrity

to maintain customer support and confidence and avoid litigation when the system falls apart at the seams. This is where the software QA bravely steps in and saves the day. Today customers quite correctly expect more for their money and are more likely to demand compensation when things go wrong. Therefore, software houses now need to be more aware of the testing role.

As I have already said today the world is awash with good software programmers. Let's face it dear readers they are everywhere. Simply look under any mouse mat and a dozen highly skilled programmers will appear demanding extra strong coffee and a connection to World of Warcraft. The same cannot yet be said for good software QA's, not yet that is.

A programmer's aim in life is to construct what they will always consider being a flawless work of art. To them, their code is a perfect masterpiece and the best code to be found anywhere on the planet, and why not, that is where their skills are. Meanwhile, a QA will do his or her best to detect and highlight any errors in the code and the logical way it works, and why not, it is after all that they are paid to do. Despite what some people think good QA's do not do this to simply annoy and anger the programmers. Rather it is a serious attempt to capture as many bugs as possible before the product falls into the hands of the poor unsuspecting customer.

Therefore, the QA role is as important as any other member of the development team. So, let us look at the foundations of software testing and get an idea of where it all started back in the early days of software development.

Foundations of Testing

Today software is everywhere and in everything. It is now an integral part of human life. Software control's many aspects of day-to-day life from Traffic lights to the humble washing machine and down to the little smartwatch strapped to your wrist. No matter who you are or what you do and where you are today you will never be far from the humble little microchip. Like it or not they do control modern life and they control us.

All these systems must be conceived, designed, coded and (hopefully) tested to a very high standard; that is after all the software lifecycle in its most basic form. The software is designed and coded by human beings who will by their very nature make mistakes and get things wrong, no one is perfect, not even programmers (however only say that in front of them if you are feeling very brave or they are asleep).

Over the years there have been some very spectacular software failures, some of which you may have heard of already, others may be new to you. On the following pages are just a few classic well-known examples of what an extremely long and tragic list is now.

The Ashley Madison Hack (2015) - Ashley Madison, the online dating site, which helped married people cheat on their partners, was hacked. Much to the delight of some people and horror of some others, the hackers published the real names, email addresses and personal details of the dating site's users. The cost of this action was 2 lives, countless marriages, a £400 million class action suit, various extortion attempts, the company's reputation and the loss of their CEO.

Royal Bank of Scotland (2009) - The Royal Bank of Scotland experienced a fault on cyber-Monday that left its account holders unable to withdraw cash or make payments. In a public statement, the CEO of RBS admitted that the fault was unacceptable and that it was caused by the failure of RBS to invest in IT for decades. The problem left many Christmas shoppers unable to buy or pay for their purchases, which further hurt online retailers. As a result, all RBS customers were unable to access their money for a day.

The Toyota Recall (2009) - There have been three separate recalls by Toyota since 2009, and amazingly all the recalls have been related to the same thing, that is the accelerator sticking. First, in August 2009 a Lexus ES350 suddenly accelerated out of control at speeds estimated to exceed 100 mph. One of the passengers called 911 before they crashed and reported that the car had "no brakes." Sadly, all four passengers were killed when the car crashed. Next, in November 2009, Toyota dealers were instructed to remove and shorten the gas pedals and update the onboard computers with a new program that would override the electronic gas pedal

when the brake pedal was pressed. Eventually, Toyota ended up recalling more than 9 million cars worldwide in 2010, but it wasn't because of a mechanical issue. The cars all suffered from a software bug that caused a lag in the anti-lock brake system. Overall these recalls, legal costs and other consequences are thought to have cost Toyota over \$3 billion, a very costly bug indeed.

Ariane 5 Flight 501 (1996) – In 1996 Europe's newest unmanned satellite-launching rocket, the Ariane 5, reused previously working software from its predecessor, the Ariane 4 rocket. Unfortunately, the Ariane 5's faster; more powerful engines exploited a bug that was not realised or detected in previous models. The rocket took exception to this oversight and self-destructed 36.7 seconds into the maiden launch.

Mariner 1 (1962) - On July 22, 1962, the first spacecraft of NASA's Mariner program blasted off on a mission to fly by Venus. Initially, all looked good as the spacecraft happily headed towards outer space, but after a few minutes, the spaceship began to yaw off course. The guidance system failed to correct the trajectory, and guidance commands failed to correct it manually. As the rocket veered off toward the busy North Atlantic shipping lanes, the range safety officer did the only thing he could do, destroy the spacecraft. Eventually, the cause was nailed down to a mis-transcription of a single punctuation mark by an engineer.

The Mars Climate Orbiter Crash (1998) – This crash was eventually root caused back to a sub-contractor who had designed the

navigation system on the orbiter using imperial units of measurement instead of the metric system that was specified by NASA. As a result, the spacecraft attempted to stabilise its orbit too low within the Martian atmosphere and subsequently crashed into the ground.

Soviet Gas Pipeline Explosion (1982) - When the CIA (allegedly) discovered that the Soviet Union was (allegedly) trying to steal sensitive U.S. technology for its operation of their trans-Siberian pipeline, CIA operatives (allegedly) introduced a bug into the Canadian-built system that would pass Soviet inspection but ultimately fail when in operation. This caused the largest man-made non-nuclear explosion in the planet's history.

The Plague in World of Warcraft (2005) - The hugely successful World of Warcraft (WoW), an online computer game created by Blizzard Entertainment, suffered an embarrassing glitch following an update to their game on September 13, 2005 – causing mass (fictional) death. Following an update to the game content, a new enemy character, Hakkar, was introduced. This character could inflict a disease called Corrupted Blood upon the playing characters that would drain their health over a period. This disease could be passed from player to player, just as in the real world, and had the potential to kill any character contracting it. This effect was meant to be strictly localised to the area of the game that Hakkar inhabited.

However, one thing was overlooked: players were able to teleport to other areas of the game while still infected and pass the disease onto others – which is exactly what happened. I cannot find any figures on the body count, but entire cities within the game world were no-go areas, with dead player's

corpses littering the streets. Fortunately, player death is not permanent in WoW and the event was soon over when the administrators of the game reset the servers and applied further software updates. Particularly interesting is the way players reactions in the game could closely reflect their reactions to a similar real-life incident. While the deaths in the World of Warcraft were fictional this has not always been the case as the next example will show.

Therac-25 (1985-1987) - The Therac-25 was a machine for administering radiation therapy, generally for treating cancer patients. It had two modes of operation. The first consisted of an electron beam targeted directly at the patient in small doses for a short amount of time. The second aimed the electron beam at high energy levels at a metal 'target' first, which would essentially convert the beam into X-rays that were then passed into the patient.

In previous models of the Therac machine, for this second mode of operation, there were physical fail-safes to ensure that this target was in place as, without it, extremely high energy beams could be mistakenly fired directly into the patient. In the new model, these physical fail-safes were replaced by software ones.

Unfortunately, there was a bug in the software: an 'arithmetic overflow' sometimes occurred during automatic safety checks. This means that the system was using a number inside its internal calculations that were too big for it to handle. If at this precise moment, the operator was configuring the machine, the safety checks would fail, and the metal target would not be moved into place. The result was that beams 100 times higher than the intended dose would be fired into a patient, giving them radiation poisoning. This happened on 6 known occasions, causing the later death of 4 unfortunate patients.

Knight Capital Group's trading violations (2012) - In August 2012 Knight Capital Group Inc., which is one of America's largest trading firms, mistakenly sent out more than four million stock orders in less

than one hour. These orders should have been spread out over days—and reversing the trades cost almost half a billion dollars. Knight Capital would have been sent into bankruptcy had it not been for a group of investors that saved the day and came up with \$400 million. The problem was that when a code change was released it was not deployed to all the servers, one server was missed, and this caused the server to use old code to create millions of orders. As a result of this error, the firm's shares lost 75% in just two days after the faulty software flooded the market with unintended trades, sending dozens of stocks into spasms. The software bug caused over \$440 million in losses, which is almost four times what the company had made in 2011.

The Cold War Missile Crisis (September 26, 1983) -

Stanislav Petrov was the duty officer of a secret bunker near Moscow responsible for monitoring the Soviet early warning satellite system. Just after midnight, they received an alert that the US had launched five Minuteman intercontinental ballistic missiles. As part of the mutually assured destruction doctrine that came into prevalence during the Cold War, the response to an attack by one power would be a revenge attack by the other.

This meant that if the attack was genuine, they needed to respond quickly. However, it seemed strange that the US would attack with just a handful of warheads: although they would cause massive damage and loss of life, it would not be even nearly enough to wipe out the Soviet opposition. Also, the radar stations on the ground were not picking up any contacts, although these couldn't detect beyond the horizon because of the curvature of Earth, which could have explained the delay.

Another consideration was the early warning system itself, which was known to have flaws and had been rushed into service in the first place. Petrov weighed all these factors and decided to rule the alert as a false alarm.

Although Petrov did not have his finger on the nuke button as such, had he passed on a recommendation to his superiors that they take the attack as real, it could have led to all-out nuclear war. Whether based on experience, intuition, or just luck, Petrov's decision was the right one.

It was later determined that the early detection software had picked up the sun's reflection from the top of clouds and misinterpreted it as missile launches.

Famous Encryption Bugs

To fix a warning issued by Valgrind, a developer of Debian patched up OpenSSL and, in the process, broke the random number generator. This patch was uploaded in September 2006 and made its way into the official release. Somehow it was not reported until April 2008. Every key generated with the broken version is compromised (you see the "random" numbers were made easily predictable), as is all data encrypted with it. This threatened many applications that rely on encryption such as S/MIME, Tor, SSL or TLS protected connections and SSH.

Heartbleed, an OpenSSL vulnerability was introduced in 2012 and eventually disclosed in April 2014, removed confidentiality from affected services, causing among other things the shutdown of the Canada Revenue Agency's public access to the online filing portion of its website following the theft of social insurance numbers.

The Apple Computer "goto fail" bug was simply a duplicated line of code which caused a public key certificate check to pass a test incorrectly.

So, while most software bugs are annoying and short-lived others can have very large and serious repercussions on human life. Not only have software bugs caused death and severe injury in the most extreme cases they have also allowed security breaches to bank systems and government systems. No one knows how much money has been stolen by exploiting these glitches, but the total probably runs into trillions.

Therefore, software testing is very important; in fact, it is very, very important. As a result, good, motivated and highly trained software QA's are also a very important part of the software development team. In many

software developments companies, the QA used to be someone from the administration or the post room who was not very busy that week, thankfully most of the software companies who used this strategy have either gone out of business or have changed their methods. In these more advanced and enlightened times, QA's are usually held in more esteem.

So, what is a bug? It is a defect, a flaw in code, software or documentation that can cause the said artefact to fail to perform its required function. When executed such a defect could cause the software to fail with unexpected and potentially dangerous results. Also, of course, these defects can nearly always be traced back to human error.

This could be an error in design or code or testing or even in hardware specification, for example, if the server does not have enough resources to handle the work put through it then it was underspecified, a human error. No matter what the root cause was and what type of human error caused it the outcome will always be called computer error by the press and the company who released the software.

So, what is the role of testing in software development, maintenance and operations? The role of the QA is to improve the quality of the software under test by finding defects and logic issues that can then be corrected. These corrections are then tested to confirm the risk of operational problems has been reduced and the quality of the software has therefore increased. The location and correction of bugs will increase confidence in the software and helps the software company meet contractual, legal requirements and any industry-specific standards.

Testing is part of quality assurance; it can help measure the quality of the software and determine if it is fit for purpose. Metrics are available for reporting if all bugs are recorded by their type, severity and priority. These metrics can help identify useful trends and highlight any potential weak points within the team. The life cycle of each bug is also captured from creation through to resolution, this builds confidence in the process and lessons learnt can help improve the quality of the processes used in design, development and testing.

The quality of testing is also important, a common mistake is to assume that

if no bugs are found then no bugs exist within the software under test. It is also very possible that the bugs exist, and they have simply not been found yet because the scope of testing is too narrow and not all available logic paths have been checked.

One thing is certain, though; if you do not find them during testing the end-users or customers will find them when they are using the system on live data and the implications can be very serious indeed. Therefore, experienced and competent QA's are a vital part of the team.

They know what questions to ask, where to look, what boundaries to probe and how to test. Because of their knowledge, they are also able to teach the less experienced and new QA's and they should always be present at important development meetings. Development meetings are a vital part of the development lifecycle. All interested parties should be present including the complete development team, which includes all the QA's.

An important topic during these meetings is how much testing is required to mark the product as done, fit for purpose and safe. There is no simple answer to this question, but a vital consideration is what risks are involved. As already seen some systems have risks involving human life, others have financial and security risks, others have much lower risks, however, a risk is a risk and they should always be considered.

So, how much testing should be considered enough?

- It depends on the risks that have been identified for the system
 - This includes technical and business risks
 - Human and other animal health risks
 - Financial risks
 - Customer confidence risks.
- It also depends on project constraints
 - Time
 - Budget

- Manpower

Therefore, testing is necessary because the software under development is likely to have defects during the early stages of the development cycle. Testing the software not only helps locate these bugs but it also helps build a good level of confidence in the reliability of the system. Reducing the number of bugs also reduces the risk and helps to avoid the development company from facing litigation, monetary loss and ultimately going out of business.

However, resources are always finite, and time is always at a premium, therefore risk must always be identified and prioritised as early as possible. The only way to do this is through teamwork, regular meetings and feedback from all parties including the stakeholders. These meetings are discussed in more detail later in the book. This enables the team to determine what to test first what to test most how thoroughly to test each item i.e. where to place emphasis and what not to test (this time).

Eight important principles of software testing to be considered

- Software testing will show the presence of defects.
- Exhaustive testing is impossible, there is not enough time left in the universe.
- Testing reduces the probability of undiscovered defects remaining in the software but finding no defects is not proof of overall correctness.
- Testing everything (all combinations of inputs and preconditions and all logic paths) is simply not feasible. Instead, risk analysis and priorities should be used to focus testing efforts in specific areas,
- To find defects early, testing activities should be started as early as possible in the software or system development life cycle and should be focused on defined objectives.
- If the same tests are repeated, they will no longer find any new defects. To overcome this, the existing test cases will need to be reviewed and revised, to exercise various parts of the software.
- Testing effort should be focused proportionally to the expected and later observed

defect density of modules. Finding and fixing defects does not help if the system built is unusable and does not fulfil the user's needs and expectations.

- Testing is done in different contexts. For example, safety-critical software is tested differently from an e-commerce site and a medical system will be tested differently to the latest game release. A good understanding of the system under test is vital.

The most visible part of testing is the actual test execution. This is what most people will associate with testing. But to be truly effective and efficient, test plans should also include time to be spent on planning the tests, designing the test cases to be used, preparing the system for the test execution and finally evaluating the results. Therefore, for each development life cycle, a full test plan is required. A test plan is a guide to how the test strategy and project test plan apply to the software under test. It is important to document any exceptions to the test strategy, e.g. only one test case design technique needed for this functional area because it is less critical than the overall project. Both dynamic testing and static testing can be used as a means of achieving similar objectives and will provide information that can be used to improve both the system being tested and the development and testing processes, the actual methods available will be discussed further on in this book.

An important consideration of testing is that different viewpoints of testing will take different objectives into account. By example, in development testing (e.g., component, integration and system testing) the main objective may be to cause as many failures as possible so that defects in the software code are identified and can be fixed before the final release.

By contrast in user acceptance testing (UAT), the main objective may be to confirm that the system works as expected, has a good, well designed user interface and will allow interested parties to gain confidence that the system has met all the requirements and is in a state of done. In some cases, the main objective of testing may be to assess the quality of the software (with no intention of fixing defects), to give information to stakeholders of the risk of releasing the system at a given time.

Maintenance testing often includes testing that is designed to ensure no new defects have been introduced during the latest development cycle, merging of changes and bug fixes. During operational testing, the main objective may be to assess system characteristics such as reliability or availability.

The test team also needs to be aware that debugging and testing are different entities. Dynamic testing can show failures that are caused by defects if implemented correctly.

In contrast debugging is the development activity that a programmer will employ to locate, analyse and remove the cause of the failure. Subsequent re-testing by a QA ensures that the fix does indeed resolve the failure. The responsibility for these activities is usually QA's test and programmer's debug. You should always remember that both are developers in the agile world.

Test control is the ongoing activity of comparing actual real time progress against the original test plan and reporting the status, including deviations from the plan at regular meetings. Test control involves taking actions necessary to meet the targets and objectives of the project. To control testing, the testing activities should be monitored and discussed throughout the project. Test planning considers feedback from monitoring and control activities.

At the end of every cycle or sprint test closure activities should be employed to collect data from the completed test activities to consolidate experience, facts, mistakes and numbers. Test closure activities occur at project milestones such as when a software system is released, a test project is completed (or cancelled), a milestone has been achieved, or a maintenance release has been completed or in the agile world at the end of every sprint. Typical closure activities include the following tasks:

- Checking which planned deliverables has been completed and delivered.
- Closing incident reports or raising change records for any that remain open.
- Documenting the acceptance of the system.
- Finalising and archiving test software, the test environment and the test infrastructure for reuse in later sprints.
- Analysing lessons learned to determine changes needed for

future releases and projects.

- Using the information gathered to improve test maturity of the team.

So that is a brief introduction to the foundations of software testing. You are still here and still reading so hopefully I have not frightened you off yet. The question is, though, are you good enough to become a talented, high quality assurance software QA? Do you still think this is the correct career path for you? Hopefully, the next chapter will answer these questions and more for you, please read on and enjoy as we investigate the QA's mindset.

The Testers Mindset

Most people can test software at a very basic level. After all, anyone should be able to click buttons, look at web pages, input text, check spelling and press [Save]. However, being able to test software proficiently over many years at an expert level requires a very special type of mind-set. Let's be honest here not everyone has this ability, and there is no shame in this, not everyone can race cars, run a marathon or fly an aeroplane either. A software QA does and must have a very different mindset to that of a software programmer and while both do not always mix well their interaction and cooperation within a development team is crucial in the agile world. Some programmers can test their own code and indeed all programmers should test at a unit level before sending updates out into the test cycle. However, separation of this responsibility to a professional, well trained QA is typically done to help focus effort and provide additional benefits, such as an independent perspective by trained and professional testing resources. Independent testing can and should be carried out at every level of testing.

A certain degree of independence (also known as avoiding the author bias or programmer arrogance) often makes the QA more effective at finding defects and failures in candidate release code. Independence is not, however, a replacement for familiarity and the best QA's are often those who are familiar with the product under test rather than QA's who have simply been contracted in for a single project.

Several levels of independence can be defined as shown here from low to high:

- Tests designed by the programmers who wrote the actual software under test (low level of independence).
- Tests designed by other programmers within the same programming team.
- Tests designed by a person(s) from a different organisational group (e.g., the test team) or test specialists.
- Tests designed by a person(s) from a different organisation

contracted in just to test the product.

Let's face it folks, testing can sometimes be a tedious and repetitive undertaking and not everyone can handle this task for a very long period of time. So when you see a quality assurance QA banging their head on their desk or throwing darts at photographs of the programmers it may well be time to move them elsewhere within your organisation. A good QA will be methodical in their approach and will have good written and verbal communication skills. To them, each day is a fresh, interesting new challenge. They will have a lot of patience and should be able to maintain a high level of concentration throughout the day. Also, when writing up their conclusions of a recent test they need to be able to explain their findings in a complete, literate but sensitive manner while providing adequate evidence to support these conclusions. Any hint of trying to indicate a programmer has done a poor job should always be avoided even if it is true.

Such negative and disparaging comments only lead to divisions within a company and these are ultimately destructive. Also making a programmer feel as though you enjoy finding fault with their work is a fast way of isolating the programming and testing teams.

If the worst does happen and such divisions do become established, they can be impossible or at the least difficult to remove until certain people are removed from the firing line or company structure itself.

Most programmers are a highly skilled but sometimes temperamental group of people, many of whom consider themselves the intellectual tip of the company iceberg that they work for and as such an untouchable elite that should not be disrespected or upset in any way. However, the world is changing and as I have already stated the world is awash with highly skilled and talented programmers.

The rise of the software QA is now unstoppable and their importance within the software development world is now indisputable. So how do you incorporate these two groups of very different and potentially at war teams with one software development company?

Probably the best solution is to reinforce the team ethos and drive home the fact that everyone is working on the same end game. Software development

is a team game. The development team is made up of programmers, quality assurance testers (QA's), product owners and stakeholders. QA's are as important as any other member of the team but not more important than anyone else. Agile is a great tool for bringing these ideas forward and that is why this book was written.

Regular meetings are an important part of building the team spirit it is also essential to ensure a well-designed, well programmed and well tested software package. Stakeholder involvement is also essential at every stage of the development cycle and I have always involved the end-user throughout the complete life cycle. Thankfully, the agile model enforces this idea home fully. It is important to remember that people and projects are usually driven by known objectives and deliverables. People tend to align their plans with the objectives set by management and other stakeholders, for example, to find defects or to confirm that software meets its objectives. Therefore, it is important to clearly state the objectives of testing at an early stage.

Identifying failures during testing may be and often is perceived as criticism against the product and against the programmer who wrote it. As a result, testing is often seen as a destructive activity in some eyes, even though it is very constructive in the management of product risks. Looking for failures in a system requires curiosity, professional pessimism, a critical eye, attention to detail, effective communication with development peers, and experience on which to base error guessing. These are a very special set of skills that make experienced software QA's a very valuable commodity.

As I have already indicated if errors, defects or failures are communicated constructively, bad feelings between the QA's and the analysts, designers and programmers can be avoided. This applies to defects found during reviews as well as in testing. The hardest group to keep happy in these circumstances is the programmers as previously mentioned they tend to be a very sensitive and temperamental breed.

The QA's and test leader will need very good interpersonal skills to communicate information about defects, progress and risks in a constructive and non-destructive way. For the programmer of the software, useful information can help them improve their skills. Defects found and fixed during testing will save time and money later and reduce risks.

Unfortunately, communication problems can occur, particularly if QA's are seen only as harbingers of doom and unwanted news about defects. However, there are several ways to improve communication and relationships between QA's and others:

- Collaborate: Start with team collaboration rather than battles – remind everyone in the team that the common goal is better quality software systems and that they are all of identical importance to the team.
- Communication: Communicate findings on the product in a neutral, fact-focused way without criticising the programmer(s) who created it, for example, write objective and factual incident reports and review findings.
- Understanding: Try to understand how the other person feels and why they react as they do. Get to know the people on your team.
- Confirmation: Confirm that the other person has understood what you have said and vice versa.

So that's what it takes to be a good QA.

So, do you still think you can be a QA?

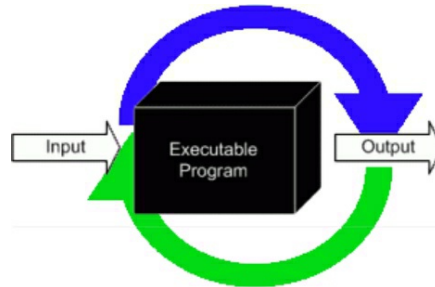
If yes then well good for you, you may well have a long, fulfilling future in the world of software development. Now let's look at the common types of testing next.

Types of Testing

So, before we move on to the world of agile software development and agile software testing let us discuss the major types of testing currently in use today. This is not a complete, exhaustive list and I am sure some of you will all know of a few other types and want to say 'what about ...' but please remember this book has already been written and published. Also, I cannot hear you but please feel free to email any suggestions for the next edition. However, most of the common styles are here, you will not use all of them, but you will come across most during your careers in IT.

These days software QA's should have a good array of testing methods and tools at their disposal. These can be generally classified within the Dynamic and Static areas. Static testing is based on the methods used for Code Reviews, Walkthroughs and Inspections. Dynamic testing methods involve a developer or QA using the computer program or parts of it. Types can also be split between Black Box and White Box as detailed below.

Black box testing



Black box testing is a method of software testing where the QA team are not required to know or understand the code and internal structure of the software that is currently under test. For example, in a black box test on software design, the QA only knows the required inputs and what the expected outcomes should be and not how the program code arrives at those outputs. The QA team will never examine the actual programming code and they do not need any further in-depth knowledge of the program other than its specifications.

The advantages of this type of testing include:

- The test is unbiased because the designer and the QA are independent of each other.
- The QA does not need knowledge of any specific programming languages.
- The test is done from the point of view of the user, not the designer.
- Test cases can be designed as soon as the specifications are complete.

Discussed next are the most common types of Black Box Testing.



Acceptance testing (also known as **User Acceptance Testing - UAT**) is a formal type of software testing that is performed by the end-user when the features have been delivered by developers, it is usually the last formal test level in a release cycle. This type of testing aims to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery. Finding defects is not the focus of acceptance testing. In some organisations, this is also known as Beta Testing. Acceptance tests are normally documented at the beginning of the sprint (in agile) and are a means for business QA and developers to work towards a common understanding and shared business domain knowledge.

Acceptance testing may occur at various times in the lifecycle, for example:

- A COTS (Commercial off the shelf) software product may be acceptance tested when it is installed or integrated.
- Acceptance testing of the usability of a component may be done during component testing.
- Acceptance testing of a new functional enhancement may come before system testing.



Accessibility Testing is designed to ensure the contents of the website can be easily accessed by disabled people. Accessibility testing is very similar to usability testing, in that it is about making sure that the website or application under test is easy for its intended audience to use. That audience includes users who access the service via a range of assistive

technologies like:

- screen readers
- voice recognition software
- trackball devices

It's important to consider a range of disabilities when you are testing any website or application service, including those with:

- cognitive and learning disabilities, e.g. dyslexia or attention deficit disorders
- visual impairments, e.g. total and partial blindness, colour blindness, poor vision
- auditory disabilities, which can also affect language
- motor skills impairments, e.g. those affected by arthritis, strokes, RSI



Ad-hoc testing, this form of software testing is usually very informal and unstructured and can be performed by any stakeholder without any reference to any test case or test design documents. The person performing Ad-hoc testing should, however, have a good understanding of the domain and workflows of the application to find defects and attempt to break the software. As a result, Ad-hoc testing is intended to find defects that were not found by existing test cases.

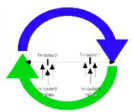


All Pairs Testing is also known as Pairwise testing. This is a black box testing approach that can be done by software QA's, developers, business analysts or any other interested stakeholder. It is a combinatorial

testing method that, for *each pair* of input parameters to a system (typically, a [software algorithm](#)), tests all possible discrete combinations of those parameters. By the careful selection of [test vectors](#), this can be done much faster than an exhaustive search of [all combinations](#) of all parameters, by "parallelising" the tests of parameter pairs.



Automated testing is a testing approach that makes use of testing tools and/or programming to run the test cases using software or custom-developed test utilities. Most of the automated tools provided capture and playback facility, however, some tools require writing extensive scripting or programming to automate test cases. One of the best tools currently available is Selenium which can be configured to work with Java and dotNet. Selenium is a fascinating and very useful tool that will be discussed later in this book.



Boundary Value Testing (BVT) is a testing technique that is based on the proven concept that “error aggregates at boundaries”. In this testing technique, testing is done extensively to check for defects at known boundary conditions. A classic example of this is if a field accepts values from 1 to 99 then testing is done for values 0, 1, 2, 98, 99 and 100 because these are the boundaries. In this example 1, 2, 98, 99 are valid while 0 and 100 would be invalid. Tests can be designed to cover both valid and invalid boundary values. When designing test cases, a test for each boundary value is chosen.

Boundary value analysis can be applied at all test levels. A good feature of this test is that it is relatively easy to apply, and its defect-finding capability is high. Detailed specifications help determine the interesting boundaries.



Bottom-up Integration testing is an integration testing approach where the testing cycle will start with smaller pieces or subsystems of the whole software and gradually build its way up until testing is covering the entire software system. The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. These units are exercised through their interfaces using black-box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested, and individual subsystems are exercised through their input interface. Test cases are constructed to test whether all the components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e. unit testing. The overall idea is a building block approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages.



Browser compatibility Testing is one of the subtypes of testing of compatibility testing performed by the test team. Browser compatibility testing is performed for web applications with a combination of different browsers and operating systems. This is also referred to as User Experience Testing and is designed to ensure the following.

- Users have the same visual experience irrespective of the browsers through which they view the web application.
- In terms of functionality, the application must behave and respond the same way across different browsers and different operating systems.



Decision Table Testing methods are a very effective way to capture system requirements that contain logical conditions (e.g., True or False), and to document internal system design. They can also be used to record complex business rules that a system has to implement. When creating decision tables, the specification is analysed, and conditions and actions that the system must meet are identified. The input conditions and actions are most often stated in such a way that they must be true or false. The decision table contains the triggering conditions, often combinations of true and false for all input conditions, and the resulting actions for each combination of conditions. Each column of the table corresponds to a business rule that defines a unique combination of conditions and which result in the execution of the actions associated with that rule. The coverage standard commonly used in decision table testing is to have at least one test per column in the table, which typically involves covering all combinations of triggering conditions.



Equivalence Partitioning is also known as Equivalence Class Partitioning. This is an extremely specialised software testing technique and not a complete type of testing by itself. Equivalence partitioning

technique is used in black box testing types. Equivalence partitioning classifies test data into Equivalence classes as positive Equivalence classes and negative Equivalence classes, such classification ensures both positive and negative conditions are tested. Partitions can also be identified for outputs, internal values, time-related values (e.g., before or after an event) and for interface parameters (e.g., integrated components being tested during integration testing). Tests can be designed to cover all valid and invalid partitions. Equivalence partitioning is applicable at all levels of testing. Equivalence partitioning can also be used to achieve input and output coverage goals. It can be applied to the human input, input via interfaces to a system, or interface parameters in integration testing.



Exploratory Testing is an informal type of testing which is conducted by QA's to learn and understand the software while at the same time looking for errors or any application behaviour that seems non-obvious or incorrect. Exploratory testing is usually done by QA's, but it can also be done by other stakeholders. Team members such as business analysts, developers, end-users etc can also undertake this task. Anyone who is interested in learning the functions of the software and at the same time looking for errors or behaviour that seems non-obvious is a potential QA for this method.



Functional testing will nearly always be required during a software test phase. There are essentially two types of functional testing;

these are Full program testing and Change testing (see Regression Testing).

Functional tests are based on the functions and features (described in documents or understood by the QA's) within the system and their interoperability with specific systems and may be performed at all test levels (e.g., tests for components may be based on a component specification).

Full program testing is most commonly used when a module is being tested before its first release into production. The test document created will be a complete step by step test of the module. Every process should be carefully tested, and the results analysed and fully documented.

When testing a web based application all functional testing should be carried out in all the most common browsers, such as Firefox, Chrome, Safari and Microsoft Edge. The browsers and versions used should also be documented in the test document. This will ensure that anyone reading the document in the future will be fully aware of which browsers and versions were used during the test. The QA should fully detail the functional test by outlining each process with as much detail as is required to confirm the thoroughness of the testing. Screenshots should also be included. These can provide a very informative graphic view which backs up the textual description of the test. The order in which the test is completed is not important if every aspect is covered.



Fuzz Testing or fuzzing is a software testing technique that involves testing with unexpected or random inputs. The software is monitored for failures or error messages that are presented due to the input errors.



Integration Testing (See also System Integration Testing) is also known as [SIT] in short, this is one of the more important

types of software testing. Once the individual units or components have been unit tested by developers and confirmed as working then the testing team will run tests that will test the connectivity among these units/component or multiple units/components. There are different approaches to Integration testing as shown below:

- **Big Bang Integration testing** is one of the integration testing approaches, in Big Bang integration testing all or all most all the modules are developed and then coupled together.
- **Incremental Integration Testing** is a bottom-up approach for continuous testing of an application as new functionality is added; Application functionality and modules should be independent enough to test separately. This type of testing is done by programmers and/or by QA's.
- **Component integration** testing tests the interactions between software components and is done after component testing
- **System integration** testing tests the interactions between different systems or between hardware and software and may be done after system testing.

An important consideration here is that the greater the scope of integration, the more difficult it becomes to isolate defects to a specific component or system, which may lead to increased risk and additional time for troubleshooting.

Systematic integration strategies may be based on the system architecture (such as top-down and bottom-up), functional tasks, transaction processing sequences, or some other aspect of the system or components. To facilitate fault isolation and detect the defects early, integration should ideally be incremental rather than “big bang”. Testing of specific non-functional characteristics (e.g., overall performance) may be included in integration testing as well as functional testing.

At each stage of the integration testing phase, QA's will concentrate fully on the integration itself. For example, if the team are integrating module X with module Y then they will be interested in testing the communications between the two modules, not the functionality of the individual module as that was

done during component testing. Both functional and structural approaches may be used.

To help test efficiently the QA's would ideally understand the architecture of the design and they should be allowed to influence the integration planning. If integration tests are planned before components or systems are built, those components can be built in the order required for most efficient testing.



Logical Access Testing is required when testing web-based applications and in these circumstances, logical access is a vitally important part of the test regime. Testing should be broken into two distinct sections; the first section is Zero Access.

Zero access is the operation of testing that a URL can only be reached after a user has successfully logged into the application using a valid User ID and a valid password, otherwise, they should be returned to the login page or a pre-defined warning page. An effective method for preparing for this test is to record all URL's that the QA encounters during the Functional test stage. It is vital that every possible URL is gathered during the functional test and then tested during this stage and this method helps reduce the number of missed URL's.

The method most often used is to log out of the application and then to paste each URL into the address bar and check the response. Your desired response will probably be for the user to be sent to a log in screen or a pre-defined error page. What you do not want is for the user to be allowed system access after they have logged out of the system.

The second section of logical access testing is Profile Access. This section is more complex than Zero Access and requires more thought while testing. In this section, you can assume the user has logged incorrectly and you should use the same URL's noted in functional testing as you used in the Zero Access section, however, this time do not paste them into the address box but navigate to them. In this section the questions to be answered are:

Does the system allow the user access to a module that their profile states they should have access to? This should be tested by logging into the system and navigating to the required module or section.

Next, this test should be undertaken on a user profile that does not have access to the module. The test should be undertaken in the same manner as 1 and both tests with results should be fully documented in the test document.

How far you take profile testing will, of course, depend on the complexity of your system. If multiple modules are dependant on profile access, then each module will need testing. If different users have different access rights to certain actions such as create records, deleting records and running reports then these will all have to be tested. Also, if user profile additionally defines a user's geographical access then this must also be tested, as demonstrated next.

Profile testing access for a Country/Region. Take for example the URL below.

<http://www.yourapplication.com/customeraccounts/welcome.do?country=de>

This address is a customer accounts module on a test server and the Country is Germany (de). Testing should be undertaken on this address with user profiles that allow and do not allow access to the given Country. To confirm this procedure another Country should also be tested in the same way and all the results should be fully documented in the test document.

These steps are very important. They endorse to the project leader and the customer that profile access is secure at a Module and a Geographical level. This type of testing is ideally suited to automation and we will discuss this later in the Selenium section.



Locale Testing Another important type of test for web-based applications is locale testing. This is another type of test that is becoming more common as web-based applications grow in number and the potential

user base becomes global. Locale testing will check the quality of your applications localisation for a target culture/locale. While it is impossible to test every potential global locale, you should certainly test a good sample to confirm the process appears to be reliable.



Maintenance Testing (See also regression testing) after the initial deployment, a good software system is often in service for years or even decades. During this time the system, its configuration data, or its environment are often corrected, changed or extended. The careful planning of releases in advance is crucial for successful maintenance testing. A distinction must be made between planned releases and bug fixes. Maintenance testing (retesting) is done on an existing live working system and is triggered by modifications, migration, or retirement of the software or system.

Modifications include planned enhancement changes (e.g., release-based), corrective and emergency changes, and changes of environment, such as planned operating system, web browser or database upgrades, planned upgrade of linked Commercial-Off-The-Shelf software, or patches to correct newly exposed or discovered vulnerabilities of the operating system or web browsers.

Maintenance testing for migration (e.g., from one platform to another or from one server to another) should include operational tests of the new environment as well as of the changed software. Migration testing is also needed when data from another application will be migrated to the system being maintained.

In addition to testing what has been changed, maintenance testing includes regression testing to parts of the system that have not been changed. The scope of maintenance testing is related to the identified risk of the change, the

size of the existing system affected and to the size of the change.

Depending on the changes to be implemented, maintenance testing may be done at any or all test levels and for any or all test types.

Determining how the existing system may be affected by changes is called impact analysis and is used to help decide how much regression testing to do. This is a critical exercise that should always be undertaken. The impact analysis may be used to determine the regression test suite.



Negative Testing is a type of software testing which calls out the “attitude to break”; these are functional and non-functional tests that are designed to break the software by entering incorrect data like incorrect date, time or string or attempt to upload a binary file when a text file is supposed to be uploaded. Another method is to enter huge text strings for input fields etc. The core difference between positive testing and negative testing is that throwing an exception is not an unexpected event in the latter. When you perform negative testing, exceptions are expected – they indicate that the application handles improper user behaviour correctly. It is generally considered a good practice to combine both the positive and the negative testing approaches



Non-Functional testing refers to aspects of the software that may not be related to a specific function or user action, such as usability or performance. Non-functional testing tends to answer such questions as ‘how well does the system perform when I save a new record?’

Non-functional testing can include (but is not limited to): Usability,

Robustness, Compatibility, Performance, Load, Stress, Endurance, Stability, Accessibility, Extensibility, Scalability, functionality and Portability. It is the testing of 'how well' the system works under normal usage.

Non-functional testing aims to verify that the software functions correctly even if the input is invalid or unexpected. The software should handle such issues in a controlled manner and respond with meaningful error messages. The program should not crash. Non-functional testing is also concerned with how well the software works when under load. Does the screen hang for a long while when connecting to a database? Are there delays when switching screens? Such tests are also known as Stress Testing and Load Testing. Whatever the title is, if the software slows down under load then this affects users working speed and they will soon become unhappy with the product. Companies will also be concerned that productivity will be compromised, and they may well eventually reject the package as not fit for purpose.

Non-functional testing and functional testing are usually completed at the same time. Some development teams will document them separately while others will merge both types into the same part of the document. Neither method is wrong if the testing is complete and correctly recorded.



Penetration Testing is a type of security testing. This is also known as PenTest in its shortened name. Penetration testing is undertaken to test how secure software and its environments (Hardware, Operating system and network) really are when subject to attack by an external or internal intruder. The intruder can be a human/hacker or a malicious program. Pentest uses methods to forcibly intrude (by brute force attack) or by using a weakness (vulnerability) to gain access to a piece of software or its data or hardware with the intent to expose ways to steal, manipulate or corrupt the data, software files or configuration. Penetration Testing is a way of ethical hacking, an experienced Penetration tester will use

the same methods and tools that a hacker would use but a Penetration tester intends to identify vulnerability and get them fixed before a real hacker or malicious program exploits it.



Regression Testing (See also maintenance testing) is any type of software testing that seeks to uncover new errors, or *regressions*, in existing functionality after changes have been made to the software, such as functional enhancements, bug fixes or configuration changes. This is sometimes confused with maintenance testing and you may wonder how they differ? In regression testing you retest the test case; therefore, you could consider regression testing to be a subset of maintenance testing/retesting. However, in the real world, this is mostly a semantics issue. More commonly different teams will use "maintenance testing " and "regression testing" interchangeably.

Changes made to existing software packages is a common way of introducing software bugs. These bugs will quite often appear in areas of the program that have not been altered by recent changes but often due to shared classes, once stable sections of code are suddenly unstable. Therefore, regression testing should always be performed alongside Confirmation Testing. This form of testing can be tedious if a QA is expected to complete this function after every upgrade. It is, however, essential if the software is to be released stably.

Regression testing intends to assure that a change, such as a bug fix, did not introduce new bugs and that the base functionality of the application has not been broken. This form of testing is not required if the module under test is a new application that has not been previously released, however, existing modules will need regression testing.

The most common method of regression testing is re-running previously run tests. The QA should locate the most recent complete test document for the

module as well as the last three change test documents (less if only one or two previous documents exist). This is one reason why every test should be recorded and stored in a secure location.

Regression testing involves retesting the unchanged parts of the module and to achieve this goal the QA should step carefully through the previous test documents checking that the results are still the same. The steps need to be carefully mirrored and should be completed in Internet Explorer (IE), Firefox and other popular browsers. Which browsers were used should also be documented?

Any crashes, unexpected results or strange responses should be reported to the programmer via your company's usual bug reporting facility, my personal preference is Bugzilla, but this is simply one of many options. When a fix is released for a reported bug in regression testing a new full regression test should then be started to ensure that the new fix has not introduced yet more new errors. This process should continue until a complete successful regression test is recorded.



Risk based testing is a type of software testing and a different approach to testing developing software. In Risk based testing the requirements and functionality of the software to be tested are prioritised as Critical, High, Medium and low. In this approach, all critical and high priority tests are tested first and then this is followed by the medium tests. The low risk functionality is then tested at the end. However, these may not be tested at all if there is no further time available for testing.



Sanity Testing is a type of testing that is carried out mostly by QA's and in some projects by developers as well. Sanity testing is a quick evaluation of the software, environment, network, external systems are up &

running, software environment is stable enough to proceed with extensive regression testing. Sanity tests are narrow and most of the time sanity tests are not documented however they do help to avoid wasting time and cost that is incurred in testing if the build has failed.



Smoke testing is a type of testing that is carried out by software QA's to check if the new build provided by the development team is stable enough i.e., major functionality is working as expected to carry out further or detailed testing. Smoke testing is intended to find “showstopper” defects that can prevent QA's from testing the application in detail. Smoke testing carried out for a build is also known as ‘build verification test’. This type of testing is very similar to Sanity Testing.



Soak Testing is a special type of performance testing, where the software under test is subjected to load over a significant duration of time, soak testing may go on for a few days or even for a few weeks. Soak testing is a type of testing that is conducted to find errors that result in degeneration of software performance with continued usage. Soak testing is extensively done for electronic devices, which are expected to run continuously for days or months or years without restarting or rebooting. With growing web applications soak testing has gained significant importance as web application availability is critical for gaining and then sustaining customer confidence which will help ensure the success of the business in question.



System Integration Testing (See also **Integration**

Testing) is also known as “SIT” and is a type of testing conducted by software testing teams. As the name suggests, the main focus of System integration testing is to test for errors related to integration among different applications, services, third-party vendor applications etc. As part of SIT, end-to-end scenarios are tested that would require software to interact (send or receive data) with other upstream or downstream applications, services, third-party application calls etc.



System Testing is a test exercise that is concerned with the behaviour of the complete and whole system or product under development. The testing scope shall be clearly addressed and defined in the Master and/or Level Test Plan for that test level. In system testing, the test environment should correspond as closely as possible to the final target or production environment platform to minimise the risk of environment-specific failures not being found in the testing phase.

System testing can include tests based on risks and/or on requirements specifications, business processes, user stories and other high-level text descriptions or models of system behaviour, interactions with the operating system, and system resources.

System testing is comprehensive and should investigate functional and non-functional requirements of the system, and data quality characteristics. The QA's will also need to deal with incomplete or undocumented requirements as they find them. System testing of functional requirements starts by using the most appropriate specification-based (black-box) techniques for the aspect of the system to be tested. Structure-based techniques (white-box) can

then be used to assess the thoroughness of the testing with respect to a structural element, such as menu structure or web page navigation.



Volume testing is a non-functional type of testing carried out by performance engineering teams. Volume testing is one of the types of performance testing. Volume testing is carried out to find the response of the software with different sizes of the data being received or to be processed by the software. E.g. If you were to be testing Microsoft Word, volume testing would be to see if Microsoft Word can open, save and work on files of different sizes (10 to 100 MB).



Vulnerability Testing involves identifying and exposing the software, hardware or network vulnerabilities that can be exploited by hackers and other malicious programs such as viruses or worms. Vulnerability Testing is a key test for software security and availability. With a huge increase in the number of hackers and malicious programs worldwide, Vulnerability Testing is now critical to the success of a Business. This test is often used in conjunction with Penetration testing.

White Box Testing

White box testing can also be known as clear box testing, transparent box testing and glass box testing. White box testing is a software testing approach, which intends to test software with knowledge of the internal code and intended working of the software.

Typically, the white box testing approach is used in Unit testing which is usually performed by software developers. White box testing intends to execute code and test statements, branches, path, decisions and data flow within the program being tested. White box testing and Black box testing complement each other as each of the testing approaches has the potential to uncover a specific category of errors. Common types of white box testing are discussed next.



API Testing is a type of testing that is like unit testing. Each of the Software APIs is tested as per the API specification. API testing is mostly done by the test team unless APIs to be tested is very complex and needs extensive coding. API testing requires understanding both the API functionality and possessing good coding skills.

So, you may ask, what is an API? Well API is short for application programming interface, which is a bit of a mouthful so let us stick with API. So, an API is a set of programming instructions and defined standards for accessing a Web-based software application or a Web tool. Typically, a software company will release its API to the public so that software developers can design new products that are powered by or make use of its service.

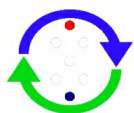
A very good example of this is when Amazon.com released its API so that Web site developers could more easily access Amazon's product information databases. Using the Amazon API, a third-party Web site can easily post direct links to Amazon products with updated prices and specific options such as "buy now".



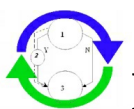
Branch Testing is a white box testing method for designing test cases to test code for every branching condition. The branch testing method is typically applied to a unit testing phase.



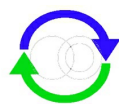
Component Testing is a type of software test that is performed by developers. Component testing is carried out after completing unit testing. Component testing involves testing a group of units as code together as a whole rather than testing individual functions, methods.



Condition Coverage Testing is a testing technique used during unit testing, where a developer tests for all the condition statements like if, if-else, case etc, in the code being unit tested.



Decision Coverage Testing is a testing technique that is used in Unit testing. The objective of decision coverage testing is to expertise and validate each decision made in the code e.g. if, if-else, case statements. The decision testing technique derives test cases to execute specific decision outcomes. Branches originate from decision points in the code and show the transfer of control to different locations in the code. Decision testing is a form of control flow testing as it follows a specific flow of control through the decision points. Decision coverage is stronger than statement coverage; 100% decision coverage guarantees 100% statement coverage, but not vice versa.



Structural Testing is the testing of the structure of the system or component. In structural testing, the QA's are required to know the internal implementations of the code. Here the QA's require knowledge of how the software is implemented, how it works. During structural testing, the tester is concentrating on how the software does it. For example, a structural technique wants to know how loops in the software are working. Different test cases may be derived to exercise the loop once, twice, and many times. This may be done regardless of the functionality of the software.

Structural testing can be used at all levels of testing. Programmers use

structural testing in component testing and component integration testing, especially where there is good tool support for code coverage. Structural testing is also used in system and acceptance testing, but the structures are different. For example, the coverage of menu options or major business transactions could be the structural element in the system or acceptance testing.



Unit testing (also known as component testing) is a type of white box testing that is performed by software developers whenever they update their code. Using white box testing techniques, testers (usually the developers creating the code implementation) verify that the code does what it is intended to do at a very low structural level. Unit testing usually involves developing stubs and drivers. Unit tests are often ideal candidates for automation. Automated tests can run as Unit regression tests on new builds or new versions of the software.

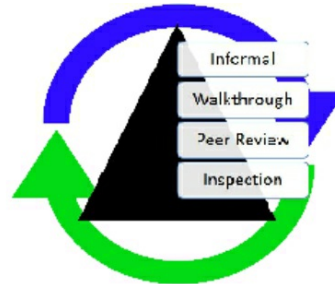
There are many useful unit testing frameworks like Junit, NUnit etc, available that can make unit testing more effective. When available, the tester will examine the low-level design of the code; otherwise, the tester will examine the structure of the code by looking at the code itself. Unit testing is generally done within a class or a component. Programmers will nearly always press the correct buttons when testing and they always test the sequence in the correct order. Rarely do they carry out destructive testing or integration testing. Unfortunately, it is difficult for a tester to carry out unit testing unless they are proficient in programming and have access to the developer's code, currently, this is rare.

Component testing may include testing of functionality and specific non-functional characteristics, such as resource-behaviour (e.g., searching for memory leaks) or robustness testing, as well as structural testing (e.g., decision coverage). Test cases should be derived from work products such as a specification of the component, the software design or the data model.

One possible approach to component testing is to prepare and automate test cases before the actual coding process. This is called a test-first approach or test-driven development (TDD, See Chapter 14). This approach is highly iterative and is based on cycles of developing test cases, before building and integrating small pieces of code, and executing the component tests correcting any issues and iterating until they pass.

This may seem a strange way of doing things at first, but with practice it's much more efficient than writing a bucket load of code, running it, and going back later to figure out everywhere it's broken (a process that is known as debugging). This process puts the programmer in a testing mindset while writing code, which leads to higher-quality code which in turn makes life easier for the tester.

Static Testing



This is a form of testing that is commonly used in reviews and walkthroughs which are employed to evaluate the correctness of the deliverable. Unlike dynamic testing, which requires the execution of software; static testing techniques rely on the manual examination (reviews) and automated analysis (static analysis) of the code or other project documentation without the execution of the code. The code is reviewed for syntax, commenting, naming convention, the size of the functions and methods etc. Static testing usually has checklists against which deliverables are evaluated.

Manual reviews are a good way of testing software work products (including code) and can be performed well ahead of dynamic test execution. Defects that are detected during reviews early in the life cycle (e.g., defects found in requirements) are often much cheaper to remove than those detected by running tests on the executing code.

In most cases, a review could be done entirely as a manual activity, but there is also tool support available if required or desired. The main manual activity is to examine a work product and make constructive comments about it. Any type of software work product can be reviewed, including requirements specifications, design specifications, raw code, test plans, test specifications, test cases, test scripts, user guides or web pages.

The benefits of reviews include early (and cheaper) defect detection and correction, improved development productivity improvement, reduced development timescales, reduced testing cost and time, fewer defects in code under test and improved communication between all affected parties. Reviews can also find critical omissions in requirements which are less likely to be found in dynamic testing.

All reviews, static analysis and dynamic testing should have the same objective, which is to find defects. These methods are complementary, and the different techniques can find different types of defects effectively and efficiently. In comparison to dynamic testing, static techniques find the causes of failures and defects rather than the actual failures themselves.

The style of the different types of reviews varies from informal, characterised by no written instructions for reviewers, to systematic, characterised by team participation, documented results of the review, and fully documented procedures for conducting the review. The formality of a review process is related to factors such as the maturity of the development process, customer requirements, any legal or regulatory requirements or the need for an audit trail.

A single software product may be the subject of more than one review during its life cycle. If more than one type of review is used, the order may vary. For example, an informal review may be carried out before a technical review, or inspection may be carried out on a requirements specification before a walkthrough with stakeholders. The main characteristics, options and purposes of common review types are:

Informal Review

- No formal process used.
- These could take the form of pair programming or a technical lead reviewing designs and code.
- The results may (or may not) be documented.
- These can vary in usefulness depending on the reviewers.
- Main purpose: an inexpensive way to get some benefit.

Walkthrough

- The meeting is always led by the author.
- These may take the form of scenarios, dry runs and peer group participation.
- They are open-ended sessions.
- They can include an optional scribe (but this should not be the author).

- They may vary in practice from quite informal to very formal.
- Main purposes: learning, gaining understanding, finding defects.

Technical Review

- A formally documented, defined defect detection process that includes peers and technical experts with optional management participation.
- These may also be performed as a peer review without management participation.
- Ideally, the review will be led by a trained moderator (who should not be the author).
- There should be pre-meeting preparation by reviewers.
- There is scope for optional use of checklists.
- These will lead to the preparation of a review report which includes the list of findings, the verdict whether the software product meets its requirements and, where appropriate, recommendations related to findings.
- They may vary in practice from quite informal to very formal.
- Main purposes: discussing, making decisions, evaluating alternatives, finding defects, solving technical problems and checking conformance to specifications, plans, regulations, and standards.

Inspection

- These are led by a trained moderator (should not be the author).
- They are usually conducted as a peer examination.
- They must have defined roles.
- They will usually include metrics gathering.
- This is a formal process that is based on rules and checklists.
- There will be specified entry and exit criteria for acceptance of the software product.
- They should always include pre-meeting preparation.
- In the end, there should be an inspection report including a list of findings.

- Main purpose: finding defects.

Remember that the objective of the static analysis is to find defects in software source code and software models. Static analysis of the code is performed without executing the software being examined by the tool. Also, static analysis can locate defects that are hard to find while dynamically testing the compiled software. As with reviews, static analysis finds defects rather than failures. Static analysis tools analyse program code (e.g., control flow and data flow), as well as the generated output such as HTML and XML.

Static analysis tools are typically used by programmers before and during component and integration testing. They are also used when committing code to configuration management tools and by designers during software modelling. Static analysis tools may produce many warning messages, which need to be very well-managed to allow the most effective use of the tool.

Serverless Testing

So, what is Serverless?

This is a very good question and there are many different definitions around. This, however, is mine, Serverless computing is a method of providing backend services on an as-used basis, which means you only pay for actual usage and the general idea is your organisation will save costs. A Serverless provider will allow users to write and deploy code without the hassle of worrying about the underlying infrastructure. This means an organisation that gets backend services from a Serverless vendor is charged based on their computation usage and do not have to reserve and pay for a fixed amount of bandwidth or number of servers, as the service is auto-scaling. You should, however, note that although called Serverless, physical servers are still used but developers do not need to be aware of them, what they are or where they are.

In the dark, early days of the web, anyone who wanted to build a web application had to own the physical hardware required to run a server, which is a cumbersome, time consuming and expensive undertaking.

Then came the cloud, where fixed numbers of servers or amounts of server space could be rented remotely on a monthly or annual basis. Developers and organisations who rent these fixed units of server space generally over-purchase to ensure that a spike in traffic or activity wouldn't exceed their monthly limits and break their applications. This means that much of the server space that was purchased was often not used and went to waste. Now in these enlightened times, cloud vendors have introduced auto-scaling models to address the issue. While a step forward the service is still not perfect for even with auto-scaling an unwanted spike in activity, such as a DDoS Attack, could end up being very expensive.

Serverless computing allows developers to purchase backend services on a flexible 'pay-as-you-go' basis, meaning that developers and organisations only must pay for the services they use. You could liken this to switching from a mobile phone plan with a monthly fixed limit, to one that only charges

for each byte of data that gets used. Overall you should save money.

Something to remember is that the term ‘Serverless’ is somewhat misleading in real life. There are still servers providing these backend services, but all the server space and infrastructure concerns are handled by the vendor. Serverless means that the developers can do their work without having to worry about servers at all. It does, however, offer new challenges to the QA team.

How to test Serverless applications

Testing Stages

Unit Tests

As completed by the programmers while designing the code. Unit tests tend to be short and closer to the code. Unit tests should cover every non-trivial path (happy path and edge cases). These tend to be the cheapest type of tests.

Integration Tests

Integration tests are those that primarily de-risk connections between various components, normally something you build and something that someone else has built, they also fill in the gaps created by what unit tests cannot cover. For example, a test that checks if my code saves things correctly to an S3 file system would be an integration test.

The primary characteristic of identifying an integration test is the type of risk it covers, and this category does not imply anything about the area of coverage. They can be automated against a single unit of code, around several components, through a subsystem or even around the entire system. These tests are managed by the QA team.

In case your expectations of the response format are incorrect, or if there’s a bug in your DynamoDB query expression, integration testing will help you solve all these issues. While performing integration testing, you’ll invoke the function locally by passing in a stubbed event as well as context objects. In case the function needs to integrate with the external services, then the function itself should be set to talk to the “real thing.”

Acceptance Testing

All the tests mentioned above can help you identify potential problems in

your code. Is there anything else that might happen? Well, of course, there is. Your functions might not have right IAM permissions set up or even it's not allowed to communicate with DynamoDB table. Are you having troubles with function's timeout setting being set too short? Even if not enough memory was allocated, that's also another issue for you to resolve.

As a professional QA engineer, you should take into consideration that there are a lot of opportunities that could lead to misconfiguration. You need to try out your functions after their deployment, so you'll be sure if everything works perfectly and as expected end-to-end. In case you're using API Gateway and Lambda, make an HTTP request against the deployed API and be sure to validate against the responses so you'll accomplish an end-to-end test. That is the way you will be able to find permissions and other configuration errors that will almost certainly be missed by unit and integration tests.

GUI Tests

If a UI client is using your Serverless application directly or not, you should make sure if the changes are compatible with the client. You're able to run automated visual tests as well as automated tests against different devices and platforms that use services like AWS Device Farm. Also, these tests can be done manually by a Q&A team, or even automated tests via Selenium-like frameworks. These tests are managed by the QA team.

Automated Testing

See Acceptance and GUI Tests.

There are new challenges with Serverless testing such as database connection failures, service restarts and other elements that cannot be emulated due to the fact there is no direct access to the AWS managed service. These will need to be handled as encountered and your automation suite should be adapted as required.

The Path to Agile



In the beginning, the universe was created, and if you are a fan of the late, great Douglas Adams you will know this has made a lot of people very angry and was widely regarded as a bad move. Many years later software development was created (some people also regarded this as a bad move) and as already discussed in the early day's very little high value testing was undertaken.

If you are an experienced, intelligent and sensible person you will know that software development is hard to do well. The question is why? Professionals in the software development business tend to be very clever, bright and hardworking people. As a rule, they do not plan to deliver software that is over budget, past its deadline, incomplete or littered with defects. Despite this these issues still, arise time after time. Computers are complex, so are networks. Many things can go wrong, software issues in web browsers and operating systems can also affect the stability of your code. Users can get it wrong and hit the wrong combination of keys on the keyboard. These are all risks that need consideration.

Agile was designed as an attempt to make the software development process better, faster and more effective. Since its initial conception, the methodology has steadily grown in popularity, usage and success. Before talking about agile let us first consider the older, traditional software methods of Waterfall and V-Model. Both methodologies were based on the following assumptions

1. The customer knew exactly what they wanted and could articulate upfront what it is they wanted in such a way that everyone involved could clearly understand the requirements.
2. The project would have a set of well defined, clear and

unambiguous requirements which had been discussed, considered and agreed by all concerned.

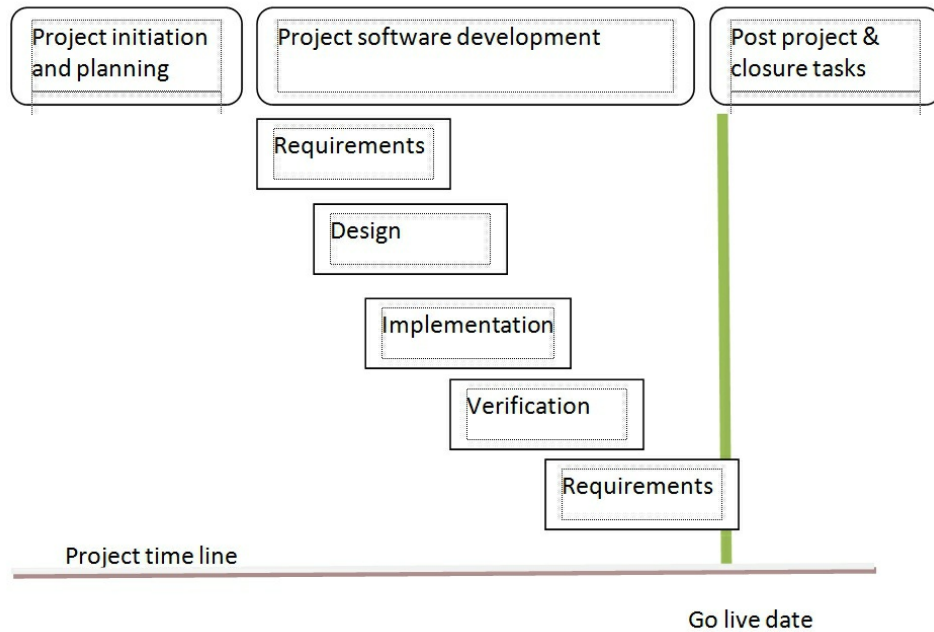
3. Everything that would be needed for the new system was known and available upfront.
4. Once the system was defined it was a set of static requirements and nothing would ever change.
5. Requirements were clearly and professionally detailed upfront. The system was then designed, coded and tested in a logical order. Such well-defined systems were always completed on time and with no major defects.



6. And of course,.....

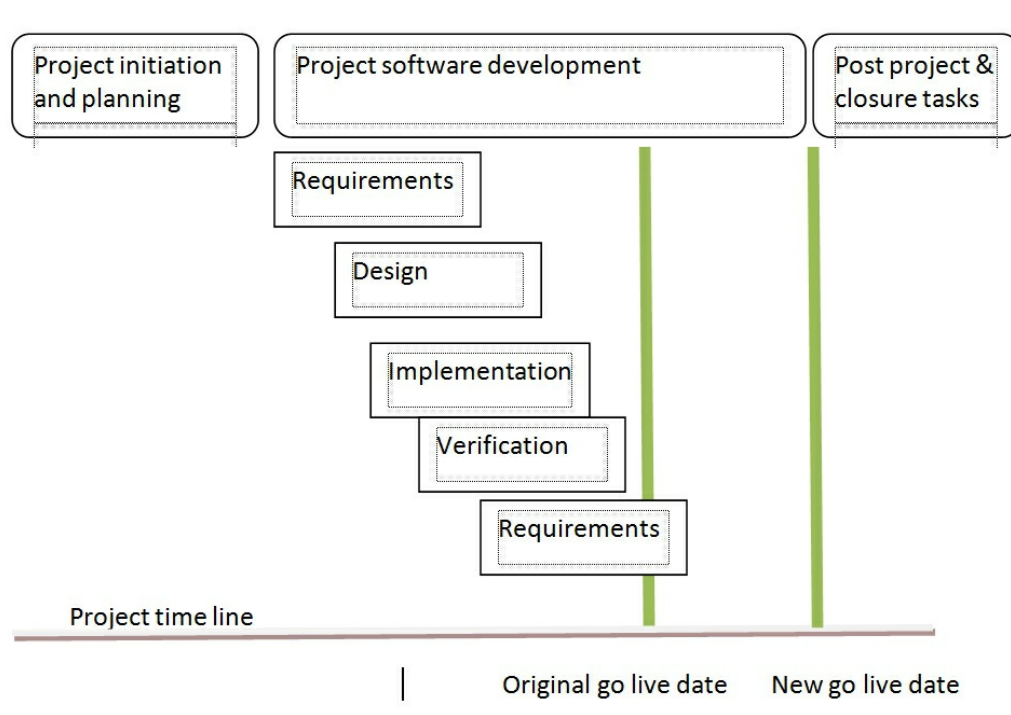
The Waterfall, in theory, was a good methodology and in a perfect world, it would have solved all software development issues. However, as we know the world is not perfect, humanity has made sure of that.

The Waterfall in theory



The waterfall is a truer picture of how many large-scale projects have fallen behind their release schedules. The waterfall is still a great methodology for small projects (and in many cases is still the best option) however with large projects its limitations are soon noticed. As a result of slipping time deadlines either testing is usually sacrificed, or parts of the application are held back for future releases.

The Waterfall in reality



To help rectify this issue the V-Model was created. In this model testing activities were aligned with the different levels of documentation produced. The model included feedback loops so that improvements could be made before the application was released. So, let us now look at this model.

The V-Model

Although many variants of the V-model exist today, a common type of V-model uses four test levels. These are:

- Component (unit) testing
- Integration testing
- Systems testing
- Acceptance testing

In the real world, a V-model project may have more, fewer or different levels of development and testing. This will depend on the project complexity and the software product. For example, there may be component integration testing after component testing, and system integration testing after system testing. While this was an improvement over the waterfall method it was still very restrictive in many areas and did not easily allow change or adaption. This opened the door for a new method and the ideas that became Agile

began to form.

In the early days, these Iterative incremental development models slowly began to gain popularity as their worth in large complex projects began to be realised. Iterative-incremental development is the process of establishing requirements, designing, building and testing a system in a series of short development cycles (sprints).

Examples are:

- Agile development models.
- Prototyping,
- Rapid Application Development (RAD)
- Rational Unified Process (RUP)

A system that is produced using these models may be tested at several test levels as part of every single iteration. An increment added to others developed previously, forms a growing partial system, which should also be tested. Regression testing is increasingly important on all iterations after the first one. Verification and validation can be carried out on each increment.

In the 1990s, these lightweight approaches gained popularity in a concerted effort to come up with an effective alternative to Waterfall. RAD, or Rapid Application Development, relied on building prototypes to allow requirements to emerge and elicit frequent feedback.

Despite the improved methods RAD still managed to get a bad name; however, this was down to bad planning and implementation on some major, high profile projects rather than the agile framework itself. For this reason, the term RAD has now been buried and replaced with Agile to allow these early mistakes to be forgotten in the dark and dusty depths of time.

The Scrum and XP (Extreme Programming) methodologies began to take root, both placing a heavy focus on short iterations to allow frequent delivery of software. In general, to serve the actual business needs and improve software project success rates. The early origins of agile go back before the software was used however the ideals of agile are easily transferable to most industries.

A brief history in the time of the agile framework

1948: The Toyota Way is conceived by Taiichi Ohno, Shigeo Shingo and Eji Toyoda.

1990: Rapid Application Development (RAD) is documented by James Martin.

1995: Ken Schwaber and Jeff Sutherland present a paper on Scrum at the OOPSLA (Object-Oriented Programming Systems, Languages and Applications conference).

1999: Kent Beck publishes Extreme Programming (XP) Explained, a very good book that everyone should read.

2001: The Agile Manifesto is created, read on for more information.

February of 2001 is now a pivotal date in the history of agile. On this date, a group of highly ambitious and talented developers who shared a joint interest in advancing lightweight development methodologies got together to discuss their views and try to forge some common ground.

From this get together the concept of agile was born. The developers who created agile all understood the importance of creating a software development model in which each iteration in the development cycle would learn from the previous iteration. The result was a methodology that was more flexible, efficient, and team-oriented than any of the previous models.

No matter what agile methods you use they all adhere to 12 core principles for guidance. It is the adherence to the guidance provided by the manifesto and principles is what makes a software development team agile, not a specific process, tool or label.

So, what is Agile and what are its foundations?

- Agile is not a development tool, rather it is a collection of evolving delivery and management frameworks for dynamic and innovative delivery environments, such as software development.

- Individually and collectively the frameworks are focused on ensuring that the highest priority is to satisfy the customer(s) through the early and continuous delivery of a valuable product, listening to the customer and by adapting to changing requirements.

The Manifesto for Agile Software Development

Individuals and interactions	over	processes and tools
Working software	over	documentation
Customer collaboration	over	contract negotiation
Responding to change	over	following a plan

The 12 principles behind the Agile Manifesto

Your highest priority is to satisfy the customer through the early and continuous delivery of valuable, usable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Businesspeople and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is a face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximising the amount of work not done--is essential.

The best architectures, requirements and designs emerge from self-organising teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

So, is the company you work for agile? Come on now, be honest and true. The answer is probably not. While it may be true that you have a task board standing tall and proud in the middle of the main meeting room and some of the teams undertake daily stand-ups which may, or may not relate to work, this does not mean your company is truly agile. The agile methodology is a framework that your organisation will either adopt fully and does not simply cherry-pick the parts they like or the parts that seem easy to implement or a

good idea at the time. That approach does not make you agile, it is like going out on a Friday night and drinking ten pints of lager or two bottles of wine before eating a fat-laden kebab or burger on the way home and saying you are healthy because you also drank a diet coke at the same time. For agile to work fully for your company you must adopt it fully.

So, think about the question again, is your company agile? Now answer honestly before reading further.

The Agile Process Framework

Agile versus traditional waterfall

Metric	Waterfall	Agile
Planning scale	Long-term	Short-term
Distance between customer and developer	Long	Short
Time between specification and implementation	Long	Short
Time to discover problems	Long	Short
Project schedule risk	High	Low
Ability to respond	Low	High

Agile Software Delivery

The level of risk associated with software development is increasing as the level of complexity in software and associated hardware also increases. As the level of risk increases so does the chance of failure and litigation if things go drastically wrong. To summarise this

- The ever-increasing complexity of systems being built or modified.
- The need to integrate new systems and technologies with legacy systems.
- Reduced timescales for delivery resulting in a reduction of testing.
- Internet-based solutions now must cater to a global market.

These pressures have given rise to the need to streamline the delivery cycle to provide greater flexibility and the ability to embrace change. These methods are now collectively called Agile Methods. The level of agile usage has increased greatly over the past decade. This is driven primarily by:

- The desire for faster (and therefore cheaper) time to market.
- A view that traditional methods are too process heavy, expensive and wasteful.

- The need to respond better to business and market change.
- Improved quality and usability.
- The realisation that new information and changing news during a development life cycle must be easily integrated.

Agile processes always value communication over documentation but at no point does any agile method say 'No Documentation' this is a myth that some companies use as a barrier to agile. This no documentation idea was a myth brought about by poor understanding of agile by those who did not want to change.

To put it correctly the concept of just enough documentation is the correct agile way. Exactly how much documentation is required will vary from project to project and affected by these requirements and others:

- Knowledge of implied requirements
- Audit Requirements
- Product Lifetime
- The fluctuation of team members
- Regulatory requirements

In other words, just enough valid documentation at just the right time for just the right audience will provide the correct information for the correct people.

Spotting when too little documentation is being produced is a critical exercise, below are some pointers that will help achieve this goal.

- Too many defects and bugs are being returned as not fixed or not fully fixed.
- At the review meeting completed products are not getting accepted as done by the product owner.
- Requirements are being misunderstood or missed completely.

- Too much time is spent gathering information at the beginning of each individual sprint.
- In long term projects, it is becoming harder and harder to understand the status quo of the system due to a lack of product documentation.

If one or more of these issues are identified, then an urgent review of the documentation process should be undertaken. When the documentation level has been optimised then the efficiency of the entire team will be improved. This opens the door to the world of being able to continuously deliver working software while allowing for and supporting changing requirements.

A reminder of the Agile Manifesto

Individuals and interactions	over	processes and tools
Working software	over	documentation
Customer collaboration	over	contract negotiation
Responding to change	over	following a plan

Also, a reminder of the 12 Principles of Agile Software Development

Your highest priority is to satisfy the customer through the early and continuous delivery of valuable, usable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Businesspeople and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is a face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximising the amount of work not done--is essential.

The best architectures, requirements and designs emerge from self-organising teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

So now you are getting into this agile development world let us look at some of the key principles:

Iterative: Agile software processes correctly acknowledge that we often get things wrong before we get them right. Therefore, agile processes focus on short cycles (usually 1-month sprints). Within each cycle, a certain set of activities is agreed and completed (if all goes well). These cycles will be started and completed in a matter of weeks. However, a single cycle (also called iteration) will probably not be enough to get the more complex element's 100% correct. Therefore, the short cycle is repeated many times to refine the deliverables and they are considered done, complete and fit for

purpose.

Incremental: An agile process does not try to build the entire system at once (that would be the good old waterfall method). Instead, it partitions the nontrivial system into increments which may be developed in parallel, at different times, and different rates. The programmers will then unit test each increment independently. When an increment is completed and tested, it is integrated into the system for further testing.

Collaborative: One cool feature of agile processes is that they help foster communication among team members. This communication is a vital part of any software development project. When a project is developed in pieces, understanding how the pieces fit together is vital to creating the finished product. However, there is more to integration than simple communication. Quickly integrating a large project while increments are being developed in parallel requires collaboration, teamwork and a good general understanding of the product under development and the desired end goal.

Adaptive: During any monthly iteration, new risks may be exposed which require some activities that were not planned previously. The agile process adapts the process to attack these new-found risks. If the goal cannot be achieved using the activities planned during the iteration, new activities can be added to allow the goal to be reached. Similarly, activities may be discarded if the risks turn out to be ungrounded.

So that is the key principles of agile but now what about the agile team at the centre of all this agile development, just who are these players? First, to help understand how the agile team works best let us first look at a Traditional Organisational Structure as shown below.

Project Leader	Development	Testing
Project Leader 1	Programmer 1	Tester 1
Project Leader 2	Programmer 2	Tester 2
Project Leader 3	Programmer 3	Tester 3

In this type of organisation, everyone is separate. Typically, the Businesspeople (project managers etc will reside on the top floor when they talk to the customers via email and telephone with frequent meetings at

expensive restaurants and tapas bars.)

The testers will be in the middle ground where they do other administration duties when not on a testing task. Finally, the programmers will be down in the basement writing highly complex code.

On a business level, there is little cross-team communication and usually, the development teams are given very little knowledge of future release plans, testers even less so. So now let us look at a modern agile Structure.

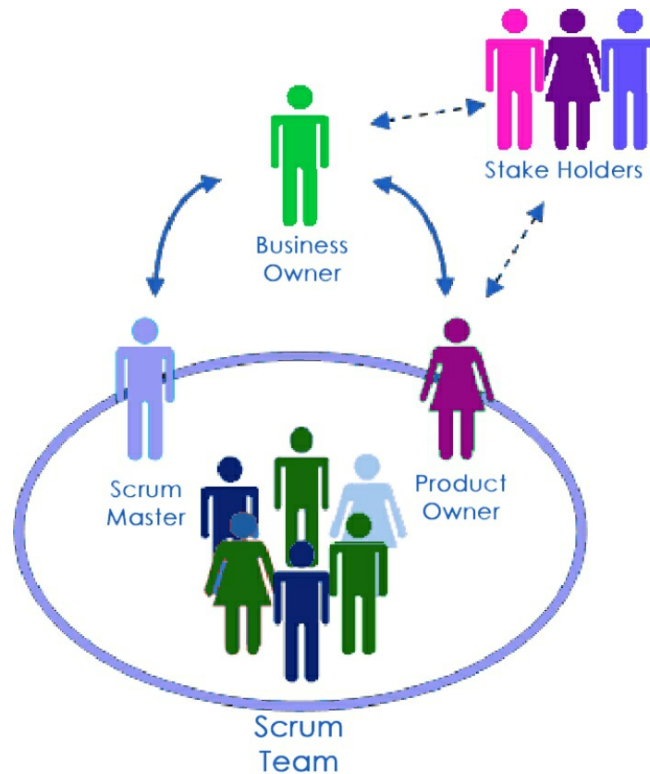
Project Leader	Development	Testing	
Product Owner 1	Programmer 1	Tester 1	Team 1
Product Owner 2	Programmer 2	Tester 2	Team 2
Product Owner 3	Programmer 3	Tester 3	Team 3

Here we see a more team-orientated structure. Such teams (cells) may change from project to project but while a team exists, they work together and communicate within the cell. All members of the team are present at regular development meetings and the daily scrums help improve the team group knowledge and help gel the cell into one tight cohesive unit. On the next page is an image that shows how the scrum team fits into the bigger picture. This also shows how and when the scrum master is involved.

So now let's have a look at these individual people and groups.

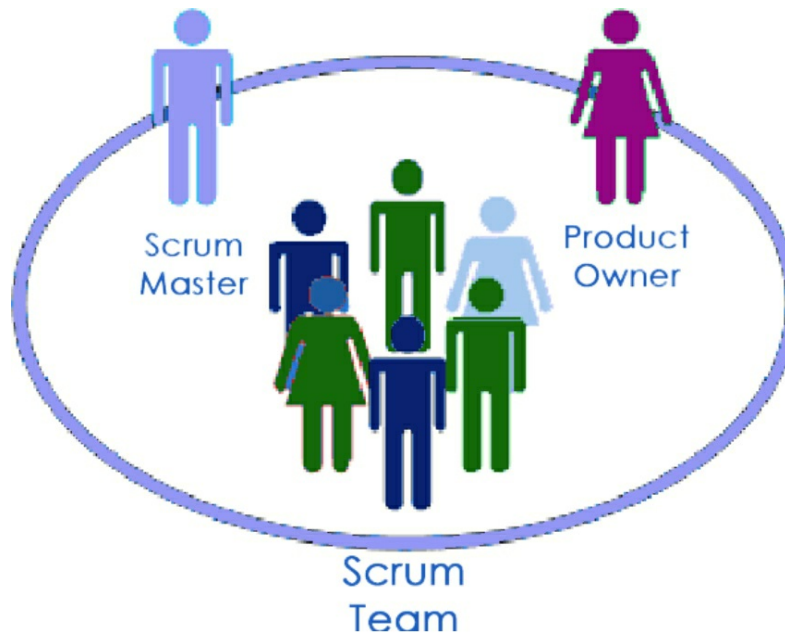
The Stakeholder(s)

A *stakeholder* is an organisation who will be financially impacted by the outcome of the solution and is clearly much more than just a standard end-user. A stakeholder may be one of the following:



- A direct or indirect user of the system.
- A manager of a group of users.
- A senior manager or company director.
- Developer(s) who are working on other systems that integrate or interact with the one under development.
- Operations or IT staff member.
- The owner who funds the project.
- Auditor.
- Your program/portfolio manager.

The Product Owner



The product owner is the one team member who speaks on behalf of the customer. This person represents the needs and desires of the stakeholder community to the agile delivery team; as a result, they should be considered an important part of the team and a vital gateway. He or she is accountable for ensuring that the development team delivers the required value to the business. The Product Owner is also responsible for maximising the value of the product and the work of the Development Team

The Product Owner writes (or has the team write) customer-centric items (also known as user stories). They clarify any details regarding the solution and are also responsible for maintaining a prioritised list of work items that the team will implement to deliver the solution. They will then rank and prioritise them, and then add them to the product backlog; this task is the sole responsibility of the product owner. While the product owner may not be able to answer all questions, it is their responsibility to track down the answer promptly so the team can stay focused on its current tasks. Each agile team, or sub-team in the case of larger projects organised into a team of teams, has a single product owner.

The Product Owner is always just one person, they are never a committee. The Product Owner may represent the desires of a committee in the Product

Backlog, but those wanting to change a Product Backlog item's priority must address the Product Owner directly.

Therefore, every scrum team should have only one Product Owner and this role should never be combined with that of the Scrum Master. In an enterprise environment, though, the Product Owner is often combined with the role of Project Manager as they have the best visibility regarding the scope of work (products).

Additionally, the product owner owns the following roles:

- Communicates the project status and represents the work of the agile team to key stakeholders
- Develops strategy and direction for the project and sets long- and short-term goals
- Understands and conveys the customers' and other business stakeholders' needs to the development team
- Gathers, prioritises and manages product requirements
- Directs the product's budget and profitability
- Chooses the release date for completed functionality
- Answers questions and makes decisions with the development team
- Accepts or rejects completed work during the sprint
- Presents the team's accomplishments at the end of each sprint

The Scrum Master

The daily scrum is overseen by a scrum master, although they do not have to attend each meeting in person. They are responsible for ensuring Scrum is fully understood and enacted daily. Scrum Masters do this by ensuring that the Scrum Team adheres to Scrum theory, practices, and rules.

The scrum master is also accountable for removing impediments to the ability of the team to deliver the product goals and deliverables. The scrum master is not a traditional team leader or project manager but acts as a buffer between the team and any distracting influences.

During the daily scrum, the scrum master ensures that the Scrum process is used as intended, for example, a short, daily meeting held every day. He or She is the enforcer of the rules of Scrum, often chairs key meetings, and challenges the team to improve. The role has also been referred to as a ***servant-leader*** to reinforce these dual perspectives.

Servant leadership is both a leadership philosophy and set of leadership practices. Traditional leadership generally involves the accumulation and exercise of power by one at the “top of the pyramid.” By comparison, the servant-leader shares power and will put the needs of others first and helps people develop and perform as highly as possible.



The Development Team

The Development Team is responsible for delivering potentially shippable increments of the product (working software) at the end of each Sprint. A development team is typically made up of 3–9 individuals with cross-functional skills who do the actual development work. These skills include the following

- Analyse the needs
- Designing the system
- Developing Code
- Testing Code (QA)
- Create/Update the documentation

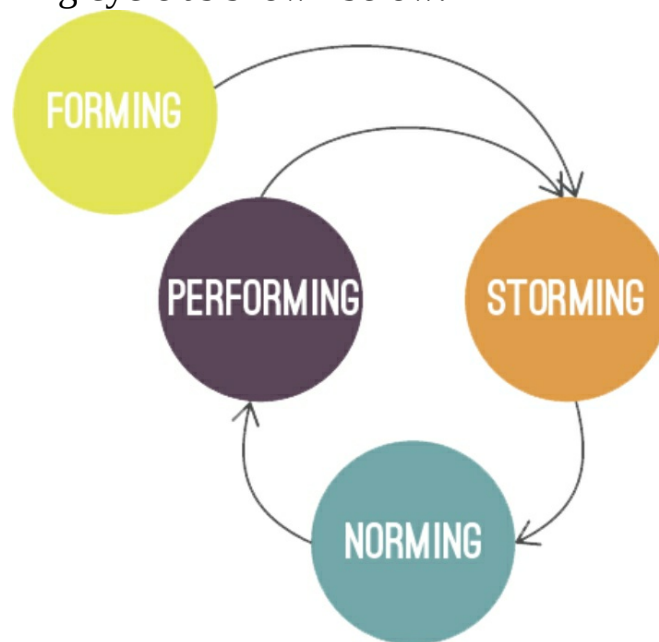
No matter what your job title or your role within the team you are as important as the other members. The role of each team member focuses on producing the actual solution for stakeholders. Team members perform testing, analysis, architecture, design, programming, planning, estimation, and many more activities as appropriate throughout the project.

The optimal Development Team size is always small enough to remain nimble and large enough to complete significant work within a Sprint. A team that is fewer than three Development Team members will suffer from decreased interaction and smaller productivity gains. Smaller Development Teams may also encounter skill constraints during any given Sprint, causing the Development Team to be unable to deliver a potentially releasable Increment. At the other end of the scale requiring more than nine members requires too much coordination. Large Development Teams generate too much complexity for an empirical process to manage. The Product Owner

and Scrum Master roles are not included in this count unless they are also executing the work of the Sprint Backlog.

It is unlikely that every team member has every single skill (at least not yet), but they have a subset of them, and good team members will strive to gain more skills over time. Team members should identify, estimate, sign-up for, and perform tasks and track their completion status.

Within the scrum environment, the development team is a self-organising entity even though there may be some level of interface with project management offices. Mature teams are built on trust and respect for each other's key skills and knowledge. New teams will need to go through a forming to performing cycle as shown below.



Forming:

- A high degree of guidance is required from management
- Individual roles are still unclear
- Processes usually not well established

Storming:

- Beginning to understand how team decisions will be made
- The team purpose is clear, but relationships are still forming

Norming

- Relationships are now well understood within the team
- Commitment to team goals now realised
- The team start to optimise team processes

Performing

- The team is now committed to performing well
- The team can now focus on being strategic
- The team runs well as a cell

When a team reaches level four (performing) they should be able to deliver high-quality increments regularly. The velocity of the team will also improve as they grow to understand each other's strengths and weaknesses. So, with the team in place it is time to go agile, I hope you are now feeling excited and eager to move forward.

For an agile team to strive onwards and start delivering deliverables that can be considered done and fit for purpose they need to work within the agile requirements specification, next we will consider some of the key points of this.

- **Active stakeholder participation:** Stakeholders should be available to provide key information promptly. They should also be able to make important decisions when required and be very actively involved in the development process using inclusive tools and techniques.
- **Prioritised requirements:** Agile teams implement requirements in priority order, as defined by their stakeholders, to provide the greatest return on investment possible. These priorities are agreed and ordered at the beginning of each sprint.
- **Requirements envisioning:** At the beginning of each agile project the team will need to invest some time to identify the complete scope of the project and to create the initial prioritised list of requirements. This effort should take a from a few days

and up to two weeks, assuming you can overcome the logistical challenges associated with getting the right people involved.

- **Test-driven development:** The basis of this is to write a single test, either at the requirements or design level. Then write just enough code to fulfil that test. TDD is a JIT approach to detailed requirements specification and a confirmatory approach to testing (see Chapter 14).
- **Daily Scrum:** In the daily scrum the team members will each answer three questions, these are: What I did yesterday; what I plan to do today; what impediments are stopping me from completing these tasks. The scrum takes place every morning at the same time and at the same place. The job of the scrum master is to manage the scrum and deal with any impediments.
- **XP and Requirements:** The XP approach is to capture requirements through User Stories. This has become a very common method in Agile and has proven a very popular and effective method.

The focus of XP is customer satisfaction. XP teams achieve high customer satisfaction by developing features when the customer needs them. New requests are part of the development team's daily routine, and the team must deal with requests whenever they become known. The team automatically organises itself around any problem that arises and solves it as efficiently as possible.

XP is based on a set of 29 well designed and intuitive rules, these are detailed as follows

- Planning
 - User stories are written.
 - Release planning creates the release schedule.
 - Release often Make frequent small releases.
 - The project is divided into iterations.
 - Iteration planning will start each iteration.
- Managing

- Optimise last Give the team a dedicated open workspace.
 - Always work at a sustainable pace.
 - A stand-up meeting starts each day.
 - Project Velocity is measured.
 - Move people around.
 - Fix XP when it breaks.
- Designing
 - Simplicity.
 - Choose a system metaphor.
 - Use CRC cards for design sessions.
 - Create spike solutions to reduce risk.
 - No functionality is added early.
 - Refactor whenever and wherever possible.
- Coding
 - The customer is always available.
 - Code must be written to agreed standards.
 - Test-Driven Development always designs the unit test first.
 - All production code is pair programmed.
 - Only one pair integrates code at a time.
 - Continuous integration.
 - Set up a dedicated integration computer.
 - Use collective ownership.
- Testing
 - All code must have unit tests.
 - All code must pass all unit tests before they can be released.
 - When a bug is found tests are created.
 - Acceptance tests are run often, and the score is published.

So those are the rules of XP, now let us discuss some of the major key features in more detail.

Pair programming

This is sometimes referred to as Peer programming. All code to be sent into production is created by two programmers working together at a single computer terminal. Pair programming has been shown to increase software quality without impacting time to deliver. At first, it may appear counter-intuitive, but 2 programmers working at a single computer will add as much functionality as two working separately except that it will usually be much higher in quality as each member of the team strive to outdo the other with well-designed and clever code. With this increased quality comes big savings later in the project as fewer bugs will require fixing.

One important consideration that management will need to consider is that pair programming is a social skill that for some programmers will take the time to learn. However, you are striving for a cooperative way to work that includes give and take from both partners regardless of experience, age and company status. Therefore, programmers will learn to cooperate if they are to fit into modern agile teams.

This is one area of agile that is most likely to experience resistance from programmers. So, of them will take on the methodology happily, others will kick up a fuss over the idea and a few may even leave rather than adjust. These are issues to be considered when planning a move to agile and whatever is decided during the planning phase should remain policy no matter what resistance is raised.

No functionality is added early

Keep the system uncluttered with extra stuff that you guess will be used later. History has demonstrated that only 10% of unrequested extra functionality that has been added will ever get used, so in effect, programmers are wasting 90% of their time by putting it in there at the early stages.

Not only is this a waste of time for the programmer it also increased the testing workload and increases risk. We are all tempted to add functionality and show how clever we are now rather than later because we see exactly how to add it or because we believe it would make the system so much better. It seems like it would be faster to add it now.

When a bug is found

When a bug is found tests should always be created to guard against it

coming back. Also, a bug that is discovered in production always requires an acceptance test be written to guard against it. As a result, all bugs should be recorded even if the programmer insists it is not necessary.

Move people around

Move people around to avoid serious knowledge loss and coding bottlenecks. If only one person on your team can work in each area and that person leaves or just has too much to do you will find your project's progress reduced to a crawl. This also stops little gangs forming who will work together and against anyone they collectively do not like or the feel to be less worthy.

Integrate often

Developers should be integrating and committing code into the code repository every few hours, whenever possible. In any case, they should never hold onto changes for more than a day on their local computer. Continuous integration often avoids diverging or fragmented development efforts caused where developers are not communicating with each other about what can be re-used, or what could be shared. Everyone needs to work with the latest version. Changes should not be made to obsolete code causing integration headaches. This also applies to QA's who are working on automation tasks, their code should also be merged into the shared repository as often as possible.

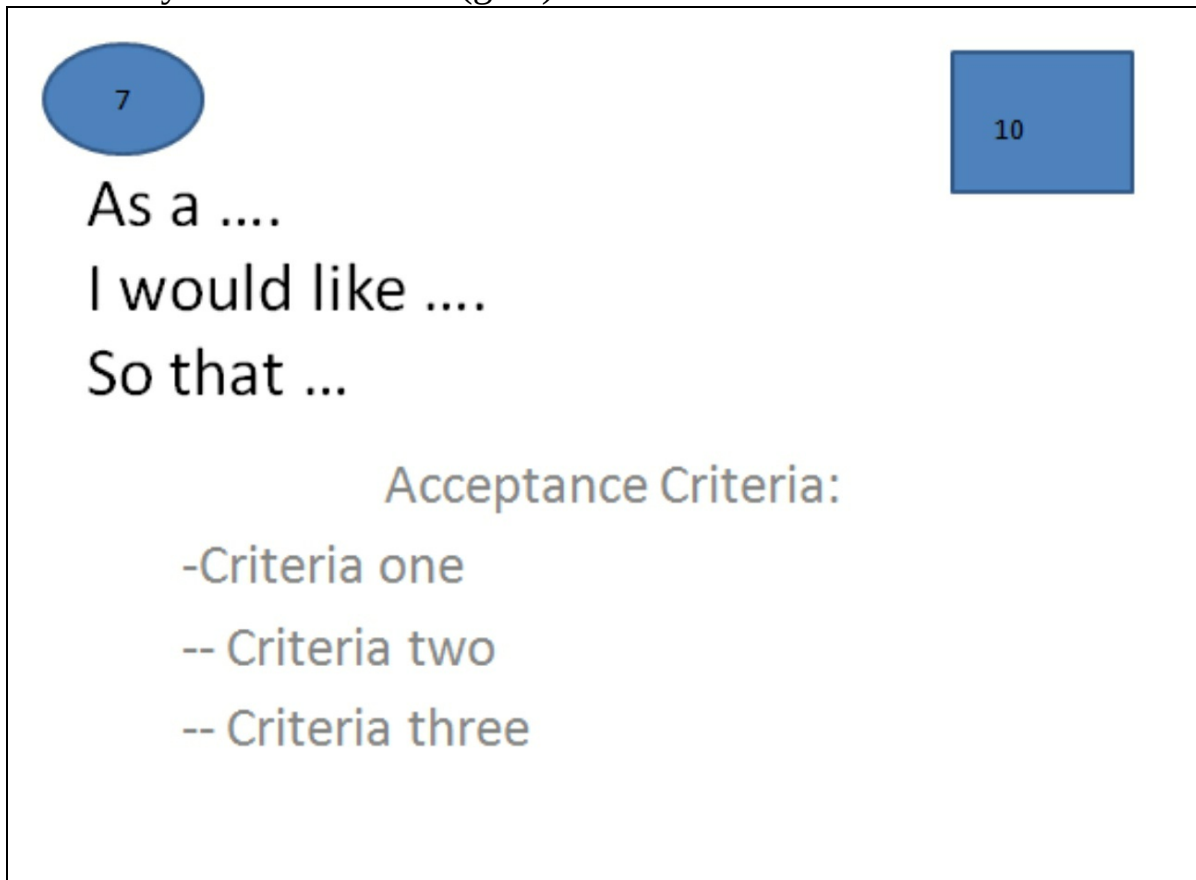
Collective Ownership

Collective Ownership encourages everyone to contribute new ideas to all segments of the project. Any developer can change any line of code to add functionality, fix bugs, improve designs or refactor. No one person becomes a bottleneck for changes. This is hard to understand at first. It's almost inconceivable that an entire team can be responsible for the system's design. Not having a single chief architect that keeps some visionary flame alive seems like it couldn't possibly work.

User Stories

These are lightweight, very brief descriptions that are written by the user/stakeholder. They should say what they want the system to do. A user story could be 'Create an Add customer page'. A user story briefly explains:

- the person using the service (actor)
- what the user needs the service for (narrative)
- why the user needs it (goal)



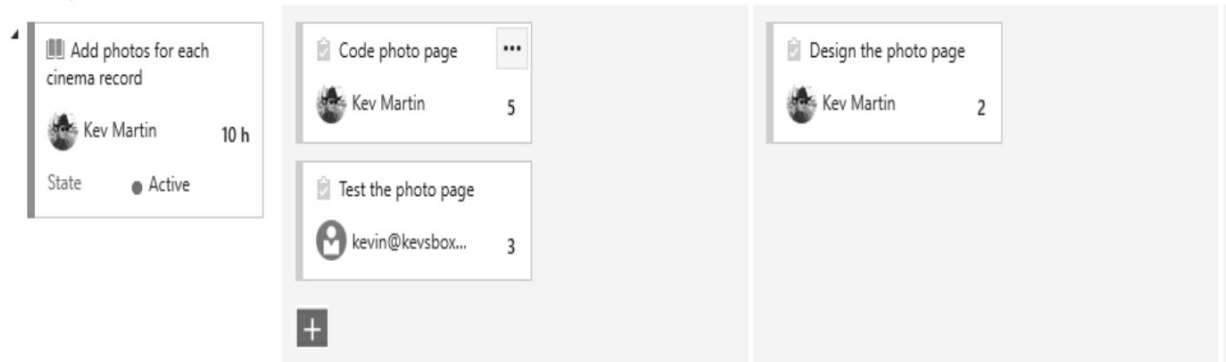
User Stories are discussed further later in the book

So those are the major key features of XP. Another interesting agile tool that can be used by itself or alongside other agile methods such as XP is Kanban. Kanban comes from the Japanese terms Kan (visual) and Ban (board) and is quite simply that, in other words, a visual board or taskboard. It is a system to control the logistical chain from a production point of view and is an inventory control system. Kanban was developed by Taiichi Ohno, an industrial engineer at Toyota, as a system to improve and maintain a high level of production. Today it is a very popular tool for use in software development.

kevsbox Team cinema photos

Backlog Board Capacity

^ Collapse all



The Kanban Method is used by organisations to manage the creation of products with an emphasis on continual delivery while not overburdening the development team. Like Scrum, Kanban is a process designed to help teams work together more effectively.

The Kanban method is rooted in four basic principles:

Start with what you do now:

The Kanban method does not prescribe a specific set of rules or process steps. The Kanban method starts with the roles and processes you have and stimulates continuous, incremental and evolutionary changes to your system.

Agree to pursue incremental, evolutionary change:

The company (or team) should agree totally and completely that continuous, incremental and evolutionary change is the way to make system improvements and make them stick.

Sweeping changes may seem more effective within an organisation but also have a higher failure rate due to resistance and fear in people within the organisation. The Kanban method encourages continuous small incremental and evolutionary changes to your current system which people tend to be more comfortable with.

Respect the current process, roles, responsibilities and titles:

It is likely that the organisation currently has some elements that work

acceptably and are worth preserving. Change for change's sake is not always the best way forward. By agreeing to respect current roles, responsibilities and job titles you can eliminate some of the initial change fears.

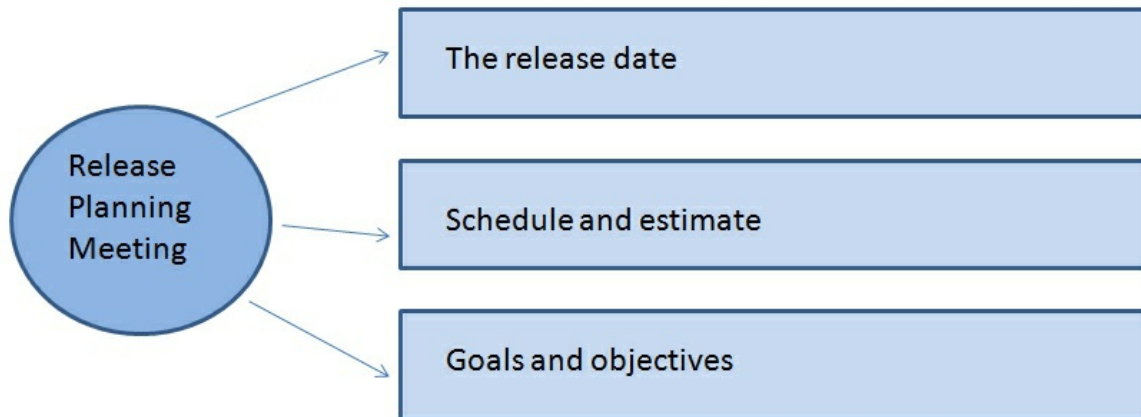
Leadership at all levels:

Acts of leadership at all levels in the organisation from individual contributors to senior management should be encouraged. Meetings are an important part of agile software development. Meetings keep everyone involved in the development lifecycle updated, informed and in contact with progress. Regular meetings also help identify forthcoming problems and roadblocks at an early stage. Next, we will discuss the five types of agile meetings.

The five agile Meetings

1: The Release Planning Meeting

A vital and important part of agile is regular meetings. There are five meetings of importance with agile, so let us now discuss the first of these.



This is the release planning meeting. This meeting takes place at the start of every release cycle. The purpose of the release planning meeting is also to get an idea of what stories the team will try to finish by the release end date.

The product owner will select the stories in a logical order of importance. There may be dependencies which will affect this order as well but in general, the highest priority will be started first.

It is also important that the team understands the possibility that not all the stories will get completed by the release end date. One of the basic premises of agile is to deliver working software, so it is important to have the highest-value stories completed first (barring dependencies) so that the software you do deliver meets the customer's needs. The final acceptance criteria should be agreed at this meeting and the result of this meeting is the product backlog.

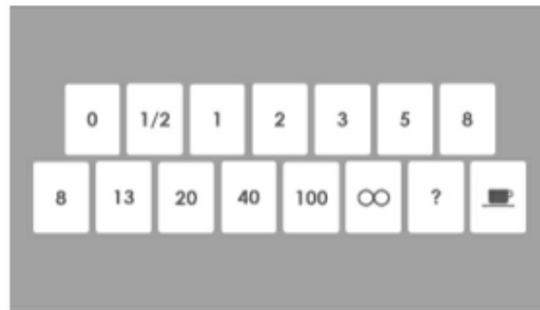
The product backlog is a dynamic list of stories. These stories may be edited, deleted or added at any time during the release cycle. As previously discussed, the list is prioritised and items with the highest priority should always be completed first. The backlog is progressively refined, adjusted and

improved during the life cycle.

When the product owner sets the scope of the next iteration, he or she needs to know that the scope is the right size for the team, in other words, there isn't too much work to get done in the iteration. Like any other developers, agile team members estimate their own work. Unlike other developers, agile team members usually estimate in points. Points represent a size-based, complexity-based approach to estimation. Points are assigned in whole numbers (1, 2, 3, 4 and so on with no fractions or decimals). A common method of sizing is to use the Fibonacci number system (1, 2, 3, 5, 8, 13 etc). These numbers represent relative sizes and complexity of work items. Small and simple tasks are one-point tasks, slightly larger/more complex tasks are two-point tasks, and so on with 10 usually being the highest in a decimal ranking system and a higher number for the Fibonacci system.

Points can be thought of as shirt sizes. There can be small, medium, large, extra-large, and potentially other sizes (extra small, extra extra-large and these days even xxxx large). These sizes are relative to the team using them; no formal regulation dictates how the much larger medium is compared to small.

Planning Poker



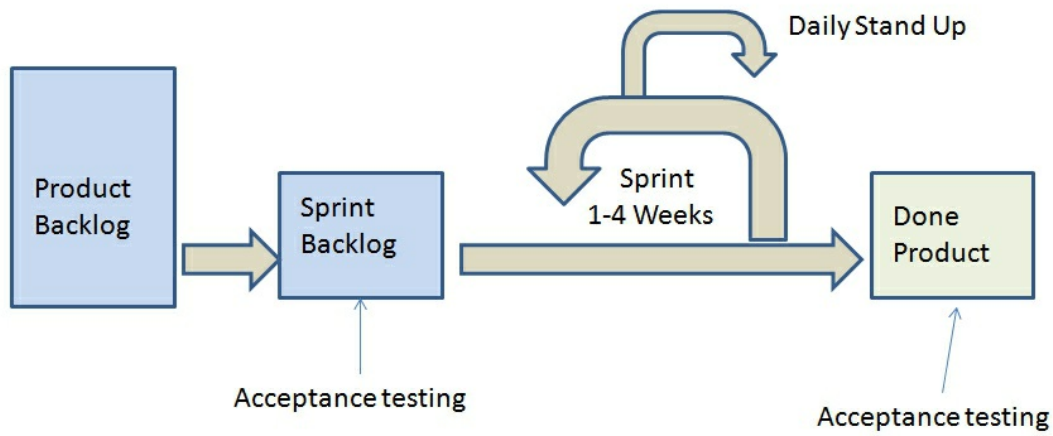
As you refine your requirements, you will also need to refine your estimates also. Planning Poker is a technique to determine user story size and build consensus with the development team members. Planning poker is a very popular and straightforward approach to estimating story size.

To play planning poker you need a deck of cards with point values on them. There are plenty of poker tools and mobile apps that are free to use, or you can do it yourself with index cards. The numbers on the cards are usually from the Fibonacci sequence.

Only the development team plays estimation poker. The team leader and product owner do not get a deck and do not provide estimates. However, the team leader can act as a facilitator and the product owner usually reads the user stories as well as providing detailed information on the stories.

In practice, one team's three-point size estimate for a work item may correlate to another team's two-point estimate for an identical work item. Teams need only agree on what size and complexity correspond to what point count and remain internally consistent in their use. These points are used to help estimate the team's velocity or work output.

2: The Iteration/Sprint Meeting



The next meeting is the iteration/sprint planning meeting. In Scrum, every iteration or sprint (typically 2-4 weeks) begins with the sprint planning meeting. At this meeting, the Product Owner and the development team will negotiate which stories the team will tackle during the current sprint.

These meetings are typically time-boxed to eight hours, this helps to focus the group and ensure no time is wasted on idle chatter such as the weekend's football results or what car each team member would like to purchase.

The final choice of what stories are to be included will always rest with the Product Owner. Typically, they will select those with the highest business value, however, the team has the power to push back and voice concerns about impediments and dependencies.

When the story list is decided and agreed this becomes the required work for the current sprint. Team members choose what they will each do. Nothing is assigned or pushed onto anyone. The result of this meeting is the sprint backlog.

So, as you can see the sprint backlog is a negotiated set of items/stories from the product backlog that a team commits to code and test during the timebox of the current sprint. Items in the sprint backlog are broken into detailed tasks for the team members to complete. The team works collaboratively to

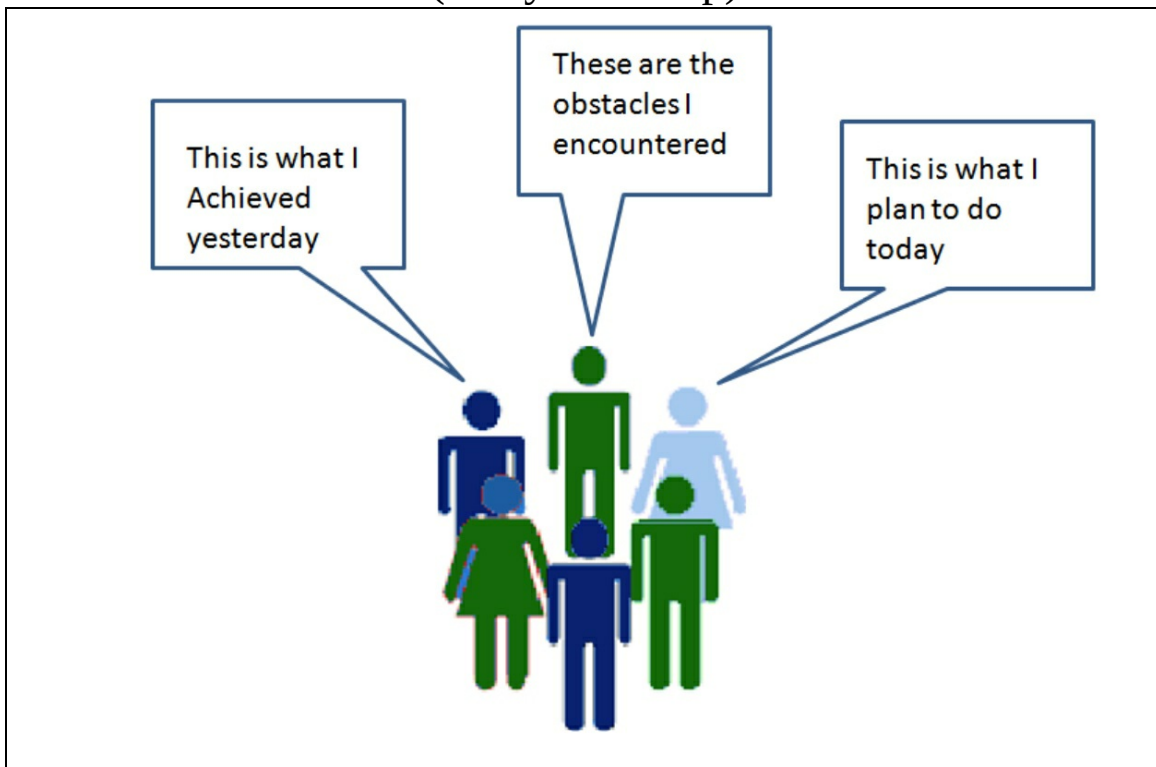
complete the items in the sprint backlog, meeting each day (during a daily scrum) to share struggles and progress and update the sprint backlog and burn down chart accordingly.

A sprint will typically last for 4 weeks, this can vary slightly but 4 weeks is by far the most common time frame. This is also known as time-boxed development. In time-boxed development, the end calendar date for the Sprint is determined at the outset. The Sprint is then complete once that day is met and all work that's going to be launched needs to be done.

This work will have passed through QA and testing and is deemed and ready for launch. Time-boxing the Sprint requires a much sharper focus on prioritising work since you can't push the end date if work isn't done.

Features that aren't complete can get pushed to the next Sprint. However other constraints or concerns may result in them being placed back in the backlog. This forces decision-making much earlier on what will and what won't make it into the Sprint. Hard decisions on prioritising work should be done right at the outset. Decisions on what features get pushed to the next Sprint must happen very early, this helps ensure that no testing cannot be completed before the end of the Sprint.

3: The Daily Scrum Meeting (Daily Stand-up)



The heart of the Scrum process is the daily stand-up meeting, also known as the daily Scrum. No other meeting captures Scrum's emphasis on communication and transparency quite like the stand-up. This meeting helps ensure that the entire development team is always on the same page within the current sprint. Every day, the scrum team will gather together, usually in a team room or private office to report on the progress made since the last meeting, goals for the next one, and any impediments blocking their path. These reports are often phrased as responses to the following three questions:

- What have I done since the last Scrum meeting (yesterday)?
- What will I do before the next Scrum meeting (tomorrow)?
- What prevents me from performing my work as efficiently as possible (impediments)?

All development team members must participate in the entire daily scrum meeting and they must be prepared to help each other. The meeting is just 15 minutes (never any longer), so everybody must participate and pay attention

in what tasks are done and what needs to be developed and make suggestions on other member's items when needed.

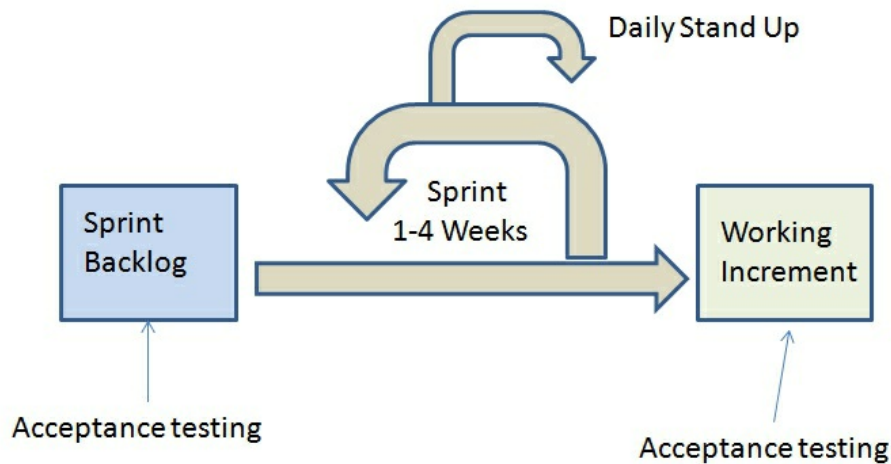
It is the scrum master's job to deal with any reported impediments and to ensure the scrum meeting happens, runs correctly and is concluded within the agreed time frame. This is the meeting that those team members who are not fully committed to agile will struggle hardest with. They will consider these fifteen minutes every morning a total waste of their time even though they would probably spend that same time slot talking about other non-work-related items around the water or vending machine. Such people will be easy to spot; they are the ones sitting quietly with folded arms and a bored look on their face. They will say very little and will probably need prompting to divulge their three answers.

This behaviour needs to be tackled early. At first friendly conversations to try and find out why they are resisting the change and finally as a last resort moving the non-committed to either another department or another company. This may sound a little harsh however you should remember that there is always no I in TEAM.

I have always found that it is best to do the daily scrum meeting in front of Kanban or the Sprint Backlog. By using this method, the team can see clearly what the remaining items are to be developed, and the team members can discuss the chosen items to be developed on that day to achieve the maximum speed configuration to achieve the sprint goal.

It is also good practice to ask individually each development team member about the feeling concerning achieving the sprint goal. If the majority answers consider difficult to achieve the goal, the team must find, in group, different ways to change this scenario. Also, you should focus solely on the daily scrum meeting; you will not have time for extra subjects besides the sprint backlog.

4: The Iteration (Sprint) Review



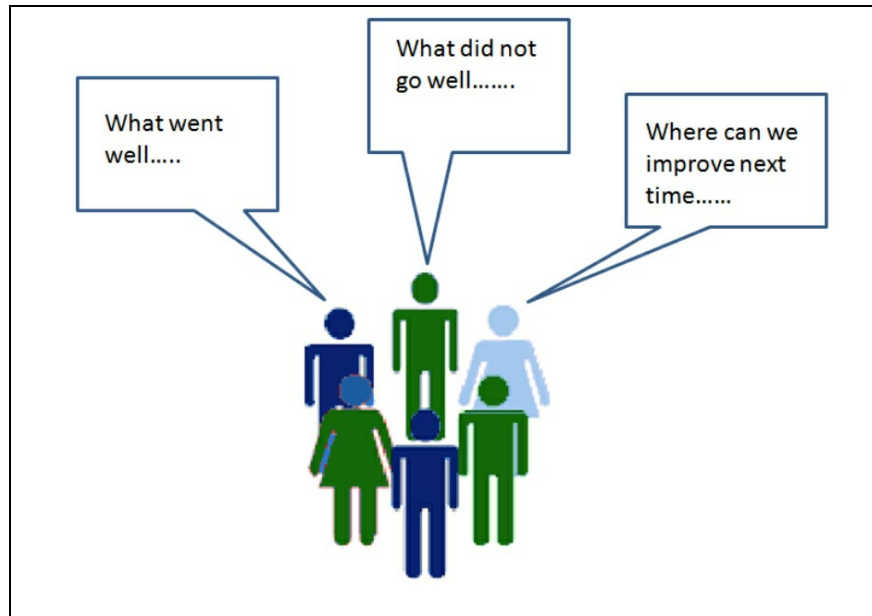
In Scrum, when the sprint ends, it's time for the team to present its work to the product owner for acceptance. This is known as the sprint review meeting. At this time, the product owner will go through the previously agreed sprint backlog and they will ask the team to present its work. The product owner checks the work against the acceptance criteria to determine if the work has been completed satisfactorily or not. Even if only 1% of a story remains to be completed by a team, the product owner must reject the story as unfinished, incomplete work carries unacceptable risks and should never be passed as done. Teams commonly discover that a story's final touches often excise the most effort and time, so awarding partial credit for an incomplete story can contribute to a slanted velocity. A sprint review is a time-boxed event of 4 hours for a monthly sprint, however, if the sprint was shorter than the review can also be shorter.

Preparation for the sprint review meeting should not take more than a few minutes. Even though the sprint review might sound formal, the essence of showcasing in agile is informality. The meeting needs to be prepared and organised, but that doesn't require a lot of flashy presentation material. Instead, the sprint review focuses on demonstrating what the development team has achieved and what is considered done.

As a result, you should gather sprint review feedback informally. The product owner or team leader can simply take notes on behalf of the development

team, as team members often are engaged in the actual presentation and resulting conversation. New user stories may also come out of the sprint review. These new user stories can be new features altogether or alterations to the existing code.

5: The Sprint Retrospective Review



When the sprint review meeting has been concluded the scrum master and the development team get together for a sprint retrospective review. This is a team led meeting however they can invite the product owner and other stakeholders if they feel their input will be valuable. This is a time-boxed event lasting 3 hours.

During this meeting the team will consider three important issues, these are:

- What went well?
- What did not go well?
- What improvements could be made for the next sprint?

Because the product owner does not sit in on these meetings this is a good opportunity for the team members to talk frankly about the sprint's successes and failures. This is an important meeting for the team because they have an opportunity to focus on its overall performance and they can identify potential strategies to improve its processes. What should be avoided always is these meetings being reduced to a blame and shame argument about who broke what and who did not do what adequately. If the meeting is poorly managed and some of the team have large ego's this can easily happen, it is at these points that the scrum master must be at their strongest. The scrum

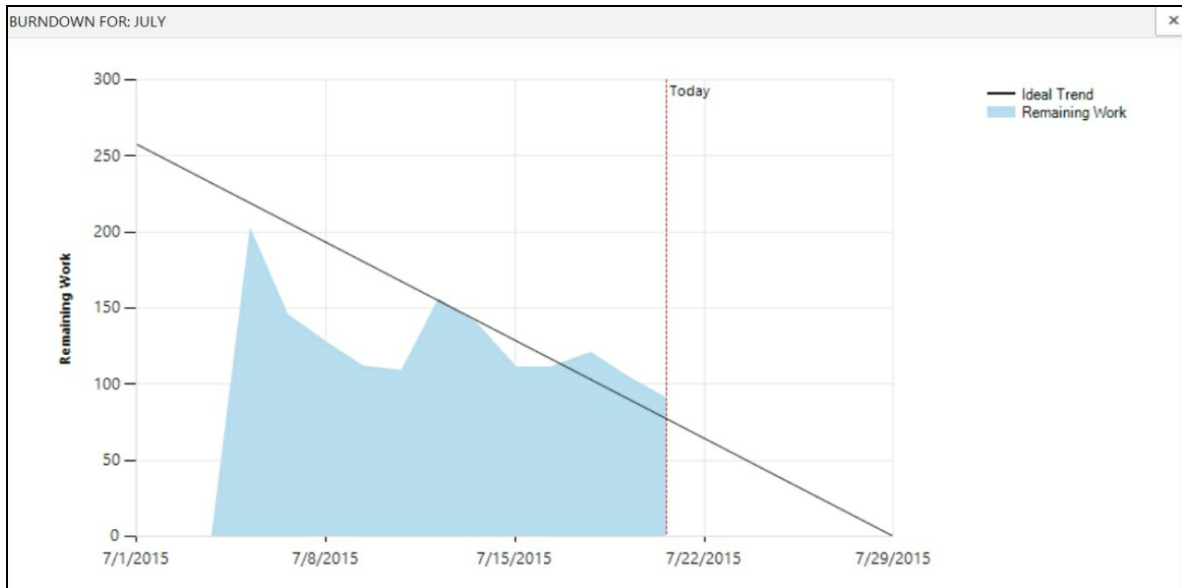
master can observe common impediments that impact the team and how they work together. From this, the scrum master can work out plans to resolve these impediments.

During every sprint the team needs current progress information; one of the most popular methods of keeping the team updated is the use of task boards. As previously mentioned, one of the popular styles is Kanban and most agile teams will deploy a task board in some form. Task boards are a good guide to progress for all the team members and stakeholders. These are very easy to understand visual representation of the current sprint state.

So, the overall goal of the iteration retrospective is to continuously improve your processes. Improving and customising processes according to the needs of each individual team increases team morale, improves efficiency, and increases velocity and work output.

We have already discussed task boards but to reiterate these are usually situated in a central location that is easily observed by most of the team that is based in the office. However electronic versions are becoming increasingly popular because they allow team members to access the data anywhere on the planet, this could be remote offices or even home workers. This is particularly useful in an organisation which has a global presence and team members can be on different continents. One example is available in Team Foundation Server which we will cover later in the book.

Another important tool for the agile team is the iteration burndown chart. This is a graphical representation of work left against remaining time within the current sprint. This is useful for predicting when all the work will be done and is also a good method for detecting potential deadline problems. Below is a typical example of a burndown chart.



These tools can help measure the team's performance which should improve over time as the team becomes more experienced and as they learn to work together as one efficient unit. This is known as team velocity.

The main idea behind velocity is to help agile teams estimate just how much work they can complete in each time period based on how quickly similar work was completed in previous sprints.

The following terminology is used in velocity tracking. To calculate velocity, a team-first must determine how many units of work each task is worth (10, 20 etc) and the length of each interval. During development, the team will keep track of all completed tasks and at the end of the interval count the number of units of work completed during the interval.

The team then writes down the calculated velocity in a chart or on a graph (paper or electronic). The first week will always provide little value but is essential to provide a basis for comparison and a datum. Each week after that, the velocity tracking will provide better information as the team provides more accurate estimates and becomes more used to the methodology.

The unit of work can either be a real unit such as hours or days or an abstract unit like story points or ideal days. Each task in the software development process should then be valued in terms of the chosen unit. By contrast, the interval is the duration of each iteration in the software development process for which the velocity is measured. The length of an interval is determined by

the team. Most often, the interval is a week, but it could also be as long as a month.

So that is the agile team, the methods they use and the tools they employ to get the job done. So, you are still here, this means that either you still want to be an Agile Quality Assurance Tester and/or you are finding this book a good read. Whatever the reason let us move forward to the next chapter, the Agile QA Tester.

The Agile QA Tester

So, you are still here, are you? Does that mean you still want to be an Agile QA Tester? If yes then good for you, so let us now look at what it takes to be a good agile tester. In previous chapters I have hinted at what makes a good tester, virtues such as patience, good people skills and a good eye for detail are essential. Other important considerations are a basic understanding of SQL (structured query language) and basic knowledge of Java or C# will also help. You will know when you have become a competent tester when you can indulge in what is known as experience-based testing.

So, what is experience-based testing?

Experience-based testing is where tests are derived from the tester's (you or your team) skill's and intuition as well as their experience with previously tested similar applications and technologies. When this style of testing is used to augment systematic techniques, these techniques can be useful in identifying special tests not easily captured by formal techniques, especially when applied to more formal approaches. However, this technique may yield widely varying degrees of effectiveness, this will depend on the QA's experience and capability. Essentially the more experienced and better trained the tester is the more valid this type of testing will become.

A commonly used experience-based technique is error guessing. In general, QA's anticipate defects based on their previous experience and knowledge of the programmer's style of coding. A well-structured approach to the error guessing technique is to enumerate a list of possible defects and to design a set of tests that attack these defects head on. This systematic approach is called fault attack. These defect and failure lists can be built based on experience, available defect and failure data, and from common knowledge about why software fails. Another technique used by experienced QA's is called exploratory testing.

This type of testing is concurrent test design, test execution, test logging and learning, based on an agreed test charter containing test objectives, and carried out within sprints. This is an approach that is most useful where there are few or inadequate specifications and severe time pressure or to augment or complement other, more formal testing. It can serve as a check on the test

process, to help ensure that the most serious defects are located and logged.

So, moving on, what is agile testing anyway? First, let's remind us what the Agile Manifesto is.

Individuals and interactions	over	processes and tools
Working software	over	documentation
Customer collaboration	over	contract negotiation
Responding to change	over	following a plan

So, using the values from the Manifesto to guide the team, we strive to deliver small chunks of business value in extremely short release cycles (as already stated, usually 4 weeks). It is important to understand that a tester on an agile project will work differently than one working on a traditional project.

The QA's need to understand the values and principles that underpin agile projects, and how QA's are an integral part of a whole-team approach together with developers and business representatives. As you have already seen the members in an agile project communicate with each other early and frequently, which helps with removing defects early and developing a quality product.

The whole team is responsible for quality as a group in agile projects, remember there is no I in the word team, no one is alone in agile. The essence of the whole-team approach lays in the QA's, programmers and the business representatives working together in every step of the development process.

QA's will need to work closely with both programmers and business representatives to ensure that the desired quality levels are achieved.

This includes supporting and collaborating with business representatives to help them create suitable acceptance tests, working with developers to agree on the testing strategy, and deciding on test automation approaches. QA's can thus transfer and extend testing knowledge to other team members and influence the development of the product.

Everyone on an agile team is a tester at some level. Anyone should be able to pick up testing tasks. If that's true, then what is special about a dedicated agile QA? If I define myself as a QA on an agile team, what does that mean? Do agile QA's need different skill sets than testers on traditional teams? What

guides them in their daily activities? Read on to find out these answers.

You should define an agile QA this way: they are a professional tester who embraces change, collaborates well with both technical and businesspeople, and understands the concept of using tests to document requirements and drive development. As already mentioned, agile QA's tend to have good technical skills, know how to collaborate with others to automate tests, and they should also be experienced exploratory QA's. They're willing to learn what customers do so that they can better understand the customers' software requirements.

Who's an agile QA? They are a team member who drives agile testing. There are many agile QA's who started in some other specialisation. A developer can become test-infected and branches out beyond unit testing. An exploratory tester, accustomed to working in an agile manner, is attracted to the idea of an agile team. Finally, professionals in other roles, such as business or functional analysts, might share the same traits and do much of the same work.

Successful projects are a result of a good team that can thrive and produce good quality work. The characteristics that make someone successful as a tester on an agile team are probably the same characteristics that make a highly valued tester on any team.

A good, experienced agile QA doesn't see themselves as a quality police officer, protecting their customers from inadequate, badly written code. They should be ready to gather and share information, to work with the customer or product owner to help them express their requirements adequately so that they can get the features they need, and to provide feedback on project progress to everyone.

Poor specifications are very often a major reason for project failure. Specification problems can result from the customer's lack of insight into their true needs, the absence of a global vision for the system, redundant or contradictory features, and other forms of miscommunication. In agile development, user stories are written to capture requirements from the perspectives of developers, QA's and business representatives. In sequential development, this shared vision of a feature is accomplished through formal

reviews after requirements are written. In agile development, this shared vision is accomplished through frequent informal reviews while the requirements are being written.

The user stories must address both functional and non-functional characteristics. Each story includes acceptance criteria for these characteristics. These criteria should be defined in collaboration between business representatives, developers, and QA's. They provide developers and QA's with an extended vision of the feature that business representatives will validate. An Agile team considers a task finished when a set of acceptance criteria have been satisfied.

This is one of the valuable roles of the agile tester; typically, they will improve the user story by identifying missing details or non-functional requirements. An agile QA can contribute by asking business representatives open-ended questions about the user story, proposing ways to test the user story and confirming the acceptance criteria. Different agile teams vary in terms of how they document user stories. Regardless of the approach taken to document user stories, the finished documentation should be concise, relevant and necessary. Also, you the QA should be taking an active role in their creation.

QA's should also always play an important role in the sprint retrospectives. Agile QA's are part of the team and can bring their unique perspective to the table. Testing occurs in each sprint and vitally contributes to success. All team members, QA's and non-testers, can and should provide input on both testing and non-testing activities. After all, that is the agile way.

Another feature of the agile way is continuous integration. Delivery of each product increment requires working, reliable, and integrated software at the end of every sprint. Continuous integration can address this challenge by merging all changes made to the software and integrating all changed components to a central repository regularly, at least once a day.

In the agile world configuration management, compilation, software build, deployment, and testing are wrapped into a single, automated, repeatable process. As a result, since programmers integrate their work constantly, build constantly and test constantly, then any defects in the code are detected much

more quickly.

The good news for agile QA's is that continuous integration allows them to run automated tests regularly, in some cases as part of the continuous integration process itself and send quick feedback to the team on the quality of the code. These test results are visible to all team members, especially when automated reports are integrated into the process. Automated regression testing can also be continuous throughout the iteration.

Good, automated regression tests cover as much functionality as possible, including user stories delivered in the previous iterations, they can never cover everything though and manual testing will always be required at some level.

Good coverage of the automated regression tests helps support building (and testing) large, complex, integrated systems. When much of the regression testing is automated, the agile QA's are freed to concentrate their manual testing on new features, implemented changes, and confirmation testing of defect fixes.

In addition to automated tests, organisations that use continuous integration typically use associated build tools to implement continuous quality control. In addition to running unit and integration tests, such tools can also run additional static and dynamic tests, measure and profile performance, extract and format documentation from the source code, and facilitate manual quality assurance processes. This continuous application of quality control aims to improve the quality of the product as well as reduce the time taken to deliver it by replacing the traditional practice of applying quality control after completing all development.

Therefore, continuous integration can provide the following benefits to the development team:

- Allows earlier detection and easier root cause analysis of integration problems and conflicting changes.
- Gives the development team regular feedback on whether the code is working, or not.
- Allows the version of the software being tested to stay within one day of the version being developed.

- Reduces regression risk associated with developer code refactoring due to rapid re-testing of the codebase after each small set of changes.
- Provides confidence that each day's development work is based on a solid foundation.
- Makes progress toward the completion of the product increment visible, encouraging developers and QA's.
- Eliminates the schedule risks associated with big-bang integration.
- Provides constant availability of executable software throughout the sprint for testing, demonstration, or education purposes.
- Reduces repetitive manual testing activities.
- Provides quick feedback on decisions made to improve quality and tests.

Another important element of agile is planning. For Agile lifecycles, two kinds of planning occur, release planning and iteration planning.

Release planning looks ahead to the release of a product, often a few months ahead of the start of a project. Release planning defines and re-defines the product backlog and may involve refining larger user stories into a collection of smaller stories.

Release planning provides the basis for the agile test approach and agile test plan spanning all the planned iterations. In release planning, business representatives establish and prioritise the user stories for the overall release, in collaboration with the team. Based on these user stories, project and quality risks are identified and high-level effort estimation is performed.

Agile QA's should always be involved in release planning and especially add value to the following activities:

- Defining testable user stories, including acceptance criteria
- Participating in project and quality risk analyses
- Estimating testing effort associated with the user stories
- Defining the necessary test levels
- Planning the testing for the release

When release planning is complete then the iteration planning for the first iteration begins. Iteration planning looks ahead to the end of a single iteration only and is concerned with the iteration backlog.

In iteration planning, the team selects user stories from the prioritised release backlog. They then elaborate the user stories, performs a risk analysis for the user stories and estimates the work needed for each user story. If a user story is found to be too vague and any attempts to clarify it have failed, the team can refuse to accept it and use the next user story based on priority.

The business representatives must answer the team's questions about each story so the team can understand what they should implement and how to test each story. The number of stories selected is based on established team velocity and the estimated size of the selected user stories. After the contents of the iteration are finalised, the user stories are broken into tasks, which will be carried out by the appropriate team members.

Agile QA's should always be involved in iteration planning and especially add value to the following activities:

- Participating in the detailed risk analysis of user stories
- Determining the testability of the user stories
- Creating acceptance tests for the user stories
- Breaking down user stories into tasks (particularly testing tasks)
- Estimating testing effort for all testing tasks
- Identifying functional and non-functional aspects of the system to be tested
- Supporting and participating in test automation at multiple levels of testing

Release plans will often change as the project proceeds, including changes to individual user stories in the product backlog. These changes may be triggered by internal or external factors. Internal factors include delivery capabilities, velocity, and technical issues. External factors include the discovery of new markets and opportunities, new competitors, or business threats that may change release objectives and/or target dates. In addition, iteration plans may change during iteration. For example, a user story that was considered relatively simple during estimation might prove far more complex and time-consuming than originally expected.

These changes can be very challenging for QA's. Agile QA's must always understand the big picture of the release for test planning purposes, and they must have an adequate test basis and test knowledge in each iteration for test development purposes. The required information must always be available to the tester early, and yet change must always be embraced according to agile principles. This dilemma requires careful decisions about test strategies and test documentation which are constantly being reviewed.

Release and iteration planning should address test planning as well as the planning for development activities and the agile tester should always ensure the point of view is heard. Test-related issues to address include:

- The scope of testing to be done and the extent of testing for those areas in scope, the test goals, and the reasons for these decisions.
- The team members who will carry out the test activities.
- The test environment and test data required for correct testing, when they are needed, and whether any additions or changes to the test environment and/or data will occur before or during the project.
- The timing, sequencing, dependencies, and prerequisites for the functional and non-functional test activities.
- The project and quality risks to be addressed.

One of the main differences between traditional lifecycles and agile lifecycles is the idea of very short iterations (2-4 weeks typically). Each of these iterations should result in working software that delivers features of added value to business stakeholders.

At the very beginning of the project, there is a release planning meeting. This will be followed by a sequence of sprints (iterations). At the beginning of each sprint, there is an iteration planning meeting. Once the sprint scope is established and agreed, the selected user stories are developed, integrated with the system, and then tested. These sprints are highly dynamic, with development, integration, and testing activities taking place through each sprint, and with considerable parallelism and overlap. In agile the testing activities occur throughout the iteration, not as a final activity.

Within the agile team, the QA's, programmers and business stakeholders all have an important role in software testing, as with traditional lifecycles.

Programmers should always perform unit tests as they develop features from the user stories. QA's then test those features as they become available. Business stakeholders also test the stories during implementation. Business stakeholders might use written test cases, but they also might simply experiment with and use the feature to provide fast feedback to the development team.

In some of the agile practices (e.g., Extreme Programming), team pairing is used. Pairing can involve QA's working together in twos to test features. Pairing can also involve a tester working collaboratively with a programmer (yes this has been known to happen without one trying to kill the other) to develop and test a feature. Pairing can be difficult when the test team is distributed across different geographical areas, but well-planned processes and tools can help enable distributed pairing on a global basis.

Agile teams should always strive to progress forward by having working software at the end of every iteration. To determine when the team will have done, working software they need to monitor the progress of all work items in the iteration and release. QA's in agile teams will need to utilise various methods to record test progress and status. These include test automation results, the recording of progress of running test tasks and stories on the task board, and Burndown charts showing the team's overall progress. These can then be communicated to the rest of the team using media such as online wiki dashboards and board emails, as well as verbally during stand-up meetings.

Agile teams can also use tools that automatically generate status reports based on completed test results and task progress.

These, in turn, can update information dashboards and emails. This is a very important method of communication because it also gathers valuable metrics from the testing process, which can be used in process future improvement. User Stories and Burndown charts are discussed in the next chapters.

The Agile Organisation

What approach is best for you

So what parts of agile does your company use (if any). Are these the best options for your company? Is the approach incomplete? Can other options offer a better solution for your company? Let's have a look at what is on offer....

Scrum



At this moment in time scrum is the most popular approach to agile software development in general use. In the scrum ethos, any adjustments to the current project are based on experience and not on theory. Because Scrum is currently the most popular agile approach it is the one I will discuss in most depth, also it is the approach I currently favour.

The scrum methodology offers four deliverables, these are:

- Product backlog: This is the full list of requirements that define the whole product.
- Sprint backlog: This is the list of requirements and associated tasks in a given sprint (remember scrum calls iterations sprints).
- Burndown charts: These are the visual representations of the progress within a sprint and within the project as a whole.
- Shippable functionality: The final usable product that meets the customer's business goals and is considered done.

The five main practices that are key to Scrum are covered elsewhere in this book. However, as a refresher, they are:

Sprint Planning – 8 hours for monthly sprint

Daily scrum – 15 minutes

Sprint Review – 4 hours for monthly sprint

Sprint Retrospective– 3 hours for monthly sprint

The Definition of Scrum

Scrum is a process framework within which teams and organisations can address difficult and complex adaptive problems, while productively and creatively delivering products that are of the highest possible value.

Scrum is:

- Lightweight
- Simple to understand
- Difficult to master

So, scrum has been around since the early 1990s. It is not a process or a technique for building products. Indeed, it is a framework within which your company can employ various processes and techniques. Scrum makes clear the relative efficacy of your product management and development practices so that you can improve.

The Scrum Theory

Scrum is founded on empirical process control theory, otherwise known as empiricism. Empiricism asserts that knowledge comes from experience and making decisions based on what is known. Scrum employs an iterative, incremental approach to optimise predictability and control risk. Three pillars uphold every implementation of empirical process control: transparency, inspection, and adaptation. So, let's have a closer look at these three pillars.

Transparency

Significant aspects of the process must always be visible to those responsible for the outcome. Therefore, transparency requires those aspects to be defined by a common standard, so all observers share a common understanding of what is being seen.

Inspection

Scrum users should frequently inspect Scrum artefacts and the progress toward a Sprint Goal to detect undesirable variances. These inspections should not be so frequent however that inspection gets in the way of the work. The inspections are most beneficial when diligently performed by skilled inspectors at the point of work.

Adaptation

If the person doing the inspection determines that one or more aspects of a process deviate outside acceptable limits and that the resulting product will be unacceptable, the process or the material being processed must be altered and adjusted. An adjustment should be made as soon as is possible to minimise further deviation.

Scrum prescribes four formal events for inspection and adaptation; these have already been mentioned but as a reminder they are:

- Sprint Planning
- Daily Scrum
- Sprint Review
- Sprint Retrospective

XP: Where the customer is put first



XP or Extreme Programming is another popular approach to software development. The main focus of XP is complete customer satisfaction, in XP the customers needs always come first. XP teams achieve high customer satisfaction by developing required features when the customer needs them.

New requests are part of the development team's normal daily routine, and the team must deal with requests whenever they pop up. The team organises itself around any problem that arises and solves it as efficiently as they possibly can. The main XP practices are else in this book.

Lean Programming: Producing JIT

leanthinking



Lean is not as popular as Scrum or XP and has not been discussed so far in this book. It is, however, a valid and productive framework for agile that should always be considered when a company is contemplating a move to agile. Lean has its origins in manufacturing. Way back in the depths of time, well okay the 1940s in Japan, a small little company called Toyota wanted to produce cars for the Japanese market but couldn't afford the massive investment that tooling up for mass production requires. Therefore they studied the way supermarkets worked, noting how consumers only buy what they need, secure with the understanding there will always be a supply of the goods they purchase. They also noted and how the stores restock shelves only as they empty. From this observation, Toyota created a JIT (just in time) process that it could translate to the factory floor.

The result was a significant reduction in the inventory of parts and finished goods and a lower investment in the machines, people, and space required. The JIT process gives workers the ability to make decisions about what is most important to do next.

The workers take responsibility for the results. Toyota's success with JIT processes has helped change mass manufacturing approaches globally and as

a result, they are now one of the largest car producers in the world.

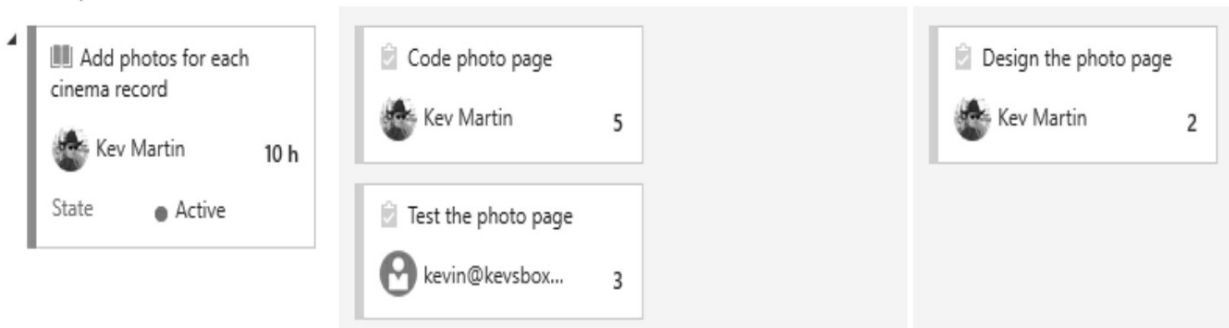
The seven tried and tested principles of lean manufacturing can be applied to software development. These can be used to optimise the whole IT value stream. Lean software development principles are as follows. Eliminate waste, build in quality, create knowledge, defer commitment, deliver quickly, respect people, and optimise the whole.

KanBan

kevsbox Team cinema photos

Backlog Board Capacity

^ Collapse all



The Kanban method is a lean methodology that we have discussed elsewhere in this book. However, as a refresher the two Kanban principles critical to success are:

1. Visualising the workflow: Teams use a Kanban board (whiteboard, corkboard or electronic board) that displays kanbans (indications of where in the process a piece of work or task currently is). The board is organised into columns, each one representing a stage in the process, a work buffer, or queue; and optional rows, indicating the allocation of capacity to classes of service. The board is updated by team members as work proceeds, and blocking issues are identified during daily meetings.
2. Limit work in progress (WIP): Limiting WIP reduces average lead time, improving the quality of the work produced and increasing the overall productivity of your team. Reducing lead time also increases your ability to deliver frequent functionality, which helps build trust with your stakeholders. To limit WIP, understand where your blocking issues are, address them quickly, and reduce queue and buffer sizes wherever you can.

Agile Modelling



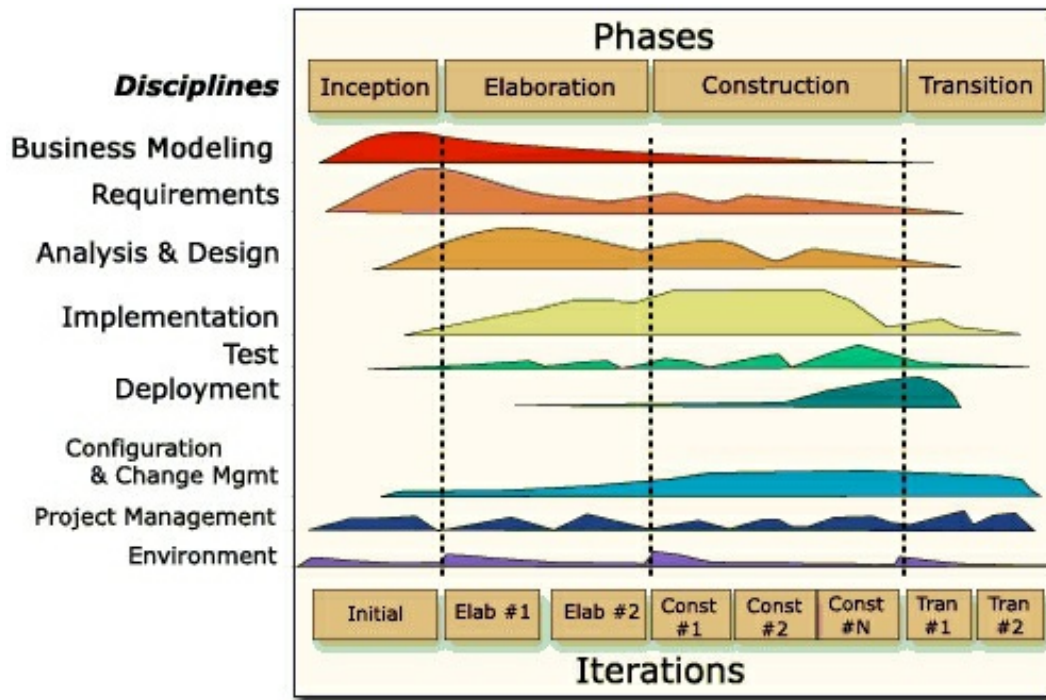
Agile Modelling (AM) is a collection of practices, principles and values that are used for modelling software that can be applied to a software development project in a lightweight and effective manner. AM was purposely designed to be a source of strategies that can be tailored to other base processes. With an Agile Model Driven Development (AMDD) approach, you typically do just enough high-level modelling at the beginning of a new project to understand the potential architecture and scope of the system under design. During construction iterations, you do modelling as part of your iteration planning activities and then take a JIT model storming approach where you model for several minutes as a precursor to several hours of coding. AMDD recommends that teams take a test-driven approach to development although doesn't insist on it.

Agile Modelling practices include the following:

- **Active stakeholder participation:** Stakeholders provide information, make decisions, and are actively involved in the development process.

- Architecture envisioning: This practice involves high-level architectural modelling to identify a viable technical strategy for your solution.
- Document continuously: Write documentation for your deliverables throughout the life cycle in parallel to the creation of the rest of the solution. Some teams choose to write the documentation one iteration behind to focus on capturing stable information.
- Document late: Write deliverable documentation as late as possible to avoid speculative ideas likely to change in favour of stable information.
- Executable specifications: Specify detailed requirements in the form of executable customer tests and your detailed design as executable developer tests.
- Iteration modelling: Iteration modelling helps identify what needs to be built and how.
- Just barely good enough artefacts: A model needs to be enough for the situation at hand and no more.
- Look-ahead modelling: Invest time modelling requirements you intend to implement in upcoming iterations. Requirements near the top of your work item list are complex so explore them before they're popped off the top to reduce overall project risk.
- Model storming: Do JIT modelling to explore the in-depth requirements that sit behind a requirement or to think through a design issue.
- Multiple models: An effective developer has a range of models in his toolkit, enabling him to apply the right model for the situation at hand.
- Prioritised requirements: Implement requirements in priority order, as defined by your stakeholders.
- Requirements envisioning: Invest your time at the start of an agile project to identify the scope of the project and create the initially prioritised stack of requirements.
- Single-source information: Capture info in one place only.
- TDD: Quickly code a new test and update your functional code to make it pass the new test.

Unified Process (UP)



The Unified Process (UP) is probably the least used methodology at this moment in time. However, it is a valid framework that can provide a valuable solution should it suit your company needs. UP uses an iterative and incremental approach within a set life cycle and focuses on the collaborative nature of software development. It can be extended to address a broad variety of project types, including OpenUP, Agile Unified Process (AUP), and Rational Unified Process (RUP).

UP divides the project into iterations focused on delivering incremental value to stakeholders in a predictable manner. The iteration plan defines what should be delivered within the iteration, and the result is ready for iteration review or shipping. UP teams like to self-organise around how to accomplish iteration objectives and commit to delivering the results. They do that by defining and “pulling” fine-grained tasks from a work items list. UP applies an iteration lifecycle that structures how micro-increments are applied to deliver stable, cohesive builds of the system that incrementally progress toward the iteration objectives.

UP structures the project life cycle into four phases, these are

- Inception
- Elaboration
- Construction
- Transition

The project life cycle provides stakeholders and team members with visibility and decision points throughout the project. This enables effective oversight and allows you to make “go or no-go” decisions at appropriate times. A project plan defines the life cycle, and the result is a released application.

Moving to agile and what to avoid

Not everything in agile is easy and straightforward. There are dangers involved with adopting agile blindly and pitfalls that need to be avoided at all costs. Below is a selection of the most common pitfalls that you and your organisation should be aware of, this is of course not a complete list and any move to agile should be carefully planned, discussed and time-framed. Nothing should be committed to until everyone within your organisation is on board and understands what is involved (At least everyone who matters in the agile process, you can probably exclude the cleaning staff).

Do not become agile zombies. Companies can easily fall into the trap that if they attend an agile workshop and mandate to ascertain easy to understand and out-of-the-box (OOTB) process, that they are now an ultra-efficient, modern agile organisation, this is far from true agile. These companies train their teams to blindly follow and enforce the anointed process while not considering which practices may need to change to meet their organisation's unique needs and requirements. Remember agile isn't a prescribed process or set of practices. It is a philosophy and framework that can be supported by adaption and a willingness to learn and no two agile approaches are the same. One single OOTB methodology that fulfils all needs doesn't exist.

For any organisation, an effective and workable agile adoption requires complete executive sponsorship and full support at the highest level. This involvement means more than simply showing up with a big smile at the kick-off meeting to say a few key words of encouragement and then disappearing back to the top floor and the golf course.

Without the executive's continued and proactive support of the overall initiative, the agile adoption is often doomed because agile initiatives require an upfront investment of resources and funding and continued backing during the initial stages. These are areas that executives typically control and if they are not fully behind the change or they do not completely understand what is required then the required funding is likely to fall short of what is essential.

Different agile companies find that they can fulfil the true spirit of the Agile

Manifesto through different approaches. Ironically, most of these approaches focus on one phase or discipline within the delivery lifecycle, oddly enough this goes against the underlying spirit that is lean, which has always advised us to consider the whole. You will find that most approaches focus on the construction phase.

Construction is typically a straightforward area to focus on when your company is taking on its agile transformation, but if companies only change the way they construct software, they should not be calling themselves agile, because they are not. The development teams could be evolving nicely along the agile path, happily delivering new working software every two or four weeks, but if the processes in other areas of the company only allow for deployment every four months or customer stakeholders aren't prepared to meet regularly, then the company is not realizing all the benefits agile can provide and as a result is not agile.

That well known but often ignored adage, "If you fail to plan, plan to fail," is true in the agile world. Therefore, the early forms of agile, typically RAD, had started to get a bad name.

Corners were often cut, and the agile principles were not always fully adopted, as a result, some early agile projects failed just as badly as earlier projects using methodologies such as waterfall. However, for some companies, the lesson was quickly learned, RAD became Agile and a reboot of the framework started.

Good planning is core to the success of any agile adoption no matter how large or small the company is. Companies should always answer these important questions and understand the answers:

- Why do we want to be agile, and what benefits will agile provide?
- How will we achieve and measure agility?
- What cultural, technological or governance barriers exist, and how do we overcome them?

Without a plan that clearly shapes the agile initiative and includes addressing and resolving any known constraints to moving to agile, it is more difficult to

control the initiative, staff it, fund it, manage blockers and maintain vital continued executive sponsorship.

You can quickly short-circuit a planned agile adoption by focusing solely on a single software or system delivery team. While a single team can gain some benefit from agile, this will be limited. To be a truly successful agile company you need to look at the whole company adopting the agile processes. Agile should always be a change in culture for the entire organisation. A good method to help this is to find champions in Operations, lines of business, product management, Marketing, and other functional areas to increase your success chances.

Moving to agile is very exciting and some people within your teams will want to rush in and start before all the planning and preparation is complete. This temptation is understandable but dangerous. If a proper roadmap for training, processes and tooling isn't outlined early in the adoption your company can soon run into issues that will affect confidence in the agile process.

Because an agile adoption isn't just a matter of a new delivery process but is also a major cultural shift, good training for the whole team is imperative and the process will fail without it. Programmers don't like change and many people like working in their little comfort world. As a result, the concept of not only changing the way they develop but adding the concept that now they must work closely with five, six, or ten other people all the time can be very frightening. An agile coach can work with these team members and help them through the initial phases of agile adoption.

Unfortunately, some companies often see agile practice training, like coaching, as an area where they can save money, sending only a few key managers to learn the new process in hope that they will then train the rest of the teams while trying to implement the new approach.

How many times have you seen managers book themselves onto courses which involve four nights in a plush hotel? The course itself will always be better suited to the actual QA's, programmers or project leaders but the managers often see it as a jolly or very nice few days out of the office and a bar bill their company will pay for.

They will intend to return with new skills that they will then pass down but, time pressures and lack of interest with the result in very little being fed back to the teams apart from jovial stories about what happened when they were all drunk.

If challenged by the lack of feedback by higher management then the response is often ‘The course was a load of rubbish and no one else should be sent to it’. This is an age-old problem that still exists to this day.

Therefore, always remember agile involves a change in behaviour and process. It is critical to send all team members to the appropriate training and provide them with ongoing training to reinforce agile values and update team members on processes that may have changed. If the right people are sent on the right courses the valuable skills will be picked up and this will increase the overall skill set of the company.

Many classic mistakes can take an agile project off the rails. Below is just a small sample of what can go wrong.

All you need is scrum (Scrum is all you need)

Some companies believe it is possible to adopt agile without technical practices at all. Hopefully, by now, you will have realised this is not the case. However, starting with scrum is never a bad idea. If you apply Scrum correctly and get everyone behind it, you will inevitably decide to try some technical practices at a retrospective meeting.

What you should never do though is rely on Scrum solely as a process that will solve all the problems you encounter. It can do that, but only if you are open-minded and willing to try other various features such as pair programming.

The Scrum Master knows everything

The Scrum Master is not an all-knowing, all-seeing deity. While it is true, they have some basic knowledge of Scrum and some agile practices, but in many cases, that’s just it. They may have no hands-on experience with software development, project management or testing. As a result, the scrum

master should not act as a project manager and prescribe what to do and what not to do. These decisions should be made at a team level. The only real things the scrum master should care about are the teams' impediments and meta-process. The meta-process is the set of rules and procedures that allow the team to reflect and improve their existing development process.

We Can Live Without Customer Feedback

One of the strengths of agile is regular feedback. Feedback from the actual customer is the most important. If your team build something that is incorrect or does not fit the customer need then you need to know this sooner rather than later. Regular feedback from the customer helps keep the project on course and keeps you off the rocks. With feedback, you can regularly correct the direction should it be required. Remember the customer is a valuable team member and should always be treated accordingly.

Extreme programming recommends having the customer's spokesperson on-site. While this is a great idea, it is rarely practical. However, you can have the customer available remotely all the time to answer questions and communicate about a project. If you can't have feedback in a reasonable amount of time, agile methods simply will not work. You may build a technically perfect product but yield zero customers' satisfaction in the end.

Self-organisation is Easy

Scrum heavily advertises self-organisation. Complexity theory has something to say about it as well. Self-organisation is based on a set of simple rules, non-linearity and interactions between agents (in the case of a scrum between people). You will never see self-organisation working with just a set of rules. Self-organisation in a software development team needs more, it needs leadership and cooperation.

Pure self-organisation assumes that a leader will emerge. That does not happen frequently and in many cases, the team will stagnate and fluctuate around mediocrity without a real leader. The leader sets a vision, motivates and pushes the team in the right direction. The leader empowers confidence, passion and self-reflection. This leads to true self-organisation eventually.

False Goal

(E.g. The customer asked us to be agile)

If you have a customer who insists you use the agile process for their project you should smile and be happy with this gift. Use this chance as a turning point for agile adoption and a valid reason for moving forward.

Unfortunately, many companies still just try to “emulate” agile adoption with a desire to get this contract. They will reluctantly send some people to cheap agile courses. They will also purchase an agile project management tool and apply Scrum superficially.

They do all that without deep goals and culture change within the company. As a result, there will be no real passion or desire to change fully. They do all that with a false goal, money. Almost inevitably there will be the following symptoms:

- The sprints will fail.
- No commitment will mean poor code.
- The scrum master will be ineffective and unmotivated.
- There will be little or no testing in each sprint resulting in bug-ridden code.

The result of “false goal” agile adoption is always a failure and a long-term disappointment with agile software development.

User Stories

7

10

As a

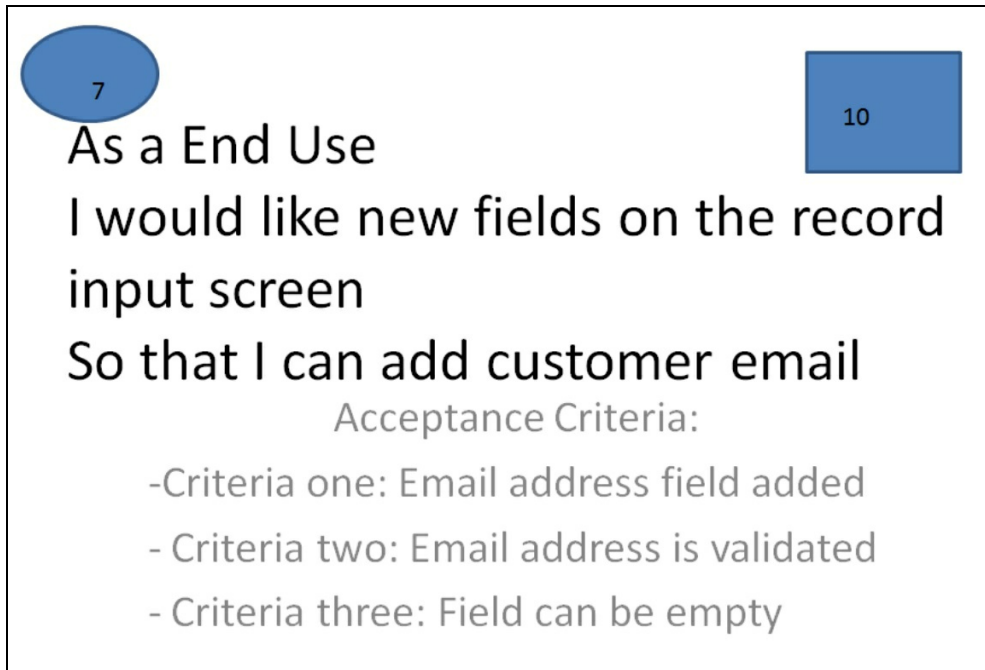
I would like

So that ...

Acceptance Criteria:

- Criteria one
- Criteria two
- Criteria three

When stakeholders realise the need for a new software system, feature set, or application, the agile process will begin with the appointed product owner defining what the new or updated software will do and what services it will provide to the end-users. Instead of following the more traditional process of product managers and business analysts writing lengthy requirements or specifications, agile takes a more efficient lightweight approach of writing down brief descriptions of the pieces and parts that are needed. These become work items and are captured in the form of user stories. A user story is a simple description of a product requirement in terms of what that requirement must accomplish for whom.



7

10

As a End Use
I would like new fields on the record
input screen
So that I can add customer email

Acceptance Criteria:

- Criteria one: Email address field added
- Criteria two: Email address is validated
- Criteria three: Field can be empty

Therefore, user stories are the agile form of requirements specifications and should explain how the system should behave with respect to a single, coherent feature or function. User stories are a short, simple description of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system. Each story should be small enough to be completed in a single sprint. Larger collections of related features, or a collection of sub-features that make up a single complex feature, may be referred to as "epics".

User stories are often written on index cards or sticky notes. These will then be arranged on boards, walls or tables to help facilitate the initial planning and discussion.

As a result, they strongly shift the focus from writing about features to discussing them. The following discussions are more important than whatever text is written.

Epics may include user stories for different development teams. These collections may be developed over a series of sprints. Each epic and its user stories should have associated acceptance criteria. The typical format for a User Story is thus:

Title	a name for the user story
As a	user or persona
I want to	take this action
So that	I get this benefit

The story should also include at least one validation step, these are steps to take to know that the working requirement for the user story is correct. That step is usually worded as follows:

- When I <take this action>, this happens <description of action>

Some typical examples are shown below.

- **Searching for customers**
 - As a user, I want to search for my customers by their status, first and last names so I can see who is still active.
- **Modify my diary**
 - As a user, I want to modify my diary but not the diary of other users.

User Stories are a central part of many agile development methodologies, such as in the XP's planning game. User stories define what must be built in the software project development lifecycle. User stories can be written by any member of the team, but they are always prioritised by the product owner to indicate which have the most value for the system. They will then be broken down into tasks and estimated by the developers.

When user stories are about to be implemented in a pending sprint the developers should have the option to talk to the product owner about it. The short stories may be difficult to interpret, may require some background knowledge or the requirements may have changed since the story was written. Clarity is very important in agile and communication is the key tool.

Every user story must at some point have one or more acceptance tests

attached, allowing the developer to test when the user story is done and allowing the customer to validate it. Without a precise formulation of the requirements, prolonged destructive and costly arguments may arise when the product is to be delivered.

User stories are a great benefit to the agile team because they offer a quick way of handling the product owner requirements without having to create large, cumbersome formalised requirement documents and without performing administrative tasks related to maintaining them.

A good agile project will quickly gather user stories to respond faster and with less overhead to rapidly changing real-world requirements.

XP and other agile methodologies have always favoured face-to-face communication over comprehensive documentation and quick adaptation to change instead of fixation on the problem. User stories can achieve this by:

- They allow the breaking of projects into small increments.
- Being small and simple they need very little maintenance.
- They make it easier to estimate the development effort.
- They help maintain close customer contact.
- They are suitable for projects which have badly written or poorly understood requirements. Iterations of discovery soon drive the refinement process and improve understanding.

Gherkin and Cucumber

Gherkin uses a set of special keywords to give structure and meaning to executable specifications. Most lines in a Gherkin document start with one of the keywords. Comments are only permitted at the start of a new line, anywhere in the feature file. They begin with zero or more spaces, followed by a hash sign (#) and some text. Also, spaces or tabs may be used for indentation.

The recommended indentation level is two spaces. Here is an example:

```
Feature: Test for the existence of Google
# This example will try and load Google
Scenario: Try to load google
    Given the web browser has successfully loaded
    When I type google into the address bar
    Then the google search page will load after I press enter
```

Each of the Given, When, Then steps are matched to a code block which is called a step definition, this we will cover late in this book.

The purpose of the Feature keyword is to provide a high-level description of a test feature file, and to group related scenarios. The first primary keyword in a Gherkin document must always be Feature, followed by a : and a short text that describes the feature. You can add free-form text underneath Feature to add more description and improve readability. For example:

```
Feature: Guess the word
    The word guess game is a turn-based game for two players.
    The Maker makes a word for the Breaker to guess. The game
    is over when the Breaker guesses the Maker's word.
```

Below is a list of keywords available.

- Feature
- Rule
- Steps
- Given
- When
- Then
- And, But, *
- Background

- Scenario outline
- Data tables

The (optional) Rule keyword has been part of Gherkin since v6. The purpose of the Rule keyword is to represent one business rule that should be implemented. It provides additional information for a feature. A Rule is used to group together several scenarios that belong to this business rule. A Rule should contain one or more scenarios that illustrate the particular rule.

Feature: Highlander

Rule: There can be only One

Example: Only One -- More than one alive

*Given there are 3 ninjas
And there are more than one ninja alive
When 2 ninjas meet, they will fight
Then one ninja dies (but not me)
And there is one ninja less alive*

Each step starts with Given, When, Then, And or But. Cucumber will always execute each step in a scenario one at a time, in the sequence you've written them in. When Cucumber tries to execute a step, it looks for a matching step definition to execute. As previously shown the step definition will then try to execute code in one of the projects class files.

Keywords are not considered when looking for a step definition. This means you cannot have a Given, When, Then, And or But step with the same text as another step.

Cucumber considers the following steps duplicates:

Given there is loads of money in my account

Then there is loads of money in my account

At first this might seem like a limitation, but it forces you to come up with a less ambiguous, clearer domain language, for example:

Given my account has a nice balance of £900,000

Then my account should have a healthy balance of £900,000

These examples are less ambiguous and easier to understand.

Given steps are used to describe the initial context of the system under test - the scene of the scenario. It is typically something that happened in the past. When Cucumber executes a Given step, it will configure the system to be in a well-defined state, such as creating and configuring objects or adding data to a test database.

The purpose of Given steps is to put the system in a known state before the user (or external system) starts interacting with the system (in the When

steps). Avoid talking about user interaction in Given's.

It's okay to have several Given steps (use And or But for number 2 and upwards to make it more readable). However try to avoid more than 3 Given steps.

Examples:

Given we are taken to the Feedback home page

Given we set the destination to the homepage

When steps are used to describe an event, or an action. This can be a person interacting with the system, or it can be an event triggered by another system.

It's strongly recommended you only have a single When step per Scenario. If you feel compelled to add more, it's usually a sign that then scenario is too complex and you should split the scenario down into multiple scenarios.

Examples:

When we search for "Plumber" in area "PO1 5QY"

When we select "SpiceTheWorld" and go to "SpiceTheWorld"

Then steps are used to describe an expected outcome, or result. The step definition of a Then step will usually use an assertion to compare the actual outcome (what the system actually does) to the expected outcome (what the step says the system is supposed to do).

An outcome should be on an observable output. That is, something that comes out of the system (report, user interface, message), and not a behaviour deeply buried inside the system (like a record in a database) although the result can be compared to a database query to confirm the actual result matches the expected result.

Examples:

Then we are taken to the "Spice Advice Centre" blog page

Then "Spice" should "be" visible

Then "SpiceTheWorld HQ" should "not be" visible

If you have successive Given's, When's, or Then's, you could write:

Example: Multiple Givens

Given one thing

Given another thing

Given yet another thing

When I open my eyes

Then I should see something

Then I shouldn't see something else

Or you could make the example more fluidly structured by replacing the successive Given's, When's, or Then's with And's and But's:

Example: Multiple Givens

Given one thing

*And another thing
And yet another thing
When I open my eyes
Then I should see something
But I shouldn't see something else*

Gherkin also supports using an asterisk (*) in place of any of the normal step keywords. This can be helpful when you have some steps that are effectively a list of things, so you can express it more like bullet points where otherwise the natural language of And etc might not read so elegantly.

For example:

*Scenario: All done
Given I am out shopping
And I have eggs
And I have milk
And I have butter
When I check my list
Then I don't need anything*

Could be expressed as:

*Scenario: All done
Given I am out shopping
* I have eggs
* I have milk
* I have butter
When I check my list
Then I don't need anything*

Occasionally you'll find yourself repeating the same Given steps in all of the scenarios in a Feature. Since it is repeated in every scenario, this is an indication that those steps are not essential to describe the scenarios; they are incidental details. You can literally move such Given steps to the background, by grouping them under a Background section. A Background is placed before the first Scenario/Example, at the same level of indentation.

For example:

*Background:
Given we are taken to the Feedback home page*

*Scenario: Providing positive feedback
When I provide positive feedback
Then that member is listed as expected*

*Scenario: Providing negative feedback
When I provide negative feedback
Then that member is listed as expected*

*Scenario: Providing average feedback
When I provide average feedback*

Then that member is listed as expected

The Scenario Outline keyword can be used to run the same Scenario multiple times, with different combinations of values. Also, the keyword Scenario Template is a synonym of the keyword Scenario Outline. Copying and pasting scenarios to use different values quickly becomes tedious and repetitive, however Scenario outlines allow us to express these scenarios more concisely through the use of a template with <>-delimited parameters.

Example:

Scenario Template: Test footer options – Trades

*Given we set the destination to the homepage
When we select footer option "<footer>"
Then we should see the url "<url>"*

Examples:

<i>footer</i>	<i>url</i>
<i>1</i>	<i>MembershipOverview</i>
<i>2</i>	<i>members.preview</i>
<i>3</i>	<i>blog/trade</i>

A Scenario Outline must contain an Examples (or Scenarios) section. Its steps are interpreted as a template which is never directly run. Instead, the Scenario Outline is run once for each row in the Examples section beneath it (not counting the first header row). The steps can use <> delimited parameters that reference headers in the examples table. Cucumber will replace these parameters with values from the table before it tries to match the step against a step definition.

Data Tables are handy for passing a list of values to a step definition:

Scenario: find a phone number from a collection

Given I have a phone book:

<i>name</i>	<i>phone</i>
<i>Cheezy</i>	<i>525-5309</i>
<i>Sneezy</i>	<i>123-4567</i>
<i>Wheezy</i>	<i>908-9999</i>
<i>Sleazy</i>	<i>666-6666</i>
<i>Freezy</i>	<i>333-3333</i>

*When I look up the phone number for "Sneezy"
Then I should see the phone number "123-4567"*

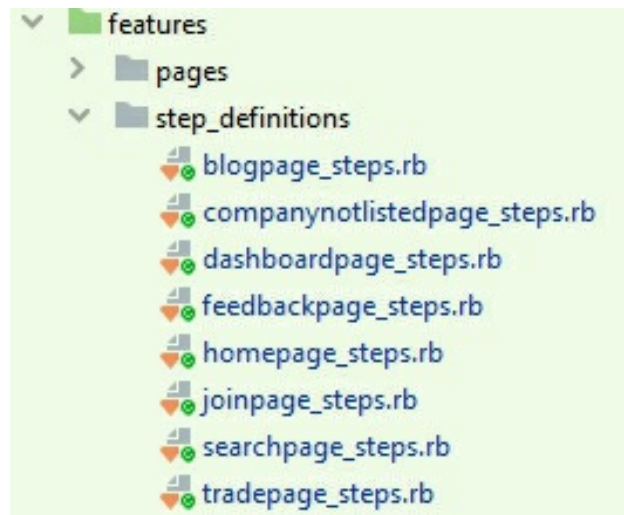
Given /I have a phone book:\$/ do |table|

```
table.hashes.each do |row|
  contact.name row['name']
  contact.phone row['phone']
end
end
```

It should be realised that regardless of the directory structure employed, Cucumber effectively flattens the features directory tree when running tests. This means that anything ending in `.rb` inside the directory in which Cucumber is run is treated as a potential step definition source and therefore all step definitions will be visible to the process.

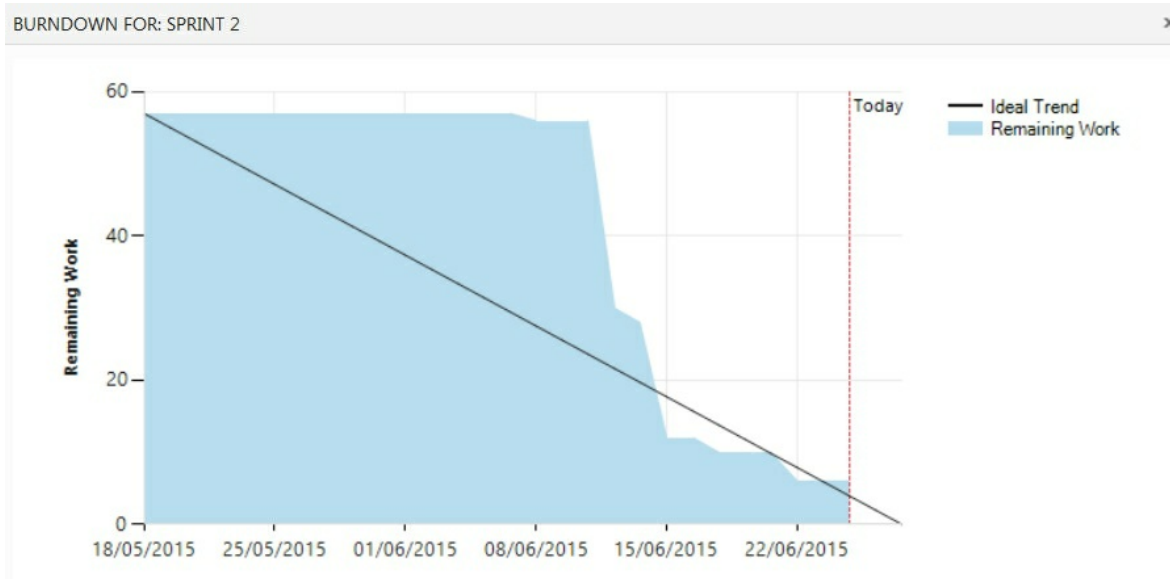
Technically it doesn't matter how you name your step definition files, or which step definitions you put in a file. You could have one giant file containing all your step definitions. However, as the project grows, the file can become messy and hard to maintain. Instead, we recommend creating a separate `*_steps.rb` file for each domain concept.

Examples:



In later chapters you will have the opportunity to put this knowledge to practical test.

Burndown Charts



Burndown charts are graphical artefacts that provide very useful information for the scrum team. There are two types of Burndown charts. The most commonly used type is the Iteration or sprint chart. There is also another type known as the Release Chart which is not as common.

These charts are a graphical representation of the work left to do versus the time remaining. The outstanding work (also known as the backlog) is usually on the vertical axis while the time along on the horizontal. The iteration Burndown chart is often used in agile as a metric for measuring progress over time and as a result, they are a good tool for working out team velocity.

The main concept behind team velocity is to help agile teams estimate how much work they can reasonably complete within a given period. The Burndown charts help achieve this by giving a benchmark of how quickly similar work was previously completed. The following terminology is commonly used in velocity tracking.

- **Unit of work:** The unit is chosen by the team to measure velocity. This can either be a real unit like hours or days or an abstract unit like story points or ideal days, the choice will vary from team to team. Each task in the software development process should then be valued in terms of the

chosen unit.

- **Interval:** The interval is the duration of each individual iteration in the software development process for which the velocity is measured. The length of an interval is determined by the team. Most often, the interval is a week, but it could also be as long as one month, although no longer.

Below is a breakdown of the elements of a Burndown chart

X-Axis	The iteration/project timeline
Y-Axis	The work that needs to be completed for the iteration/project. The time or story point estimates for the work remaining will be represented by this axis.
Project Start Point	This is the farthest point to the left of the chart and occurs at day 0 of the iteration/project.
Project End Point	This is the point that is farthest to the right of the chart and occurs on the predicted last day of the iteration/project.
Ideal Work Remaining Line	This is a straight line that connects the start point to the endpoint. At the start point, the ideal line shows the sum of the estimates for all the tasks (work) that needs to be completed. At the endpoint, the ideal line intercepts the x-axis showing that there is no work left to be completed. This line is a mathematical calculation based on estimates, and the estimates are more likely to be in error than the work. The goal of a burndown chart is to display the progress toward completion and give an estimate of the likelihood of timely completion.
Actual Work Remaining Line	This shows the actual work remaining. At the start point, the actual work remaining is the same as the ideal work remaining but as time progresses; the actual work line fluctuates above and below the ideal line depending on how effective the team is. In general, a new point is added to this line each day of the project. Each day, the sum of the time or story point estimates for work that was recently completed is subtracted from the last point in the line to determine the next point.

To calculate the team velocity, the team first must determine how many

units of work each task is worth and the length of each individual iteration. During development, the team will then keep track of all completed tasks and, at the end of the interval, count the number of units of work completed during the interval. The team then writes down the calculated velocity in a chart or on a graph which is then made available to all interested parties.

It is normal for the first few weeks to provide little data of value but is essential to provide a basis for comparison. Each week after that, the velocity tracking will provide better information as the team provides better estimates and becomes more used to the methodology. Also, velocity should also improve as the team becomes more experienced and more confident in their abilities to achieve within agile.

Test Driven Vs Behaviour Driven

Test-Driven Development (TDD)

When I first encountered TDD, the idea appeared to be simple, however, it soon became obvious this was no one-trick pony. TDD is a very widespread software development technique that involves writing automated test cases before writing the actual functional code. This technique is now popular in agile methodologies as it helps drive development towards delivering a complete, shippable artefact at the end of a sprint. The process itself can be divided into the following steps:

1. First, a programmer, with the assistance of requirement documents, will write a new automated test case(s) for the ticket.
2. The development team will then execute these automated test scripts against the currently developed software. These tests should fail at this point because the new functionality and features are implemented.
3. If the tests pass then they are incorrect, and they will need re-engineering.
4. The programmers will then write the actual code to produce a tested deliverable at the end of the sprint which will pass the tests.
5. If the programmer has written the code well then, the next test run will see that their tests pass.
6. The developer can then refactor their code with comments, enhancements as they wish because if the tests fail again, they will know it is them that has broken it.

Behaviour Driven Development (BDD)

BDD is a software development technique that defines the user behaviour before writing test automation scripts or the functional pieces of code. Used in an agile sprint, this method ensures that a shippable product is generated at the end of a sprint. At its core, BDD expands on TDD by narrowing the notion of behaviour. BDD states that tests should be defined in terms of the behaviour of a unit. The basic outline is shown below.

The behaviour of the user is defined by a product owner/business analyst/QA in simple English as a User Story, for example.

As a [role].

I want [feature].

So that [benefit].

Acceptance Criteria: Goes here

These are then converted to automated scripts to run against functional code, for example.

Given [context]

And [more context]

When [event]

Then [outcome]

And [more outcomes]

The programmer then starts writing the functional code.

And the QA writes the test script which is then executed against the new code.

BDD is easy to read and the behaviour is defined in English. This gives a common ground for all stakeholders involved in the project. As a result, the requirements are understood by all and this reduces the risk of developing code that wouldn't stand up to the accepted behaviour of the user.

TDD vs. BDD

So, let us summarise the advantages of BDD over TDD.

- BDD is in a more readable format by every stakeholder since it is in plain English, unlike TDD test cases written in programming languages such as C# and Java.
- BDD explains the behaviour of an application for the end-user while TDD focuses on how functionality is implemented. Changes on functionality can be accommodated with less impact in BDD as opposed to TDD.
- BDD enables all the stakeholders to all be on the same page with requirements which makes acceptance criteria easier to write and easier to agree on.

Despite this, the choice between TDD and BDD can still be a complicated one. The choice can depend on if there is an appropriate testing framework for your given target language. Also, there is a need to consider the current skill set of the entire team. One advantage with BDD is that Behaviour-specific tests can be executed when a project first starts, while a product is still in development and when a product is completed. At a minimum, BDD requires that the behavioural tests are created before development starts, this will help ensure the team fully understands the acceptance criteria. Before development begins, all the behavioural tests should fail, but as the development of the product progresses, the tests will begin to pass as the new functionality appears. Once all the behavioural tests are considered passing, the product can be considered a deliverable artefact.

Automated Testing with Selenium



Test automation at all levels of testing occurs in many agile teams, and this can mean that QA's spend time creating, executing, monitoring and maintaining automated test cases and the end results. Because of the heavy use of test automation, a higher percentage of manual testing on agile projects tends to be done using experience-based and defect-based techniques such as software attacks, exploratory testing and error guessing.

While programmers should focus on creating unit tests, QA's should focus on creating automated integration, system and system integration tests which can be reused with little or no modification in a later development cycle. This leads to a tendency for agile teams to favour QA's with a strong technical and test automation background.

Such highly skilled QA's are in great demand and can expect better employment terms than lesser-skilled, manual testers.

One goal of the automated tests is to confirm that the build is indeed functioning reliably and is installable. Should any of the automated test scripts fail then the agile team will investigate why. If the reason is that logic has changed then the script will be updated. If, however, the failure has exposed some form of bug then the team should attempt to fix the underlying defect in time for the next code check-in. This should be undertaken whenever and as often as is possible.

While this requires an investment in real-time cost your team should have a viable test reporting system which to provide good visibility into test results. The payback in results makes this effort very worthwhile. This approach helps reduce expensive and inefficient cycles of "build-install-fail-rebuild-reinstall" that can occur in many traditional projects since changes that break the build or cause the software to fail to install are detected quickly.

One of the more popular automation tools on the market today is Selenium. You may well have heard of selenium and possibly even used it. If not, then you are probably thinking; what on earth is this Selenium thing?

Well, selenium is a portable software testing framework for web applications. Selenium also provides a test domain-specific language called Selenese to write tests in several popular programming languages.

These languages include C#, Java, Perl, PHP, Python and Ruby. These tests can then be modified, saved and run against most modern web browsers. Another advantage of selenium is that it is cross platform and it will deploy on the Windows, Linux, and Macintosh platforms. It is open-source software, released under the Apache 2.0 license, and can be downloaded and used without charge, another good bonus.

I currently work with Selenium and C# and this is by far my preferred combination therefore the examples and information to follow is based around that flavour. So, read on you intrepid QA's.

Selenium and ASP.Net

Selenium can be integrated into Visual Studio just as easily as it can be used with other programming languages such as Eclipse. As I previously mentioned, Visual Studio is the author's preferred tool for writing automated tests. If you do not already have it, then at the time of publication there is a free version of Visual Studio available. This is called Visual Studio Community 2019 (VSC2019) and it is available at this URL:

<https://visualstudio.microsoft.com/downloads/>

Community

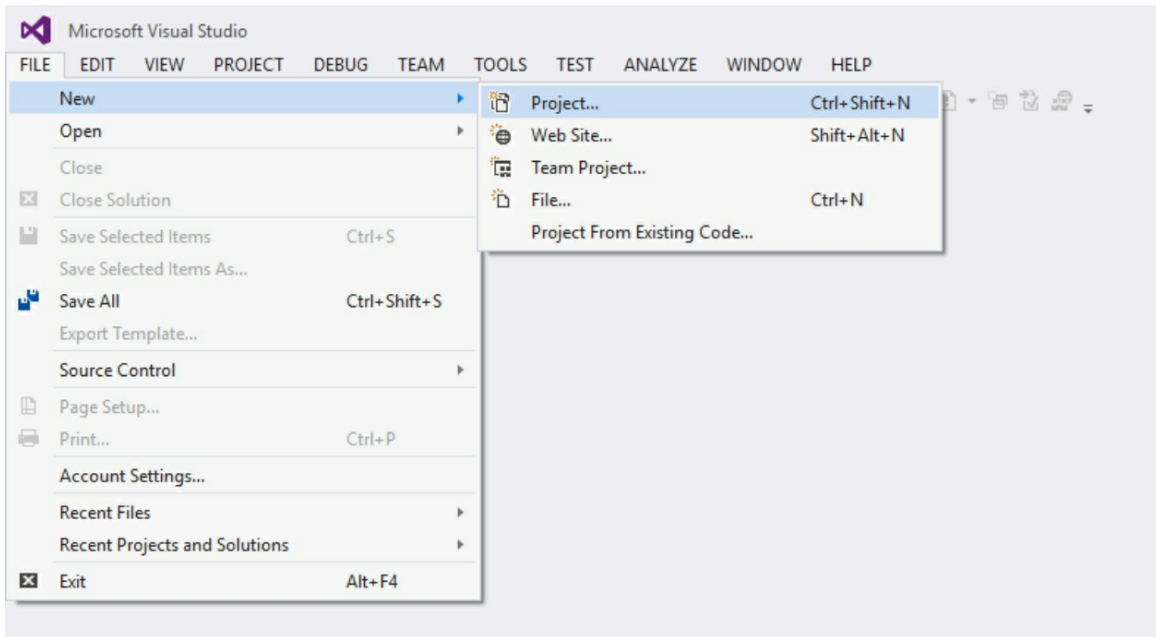
Powerful IDE, free for
students, open-source
contributors, and individuals

Free download ↓

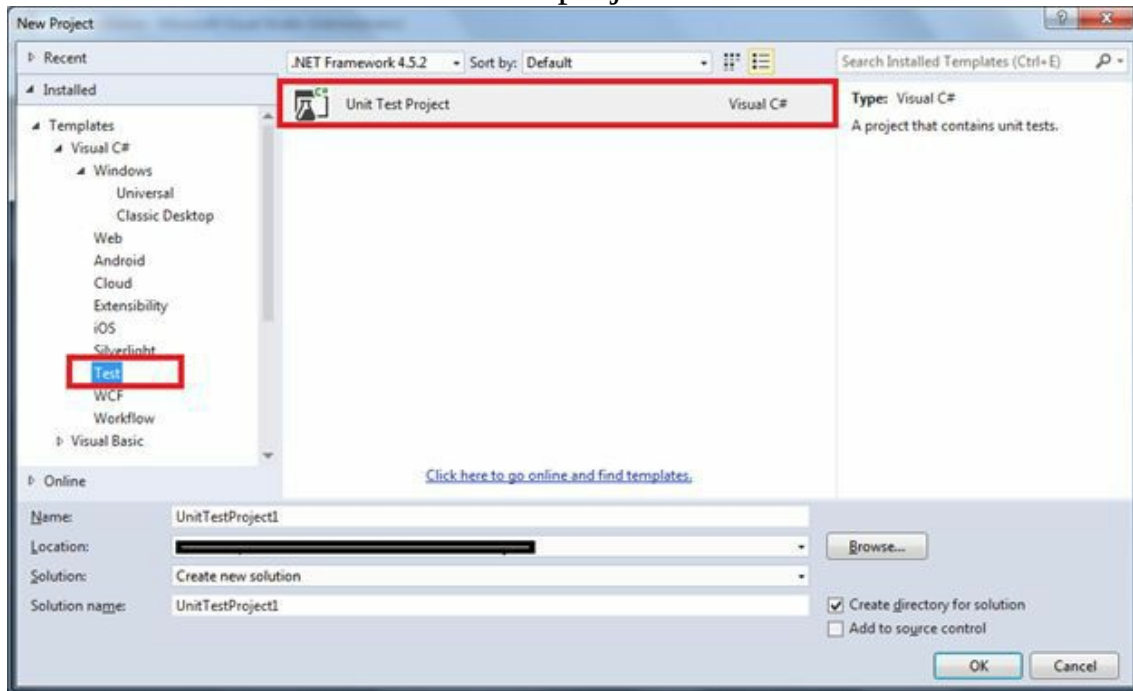
This is a fully functional version of Visual Studio so if you have not yet installed VSC2019 then get your copy now. The initial download is quite fast, but the actual install can take a while, however, it only must be done once.

Once VCS2019 has been installed the next step is to create a new project and then download and add the plug-in.

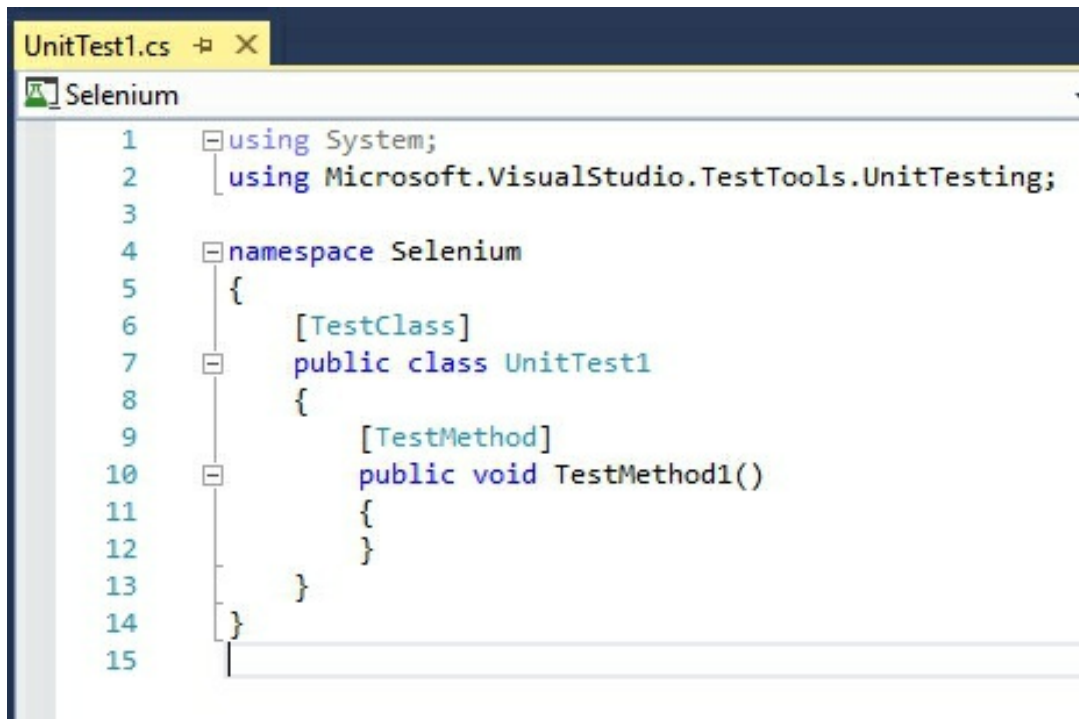
So, first Launch Visual Studio and navigate to File > New > Project.



Then Select Visual C# > Test > Your project name > Click the OK button.

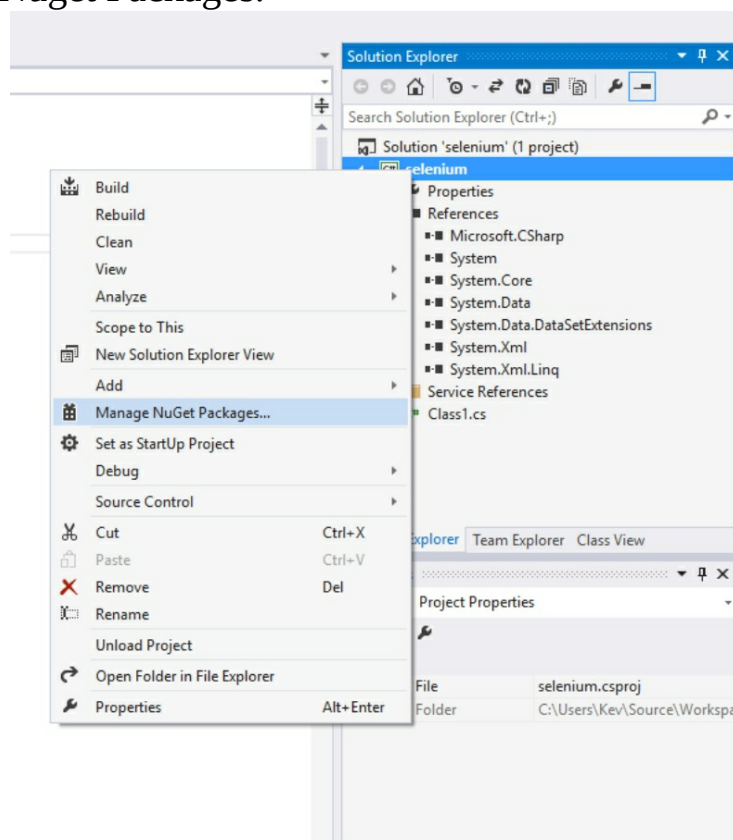


After a few moments the UnitTest1.cs file will be created. This you can rename as you feel appropriate.

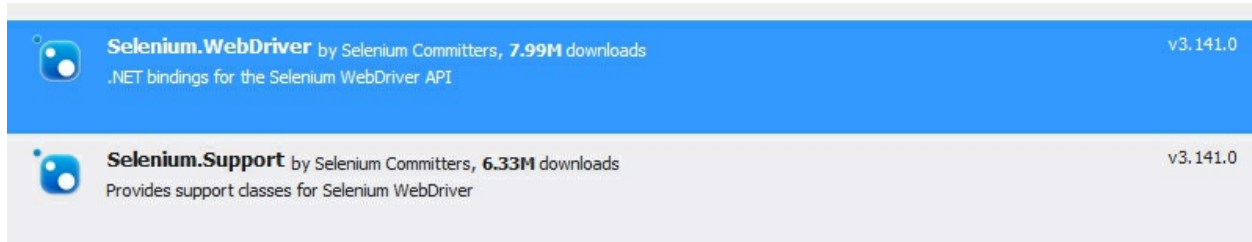


```
1 using System;
2 using Microsoft.VisualStudio.TestTools.UnitTesting;
3
4 namespace Selenium
5 {
6     [TestClass]
7     public class UnitTest1
8     {
9         [TestMethod]
10        public void TestMethod1()
11        {
12        }
13    }
14 }
15
```

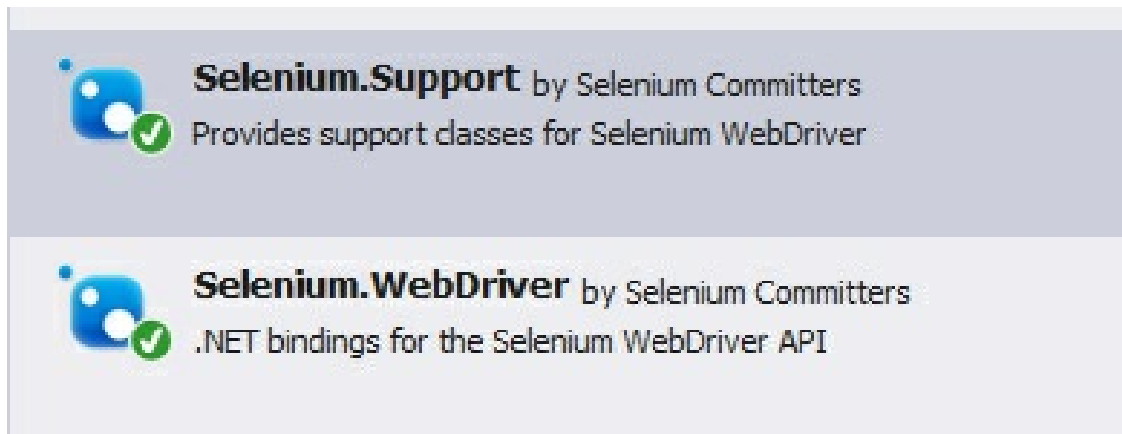
Next right-click on the Project file in the Solution Explorer Window and Select Manage NuGet Packages.



Now conduct a search using Online and the search term Selenium the options for both the Selenium WebDriver and Selenium WebDriver Support Classes should appear. Ensure you install both packages to your project, they are essential.



Install is a simple one-button click for each package. The result will be like below.



You can also achieve the same result with the Package Manager Console. Now using this method simply type in the following two commands.

- Install-Package Selenium.WebDriver
- Install-Package Selenium.Support

Press return after each command and allow the update to complete before moving to the next option.

You should then repeat this process and download the NUnit package by Charlie Poole



NUnit by Charlie Poole, Rob Prouse, **29.2M** downloads

NUnit is a unit-testing framework for all .NET languages with a strong TDD focus.

While both the Internet Explorer and Firefox drivers are included in the Selenium package, you will need to download an additional web driver for Chrome should you wish to test with that web browser as well? At the time of writing the author is currently testing with Chrome and this moment in time it is his preferred browser.

At the time of publication, the download URL was:

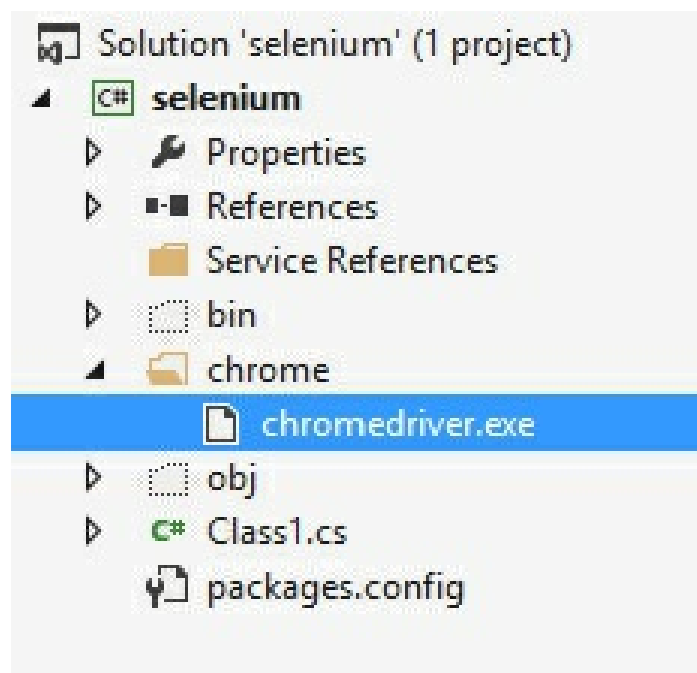
<https://sites.google.com/a/chromium.org/chromedriver/downloads>

ChromeDriver 90.0.4430.24

Supports Chrome version 90

Once again please be aware this may change over time and a search may be required.

When and if you download this driver, you need to copy it to your project, one option is as per the example below, where the driver is copied to a folder called chrome. You could also copy it to the project root folder if preferred.



Now right click on the chromedriver.exe and select Properties. Ensure the Build Action is set to the value of [Content]. Also, check that Copy to Output Directory is set to the [Copy Always] value. This will ensure that chromedriver.exe is always in the folder of the running assembly so it can be used.

So that is it. You now have an empty test class which is configured for basic Selenium testing. Next, we will start to look at some of the more important Selenium commands and see how they can help you create reliable, reusable automated test cases.

The first Selenium test

To get this section going let us start with a very simple example which will do one thing, which is 'load a web page in maximised mode'. That is all it will do but it is a good example of how simple things can be.

So, open the class file UnitTest1.cs (or whatever you may have renamed it as) and type in the code below

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;

namespace selenium
{
    [TestClass]
    public class Class1
    {
        private IWebDriver driver;

        [TestMethod]
        public void TheTest()
        {
            IWebDriver _driver = new ChromeDriver();
            _driver.Manage().Window.Maximize();
            _driver.Navigate().GoToUrl("https://www.kevsbox.com");
            _driver.Quit();
        }
    }
}
```

Now run the code and you will see <https://www.kevsbox.com> appear and then close and the program terminates.

 <https://www.kevsbox.com>

rolled by automated test software.



So, there you have it, your first automated tests have been successfully run. Next, we will look at some of the more complex commands available for your test cases.

The Selenium Command Set

Finding elements

No matter if you are using Java, Ruby or C#, one of the critical requirements of automated testing will always be the ability to identify and locate elements from the web page under test and then being able to perform tests on these elements that will confirm the data returned is both valid and correct. This means that the test tool in use must be able to recognise the web elements quickly, correctly and effectively.

Selenium WebDriver (Watir for Ruby) is a tool that provides one of the most advanced techniques for locating elements on web pages in various popular web browsers. Selenium's very feature-rich API provides reliable multiple locator strategies such as ID, Name, CSS selectors, XPath etc. With Selenium, you are also able to implement custom locator strategies for locating elements. Essentially these locators are the mainstay of your tests. Using the right locator for the situation ensures the tests are faster, more reliable and have a lower maintenance overhead in future releases. In any web development project regardless of what development language is being used, it is always good practice to assign meaningful attributes such as Name, IDs or Class to all the elements that exist on the web page. This makes the application more readable, testable and conforms to existing accessibility standards. There are rare occasions, however, when following these practices is simply not possible. For such scenarios, you will have to use advanced locator strategies such as CSS selector and the XPath function.

While both CSS selector and XPath are popular among most Selenium users, the CSS selector option is always recommended over XPath due to its simplicity, speed, and performance advantages.

Locating elements in Selenium WebDriver is done by using the `findElement()` and `findElements()` methods provided by `WebDriver` and `WebElement` class.

The `findElement()` method returns a `WebElement` object based on a specified search criterion or throws up an exception if it does not find any element

matching the search criteria.

The `findElements()` method returns a list of `WebElements` matching the search criteria. If no elements are found, it returns an empty list.

Find methods take a locator or query object as an instance of `By` class as an argument. Selenium WebDriver provides `By` class to support various locator strategies.

The following table lists various locator strategies currently supported by the Selenium WebDriver:

Strategy	Syntax
By ID	C#: <code>driver.FindElement(By.Id(<elementID>))</code> Java: <code>driver.findElement(By.id(<element ID>))</code> Ruby: <code>browser.div(id: "header")</code>
By name	C#: <code>driver.FindElement(By.Name(<element name>))</code> Java: <code>driver.findElement(By.name(<element name>))</code> Ruby: <code>browser.text_field(name: 'header_name')</code>
By class name	C#: <code>driver.FindElement(By.ClassName(<element class>))</code> Java: <code>driver.findElement(By.className(<element class>))</code> Ruby: <code>browser.text_field(class: 'header_name')</code>
By tag name	C#: <code>driver.FindElement(By.TagName(<htmltagname>))</code> Java: <code>driver.findElement(By.tagName(<htmltagname>))</code> Ruby: <code>browser.element (tag_name: 'div')</code>
By link text	C#: <code>driver.FindElement(By.LinkText(<linktext >))</code> Java: <code>driver.findElement(By.linkText(<linktext>))</code> Ruby: <code>browser.button(text: "Button 2")</code>
By partial link	C#: <code>driver.FindElement(By.PartialLinkText(<linktext >))</code> Java: <code>driver.findElement(By.partialLinkText(<linktext>))</code> Ruby: <code>browser.button(text: /. *Button* ./)</code>
By CSS	C#: <code>driver.FindElement(By.CssSelector(<css selector >))</code> Java: <code>driver.findElement(By.cssSelector(<css selector>))</code>
By XPath expression	C#: <code>driver.FindElement(By. XPath(<xpath query expression>))</code>

Java: `driver.findElement(By.xpath (<xpath query expression>))`

Ruby: `browser.element (xpath: '//h1[@id='header']')`

How to do it...

Locating elements using id, name, or class attributes are the preferred way to find elements in Selenium. So, let's try using these methods to locate elements as described in the following sections, please note all the following examples are C# followed by Java. Ruby examples will be shown in the full example later in the book.

Locate by ID

By far Ids is the preferred method to locate elements on a web page. This is because The W3C standard recommends that developers provide an id attribute for elements that are unique to each element. Having a unique id attribute provides a very explicit and reliable way to locate elements on the page. If for any reason the Ids are not unique, or they are auto-generated this method should not be used.

With this method, the first element with the id attribute value matching the location will be returned. If no element has a matching id attribute, a *NoSuchElementException* will be raised.

Below is an example of how to use this method

```
<form name="userId">Login  
User ID: <input id="userid" type="text" name="login" />  
Password: <input id="password" type="password" name="password" />  
<input type="submit" name="signin" value="SignIn" />  
</form>
```

As you can see both input boxes have a unique id value, these can be used to locate the element. The example code for this is shown below.

C#

```
driver.FindElement(By.Id("id")).Click();
```

Java

```
driver.findElement (By.id ("userid"));
```

Locate by Name

The name attribute is another fast way to locate an element. However, you must also be aware that the name may not be unique. With this method the first matching element will be returned, if no element has a matching name attribute, a NoSuchElementException will be raised.

Below is an example of how to use this method

```
<form name="userId">Login  
User ID: <input id="userid" type="text" name="login" />  
Password: <input id="password" type="password" name="password" />  
<input type="submit" name="signin" value="SignIn" />  
</form>
```

As you can see both input boxes have a name value, these can be used to locate the element. The example code for this is shown below.

C#

```
driver.FindElement(By.Name("name")).Click();
```

Java

```
driver.findElement (By.name ("login"));
```

Locate by XPath

XPath is the language used for locating nodes in XML documents. As HTML can also be an implementation of XML (XHTML), you lucky Selenium users can utilise this powerful language to locate elements in the web applications under test. XPath extends way beyond the simpler methods of locating by id and name attributes, it opens new possibilities such as locating the fifth checkbox on the web page under test.

One of the main reasons for using XPath is when you don't have a suitable id or name attribute for the element you wish to locate (for example they are not unique). You can use the XPath feature to either locate the element in absolute terms or relative to an element that does have an id or name attribute. This is not as preferred as ID or Name and can make your test cases less robust but there are times when no other option is available.

Below is an example of how to use this method

```
<html>
<html> <body>
  <form id="loginForm">
    <input name="userid" type="text" />
    <input name="password" type="password" />
    <input name="cButton" type="submit" value="Login" />
    <input name="cButton" type="button" value="Clear" />
  </form>
</body></html>
```

In this example the form elements can be located like this:

C#

- login_form =
driver.FindElement(By.XPath("/html/body/form[1]"))
- login_form = driver.FindElement(By.XPath("//form[1]"))
- login_form =
driver.FindElement(By.XPath("//form[@id='loginForm']"))

Java

- login_form =
driver.findElement_by_xpath("/html/body/form[1]"))
- login_form = driver.findElement_by_xpath("//form[1]"))

- login_form =
driver.find_element_by_xpath("//form[@id='loginForm']")

The username element can be located like this:

C#

- username =
driver.FindElement(By.XPath("//form[input/@name= userid]"))
- username =
driver.FindElement(By.XPath("//form[@id='loginForm']/input[1]"))
- username = driver.FindElement(By.XPath ("//input[@name=userid]"))

Java

- username =
driver.find_element_by_xpath("//form[input/@name= userid]"))
- username =
driver.find_element_by_xpath("//form[@id='loginForm']/input[1]"))
- username = driver.find_element_by_xpath("//input[@name=userid]"))

Locate by LinkText and PartialLinkText

This is a very useful method to use when you know what the link text will be within an anchor tag. With this method, the first element with the link text value matching the location will be returned. If no element has a matching link text attribute, a *NoSuchElementException* will be raised instead.

Below is an example of how to use this method

```
<html> <body>
  <p>Are you sure you want to delete this record?</p>
  <a href="delete.html">Confirm</a>
  <a href="cancel.html">Cancel</a>
</body></html>
```

The find by link command will be

C#

- `continue_link = driver.FindElement(By.LinkText ("Confirm"))`

Java

- `continue_link = driver.find_element_by_link_text('Confirm')`

Also, the find by partial link will be

C#

- `confirm_link = driver.FindElement(By.PartialLinkText ('Confi'))`

Java

- `confirm_link =
driver.find_element_by_partial_link_text('Confi')`

Locate by Tag Name

Use this when you want to locate an element by tag name. This is a limited method which is used less frequently than the methods already discussed. With this method, the first element with the given tag name will be returned. If no element has a matching tag name, a *NoSuchElementException* will be raised instead.

Below is an example of how to use this method

```
<html> <body>
  <h1>Welcome to Kevsbox.com</h1>
  <p>This is one cool site dude</p>
</body></html>
```

The find by link command will be

C#

- headingLink = driver.FindElement(By.TagName('h1'))

Java

- headingLink = driver.find_element_by_tag_name('h1')

Locate by CSS (Cascading Style Sheets)

Use this method when you want to locate an element by CSS selector syntax. With this method, the first element with the matching CSS selector will be returned. If no element has a matching CSS selector then as per usual a *NoSuchElementException* will be raised.

Below is an example of how to use this method

```
<html> <body>  
  <p class="content">This is kevsbox.com</p>  
</body></html>
```

In this example the “p” element can be located like this:

C#

```
content = driver.FindElement(By.CssSelector ('p. Content'))
```

Java

```
content = driver.find_element_by_css_selector('p. Content')
```

Browser Commands

In this section, we will discuss some of the browser commands which are available to you for use in your automated scripts. This is by no means a complete list, but I am sure you will find these commands very useful in the future. Please note that all these examples are in C#

Navigate to a Web Page

The Navigate command is used to load a new web page in the current browser window, below is an example of how this works.

- `driver.Navigate().GoToUrl("https://www.kevsbox.com");`

Get the title of the page

The Title method fetches the Title of the current page. This method accepts nothing as a parameter and returns a String value, for example.

- `string Title = driver.Title;`

Get the current URL

The URL method returns a string representing the Current URL which is opened in the browser, for example.

- `string CurUrl = driver.Url;`

Get the current page HTML source

The PageSource method returns the Source Code of the page, for example.

- `string pSource = driver.PageSource;`

Close

This method closes only the current window the WebDriver is currently controlling, for example.

- `driver.Close();`

Quit

This method closes all windows opened by the WebDriver, for example.

- `driver.Quit();`

Navigation History

- `driver.Navigate().Back();`
- `driver.Navigate().Refresh();`
- `driver.Navigate().Forward();`

Maximise the browser window

- `driver.Manage().Window.Maximize();`

Cookies

Add a new cookie

- `Cookie cookie = new OpenQA.Selenium.Cookie("key", "value");`
- `driver.Manage().Cookies.AddCookie(cookie);`

Return all cookies

- `var cookies = driver.Manage().Cookies.AllCookies;`

Delete a cookie by name

- `driver.Manage().Cookies.DeleteCookieNamed("CookieName");`

Delete all cookies

- `driver.Manage().Cookies.DeleteAllCookies();`

Other cool commands

Switching Windows or Tabs

- `ReadOnlyCollection<string> windowHandles = driver.WindowHandles;`
- `string firstTab = windowHandles.First();`
- `string lastTab = windowHandles.Last();`
- `driver.SwitchTo().Window(lastTab);`

Example

```
public void SwitchToWindow(Expression<Func<IWebDriver, bool>> predicateExp,
IWebDriver driver)
{
    var predicate = predicateExp.Compile();
    foreach (var handle in driver.WindowHandles)
    {
        driver.SwitchTo().Window(handle);
        if (predicate(driver))
        {
            return;
        }
    }
    throw new ArgumentException(string.Format("Unable to find window with condition: '{0}'", predicateExp.Body));
}
```

Switch to frames

- `_driver.SwitchTo().Frame(1);`
- `_driver.SwitchTo().Frame("frameName");`
- `IWebElement element = _driver.FindElement(By.Id("id"));`
- `_driver.SwitchTo().Frame(element);`

Switch to the default document

- `driver.SwitchTo().DefaultContent();`

Taking a full-screen screenshot

- `Screenshot screenshot = ((ITakesScreenshot)driver).GetScreenshot();`
- `screenshot.SaveAsFile(@"pathToImage", ImageFormat.Png);`

Wait until a page is fully loaded via JavaScript

- `WebDriverWait wait = new WebDriverWait(_driver, TimeSpan.FromSeconds(30));`
- `wait.Until((x) =>`
`{`
 `return((IJavaScriptExecutor)_driver).ExecuteScript(`
 `"return document.readyState").Equals("complete");`
`});`

A complete C# Example

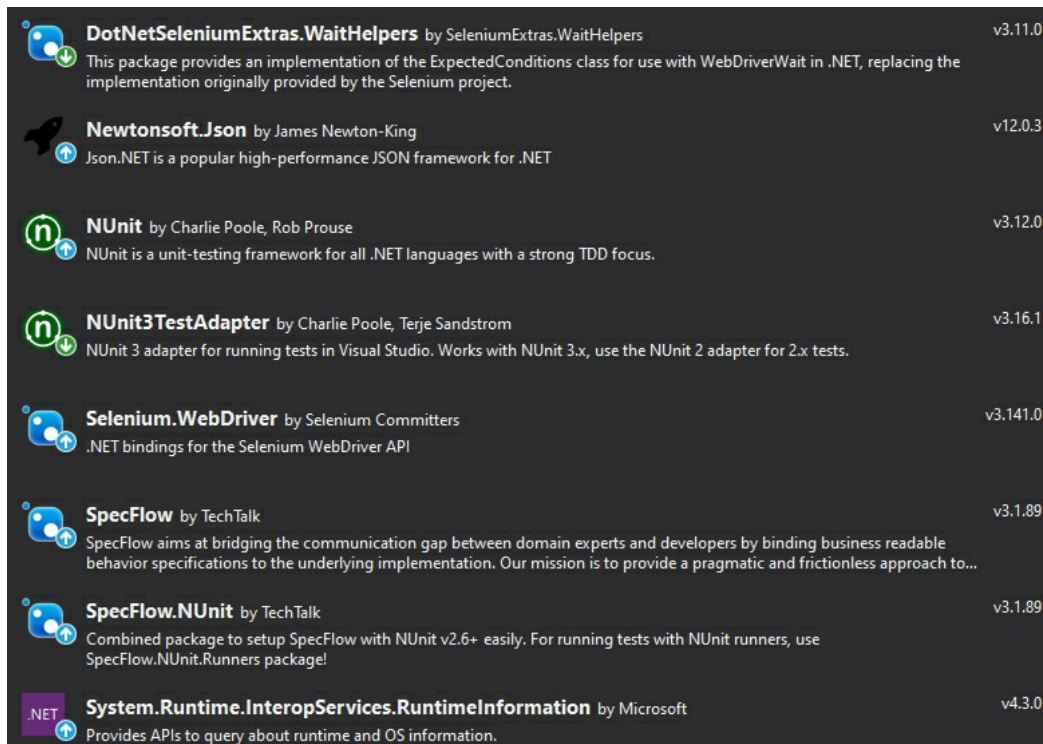
So, let us now look at a complete worked example using Visual Studio, C#, Selenium, NUnit, SpecFlow and MsBuild. The image on the following page shows the installed packages.

For this example, I will assume you have already installed Visual Studio and Selenium. The other packages can be installed if required using NuGet. For example, to install MySql using this command –

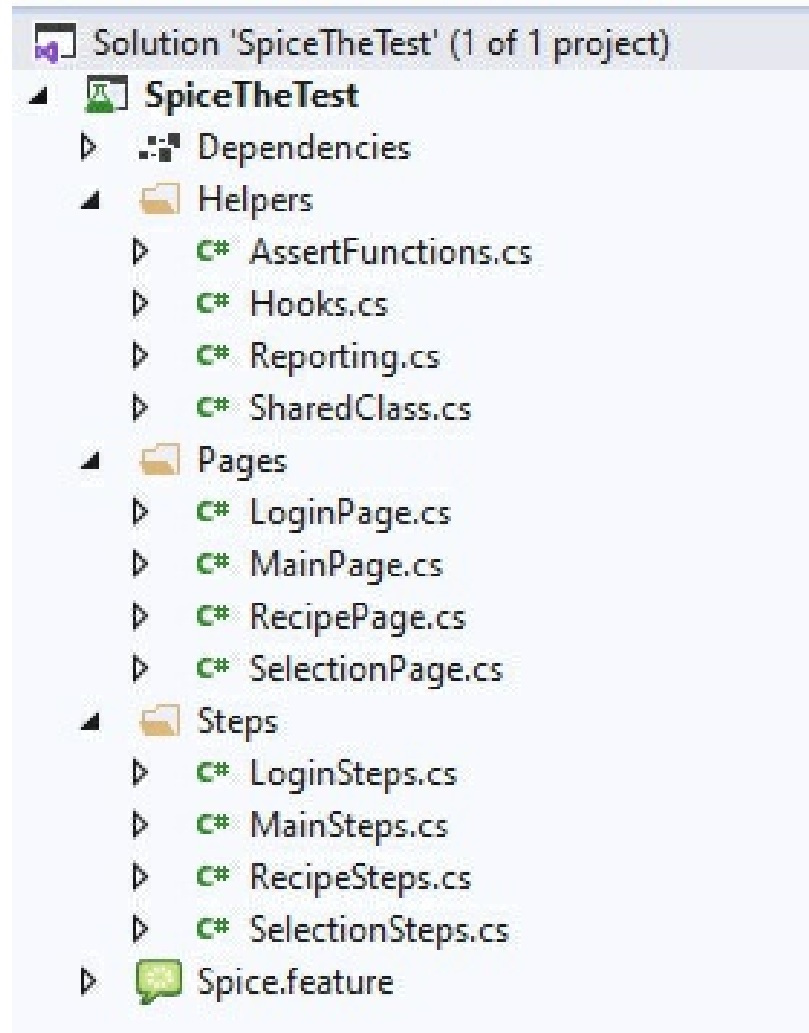
Install-Package MySql.Data -Version 6.9.9

Visual Studio will then install the package.

You should also note that these version numbers will change over time and by the time you get to read this book they will almost certainly be different.



Okay, so when you're ready to continue you will notice that below is the code for this example.



First, we have the project layout. So, the first thing you may notice here is the SpiceTest.feature file, so let us have a look at a sample of the contents of this file.

@SpiceRegression

Feature: SpiceFeature

Scenario Outline: Navigate to the search option and click the Search button

Given we set the destination to homepage

When we search with "<Search>"

Then we validate the destination includes "<Destination>"

Examples:

Search Destination
shark Home/Recipe/89
kevin Home/Search

Scenario Outline: Navigate Recipes using the site menu system

Given we set the destination to homepage
When we select this menu option "<toplevel>" and "<sublevel>"
Then we validate the destination includes "<URL>"

Examples:

	toplevel		sublevel		URL	
	1		1		Home/Selection/CH	
	1		2		Home/Selection/IN	
	1		3		Home/Selection/CK	
	1		4		Home/Selection/CA	
	1		5		Home/Selection/TH	
	1		6		Home/Selection/TR	

Scenario Outline: Navigate Information using the site menu system

Given we set the destination to homepage
When we select this info menu option "<toplevel>" and "<sublevel>"
Then we validate the destination includes "<URL>"

Examples:

	toplevel		sublevel		URL	
	2		1		Info/List/CH	
	2		2		Info/List/SP	
	2		3		Info/List/CT	
	2		4		Info/List/GL	
	2		5		Info/History/CU	
	2		6		Info/History/CT	
	2		7		Info/Scoville	
	2		8		Info/History/GR	

Scenario: View a vegan recipe

Given we set the destination to homepage
When we select this menu option "1" and "6"
And we then select a vegan recipe
Then the vegan symbol is displayed

Scenario: Print the selected recipe

Given we set the destination to homepage
When we select this menu option "1" and "5"
And we then select a thai meat recipe
Then the recipe can be printed

Scenario: Register a new account

Given we set the destination to login page
When we create a new account
Then the logged in message is visible
And I can then logout

Scenario: Login and add a recipe to your folder

Given we login as the test user
When we add a recipe to folder
Then the recipe is visible in the folder page

Scenario: Login and add and remove recipe to your folder

Given we login as the test user
When we add a recipe to folder
Then the recipe is visible in the folder page
And then empty the folder

Scenario: Login and add a recipe to your menu

Given we login as the test user
When we add a recipe to menu
Then the recipe is visible in the menu page

Scenario: Login and add and remove recipe to your menu

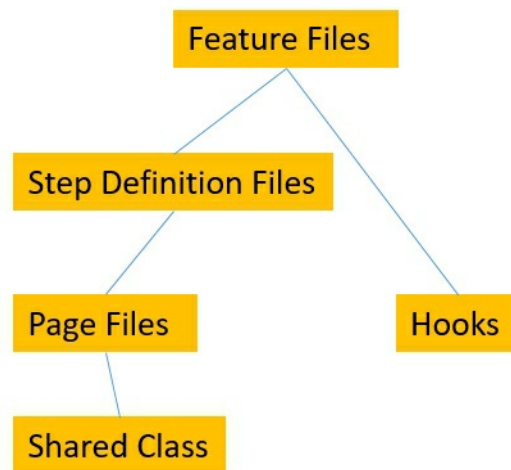
Given we login as the test user
When we add a recipe to menu
Then the recipe is visible in the menu page
And then empty the folder

So, what you see here in the feature file and this functionality is introduced by the SpecFlow package. This enables the User Stories to be converted into Gherkin syntax test scenarios.

A feature file may contain multiple scenarios used to describe the feature's acceptance tests. Scenarios have a name and can consist of multiple scenario steps. Currently, three types of steps define either the preconditions, actions or verification steps that make up the acceptance test (these three types are often referred to as arrange, act and assert).

The different types of steps always begin with either the Given, When or Then keywords respectively (in English feature files). Also, any subsequent steps of the same type can be linked using the [And] and [But] keywords. As you will see the Gherkin language fits in very well with the Agile frameworks and this is a useful tool for translating User stories into easy to read and easy to understand acceptance tests.

You will also notice the *Scenario Outline* syntax; this is an extremely useful feature which cuts down on code duplication and allows similar tests to share function by having variables passed to each loop as detailed in the Examples list.



Let us now look at the Hooks class file. This is called when a new test run is invoked.

```
using OpenQA.Selenium.Chrome;
using System;
using AventStack.ExtentReports;
using TechTalk.SpecFlow;
using OpenQA.Selenium;

namespace SpiceTheTest
{
    [Binding]
    public sealed class Hooks : SharedClass
    {
        private readonly ScenarioContext _scenarioContext;
        private static IJavaScriptExecutor _javascriptExecutor;

        public Hooks(ScenarioContext scenarioContext)
        {
            _scenarioContext = scenarioContext;
        }

        [BeforeScenario]
        public void BeforeScenario()
        {
            Test = Extent.CreateTest(_scenarioContext.ScenarioInfo.Title.ToString());
        }

        [AfterScenario]
        public void AfterScenario()
        {
            var status = _scenarioContext.ScenarioExecutionStatus.ToString();
            if (status != "OK")
                WriteToReport(_scenarioContext.TestError.StackTrace, Status.Fail);
        }
    }
}
```

```

        Driver.Manage().Cookies.DeleteAllCookies();
    }

    [BeforeFeature]
    [Scope(Tag = "SpiceRegression")]
    public static void BeforeFeature(FeatureContext featureContext)
    {
        ResetReportVariables(featureContext.FeatureInfo.Title);
        var options = new ChromeOptions();
        options.AddArguments("disable-browser-side-navigation");
        options.AddArguments("disable-infobars");
        options.AddArgument("ignore-certificate-errors");
        options.AddArgument("ignore-ssl-errors");
        options.AddArgument("disable-popup-blocking");
        options.AddArguments("start-maximized");
        options.AddArguments("no-sandbox");
        Driver = new ChromeDriver(options);
        Driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(30);
        Driver.Manage().Timeouts().AsynchronousJavaScript = TimeSpan.FromSeconds(30);
        Driver.Manage().Timeouts().PageLoad = TimeSpan.FromSeconds(30);
        Driver.Manage().Cookies.DeleteAllCookies();
    }

    [AfterFeature]
    public static void AfterFeature(FeatureContext featureContext)
    {
        try
        {
            Driver.Manage().Cookies.DeleteAllCookies();
        }
        catch (Exception ex)
        {
            Console.WriteLine("Already logged out - ignore : {0}", ex);
        }
        CloseUpReport(featureContext.FeatureInfo.Title);
        Extent.Flush();
        try
        {
            Driver.Close();
            Driver.Quit();
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error: " + ex);
        }
    }
}

```

As you can see there are Before and After feature calls and Before and After Scenario calls. So, these functions are good for initialising and setting up before each Feature and each Scenario as required. You may also notice that

Tags can be assigned so it is possible to have different calls depending on what Tag is being used.

There are also After calls available which are particularly useful for after test operations such as removing test data, report creation and whatever else you require.

The next files to look at are the step definition files. These link the scenarios within the Feature file to actual code that will control the tests. There are 4 files in total and each one relates to a page class.

File 1 – LoginSteps.cs

```
using TechTalk.SpecFlow;
```

```
namespace SpiceTheTest.Steps
{
    [Binding]
    public sealed class LoginSteps : Pages.LoginPage
    {
        [Given(@"we set the destination to login page")]
        public void GivenWeSetTheDestinationToLoginPage()
        {
            Visit();
        }

        [When(@"we create a new account")]
        public void WhenWeCreateANewAccount()
        {
            CreateAndLogin();
        }

        [Given(@"we login as the test user")]
        public void GivenWeLoginAsTheTestUser()
        {
            LoginTestUser();
        }

        [Then(@"the logged in message is visible")]
        public void ThenTheLoggedInMessageIsVisible()
        {
            Assert.IsTrue(IsElementVisible(Login));
        }

        [Then(@"I can then logout")]
        public void ThenICanThenLogout()
        {
            UserLogout();
        }
    }
}
```

```
}
```

File 2 – MainSteps.cs

```
using TechTalk.SpecFlow;

namespace SpiceTheTest
{
    [Binding]
    public sealed class MainSteps : MainPage
    {
        [Given(@"we set the destination to homepage")]
        public void GivenWeSetTheDestinationToHomepage()
        {
            Visit();
        }

        [When(@"we search with ""(.*)""")]
        public void WhenWeSearchWith(string searchString)
        {
            DoASearch(searchString);
        }

        [Then(@"we validate the destination includes ""(.*)""")]
        public void ThenWeValidateTheDestinationIncludes(string url)
        {
            Assert.IsTrue(Driver.Url.Contains(url));
        }

        [When(@"we select this menu option ""(.*)"" and ""(.*)""")]
        public void WhenWeSelectThisMenuOptionAnd(int menu1, int menu2)
        {
            SelectMenuItem(menu1, menu2);
        }

        [When(@"we select this info menu option ""(.*)"" and ""(.*)""")]
        public void WhenWeSelectThisInfoMenuOptionAnd(int menu1, int menu2)
        {
            SelectMenuItem(menu1, menu2);
        }
    }
}
```

File 3 – RecipeSteps.cs

```
using TechTalk.SpecFlow;

namespace SpiceTheTest
{
```

```

[Binding]
public sealed class RecipeSteps : RecipePage
{
    [Then(@"the vegan symbol is displayed")]
    public void ThenTheVeganSymbolIsDisplayed()
    {
        CheckVeganSymbol();
    }

    [Then(@"the recipe can be printed")]
    public void ThenTheRecipeCanBePrinted()
    {
        PrintRecipe();
    }

    [When(@"we add a recipe to folder")]
    public void WhenWeAddARecipeToFolder()
    {
        SelectARecipe();
        AddToFolder();
    }

    [When(@"we add a recipe to menu")]
    public void WhenWeAddARecipeToMenu()
    {
        SelectARecipe();
        AddToMenu();
    }
}

```

File 4 – SelectionSteps.cs

```

using TechTalk.SpecFlow;

namespace SpiceTheTest
{
    [Binding]
    public sealed class SelectionSteps : SelectionPage
    {
        [When(@"we then select a vegan recipe")]
        public void WhenWeThenSelectAVeganRecipe()
        {
            SelectVegan();
        }

        [When(@"we then select a thai meat recipe")]
        public void WhenWeThenSelectAThaiMeatRecipe()
        {
            SelectThaiMeat();
        }

        [Then(@"the recipe is visible in the folder page")]
    }
}

```



```

    public void ThenTheRecipeIsVisibleInTheFolderPage()
    {
        Assert.IsTrue(FolderContainsRecipe(0));
    }

    [Then(@"the recipe is visible in the menu page")]
    public void ThenTheRecipeIsVisibleInTheMenuPage()
    {
        Assert.IsTrue(FolderContainsRecipe(1));
    }

    [Then(@"then empty the folder")]
    public void ThenThenEmptyTheFolder()
    {
        Assert.IsTrue(FolderCountIsZero()==0);
    }
}

```

So here you can see where the scenarios in the Feature files are linked to actual functions in the pages through the definition files.

For example, in the Feature File we have [Given we set the destination to homepage]

This is linked to the following code in MainSteps.cs which inherits class MainPage

```

[Given(@"we set the destination to homepage")]
public void GivenWeSetTheDestinationToHomepage()
{
    Visit();
}

```

From here the function Visit is executed. The function is found in the inherited class file [MainPage] which was defined here

```

public sealed class MainSteps : MainPage

```

The actual function is listed below

```

protected static string PageUrl = "https://www.spicetheworld.com/";
protected void Visit()
{
    Driver.Navigate().GoToUrl(PageUrl);
}

```

This function will attempt to load a URL which is defined in PageUrl. In this instance PageUrl is always <https://www.spicetheworld.com/> and URL will be the page to be displayed at this location.

Many functions will test to see if a condition is true or equal to an expected result. This is how a test is defined. If the expected result has been attained

then the test can be termed a pass, if not then it is a failure. For example:

```
protected void CheckSearchResult(string retPath)
{
    try
    {
        Assert.AreEqual(_driver.Url.ToString(), BaseUrl + retPath);
        TakeScreenshot(_driver);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error: {0}", ex);
        Assert.Fail("Error: {0}", ex);
    }
}
```

In this example, the actual URL of the browser is tested to see it matches the URL in the variables BaseUrl and retPath. If there is a match then the test will pass, if there is not then the test will fail.

So many functions will test for a condition. In every example, if the test runs correctly a true value will be returned hence the Assert.IsTrue. Any variation of this is caught in the catch and the error is logged in the output report.

Assert.AreEqual is only one possible option here, there are many others and below is just a small selection.

- Assert.IsFalse
- Assert.IsNull
- Assert.Equals

Each of the Step Definition files listed above inherits a page class. As expected there are also 4 of those, so let us now look at these files.

File 1 – LoginPage.cs

```
public class LoginPage : SpiceClass
{
    protected static string PageUrl = "https://spicetheworld.com/login.aspx";

    protected By UserString = By.Name("UserString");
    protected By PassString = By.Name("PassString");
    protected By Login = By.CssSelector("[value = 'Login']");
    protected By LoggedIn = By.Name("LoggedIn");
    protected By Resources = By.Id("resources");
    protected By Logout = By.Id("logout");

    protected void Visit()
    {
```

```

        WriteToReport("Load page " + PageUrl);
        Driver.Navigate().GoToUrl(PageUrl);
    }

    protected void LoginTestUser()
    {
        Visit();
        WriteToReport(GetTheCurrentMethod());
        FillTextBox(UserString, "xxxxxx@kevsbox.com");
        FillTextBox(PassString, "xxxxxxxxxx");
        ClickElement(Login);
    }

    protected void CreateAndLogin()
    {
        WriteToReport(GetTheCurrentMethod());
        var newUser = DateTime.Now.ToString("ddMMyyyyhhmmss");
        FillTextBox(UserString, newUser + "@kevsbox.com");
        FillTextBox(PassString, newUser);
        ClickElement(Login);
    }

    protected void UserLogout()
    {
        WriteToReport(GetTheCurrentMethod());
        GetMenuHover(Resources);
        if(IsElementVisible(Login))
            ClickElement(Login);
        Assert.IsTrue(IsElementVisible(Login));
    }
}

```

File 2 – MainPage.cs

```

public class MainPage : SpiceClass
{
    protected static string PageUrl = "https://www.spicetheworld.com/";

    protected By SearchString = By.Name("SearchString");
    protected By SearchBtn = By.CssSelector("[value='Search']");
    protected By SelectIt = By.Name("selectit");
    protected By Recipes = By.Id("recipes");
    protected By Information = By.Id("information");
    protected By Resources = By.Id("resources");
    protected By Scoville = By.Id("scoville");
    protected By Grow = By.Id("grow");
    protected By HistoryChilli = By.Id("historychilli");
    protected By HistoryCurry = By.Id("historycurry");
    protected By Glossary = By.Id("glossary");
    protected By CurryTypes = By.Id("currytypes");
}

```

```

protected By Spice = By.Id("spice");
protected By Chilli = By.Id("chilli");
protected By Chillirec = By.Id("chillirec");
protected By Curryrec = By.Id("curryrec");
protected By Chinrec = By.Id("chinrec");
protected By Caribrec = By.Id("caribrec");
protected By Thairec = By.Id("thairec");
protected By Ttherrec = By.Id("otherrec");

protected void Visit()
{
    WriteToReport("Load page " + PageUrl);
    Driver.Navigate().GoToUrl(PageUrl);
}

protected void DoASearch(string searchString)
{
    WriteToReport(GetTheCurrentMethod());
    FillTextBox(SearchString, searchString);
    ClickElement(SearchBtn);
    if (IsElementVisible(SelectIt, 10))
        ClickElement(SelectIt);
}

protected void SelectMenuItem(int menu1, int menu2)
{
    WriteToReport(GetTheCurrentMethod());
    SelectMenu1(menu1);
    if (menu1==1)
        SelectMenu2(menu2);
    if (menu1==2)
        SelectInfoMenu(menu2);
}

private void SelectMenu1(int menu)
{
    WriteToReport(GetTheCurrentMethod());
    if (menu == 1) GetMenuHover(Recipes);
    else if (menu == 2) GetMenuHover(Information);
    else GetMenuHover(Resources);
}

private void SelectMenu2(int menu)
{
    WriteToReport(GetTheCurrentMethod());
    if (menu == 1) ClickElement(Chillirec);
    else if (menu == 2) ClickElement(Curryrec);
    else if (menu == 3) ClickElement(Chinrec);
    else if (menu == 4) ClickElement(Caribrec);
    else if (menu == 5) ClickElement(Thairec);
    else if (menu == 6) ClickElement(Ttherrec);
}

```

```

    }

    private void SelectInfoMenu(int menu)
    {
        WriteToReport(GetTheCurrentMethod());

        if (menu == 1) ClickElement(Chilli);
        else if (menu == 2) ClickElement(Spice);
        else if (menu == 3) ClickElement(CurryTypes);
        else if (menu == 4) ClickElement(Glossary);
        else if (menu == 5) ClickElement(HistoryCurry);
        else if (menu == 6) ClickElement(HistoryChilli);
        else if (menu == 7) ClickElement(Scoville);
        else if (menu == 8) ClickElement(Grow);
    }
}

```

File 3 – RecipePage.cs

```

public class RecipePage : SpiceClass
{
    protected static string PageUrl = "https://www.spicetheworld.com/Home/Recipe/";

    protected By VeganSafe = By.CssSelector("[title='Vegan safe']");
    protected By PrintLink = By.Id("printLink");
    protected By BodyPop = By.Id("bodyPop");
    protected By Name = By.Id("name");
    protected By WalletLink = By.Id("walletLink");
    protected By MenuLink = By.Id("menuLink");

    protected void SelectARecipe()
    {
        WriteToReport(GetTheCurrentMethod());
        Random rnd = new Random();
        Driver.Navigate().GoToUrl(PageUrl + rnd.Next(1, 100));
        RecipeName = Driver.FindElement(Name).Text;
    }

    protected void AddToMenu()
    {
        WriteToReport(GetTheCurrentMethod());
        if (IsElementVisible(MenuLink))
            ClickElement(MenuLink);
    }

    protected void AddToFolder()
    {
        WriteToReport(GetTheCurrentMethod());
        if (IsElementVisible(WalletLink))
            ClickElement(WalletLink);
    }
}

```

```

    protected void CheckVeganSymbol()
    {
        WriteToReport(GetTheCurrentMethod());
        Assert.IsTrue(IsElementVisible(VeganSafe));
    }

    protected void PrintRecipe()
    {
        WriteToReport(GetTheCurrentMethod());
        if (IsElementVisible(PrintLink))
            ClickElement(PrintLink);
        IList<string> tabs = new List<string>(Driver.WindowHandles);
        Driver.SwitchTo().Window(tabs[tabs.Count-1]);
        Assert.IsTrue(IsElementVisible(BodyPop));
    }
}

```

File 4 – SelectionPage.cs

```

public class SelectionPage : SpiceClass
{
    protected static string PageUrl = "https://www.spicetheworld.com/Home/Selection";

    protected By SelectIt = By.Name("selectit");
    protected By Deleteit = By.Name("deleteit");

    protected void Visit()
    {
        WriteToReport("Load page " + PageUrl);
        Driver.Navigate().GoToUrl(PageUrl);
    }

    protected int FolderCountIsZero()
    {
        WriteToReport(GetTheCurrentMethod());
        try
        {
            IList<IWebElement> recipeLinks = Driver.FindElements(Deleteit);
            do
            {
                foreach (var recipeLink in recipeLinks)
                {
                    recipeLink.Click();
                    break;
                }
                if (recipeLinks.Count != 1)
                    recipeLinks = Driver.FindElements(Deleteit);
                else
                    break;
            }
            while (recipeLinks.Count != 0);
        }
    }
}

```

```

        return 0;
    }
    catch
    {
        return -1;
    }
}

protected bool FolderContainsRecipe(int dest)
{
    WriteToReport(GetTheCurrentMethod());
    if (dest==0)
        Driver.Navigate().GoToUrl(PageUrl + "/WL");
    else
        Driver.Navigate().GoToUrl(PageUrl + "/RP");
    IList<IWebElement> recipeLinks = Driver.FindElements(SelectIt);
    foreach (var recipeLink in recipeLinks)
    {
        if (recipeLink.Text.ToLower().Equals(RecipeName.ToLower()))
        {
            return true;
        }
    }
    return false;
}

protected void SelectVegan()
{
    WriteToReport(GetTheCurrentMethod());
    IList<IWebElement> recipeLinks = Driver.FindElements(SelectIt);
    foreach (var recipeLink in recipeLinks)
    {
        if(recipeLink.Text.ToLower().Contains("vegan"))
        {
            recipeLink.Click();
            break;
        }
    }
    //now select the first recipe
    recipeLinks = Driver.FindElements(SelectIt);
    foreach (var recipeLink in recipeLinks)
    {
        recipeLink.Click();
        break;
    }
}

protected void SelectThaiMeat()
{
    WriteToReport(GetTheCurrentMethod());
    IList<IWebElement> recipeLinks = Driver.FindElements(SelectIt);
    foreach (var recipeLink in recipeLinks)

```

```

    {
        if (recipeLink.Text.ToLower().Contains("meats"))
        {
            recipeLink.Click();
            break;
        }
    }
    //grab first recipe
    recipeLinks = Driver.FindElements(SelectIt);
    recipeLinks[0].Click();
}
}

```

You probably noticed that the page classes inherit the shared class SharedClass so for your reference the full SpiceClass.cs is shown below.

```

public class SpiceClass
{
    public static string RecipeName = "";

    protected void FillTextBox(By by, string theText)
    {
        try
        {
            Driver.FindElement(by).Clear();
            Driver.FindElement(by).SendKeys(theText);
        }
        catch (Exception err)
        {
            WriteToReport("Error: " + err.Message);
        }
    }

    /*
    * locates the element by with the webdriver driver
    * scrolls to the element
    * then clicks the element
    * if this method does not work then use FireEvent
    */
    protected void ClickElement(By by)
    {
        try
        {
            var link = Driver.FindElement(by);
            var js = $"window.scroll(0, {link.Location.Y});";
            ((IJavaScriptExecutor)Driver).ExecuteScript(js);
            Driver.FindElement(by).Click();
        }
        catch (Exception err)
        {
        }
    }
}

```



```

        WriteToReport("Error: " + err.Message);
    }
}

protected void GetMenuHover(By hoverLink)
{
    WriteToReport(GetTheCurrentMethod() + " " + hoverLink);
    try
    {
        IList<IWebElement> menuLinks = Driver.FindElements(hoverLink);
        foreach (var result in menuLinks)
        {
            var action = new Actions(Driver);
            action.MoveToElement(result).Perform();
            break;
        }
    }
    catch (Exception err)
    {
        WriteToReport("Error: " + err.Message);
    }
}

protected bool IsElementVisible(By by, int span = 0)
{
    bool isFound;
    try
    {
        Driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(span);
        isFound = IsThisElementVisible(Driver.FindElement(by));
        Driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(60);
    }
    #pragma warning disable
    catch (Exception ex)
    #pragma warning restore
    {
        isFound = false;
    }
    return isFound;
}

protected bool IsThisElementVisible(IWebElement element)
{
    return element.Displayed && element.Enabled;
}

/**
 * This method will return the current working method
 * that is being called by selenium
 */
[MethodImpl(MethodImplOptions.NoInlining)]

```

```

    public string GetTheCurrentMethod()
    {
        try
        {
            var st = new StackTrace();
            var gf = st.GetFrame(1);
            return "In procedure " + gf.GetMethod().Name;
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error: {0}", ex);
            return ex.Message;
        }
    }
}

```

So the SharedClass inherits from the assertFunctions class so here that is also.

```

public class AssertFunctions : Reporting
{
    /// <summary>
    /// Return true if passed value is not 0
    /// Will accept any numeric type
    /// </summary>
    /// <typeparam name="T"></typeparam>
    /// <param name="by"></param>
    /// <param name="legend"></param>
    public void AssertIsZero<T>(T by, string legend, bool IsZero)
    {
        try
        {
            WriteToReport(Status.Pass, "Pass: condition is true for " + legend);
            if (IsZero) Assert.That(by.Equals(0));
            else Assert.That(!by.Equals(0));
        }
        catch (Exception ex)
        {
            TakeScreenshot();
            WriteToReport(Status.Fail, "Fail error: " + ex + " for " + legend);
            Assert.That(false);
        }
    }

    /// <summary>
    /// AssertTrue
    /// Assert is as value of result
    /// </summary>
    /// <param name="result"></param>
    public void AssertTrue(bool result, string TheMessage)
    {

```

```

        WriteToReport(result ? Status.Pass : Status.Fail,
            result ? "Pass: Assert is true" : "Fail: Assert is false " + TheMessage);
        Assert.That(result);
    }

```

```

    /// <summary>
    /// AssertUrlContains
    /// checks that the URL contains the compare text
    /// true if found
    /// false if not
    /// </summary>
    /// <param name="comparing"></param>
    /// <param name="driver"></param>
    public void AssertUrlContains(string comparing, IWebDriver driver)
    {
        var currentUrl = driver.Url;
        try
        {
            WriteToReport(Status.Pass, "Pass: URL is correct");
            Assert.That(currentUrl.ToLower().Contains(comparing.ToLower()));
        }
        catch (Exception ex)
        {
            TakeScreenshot();
            WriteToReport(Status.Fail, "Fail error: " + ex);
            Assert.That(false);
        }
    }

```

```

    /// <summary>
    /// AssertTitleContains
    /// checks that the Title contains the compare text
    /// true if found
    /// false if not
    /// </summary>
    /// <param name="comparing"></param>
    /// <param name="driver"></param>
    public void AssertTitleContains(string comparing, IWebDriver driver)
    {
        var currentTitle = driver.Title;
        try
        {
            WriteToReport(Status.Pass, "Pass: Title is correct");
            Assert.That(currentTitle.ToLower().Contains(comparing.ToLower()));
        }
        catch (Exception ex)
        {
            TakeScreenshot();
            WriteToReport(Status.Fail, "Fail error: " + ex);
            Assert.That(false);
        }
    }

```

```

    }
}

/// <summary>
/// AssertContains
/// checks that the Container string contains the Matcher string
/// true if found
/// false if not
/// </summary>
/// <param name="container"></param>
/// <param name="matcher"></param>
public void AssertContains(string container, string matcher)
{
    WriteToReport("Assert " + container + " contains " + matcher);
    try
    {
        WriteToReport(Status.Pass, "Pass: Element is correct " + container + " - " +
matcher);
        Assert.That(container.Contains(matcher));
    }
    catch (Exception ex)
    {
        TakeScreenshot();
        WriteToReport(Status.Fail, "Fail error: " + ex);
        Assert.That(false);
    }
}

/// <summary>
/// Assert true if string does not contain value in matcher
///
/// </summary>
/// <param name="container"></param>
/// <param name="matcher"></param>
public void AssertNotContains(string container, string matcher)
{
    try
    {
        WriteToReport(Status.Pass, "Pass: Element is correct");
        Assert.That(!container.Contains(matcher));
    }
    catch (Exception ex)
    {
        TakeScreenshot();
        WriteToReport(Status.Fail, "Fail error: " + ex);
        Assert.That(false);
    }
}

/// <summary>

```

```

/// AssertIsTrue
///
/// </summary>
/// <param name="by"></param>
public void AssertIsTrue(bool by)
{
    try
    {
        WriteToReport(Status.Pass, "Pass: condition is true");

        Assert.That(by);
    }
    catch (Exception ex)
    {
        TakeScreenshot();
        WriteToReport(Status.Fail, "Fail error: " + ex);
        Assert.That(false);
    }
}

/// <summary>
/// AssertEquals
/// checks that the Container string equals the Matcher string
/// true if found
/// false if not
/// Template function so can accept string, int or bool
/// </summary>
/// <typeparam name="T"></typeparam>
/// <param name="container"></param>
/// <param name="matcher"></param>

public void AssertEquals<T>(T container, T matcher)
{
    try
    {
        Assert.That(container.Equals(matcher));
        WriteToReport(Status.Pass, "Pass: " + container + " does match " + matcher);
    }
    catch (Exception ex)
    {
        TakeScreenshot();
        WriteToReport(Status.Fail, "Fail error: " + ex);
        Assert.That(false);
    }
}

/// <summary>
/// AssertIsTrue
///
/// </summary>

```

```

    /// <param name="by"></param>
    public void AssertIsFalse(bool by)
    {
        try
        {
            Assert.That(!by);
            WriteToReport(Status.Pass, "Pass: condition is true");
        }
        catch (Exception ex)
        {
            TakeScreenshot();
            WriteToReport(Status.Fail, "Fail error: " + ex);
            Assert.That(false);
        }
    }
}

```

Finally the AssertFunctions class inherits from Reporting so lets us also see that

```

public class Reporting
{
    protected static ExtentReports Extent;
    protected static ExtentTest Test;
    protected static IWebDriver Driver = null;

    /// <summary>
    /// CloseUpReport
    /// Close the report and rename it
    /// </summary>
    protected static void CloseUpReport(string featureName)
    {
        Extent.Flush();
        //wait for flush to complete
        var sw = new Stopwatch();
        sw.Start();
        while (sw.Elapsed < TimeSpan.FromSeconds(3))
        {
            //sleep
        }
        //rename report file from index.html to a unique name
        var newFileName = featureName.Trim() +
DateTime.Now.ToString("ddMMyyyyHHmmss") + ".html";
        newFileName = newFileName.Replace(" ", string.Empty);
        const string oldFileName = "index.html";
        var pathName =
Path.Combine(Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location) +
"\\TestResults\\");
        File.Move(pathName + oldFileName, pathName + newFileName);
    }
}

```

```

        protected static void ResetReportVariables(string featureName)
        {
            var fileName = featureName.Trim() + DateTime.Now.ToString("ddMMyyyyHHmmss")
+ ".html";
            fileName = fileName.Replace(" ", string.Empty);
            var PathName =
Path.Combine(Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location) +
"\\TestResults\\");

            var htmlReporter = new ExtentHtmlReporter(PathName + fileName);
            htmlReporter.Config.DocumentTitle = "Automation testing report for feature " +
featureName;
            htmlReporter.Config.ReportName = fileName;
            htmlReporter.Config.Theme =
AventStack.ExtentReports.Reporter.Configuration.Theme.Dark;

            Extent = new ExtentReports();
            Extent.AttachReporter(htmlReporter);
            Extent.AddSystemInfo("Host Name", Environment.MachineName);
            Extent.AddSystemInfo("Environment", "Spicetheworld.com");
            Extent.AddSystemInfo("User Name",
System.Security.Principal.WindowsIdentity.GetCurrent().Name);
        }

        /*
        * Sends theMessage to the report page
        */
        public static void WriteToReport(string theMessage, Status status = Status.Info)
        {
            try
            {
                Test.Log(status, theMessage);
            }
            catch (Exception ex)
            {
                Test.Log(status, theMessage + "Error: " + ex.Message);
            }
        }

        public void WriteToReport(Status status, string theMessage)
        {
            try
            {
                Test.Log(status, theMessage);
            }
            catch (Exception ex)
            {
                Test.Log(status, theMessage + "Error: " + ex.Message);
            }
        }
    }

```

```

    protected void TakeScreenshot()
    {
        try
        {
            Screenshot ss = ((ITakesScreenshot)Driver).GetScreenshot();
            Test.AddScreenCaptureFromBase64String(ss.ToString(), "Screenshot");
        }
        catch
        {
        }
    }
}

```

Also, for completeness here is the hook file, this is called SpiceHook.cs

```

public sealed class SpiceHook: SpiceClass
{
    [BeforeFeature(Order = 1)]
    [Scope(Tag = "SpiceRegression")]
    public static void BeforeFeature()
    {
        BeforeTheFeature();
    }

    [BeforeScenario]
    public void BeforeScenario()
    {
        //TODO: implement logic that has to run before executing each scenario
    }

    [AfterScenario]
    public void AfterScenario()
    {
        //TODO: implement logic that has to run after executing each scenario
    }

    [AfterFeature]
    public static void AfterFeature()
    {
        AfterTheFeature();
    }
}

```

One thing I have not included much of in this example is comments. Comments are extremely important and extremely useful. Good comments will make your code easy to understand and easy to maintain. They are not only helpful to you, but also other current and future members of your QA team will also find them useful. Comments are a good guide to help explain what you were thinking when you wrote the test code, this is invaluable when

you re-visit the same code a few years later and then you think to yourself *what is that supposed to do?* In the same way, programmers should always comment on their code, so should QA's. Never let anyone say that comments are not required in testing code or that the code should explain itself. Such people are either too lazy to write comments or too stupid to understand their value.

So that is it a partial working example of a C#/Selenium test suite that I have used on www.spicetheworld.com. The complete suite also handles backend maintenance, so I was unable to share this with you but hopefully what has been shown is helpful. Now let us look at the same project in Ruby.

A complete Ruby example






























So, let us now look at a complete worked example using Ruby and Watir. The image on the following page shows the installed gems in the RubyMine application. Like Visual Studio, RubyMine is a dedicated Ruby and Rails development environment with a good user base and support. The IDE provides a wide range of essential tools for Ruby developers, tightly integrated to create a convenient, stable environment for productive Ruby development and stable automated test suites.

For this example, I will assume you have already installed RubyMine, Ruby and Watir. So, here are the contents of the gem file.

```
source 'https://rubygems.org'

gem 'watir'
gem 'cucumber'
gem 'data_magic'
gem 'faker'
gem 'page-object'
gem 'rake'
gem 'rspec'
gem 'gherkin'
gem 'require_all'
gem 'faker'
gem 'fig_newton'
gem 'ffi'
```

Okay, so when you're ready to continue you will notice that on the following page is the code for this example.

- ▼  SpiceTheWorld [cucumber_and_cheese] C:\Users\kev\...
 - ▼  config
 -  config.yml
 - ▼  features
 - ▼  pages
 -  loginpage.rb
 -  mainpage.rb
 -  menuselectionpage.rb
 -  printrecipepage.rb
 -  recipepage.rb
 -  selectionpage.rb
 -  submitpage.rb
 - ▼  step_definitions
 -  loginsteps.rb
 -  mainsteps.rb
 -  menuselectionsteps.rb
 -  printrecipesteps.rb
 -  recipesteps.rb
 -  selectionsteps.rb
 -  submitsteps.rb
 - ▼  support
 -  env.rb
 -  hooks.rb
 -  shared.rb
 -  SpiceTheWorld.feature
 -  cucumber.yml
 -  Gemfile
 -  Gemfile.lock
 -  Rakefile

So hopefully you will see that the structure is fairly similar to the C# project, this is intentional as I have tried to maintain the page object aspect to both projects. Now let us start looking at the files within the project and as with the previous chapter, we will start with the feature file which is called `SpiceTheWorld.feature`

@SpiceRegression

Feature: SpiceFeature

Full automated testing of SpiceTheWorld.com public pages

Scenario Outline: Navigate to the search option and click the Search button

Given we set the destination to homepage

When we search with "<Search>"

Then we validate the destination includes "<Destination>"

Examples:

Search Destination
shark selection.aspx?ndx=SRCHshark
kevin Default.aspx

Scenario Outline: Navigate using the site menu system

Given we set the destination to homepage

When we select this menu option "<toplevel>" and "<sublevel>"

Then we validate the destination includes "<URL>"

Examples:

toplevel sublevel URL
recipes curry selection.aspx?ndx=IN
recipes thai selection.aspx?ndx=TH
information spices list.aspx?ndx=SP
information gloss list.aspx?ndx=GL
information grow History.aspx?ndx=GR
resources contact contact.aspx

Scenario Outline: Data entry test

Given we set the destination to homepage

And we decide to submit a recipe

When the data is submitted "<name>","<descr>","<serves>","<uname>","<email>","<ingred>","<method>"

Then the expected result is "<result>"

Examples:

name descr serves uname email ingred method result
inpTest1 5
inpTest2 description 5
inpTest3 description 4-6 4
inpTest4 description 4-6 Kev 3

inpTest5	description	4-6	Kev		ingred		2	
inpTest6	description	4-6	Kev		ingred	method	1	
inpTest7	description	4-6	Kev	fred	ingred	method	0	
inpTest8	description	4-6	Kev	kev@keysbox.com	ingred	method		
0								

Scenario: Register a new account

Given we set the destination to login page

When we create a new account

Then the logout button is now visible

Scenario: Login and add a recipe to your folder

Given we login as the test user

When we add a recipe to folder

Then the recipe is visible in the folder page

Scenario: Login and empty your folder

Given we login as the test user

When we add a recipe to folder

Then we can remove all recipes from the folder

Scenario: Login and add a recipe to your menu

Given we login as the test user

When we add a recipe to menu

Then the recipe is visible on the menu page

Scenario: Print the selected recipe

Given we set the destination to homepage

When we view a random recipe

Then the recipe can be printed

Let us now look at the hooks.rb file. As before this is called when a new test run is invoked.

require 'watir'

Before do |scenario|

options = Selenium::WebDriver::Chrome::Options.new(args: ['disable-infobars', 'ignore-certificate-errors'])

options.add_preference(:credentials_enable_service, false)

options.add_preference(:password_manager_enabled, false)

driver = Selenium::WebDriver.for :chrome, options: options

accept_next_alert = true

Watir.default_timeout = 60

driver.manage.timeouts.script_timeout = 60

driver.manage.timeouts.page_load = 60

driver.manage.timeouts.implicit_wait = 30

```

driver.manage.timeouts.script_timeout = 30
verification_errors = []
begin
  driver.manage.window.maximize
rescue StandardError => e
  puts "Driver failed when trying to maximise the window: #{e}"
end
@browser = Watir::Browser.new driver
end

```

```

After do
  @browser.close
end

```

Within a rubymine project you will also have a file called env.rb

```

require 'rspec'
require 'page-object'
require 'data_magic'
require 'faker'

```

World(PageObject::PageFactory)

Also, we have a shared class which is where I store logic that would otherwise be duplicated, in this example it is a very small file and is called shared.rb

```

class Shared
  $recipe_name = ''
end

```

As you can see there are Before and After feature calls and Before and After Scenario calls. So, these functions are good for initialising and setting up before each Feature and each Scenario as required. You may also notice that Tags can be assigned so it is possible to have different calls depending on what Tag is being used.

There are also After calls available which are very useful for after test operations such as removing test data, report creation and whatever else you require.

The next files to look at are the step definition files. These link the scenarios within the Feature file to actual code that will control the tests. Once again there are 7 files in total and each one relates to a page class.

loginsteps.rb

```

Given(/^we set the destination to login page$/) do
  visit(LoginPage)
end

```

end

When(/^we create a new account\$/) **do**

on(*LoginPage*).create_user

end

Then(/^the logout button is now visible\$/) **do**

expect(on(*LoginPage*).test_logout).to be **true**

end

Given(/^we login as the test user\$/) **do**

visit(*LoginPage*)

on(*LoginPage*).user_login()

end

mainsteps.rb

Given(/^we set the destination to homepage\$/) **do**

visit(*MainPage*)

end

When(/^we navigate to the destination "([^"]*)"\$/) **do** |url|

on(*MainPage*).load_destination(url)

end

Then(/^we validate the destination includes "([^"]*)"\$/) **do** |url|

expect(@browser.url.downcase).to include(url.downcase)

end

Then(/^we search with "([^"]*)"\$/) **do** |dest|

on(*MainPage*).start_search(dest)

end

When(/^we select this menu option "([^"]*)" and "([^"]*)"\$/) **do** |top, sub|

on(*MainPage*).use_menu(top, sub)

end

Given(/^we decide to submit a recipe\$/) **do**

on(*MainPage*).use_menu("resources", "submit")

end

menuselectionsteps.rb

Then(/^the recipe is visible on the menu page\$/) **do**

on(*MenuSelectionPage*).goto_menu_page

expect(@browser.text.include?(\$recipe_name)).to be **true**

end

printrecipesteps.rb

```
Then(/^the recipe can be printed$/) do
  on(PrintRecipePage).print_recipe
  expect(@browser.url.downcase).to include('printrecipe.aspx')
end
```

recipesteps.rb

```
When(/^we add a recipe to folder$/) do
  on(RecipePage).select_random_recipe(1)
end
```

```
When(/^we add a recipe to menu$/) do
  on(RecipePage).select_random_recipe(2)
end
```

```
When(/^we view a random recipe$/) do
  on(RecipePage).select_random_recipe(3)
end
```

selectionsteps.rb

```
Then(/^the recipe is visible in the folder page$/) do
  on(SelectionPage).goto_folder_page
  expect(@browser.text.include?($recipe_name)).to be true
end
```

```
Then(/^we can remove all recipes from the folder$/) do
  on(SelectionPage).empty_my_folder
end
```

submitsteps.rb

```
When(/^the data is submitted "([^"]*)" "([^"]*)" "([^"]*)" "([^"]*)" "([^"]*)" "([^"]*)" "([^"]*)"$/) do |recipe_name, description, serves, your_name, email, ingredients, method|
  on(SubmitPage).submit_recipe(recipe_name, description, serves, your_name, email, ingredients, method)
end
```

```
Then(/^the expected result is "([^"]*)"$/) do |result|
  expect(on(SubmitPage).return_error_count).to eql(result.to_i)
end
```

Each of the Step Definition files list above link to a page class, a good example of this being [on(SelectionPage).empty_my_folder]. As expected, there are also 7 of those, so let is now look at these files.

loginpage.rb

```
class LoginPage
  include PageObject

  page_url "https://spicetheworld.com/login.aspx"

  a(:submit, id: 'submit')
  text_field(:txt_login, id: 'TxtLogin')
  text_field(:txt_pass, id: 'TxtPass')
  text_field(:txt_email, id: 'TxtEmail')
  text_field(:txt_names, id: 'TxtNames')
  text_field(:txt_pass1, id: 'TxtPass1')
  text_field(:txt_pass2, id: 'TxtPass2')
  button(:register, id: 'BtnRegister')
  button(:login, id: 'BtnLogin')
  link(:logout, text: 'Logout')

  def create_user
    self.txt_email = randon_string(8) + "@" + randon_string(5) + ".com"
    self.txt_names = "Test User"
    self.txt_pass1 = "xyzABC123"
    self.txt_pass2 = "xyzABC123"
    register
  end

  def user_login
    self.txt_login = "testuser@kevsbox.com"
    self.txt_pass = "xyzABC123"
    login
  end

  def test_logout
    logout_element.visible?
  end

  private
  def randon_string(len)
    rand(36**len).to_s(36)
  end
end
```

mainpage.rb

```
class MainPage
  include PageObject
```

```
page_url "https://spicetheworld.com"
```

```
text_field(:search, id: 'Menu1_searchTxt')
a(:recipes, id: 'recipes')
a(:information, id: 'information')
a(:resources, id: 'resources')
a(:curryrec, id: 'curryrec')
a(:thairec, id: 'thairec')
a(:spices, id: 'spices')
a(:glossary, id: 'glossary')
a(:grow, id: 'grow')
a(:contact, id: 'contact')
a(:submit, id: 'submit')
```

```
def use_menu(toplevel, sublevel)
  hover_toplevel(toplevel)
  select_sublevel(sublevel)
end
```

```
def load_destination(url)
  @browser.goto(EnvConfig['url'] + url)
end
```

```
def start_search(dest)
  #input the search text followed by Enter key
  self.search = dest
  @browser.text_field(id => "Menu1_searchTxt").send_keys :enter
end
```

```
private
#private def's only visible with mainpage
def hover_toplevel(toplevel)
  if (toplevel=="recipes")
    @browser.element(id: "recipes").hover
  elsif (toplevel=="information")
    @browser.element(id: "information").hover
  else
    @browser.element(id: "resources").hover
  end
end
```

```
def select_sublevel(sublevel)
  if(sublevel=="curry")
    curryrec
  elsif (sublevel=="thai")
    thairec
  elsif (sublevel=="spices")
    spices
  end
end
```

```
    elsif (sublevel=="gloss")
      glossary
    elsif(sublevel=="grow")
      grow
    elsif (sublevel=="contact")
      contact
    elsif (sublevel=="submit")
      submit
    end
  end
end
```

menuselectionpage.rb

```
class MenuSelectionPage < Shared
  include PageObject

  page_url "https://spicetheworld.com/menuselection.aspx"

  def goto_menu_page
    @browser.goto("https://spicetheworld.com/menuselection.aspx?ndx=YM")
  end
end
```

printrecipepage.rb

```
class PrintRecipePage < Shared
  include PageObject

  page_url "https://spicetheworld.com/printRecipe.aspx"

  a(:printnow, class: 'print')

  def print_recipe
    @browser.windows()[1].use()
  end
end
```

recipepage.rb

```
class RecipePage < Shared
  include PageObject

  page_url "https://spicetheworld.com/recipe.aspx"

  link(:folderbutton, id: "FolderButton")
  link(:menubutton, id: "MenuButton")
  link(:printlink, id: "printLink")
  p(:recipe_name, id: 'name')

  def select_random_recipe(dest)
    @browser.goto("https://spicetheworld.com/recipe.aspx?ndx=" + rand(140).to_s)
    $recipe_name = recipe_name_element.text
    if (dest == 1)
      add_to_folder
    elsif (dest == 2)
      add_to_menu
    else
      select_print
    end
  end

  private
  def add_to_folder
    if folderbutton_element.exists?
      folderbutton
    end
  end
end
```

```
def add_to_menu
  if menubutton_element.exists?
    menubutton
  end
end
```

```
def select_print
  if printlink_element.exists?
    printlink
  end
end
```

```
end
```

selectionpage.rb

```
class SelectionPage < Shared
  include PageObject

  page_url "https://spicetheworld.com/selection.aspx"

  link(:delete_it, name: "deleteit")

  def goto_folder_page
    @browser.goto("https://spicetheworld.com/selection.aspx?ndx=WL")
  end

  def empty_my_folder
    @browser.goto("https://spicetheworld.com/selection.aspx?ndx=WL")
    if delete_it_element.present?
      delete_it
    end
  end
end
```

submitpage.rb

```
class SubmitPage
  include PageObject

  page_url "https://spicetheworld.com/submit.aspx"

  text_field(:name, id: 'name')
  text_field(:description, id: 'description')
```

```

text_field(:serves, id: 'serves')
text_field(:strength, id: 'strength')
text_field(:prep, id: 'prep')
text_field(:cook, id: 'cook')
text_field(:yname, id: 'yname')
text_field(:ymail, id: 'ymail')
checkbox(:veganChk, id: 'veganChk')
checkbox(:veggyChk, id: 'veggyChk')
text_area(:ingreds, id: 'ingreds')
text_area(:method, id: 'method')
button(:submit_button, id: 'submitButton')

```

```

def return_error_count
  #return the number of li elements that currently exist with the
  #div element ValidationSummary1
  counter = @browser.div(:id => "ValidationSummary1").ul.lis.length
  counter
end

```

```

def submit_recipe(recipe_name, description, serves, your_name, email, ingredients, method)
  self.name = recipe_name
  self.description = description
  self.serves = serves
  self.strength = 3
  self.prep = 20
  self.cook = 30
  self.yname = your_name
  self.ymail = email
  self.ingreds = ingredients
  self.method = method
  veganChk_element.check
  veggyChk_element.check
  submit_button
end

```

end

So, there you go, that is the class files. Now let us look at some of the logic in here. A test field is defined as this -

```
[text_field(:strength, id: 'strength')]
```

You will have noticed data can be entered thus –

```
self.strength = 3
```

It is as simple as that, also to click a button that has been defined thus –

```
button(:submit_button, id: 'submitButton')
```

The command submit_button will do the job.

Another cool feature to note is inheritance, this line –

```
class SelectionPage < Shared
```

Means that SelectionPage class inherits Shared class and can use everything within the class.

One other nice feature is the private keyword

```
private
```

```
def random_string(len)
```

```
  rand(36**len).to_s(36)
```

```
end
```

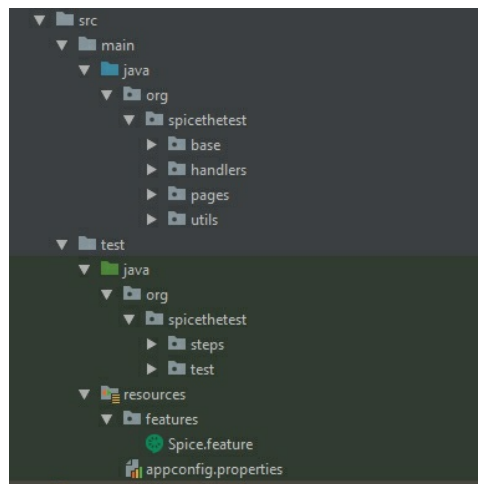
Any definitions listed below this keyword are private and only visible to the class they belong in.

So that is it a partial working example of a Ruby/Watir test suite that I have used on www.spicetheworld.com. The complete suite also handles backend maintenance, so I was unable to share this with you but hopefully what has been shown is helpful. Now let us look at another important skill for any agile QA, structured Query Language.

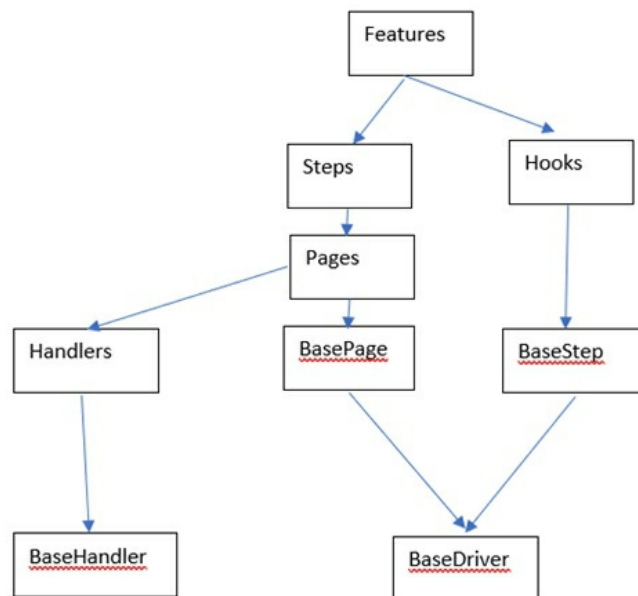
A Complete Java Example

So, let us now look at a complete working example using Java and IntelliJ. Like Visual Studio and RubyMine, IntelliJ is a dedicated Java development environment with an extremely good user base and support. The IDE is very intuitive and responsive while providing a wide range of essential tools for Java developers with which to create high quality applications as well as very stable test projects.

For this example, we will assume you have already installed IntelliJ and all required dependencies, this is beyond the scope of this book. So, here is the layout of the project.



Also here is how the files interact with each other.



Now just like our previous examples we will start with the feature file, in this case the Spice.feature file.

@SpiceRegression

Feature: SpiceFeature

Scenario: 01 Start the test run

Given we set the destination to homepage

When we search with "shark" for "45"

Then we validate the destination includes "Home/Recipe/89"

Scenario: 02 Navigate to the search option and click the Search button

Given we set the destination to homepage

When we search with "kevin" for "10"

Then we validate the destination includes "Home/Search"

Scenario Outline: 03 Navigate Recipes using the site menu system

Given we set the destination to homepage

When we select this menu option "<toplevel>" and "<sublevel>"

Then we validate the destination includes "<URL>"

Examples:

	toplevel	sublevel	URL
1	1		Home/Selection/CH
1	2		Home/Selection/IN
1	3		Home/Selection/CK
1	4		Home/Selection/CA
1	5		Home/Selection/TH
1	6		Home/Selection/TR

Scenario Outline: 09 Navigate Informaton using the site menu system

Given we set the destination to homepage

When we select this info menu option "<toptlevel>" and "<sublevel>"

Then we validate the destination includes "<URL>"

Examples:

toplevel	sublevel	URL
2	1	Info/List/CH
2	2	Info/List/SP
2	3	Info/List/CT
2	4	Info/List/GL
2	5	Info/History/CU
2	6	Info/History/CT
2	7	Info/Scoville
2	8	Info/History/GR

Scenario: 17 View a vegan recipe

Given we set the destination to homepage

When we select this menu option "1" and "6"

And we then select a vegan recipe

Then the vegan symbol is displayed

Scenario: 18 Print the selected recipe

Given we set the destination to homepage

When we select this menu option "1" and "5"

And we then select a thai meat recipe

Then the recipe can be printed

Scenario: 19 Register a new account

Given we set the destination to login page

When we create a new account

Then the logged in message is visible

And I can then logout

Scenario: 20 Login and add a recipe to your folder

Given we login as the test user

When we add a recipe to folder

Then the recipe is visible in the folder page

Scenario: 21 Login and add and remove recipe to your folder

Given we login as the test user

When we add a recipe to folder

Then the recipe is visible in the folder page

And then empty the folder

Scenario: 22 Login and add a recipe to your menu

Given we login as the test user

When we add a recipe to menu

Then the recipe is visible in the menu page

Scenario: 23 Login and add and remove recipe to your menu

Given we login as the test user

When we add a recipe to menu

Then the recipe is visible in the menu page

And then empty the folder

As with the previous C# and Ruby examples the feature file links to the Steps classes and we also have the special hooks class so let us look at them next beginning with hooks.

Hooks

```
package org.spicethetest.steps;
import cucumber.api.Scenario;
import cucumber.api.java.After;
import cucumber.api.java.Before;
import org.openqa.selenium.WebDriver;
import org.spicethetest.base.BaseDriver;
import org.spicethetest.base.BaseStep;
import java.util.ArrayList;

public class Hooks extends BaseStep
{
    @Before
    public static void BeforeFeature(Scenario scenario)
    {
        System.out.println("Before scenario " + scenario);
    }

    @After
    public static void AfterFeature(Scenario scenario)
    {
        System.out.println("After scenario " + scenario);
        WebDriver driver = BaseDriver.getDriver();
        try {
            if(null != driver)
                driver.quit();
        } catch (Exception e) {
            e.printStackTrace();
        }
        BaseDriver.resetDriver();
        winHandleList = new ArrayList<>();
    }
}
```

Then we have out standard steps classes.

MainSteps

```

package org.spicethetest.steps;
import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;
import org.spicethetest.pages.MainPage;

import static org.junit.Assert.assertTrue;

public class MainSteps extends MainPage
{
    @Given("^we set the destination to homepage$")
    public void we_set_the_destination_to_homepage() throws Throwable {
        visit();
    }

    @When("^we search with \"([^\"]*)\" for \"([^\"]*)\"$")
    public void we_search_with(String searchStr, int span) throws Throwable {
        doASearch(searchStr, span);
    }

    @Then("^we validate the destination includes \"([^\"]*)\"$")
    public void we_validate_the_destination_includes(String url) throws Throwable {
        assertTrue(_driver.getCurrentUrl().toString().contains(url));
    }

    @When("^we select this menu option \"([^\"]*)\" and \"([^\"]*)\"$")
    public void we_select_this_menu_option_and(int menu1, int menu2) throws Throwable {
        selectMenuItem(menu1, menu2);
    }

    @When("^we select this info menu option \"([^\"]*)\" and \"([^\"]*)\"$")
    public void we_select_this_info_menu_option_and(int menu1, int menu2) throws Throwable {
        selectMenuItem(menu1, menu2);
    }
}

```

LoginSteps

```

package org.spicethetest.steps;
import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;
import org.spicethetest.pages.LoginPage;
import static org.spicethetest.handlers.AssertHandler.assertElementPresent;

public class LoginSteps extends LoginPage

```

```

{
    @Given("^we set the destination to login page$")
    public void we_set_the_destination_to_login_page() throws Throwable {
        visit();
    }

    @When("^we create a new account$")
    public void we_create_a_new_account() throws Throwable {
        createAndLogin();
    }

    @Then("^the logged in message is visible$")
    public void the_logged_in_message_is_visible() throws Throwable {
        assertElementPresent(loggedIn);
    }

    @Then("^I can then logout$")
    public void i_can_then_logout() throws Throwable {
        userLogout();
    }

    @Given("^we login as the test user$")
    public void we_login_as_the_test_user() throws Throwable {
        loginTestUser();
    }
}

```

RecipeSteps

```

package org.spicethetest.steps;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;
import org.spicethetest.pages.RecipePage;

public class RecipeSteps extends RecipePage
{
    @Then("^the vegan symbol is displayed$")
    public void the_vegan_symbol_is_displayed() throws Throwable {
        checkVeganSymbol();
    }

    @Then("^the recipe can be printed$")
    public void the_recipe_can_be_printed() throws Throwable {
        printRecipe();
    }

    @When("^we add a recipe to folder$")
    public void we_add_a_recipe_to_folder() throws Throwable {
        selectARecipe();
    }
}

```

```

        addToFolder();
    }

    @When("^we add a recipe to menu$")
    public void we_add_a_recipe_to_menu() throws Throwable {
        selectARecipe();
        addToMenu();
    }
}

```

Selection Steps

```

package org.spicethetest.steps;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;
import org.spicethetest.pages.SelectionPage;
import static org.junit.Assert.assertTrue;

public class SelectionSteps extends SelectionPage
{
    @When("^we then select a vegan recipe$")
    public void we_then_select_a_vegan_recipe() throws Throwable {
        selectVegan();
    }

    @When("^we then select a thai meat recipe$")
    public void we_then_select_a_thai_meat_recipe() throws Throwable {
        selectThaiMeat();
    }

    @Then("^the recipe is visible in the folder page$")
    public void the_recipe_is_visible_in_the_folder_page() throws Throwable {
        assertTrue(folderContainsRecipe(0));
    }

    @Then("^the recipe is visible in the menu page$")
    public void the_recipe_is_visible_in_the_menu_page() throws Throwable {
        assertTrue(folderContainsRecipe(1));
    }

    @Then("^then empty the folder$")
    public void then_empty_the_folder() throws Throwable {
        assertTrue(folderCountIsZero()==0);
    }
}

```

So as you are all now probably aware the Steps classes link to Page classes just like they do in the C# and Ruby examples, so they are shown next.

MainPage

```
package org.spicethetest.pages;

import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.spicethetest.base.BasePage;
import org.spicethetest.handlers.ActionHandler;
import static org.spicethetest.handlers.ActionHandler.*;

public class MainPage extends BasePage
{
    protected static String PageUrl = "https://spicetheworld.com/";

    @FindBy(name = "SearchString") public static WebElement searchString;
    @FindBy(css = "[value='Search']") public static WebElement searchBtn;
    @FindBy(id = "recipes") public static WebElement recipes;
    @FindBy(id = "information") public static WebElement information;
    @FindBy(id = "resources") public static WebElement resources;
    @FindBy(id = "scoville") public static WebElement scoville;
    @FindBy(id = "grow") public static WebElement grow;
    @FindBy(id = "historycurry") public static WebElement historycurry;
    @FindBy(id = "historychilli") public static WebElement historychilli;
    @FindBy(id = "glossary") public static WebElement glossary;
    @FindBy(id = "currytypes") public static WebElement currytypes;
    @FindBy(name = "selectit") public static WebElement selectit;
    @FindBy(id = "spice") public static WebElement spice;
    @FindBy(id = "chilli") public static WebElement chilli;
    @FindBy(id = "chillirec") public static WebElement chillirec;
    @FindBy(id = "curryrec") public static WebElement curryrec;
    @FindBy(id = "chinrec") public static WebElement chinrec;
    @FindBy(id = "caribrec") public static WebElement caribrec;
    @FindBy(id = "thairec") public static WebElement thairec;
    @FindBy(id = "otherrec") public static WebElement otherrec;

    protected void visit()
    {
        System.out.println("Load page " + PageUrl);
        navigateTo(PageUrl);
        ActionHandler.waitForPageLoad();
    }

    protected void doASearch(String searchStr, int span)
    {
        System.out.println("DoASearch");
    }
}
```



```

        ActionHandler.setText(searchString, searchStr);
        ActionHandler.click(searchBtn);
        if (isElementPresent(selectit))
            ActionHandler.click(selectit);
    }

    protected void selectMenuItem(int menu1, int menu2) {
        System.out.println("SelectMenuItem");
        selectMenu1(menu1);
        if (menu1 == 1)
            selectMenu2(menu2);
        if (menu1 == 2)
            selectInfoMenu(menu2);
    }

    protected void selectMenu1(int menu) {
        System.out.println("SelectMenu1");
        if (menu == 1) moveToElement(recipes);
        else if (menu == 2) moveToElement(information);
        else moveToElement(resources);
    }

    protected void selectMenu2(int menu) {
        System.out.println("SelectMenu2");
        if (menu == 1) click(chillirec);
        else if (menu == 2) click(curryrec);
        else if (menu == 3) click(chinrec);
        else if (menu == 4) click(caribrec);
        else if (menu == 5) click(thairec);
        else if (menu == 6) click(otherrec);
    }

    protected void selectInfoMenu(int menu) {
        System.out.println("SelectInfoMenu");
        if (menu == 1) click(chilli);
        else if (menu == 2) click(spice);
        else if (menu == 3) click(currytypes);
        else if (menu == 4) click(glossary);
        else if (menu == 5) click(historycurry);
        else if (menu == 6) click(historychilli);
        else if (menu == 7) click(scoville);
        else if (menu == 8) click(grow);
    }
}

```

LoginPage

```

package org.spicethetest.pages;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.spicethetest.base.BasePage;
import org.spicethetest.handlers.ActionHandler;
import java.text.SimpleDateFormat;
import java.util.Date;
import static org.spicethetest.handlers.ActionHandler.*;
import static org.spicethetest.handlers.AssertHandler.assertElementPresent;

public class LoginPage extends BasePage
{

    protected static String PageUrl = "https://spicetheworld.com/User/Login";

    @FindBy(id = "resources") public static WebElement resources;
    @FindBy(id = "logout") public static WebElement logout;
    @FindBy(name = "UserString") public static WebElement userString;
    @FindBy(name = "PassString") public static WebElement passString;
    @FindBy(name = "LoggedIn") public static WebElement loggedIn;
    @FindBy(css = "[value = 'Login']") public static WebElement login;

    protected void visit()
    {
        System.out.println("Load page " + PageUrl);
        navigateTo(PageUrl);
        ActionHandler.waitForPageLoad();
    }

    protected void createAndLogin()
    {
        System.out.println("CreateAndLogin");
        Date date = new Date(System.currentTimeMillis());
        SimpleDateFormat formatter = new SimpleDateFormat("ddMMyyyyHHmmss");
        String newUser = formatter.format(date);
        setText(userString, newUser + "@kevsbox.com");
        setText(passString, newUser);
        click(login);
    }

    protected void loginTestUser()
    {
        visit();
        System.out.println("LoginTestUser");
        setText(userString, "tester@kevsbox.com");
        setText(passString, "Pr1vatePW");
        click(login);
    }
}

```

```

    protected void userLogout()
    {
        System.out.println("UserLogout");
        moveToElement(resources);
        waitUntilElementVisible(logout);
        click(logout);
        moveToElement(resources);
        assertElementPresent(login);
    }
}

```

RecipePage

```

package org.spicethetest.pages;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.spicethetest.base.BaseDriver;
import org.spicethetest.base.BasePage;
import java.util.Random;
import static org.junit.Assert.assertTrue;
import static org.spicethetest.handlers.ActionHandler.*;
import static org.spicethetest.handlers.AssertHandler.assertElementPresent;

public class RecipePage extends BasePage
{
    protected static String PageUrl = "https://spicetheworld.com/Home/Recipe/";

    @FindBy(id = "printLink") public static WebElement printLink;
    @FindBy(id = "bodyPop") public static WebElement bodyPop;
    @FindBy(id = "name") public static WebElement name;
    @FindBy(id = "walletLink") public static WebElement walletLink;
    @FindBy(id = "menuLink") public static WebElement menuLink;
    @FindBy(css = "[title='Vegan safe']") public static WebElement veganSafe;

    protected void visit()
    {
        System.out.println("Load page " + PageUrl);
        navigateTo(PageUrl);
        waitForPageLoad();
    }

    protected void checkVeganSymbol()
    {
        System.out.println("CheckVeganSymbol");
        assertElementPresent(veganSafe);
    }
}

```

```

    protected void selectARecipe()
    {
        System.out.println("CheckVeganSymbol");
        Random rnd = new Random();
        navigateTo(PageUrl + rnd.nextInt((100-1)+1));
        BaseDriver.recipeName = getElementText(name);
    }

    protected void addToMenu()
    {
        System.out.println("CheckVeganSymbol");
        if (isElementPresent(menuLink))
            click(menuLink);
    }

    protected void addToFolder()
    {
        System.out.println("CheckVeganSymbol");
        if (isElementPresent(walletLink))
            click(walletLink);
    }

    protected void printRecipe()
    {
        System.out.println("CheckVeganSymbol");
        if (isElementPresent(printLink))
            click(printLink);
        for (String windowHandle : _driver.getWindowHandles())
        {
            _driver.switchTo().window(windowHandle);
        }
        assertTrue(isElementPresent(bodyPop));
    }
}

```

SelectionPage

```

package org.spicethetest.pages;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.spicethetest.base.BaseDriver;
import org.spicethetest.base.BasePage;
import org.spicethetest.handlers.ActionHandler;
import java.util.List;

```

```

public class SelectionPage extends BasePage
{

```

```
protected static String PageUrl = "https://spicetheworld.com/Home/Selection";  
@FindBy(name = "selectit") public static WebElement selectit;  
@FindBy(name = "deleteit") public static WebElement deleteit;
```

```
protected void visit()  
{  
    System.out.println("Load page " + PageUrl);  
    navigateTo(PageUrl);  
    ActionHandler.waitForPageLoad();  
}
```

```
protected void selectVegan()  
{  
    System.out.println("SelectVegan");  
    for (WebElement recipeLink : _driver.findElements(By.name("selectit")))  
    {  
        if(recipeLink.getText().toLowerCase().contains("vegan"))  
        {  
            recipeLink.click();  
            break;  
        }  
    }  
    for (WebElement recipeLink : _driver.findElements(By.name("selectit")))  
    {  
        recipeLink.click();  
        break;  
    }  
}
```

```
protected void selectThaiMeat()  
{  
    System.out.println("SelectThaiMeat");  
    for (WebElement recipeLink : _driver.findElements(By.name("selectit")))  
    {  
        if(recipeLink.getText().toLowerCase().contains("meats"))  
        {  
            recipeLink.click();  
            break;  
        }  
    }  
    for (WebElement recipeLink : _driver.findElements(By.name("selectit")))  
    {  
        recipeLink.click();  
        break;  
    }  
}
```

```

protected Boolean folderContainsRecipe(int dest)
{
    System.out.println("FolderContainsRecipe");
    if (dest==0)
        navigateTo(PageUrl + "/WL");
    else
        navigateTo(PageUrl + "/RP");

        for (WebElement recipeLink : _driver.findElements(By.name("selectit")))
        {
            if (recipeLink.getText().toLowerCase().equals(BaseDriver.recipeName.toLowerCase()))
            {
                return true;
            }
        }
    return false;
}

protected int folderCountIsZero()
{
    System.out.println("FolderCountIsZero");
    try
    {
        List<WebElement> recipeLinks = _driver.findElements(By.name("deleteit"));
        do
        {
            for (WebElement recipeLink : recipeLinks)
            {
                recipeLink.click();
                break;
            }
            if (recipeLinks.size() != 1)
                recipeLinks = _driver.findElements(By.name("deleteit"));
            else
                break;
        }
        while (recipeLinks.size() != 0);
        return 0;
    }
    catch (Exception ex)
    {
        System.console().writer().println("Error: " + ex);
        return -1;
    }
}
}

```

Now let's as look at the handlers which are called by some of the Page classes.

AssertHandler

```
package org.spicethetest.handlers;
import org.openqa.selenium.WebElement;
import static org.junit.Assert.assertTrue;

public class AssertHandler extends BaseHandler{
    public static void assertElementPresent(WebElement element)
    {
        assertTrue("Element " + element + "is not present", isElementPresent(element));
    }
}
```

ActionHandler

```
package org.spicethetest.handlers;
import org.junit.Assert;
import org.openqa.selenium.*;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.FluentWait;
import org.openqa.selenium.support.ui.Wait;
import org.spicethetest.base.BaseDriver;
import java.util.NoSuchElementException;
import java.util.concurrent.TimeUnit;

public class ActionHandler extends BaseHandler{
    public static JavascriptExecutor js;

    public static JavascriptExecutor setJavaScriptExecutor()
    {
        js = ((JavascriptExecutor) BaseDriver.getDriver());
        return js;
    }

    public static void click(WebElement element)
    {
        click(element, BaseDriver.WAIT_IN_SEC);
    }

    public static void click(WebElement element, int waitTimeInSeconds)
    {

```

```

boolean elementFound = isElementPresent(element);
if (elementFound)
{
    BaseDriver.setWait(waitTimeInSeconds);
    BaseDriver.getWait().until(ExpectedConditions.elementToBeClickable(element));
    try{
        element.click();
    }
    catch (WebDriverException e){
        scrollElementIntoView(element);
        element.click();
    }
}
else{
    Assert.fail("Unable to locate the weblement: " + element);
}
}

```

```

public static void scrollElementIntoView(WebElement element)
{
    setJavaScriptExecutor();
    try{
        js.executeScript("arguments[0].scrollIntoView();", element);
        js.executeScript("window.scrollTo(0,0);", element);
    }
    catch (Exception e){
        //System.out.println("Error while scrolling JS");
    }
}

```

```

public static void setText(WebElement element, String text)
{
    setText(element, text, false);
}

```

```

public static void setText(WebElement element, String text, Boolean giveFocus)
{
    setText(element, text, BaseDriver.WAIT_IN_SEC, giveFocus, true);
}

```

```

public static void setText(WebElement element, String text, int waitTimeInSeconds,
    Boolean giveFocus, Boolean waitForJs)
{
    BaseDriver.setWait(waitTimeInSeconds);
    if (isElementPresent(element, waitTimeInSeconds))
    {
        BaseDriver.getWait().until(ExpectedConditions.visibilityOf(element));
    }
}

```



```

        if (giveFocus){
            click(element);
            BaseDriver.waitFor()
                .until(ExpectedConditions.not(ExpectedConditions.attributeToBe(element, "readonly",
"readonly"))));
        }
        try{
            BaseDriver.waitFor().until(ExpectedConditions.not(ExpectedConditions.attributeToBe(element,
"readonly", "readonly")));
            element.clear();
        }
        catch (InvalidElementStateException ee){
            BaseDriver.waitFor()
                .until(ExpectedConditions.not(ExpectedConditions.attributeToBe(element, "readonly",
"readonly")));
            element.clear();
        }
        element.sendKeys(text);
    }
    else{
        Assert.fail("Cannot find element:" + element);
    }
}

/**
 * Move mouse to cursor to WebElement
 * @param element
 */
public static void moveToElement(WebElement element)
{
    Actions actions = new Actions(BaseDriver.getDriver());
    actions.moveToElement(element).click().perform();
}

private static Wait<WebDriver> fluentWait()
{
    return new FluentWait<>(BaseDriver.getDriver())
        .withTimeout(120, TimeUnit.SECONDS)
        .pollingEvery(10, TimeUnit.SECONDS)
        .ignoring(NoSuchElementException.class)
        .ignoring(StaleElementReferenceException.class);
}

public static WebElement waitUntilElementEnabled(final WebElement element)
{
    return fluentWait().until(ExpectedConditions.elementToBeClickable(element));
}

```

```
}
```

```
public static WebElement waitUntilElementVisible(final WebElement element)
{
    return fluentWait().until(ExpectedConditions.visibilityOf(element));
}
```

```
/**
 * Use this method when click() is working
 * @param webel
 */
public static void sendKeys_ENTER(WebElement webel) {
    if (webel.isEnabled()) {
        webel.sendKeys(Keys.ENTER);
    } else {
        Assert.fail("Enter key cannot be sent:");
    }
}
```

```
public static String getElementText(WebElement element)
{
    String text = element.getText();
    if (text.equalsIgnoreCase(""))
    {
        text = element.getAttribute("value");
    }
    return text;
}
```

```
public static void keyPress_TAB(WebElement webelement) {
    webelement.sendKeys(Keys.TAB);
}
```

```
public static void waitForPageLoad() {
    //Awaitility.await().until(ActionHandler::isloadComplete);
    boolean loaded = false;
    while (!loaded) {
        loaded = isloadComplete();
    }
}
```

```

    public static boolean isloadComplete()
    {
        try {
            return ((JavascriptExecutor) BaseDriver.getDriver()).executeScript("return
document.readyState").equals("loaded")
                || ((JavascriptExecutor) BaseDriver.getDriver()).executeScript("return
document.readyState").equals("complete");
        }
        catch(Exception e){
            System.out.println("\n.... JAVA SCRIPT ERROR :\n");
            //e.printStackTrace();
            return false;
        }
    }
}

```

```

    public static String getAttributeValue(WebElement element, String attribute)
    {
        String text = element.getAttribute(attribute);
        return text;
    }
}

```

Base Handler

```

package org.spicethetest.handlers;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebElement;
import org.spicethetest.base.BaseDriver;

```

```

public class BaseHandler {
    public static JavascriptExecutor js;
    public static final int numberOfTries = 3;

    public static boolean isElementPresent(WebElement element)
    {
        return isElementPresent(element, numberOfTries);
    }
}

```

```

    public static boolean isElementPresent(WebElement element, int customNumberOfTries) {
        boolean elementFound = false;
        while (customNumberOfTries > 0 && !elementFound) {

```

```

    try {
        BaseDriver.waitForElement(element);
        elementFound = element.isDisplayed();
        break;
    } catch (Exception e) {
        customNumberOfTries--;
        ActionHandler.scrollElementIntoView(element);
    }
}
return elementFound;
}
}

```

Finally we have some base classes which are inherited by other classes.

BasePage

```

package org.spicethetest.base;
public class BasePage extends BaseDriver
{
    //public WebDriver _driver;
    public BasePage()
    {
        _driver = BaseDriver.getDriver();
    }

    public void navigateTo(String url)
    {
        BaseDriver.getDriver().get(url);
    }
}

```

BaseStep

```

package org.spicethetest.base;

import org.openqa.selenium.support.PageFactory;
import org.spicethetest.pages.LoginPage;
import org.spicethetest.pages.MainPage;
import org.spicethetest.pages.RecipePage;
import org.spicethetest.pages.SelectionPage;

import java.util.ArrayList;
import java.util.List;

```

```

public class BaseStep extends BaseDriver
{
    public static List<String> winHandleList = new ArrayList<String>();
    //public WebDriver _driver;
    public static MainPage mainpage = new MainPage();
    public static LoginPage loginpage = new LoginPage();
    public static RecipePage recipepage = new RecipePage();
    public static SelectionPage selectionpage = new SelectionPage();

    public BaseStep()
    {
        _driver = BaseDriver.getDriver();
        mainpage = PageFactory.initElements(_driver, MainPage.class);
        loginpage = PageFactory.initElements(_driver, LoginPage.class);
        recipepage = PageFactory.initElements(_driver, RecipePage.class);
        selectionpage = PageFactory.initElements(_driver, SelectionPage.class);
    }
}

```

BaseDriver

```

package org.spicethetest.base;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.edge.EdgeDriver;
import org.openqa.selenium.edge.EdgeOptions;
import org.openqa.selenium.ie.InternetExplorerDriver;
import org.openqa.selenium.ie.InternetExplorerOptions;
import org.openqa.selenium.remote.CapabilityType;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.concurrent.TimeUnit;

```

```

public class BaseDriver {

    static BaseUtility util = new BaseUtility();
    public static int WAIT_IN_SEC=10;
    public static WebDriver _driver = null;
    private static String localBrowser;

```

```
private static WebDriverWait wait = null;
public static String recipeName;
```

```
public static void resetDriver()
{
    _driver = null;
}
```

```
public static WebDriver getDriver()
{
    if(_driver != null){
        return _driver;
    }
```

```
        if(System.getProperty("remote") == null &&
!"true".equals(util.getPropertyObject().getProperty("remote")))
        {
            localBrowser = util.getPropertyObject().getProperty("browser");
            switch (localBrowser) {
                case "ie":
                    _driver = createIEDriver();
                    break;
                case "edge":
                    _driver = createEdgeDriver();
                    break;
                case "chrome":
                    _driver = createChromeDriver();
                    break;
            }
        }
        else {
            System.out.println("\n..... Running on Remote Browser.....");
            _driver = createRemoteWebDriver(util.getPropertyObject().getProperty("remotebrowser"));
        }
    }
```

```
        _driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        _driver.manage().timeouts().pageLoadTimeout(10, TimeUnit.SECONDS);
        _driver.manage().window().maximize();

        return _driver;
    }
```

```

private static WebDriver createIEDriver()
{
    String driverpath =
System.getProperty("user.dir")+"\\_drivers\\win32_3.6\\IEDriverServer.exe";
    System.setProperty("webdriver.ie.driver",driverpath);
    InternetExplorerOptions options = new InternetExplorerOptions();
    options.setCapability("ignoreZoomSetting", true);
    options.setCapability("nativeEvents",false);
    options.setCapability("unexpectedAlertBehaviour", "accept");
    options.setCapability("ignoreProtectedModeSettings", true);
    options.setCapability("disable-popup-blocking", true);
    options.setCapability("enablePersistentHover", true);
    options.setCapability("requireWindowFocus", true);
    options.setCapability(InternetExplorerDriver.INTRODUCE_FLAKINESS_BY_IGNORING_SECURITY_DOMAINS,
false);
    options.setCapability(CapabilityType.PAGE_LOAD_STRATEGY, "normal");
    return new InternetExplorerDriver(options);
}

```

```

private static WebDriver createEdgeDriver() {
    String driverpath = System.getProperty("user.dir")+"\\_drivers\\EdgeDriverServer.exe";
    System.setProperty("webdriver.edge.driver",driverpath);
    EdgeOptions options = new EdgeOptions();
    return new EdgeDriver();
}

```

```

private static WebDriver createChromeDriver() {
    String driverpath = System.getProperty("user.dir")+"\\_drivers\\chromedriver.exe";
    System.setProperty("webdriver.chrome.driver",driverpath);

```

```

        ChromeOptions chromeOptions = new ChromeOptions();
        chromeOptions.addArguments("ignore-certificate-errors");
        chromeOptions.addArguments("disable-infobars");
        return new ChromeDriver(chromeOptions);
}

```

```

private static WebDriver createRemoteWebDriver(String browser) {

```

```

    WebDriver driver = null;
    String urlString = "";
    URL serverUrl = null;
    DesiredCapabilities caps = new DesiredCapabilities();

```

```

    switch(browser) {

```

```

case "ie":
    caps.setCapability("os", "Windows");
    caps.setCapability("os_version", "XP");
    caps.setCapability("browser", "IE");
    caps.setCapability("browser_version", "6.0");
    caps.setCapability("browserstack.local", "false");
    caps.setCapability("browserstack.selenium_version", "3.5.2");
    break;
case "chrome":

    caps.setCapability("browser", "chrome");
    caps.setCapability("browser_version", "73");
    caps.setCapability("os", "Windows");
    caps.setCapability("os_version", "10");
    break;
}

    caps.setCapability("browserstack.local", "true");
    String username = "ittesters2";
    String accessKey = "uyzJ5DoHDNQe8pfwzcqQ";
    String URLStr = "https://" + username + ":" + accessKey +
"@hub.browserstack.com/wd/hub";

    try {
        serverUrl = new URL(URLStr);
    } catch (MalformedURLException mue) {mue.printStackTrace();}
    driver = new RemoteWebDriver(serverUrl, caps);
    return driver;

}

public static void setWait(int waitTimeInSec){
    wait = new WebDriverWait(getDriver(), waitTimeInSec);
}

public static WebDriverWait getWait(){
    if(wait==null){
        wait = new WebDriverWait(getDriver(), WAIT_IN_SEC);
    }
    return wait;
}

```



```

    public static WebElement waitForElement(WebElement elementToWaitFor) {
return waitForElementToBeVisible(elementToWaitFor, WAIT_IN_SEC);
    }

```

```

    public static WebElement waitForElementToBeVisible(WebElement elementToWaitFor, Integer
waitTimeInSeconds)
    {
        if(waitTimeInSeconds!=null) {
            setWait(waitTimeInSeconds);
        }
        return getWait().until(ExpectedConditions.visibilityOf(elementToWaitFor));
    }
}

```

BaseUtility

```

package org.spicethetest.base;
import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;

```

```

public class BaseUtility {
    private Properties propObj;

```

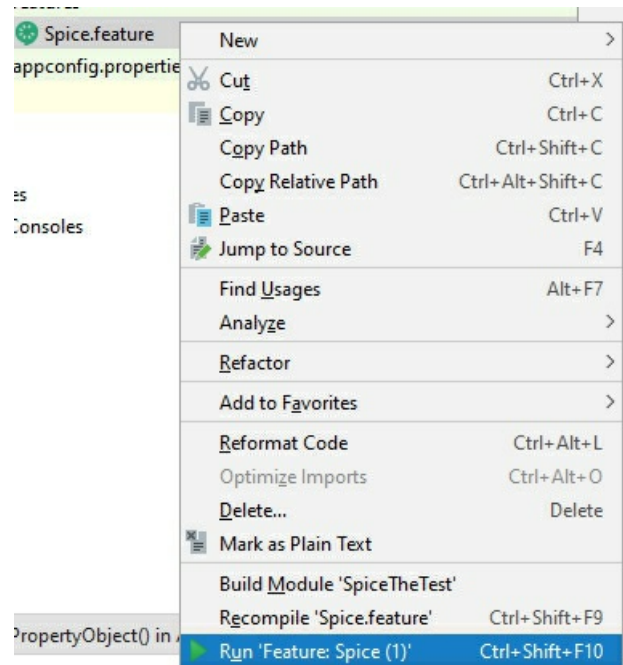
```

    public Properties getPropertyObject()
    {
        if(propObj != null){
            return propObj;
        }

        try {
            propObj = new Properties();
            InputStream input =
getClass().getClassLoader().getResourceAsStream("appconfig.properties");
            propObj.load(input);
        }catch(IOException io) {
            io.printStackTrace();
        }
        return propObj;
    }
}

```

So that is the complete project, time to run and see what we get.



Providing there are no errors the test run will begin.

```
Testing started at 12:56 ...
"C:\Program Files\Java\jdk1.8.0_201\bin\java.exe" -Dorg.jetbrains.run.directory=C:\Dev\SpiceTheTest\src\test\resources\features
Starting ChromeDriver 89.0.4389.23 (61b08ee2c50024bab004e48d2b1b083cdbc579-refs/branch-heads/4389@{#294}) on port 3108
Only local connections are allowed.
Please see https://chromedriver.chromium.org/security-considerations for suggestions on keeping ChromeDriver safe.
ChromeDriver was started successfully.
[1619438218.382][WARNING]: This version of ChromeDriver has not been tested with Chrome version 90.
Apr 26, 2021 12:56:58 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: W3C
Before scenario cucumber.runtime.ScenarioImpl@49cb9cb5Load page https://spicetheworld.com/
```

When complete we will be presented with the results.

✓ Test Results	3 m 18 s 126 ms
✓ Feature: SpiceFeature	3 m 18 s 126 ms
> ✓ Scenario: 01 Start the test run	5 s 770 ms
> ✓ Scenario: 02 Navigate to the search option and click the Search button	1 m 34 s 697 ms
> ✓ Scenario Outline: 03 Navigate Recipes using the site menu system	23 s 686 ms
> ✓ Scenario Outline: 09 Navigate Informaton using the site menu system	29 s 261 ms
> ✓ Scenario: 17 View a vegan recipe	4 s 948 ms
> ✓ Scenario: 18 Print the selected recipe	5 s 867 ms
> ✓ Scenario: 19 Register a new account	5 s 461 ms
> ✓ Scenario: 20 Login and add a recipe to your folder	6 s 87 ms
> ✓ Scenario: 21 Login and add and remove recipe to your folder	8 s 131 ms
> ✓ Scenario: 22 Login and add a recipe to your menu	6 s 861 ms
✓ Scenario: 23 Login and add and remove recipe to your menu	7 s 357 ms
✓ Given we login as the test user	4 s 16 ms
✓ When we add a recipe to menu	1 s 624 ms
✓ Then the recipe is visible in the menu page	531 ms
✓ And then empty the folder	1 s 186 ms

So that is it a partial working example of my Java test suite that I have used on www.spicetheworld.com. Just like the C# and Ruby versions the complete suite also handles backend maintenance, so I was unable to share this with you but hopefully what has been shown is helpful.

Automated API testing with C#

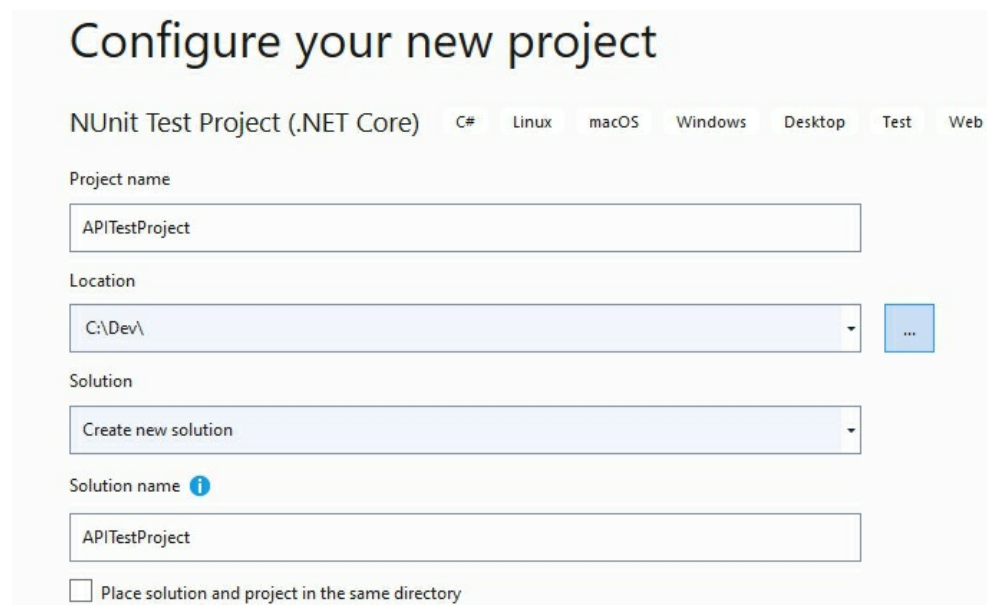
Not only are you able to test API functionality in Postman and Jmeter but you can also test them within C# and Java. In C# this can be achieved with the client library known as RestSharp and that is what we will look at next.

For those that do not know RestSharp is an easy to use, comprehensive, open-source HTTP client library that works with DotNet/C#. It can be used to build robust applications by making it easy to interface with public APIs and quickly access data without the complexity of dealing with raw HTTP requests.

So let us see in action. To do this you need to create a new Nunit Test Project

in Visual Studio.

What you name this project is up to you.



Configure your new project

NUnit Test Project (.NET Core) C# Linux macOS Windows Desktop Test Web

Project name

APITestProject

Location

C:\Dev\

Solution

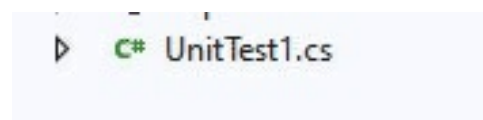
Create new solution

Solution name ⓘ

APITestProject

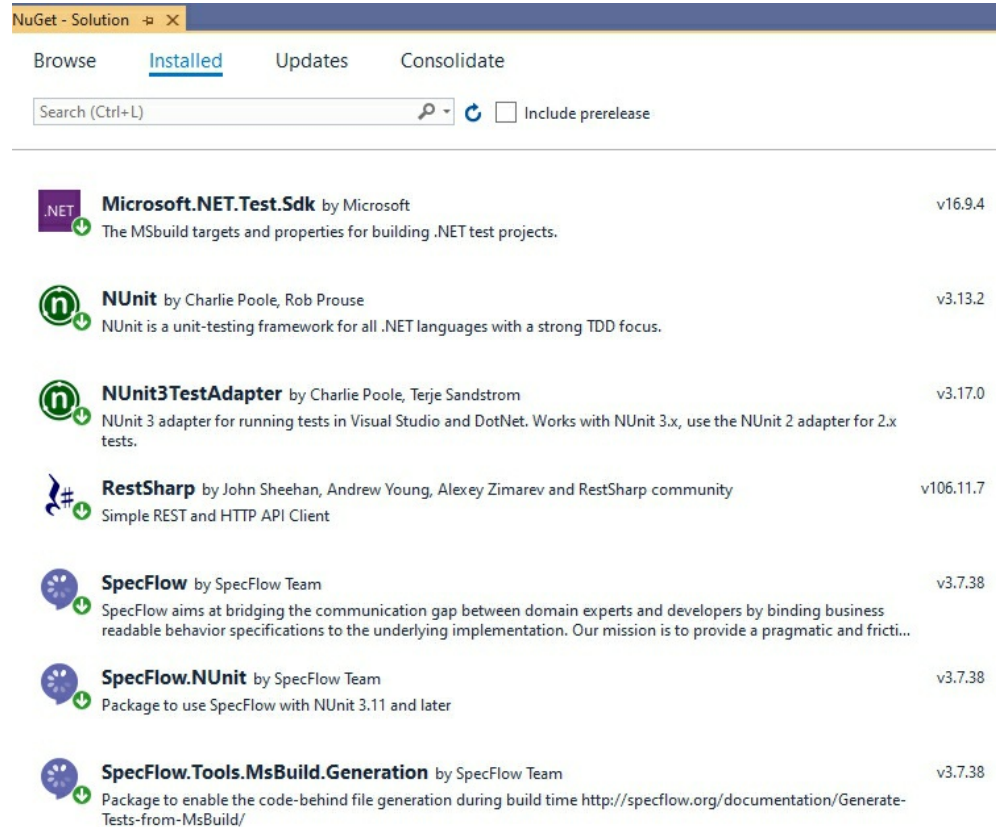
☐ Place solution and project in the same directory

When created you can happily delete this file, it is not required.

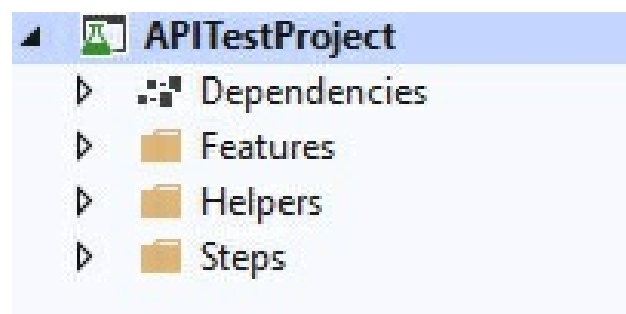


Now add these packages if they are not installed. The actual version may be

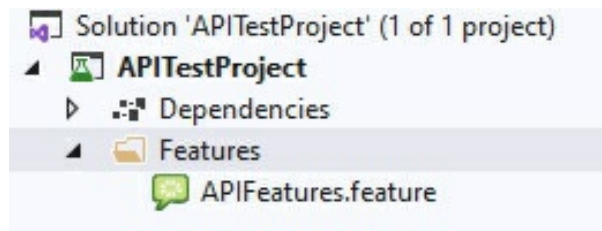
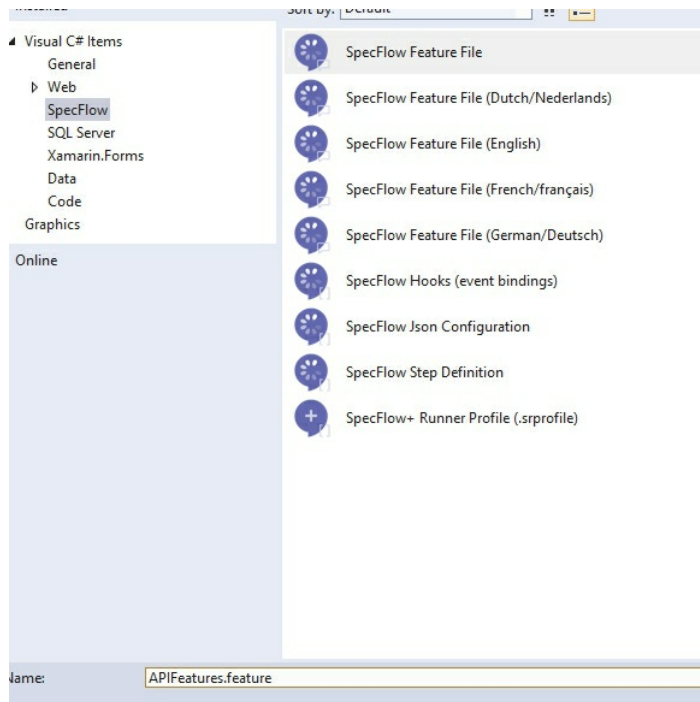
different when you do this, if so go for the latest.



Next create these folders.



In the Features folder create your feature file.



Now add this code below.

@TestApi

Feature: ApiRestSharp

Sample restsharp runs

Scenario: Create Trello Board

Given we create a trello board

Then the board will exist

Scenario: Create Trello Todo list

Given we create a trello board

When we then create the todo list

Then the list will exist

Scenario: Delete all Trello boards

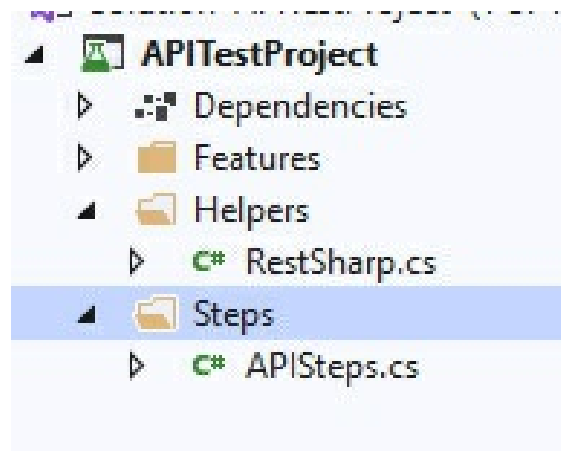
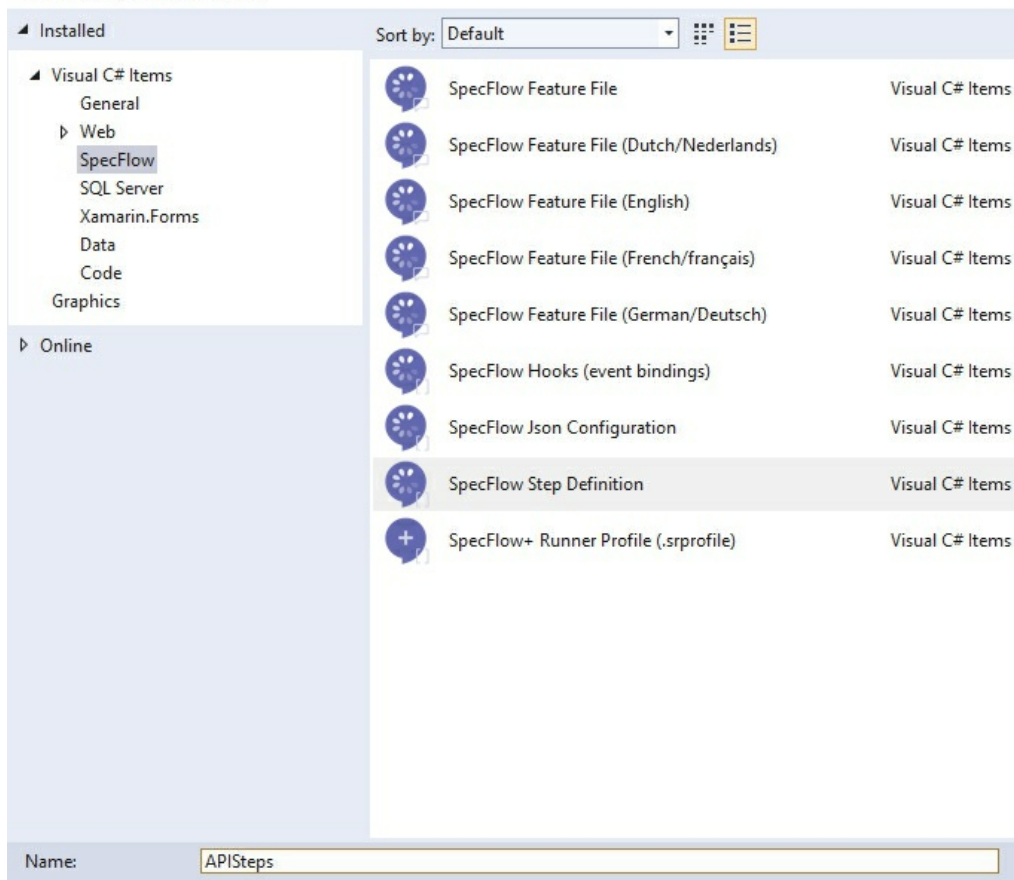
Given we have a list of trello boards

When we have an array of board ids

Then we can delete the boards

Now switch to the Steps folder and insert a steps class.

Add New Item - APITestProject



Replace the code within the APISteps file with this code.

```
using APITestProject.Helpers;  
using TechTalk.SpecFlow;
```

```

namespace APITestProject.Steps
{
    [Binding]
    public sealed class APISteps
    {
        private readonly RestSharpHelper rSharp = new RestSharpHelper();
        private readonly AssertFunctions assertFunctions = new AssertFunctions();
        private readonly ScenarioContext _scenarioContext;

        public APISteps(ScenarioContext scenarioContext)
        {
            _scenarioContext = scenarioContext;
        }

        #region Given
        [Given(@"we have a list of trello boards")]
        public void GivenWeHaveAListOfTrelloBoards()
        {
            rSharp.CreateTrelloBoardList();
        }

        [Given(@"we create a trello board")]
        public void GivenWeCreateATrelloBoard()
        {
            rSharp.CreateTrello();
            rSharp.SaveId();
        }
        #endregion

        #region when
        [When(@"we have an array of board ids")]
        public void WhenWeHaveAnArrayOfBoardIds()
        {
            rSharp.BuildIdArray();
        }

        [When(@"we then create the todo list")]
        public void WhenWeThenCreateTheTodoList()
        {
            rSharp.CreateTodo();
        }
        #endregion

        #region Then
        [Then(@"we will get back a valid list")]
        [Then(@"the board will exist")]
        public void ThenTheBoardWillExist()
        {
            assertFunctions.AssertIsTrue(rSharp.restResponse.IsSuccessful);
            assertFunctions.AssertIsTrue(rSharp.restResponse.StatusCode.ToString().Equals("OK"));
            assertFunctions.AssertIsTrue(rSharp.restResponse.Content.ToString().Contains("\closed\

```



```

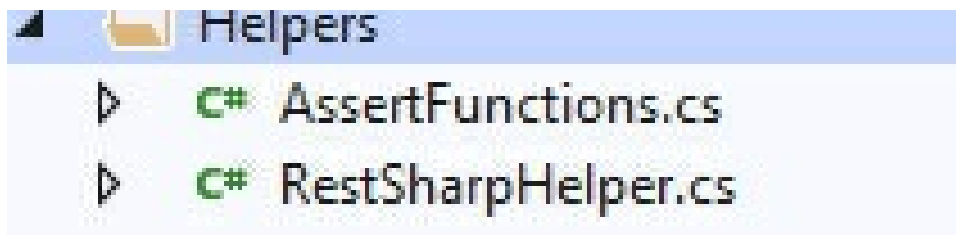
    }

    [Then(@"the list will exist")]
    public void ThenTheListWillExist()
    {
        assertFunctions.AssertIsTrue(rSharp.restResponse.IsSuccessful);
        assertFunctions.AssertIsTrue(rSharp.restResponse.StatusCode.ToString().Equals("OK"));
        assertFunctions.AssertIsTrue(rSharp.restResponse.Content.ToString().Contains("\closed\
    }

    [Then(@"we can delete the boards")]
    public void ThenWeCanDeleteTheBoards()
    {
        assertFunctions.AssertIsTrue(rSharp.DeleteTrelloBoards());
    }
    #endregion
}}

```

Now move to the Helpers folder, in this location we need 2 classes as below.



First we have the AssertFunctions class

```

using System;
using NUnit.Framework;

namespace APITestProject.Helpers
{
    class AssertFunctions
    {
        /// <summary>
        /// AssertIsTrue
        ///
        /// </summary>
        /// <param name="by"></param>
        public void AssertIsTrue(bool by)
        {
            try

```

```

        {
            Assert.That(by);
            Console.WriteLine("Pass: condition is true");
        }
        catch (Exception ex)
        {
            // TakeScreenshot();
            Console.WriteLine("Fail error: " + ex);
            Assert.That(false);
        }
    }
}

```

Then we have the RestSharpHelper class

```

using System.Collections.Generic;
using Newtonsoft.Json;
using RestSharp;

namespace APITestProject.Helpers
{
    class RestSharpHelper
    {
        public IRestResponse restResponse = null;
        public static string boardId = "";
        public List<string> boardIds = new List<string>();

        public void SaveId()
        {
            dynamic newBoard = JsonConvert.DeserializeObject(restResponse.Content);
            foreach (var board in newBoard)
            {
                if (board.Name == "id")
                {
                    boardId = board.First;
                    break;
                }
            }
        }

        public bool DeleteTrelloBoards()
        {
            bool IsSuccessful = true;
            foreach (string id in boardIds)
            {
                IsSuccessful = DeleteTheTrelloBoard(id);
            }
            return IsSuccessful;
        }

        private bool DeleteTheTrelloBoard(string id)
        {

```

```

        RestClient restClient = new RestClient("https://api.trello.com/1/boards/" + id);
        RestRequest restRequest = new RestRequest(Method.DELETE);
        restRequest.AddParameter("token", "ServerTokenSecretServerToken");
        restRequest.AddParameter("key", "ServerKeySecretServerKey");
        restResponse = restClient.Execute(restRequest);
        return restResponse.IsSuccessful;
    }

    public void BuildIdArray()
    {
        dynamic boardArray = JsonConvert.DeserializeObject(restResponse.Content);
        foreach (var board in boardArray)
        {
            string boardId = board.id.ToString();
            boardId = boardId.Replace("{", "").Replace("}", "");
            boardIds.Add(boardId);
        }
    }

    public void CreateTrelloBoardList()
    {
        RestClient restClient = new RestClient("https://api.trello.com/1/organizations/");
        RestRequest restRequest = new RestRequest("/keysbox/boards", Method.GET);
        restRequest.AddParameter("token", "ServerTokenSecretServerToken");
        restRequest.AddParameter("key", "ServerKeySecretServerKey");
        restResponse = restClient.Execute(restRequest);
    }

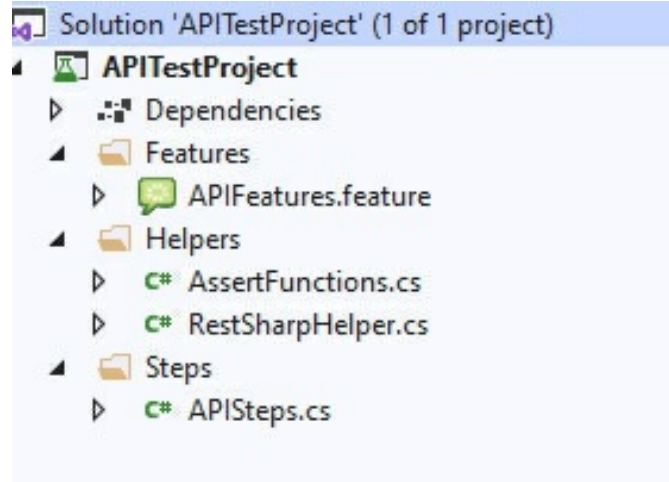
    public void CreateTodo()
    {
        RestClient restClient = new RestClient("https://api.trello.com/1/lists");
        RestRequest restRequest = new RestRequest("/", Method.POST);
        restRequest.AddParameter("name", "Todo List");
        restRequest.AddParameter("key", "ServerKeySecretServerKey");
        restRequest.AddParameter("token", "ServerTokenSecretServerToken");
        restRequest.AddParameter("idBoard", boardId);
        restResponse = restClient.Execute(restRequest);
    }

    public void CreateTrello()
    {
        RestClient restClient = new RestClient("https://api.trello.com/1/boards");
        RestRequest restRequest = new RestRequest("/", Method.POST);
        restRequest.AddParameter("defaultLists", "false");
        restRequest.AddParameter("key", "ServerKeySecretServerKey");
        restRequest.AddParameter("token", "ServerTokenSecretServerToken");
        restRequest.AddParameter("name", "Boardname");
        restResponse = restClient.Execute(restRequest);
    }

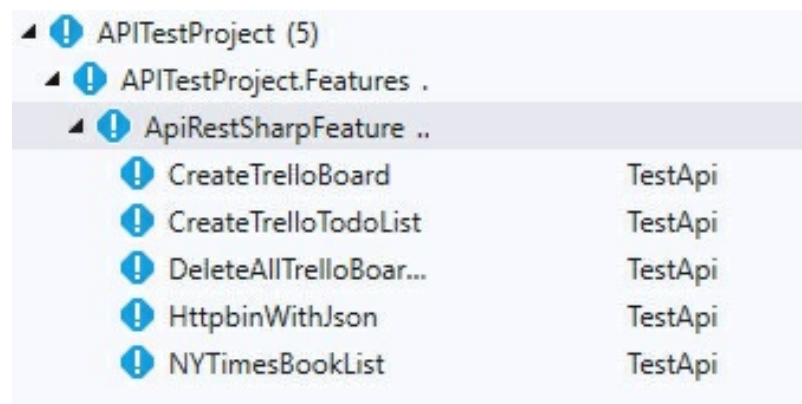
```

```
}  
}  
}
```

So now you should have a project layout like this.



If you build the project you will see this in you test explorer.



Now let us look at each of these tests in turn.

CreateTrelloBoard

To run the Trello tests you will need an account at <https://trello.com>. Once created you will be able to generate your own Server Token.

Server Token

read and write access on your account
read and write access on all your boards
read and write access on all your Workspaces
Approved: 29 Mar at 14:22
Never expires

Revoke

Full details on creating Tokens and keys can be found here:

<https://developer.atlassian.com/cloud/trello/guides/rest-api/api-introduction/>

Once setup the token and key can be used within this API test to create a new Trello board on your account. The assert functions will then confirm the request was good (or not).

CreateTrelloTodoList is similar to the first test expect that this one will create a board and then a todo list which is attached to the board. The assert functions will then confirm the request was good (or not).

DeleteAllTrelloBoards will do just that, all boards on your account will be removed so ensure you are pointing at a test account before running this example. As before the assert functions will then confirm the request was good (or not).

So there you go C#/RestSharp or Java/RestAssured both offer powerful tools for API testing which can be further enhanced with reporting and database features to validate the API has created the correct data. Try these samples out if you wish but remember you will need a Trello account. Next we will look at load and stress testing with Jmeter.

Load Testing with Jmeter

So, what is JMeter you may ask? This is a good question, let me explain a little. JMeter, also known by some as Apache JMeter is a free, open source java-based application with a well-designed and helpful graphical user interface. It is designed to analyse and measure the performance and load functional behaviour of web applications of all sizes and variety of other services.

JMeter is mainly used for testing Web applications and FTP applications. However, it is also applicable in functional testing, database connections, web services to name but a few. Users can perform various testing activities such as stress, performance, load, functional and regression testing.

So why is load and stress testing so important you may think? Code often runs well under light loads in the development environment but can break under heavy production load. Therefore stress and load testing can:

- Expose threading bugs
- Verify code meets performance criteria
- Expose possible application bottlenecks
- Help capacity planning and future dev plans

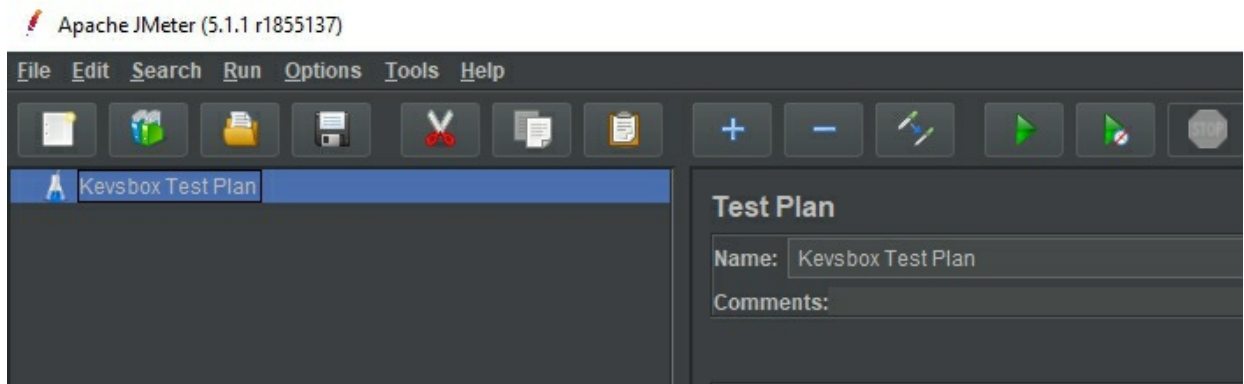
There are several stages of load testing.

- First, verify the application server environment and check that there are no errors in the application.
- Make sure that enough machine resources are allocated for testing and that the configuration settings are correct.
- Warm up the system and application server since “cold” machines might perform poorly because data is not in memory or cache, data structures might need to be created, and so forth.
- When you run the test load, collect as much performance data as possible.
- Finally, verify that the results of the load testing are reliable and repeatable.

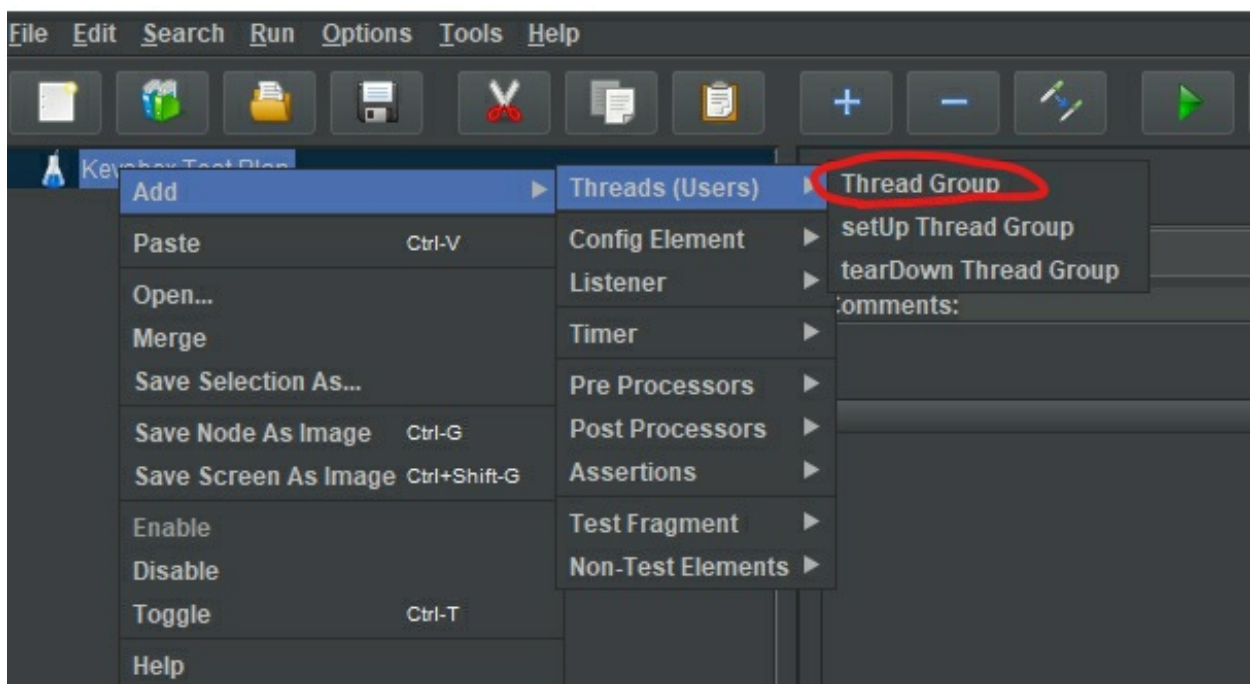
Now let's get going with JMeter. If you have not yet installed JMeter then at the time of writing it can be download from here -

https://jmeter.apache.org/download_jmeter.cgi.

Once installed fire up JMeter and we can get started. Below is a nice, shiny, fresh project, all I have done so far is rename the test plan. The test plan is your JMeter script, it will determine the flow of the load test.



Next, we require a Thread Group. Thread groups determine the user flow and simulate how users behave on the app. Each thread represents a user.



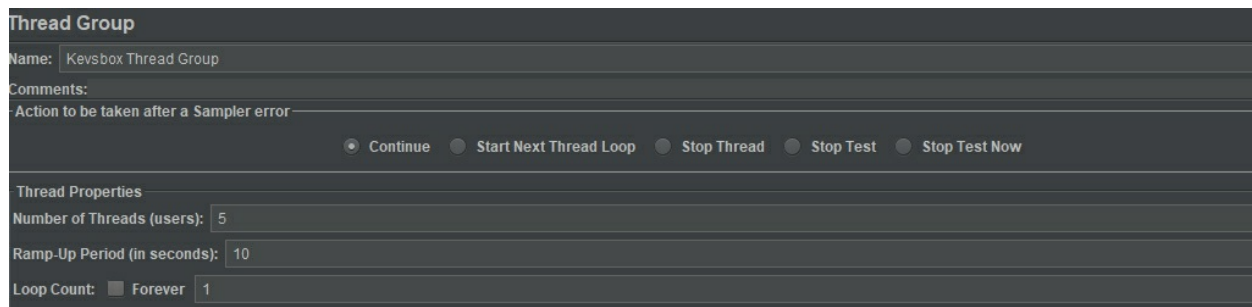
Now you will need to configure the Thread Group by setting the following:

- Name: Provide a custom name or, if you prefer, simply leave it named the default “Thread Group”. In this example I am using “Kevsbox Thread Group”.
- Number of Threads: Each thread will execute the test plan in its entirety and completely independently of other test threads. Multiple threads are used to simulate concurrent connections to

your server application. For this example, 5.

- Set the ramp-up period - The ramp-up period tells JMeter how long to take to "ramp-up" to the full number of threads chosen. If 10 threads are used, and the ramp-up period is 100 seconds, then JMeter will take 100 seconds to get all 10 threads up and running. Each thread will start 10 (100/10) seconds after the previous thread was begun. If there are 30 threads and a ramp-up period of 120 seconds, then each successive thread will be delayed by 4 seconds. For this example, we will use 10.
- Loop Count: How many times the test should repeat. Let's say 1 time (no repeat, just run once).

So now we should see something like this.



The screenshot shows the 'Thread Group' configuration window in JMeter. The 'Name' field is set to 'Keystbox Thread Group'. The 'Comments' field is empty. The 'Action to be taken after a Sampler error' section has five radio buttons: 'Continue' (selected), 'Start Next Thread Loop', 'Stop Thread', 'Stop Test', and 'Stop Test Now'. The 'Thread Properties' section has three fields: 'Number of Threads (users)' set to 5, 'Ramp-Up Period (in seconds)' set to 10, and 'Loop Count' set to 1 with the 'Forever' checkbox unchecked.

JMeter has two types of Controllers: Samplers and Logical Controllers. These drive the processing of a test. Samplers tell JMeter to send requests to a server. For example, add an HTTP Request Sampler if you want JMeter to send an HTTP request.

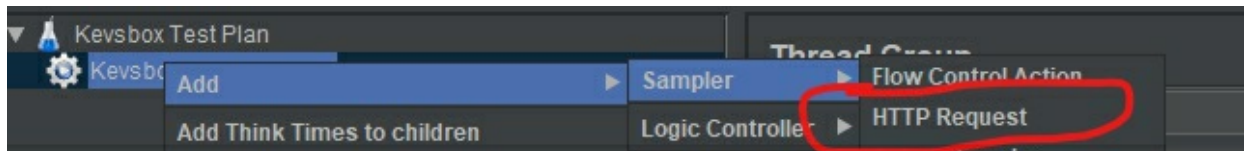
You can also customize a request by adding one or more Configuration Elements to a Sampler. Logical Controllers let you customize the logic that JMeter uses to decide when to send requests. For example, you can add an Interleave Logic Controller to alternate between two HTTP Request Samplers.

Samplers tell JMeter to send requests to a server and wait for a response. They are processed in the order they appear in the tree. Controllers can be used to modify the number of repetitions of a sampler.

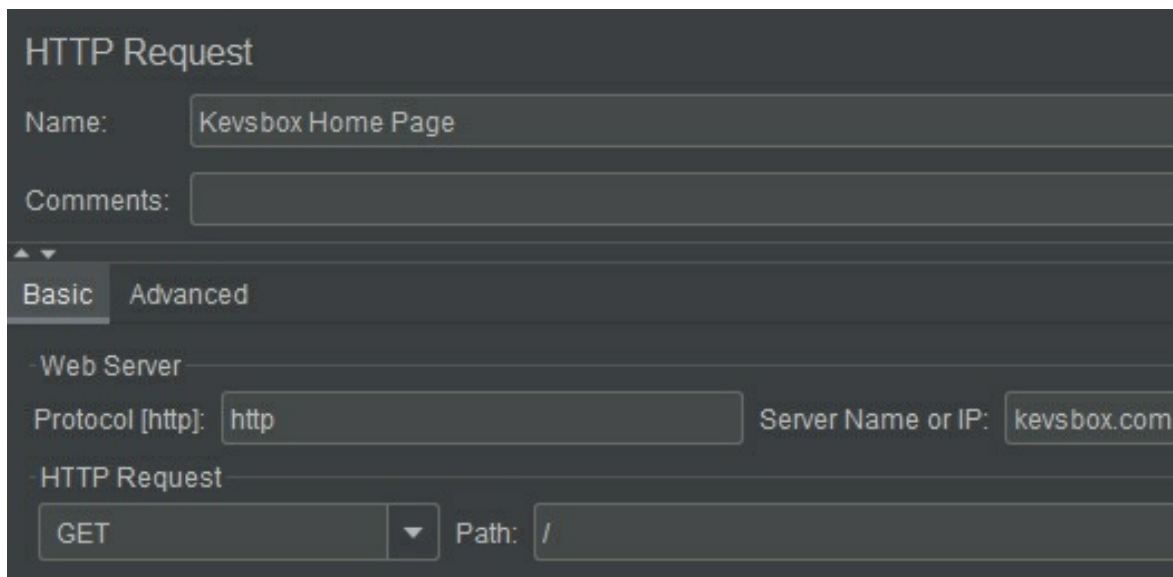
- JMeter samplers include:
 - FTP Request
 - HTTP Request

- JDBC Request
- Java object request
- LDAP Request
- SOAP/XML-RPC Request
- Webservice (SOAP) Request

So next we need to add some HTTP request samplers, we do after all, need to load some web pages. Under the Thread Group add a HTTP Request.



When added complete the request to look something like this

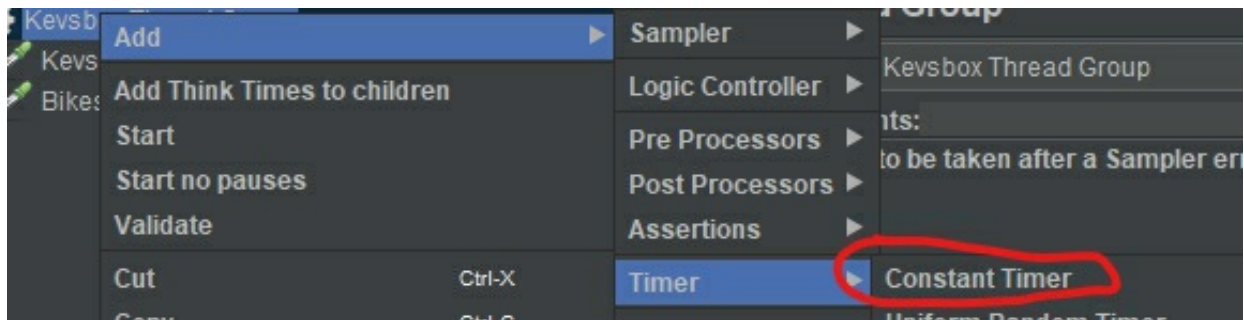
A screenshot of the 'HTTP Request' configuration dialog in JMeter. The 'Name' field is 'Kevsbox Home Page'. The 'Comments' field is empty. The 'Basic' tab is selected. Under 'Web Server', the 'Protocol [http:]' is 'http' and the 'Server Name or IP:' is 'kevsbox.com'. Under 'HTTP Request', the method is 'GET' and the 'Path:' is '/'. There are expandable sections for 'Web Server' and 'HTTP Request'.

Now repeat this process as below

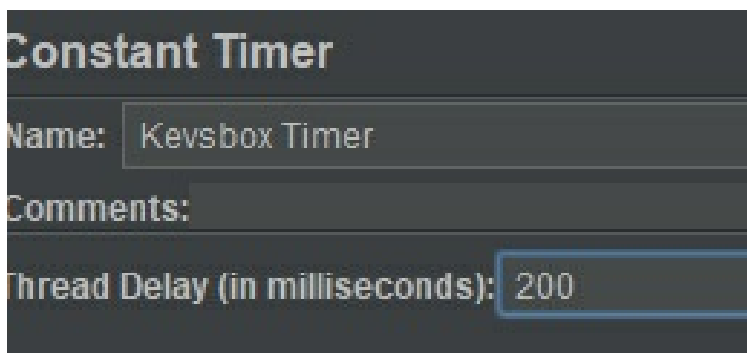


The screenshot shows the 'HTTP Request' configuration window. The 'Name' field is set to 'Kevs Bikes'. The 'Comments' field is empty. The 'Basic' tab is selected. Under the 'Web Server' section, the 'Protocol [http:]' is set to 'https' and the 'Server Name or IP:' is 'kevsbox.com'. Under the 'HTTP Request' section, the method is set to 'GET' and the 'Path:' is '/bikes.html'.

Now we will add a timer to help with the tests. When users click on your website or app, they naturally have pauses and delays. These need to be considered. Constant timers are the most used in JMeter. They simply determine how many milliseconds to wait between requests.



Then set it to wait for 200 milliseconds.

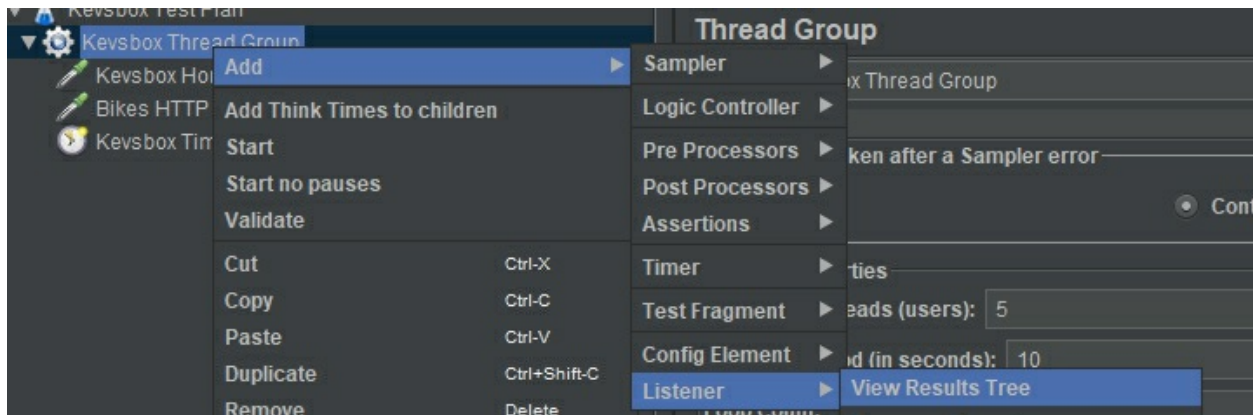


The screenshot shows the 'Constant Timer' configuration window. The 'Name' field is set to 'Kevsbox Timer'. The 'Comments' field is empty. The 'Thread Delay (in milliseconds):' field is set to '200'.

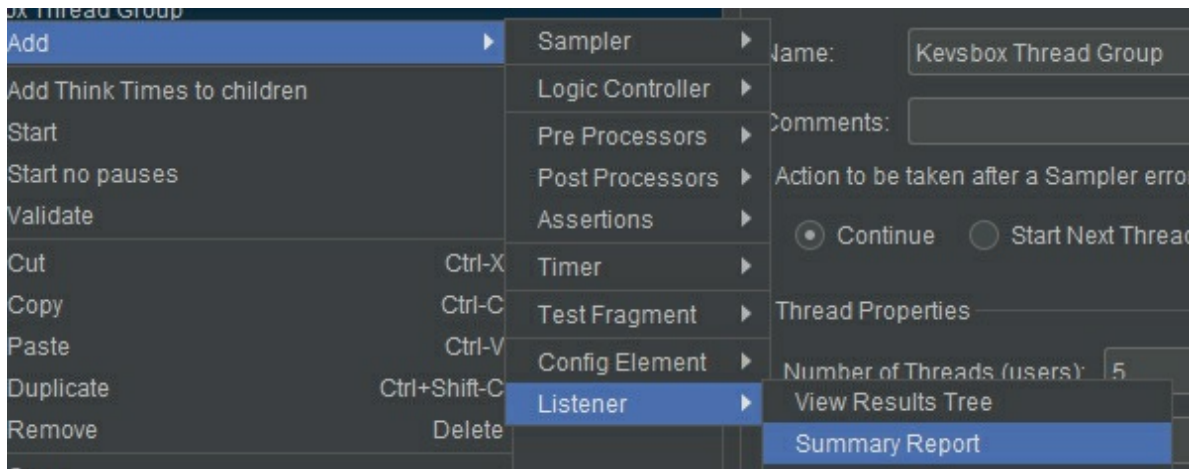
Finally, we need a way of viewing the results after running the test. This is achieved through listeners, a recording mechanism that shows results, including logging and debugging information.

The View Results Tree is the most common Listener and we will use that one

in this example.



Another option is to add a summary report.



This is a typical output for a summary report

Summary Report

Comments:

Write results to file / Read from file

Filename: Browse... Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
Page	4000	3459	431	151496	5619.60	0.65%	8.8/sec	93.29	10888.1
n	4000	2777	275	285637	7482.32	0.40%	8.8/sec	90.50	10490.0
al	4000	3403	420	245369	6756.56	0.27%	8.8/sec	94.88	10994.2
zation UK	4000	3337	367	196370	5676.11	0.20%	8.8/sec	93.99	10889.0
AL	16000	3244	275	285637	6436.85	0.38%	35.0/sec	369.19	10815.3

Label: In the label section you will be able to see all the recorded http request, during test run or after test run.

Samples: Samples denote to the number of http request ran for given thread, or the average time taken to receive the web pages.

For Example we have one http request and we run it with 5 users, than the number of samples will be $5 \times 1 = 5$. So if the sample ran two times for the single user, than the number of samples for 5 users will be $5 \times 2 = 10$.

Average: Average is the average response time for that particular http request. This response time is in millisecond. This an Arithmetic mean for all responses (sum of all times / count)

Min: Min denotes to the minimum response time taken by the http request.

Max: Max denotes to the maximum response time taken by the http request.

Std. Deviation: This shows how many exceptional cases were found which were deviating from the average value of the receiving time. The lesser this value more consistent the time pattern is assumed.

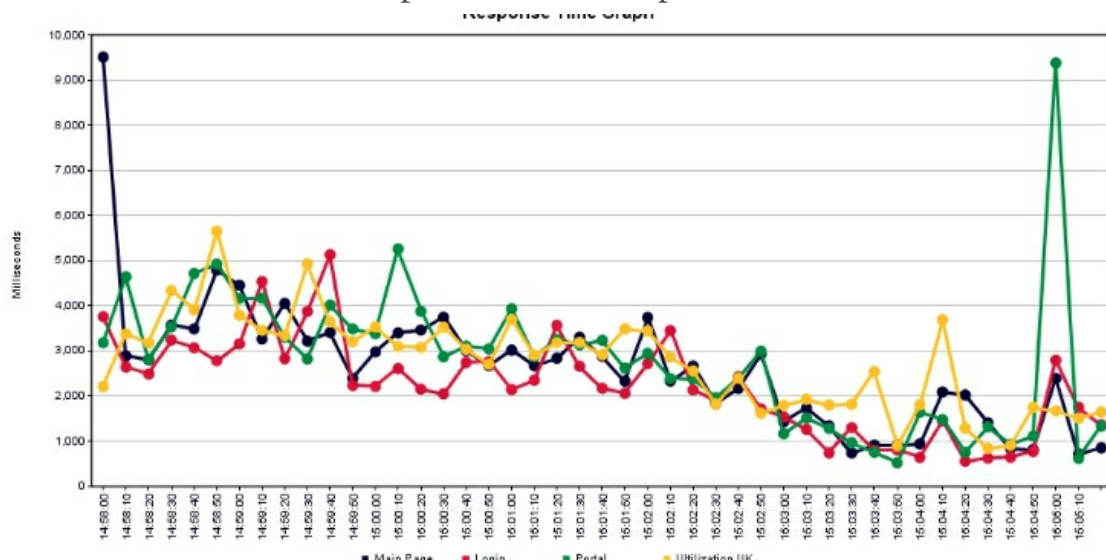
Error %: This denotes the error percentage in samples during run. This error can be of 404(file not found) or may be exception or any kind of error during test run will be shown in Error %. In the above image the error % is zero, because all the requests ran successfully.

Throughput: The throughput is the number of requests per unit of time (seconds, minutes, hours) that are sent to your server during the test. Larger is better.

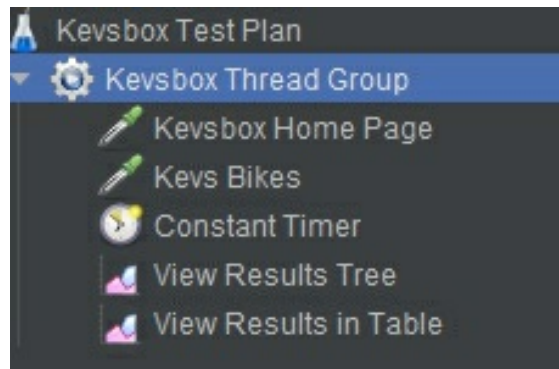
KB/Sec: The throughput measured in Kilobytes per second

Avg. Bytes: Average response size of the sample response in bytes.

And there is also the Response Time Graph

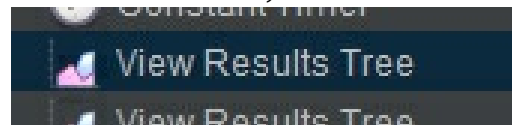


So if you add all of these you will have a project like this

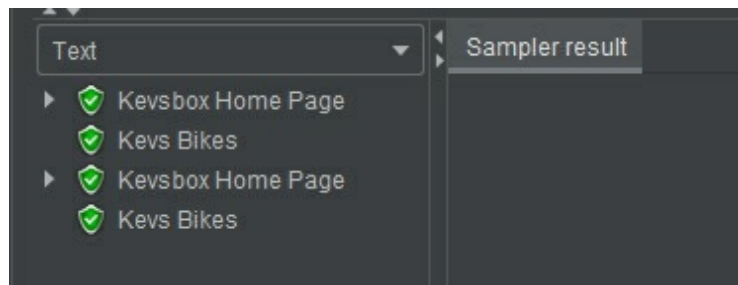


So now we are ready to run this test. So click Save and your test will be saved as a .jmx file.

Now it is time to run the tests, click on View Results Tree.



Now start the test run  and watch the results appear



When complete the Results Tree will look something like this.

Text

- ▶ Kevsbox Home Page
- Kevs Bikes
- ▶ Kevsbox Home Page
- Kevs Bikes
- ▶ Kevsbox Home Page
- Kevs Bikes
- ▶ Kevsbox Home Page
- Kevs Bikes
- ▶ Kevsbox Home Page
- Kevs Bikes

Sampler result Request Response data

Thread Name:Kevsbox Thread Group 1-1
Sample Start:2021-04-22 15:07:05 BST
Load time:443
Connect Time:47
Latency:83
Size in bytes:13844
Sent bytes:226
Headers size in bytes:525
Body size in bytes:13319
Sample Count:1
Error Count:0
Data type ("text"|"bin"|""):text
Response code:200
Response message:OK

HTTPSampleResult fields:
ContentType: text/html
DataEncoding: null

The View Results in Table

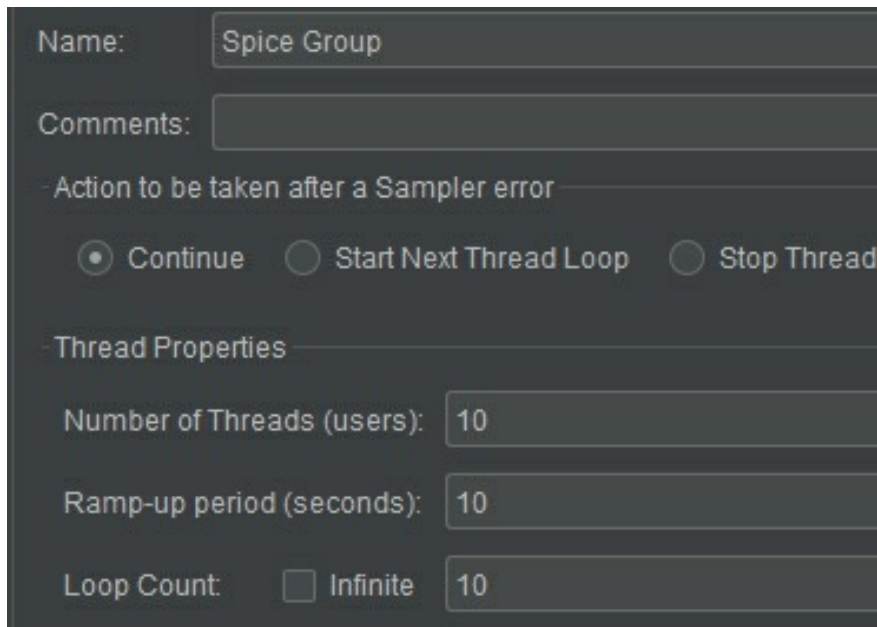
Sample #	Start Time	Thread Name	Label	Sample Time(ms)
1	15:07:05.921	Kevsbox Thread Group...	Kevsbox Home Page	443
2	15:07:06.585	Kevsbox Thread Group...	Kevs Bikes	45
3	15:07:07.849	Kevsbox Thread Group...	Kevsbox Home Page	202
4	15:07:08.266	Kevsbox Thread Group...	Kevs Bikes	39
5	15:07:09.861	Kevsbox Thread Group...	Kevsbox Home Page	175
6	15:07:10.237	Kevsbox Thread Group...	Kevs Bikes	32
7	15:07:11.871	Kevsbox Thread Group...	Kevsbox Home Page	168
8	15:07:12.241	Kevsbox Thread Group...	Kevs Bikes	31
9	15:07:13.849	Kevsbox Thread Group...	Kevsbox Home Page	192
10	15:07:14.245	Kevsbox Thread Group...	Kevs Bikes	30

Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
	13844	226	83	47
	20492	123	43	0
	13844	226	58	27
	20492	123	38	0
	13844	226	55	23
	20492	123	31	0
	13844	226	52	22
	20492	123	30	0
	13844	226	54	24
	20492	123	29	0

So that is a simple run but there is a lot more that you can do with JMeter and while this book is not an advanced training solution for this topic we will

delve a little further into this interesting area of testing.

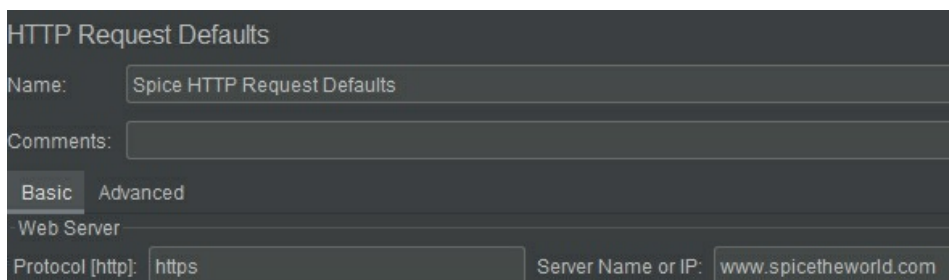
The next test suite offers some more interest scenarios. First we have the Thread Group



The screenshot shows the configuration for a Thread Group named "Spice Group". It includes a "Comments" field, a section for "Action to be taken after a Sampler error" with radio buttons for "Continue" (selected), "Start Next Thread Loop", and "Stop Thread", and a "Thread Properties" section. The "Thread Properties" section contains three fields: "Number of Threads (users)" set to 10, "Ramp-up period (seconds)" set to 10, and "Loop Count" with a checkbox for "Infinite" (unchecked) and a value of 10.

Here we have a 10 second ramp-up and 10 users who will loop 10 times also, so 100 executions at this point.

Next we have a HTTP Request Defaults element, this is where the base URL is held



The screenshot shows the configuration for an HTTP Request Defaults element named "Spice HTTP Request Defaults". It includes a "Comments" field and two tabs: "Basic" (selected) and "Advanced". Under the "Basic" tab, there is a "Web Server" section with two fields: "Protocol [http]" set to "https" and "Server Name or IP" set to "www.spicetheworld.com".

Next we have a loop controller which is set to loop 3 times. So now we have 100 at 3 loops so 300 executions at this point.

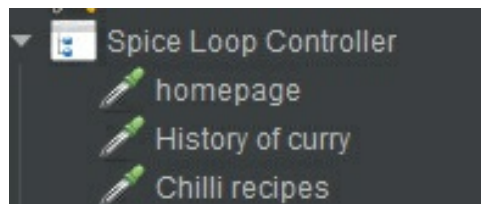
Loop Controller

Name:

Comments:

Loop Count: ☐ Infinite

The next elements must go below the loop controller as child elements.



These are all HTTP requests, add 3 as shown here, you will notice the protocol and Server Name/IP are blank, this are taken from the HTTP Request Defaults Element, all that is required is the Path.

Web Server

Protocol [http]: Server Name or IP:

HTTP Request

GET Path:

Protocol [http]: Server Name or IP:

HTTP Request

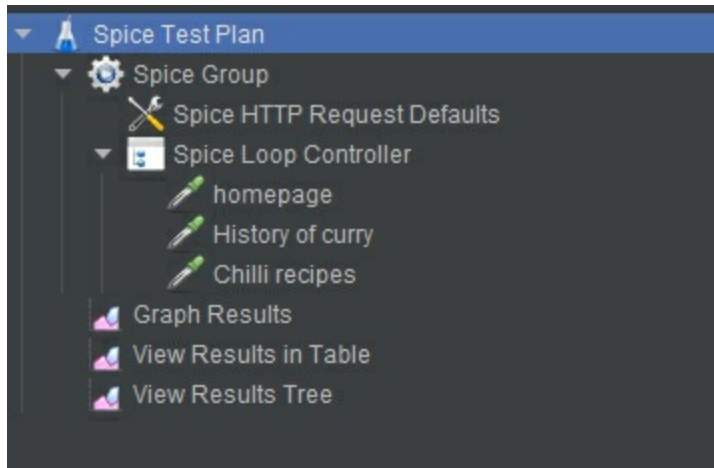
GET Path:

Protocol [http]: Server Name or IP:

HTTP Request

GET Path:

So now we have 3 HTTP requests for each of the 300 current requests giving a total request count of 900 test requests. Now add your reports so you have a layout like this.



Save and run as before.

All done, good let us now look at the results.

No of Samples 900

Number of samples should be 900, and it is

No errors and all loops are complete



The Graph has data



Table output is all green

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status
865	15:37:11.178	Spice Group 1-10	History of curry	28	✓
866	15:37:11.207	Spice Group 1-9	Chilli recipes	29	✓
867	15:37:11.207	Spice Group 1-10	Chilli recipes	29	✓
868	15:37:11.237	Spice Group 1-10	homepage	27	✓
869	15:37:11.264	Spice Group 1-10	History of curry	43	✓
870	15:37:11.307	Spice Group 1-10	Chilli recipes	34	✓
871	15:37:11.341	Spice Group 1-10	homepage	26	✓
872	15:37:11.367	Spice Group 1-10	History of curry	29	✓
873	15:37:11.396	Spice Group 1-10	Chilli recipes	31	✓
874	15:37:11.427	Spice Group 1-10	homepage	30	✓
875	15:37:11.458	Spice Group 1-10	History of curry	29	✓
876	15:37:11.487	Spice Group 1-10	Chilli recipes	30	✓
877	15:37:11.517	Spice Group 1-10	homepage	34	✓
878	15:37:11.551	Spice Group 1-10	History of curry	27	✓
879	15:37:11.578	Spice Group 1-10	Chilli recipes	28	✓
880	15:37:11.606	Spice Group 1-10	homepage	28	✓
881	15:37:11.634	Spice Group 1-10	History of curry	31	✓

And the tree output is also good

Text

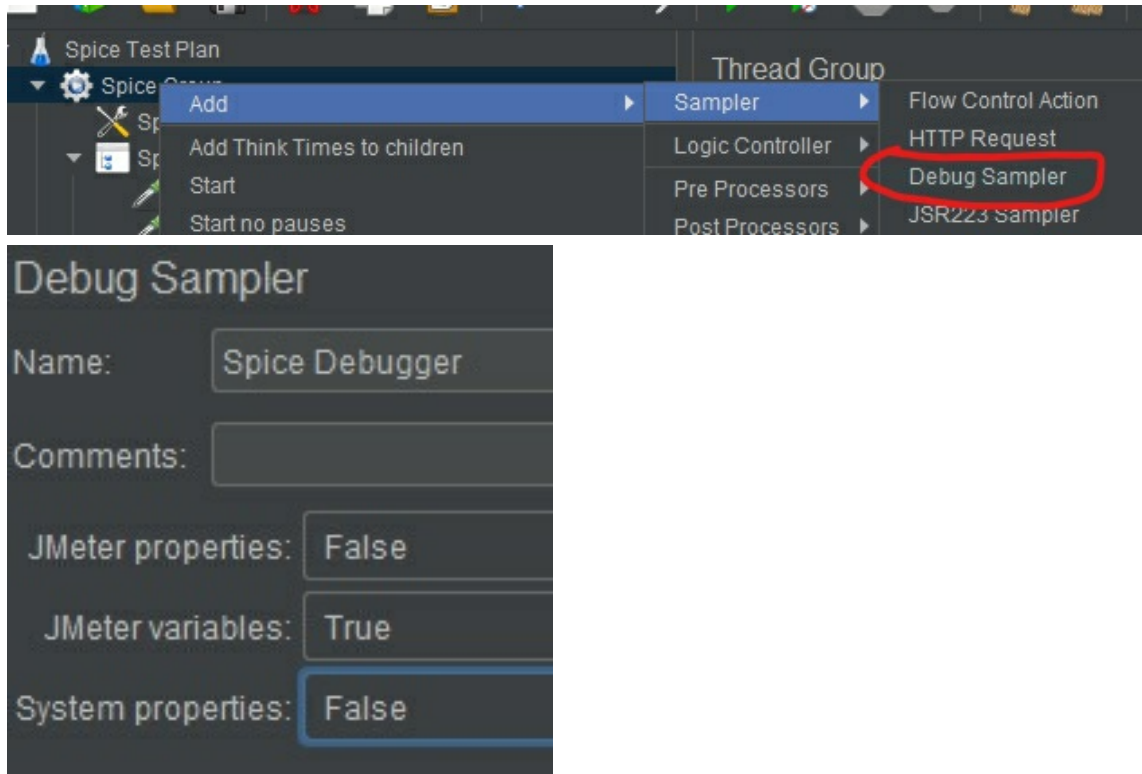
- ✓ homepage
- ✓ Chilli recipes
- ✓ History of curry
- ✓ homepage
- ✓ Chilli recipes
- ✓ History of curry
- ✓ homepage
- ✓ Chilli recipes
- ✓ History of curry
- ✓ Chilli recipes
- ✓ homepage
- ✓ History of curry
- ✓ homepage
- ✓ Chilli recipes
- ✓ History of curry
- ✓ homepage
- ✓ Chilli recipes


Sampler resultRequestResponse data

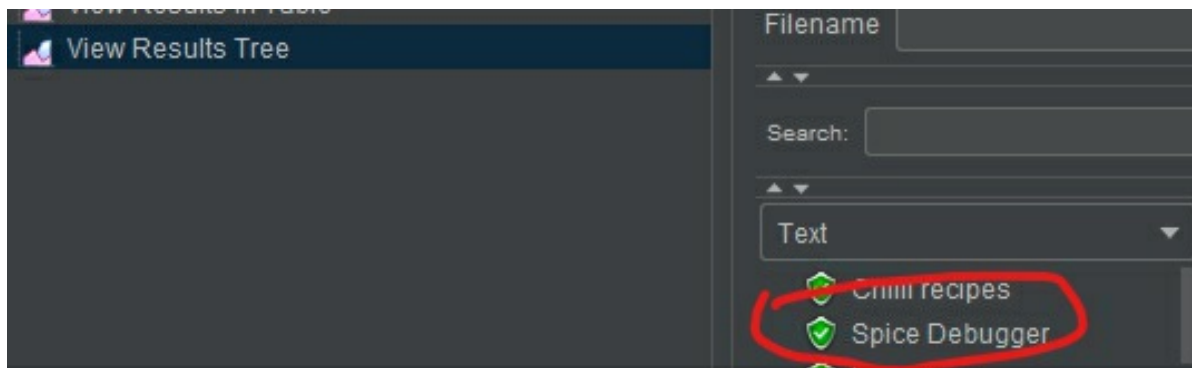
Thread Name:Spice Group 1-4
Sample Start:2021-04-22 15:37:05 BST
Load time:35
Connect Time:0
Latency:35
Size in bytes:18332
Sent bytes:123
Headers size in bytes:571
Body size in bytes:17761
Sample Count:1
Error Count:0
Data type ("text"|"bin"|""):text
Response code:200
Response message:OK

HTTPSsampleResult fields:
ContentType: text/html; charset=utf-8
DataEncoding: utf-8

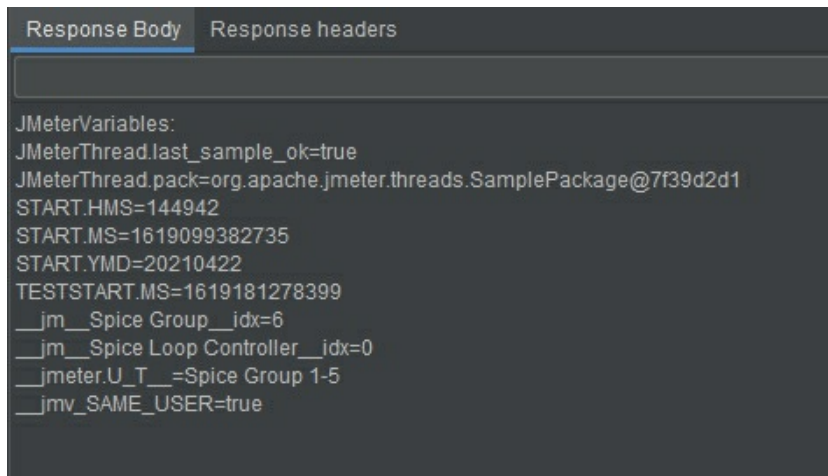
Okay so now we are getting the hang of this let us look a bit further.
Next we will add a Debug Sampler.



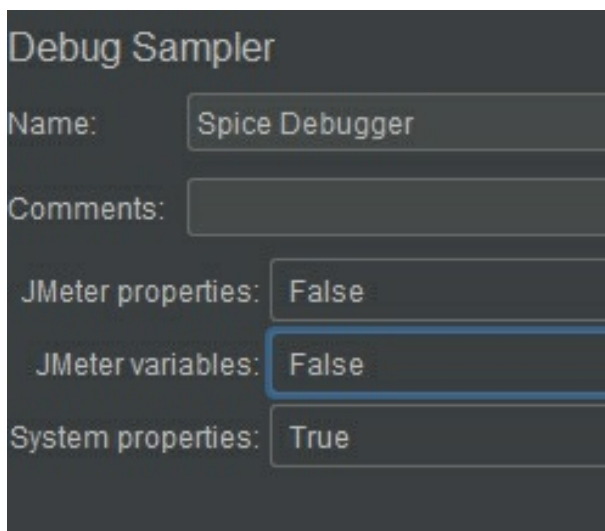
Click  here to remove all previous reporting data and save your project. Next re-run the project and wait for the completion. Now look down the results in the View Results Tree, so should see some Debugger outputs.



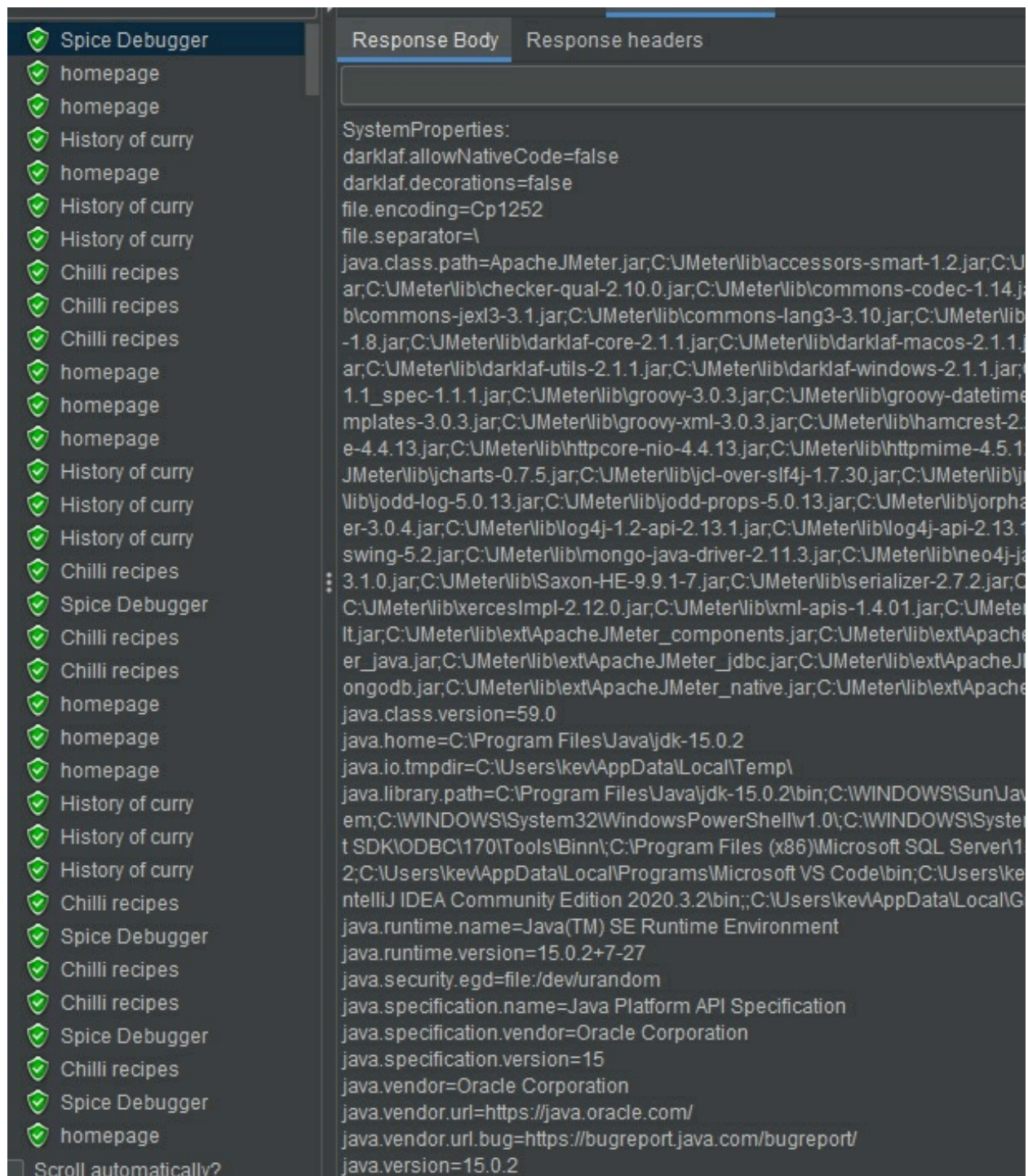
So now click on one of these and look at the output where you will see the JMeter variables.



Now let's look at the System properties.



Make these changes, Save, Reset and Re-run.



This time you will see the system properties. Now if we go the the Test Plan and add 2 variables thus.



Test	5
Test	4
Name:	

Debug Sampler

Name:

Comments:

JMeter properties:

JMeter variables:

System properties:

Make these changes, Save, Reset and Re-run.

Response Body	Response headers
---------------	------------------

```

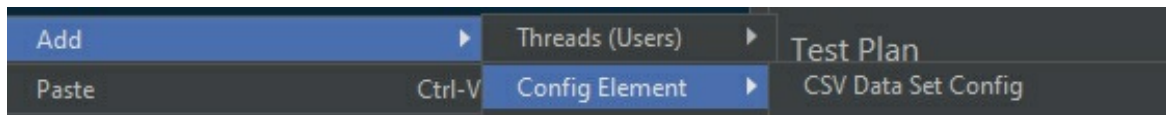
JMeterVariables:
JMeterThread.last_sample_ok=true
JMeterThread.pack=org.apache.jmeter.threads.SamplePackage@7b324ace
START.HMS=144942
START.MS=1619099382735
START.YMD=20210422
TESTSTART.MS=1619181919449
Test1=1
Test2=2
__jm__Spice Group__idx=0
__jm__Spice Loop Controller__idx=0
__jmeter.U_T__=Spice Group 1-1
__jmv_SAME_USER=true
  
```

You will now also see the two variables that you have created.

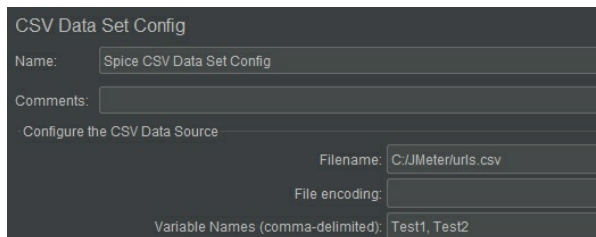
Let us now take this a step further, here we have a CSV file called urls.csv and as you can see it has 2 lines of data.

SAEXT.DLL	xray.txt	urls.csv
urls	1	www.spicetheworld.com
	2	www.kevsbox.com

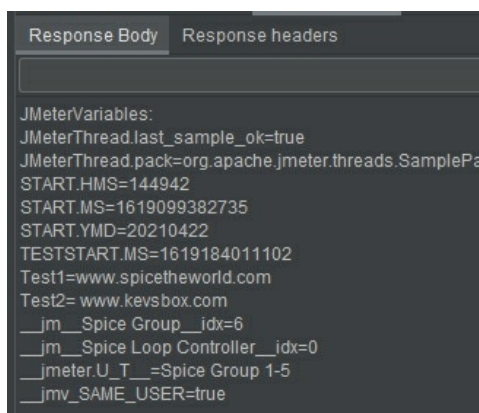
So let's now add a CSV Data Set Config element to the project.



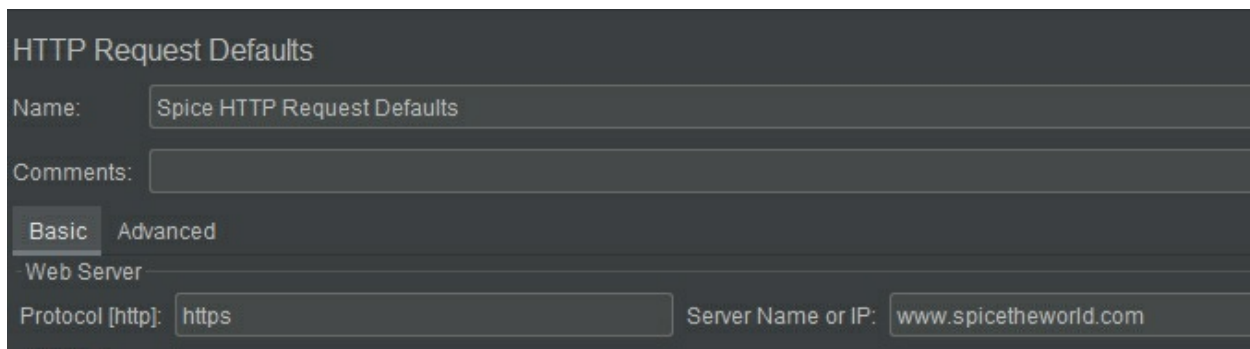
Now we need to add the Filename and assign the 2 lines to variables test1 and test2



Once again make these changes, Save, Reset and Re-run. This time you will see the values from the CSV have been added and assigned to the 2 variables.



This is how you add external data to a JMeter project. You may now be thinking what is the point? Well the reason is we can use this data as part of the test run. In this example we read in 2 URL's the first of which is the base URL we are testing against, so let us make use of it now. So go back to the HTTP Request Defaults panel.



Reset the server name to use variable Test1.

The screenshot shows the 'HTTP Request Defaults' configuration window. The 'Name' field is set to 'Spice HTTP Request Defaults'. The 'Comments' field is empty. The 'Basic' tab is selected, showing the 'Web Server' section. The 'Protocol [http]' is set to 'https' and the 'Server Name or IP' is set to '\${Test1}'.

Once again make these changes, Save, Reset and Re-run. This time the tests will run exactly the same as previous runs. However if there was a problem during a test run you can now view any or all of these sets and inspect the values which may well help you determine why the tests failed.

The screenshot shows three settings, all set to 'True': 'JMeter properties:', 'JMeter variables:', and 'System properties:'.

The next cool feature of JMeter we are going to investigate is the ability to test API function calls. For the purposes of the demonstration will we be using a local instance of JIRA.

JIRA is a well-known and popular bug tracking and test management offering. JIRA is a completely REST API based tool and there is an API URI for each published functionality.

First, unless you have previously done so, you will need to download JIRA. At the time of writing the download URL was -

<https://www.atlassian.com/software/jira/download>.

Select the version for your computer and download.

Long Term Support release

Jira Software 8.13.6

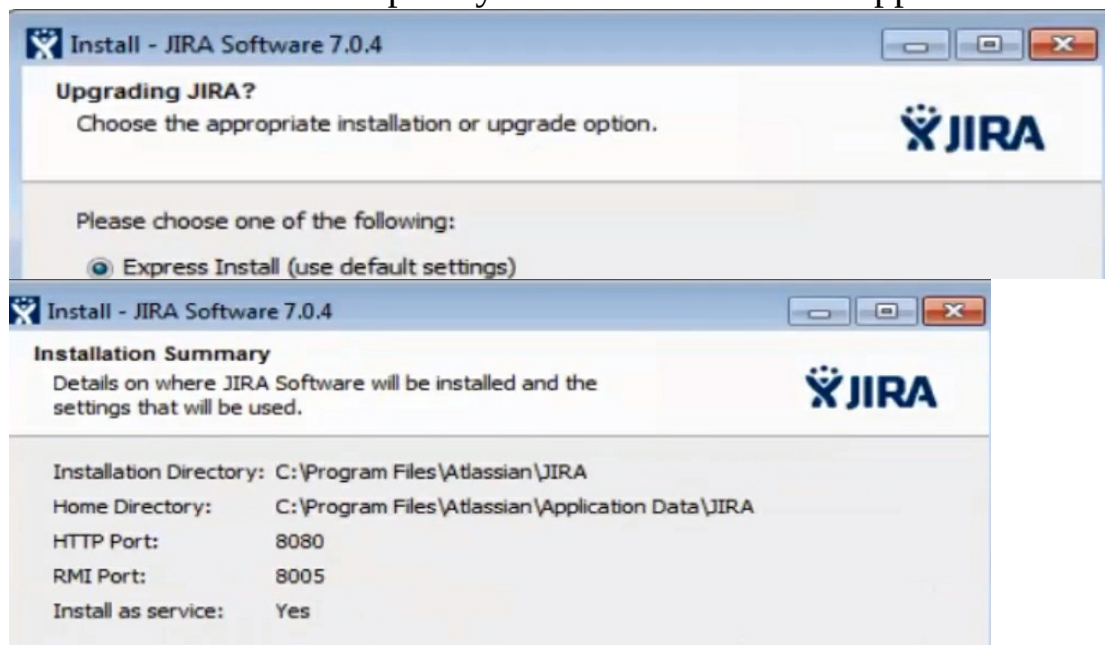
Recommended for large organizations looking for long-term support.

[Why choose Long Term Support releases?](#)

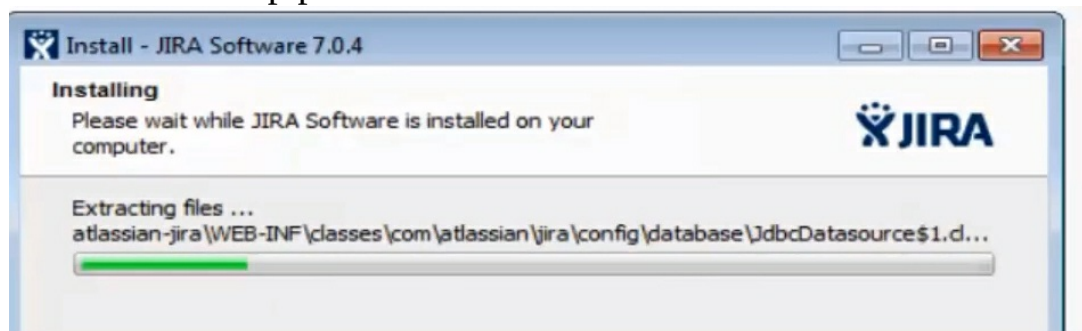
Windows 64 Bit

Download

When download is complete you can then install the application.



Note that the http port is 8080.

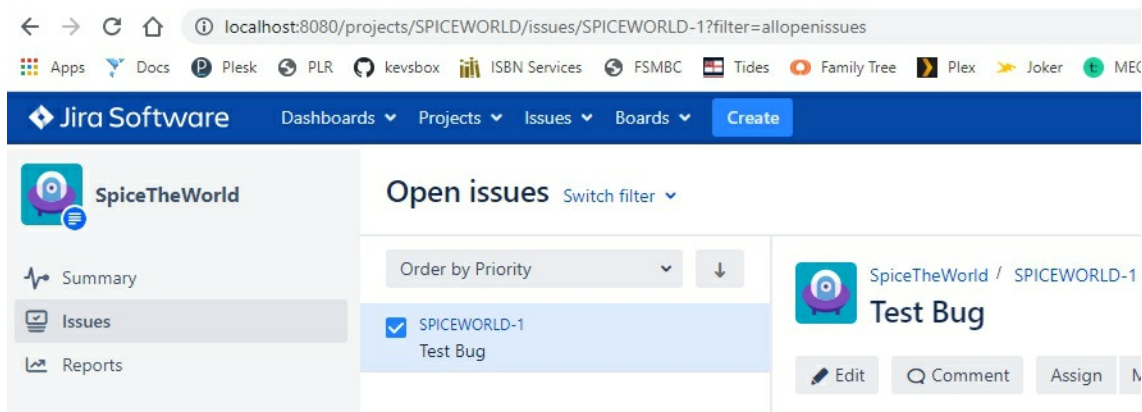


When complete the JIRA service will start.



Also you can launch the application in your web browser at <http://localhost:8080/> . Now the actual setting up of JIRA is beyond the scope of this discussion but once JIRA is up and running you should setup a task management project and you should create a user, then you will be ready



to continue.



Users and roles

Project lead  kev@kevsbox.com Default Assignee Unassigned

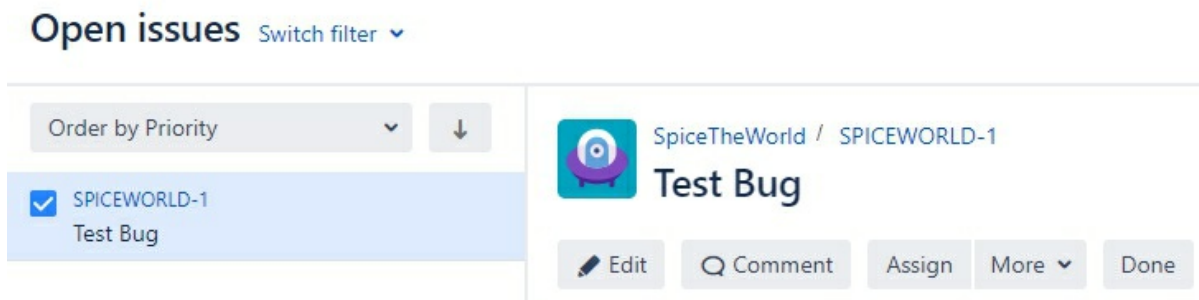
Search  Roles 

Name	Email address	Roles
 jira-administrators jira-administrators	--	Administrators
 kev@kevsbox.com kev	kev@kevsbox.com	Administrators

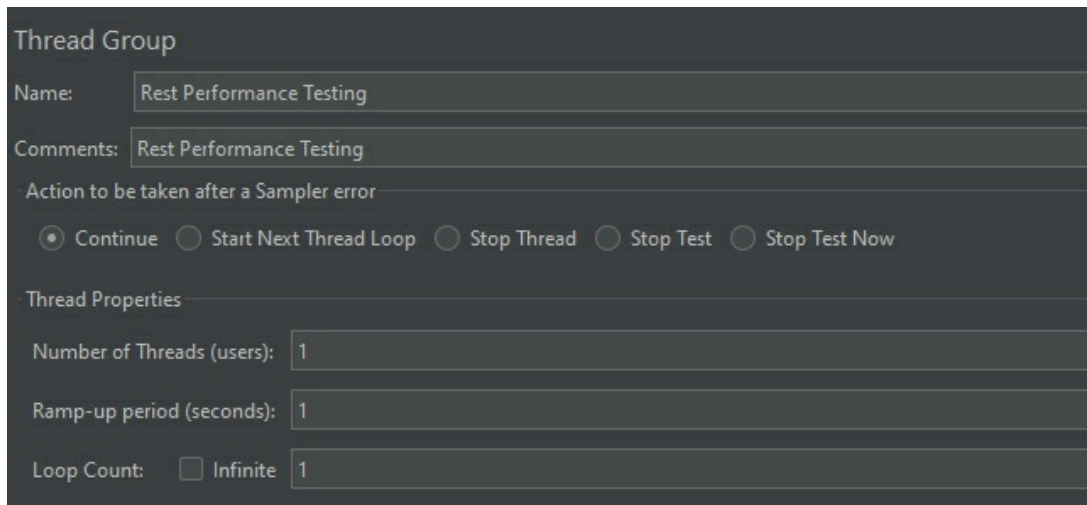
Also the full list of JIRA API's available can be seen here:

<https://docs.atlassian.com/software/jira/docs/api/REST/6.1.7/>

Okay let us try a GET call on the local JIRA service. You will need a ticket in your test project, I have an example in my project as shown below.



I can use a GET call on this ticket in JMeter and retrieve the data, so start a new project in JMeter then create the Thread Group.



Thread Group

Name: Rest Performance Testing

Comments: Rest Performance Testing

Action to be taken after a Sampler error

☒ Continue ☐ Start Next Thread Loop ☐ Stop Thread ☐ Stop Test ☐ Stop Test Now

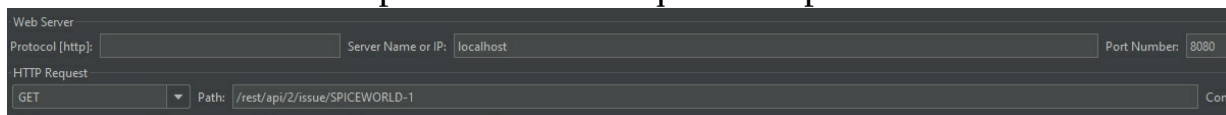
Thread Properties

Number of Threads (users): 1

Ramp-up period (seconds): 1

Loop Count: ☐ Infinite 1

Inside the Thread Group add a HTTP request sampler.



Web Server

Protocol (http): Server Name or IP: localhost Port Number: 8080

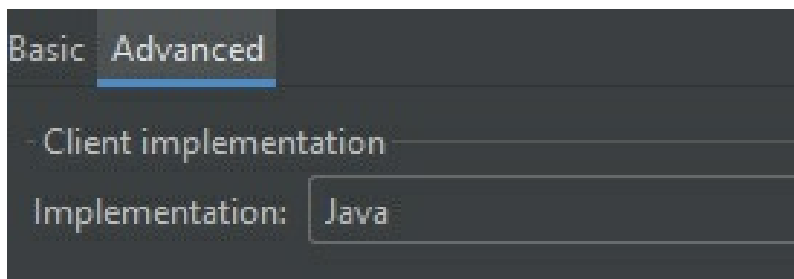
HTTP Request

GET Path: /rest/api/2/issue/SPICEWORLD-1

Note: The Server Name is localhost, you want to talk to the local JIRA service. The port must be 8080. Also this is a Get request. Finally the Path is the API we are calling with the name of the ticket added, in this case:

/rest/api/2/issue/SPICEWORLD-1

Now switch to the advanced tab



Basic Advanced

Client implementation

Implementation: Java

Implementation must be Java.

Now add a Results Tree Listener

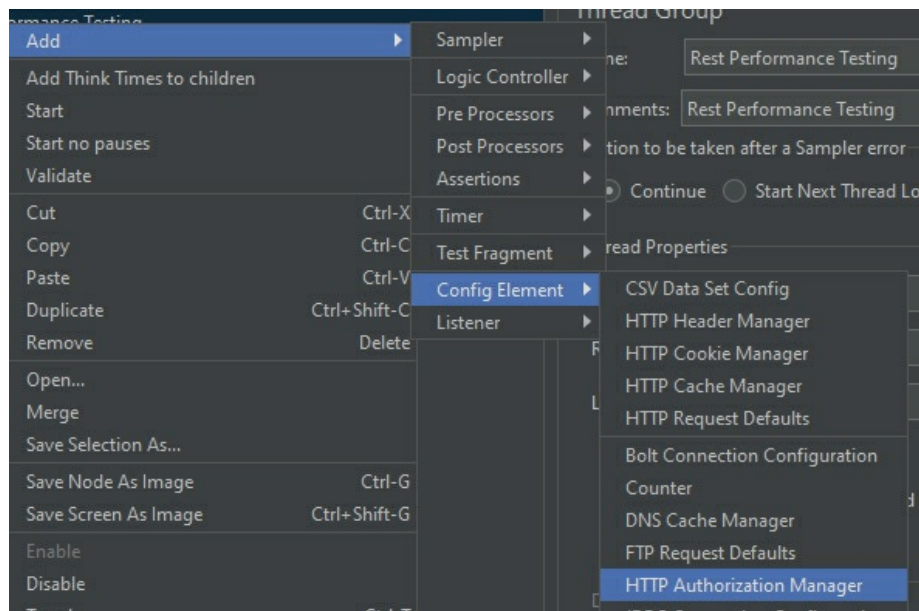


View Results Tree

Name: Rest View Results Tree

Comments:

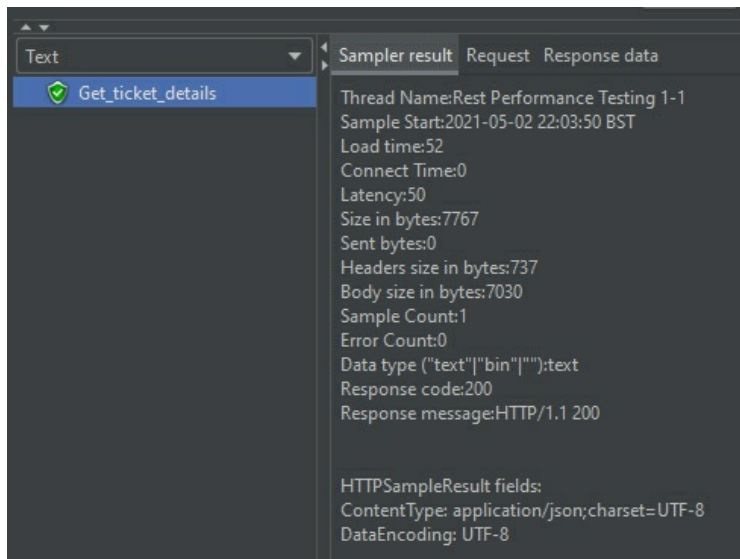
Finally we will need to supply the username and password of the user that you created and added to the project. So now add a HTTP Authorization Manager element.



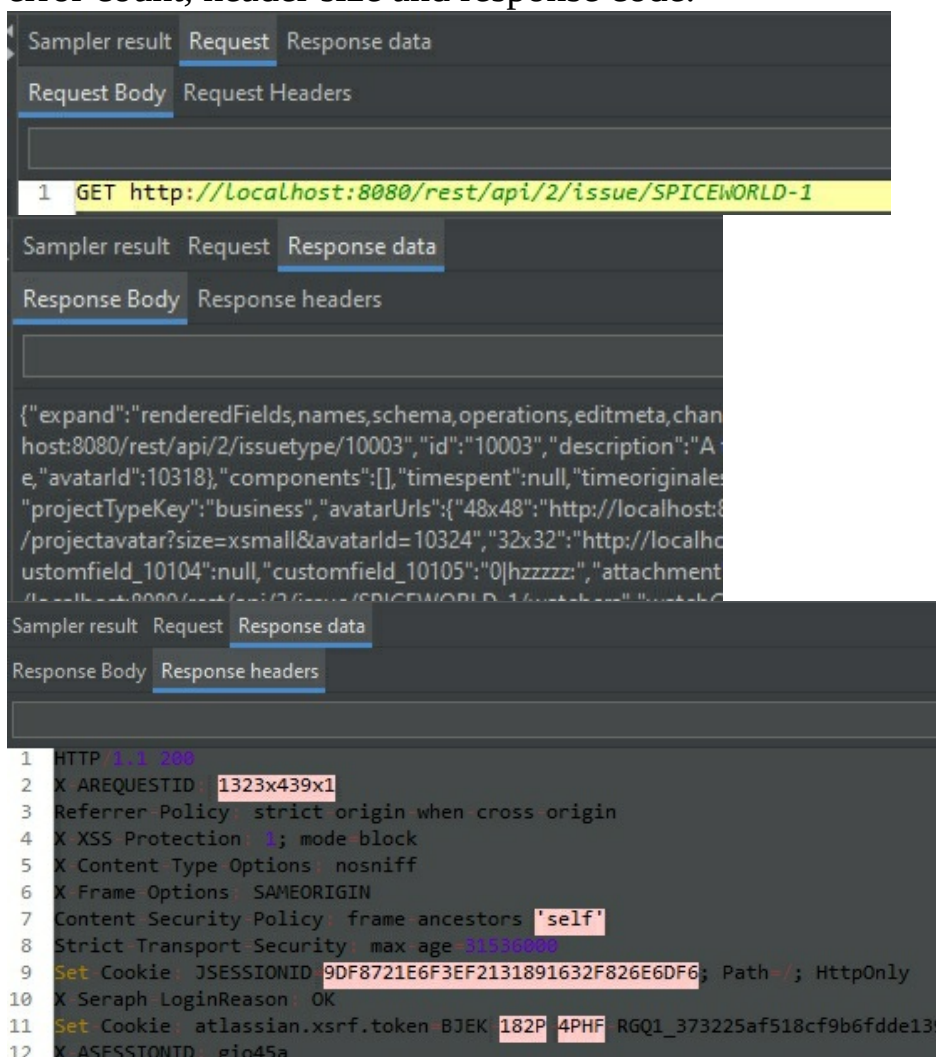
Use the Add button to enter the user details

A screenshot of the 'HTTP Authorization Manager' configuration dialog box. At the top, there are four buttons: 'Add', 'Delete', 'Load', and 'Save'. Below these buttons, the 'Name' field is filled with 'HTTP Authorization Manager'. The 'Comments' field is empty. Under the 'Options' section, there are two checkboxes: 'Clear auth on each iteration?' (unchecked) and 'Use Thread Group configuration to control clearing' (unchecked). At the bottom, there is a table titled 'Authorizations Stored in the Authorization Manager' with three columns: 'Base URL', 'Username', and 'Password'. The first row contains the values 'http://localhost:8080', 'kev', and '.....'.

Now save and run in the usual fashion, if all goes well you will see something like this.

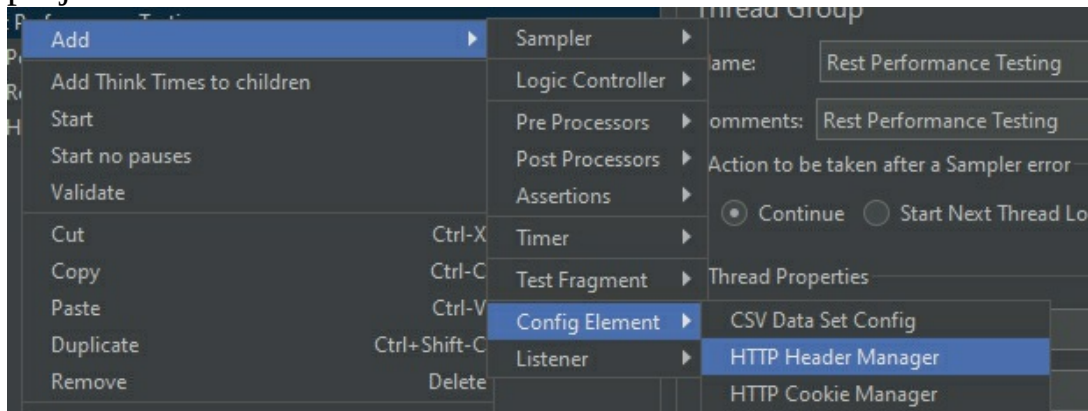


As you can see with this output the response times are all displayed as is the error count, header size and response code.

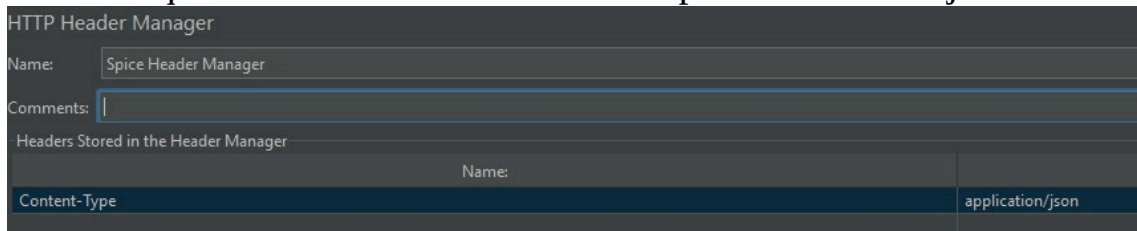


If that all worked then great, let's move on now and try a test using the POST method. As you probably know GET actions return existing data if a match is found while POST methods are designed to create new records, JIRA has API functions to accomplish this so let us try one of them out.

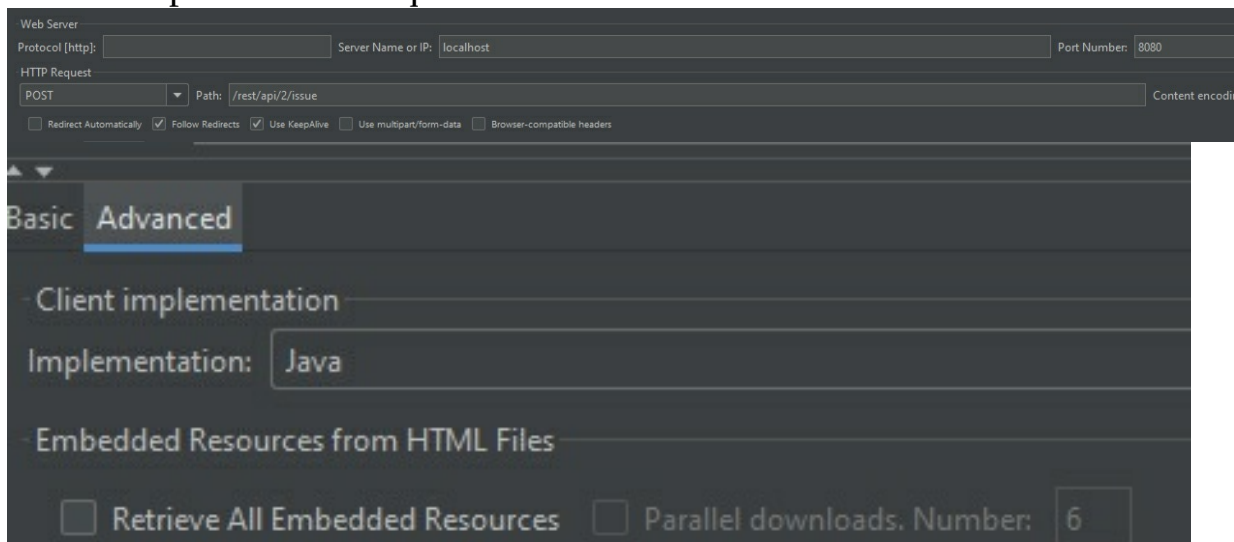
First we insert new a new HTTP header manager element in our JMeter project.



This is required so that we can ensure the posted data is in json format.



Now set up the HTTP request similar as below.




In the body data section add the task data, this will vary depending on your JIRA account. The key you would have created while creating your JIRA project.

```
Parameters Body Data Files Upload
1 [{"fields":{
2   "project":
3   {"key": "SPICEWORLD"},
4   "description": "Spice JMeter Issue",
5   "priority": {"name": "Medium"},
6   "reporter": {"name": "Kev"},
7   "assignee": {"name": "Kev"},
8   "summary": "JMeter rest test",
9   "issuetype":{"name": "Task"}
10  }
11 }
12 }
```

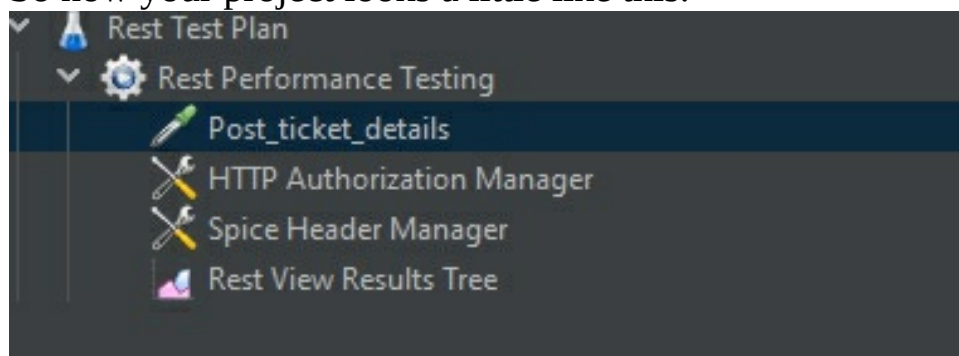
If you are not sure on the key then go to JIRA and start the process of creating a new ticket, you will then see the key.

Create Issue

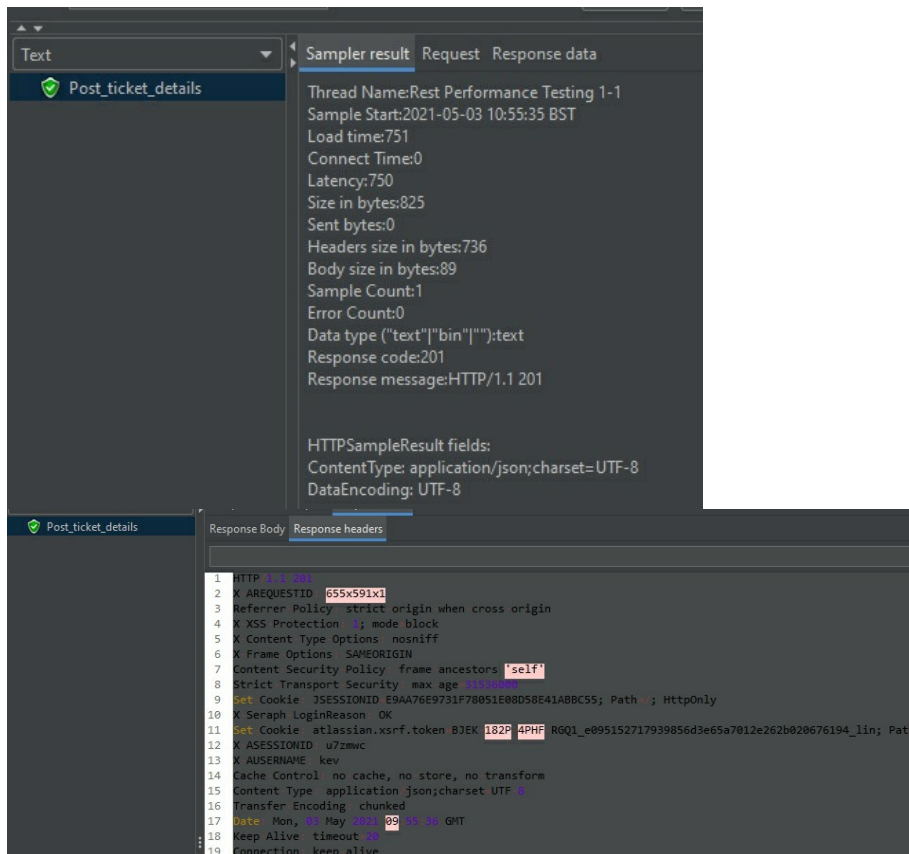
Project*  SpiceTheWorld (SPICEWORLD) ▼

Issue Type* ☒ Task ▼ ?

So now your project looks a little like this.




Note that the Authorization Manager settings are the same as for the previous GET example. So now save, clear and run in the usual manner and check the results, hopefully it will be like this.








Now to confirm, go back to JIRA and check the ticket now exists.



So there you have it, a JMeter POST test which returns success and the response times. So next we will look at PUT. This action will attempt to amend an existing record which has previously been created by a PUSH. What we will do here is amend the record created by our last POST call, then the priority was set to Medium, now I want to change this up to High.


 SpiceTheWorld / SPICEWORLD-2

JMeter rest test

 Edit  Comment  Assign  More  Done

▼ Details

Type: ☒ Task

Priority:  Medium

Labels: None

First the HTTP request is changed to type PUT and the ticket ID becomes part of the Path setting.

HTTP Request

Name: Post_ticket_details

Comments:

...

Basic

Advanced

Web Server

Protocol (http): Server Name or IP: localhost Port Number: 8080

HTTP Request

PUT Path: /rest/api/2/issue/SPICEWORLD-2

Content

☐ Redirect Automatically ☒ Follow Redirects ☒ Use KeepAlive ☐ Use multipart/form-data ☐ Browser-compatible headers

Parameters Body Data Files Upload

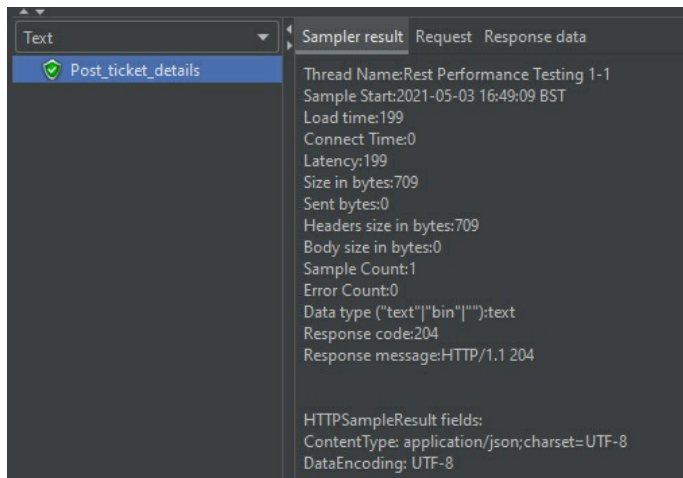
File Path Parameter Name MIME Type

The body data needs a couple of changes also

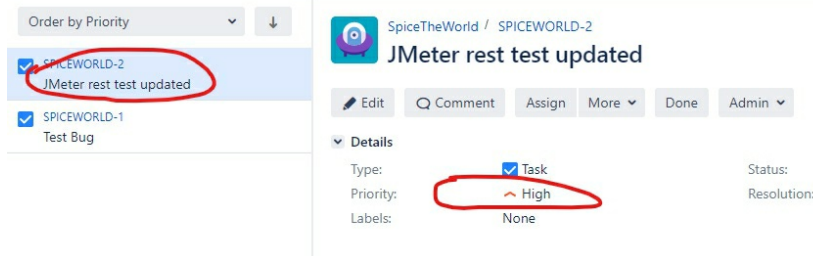
Parameters Body Data Files Upload

```
1 [{"fields": {
2   "project" :
3     {"key": "SPICEWORLD"},
4   "description": "Spice JMeter Issue",
5   "priority": {"name": "High"},
6   "reporter": {"name": "Kev"},
7   "assignee": {"name": "Kev"},
8   "summary": "JMeter rest test updated",
9   "issuetype": {"name": "Task"}
10  }
11  ]
```

Everything else can remain the same as the previous POST action. Priority is now High and summary also has the word updated insert. So now save, clear and run.



Everything looks good in JMeter, but what about JIRA?

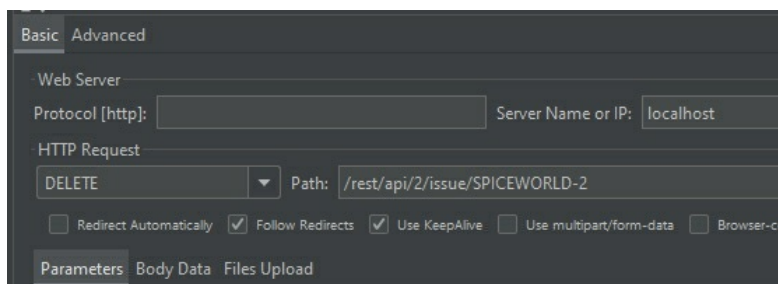


Success, the ticket in JIRA has also been updated. The final API call we will look at here is the DELETE method. This we will use to delete the ticket we have just created and amended.

To Delete this JIRA record we need to make 2 small changes to our last project.

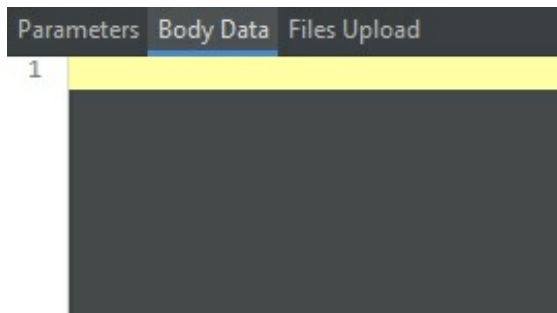
1. Change the request

to type DELETE.

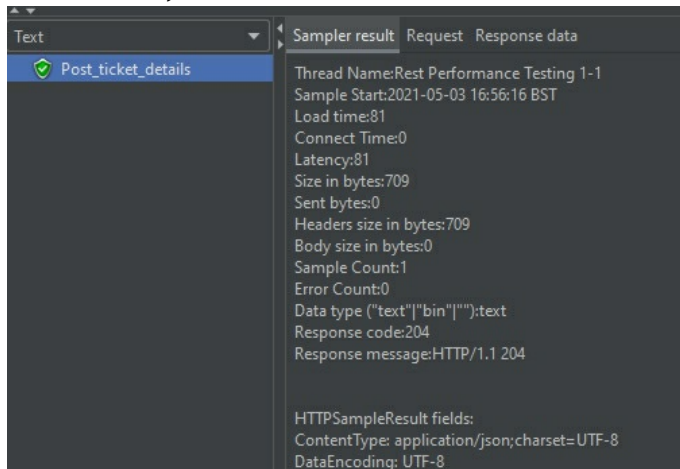


2. Remove the body

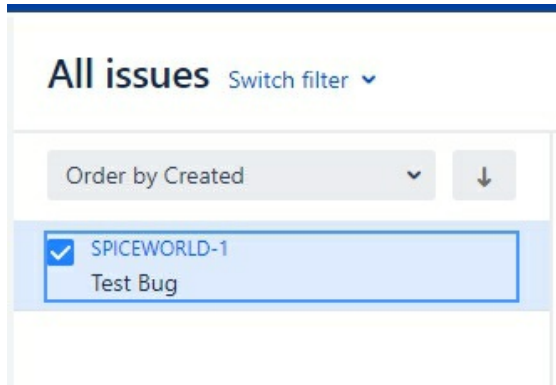
data, it is no longer required.



Now save, clear and run etc.



Then look for the ticket in Jira, it should be gone.

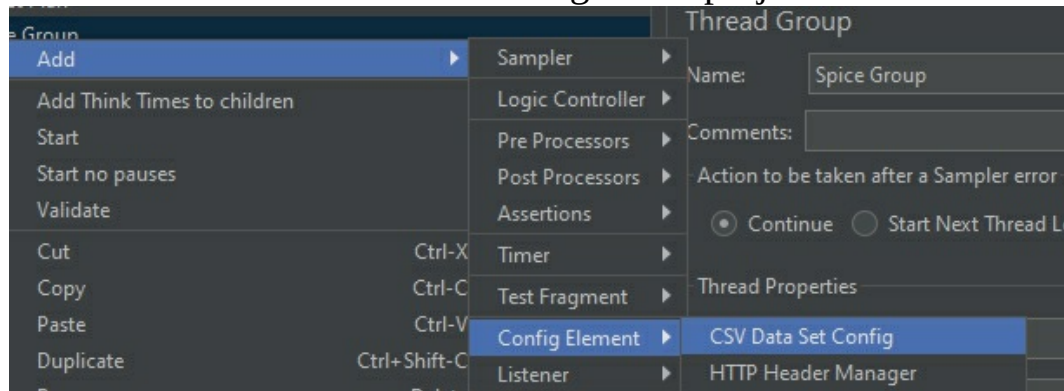


In my example it has been removed as expected. So there you go JMeter using API calls to updated JIRA and recording the results and response times, pretty cool eh?

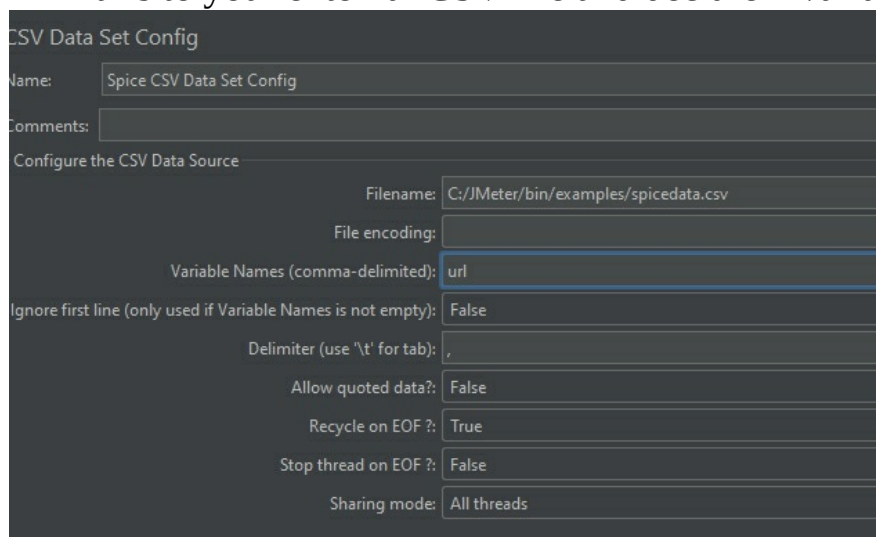
Now let's look at one final thing. Let us say we want to test the response times of a website when under load. We also want to test different pages within the website and these URL's are stored in a file.

We want 25 users to hit all 15 URL's 10 times. Let us put together a project to accomplish this task and produce some nice reports after the run is complete.

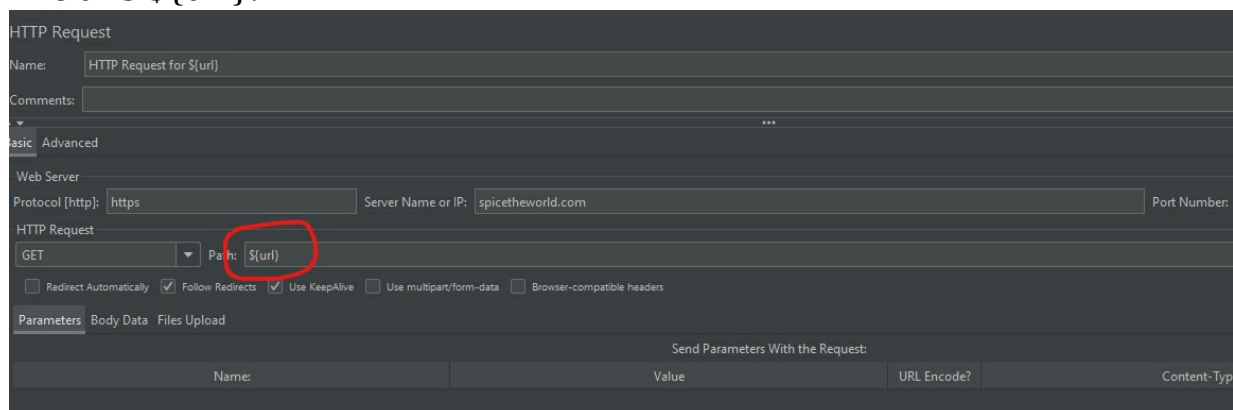
First we add a CSV Data Set Config to our project as below.



Link this to your external CSV file and declare 1 variable, I have called It url.



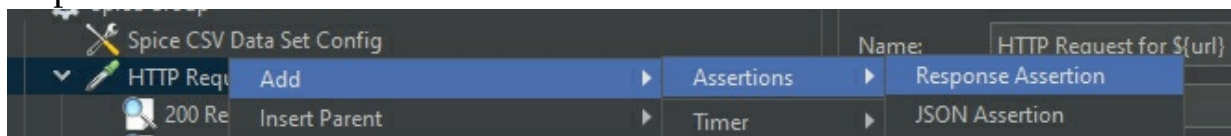
Now in your HTTP Request element set the path to link to the variable url like this `${url}`.



This will ensure the paths list in the CSV are read in and set to the path value. Below is the contents of the CSV file (spicedata.csv).

```
/
/Info/List/CH
/Info/List/GL
/Info/History/CT
/Home/Selection/CH
/Home/Recipe/72
/Home/Selection/CA
/Home/Selection/CA%239
/Home/Recipe/155
/Home/SiteMap
/Home/Recipe/81
/User/Login
/Info/History/GR
/Home/Recipe/140
/Home/Selection/TR%2314
```

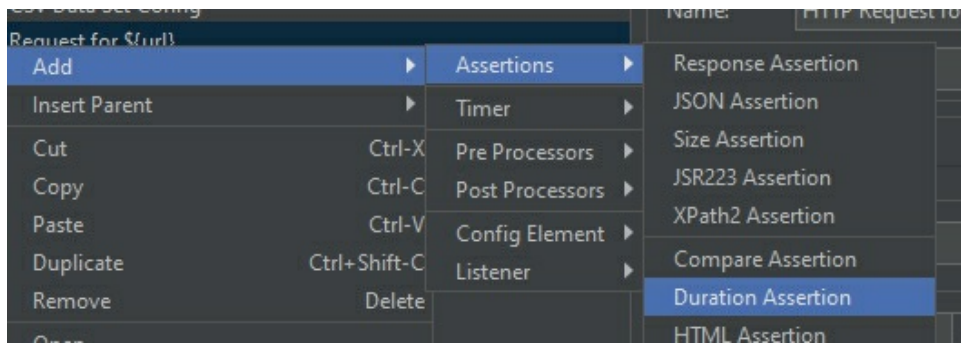
Next we want a couple of assertions to test on each iteration. First let us add a response assertion.



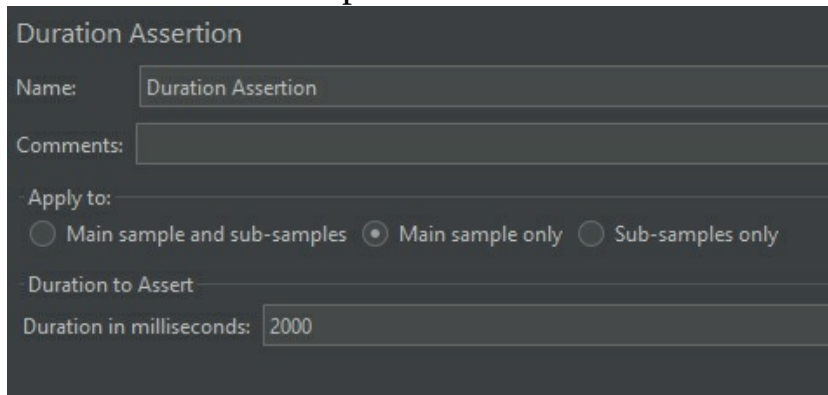
In this example if the response code is 200 we have a pass, anything else will be a fail.

A screenshot of the 'Response Assertion' configuration dialog in JMeter. The 'Name' field is set to '200 Response Assertion'. The 'Apply to:' section has 'Main sample only' selected. The 'Field to Test' section has 'Response Code' selected. The 'Pattern Matching Rules' section has 'Substring' selected. The 'Patterns to Test' list contains the value '200'.

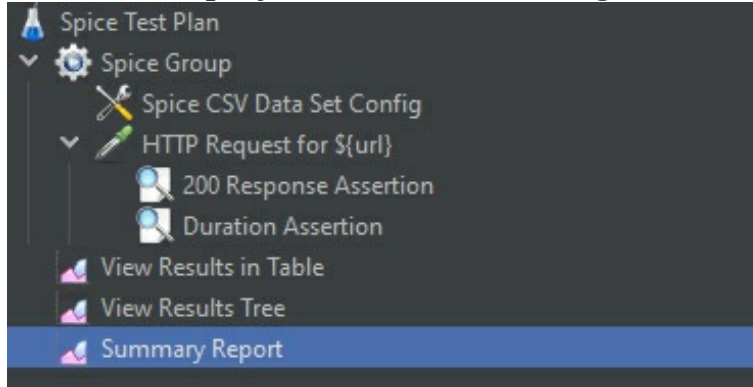
Assert 2 will be a duration test.



In this case if the response is slower than 2 seconds then we have a fail, otherwise we have a pass.



So our final project will be something like this.



So for one more time Save, Clean and Run. Hopefully, your results will be like mine on the following page.

Text

- ✓ HTTP Request for /Home/Recipe/81
- ✓ HTTP Request for /User/Login
- ✓ HTTP Request for /Info/History/GR
- ✓ HTTP Request for /Home/Recipe/140
- ✓ HTTP Request for /Home/Selection/TR%2314
- ✓ HTTP Request for /
- ✓ HTTP Request for /Info/List/CH
- ✓ HTTP Request for /Info/History/CT
- ✓ HTTP Request for /Info/List/GL
- ✓ HTTP Request for /Home/Recipe/72
- ✓ HTTP Request for /Home/Selection/CH
- ✓ HTTP Request for /Home/Selection/CA
- ✓ HTTP Request for /Home/Selection/CA%239
- ✓ HTTP Request for /Home/Recipe/155
- ✓ HTTP Request for /Home/Recipe/81
- ✓ HTTP Request for /Home/SiteMap
- ✓ HTTP Request for /User/Login
- ✓ HTTP Request for /Info/History/GR
- ✓ HTTP Request for /Home/Recipe/140
- ✓ HTTP Request for /Home/Selection/TR%2314
- ✓ HTTP Request for /
- ✓ HTTP Request for /Info/List/CH
- ✓ HTTP Request for /Info/History/CT
- ✓ HTTP Request for /Info/List/GL
- ✓ HTTP Request for /Home/Selection/CH
- ✓ HTTP Request for /Home/Recipe/72
- ✓ HTTP Request for /Home/Selection/CA
- ✓ HTTP Request for /Home/Selection/CA%239
- ✓ HTTP Request for /Home/Recipe/155
- ✓ HTTP Request for /Home/Recipe/81
- ✓ HTTP Request for /Home/SiteMap
- ✓ HTTP Request for /User/Login
- ✓ HTTP Request for /Info/History/GR
- ✓ HTTP Request for /Home/Recipe/140
- ✓ HTTP Request for /Home/Selection/TR%2314

This also.

141	20-43:26.109	Spice Group 1-10	HTTP Request for /Ho...	156	✓	18193	133	123	93
142	20-43:26.245	Spice Group 1-9	HTTP Request for /Ho...	121	✓	15223	136	121	93
143	20-43:26.266	Spice Group 1-10	HTTP Request for /Ho...	122	✓	13976	140	122	91
144	20-43:26.388	Spice Group 1-10	HTTP Request for /Ho...	151	✓	20756	134	123	87
145	20-43:26.539	Spice Group 1-10	HTTP Request for /Ho...	151	✓	37556	131	130	100
146	20-43:26.690	Spice Group 1-10	HTTP Request for /Ho...	153	✓	17893	133	153	122
147	20-43:26.843	Spice Group 1-10	HTTP Request for /User...	131	✓	13617	129	131	105
148	20-43:26.974	Spice Group 1-10	HTTP Request for /Info...	149	✓	18197	134	128	101
149	20-43:27.123	Spice Group 1-10	HTTP Request for /Ho...	165	✓	18502	134	143	113
150	20-43:27.288	Spice Group 1-10	HTTP Request for /Ho...	141	✓	14843	141	141	106

Let's also have a look at the Summary Report

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec
HTTP Request for /	10	161	142	235	25.10	0.00%	1.0/sec	18.40	0.12
HTTP Request for /Info/List/CH	10	165	147	201	15.61	0.00%	1.1/sec	32.78	0.13
HTTP Request for /Info/List/GL	10	172	155	189	11.50	0.00%	1.1/sec	36.39	0.14
HTTP Request for /Info/History/CT	10	140	122	153	9.06	0.00%	1.1/sec	17.50	0.14
HTTP Request for /Home/Selection/CH	10	168	153	204	15.11	0.00%	1.1/sec	19.17	0.14
HTTP Request for /Home/Selection/CA	10	133	126	139	3.85	0.00%	1.1/sec	16.19	0.14
HTTP Request for /Home/Recipe/72	10	162	150	215	17.82	0.00%	1.1/sec	23.59	0.14
HTTP Request for /Home/Selection/CA%239	10	137	120	164	12.96	0.00%	1.1/sec	15.01	0.15
HTTP Request for /Home/Recipe/155	10	159	144	182	12.66	0.00%	1.1/sec	22.04	0.14
HTTP Request for /Home/SiteMap	10	159	143	176	10.49	0.00%	1.1/sec	39.56	0.14
HTTP Request for /Home/Recipe/81	10	137	123	149	7.04	0.00%	1.1/sec	18.77	0.14
HTTP Request for /User/Login	10	132	113	144	9.09	0.00%	1.1/sec	14.18	0.13
HTTP Request for /Info/History/GR	10	153	139	169	8.59	0.00%	1.1/sec	18.76	0.14
HTTP Request for /Home/Recipe/140	10	160	144	192	14.64	0.00%	1.0/sec	18.86	0.14
HTTP Request for /Home/Selection/TR%2314	10	132	118	153	11.08	0.00%	1.0/sec	15.06	0.14
TOTAL	150	151	113	235	19.26	0.00%	13.3/sec	269.79	1.72

So, what have we got here?

Label: In the label section you will be able to see all the recorded http request, during test run or after test run.

Samples: Samples denote to the number of http request ran for given thread, or the average time taken to receive the web pages. For Example we have one http request and we run it with 5 users, than the number of samples will be $5 \times 1 = 5$. Same if the sample ran two times for the single user, than the number of samples for 5 users will be $5 \times 2 = 10$.

Average: Average is the average response time for that particular http request. This response time is in millisecond. This an Arithmetic mean for all responses (sum of all times / count)

Min: Min denotes to the minimum response time taken by the http request.

Max: Max denotes to the maximum response time taken by the http request.

Std.Deviation: This shows how many exceptional cases were found which were deviating from the average value of the receiving time. The lesser this value more consistent the time pattern is assumed.

Error %: This denotes the error percentage in samples during run. This error can be of 404(file not found) or may be exception or any kind of error during test run will be shown in Error %. In the above image the error % is zero, because all the requests ran successfully.

Throughput: The throughput is the number of requests per unit of time (seconds, minutes, hours) that are sent to your server during the test. Larger is better.

KB/Sec: The throughput measured in Kilobytes per second

Looking good so far but one final step, let us now run this project via the command line and create a html report. My command line is below, yours may differ slightly depending on your folder setup. I executed this command line from my c:\jmeter folder. Also note this command could just as easily be

placed in a batch file.

```
jmeter -n -t examples\spicetheloop.jmx -l c:\jmeter\bin\html\log.jtl -e -o c:\jmeter\bin\html
```

So you may be wondering what the options mean, well a full list can be seen here:

<https://jmeter.apache.org/usermanual/get-started.html>






These are the options I used.

- n** This specifies JMeter is to run in cli mode
- t** [name of JMX file that contains the Test Plan]
- l** [name of JTL file to log sample results to]
- e** generate report dashboard after load test
- o** output folder where to generate the report dashboard after load test. Folder must not exist or be empty

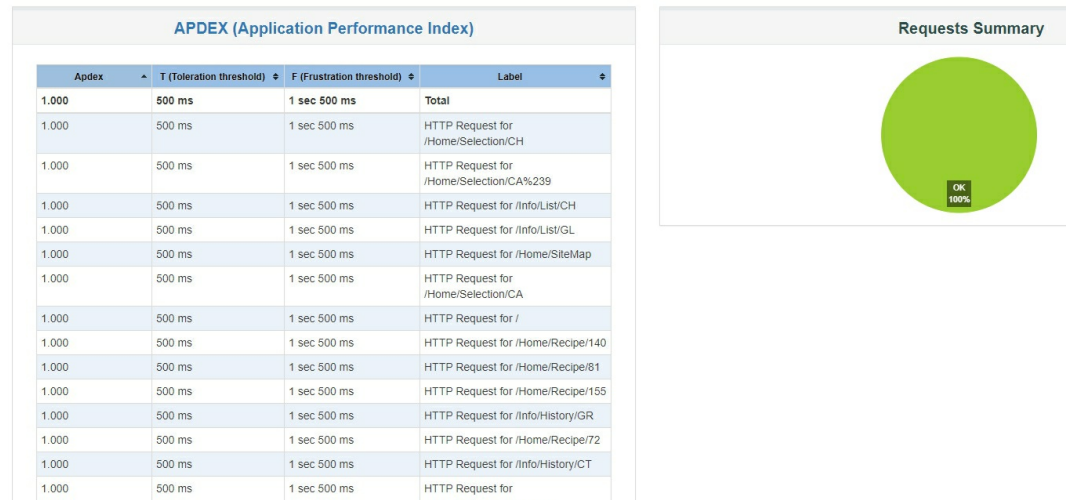
Now run the command line and wait for the completion.

```
C:\JMeter\bin>jmeter -n -t examples\spicetheloop.jmx -l c:\jmeter\bin\html\log.jtl -e -o c:\jmeter\bin\html
Creating summariser <summary>
Created the tree successfully using examples\spicetheloop.jmx
Starting standalone test @ Mon May 03 21:26:23 BST 2021 (1620073583906)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
summary + 78 in 00:00:06 = 12.8/s Avg: 156 Min: 124 Max: 413 Err: 0 (0.00%) Active: 3 Started: 7 Finished: 4
summary + 72 in 00:00:05 = 13.6/s Avg: 153 Min: 122 Max: 197 Err: 0 (0.00%) Active: 0 Started: 10 Finished: 10
summary = 150 in 00:00:11 = 13.2/s Avg: 154 Min: 122 Max: 413 Err: 0 (0.00%)
Tidying up ... @ Mon May 03 21:26:35 BST 2021 (1620073595339)
... end of run
```

Check the html folder, hopefully the report will exist.

	content	03/05/2021 21:23	File folder	
	sbadmin2-1.0.7	03/05/2021 21:23	File folder	
	index	03/05/2021 21:23	Chrome HTML Do...	10 KB
	log.jtl	03/05/2021 21:23	JTL File	23 KB
	statistics	03/05/2021 21:23	JSON File	9 KB

Open the report and enjoy the results.



So there you have a good introduction to the power of JMeter. This chapter is by no means a complete, advanced guide, more of a taster if you like. If you want to go deeper into this fascinating and interesting area of testing then you will find plenty of online courses and books to help you become an expert.

API Testing with Postman

A particularly important skill for any agile tester is the ability to test API's. Now I am sure most of you know what I mean by API but for those who do not here is a brief explanation.

API stands for **Application Programming Interface**. At some point or another, most large companies have built APIs for their customers, or for internal use. A good example is eBay, they have an entire range of API's which allow registered developers to create applications which can list products, retrieve sales and much more.

So a typical API could be considered an interface between you (the client) and a service. You will ask the service for some information by sending it then required parameters. Supplying the parameters are valid then the request information is returned to you (if it exists).

The essence of an API is that they make it possible for any two separate applications to transfer and share data between them. A good example is a front-end system asking for and receiving information from a back-end database system. They also make it easier for an application's users to execute actions without having to use the application's GUI. From the developers' view it is an effortless way to execute certain functionalities of their app and rigorously test them it as well.

However, Using APIs daily can become cumbersome and time consuming, as you may well have dozens or even hundreds of APIs that require testing. That can make it difficult to keep up with their exact request's address(es), header(s), authorization credential(s) etc., and by that make it harder to test the API for functionality, security, and exception handling. So, to help with these functions' tools have been created to make life a lot easier, one of the more popular of these tools is known as Postman.

Postman is a powerful tool used to test web services and APIs. It allows you to create a request with the required HTTP method and parameters, send the request, and inspect the results. It provides a sleek user interface with which to make HTML requests, without the need of writing a bunch of code just to test an API's functionality. There are also some genuinely nice, advanced

options which we will explore as we move through this chapter.

So first you need to install Postman (providing you have not already done so). At the time of publication, the download URL was <https://www.postman.com/downloads/>. The good news is that it is free so get yourself ready and let us start to learn Postman.

First load up Postman, you will see a screen like the one shown on the following page.



So let us start with a simple GET request, type in `https://api.github.com/users/1` in the input box shown below then click the Send button.



You will then see a similar response to this one in what is called the JSON format.

Body Cookies Headers (24) Test Results

Pretty

Raw

Preview

Visualize

JSON ▼



```
1 {
2   "login": "1",
3   "id": 1825798,
4   "node_id": "MDQ6VXN1cjE4MjU3OTg=",
5   "avatar_url": "https://avatars.githubusercontent.com/u/1825798?v=4",
6   "gravatar_id": "",
7   "url": "https://api.github.com/users/1",
8   "html_url": "https://github.com/1",
9   "followers_url": "https://api.github.com/users/1/followers",
10  "following_url": "https://api.github.com/users/1/following{/other_user}",
11  "gists_url": "https://api.github.com/users/1/gists{/gist_id}",
12  "starred_url": "https://api.github.com/users/1/starred{/owner}/{/repo}",
13  "subscriptions_url": "https://api.github.com/users/1/subscriptions",
14  "organizations_url": "https://api.github.com/users/1/orgs",
15  "repos_url": "https://api.github.com/users/1/repos",
16  "events_url": "https://api.github.com/users/1/events{/privacy}",
17  "received_events_url": "https://api.github.com/users/1/received_events",
18  "type": "User",
19  "site_admin": false,
20  "name": "Michael",
21  "company": null,
22  "blog": "",
23  "location": "San Francisco, CA",
24  "email": null,
25  "hireable": null,
26  "bio": null,
27  "twitter_username": null,
28  "public_repos": 5,
29  "public_gists": 0
```

So what did we do here? Well we sent a message to the GitHub API that we wanted to GET information about user 1. In response GitHub returned the requested information about user 1. This is an API in its simplest form and things we become more interesting as we move forward.

But first what is JSON, well JSON stands for JavaScript Object Notation and it is a lightweight format for storing and transporting data. Each data object has a name (for example “url”) and a value (example “https://www.kevsbooks.com”) and each data object is separated by commas. For more information on JSON please check the links in the appendix

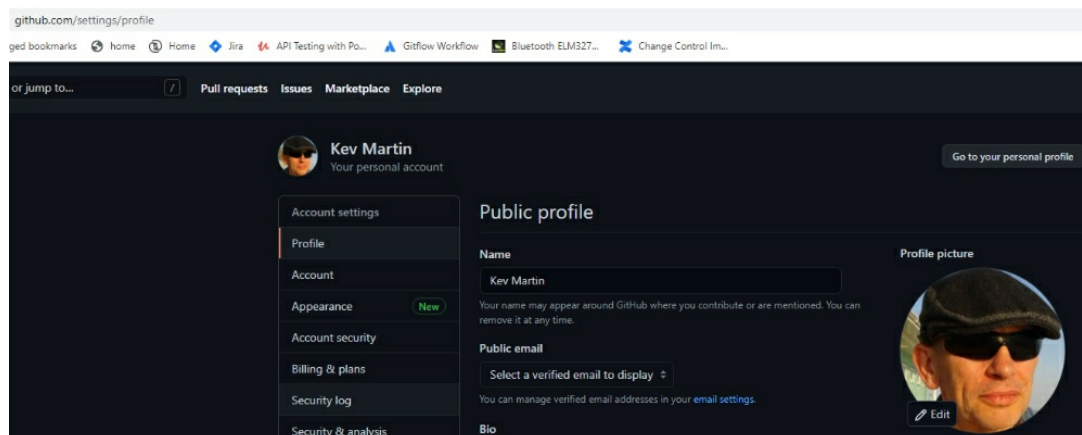
Now getting back to API's. The first step when trying to understand how a set of APIs function is to read the supplied documentation.

Unfortunately, API documentation can be poor or limited so when working on a new project always ask for a complete documentation. This should be in the form of an API agreement which will inform you what an API does. An API contract is a shared understanding of what the capabilities of the interface are, allowing for applications to be programmed on top of and tested in Postman. The information supplied should include a description of each API, the endpoint URL, a full list of the parameters you can pass, what parameters you should expect in return and what (if any) authorisation information you will need to supply.

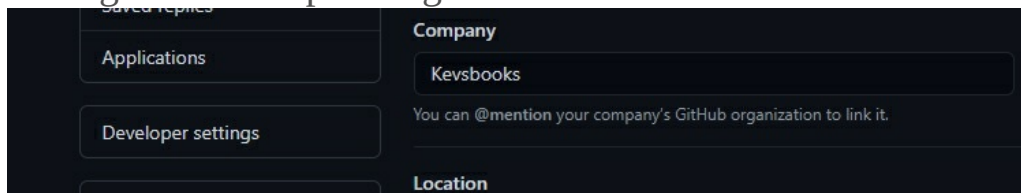
In the APIs for testing are inhouse and are being or have been developed by your companies own developers then these people can also be a reliable source of information on how you can test the API. Also being able to access the actual code is another way of understanding how to test them.

Now let us look at authentication. Some APIs are public and the allow anyone to interact with it. Others will require authentication authorisation so to interact with these you will need to authenticate yourself in some way. How this works will vary from one API to another and therefore it is important to read the API documentation in advance.

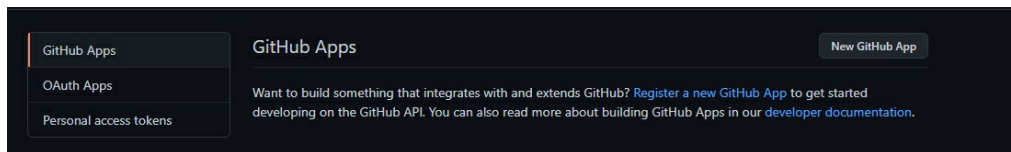
With public APIs a common way of gaining access is through a personal token. There can be generated within the APIs website and they allow you access to their public APIs. Let us see a real example of this, if you have a GitHub account then sign in and go here <https://github.com/settings/profile>



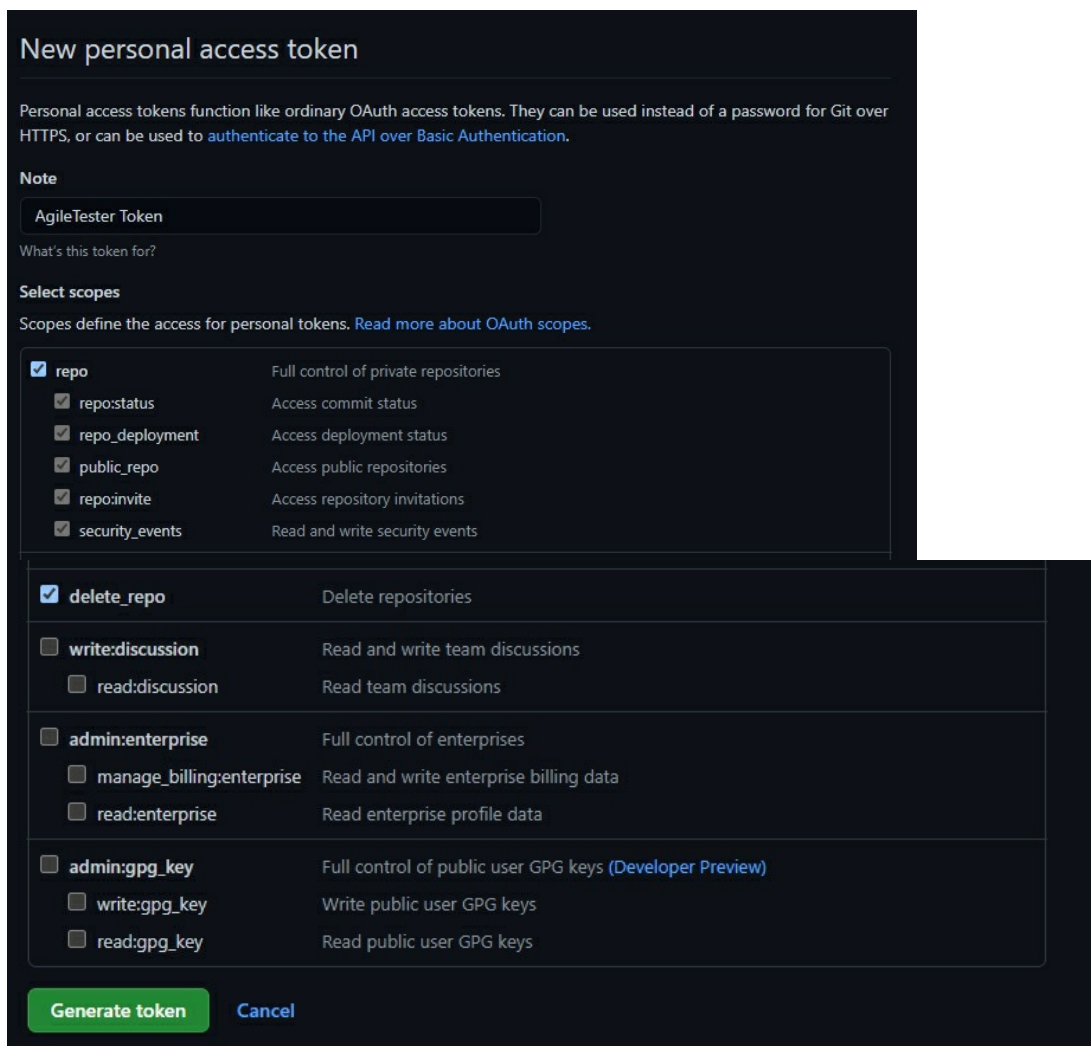
Then go to Develop settings



Then select Personal access tokens

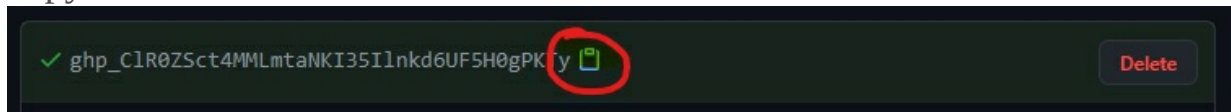


Now create a new token

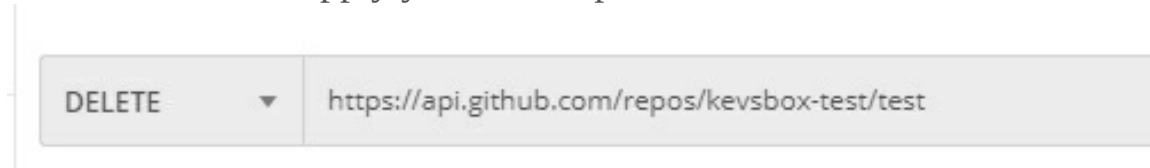


This will generate the new token and one of the permissions granted is that of removing repos. The token will now appear in your tokens list and you can

copy it here –



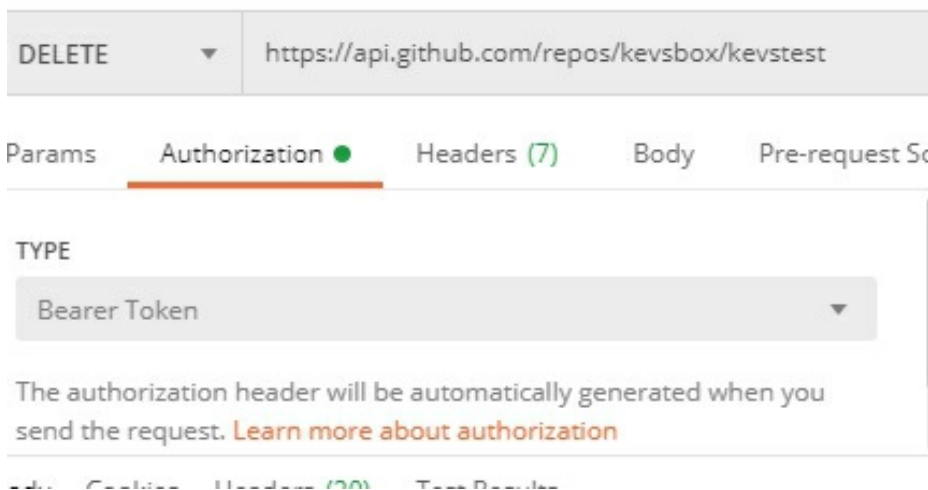
Now return to postman and create a DELETE call like below then click send, note will have to supply your own repo.



The returned message is as expected in this case.



So let us handle this by adding our token, in Authorization select Bearer Token



Then add in the token you just created.



Now try again and this time you should see no error. You should however see a 204 No Content message, this confirms the repo was removed and no longer exists.



Status: 204 No Content Time: 579 ms Size: 974 B

Another method of authentication will be the use of a username and password. If an API needs this method, then simply follow these steps. Set authorisation type to Basic Auth and then input the valid username and password combination.

TYPE

Basic Auth

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative

Username

Password

☐ Show Password

When testing API's there are some risks to be considered which will need to become part of your test strategy. The first risk is if tests start to fail should an API change. Hopefully, you will be informed of any changes to in-house API's and therefore you will be able to update your tests accordingly. The problems can arise with third party API's and should your tests begin to fail then this is the first place to look. Check the online documentation for the API, is there a new release?

The next risk is availability. If the API you are testing against is a test version of the APIs and there may be times the Test API is not available, this will need to be factored into any testing you have planned. You will also need to ensure that access to the internet is always available when testing.

Other considerations, especially with test systems, is performance and timeouts. These are factors that need to be considered when testing APIs.

So now let us look further a GET requests, these are the most common type of API tests you will see. You might think that these are remarkably simple, you send a request to a server, and you get a response, that it all done. Not quite, there is more to it than that. The first question is how do you know that what has been returned is correct?

Well the first thing to check is does the returned data match what should have been returned according to the API documentation. Is there returned data in the correct format? Are all expected fields returned with data? Next repeat the

call with slightly different parameters, does the returned information appear in a consistent format with the earlier call.

To check this out let us do a get call on this endpoint - <https://api.spacexdata.com/v3/launches> and confirm that what is returned is in fact all SpaceX launches, if correct thousands of lines of data will be returned with below an exceedingly small example.

```
{
  "flight_number": 1,
  "mission_name": "FalconSat",
  "mission_id": [],
  "upcoming": false,
  "launch_year": "2006",
  "launch_date_unix": 1143239400,
  "launch_date_utc": "2006-03-24T22:30:00.000Z",
  "launch_date_local": "2006-03-25T10:30:00+12:00",
  "is_tentative": false,
  "tentative_max_precision": "hour",
  "tbd": false,
  "launch_window": 0,
  "rocket": {
    "rocket_id": "falcon1",
    "rocket_name": "Falcon 1",
    "rocket_type": "Merlin A",
    "first_stage": {
      "cores": [
```

So let's now reduce the data returned by adding some parameters. If you look at the returned JSON data we have a name of "rocket_name" so let us, try a search for all launches of the Falcon 1 rocket.

GET https://api.spacexdata.com/v3/launches?rocket_name=Falcon 1

Params **Authorization** Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
<input checked="" type="checkbox"/> rocket_name	Falcon 1
Key	Value

Body Cookies (1) Headers (30) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "flight_number": 1,
3   "mission_name": "FalconSat",
4   "mission_id": [],
5   "upcoming": false,
6   "launch_year": "2006",
7   "launch_date_unix": 1143239400,
8   "launch_date_utc": "2006-03-24T22:30:00.000Z",
9   "launch_date_local": "2006-03-25T10:30:00+12:00",
10  "is_tentative": false,
11  "tentative_max_precision": "hour",
12  "tbd": false,
13  "launch_window": 0,
14  "rocket": {
15    "rocket_id": "falcon1",
16    "rocket_name": "Falcon 1",
17  }
```

Now if you look through the returned records we have just over 700 lines and all rockets were Falcon 1. So next lets add a date range

GET https://api.spacexdata.com/v3/launches?rocket_name=Falcon 1&launch_year=2009

Params **Authorization** Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
<input checked="" type="checkbox"/> rocket_name	Falcon 1
<input checked="" type="checkbox"/> launch_year	2009
<input type="checkbox"/> order	launch_year
Key	Value

This time we get back 2 records. Some APIs will also allow you to sort the data on specific fields and some will also allow you to limit the number of records returned. This is not true of all APIs so it is important to read the documentation fully.

So far we have look at GET's. These are static calls that pull back existing data. Now we will look at POST calls which are designed to create new records.

In this example we are going to add a new Board to my Trello account, the first thing you will notice that instead of GET we use POST. The next thing you will notice is that some of the parameter data is stored in variables. You could just as easily replace this with typed data. We will talk about variables a little later.

POST

https://api.trello.com/1/boards/?defaultLists=false&key={{trelloKey}}&token={{trelloToken}}&name=KevsBooks

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
<input checked="" type="checkbox"/> defaultLists	false
<input checked="" type="checkbox"/> key	{{trelloKey}}
<input checked="" type="checkbox"/> token	{{trelloToken}}
<input checked="" type="checkbox"/> name	KevsBooks
Key	Value

Now if we send this request then hopefully we will get a success (200) message.

 Status: 200 OK Tii

```
"id": "607b412566bbb86d0b2262fa",
"name": "KevsBooks",
"desc": "",
"descData": null,
"closed": false,
"idOrganization": "6061d4b20edcbb79da4fc261",
```

So there you go the new record was created and if you repeat this on your own Trello account you will find the new board exists on your own account.

There you have a simple example of a POST request. Now let us look at a PUT request.

POST and PUT are similar and often use the same authentication methods but where they differ is that POST creates a record while PUT will try to change an existing record. So look back at the Trello board we just created and now let us send a PUT command which will rename it.

If you look back at the last post you will see the newly created board was given an id

```
"id": "607b412566bbb86d0b2262fa",
"name": "KevsBooks",
"desc": "",
"descData": null,
"closed": false,
"idOrganization": "6061d4b20edcbb79da4fc261",
```

This now becomes part of the url for the PUT, therefore we now have

<https://api.trello.com/1/boards/607b412566bbb86d0b2262fa>

We still need the name, key, and token parameters. The key and token parameters stay the same, but the name value will be what we want to rename to board to, therefore we now have.

PUT	https://api.trello.com/1/boards/607b412566bbb86d0b2262fa?key={{trelloKey}}&token={{trelloToken}}&name=KevsBooks Renamed
Params	Authorization Headers (7) Body Pre-request Script Tests Settings
Query Params	
KEY	VALUE
<input checked="" type="checkbox"/> key	{{trelloKey}}
<input checked="" type="checkbox"/> token	{{trelloToken}}
<input checked="" type="checkbox"/> name	KevsBooks Renamed
Key	Value

The result is once again a 200

```
1  {
2    "id": "607b412566bbb86d0b2262fa",
3    "name": "KevsBooks Renamed",
4    "desc": "",
5    "descData": null,
6    "closed": false,
```

So that now gives as a good working example of a PUT. Next, we will remove this board with a DELETE call.

If you look at the example, call below you will notice it is all but the same as the earlier PUT command apart from the fact it is now a DELETE call. The id is still part of the url and the parameters are also the same. So try and send

this request.

DELETE	https://api.trello.com/1/boards/607b412566bbb86d0b2262fa?key={{trelloKey}}&token={{trelloToken}}&name=KevsBooks Renamed
Params Authorization Headers (6) Body Pre-request Script Tests Settings	
Query Params	
KEY	VALUE
<input checked="" type="checkbox"/> key	{{trelloKey}}
<input checked="" type="checkbox"/> token	{{trelloToken}}
<input checked="" type="checkbox"/> name	KevsBooks Renamed
Key	Value

Again, we get a 200 success and we returned data is null, this is correct, the record was removed and therefore all that could be returned was null. You should always be incredibly careful with the DELETE command, once a record is removed it cannot be recovered, when it is gone it is really gone.

Body	Cookies	Headers (40)	Test Results (1/1)	Status: 200 OK
Pretty	Raw	Preview	Visualize	JSON
1	2	3	{"_value": null}	

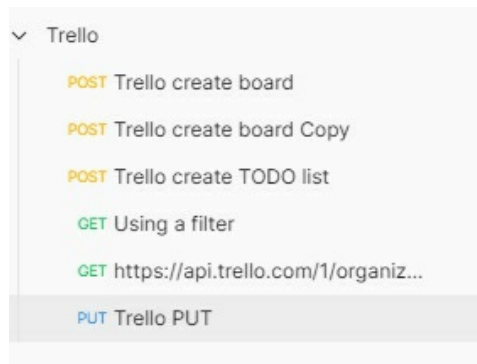
So that is the four most common types of calls. We have looked at GET, POST, PUT and DELETE. Now it is time to look at some more advanced features of Postman. One feature about Postman is that it can be used to automate testing and a key feature of automation in Postman is collections, so let us look at this feature next.

Collections are a fantastic way of organising your tests, especially as the number of saved tests begins to grow. Below is a list of stored collections in my Postman

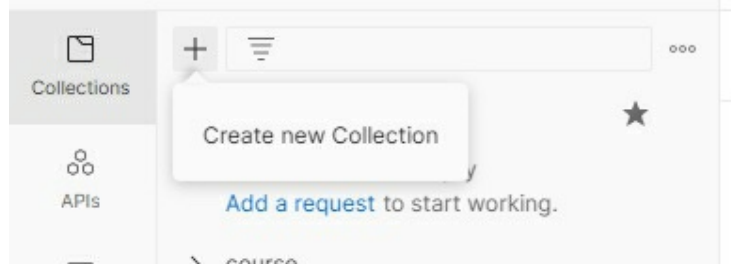
- > course
- > github
- > httpbin
- > SpaceX
- > Trello

Now if I expand the Trello collection you will then see the tests I have stored

within



To create a new collection I can either use this option



Or when I save a new test I can choose to save it to an existing collection or create a new one at that time.

SAVE REQUEST

Requests in Postman are saved in collections (a group of requests).

[Learn more about creating collections](#)

Request name

Trello PUT V2

Request description (Optional)

Make things easier for your teammates with a complete request description.

Descriptions support [Markdown](#)

Select a collection or folder to save to:

◀ Trello

+ Create Folder

POST

Trello create board

🔒

POST

Trello create board Copy

🔒

POST

Trello create TODO list

🔒

So collections are a simple but effective way to store and sort your Postman tests. Collections also have some immensely powerful options which you can take advantage of, now let us look at each of these.

First we have Authorization. At a collection level any test can inherit anything stored here including bearer tokens. Therefore, you do not need to add these at a test level if they are the same for each test in the collection, this is quite common when testing a particular API.

Authorization Pre-request Script Tests Variables

This authorization method will be used for every request in this collection. You can override this by specifying one in the request.

Type No Auth

This collection does not use any authorization. [Learn more about authorization](#)

To enable this feature simply add the token to the Collection and in each test set Authorization to inherit.

Params Authorization Headers (6) Body Pre-request Script

Type Inherit auth from parent

Now let us chat about Pre-request scripts. These will execute before any test in your collection, a good example being the need to generate a random name before each test. We investigate this further a little further on in this chapter. Next we have Tests, these are scripts which will run after each of your requests, for example this one below will pass on success and fail otherwise.

Trello

Authorization Pre-request Script Tests Variables

These tests will execute after every request in this collection. [Learn more about tests](#)

```
1 pm.test("Status code is 200", function () {
2   |   pm.response.to.have.status(200);
3   | });
```

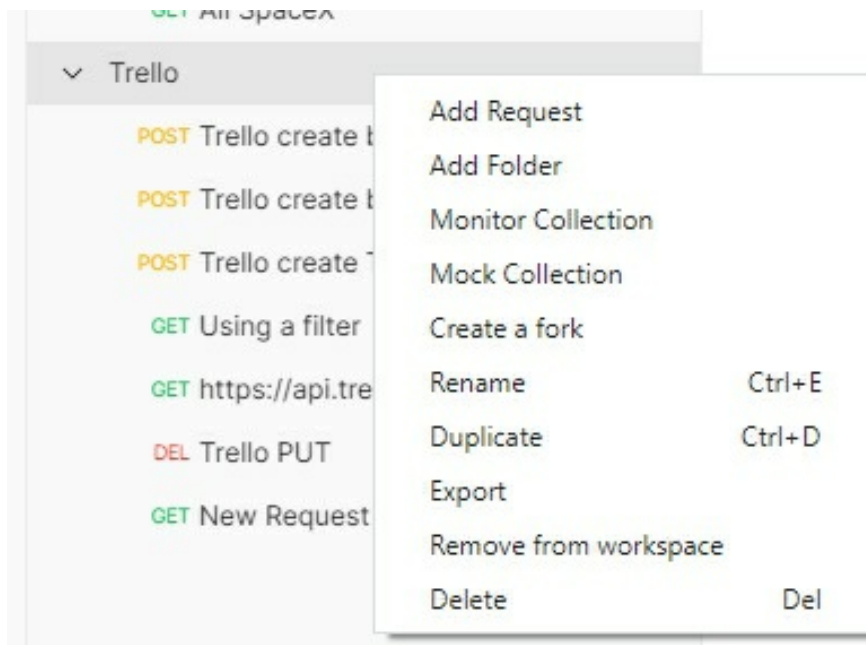
This test will be executed after every request in your collection. Finally, we have variables. Once again these are available at the collection level so the variable shown below would be available to every request within the selected collection.

Authorization Pre-request Script Tests **Variables**

These variables are specific to this collection and its requests. [Learn more about collection variables.](#)

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ
<input checked="" type="checkbox"/>	boardname	KevsBooks	

Adding a new request to a collection is extremely easy also. Select the required collection and press the right mouse button. Select 'Add Request' and create as normal. When you save the request it will be added to your collection.



Now if we create a new variable to our collection like so –

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ
<input checked="" type="checkbox"/>	Nationality	CA	CA
<input checked="" type="checkbox"/>			

Then we create a new request within the collection which will use this variable

GET

▼

https://randomuser.me/api?nat={{Nationality}}

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Query Params

	KEY	VALUE
<input checked="" type="checkbox"/>	nat	{{Nationality}}
	Key	Value

Then we will see in this example the response has returned an address in Canada which matches the CA value

```
{
  "city": "Maidstone",
  "state": "Nova Scotia",
  "country": "Canada",
  "postcode": "X8N 7P2",
```

Using variable is really that simple. Also, you can change the value anytime it is needed, for example lets us change to Germany.

These variables are specific to this collection and its requests. [Learn more about collection variables.](#)

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ
<input checked="" type="checkbox"/>	Nationality	CA	DE

Then return the request and check the result

```
{
  "city": "Diez",
  "state": "Baden-Württemberg",
  "country": "Germany",
  "postcode": 30452,
```

Now let us look at Tests within a request. When you click on the tests tab you will see something like this on the next page.

You will also notice a list of predefined snippets on the right

SNIPPETS

Get an environment variable

Get a global variable

Get a variable

Get a collection variable

Set an environment variable

Set a global variable

Set a collection variable

Clear an environment variable

Clear a global variable

Clear a collection variable

Status code: Code is 200

Scroll down the list until you see **Status code: Code is 200** as select this snippet. Your first test is then added with no coding by your good self.

```
1 pm.test("Status code is 200", function () {  
2     pm.response.to.have.status(200);  
3 });
```

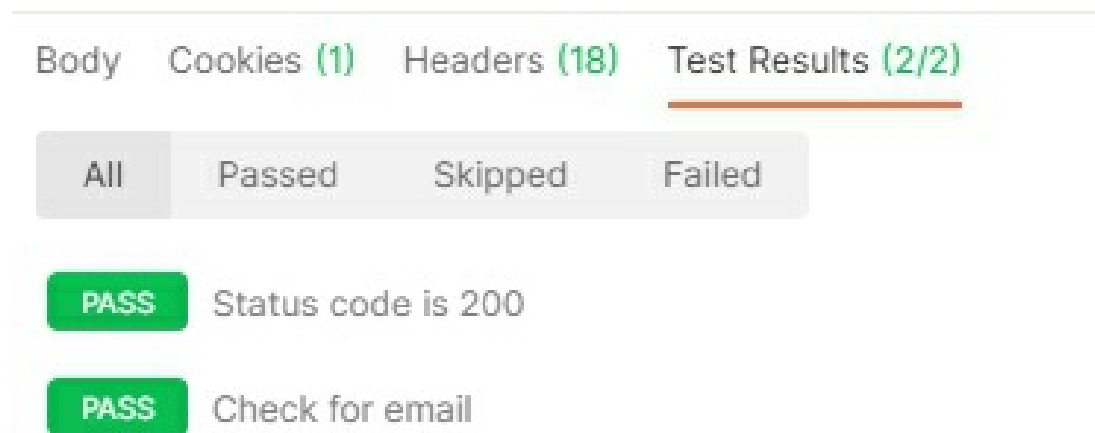
Postman uses the Chai assertion library, for more information on this library

please refer to <https://www.chaijs.com/api/bdd/>

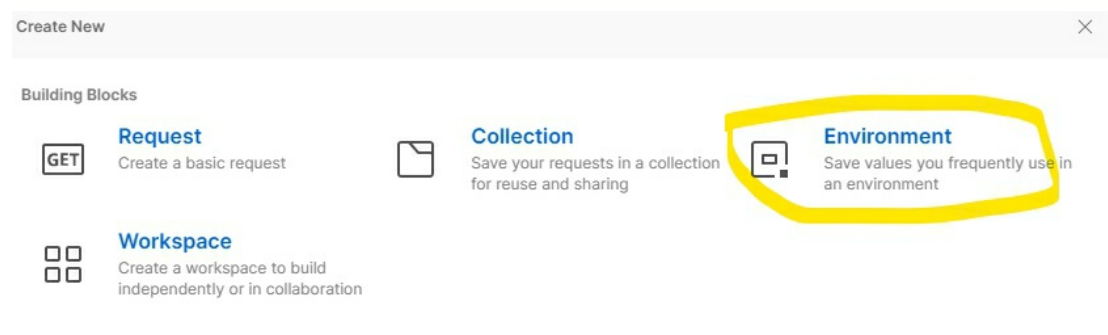
Next add this snippet Response body: JSON value check to the tests and adjust to this

```
pm.test("Check for email", function () {  
  var jsonData = pm.response.json();  
  pm.expect(jsonData.results[0].email).to.include('@');  
});
```

Now this test will expect the returning body to have a key called email and as with any email address the data should include the @ symbol. Now run the request and both tests will be executed after the run and both will hopefully pass.



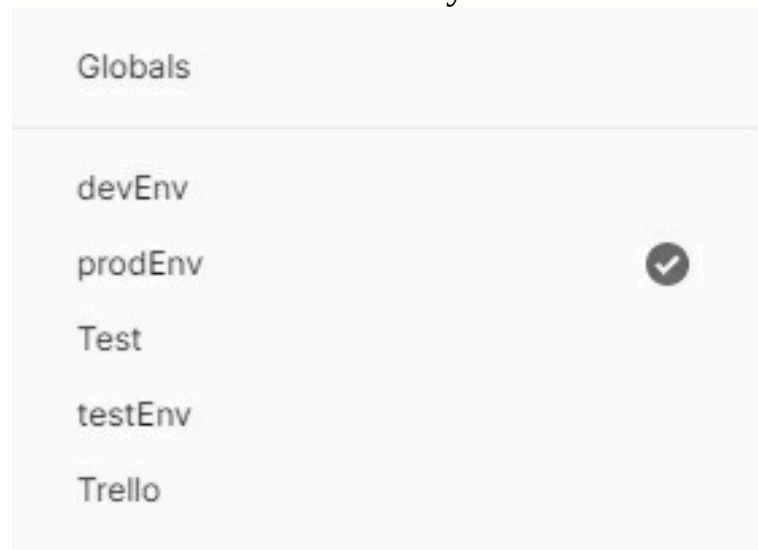
In this case they indeed did pass. So now you can run tests against your Postman requests let us now look at sharing data between requests. To achieve this we can use Environments.



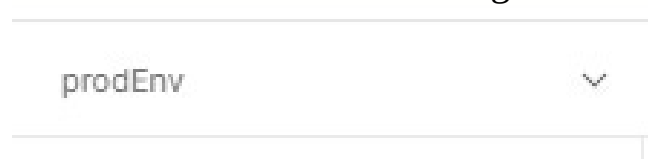
This is my prodEnv Environment


prodEnv		Edit
VARIABLE	INITIAL VALUE	CURRENT VALUE
url	https://www.spicetheworld.com	https://www.spicetheworld.com
environment	Production	Production

And below is a list of all my Environment in Postman

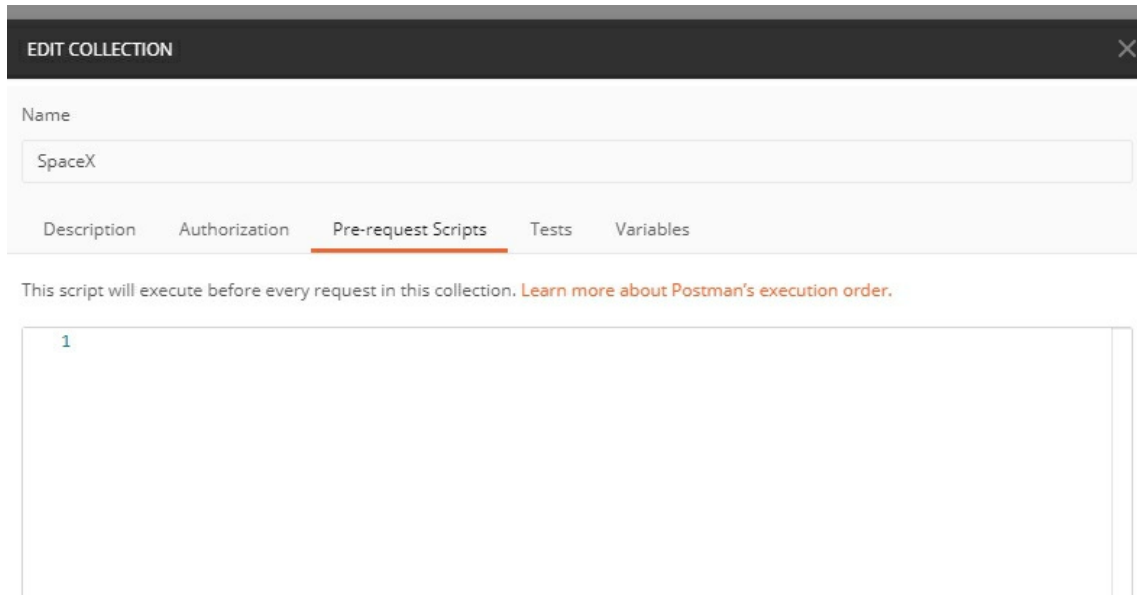


You can then select an existing Environment from this pulldown list



Then view the contents by clicking here  , and of course all variables within the select Environment are available in your requests as are those in the Global section.

Another part of automation within Postman is the use of Pre-request scripts so let us look at one of my saved collections and edit the Pre-request section.



Now what we want to do here is run a check to ensure the environment is available before trying any requests. So let us now add this code -

```
pm.sendRequest("https://api.spacexdata.com", function(err, response){  
    if(response.status == 'OK'){  
        pm.environment.set("Item", 'true')  
    }else{  
        pm.environment.set("Item", 'false')  
    }  
})
```

This will test that the URL is available. If it is then the variable Item will be true otherwise it will be false. This variable can then be used with the collection requests to figure out if they should be run or not.

The next part of Postman we need to look at is Mocking. To create a mock, you simply click the New button and then select Mock Server.



Next choose 'Select an existing collection' and finally we create the Mock server. When created you will see the Mock server URL and you can copy this into a new request, however if you try and run this as-is you will get an error message. So, what we do now is create an example. Go back to your

original request and re-run. After the request has run select Examples and save this result as an example.

This example has the URL -

https://api.spacexdata.com/v3/launches?launch_year=2009&limit=2

Your Mock has the URL -

<https://87bfb5b3-a025-4f77-9700-a174c0fee335.mock.pstmn.io/>

This needs to change to

https://87bfb5b3-a025-4f77-9700-a174c0fee335.mock.pstmn.io/v3/launches?launch_year=2009&limit=2

Make this change and re-run the Mock and this time you should get a success 200 message and the expected body data.

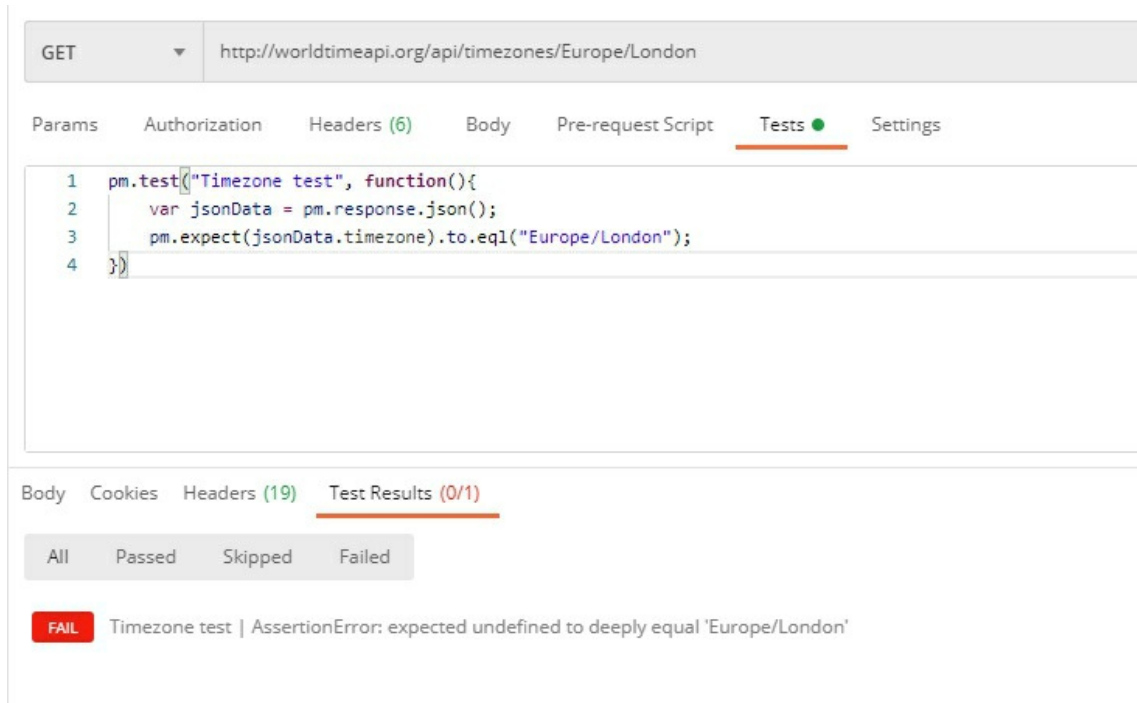
By adding a mock server to your collection and adding examples to your requests, you can then simulate the behaviour of a real-world API on your own machine. When you send a request to a mock server, Postman will match the request configuration to the examples you have saved for the request and respond with the data you added to the example.

So, mocking means you are creating a fake version of an external or internal service that can stand in for the real one, helping your tests run more quickly and more reliably. When your implementation interacts with an object's properties, rather than its function or behaviour, a mock can be used.

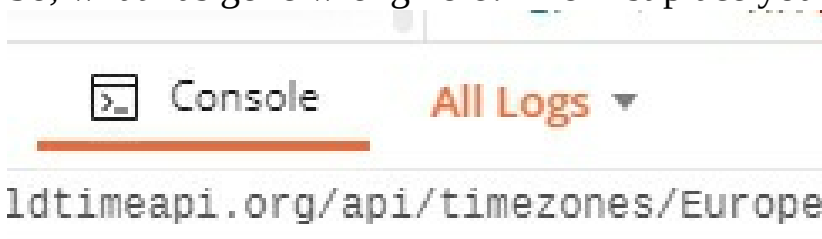
There will be times when the API under test does not work as expected, so let us look at a simple example and see how we can figure out what is going wrong. In this example I use the endpoint

<http://worldtimeapi.org/api/timezones/Europe/London>

This example has failed before Europe/London was undefined, also if you look at the response Body you will see a lot of responses that are not Europe.



So, what has gone wrong here? The first place you can check is the Console.



Open the console and run the request again, then check for any error messages.

```
Find and Replace Console All Logs ▼
▼ GET http://worldtimeapi.org/api/timezones/Europe/London
  ▼ Proxy
    href: "http://andweb1.simplyhealth.co.uk:8080/"
  ► Network
  ▼ Request Headers
    User-Agent: "PostmanRuntime/7.26.5"
    Accept: "*/*"
    Postman-Token: "72f21974-d2e0-4fbd-aada-ac0c8261b53b"
    Host: "worldtimeapi.org"
    Accept-Encoding: "gzip, deflate, br"
    Connection: "keep-alive"
  ▼ Response Headers
    Access-Control-Allow-Credentials: "true"
    Access-Control-Allow-Origin: "*"
    Access-Control-Expose-Headers: ""
    Cache-Control: "max-age=0, private, must-revalidate"
    Content-Type: "application/json; charset=utf-8"
```

In this instance there is nothing obvious in the console so probably the next place to check will be the API documentation, and in this case the issue was discovered. Initially I have typed in -
`http://worldtimeapi.org/api/timezones/Europe/London`

However, the documentation indicates that this should be -
`http://worldtimeapi.org/api/timezone/Europe/London`

Can you see the difference? One little typo. So now we update the GET URL and try again.

The screenshot displays the Postman interface. At the top, a GET request is configured to `http://worldtimeapi.org/api/timezone/Europe/London`. The 'Tests' tab is active, showing a JavaScript test function:

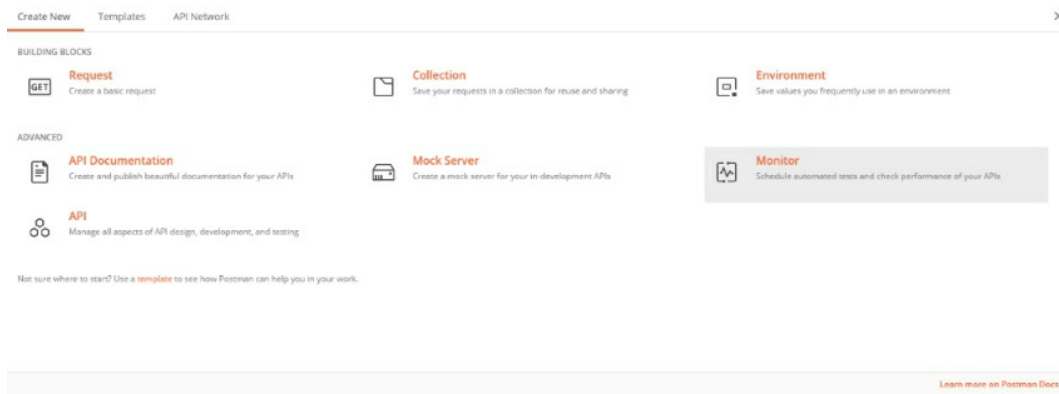
```
1 pm.test("Timezone test", function(){
2     var jsonData = pm.response.json();
3     pm.expect(jsonData.timezone).to.eql("Europe/London");
4 })
```

Below the test editor, the 'Test Results' tab shows a successful result (1/1). The 'JSON' tab is selected, displaying the response body as a JSON object:

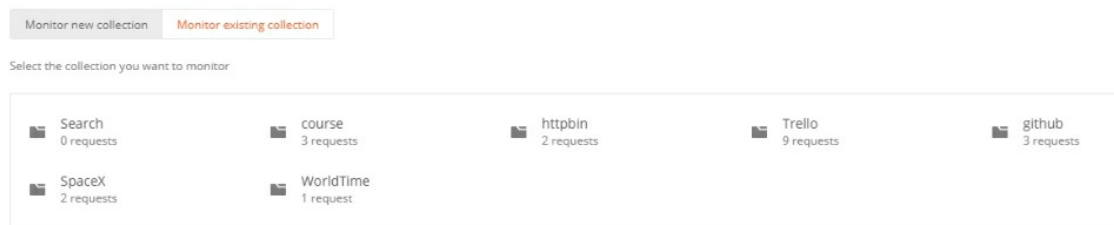
```
1 {
2     "abbreviation": "BST",
3     "client_ip": "81.145.151.253",
4     "datetime": "2021-04-19T11:36:38.504840+01:00",
5     "day_of_week": 1,
6     "day_of_year": 109,
7     "dst": true,
8     "dst_from": "2021-03-28T01:00:00+00:00",
9     "dst_offset": 3600,
10    "dst_until": "2021-10-31T01:00:00+00:00",
11    "raw_offset": 0,
12    "timezone": "Europe/London",
13    "unixtime": 1618828598,
14    "utc_datetime": "2021-04-19T10:36:38.504840+00:00",
15    "utc_offset": "+01:00",
16    "week_number": 16
17 }
```

Now we get the expected response, and the test has passed. Happy days indeed. So, when things do not go as expected. Explore and debug, the problem is often a very simple one.

Another useful tool is the Monitor feature, this allows you to schedule runs.



Here you can either select to monitor an existing collection or create a new one.



Then you can set the monitor up to run daily, hourly etc. There are other options you can also explore. These include emailing the results and selecting a region.

The screenshot shows the 'Monitor' configuration form. It includes fields for 'Monitor name' (WorldTime), 'Version Tag' (CURRENT), 'Use an environment (optional)' (No Environment), 'Monitor run frequency' (Week Timer, Every Day, 12:00 AM), 'Regions' (Automatically Select Region, Manually Select Region), and a checkbox for 'Receive email notifications for run failures and errors' (checked). Below the checkbox is an email address field (kev@kevsbox.com) and a note 'You can add up to 5 notification recipients'. At the bottom, there is a 'Stop notifications after' field (3) and a 'consecutive failures' label. At the very bottom, there are 'Back' and 'Create' buttons.

When you have created the monitor you will see something like this below.



WorldTime monitor created

Runs at a schedule to check the performance of your requests

NEXT STEPS

You can also go to the web dashboard where you can manually start a run, view results and even edit the monitor settings.

- **Check monitor results**

See the monitor run results on the [web dashboard](#)

After a few runs the history graph will begin to make sense and will become a useful tool for reporting you test results.



Another cool feature of Postman is the ability to use data from a CSV file. Next we have a nice, simple example of this feature.

people 19/04/2021 14:15 Microsoft Excel C... 1 KB

So we have this file and the contents are -
person
keybox
tester
fred

The first row is the column name and the next three rows are treated as data.

So back in Postman we can adapt an existing request thus –

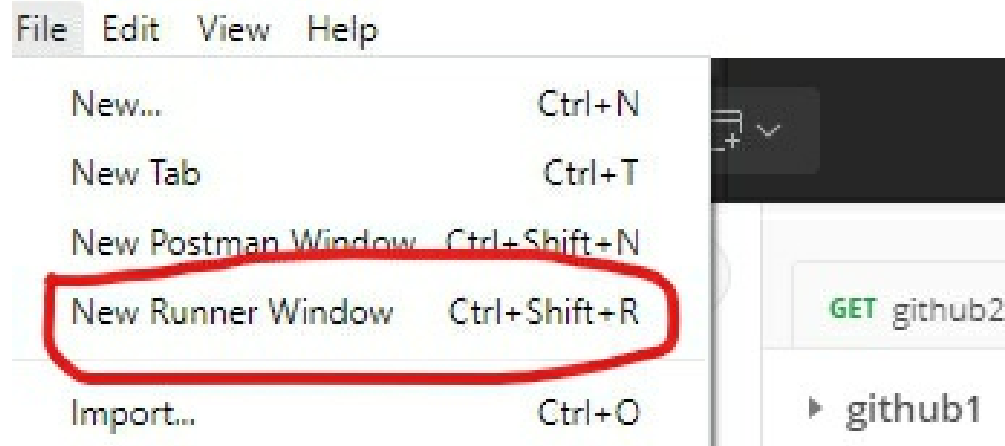
<https://api.github.com/users/kevsbox>

To –

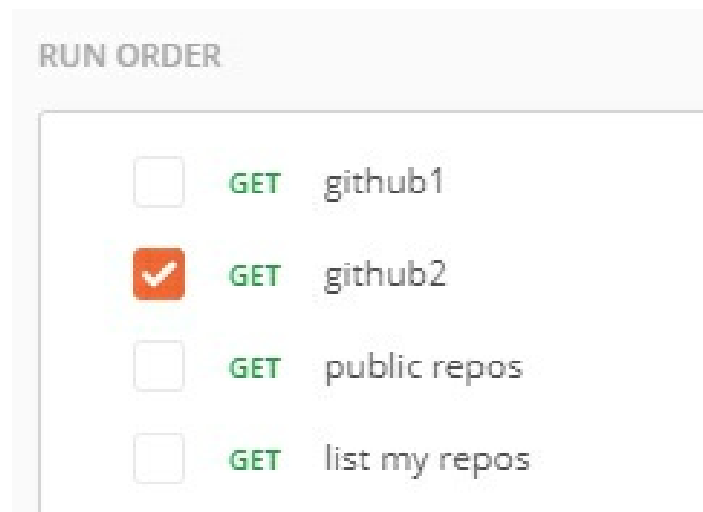
<https://api.github.com/users/{{person}}>

Now make sure the value inside `{{}}` matches the column name, also make sure you save the request before trying to run it.

Next we go to.



Select the required collection and make sure your update request is the only one ticked.



Select File

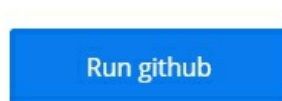
Now select the data file

Data Select File people.csv X

Data File Type text/csv ▼ Preview




You can preview this file if you wish.

Iteration	person
1	"kevsbox"
2	"tester"
3	"fred"



Now click the Run button

If all goes well you will see a collection of passes like those shown on the following page.

Iteration 1					
	GET	github2	https://api.github.com/u...	/ github2	200 OK 437 ms 2.439 KB
This request does not have any tests.					
Iteration 2					
	GET	github2	https://api.github.com/u...	/ github2	200 OK 343 ms 2.389 KB
This request does not have any tests.					
Iteration 3					
	GET	github2	https://api.github.com/u...	/ github2	200 OK 371 ms 2.48 KB
This request does not have any tests.					

Another great feature of Postman is the ability to import API definitions via Swagger/Open API. These files are often created by the API provider, and they will include all the endpoints currently available along with information on how to execute them. Below is a sample from one such yaml file.

```
swagger: "2.0"
info:
  version: 1.0.0
```

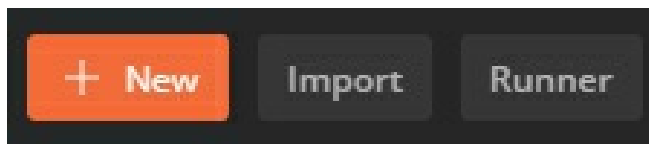


```

title: Swagger Petstore
license:
  name: MIT
host: petstore.swagger.io
basePath: /v1
schemes:
  - http
consumes:
  - application/json
produces:
  - application/json
paths:
  /pets:
    get:
      summary: List all pets
      operationId: listPets
      tags:
        - pets
      parameters:
        - name: limit
          in: query
          description: How many items to return at one time (max 100)
          required: false
          type: integer
          format: int32
      responses:
        "200":
          description: A paged array of pets
          headers:
            x-next:
              type: string
              description: A link to the next page of responses
          schema:
            $ref: '#/definitions/Pets'
        default:
          description: unexpected error
          schema:
            $ref: '#/definitions/Error'
    post:

```

As you can see there are definitions for both GET and POST. So to make use of these files you first need to download them and then import it into Postman.



After clicking Import locate the yaml file and click Import

Import

Confirm your import

Double check the format is correct and select which files you want to import.

NAME	FORMAT	IMPORT AS
Swagger Petstore	Swagger 2.0	API

☒ Generate collection from imported APIs

Link this collection as

Documentation

Show advanced settings

Cancel

Import

You will then be asked to confirm the import.

Import

Confirm your import

Double check the format is correct and select which files you want to import.

NAME	FORMAT	IMPORT AS
Swagger Petstore	Swagger 2.0	API

☒ Generate collection from imported APIs

Link this collection as

Documentation

Show advanced settings

Cancel

Import

Then if all has gone well you will see the message below.

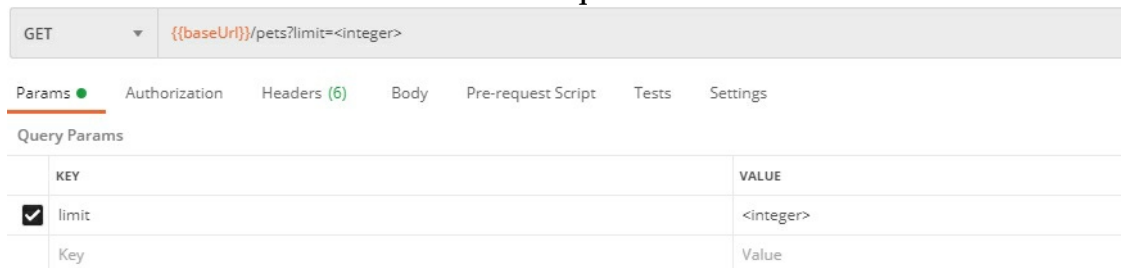
Import complete

 1 API and 1 collection were imported.

Then in the left hand column you will see the new requests in the own collection



And these can all be executed as required.



The final feature to consider here is how to validate the API schema contract using tv4. Most API suppliers will provide schema information on their API, this is a good example -



When called this will return details of properties within the API calls, this will include field types and if they are optional etc.

```

"description": "A person within the Star Wars universe",
"title": "People",
"required": [
  "name",
  "height",
  "mass",
  "hair_color",
  "skin_color",
  "eye_color",
  "birth_year",
  "gender",
  "homeworld",
  "films",
  "species",
  "vehicles",
  "starships",
  "url",
  "created",
  "edited"
]

```

1.

So, to validate the API copy the returned body into the Tests box and add this line before so you have something like this -

```

1  const sw_schema =
2  {
3  |   "description": "A person within the Star Wars universe",
4  |   "title": "People",
5  |   "required": [
6  |     "name",
7  |     "height",
8  |     "mass",
9  |     "hair_color",
10 |     "skin_color",
11 |     "eye_color",
12 |     "birth_year",
13 |     "gender",
14 |     "homeworld",
15 |     "films",
16 |     "species",
17 |     "vehicles",
18 |     "starships",
19 |     "url",
20 |     "created",
21 |     "edited"
22 |   ]
23 | }

```

```

pm.test("schema validation", function(){
  const validationResult = tv4.validateResult(pm.response.json(), sw_schema, false, true)
  pm.expect(validationResult.valid).to.be.true;
});

```

Now scroll to the very bottom of the tests box and add this.
Now change the GET request to read –

<https://swapi.dev/api/people/1/> and click Send.

Body Cookies Headers (10) Test Results (1/1)

All Passed Skipped Failed

PASS schema validation

In this example we get a pass. The request looked at all the fields in the response and confirm that all required fields were there, and all data types were also correct. This is a powerful way to validate the schema contract.

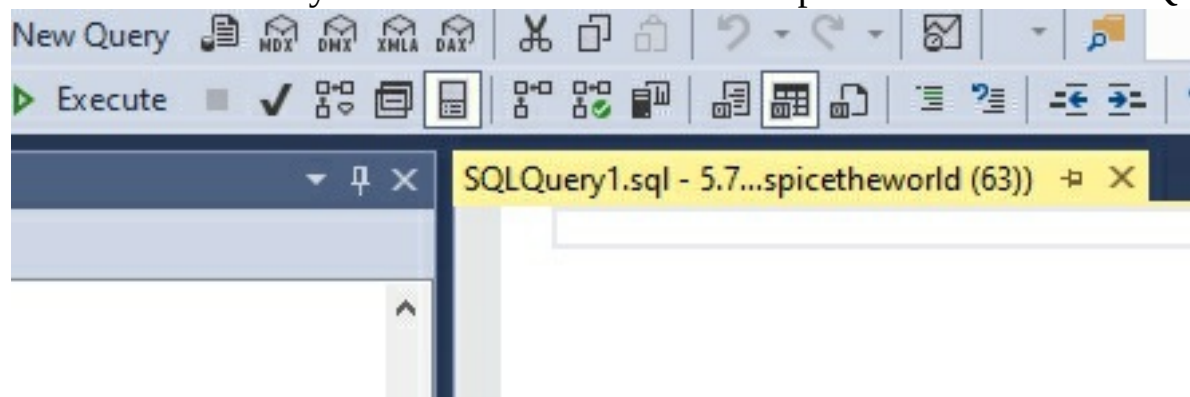
Structured Query Language

Structured Query Language (from now on referred to as SQL) is the standard language for relational database management systems. SQL statements are used to perform tasks such as add or update data on a database or retrieve data from a database. Some common relational database management systems that use SQL are Oracle, MySQL, Microsoft SQL Server, Access and PostGre SQL. Although most database systems use SQL, most of them also have their own additional proprietary extensions that are usually only valid on their system. However, the standard SQL commands such as Insert, Select, Update, Delete, Create and Drop can be used on all these systems. A competent Agile QA should have basic knowledge of SQL to perform their daily testing tasks. This will enable you to add and alter test data, refresh databases and understand the end results of tests that manipulate such data. This chapter will provide you with an introduction to the basics of each of these commands as well as allow you to put them to practice using the SQL Interpreter.

For these examples, I will be using a MSSQL database however the commands will work on any of the other systems with little or no alteration.

First, we need to create a new Database. This is achieved by the “CREATE DATABASE” command. For these examples we will assume you have Microsoft SQL Management Studio or something similar installed and you have access to a MSSQL server. So, to create the database follow these steps with Management Studio:

Connect to your database and open a New Query



Run this command - CREATE DATABASE spicetheworld;

This will create an empty database called spicetheworld. You can of course

use a different name if desired.

Next, you will need to create some tables within this new database. So, let us first start with a table to hold records on motorbikes. So now execute this command

```
CREATE TABLE MotorBikes (  
    Id int NOT NULL IDENTITY(1,1),  
    Reg NVARCHAR(20) NOT NULL ,  
    Vin NVARCHAR(40) NOT NULL ,  
    MainColour NVARCHAR(45) NULL ,  
    EngineCC NVARCHAR(10) NULL ,  
    PRIMARY KEY (Id)  
);
```

This will create a new table which is ready for data. The 'Id' field is the main index field and is automatically populated with a unique number every time a new record is created. Indexes are an especially important element of tables, especially if you are dealing with thousands or millions of records. They should be used for columns which are unique elements, used to link tables or used commonly for searching. However not every column should be indexed, there is a drawback, the more indexes the larger the database and eventually, the speed gained by using indexes will be lost. If you view the design of this new table, you will observe something like this.

Column Name	Data Type	Allow Nulls
Id	int	<input type="checkbox"/>
Reg	nvarchar(20)	<input type="checkbox"/>
Vin	nvarchar(40)	<input type="checkbox"/>
MinColour	nvarchar(45)	<input checked="" type="checkbox"/>
EngineCC	nvarchar(10)	<input checked="" type="checkbox"/>

Whenever possible use columns on the type [Int] for indexes and try to avoid using Nvarchar. Indexing by text will generate large and slow indexes. Note the 'spicetheworld' entry, this is the name of the Schema I am using for these examples, yours will be different.

You should also note that it is possible to create 1 table from all or part of an existing table, for example:

```
CREATE TABLE TestTable AS SELECT customername, contactname FROM customers;
```

Right now, let us put some data into this table, little can be done with empty tables after all. So, let us execute these commands next.

```
INSERT INTO spicetheworld.motorbikes (Reg, Vin, MainColour, EngineCC) VALUES ('X111WWW', '123456', 'Black', '500');
```

```
INSERT INTO spicetheworld.motorbikes (Reg, Vin, MainColour, EngineCC) VALUES ('RE15EWE', '78901', 'Rust', '750');
```

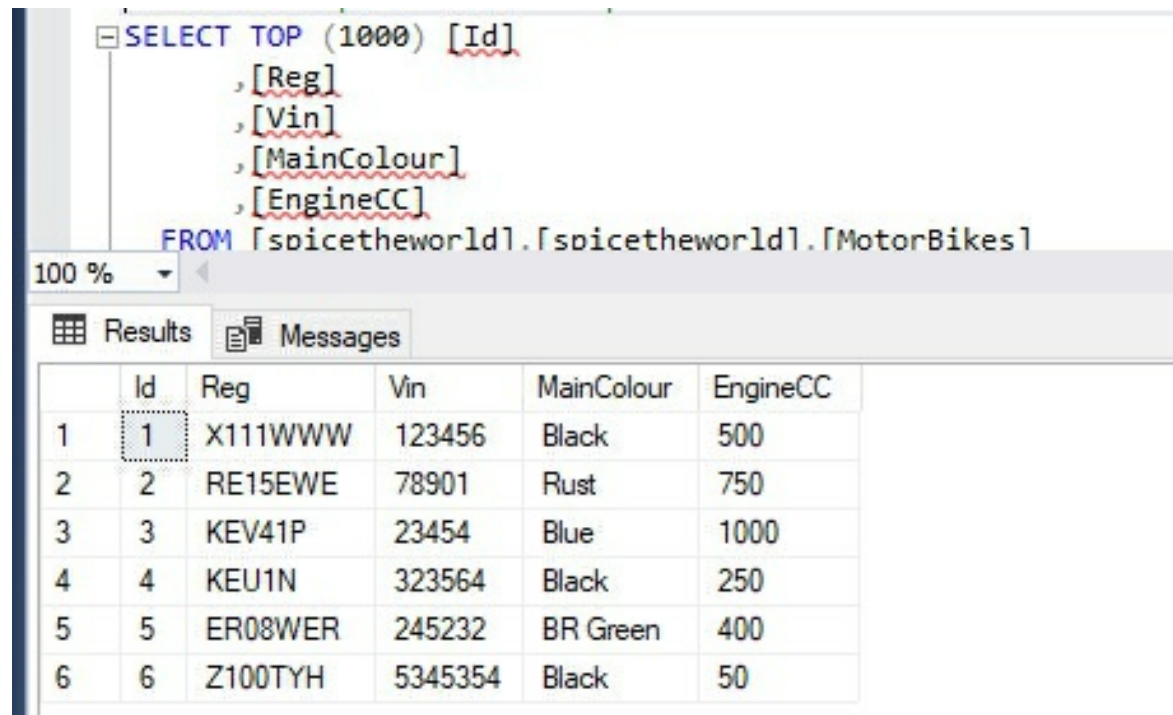
```
INSERT INTO spicetheworld.motorbikes (Reg,Vin,MainColour,EngineCC) VALUES ('KEV41P', '23454', 'Blue', '1000');
```

```
INSERT INTO spicetheworld.motorbikes (Reg,Vin,MainColour,EngineCC) VALUES ('KEU1N', '323564', 'Black', '250');
```

```
INSERT INTO spicetheworld.motorbikes (Reg,Vin,MainColour,EngineCC) VALUES ('ER08WER', '245232', 'BR Green', '400');
```

```
INSERT INTO spicetheworld.motorbikes (Reg,Vin,MainColour,EngineCC) VALUES ('Z100TYH', '5345354', 'Black', '50');
```

This will create 6 records, feel free to add any additional records if you wish, you will note we did not add any data for the Id column, remember this is an auto-increment column as shown here.



The screenshot shows a SQL query window with the following text:

```
SELECT TOP (1000) [Id]
, [Reg]
, [Vin]
, [MainColour]
, [EngineCC]
FROM [spicetheworld].[spicetheworld].[MotorBikes]
```


Below the query window, the 'Results' tab is active, displaying a table with 6 rows and 6 columns. The first row is highlighted with a dotted border.

	Id	Reg	Vin	MainColour	EngineCC
1	1	X111WWW	123456	Black	500
2	2	RE15EWE	78901	Rust	750
3	3	KEV41P	23454	Blue	1000
4	4	KEU1N	323564	Black	250
5	5	ER08WER	245232	BR Green	400
6	6	Z100TYH	5345354	Black	50

Now we can do little with one table so now create these tables and insert the test data as well

```
CREATE TABLE spicetheworld.customers (  
  Id INT NOT NULL IDENTITY(1,1),  
  Fullname NVARCHAR(45) NULL ,
```



Address1 NVARCHAR(45) NULL ,
 Address2 nVARCHAR(45) NULL ,
 Town NVARCHAR(45) NULL ,
 County NVARCHAR(45) NULL ,
 Postcode NVARCHAR(15) NULL ,
 Email NVARCHAR(75) NULL ,
 Mobile NVARCHAR(45) NULL ,
 PRIMARY KEY (Id));

Column Name	Data Type	Allow Nulls
 Id	int	<input type="checkbox"/>
Fullname	nvarchar(45)	<input checked="" type="checkbox"/>
Address1	nvarchar(45)	<input checked="" type="checkbox"/>
Address2	nvarchar(45)	<input checked="" type="checkbox"/>
Town	nvarchar(45)	<input checked="" type="checkbox"/>
County	nvarchar(45)	<input checked="" type="checkbox"/>
Postcode	nvarchar(15)	<input checked="" type="checkbox"/>
Email	nvarchar(75)	<input checked="" type="checkbox"/>
Mobile	nvarchar(45)	<input checked="" type="checkbox"/>



INSERT INTO spicetheworld.customers (Fullname, Address1, Address2, Town, County, Postcode, Email, Mobile) VALUES ('Fred Blogs', '123 This Road', 'Hampstead', 'London', 'Middx', 'NW3 2DD', 'fred@bloggs.com', '1212213123');

INSERT INTO spicetheworld.customers (Fullname, Address1, Address2, Town, County, Postcode, Email, Mobile)VALUES ('Kev Martin', '44 Queens Road', 'Copnor', 'Portsmouth', 'Hants', 'PO1 4RT', 'kev@kevin.com', '2323123121');

INSERT INTO spicetheworld.customers (Fullname, Address1, Address2, Town, County, Postcode, Email, Mobile)VALUES ('Mick Jagger', '123 Angie Street', ' ', 'New York', 'New York', 'NY', 'mick@stones.com', '4234231211');

 SELECT TOP (1000) [Id]
 , [Fullname]
 , [Address1]
 , [Address2]
 , [Town]
 , [County]
 , [Postcode]
 , [Email]
 , [Mobile]
 FROM [spicetheworld].[spicetheworld].[customers]

0 %

 Results  Messages

	Id	Fullname	Address1	Address2	Town	County	Postcode	Email	Mobile
1	1	Fred Blogs	123 This Road	Hampstead	London	Middx	NW3 2DD	fred@bloggs.com	1212213123
2	2	Kev Martin	44 Queens Road	Copnor	Portsmouth	Hants	PO1 4RT	kev@kevin.com	2323123121
3	3	Mick Jagger	123 Angie Street		New York	New York	NY	mick@stones.com	4234231211

So now we have a table of motorbikes and a table of customers. Please remember this is only demonstration data, in a real system there would be a lot more fields and (hopefully) a lot more records. Now let us create one more table, this is a table that will link customers to motorbikes they have purchased, and this will include the date of sale and how much they paid.

This is the table.

```
CREATE TABLE spicetheworld.BikeSales (  
  Id INT NOT NULL IDENTITY(1,1),  
  SaleDate DATE NOT NULL ,  
  SaleValue FLOAT NOT NULL ,  
  BikeId INT NOT NULL ,  
  CustomerId INT NOT NULL ,  
  PRIMARY KEY (Id)  
);
```

Create the table and then add this sales data.

```
INSERT INTO spicetheworld.bikeSales (saleDate,saleValue,bikeId,customerId) VALUES ('2017-01-01', 500, 1, 1);
```

```
INSERT INTO spicetheworld.bikeSales (saleDate,saleValue,bikeId,customerId) VALUES ('2017-02-02', 700, 2, 2);
```

```
INSERT INTO spicetheworld.bikeSales (saleDate,saleValue,bikeId,customerId) VALUES ('2017-03-03', 1200, 3, 3);
```

```
INSERT INTO spicetheworld.bikeSales (saleDate,saleValue,bikeId,customerId) VALUES ('2017-04-04', 600, 4, 1);
```

```
INSERT INTO spicetheworld.bikeSales (saleDate,saleValue,bikeId,customerId) VALUES ('2017-05-05', 600, 5, 2);
```

```
INSERT INTO spicetheworld.bikeSales (saleDate,saleValue,bikeId,customerId) VALUES ('2017-06-06', 999, 6, 3);
```

```
INSERT INTO spicetheworld.bikeSales (saleDate,saleValue,bikeId,customerId) VALUES ('2017-0808', 650, 5, 1);
```

Once executed you will find a new table with the following data

The screenshot shows a SQL query in the query editor and its results in the Results pane. The query is a SELECT TOP (1000) statement with columns Id, SaleDate, SaleValue, BikeId, and CustomerId, sourced from the spicetheworld database's BikeSales table. The Results pane shows a table with 7 rows and 5 columns: Id, SaleDate, SaleValue, BikeId, and CustomerId. The first row is highlighted.

```

SELECT TOP (1000) [Id]
, [SaleDate]
, [SaleValue]
, [BikeId]
, [CustomerId]
FROM [spicetheworld].[spicetheworld].[BikeSales]

```

	Id	SaleDate	SaleValue	BikeId	CustomerId
1	1	2017-01-01	500	7	1
2	2	2017-02-02	700	8	2
3	3	2017-03-03	1200	10	3
4	4	2017-04-04	600	12	1
5	5	2017-05-05	600	7	2
6	6	2017-06-06	999	9	3
7	8	2017-08-08	650	8	1

Now at first glance, this data appears difficult to understand. While you have a Sale Date and a Sale Value there is no data relating to the Customer or the Motorbike. This is of course by design. By creating this table we have reduced data duplication to a minimum because the required data is already stored in other tables, to view the motorbike and customer details you simply need to link the tables together using SQL like this.

```

SELECT bs.SaleDate, bs.SaleValue, mb.Reg, cs.Fullname, cs.Email, cs.Mobile
FROM spicetheworld.BikeSales bs
INNER JOIN spicetheworld.Motorbikes mb ON mb.Id = bs.BikeId
INNER JOIN spicetheworld.Customers cs ON cs.Id = bs.CustomerId;

```

This will return more useful data, thus.

	SaleDate	SaleValue	Reg	Fullname	Email	Mobile
1	2017-01-01	500	X111WWW	Fred Blogs	fred@bloggs.com	1212213123
2	2017-02-02	700	RE15EWE	Kev Martin	kev@kevin.com	2323123121
3	2017-03-03	1200	KEV41P	Mick Jagger	mick@stones.com	4234231211
4	2017-04-04	600	KEU1N	Fred Blogs	fred@bloggs.com	1212213123
5	2017-05-05	600	ER08WER	Kev Martin	kev@kevin.com	2323123121
6	2017-06-06	999	Z100TYH	Mick Jagger	mick@stones.com	4234231211
7	2017-08-08	650	ER08WER	Fred Blogs	fred@bloggs.com	1212213123

So, by joining together three tables, we are now able to see who purchased what bike, how much they paid and the date of the sale. All of this has been achieved with no data duplication at all.

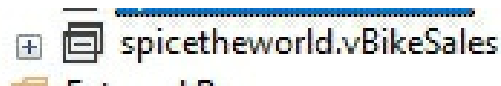
There is an easier way of returning such data if it is required regularly. Rather

than type in the SQL every time (and this is a remarkably simple example) you can store it as a read-only view and simply execute that.

To create a view that will replicate the above SQL type this script in and execute.

```
CREATE VIEW vBikeSales AS
SELECT bs.SaleDate, bs.SaleValue, mb.Reg, cs.Fullname, cs.Email, cs.Mobile
FROM spicetheworld.BikeSales bs
INNER JOIN spicetheworld.Motorbikes mb ON mb.Id = bs.BikeId
INNER JOIN spicetheworld.Customers cs ON cs.Id = bs.CustomerId;
```

You should now see a new View called vbikeSales like below.



Now every time you want to view the data in this format all that is required is you execute the view with this command `SELECT * FROM spicetheworld.vBikeSales;`

So that is it, this was a quite easy introduction to the bases of SQL for anyone who has no or little experience with the language. As I have already stated every agile QA should have at least a basic knowledge of this area, however hopefully you will take this further and continue to study this interesting subject area. To help you get start here follows some more examples of SQL (Structured Query Language).

First some more of the most common SQL commands

- `SELECT` - extracts data from a database
- `UPDATE` - updates data in a database
- `DELETE` - deletes data from a database

First we have the **Select** command, you have already seen this in action above and in its most simple form we have

```
Select * from table_name;
```

This will select all rows and all columns from the selected table in the order they were stored.

You can also just select a subset of the columns

```
Select name, email from table_name;
```

Next we have `Select Distinct`, this command will only return distinct values, for example if you table has 3 Fred Bloggs then the command below will only return 1 record.

Select Distinct fullName from Customers;

However this will return a count for each distinct name

Select Count(*) As DistinctCustomers From (Select Distinct fullName from Customers);

The next SQL clause is the **Where** clause, this is used to extraxt records that match a specified condition. This example will only return records which match the vehicle reg.

SELECT * FROM spicetheworld.vBikeSales WHERE Reg = 'KEV41P'

	SaleDate	SaleValue	Reg	Fullname	Email	Mobile
1	2017-03-03	1200	KEV41P	Mick Jagger	mick@stones.com	4234231211

This is a numeric clause match and the result will be the same

SELECT * FROM spicetheworld.vBikeSales WHERE SaleValue= 1200

Now let us look at the And, Or and Not operators

SELECT * FROM spicetheworld.vBikeSales WHERE Reg = 'KEV41P' AND SalesValue=1200

Here only records that fully match the Reg and SalesValue will be returned

SELECT * FROM spicetheworld.vBikeSales WHERE Reg = 'KEV41P' OR SalesValue=1200

Here records that fully match the Reg or SalesValue will be returned

SELECT * FROM spicetheworld.vBikeSales WHERE NOT Reg = 'KEV41P'

Here records were the reg does not match 'KEV41P' will be returned

These operators can also be combined thus

SELECT * FROM spicetheworld.vBikeSales WHERE Reg = 'KEV41P' AND (SaleValue=1200 or SaleValue=10)

SELECT * FROM spicetheworld.vBikeSales WHERE NOT SaleValue=500 AND NOT Reg = 'KEU1N'

Another useful command is ORDER BY, with this you can specify exactly in what order the records will be returned, so here are a few examples. ASC means soft ascending while DESC means descending.

SELECT * FROM spicetheworld.vBikeSales ORDER BY SaleDate ASC

	SaleDate	SaleValue	Reg	Fullname	Email	Mobile
1	2017-01-01	500	X111WWW	Fred Blogs	fred@bloggs.com	1212213123
2	2017-02-02	700	RE15EWE	Kev Martin	kev@kevin.com	2323123121
3	2017-03-03	1200	KEV41P	Mick Jagger	mick@stones.com	4234231211
4	2017-04-04	600	KEU1N	Fred Blogs	fred@bloggs.com	1212213123
5	2017-05-05	600	ER08WER	Kev Martin	kev@kevin.com	2323123121
6	2017-06-06	999	Z100TYH	Mick Jagger	mick@stones.com	4234231211
7	2017-08-08	650	ER08WER	Fred Blogs	fred@bloggs.com	1212213123

SELECT * FROM spicetheworld.vBikeSales ORDER BY FullName, Reg ASC

	SaleDate	SaleValue	Reg	Fullname	Email	Mobile
1	2017-08-08	650	ER08WER	Fred Blogs	fred@bloggs.com	1212213123
2	2017-04-04	600	KEU1N	Fred Blogs	fred@bloggs.com	1212213123
3	2017-01-01	500	X111WWW	Fred Blogs	fred@bloggs.com	1212213123
4	2017-05-05	600	ER08WER	Kev Martin	kev@kevin.com	2323123121
5	2017-02-02	700	RE15EWE	Kev Martin	kev@kevin.com	2323123121
6	2017-03-03	1200	KEV41P	Mick Jagger	mick@stones.com	4234231211
7	2017-06-06	999	Z100TYH	Mick Jagger	mick@stones.com	4234231211

SELECT * FROM spicetheworld.vBikeSales ORDER BY FullName DESC, Reg ASC

	SaleDate	SaleValue	Reg	Fullname	Email	Mobile
1	2017-03-03	1200	KEV41P	Mick Jagger	mick@stones.com	4234231211
2	2017-06-06	999	Z100TYH	Mick Jagger	mick@stones.com	4234231211
3	2017-05-05	600	ER08WER	Kev Martin	kev@kevin.com	2323123121
4	2017-02-02	700	RE15EWE	Kev Martin	kev@kevin.com	2323123121
5	2017-08-08	650	ER08WER	Fred Blogs	fred@bloggs.com	1212213123
6	2017-04-04	600	KEU1N	Fred Blogs	fred@bloggs.com	1212213123
7	2017-01-01	500	X111WWW	Fred Blogs	fred@bloggs.com	1212213123

Now let us look at the UPDATE command. This command is used to modify existing records in a table, for example.

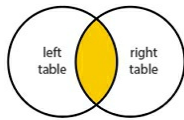
UPDATE spicetheworld SET name=fred, email=fred@bloggs.com WHERE id=3

If a record in table spicetheworld with an id of 3 exists then the name and email fields for said record will be updated.

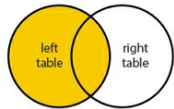
Next we have the DELETE command, this does exactly what it says on the time, so for example

DELETE FROM spicetheworld WHERE id=3

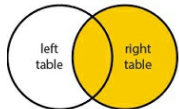
Finally in this section let's have a look at the JOIN command. This command is used to combine rows from two or more tables based on a related column between them. We have already used them in previous examples but let us look at the most common different types of joins available.



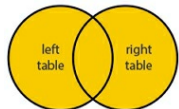
(INNER) JOIN: This method will return records that have matching values in both of the related tables.



LEFT (OUTER) JOIN: This method returns all records from the left table, and only the matched records from the right table.



RIGHT (OUTER) JOIN: Returns all records from the right table, and only the matched records from the left table.



FULL (OUTER) JOIN: Returns all records when there is a match in either the left or right table.

So that is your introduction to SQL, this now takes us very neatly into our next subject, Stored Procedures.

Stored Procedures

Stored Procedures enables the placing of database-intensive operations into stored procedures. This is a powerful tool which lets you define an API for your database application. You can then reuse this API across multiple applications and multiple programming languages whenever it is required. This ability also avoids duplicating database code and using SQL within your code thus saving time and effort when you make updates due to future schema changes.

Now let us look at one of our SQL statements from the previous chapter and then create a Stored Procedure for it.

```
SELECT bs.saleDate, bs.saleValue, mb.reg, cs.fullname, cs.email, cs.mobile
FROM spicetheworld.bikeSales bs
INNER JOIN spicetheworld.motorbikes mb ON mb.id = bs.bikeId
INNER JOIN spicetheworld.customers cs ON cs.id = bs.customerId
```

Remember it, good. Now let us create a stored procedure for this SQL

```
DELIMITER //
CREATE PROCEDURE SelectAllBikeSales
BEGIN
    SELECT bs.saleDate, bs.saleValue, mb.reg, cs.fullname, cs.email, cs.mobile
    FROM spicetheworld.bikeSales bs
    INNER JOIN spicetheworld.motorbikes mb ON mb.id = bs.bikeId
    INNER JOIN spicetheworld.customers cs ON cs.id = bs.customerId
END //
DELIMITER ;
```

When this SQL is executed then the stored procedure will be created. So, to run the new procedure simply call it thus:

CALL SelectAllBikeSales;

A nice and easy example, so what about if we just want to return bike sales history of one customer, again easy to do with a stored procedure.

```
DELIMITER //
CREATE PROCEDURE SelectAllBikeSalesForCustomer
(IN email CHAR(75))
BEGIN
    SELECT bs.saleDate, bs.saleValue, mb.reg, cs.fullname, cs.email, cs.mobile
    FROM spicetheworld.bikeSales bs
    INNER JOIN spicetheworld.motorbikes mb ON mb.id = bs.bikeId
    INNER JOIN spicetheworld.customers cs ON cs.id = bs.customerId
    WHERE cs.email = email
```



```
END //  
DELIMITER ;
```

When this SQL is executed then the stored procedure will be created. So, to run the new procedure simply call it thus:

```
CALL SelectAllBikeSalesForCustomer('kev@kevsbox.com');
```

Once again, a nice easy example. Let us do one more, in this example, we want all sales for the given vehicle registration.

```
DELIMITER //  
CREATE PROCEDURE SelectAllBikeSalesForReg  
(IN reg CHAR(20))  
BEGIN  
    SELECT bs.saleDate, bs.saleValue, mb.reg, cs.fullname, cs.email, cs.mobile  
    FROM spicetheworld.bikeSales bs  
    INNER JOIN spicetheworld.motorbikes mb ON mb.id = bs.bikeId  
    INNER JOIN spicetheworld.customers cs ON cs.id = bs.customerId  
    WHERE mb.reg = reg  
END //  
DELIMITER ;
```

So, there you go, a nice easy introduction into stored procedures. These are very simple examples and this powerful tool can be used to create some very complex procedures but that is beyond the scope of this book but there are plenty of very good publications for MySql and Microsoft SQL out there for further reading.

Continuous Integration and Deployment

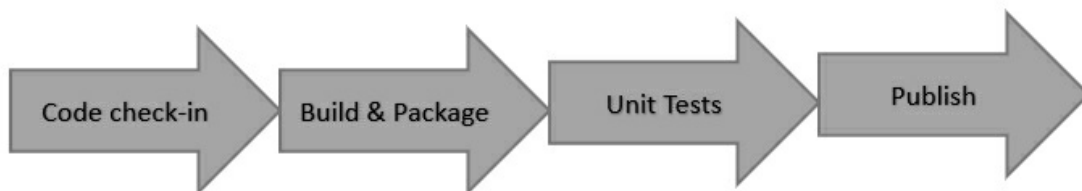
Automation is a good aid when trying to achieve Continuous Integration and Deployment and as part of the overall scheme, this practice can vastly improve the quality of code and the speed of development. Continuous integration (CI) is the practice of regularly integrating and testing your solution to incorporate changes made to its definition. These changes can include updating the source code, changing database schema or simply updating a text based configuration file.

For the best results when one or more changes are checked into your configuration management system, the solution should then be rebuilt and recompiled, retested and finally any code or schema analysis performed on it. If it is impractical to do this every time a change is saved you should strive to do so at least once if not several times a day, that is the essence of continuous integration.

Moving this process one step forward continuous deployment (CD) enhances CI by automatically deploying successful builds. For example, when the build is successful on a programmer's workstation, then the team may automatically deploy their changes to the project integration environment, which would invoke the CI system there. A successful integration in that environment could trigger an automatic deployment into another environment and so on.

On a developer's workstation, the integration job could run by example at specific times, perhaps once an hour, or better every time that they check in something that is part of the build. The whole process of continuously integrating a developer's code with the rest of a team's code and then running automated test regressions in an integration environment is the essence of CI.

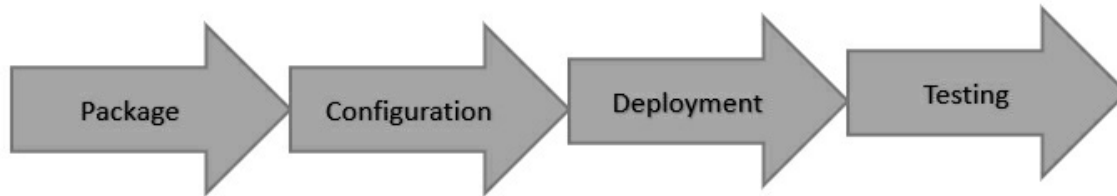
Continuous Integration



This is a critical part of agile to ensure integration is done right. CI always

ensures high-quality working software, and CD ensures that the software is running in the right place. When used with Selenium or another good automation tool good testing results can also soon be achieved.

Continuous Deployment



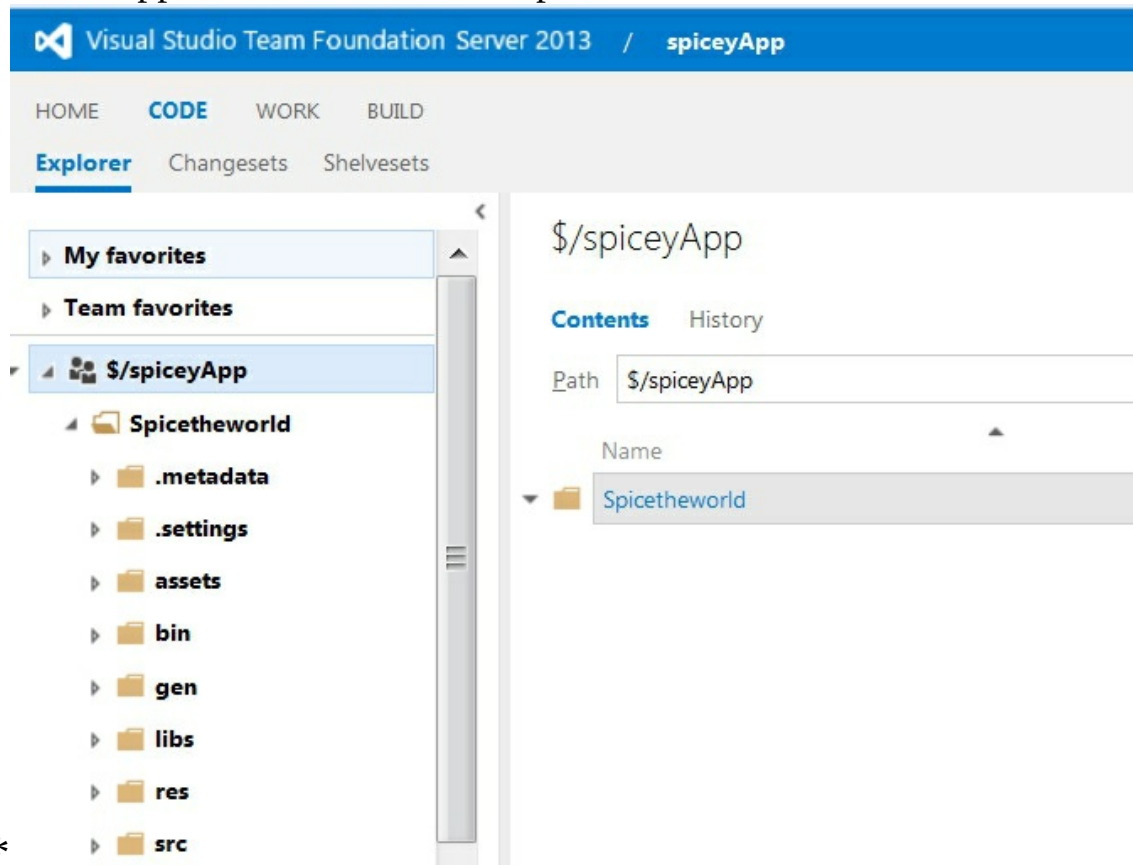
The longer a branch of code remains checked out on a programmer's computer, the greater the risk of multiple integration conflicts and failures when the branch is finally reintegrated into the main trunk. When programmers submit code to the repository, they should first update their code to reflect the changes in the repository since they took their local copy. The more changes the repository contains, the more work programmers must do before submitting their changes.

There is a risk that eventually the repository may become so different from the programmer's local copy that the team can enter what is sometimes referred to as "merge hell", or "integration hell". This is where the time it takes to integrate exceeds the time it took to make their original changes and conflicts are everywhere. This is where continuous integration becomes a critical practice. It involves integrating early and often, to avoid the pitfalls of "integration hell".

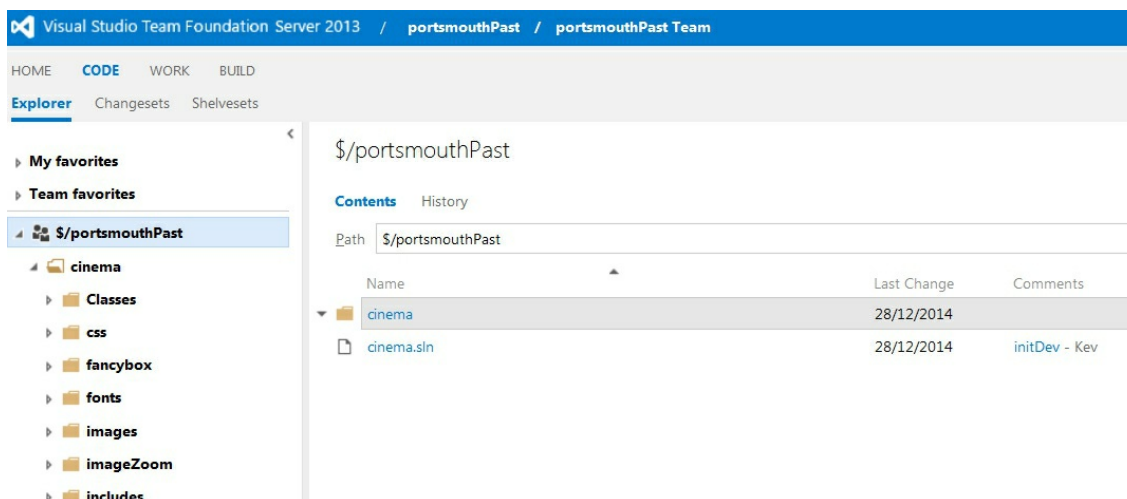
Regular integration requires a reliable repository where the main trunk of code and any development branches are stored. Offsite backups of this code are also essential, should the unthinkable ever happen and the source server should go down or go missing at least code would still exist and a few hours work may be lost rather than a few years.

There are many options for repository management on the market today. I prefer the built-in repository options that are available on Team Foundation Server. This multi functioned and well-designed application allows for safe storage of both Visual Studio Code and Eclipse Code. Shown below is the repository for an Android application which is designed in Eclipse and all changes are stored within Team Foundation Server. Below is an example screenshot of this package. This example shows the source code structure for

an android application written in Eclipse.



By contrast, the next screenshot is from the same Team Foundation Server but this time it is a visual studio ASPX/C# application.



In the next chapter, we will look further at CI/CD with a real project which uses the power of Azure and how this can help the agile tester in their quest to become an expert QA Tester.

CI/CD with Azure

Before we get going to may already know about Azure, of not here is a brief introduction. **Microsoft Azure**, commonly referred to as Azure, is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through Microsoft-managed data centres. The service provides software as a service (SaaS), platform as a service (PaaS) and infrastructure as a service (IaaS) and supports many different programming languages, tools, and frameworks, including both Microsoft-specific and third-party software and systems.

To sign-up or sign-in to Azure simply go to this URL and get yourself sorted.
<https://portal.azure.com/>.

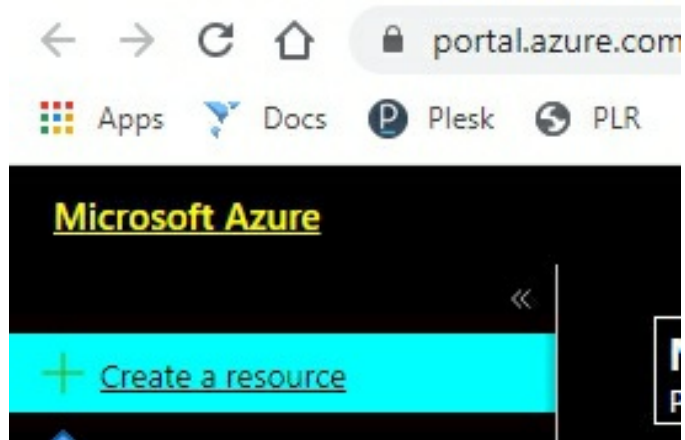
For general information about the services try this URL
<https://azure.microsoft.com/en-gb/>.

For the purpose of this demonstration I will be linking with my stored code at Azure DevOps (dev.azure.com/myAccount/), you may also have such an account if not then why not sign up for one, it is free.

Here I will walk through the steps that will show you the true power of the dark side, sorry I mean Azure DevOps. If the then feel the desire to learn this further there are plenty of books and websites that can offer a lot of good, helpful information on this subject.

So once you are all setup and ready to go follow my steps here or simply just read through.

Step 1 – Create Storage Accounts. In your azure portal select this link.



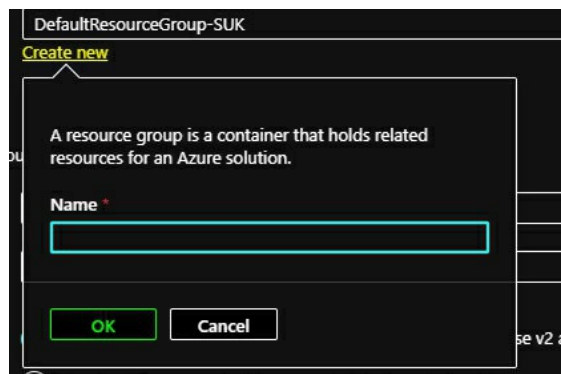
Now either select from the list or type in storage in the search box.



Then select storage account->Create.



Next click the Create new link



And type in a label of your choice. Then complete the page something like mine below.

Project details

Select the subscription in which to create the new storage account. Choose a new or existing resource group to organize and manage your storage account together with other resources.

Subscription *

Resource group * [Create new](#)

Instance details

If you need to create a legacy storage account type, please click [here](#).

Storage account name ⓘ *

Region ⓘ *


Performance ⓘ *

☒ Standard: Recommended for most scenarios (general-purpose v2 account)


☐ Premium: Recommended for scenarios that require low latency.

Redundancy ⓘ *

Keep all other options as per default and step through to create, when complete you will see a message like this on the following page.



Your deployment is complete



Deployment name: spicedevstorageaccount_1620314298120

Subscription: [Kevsbox Azure](#)

Resource group: [SpiceDevStorageGroup](#)

Deployment details [\(Download\)](#)

Next steps


[Go to resource](#)

If you then go to resource you will see this page.

[Open in Explorer](#)
[Delete](#)
[Move](#)
[Refresh](#)
[Feedback](#)

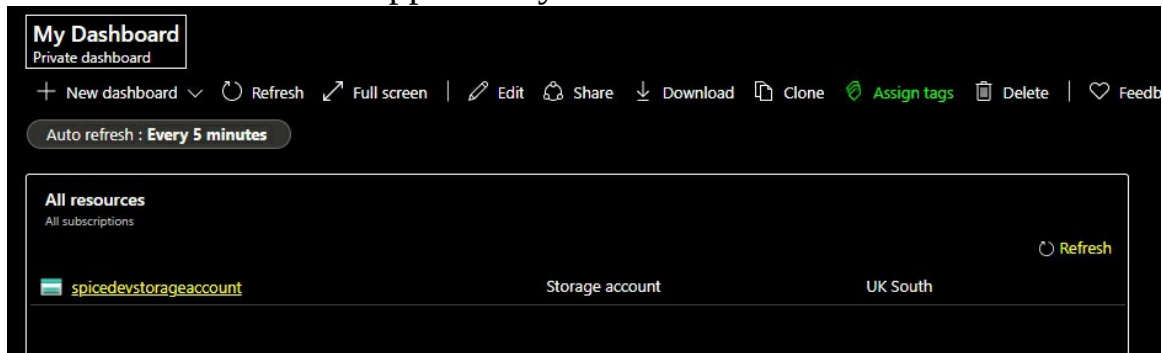
Microsoft recommends upgrading to the new alerts platform to ensure no interruptions in your alerts. Classic alerts will be retired starting in 2021. Upgrade to the new alerts platform. [Learn more](#)

Essentials

Resource group (change)	: SpiceDevStorageGroup	Performance/Access tier	: Standard/Hot
Location	: UK South	Replication	: Locally-redundant storage (LRS)
Subscription (change)	: Kevsbox Azure	Account kind	: StorageV2 (general purpose v2)
Subscription ID	: 1877d6e8-9377-4b8b-86be-94d1740a2ed5	Provisioning state	: Succeeded 
Disk state	: Available	Created	: 06/05/2021, 16:18:20

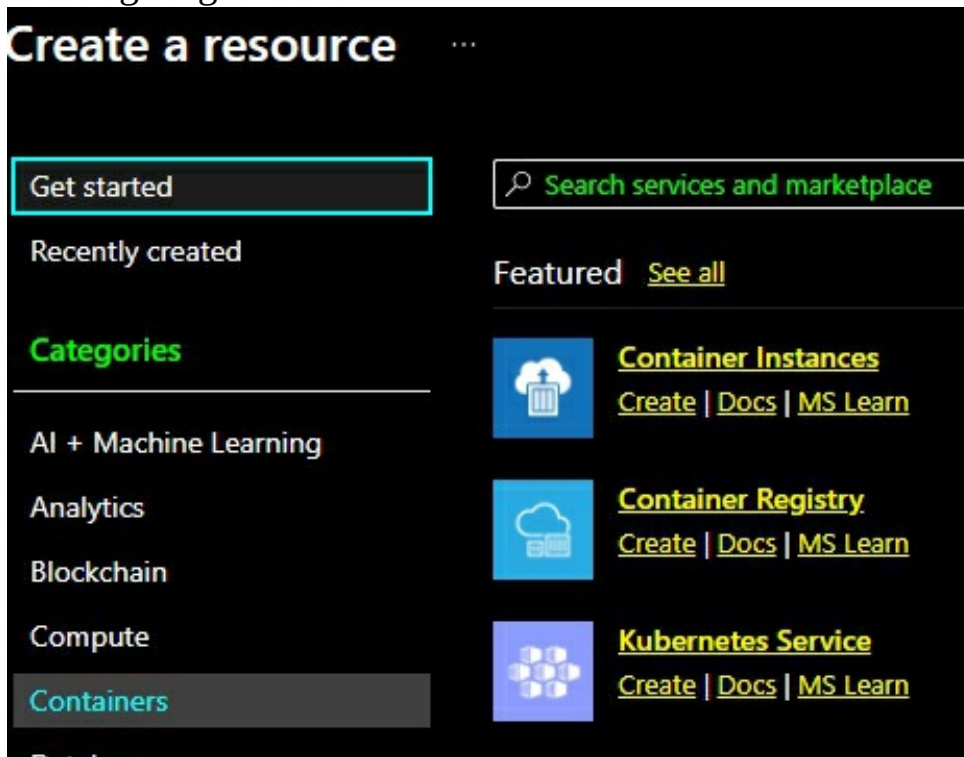
Tags [\(change\)](#) :

The resource will also appear on your dashboard.



Step 2. Create the container registry

Once again go to Create a resource. This time locate Container registry.



Select create and step through the process. Resource group should be the one you created in Step 1. Then complete the remaining options.

Project details

Subscription * Kevsbox Azure ▼

Resource group * SpiceDevStorageGroup ▼
[Create new](#)

Instance details

Registry name * SpiceDevRegistry ✓
.azurecr.io

Location * UK South ▼

Availability zones ⓘ ☒ Enabled

ℹ Availability zones are enabled on premium registries and in regions that support availability zones. [Learn more](#)

SKU * ⓘ Basic ▼

Step through the remaining pages with default options and create.

✓ **Your deployment is complete**

📄 Deployment name: Microsoft.ContainerRegistry
Subscription: [Kevsbox Azure](#)
Resource group: [SpiceDevStorageGroup](#)

Start time: 5/6/2021, 4:25:11 PM
Correlation ID: 62a80b49-d34f-4a0d-b084-fef6eb7cdf8d

▼ **Deployment details** [\(Download\)](#)

^ **Next steps**

[Go to resource](#)

Go to the resource to confirm.

→ Move	🗑 Delete	⚙ Update
^ Essentials		
Resource group (change) :	SpiceDevStorageGroup	Login server : spicedevregistry.azurecr.io
Location :	UK South	Creation date : 5/6/2021, 4:25 PM GMT+1
Subscription (change) :	Kevsbox Azure	SKU : Basic
Subscription ID :	1877d6e8-9377-4b8b-86be-94d1740a2ed5	Provisioning state : Succeeded

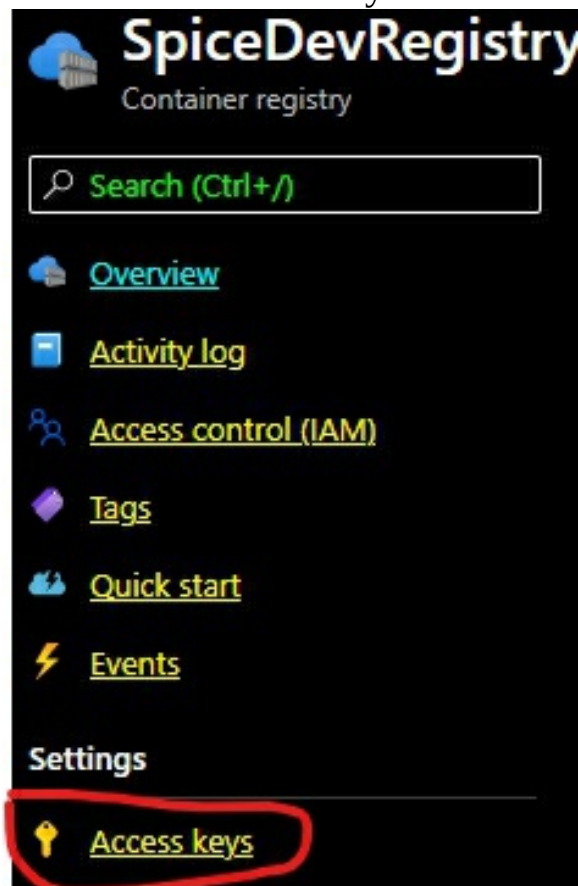
Finally check the dashboard.

📄 SpiceDevRegistry	Container registry	UK South
📄 spicedevstorageaccount	Storage account	UK South

Step 3. In this step we need to enable Admin user in the Container registry and create a Username with a password. So first select your Container registry from your dashboard.

All resources			...
All subscriptions			Refresh
 SpiceDevRegistry	Container registry	UK South	
 spicedevstorageaccount	Storage account	UK South	

Then select Access Keys.



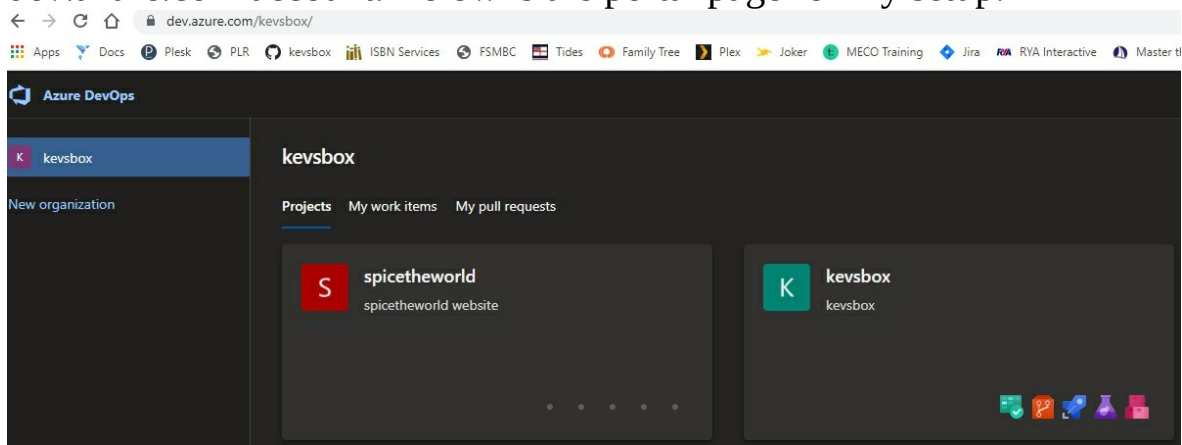
You will now need to enable Admin user

Registry name	<input type="text" value="SpiceDevRegistry"/>	
Login server	<input type="text" value="spicedevregistry.azurecr.io"/>	
Admin user ⓘ	<input type="checkbox"/> Disabled	

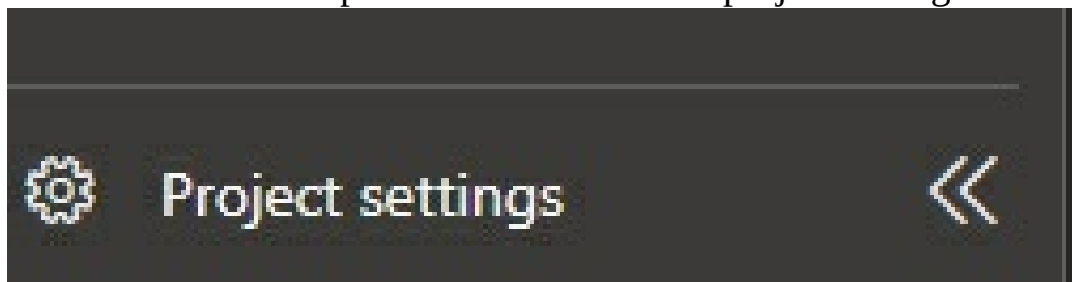
When enabled a Username and 2 passwords will be created, make a note of these before proceeding. Mine of course are hidden.

Registry name	<input type="text" value="SpiceDevRegistry"/>	
Login server	<input type="text" value="spicedevregistry.azurecr.io"/>	
Admin user ⓘ	<input checked="" type="checkbox"/> Enabled	
Username	<input type="text"/>	
Name	Password	Regenerate
password	<input type="password"/>	<input type="button" value="🔄"/>
password2	<input type="password"/>	<input type="button" value="🔄"/>

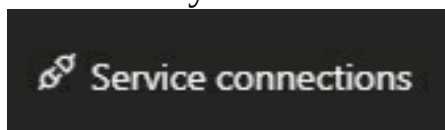
Step 4: Service Connections. For this step we now switch to the dev.azure.com account. Below is the portal page for my setup.



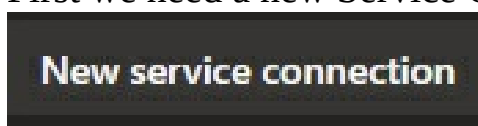
From here I select “spicetheworld” and then project settings.

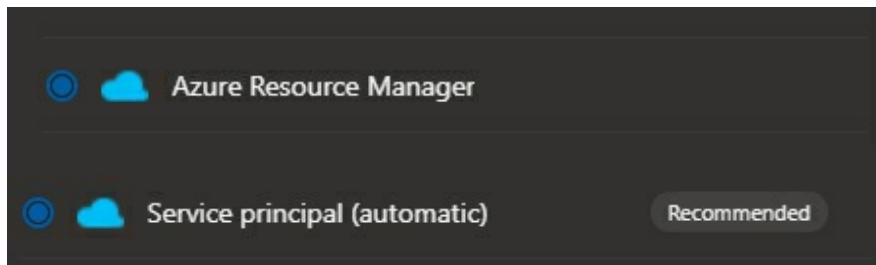


Followed by Service connections.



First we need a new Service Connection





On the next page you will probably be prompted to login to your Microsoft account, this is standard procedure and nothing to worry about. Once logged in you will be linked to your Azure account subscription. You will also be able to select the Resource group you created earlier.

Finally you will need to provide a Service connection name, the description is optional.

A dark-themed dialog box titled 'New Azure service connection' with a close button (X) in the top right corner. Below the title is the subtitle 'Azure Resource Manager using service principal (automatic)'. The 'Scope level' section has three radio buttons: 'Subscription' (selected), 'Management Group', and 'Machine Learning Workspace'. The 'Subscription' section has a dropdown menu showing 'Kevsbox Azure (1877d6e8-9377-4b8b-86be-94d1740a2ed5)'. The 'Resource group' section has a dropdown menu showing 'SpiceDevStorageGroup'. The 'Details' section has a 'Service connection name' text field containing 'SpiceDevServiceConnection' and a 'Description (optional)' text field containing 'Spice Dev Service Connection'. The 'Security' section has a checked checkbox labeled 'Grant access permission to all pipelines'.

Then click save, after a couple of moments your Service connection will be created. Now add a second Service connection, this time select type Docker registry.

New service connection

Choose a service or connection type

☐ Docker Host

☒ Docker Registry

Choose type Azure Container Registry, the rest is the same as before.

☐ Docker Hub ☐ Others ☒ Azure Container Registry

Subscription

Keysbox Azure (1877d6e8-9377-4b8b-86be-94d1740a2ed5) ▾

Azure container registry

SpiceDevRegistry ▾

Details

Service connection name

SpiceDevDockerConnection

Step 5: In this step we now create the pipeline.

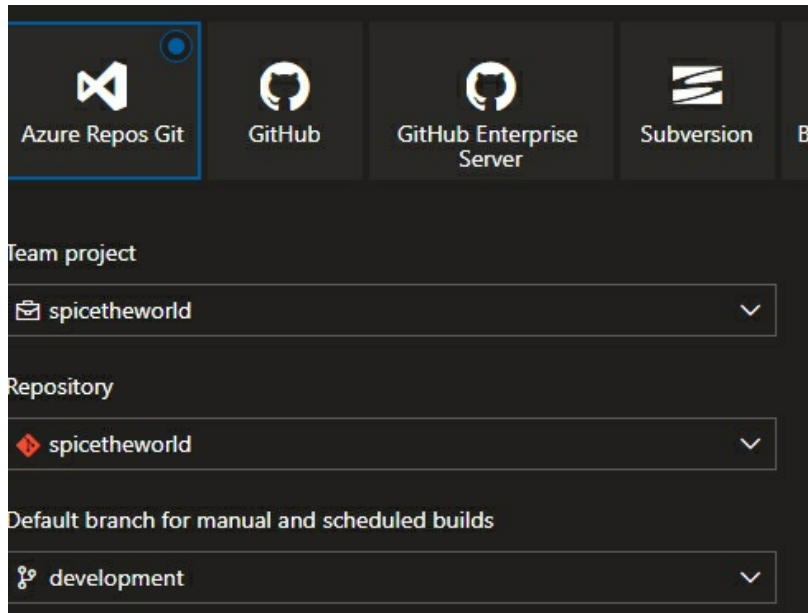
Pipelines

New pipeline

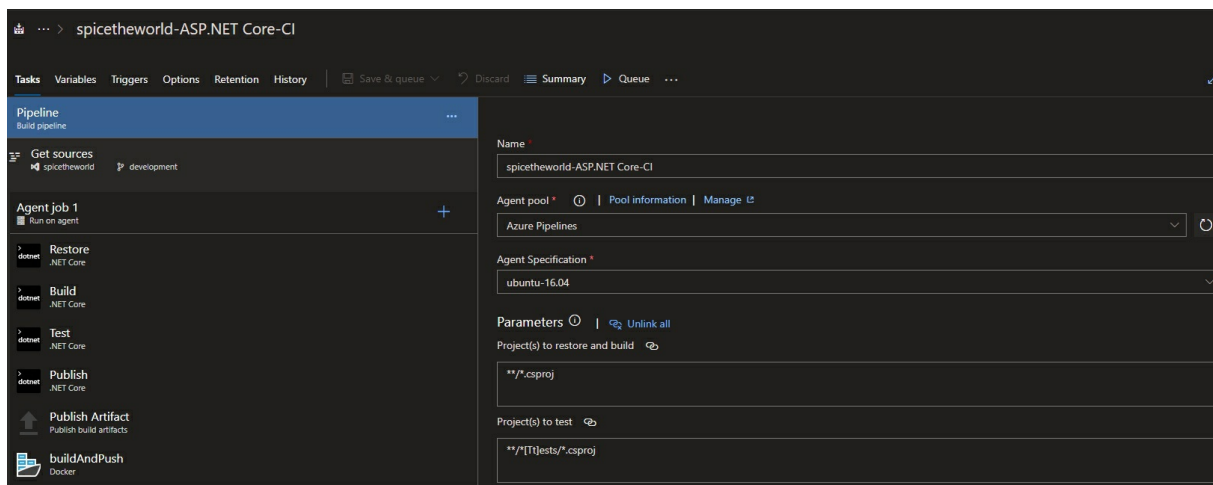
Select Use the classic editor

Use the classic editor to create a pipeline without YAML.

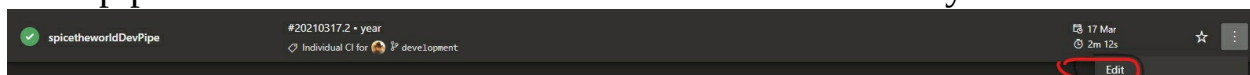
Next select your project, repository and branch.



Next you will select your template, this will be decided by the project type, in my example it is ASP.NET Core because this matches my project.



To create click Save & queue->Save and Run. At this point you will have a basic pipeline but now we need to enhance the functionality.

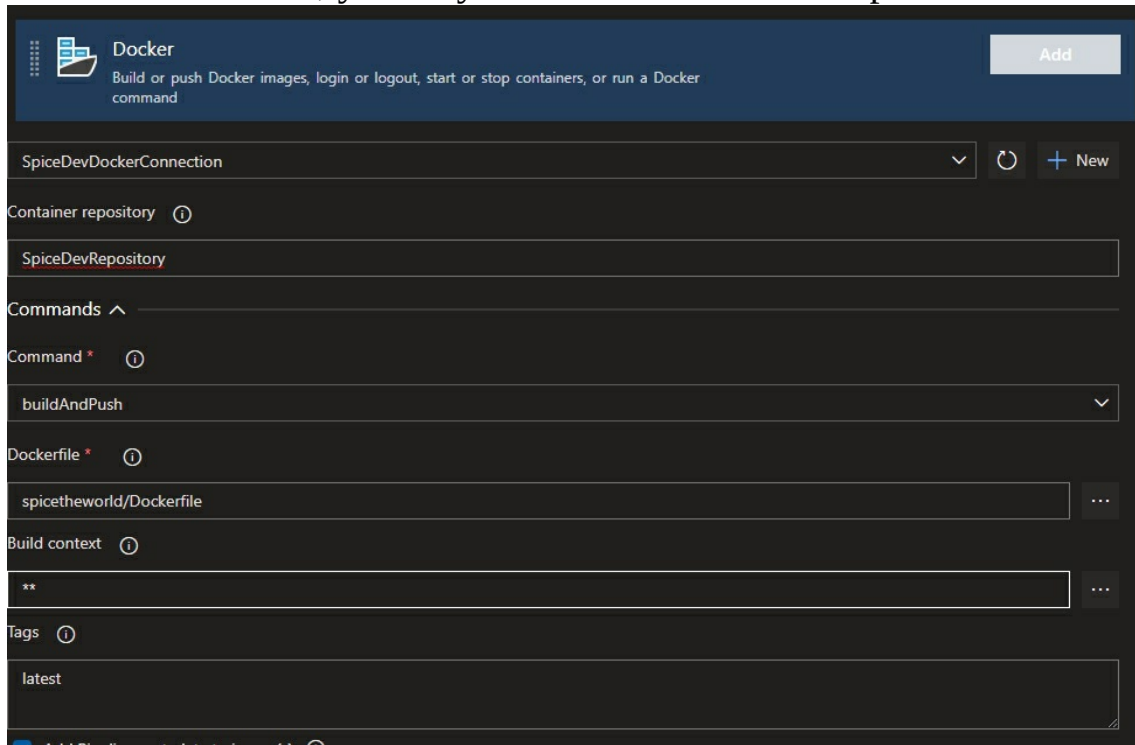


So now edit pipeline and add Docker build and Push. To do this click on the

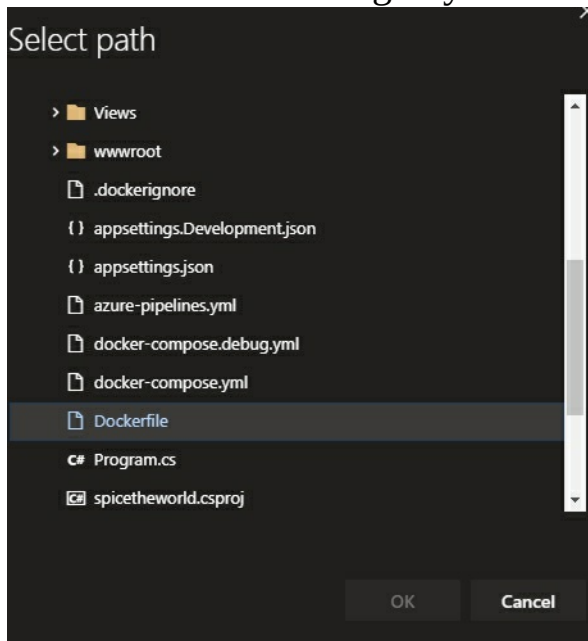
+ link shown below.



Then select Docker, you may have to search for this option.



Select the Container registry and add the Container repository name.



The Docker file you select from the project repository.

To complete Select Save & queue->Save and run. When the run is complete return to your portal and azure.com. From your dashboard select your container registry.



For your information below is the contents of the Docker file.

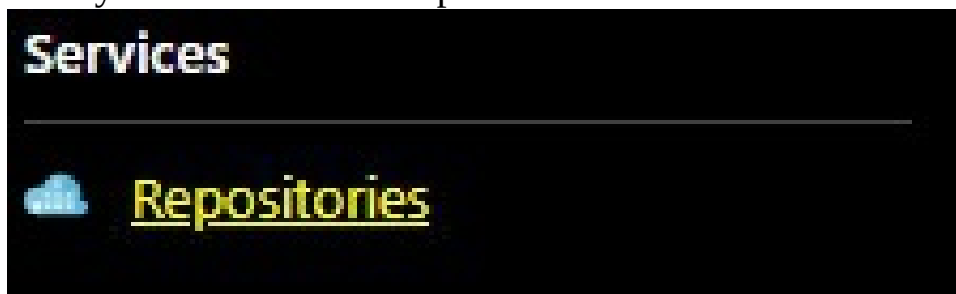
```
FROM mcr.microsoft.com/dotnet/aspnet:3.1 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443
```

```
FROM mcr.microsoft.com/dotnet/sdk:3.1 AS build
WORKDIR /src
COPY ["spicetheworld.csproj", "./"]
RUN dotnet restore "spicetheworld.csproj"
COPY . .
WORKDIR "/src/"
RUN dotnet build "spicetheworld.csproj" -c Release -o /app/build
```

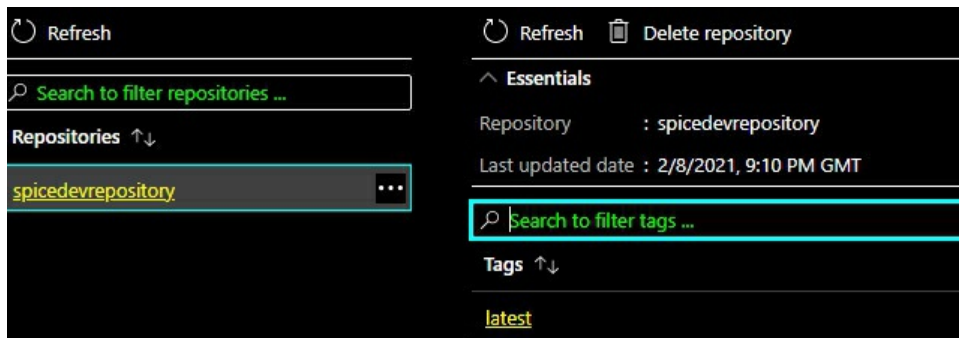
```
FROM build AS publish
RUN dotnet publish "spicetheworld.csproj" -c Release -o /app/publish
```

```
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "spicetheworld.dll"]
```

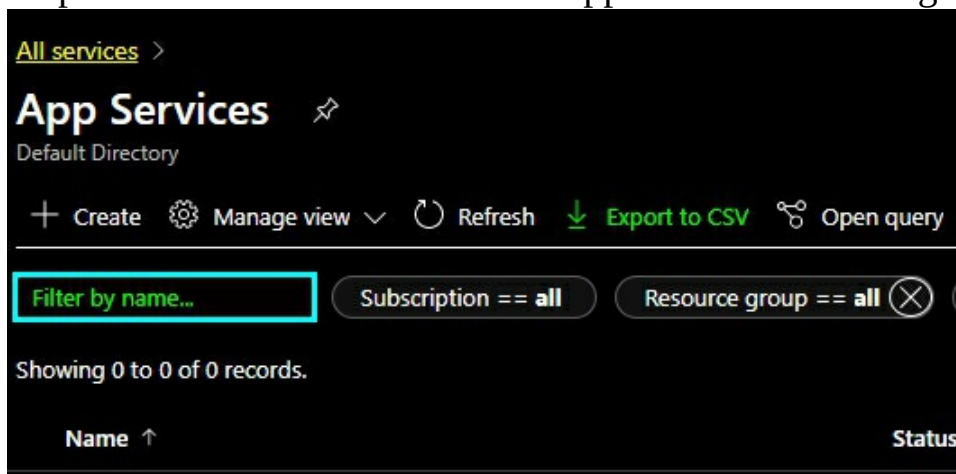
Next you need to select Repositories



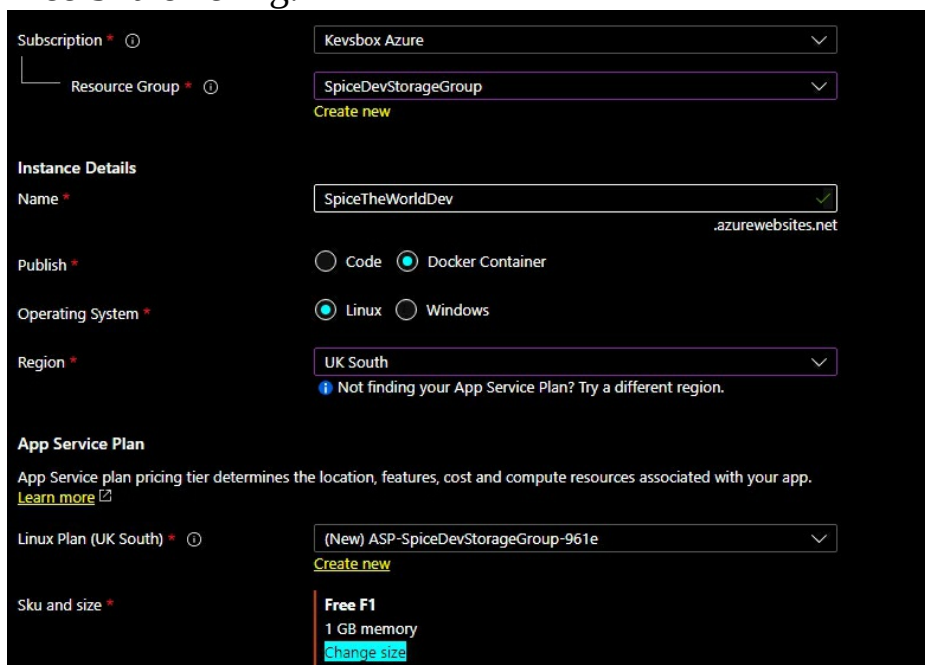
Now you will see your repository.



Step 6: Now locate All Services->App Service for hosting.



And select Create app service. Complete like below and be sure to select the Free Sku offering.



Continue to the Docker tab and also complete as below.

Basics **Docker** Monitoring Tags Review + create

Pull container images from Azure Container Registry, Docker Hub or a private Docker repository. App Service will deploy the containerized app with your preferred dependencies to production in seconds.

Options: Single Container

Image Source: Azure Container Registry

Azure container registry options

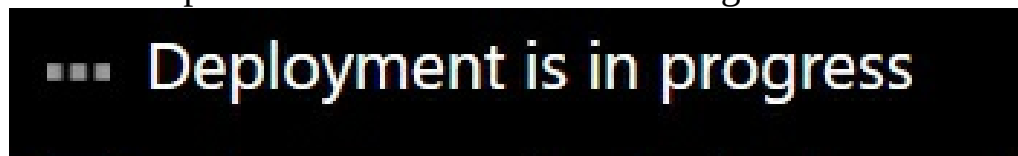
Registry: SpiceDevRegistry

Image: spicedevrepository

Tag: latest

Startup Command: docker run -p 80:80 spicedevrepository:latest

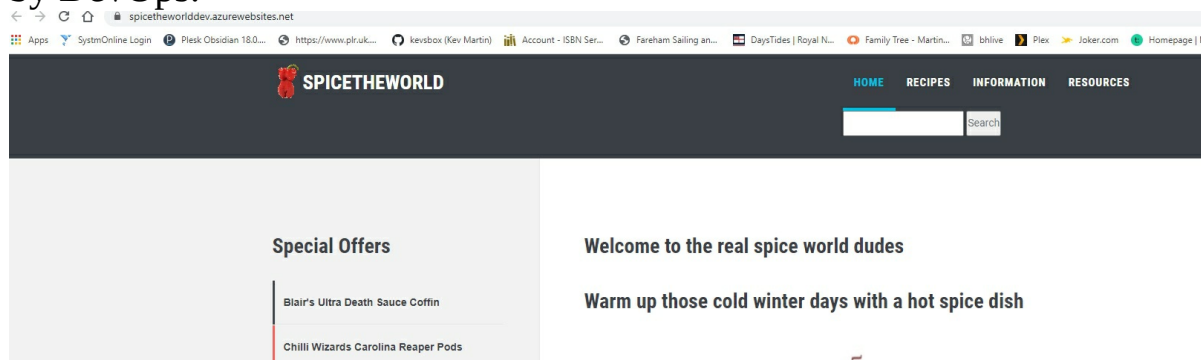
Everything else can stay as default so complete and save.
When complete save and wait for this message.



When deployed you will see an option to go to resource, select this and you should see the service is running.

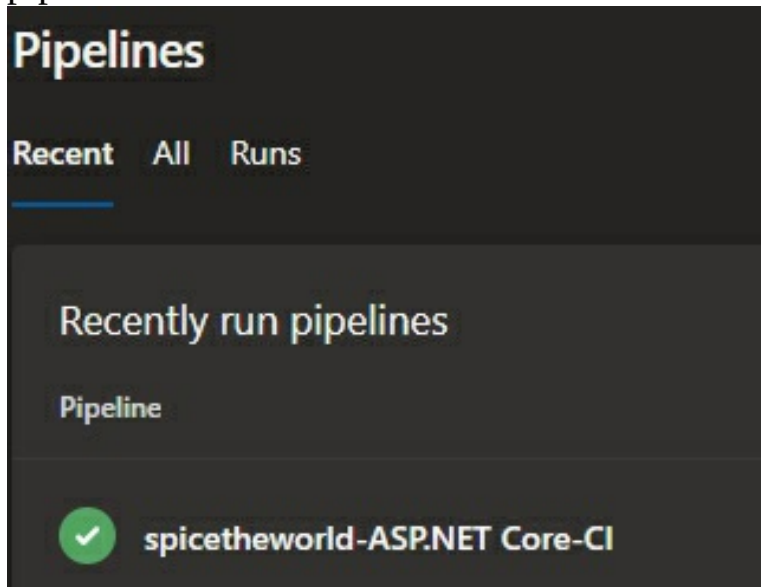
Resource group (change)	: SpiceDevStorageGroup	URL	: https://spicetheworlddev.azurewebsites.net
Status	: Running	Health Check	: Not Configured
Location	: UK South	App Service Plan	: ASP-SpiceDevStorageGroup-961e (F1: Free)
Subscription (change)	: Keybox Azure	FTP/deployment username	: No FTP/deployment user set
Subscription ID	: 1877d6e8-9377-4b8b-86be-94d1740a2ed5	FTP hostname	: ftp://waws-prod-ln1-045.ftp.azurewebsites.windows.net/site/wwwroot
		FTPS hostname	: ftps://waws-prod-ln1-045.ftp.azurewebsites.windows.net/site/wwwroot

You will also notice a URL on the page, this is where your application will reside should the deployment pass. In this case my test URL is **<https://spicetheworlddev.azurewebsites.net/>**. This test site is not always up but if you try it you may well see a page like that below, this was deployed by DevOps.

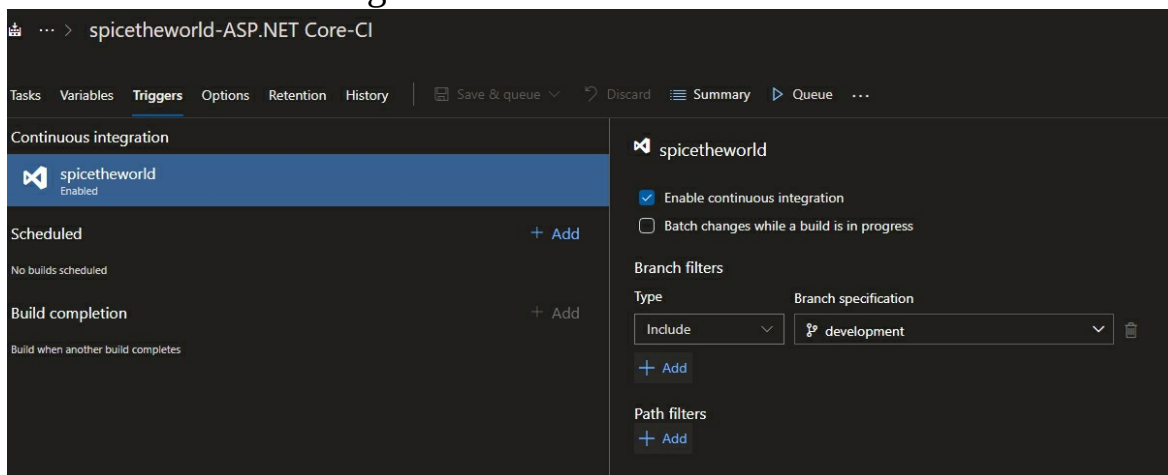


Finally we need to enable continuous integration. This is done in 2 places.
First you need to return to your dev.azure.com account at locate your

pipeline.

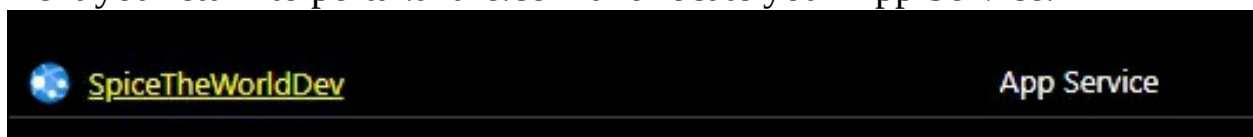


Now click on the pipeline and then select Edit. Go to the Triggers tab and enable continuous integration.



Then Save & Queue etc.

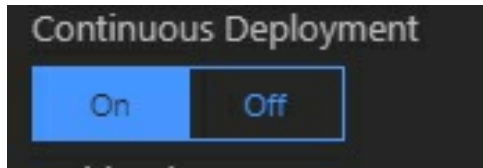
Next you return to portal.azure.com and locate your App Service.



Click on the App Service and select.

Container settings (Classic)

On this page you simply need to turn this option on.



From this point on every time you update your source code and submit the changes the Continuous Deployment will kick in, the code will be built, unit tests (if you have created them) will be run and if all goes well the updated application will be deployed to your test server. Now that is cool.

So what you need to remember here is that now all of the services and pipelines have been created your CI/CD DevOps should kick in every time you commit code to your repository on the branch that you have setup during this process. There is a lot more to DevOps than this but I hope this small walk-through will give you a taste for what is possible.

Conclusion

So here we are then, over two hundred pages into this nice little book, and you are still here, so hopefully, you have enjoyed this publication and found the contents interesting, useful and valid to your needs.

If you become a full-time agile tester in the future or you already there, then I hope you will enjoy your career. Testing can at times become very monotonous and as deadlines loom it can also become very stressful. As I have already pointed out not everyone is suited for software testing. Others can do it but only for a short time. People with the right mindset and temperament for software testing as a career are a scarce but valuable commodity. If you are one of these people then good for you, grow your skills and enhance your career, you will always be in demand.

Being a software tester in an agile team can be a more rewarding experience than working in a traditional organisational setup. Being part of a closely-knit team who are together in regular meetings helps build knowledge and confidence in the product. This, in turn, helps make the testing process a more rewarding and less stressful experience. As your knowledge and skillset grow so will your confidence in becoming a more pro-active team member. This may well annoy some old school programmers who may still think you should be seen and not heard but this is, to put it bluntly, their problem.

So, if you stay in software testing for a long period of time or you eventually move on to other things such as project management, I hope you enjoy the agile experience and I hope you have enjoyed reading this book. So, to conclude in the next chapter we will look at some of the more common agile myths.

The Agile Myths

Agile has been around for a few years now and is now considered a tried, trusted and established development approach. However, for companies that have not yet adopted agile it still represents a new way of life and as a result of the way agile works can still cause unrest among those who have never used it. The prospect of working in iterations instead of with a linear approach can be unsettling to managers and programmers who are deciding whether to make the leap to agile. They fear that focused efforts will be compromised and that control over projects and development teams will be sacrificed. In reality, of course, nothing is further from the truth, so let's get rid of some of the myths that surround agile.

Agile Is a Fad

No, agile is not a new fad. The Agile Manifesto was published in 2001; the Scrum Pattern language was presented in 1995 during the Object Oriented Programming, Systems, and Languages (OOPSLA) conference and there are some who trace agile's roots back further still.

So, the agile approach to project management is far from a fad. Agile has been in use for many decades even though it was only recently formalised with the Agile Manifesto and its associated principles.

Agile exists because it works. Compared with traditional project management approaches, agile is better at producing successful projects.

Agile Means No Documentation

While some people believe that being agile means one doesn't need any documentation, that's hardly the truth; you can have as much documentation as you like in agile. Documentation is just another deliverable; if it brings you value then schedule it and produce it like anything else. Agile teams keep documentation as lightweight as possible, but they do document their solutions as they go. They follow strategies, such as documenting continuously and writing executable specifications.

Agile Is a Silver Bullet

Proponents of Agile will sometimes claim that moving to Agile will "fix all your problems." That isn't the case. Agile isn't needed for every team in every situation. It isn't a cure-all. Agile is a superb solution for projects that are in development or undergoing radical changes.

It's important to stress that the Agile Manifesto is a set of values and principles that define a core aptitude for software development.

These values point to collaboration, rapid feedback loops and quality. If your project has a stable customer base and isn't undergoing a lot of change in the code, you may not need to use agile for that project. But for projects that are creating a new product or major updates then agile is the best way to go.

It's enough for our Development Team to Be Agile

This is so far from the truth that travelling to Pluto would be closer. For agile to work properly within your company, all the teams must buy in fully. So, if your development team is agile, but your testing team is still a group of admin staff pulled in when needed, you will not get your best results, in fact, things could get worse. Your agile delivery process is only going to be as effective as your slowest group. To make agile succeed at its greatest potential, make each piece of the chain as efficient as possible.

Agile Won't Work at My Company

When I hear this statement I always think why? What is your problem? I never think why the company cannot adopt agile I always think why this person says it will not work.

For many companies, the biggest challenge they face when considering changing to agile is the cultural change involved when implementing the changes. For some people, this is a major challenge and they would rather it did not happen. Agile has explicit methods of frequent feedback and loops, which means that programmers, QA's and managers may feel more exposed to scrutiny. But that doesn't mean that agile won't work at your company, it may mean some people should no longer be working at your company though. So, remember folks, agile is a team approach. Roles are cross-functional and shared. Programmers become testers and more frequent delivery creates more exposure and personal accountability.

There's Only One Way to Do Agile

This is so very wrong. The Agile Manifesto consists of four values and 12 principles; it doesn't document implementation details. There are many interpretations of Agile, including Scrum, XP, Kanban and Feature-Driven Development, to name a few. Each style has benefits, as well as weaknesses, and you must evaluate your own specific situation to determine which interpretation is the best match. If you're adhering to the Agile Manifesto's values and principles, you should be considered Agile.

Agile Isn't Disciplined

Sometimes agile can seem chaotic because it's a very collaborative process. Agile is a departure from the rigid assembly-line process. The iterative approach requires rapid response times and flexibility from the team. Agile demands greater discipline than what's typical of traditional teams. Agile requires teams to reduce the feedback cycle on many activities, incrementally deliver a consumable solution, work closely with stakeholders throughout the life cycle, and adopt individual practices which require discipline.

Agile Is Only Effective for Collocated Teams

In an ideal world, the whole team would be located within proximity of one another. However, currently, most development teams are distributed around the globe. While this can be a challenge with the right tools and planning it can still work. Just remember, to succeed, you need to adopt practices and tooling that build team cohesion. Careful planning is the key here.

Agile Means "We Don't Plan"

With agile's reliance on collaboration instead of big documents, it may well seem like no real planning occurs. But the planning is incremental, well understood and evolutionary.

Agile Is Unsuitable for Regulated Environments

Regulated environments are those that are subject to some regulatory mandates, such as defence organisations, medical suppliers, financial companies and government departments to name but a few. These organisations are audited from time to time for compliance with regulations. With agile, these organisations can feel confident when they endure these audits. They benefit from faster delivery of data and higher quality of their output.

Glossary

A

Acceptance Test

Acceptance tests are tests that define the business value each story must deliver. They may verify functional requirements or non-functional requirements such as performance or reliability. Although they are used to help guide development, it is at a higher level than the unit-level tests used for code design in test-driven development. An acceptance test is a broad term that may include both business-facing and technology-facing tests.

Agile Operating Model

The holistic and simple definition of what an organisation, program, project or team mean when they use the term “Agile”. This could range from a single agile framework or an integrated implementation of many frameworks, the latter being much more likely. Agile operating models align with the “Agile Manifesto”.

Agile Persona

Someone (this could be a single person or a group) who will interact with the system being built, also known as a “user”.

Agile Project Management

The style of project management used to support agile software development. Scrum is the most widely used agile project management practice. XP practices also include practices that support agile project management.

Application Programming Interface (API)

APIs enable other software to invoke some piece of functionality. The API may consist of functions, procedures, or classes that support requests made by other programs.

Azure

Azure is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through Microsoft-managed data centres. It provides software as a service (SaaS), platform as a service (PaaS) and infrastructure as a service (IaaS) and supports many

different programming languages, tools, and frameworks.

B

Backlog

An ordered list of requirements/stories that the customer wants.

Baseline Plan

The plan that defines the start point from which an evolving product starts, normally high level.

Behaviour Driven Development (BDD)

Behaviour-driven development (or BDD) is an agile software development technique that encourages collaboration between developers, QA and non-technical or business participants in a software project. BDD focuses on obtaining a clear understanding of desired software behaviour through discussion with stakeholders.

Best Practice

The learned best approach for something at a point in time, best practice evolves over time.

Bugs

A software bug is a problem causing a program to crash or produce invalid output. It is caused by insufficient or erroneous logic and can be an error, mistake, defect or fault.

Build

A build is a process of converting source code into a deployable artefact that can be installed to run the application. The term “build” also refers to the deployable artefact.

Burn-down Chart

A chart showing the evolution of remaining effort against time. Burn-down charts are an optional implementation within Scrum to make progress transparent.

Burn-up Chart

A chart showing the evolution of an increase in a measure against time. Burn-up charts are an optional implementation within Scrum to make progress

transparent.

Business

The customers, stakeholders and users involved with the product.

C

C#

C# is a programming language developed and launched by Microsoft in 2001. C# is a simple, modern, and object-oriented language that provides modern day developers flexibility and features to build software that will not only work today but will be applicable for the long term plans of any organisation.

Chai insertion library

Chai is a BDD/TDD assertion library for node and the browser that can be easily paired with any JavaScript testing framework.

ChromeDriver

WebDriver is an open-source development tool for automated testing of webapps across most popular browsers. It provides capabilities for navigating to web pages, user input, JavaScript execution, and much more. ChromeDriver is a standalone server that implements the W3C WebDriver standard.

Command and Control

This is a style of management where the manager commands the team to do something and then controls them to do it. This style of management is the opposite of Agile self-organising teams and is counter to everything agile.

Commitment Plan

Typically, a detailed forecast for a short period of time they are also known as iteration/ sprint (or time-box) plans.

Component

A component is a larger part of the overall system that may be separately deployable. For example, on the Windows platform, dynamic linked libraries (DLLs) are used as components, Java Archives (JAR files) are components on the Java platform, and a service-oriented architecture (SOA) uses Web Services as components.

Component Test

A component test verifies a component's behaviour. Component tests help with the component design by testing interactions between objects.

Conditions of satisfaction

Conditions of satisfaction also called satisfaction conditions or conditions of business satisfaction, are key assumptions and decisions made by the customer team to define the desired behaviour of the code delivered for a given story. Conditions of satisfaction are criteria by which the outcome of a story can be measured. They evolve during conversations with the customer about high-level acceptance criteria for each story. Discussing conditions of satisfaction helps identify risky assumptions and increases the team's confidence in writing and correctly estimating all the tasks to complete the story.

Context-driven testing

Context-driven testing follows seven principles, the first being that the value of any practise depends on its context. Every new project and every new application may require different ways of approaching a project.

Continuous Deployment

Continuous Deployment (CD) is the process that takes fully validated Features in a staging environment and deploys them into the production environment, where they are readied for a final release.

Continuous Integration

Continuous Integration is the software development practice where members of a development team integrate their work frequently, usually at least daily. Each integration is verified by an automated build (including unit tests) to detect integration errors as quickly as possible.

Cost of Delay

The cost of delaying an investment decision, tend to grow over time.

Cucumber

Cucumber is a software tool that supports behaviour-driven development (BDD). It runs automated acceptance tests written in a behaviour-driven development (BDD) style.

Customer

The person/people who own the product (e.g. usually known as a ‘Product Owners’ or ‘Business Ambassadors’ in certain frameworks).

Customer Team

The customer team identifies and prioritises the features needed for the business. In Scrum, these features become epics or themes, which are further broken into stories and comprise the product backlog. Customer teams include all stakeholders outside of the development team, such as business experts, subject-matter experts, and end-users. QA’s and developers work closely with the customer team to specify examples of the desired behaviour for each story and turn those examples into tests to guide development.

Customer Test

A customer test verifies the behaviour of a slice or piece of functionality that is visible to the customer and related directly back to a story or feature. The terms “business-facing test” and “customer-facing test” refer to the same type of test as a customer test.

D

Daily Scrum/Stand-up

The daily time-boxed event of a maximum of 15 minutes sometimes less. The scrum is for the Development Team to re-plan the next day of development work during a Sprint. Updates are reflected in the Sprint Backlog.

Decomposition

The process of breaking user stories down into a) smaller, more executable user stories or b) tasks. Likewise, epics may be decomposed into user stories, and tasks may be decomposed into more fine-grained tasks. Decomposition is usually performed during backlog grooming and iteration planning and is an important precursor to story sizing (estimation). Decomposition may also occur throughout the development process. In the typical product backlog, user stories will become finer-grained as they near implementation and are larger and less detailed the further down the queue they reside.

Definition of Done

Normally a list of working features that define the complete product that must be delivered; must be standard across the team.

Definition of Ready

Normally a list that defines when artefacts within the delivery process are ready (e.g. story ready to go into iteration/ sprint). This can vary from organisation to organisation.

Development Team

Develop and test the product. The team is self-organised: There is no team leader, so the team makes the decisions. The team is also cross-functional.

DevOps

Viewing the development and operation of a software system as one continuous delivery value process.

E

Emergence

The process of the coming into existence or prominence of new facts or new knowledge of a fact, or knowledge of a fact becoming visible unexpectedly.

Empiricism

Process control type in which only the past is accepted as certain and in which decisions are based on observation, experience and experimentation. Empiricism has three pillars: transparency, inspection and adaptation.

Environment

This is the combination of all factors within an organisation, project, team etc. that drives suitability of a delivery or governance framework. In a dynamic environment, where things change all the time, an agile framework would be suitable.

Epic

An epic is a piece of functionality, or feature, described by the customer and is an item on the product backlog. An epic is broken up into related stories that are then sized and estimated. Some teams use the term “theme” instead of epic.

Estimation

The process of agreeing on a size measurement for the stories, as well as the tasks required to implement those stories, in a product backlog.

Exploratory testing

Exploratory testing is an interactive testing method that combines test design with test execution and focuses on learning about the application.

Extreme Programming (XP)

An Agile software development methodology that emphasises customer involvement, transparency, testing and frequent delivery of working software. The Extreme Programming cannon includes a Customer Bill of Rights and a Developer Bill of Rights and lists its core values as communication, simplicity, feedback, courage and respect. XP is a developer-centric methodology, and unlike Scrum, it prescribes specific coding practices like

Pair Programming, in which two developers work side by side on a single machine, automated unit testing, and frequent integration. Another key practice in XP is refactoring or the continual improvement of design over many iterations.

F

Facilitated Workshops

Where groups of people come together in a forum to achieve a stated objective, the achievement of which is facilitated by a workshop facilitator. Many activities (such as planning) within Agile are delivered within facilitated workshops.

Feature

A feature of the system that the customer wants. These are normally described as a story and ordered within a backlog.

Feature creep

Feature creep occurs when software becomes complicated and difficult to use as a result of too many features.

Functional tests

Functional tests verify the system's expected behaviour given a set of inputs and/or actions.

Forecast

The selection of items from the Product Backlog that a Development Team considers feasible for implementation in a Sprint.

G

Gherkin

Gherkin uses a set of special keywords to give structure and meaning to executable specifications. Common examples being Give, When, Then.

GitHub

GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere on the planet (one day even further).

I

Impediment

In Scrum, any obstacle preventing a developer or team from completing work. One of the three focusing questions each member of a Scrum team answers during the daily Stand-Up Meeting is: What impediments stand in your way?

Increment

A piece of working software that adds to previously created increments, where the sum of all the Increments will form a complete product.

IntelliJ

IntelliJ IDEA is an advanced, integrated development environment written in Java for developing computer software and automated test software. It is developed by JetBrains and is available as an Apache 2 Licensed community edition, and in a proprietary commercial edition. Both can be used for commercial development.

Iteration/ Sprint

An iteration is a short development cycle, generally from one to four weeks, at the end of which production-ready code can potentially be delivered. Several iterations, each one the same length, may be needed to deliver an entire theme or epic. Some teams release the code to production each iteration, but even if the code isn't released, it is ready for release.

Iteration/ Sprint Goal

The goal that the entire team commit to in relation to an iteration/ sprint plan.

Iteration/ Sprint Plan

The forecast of what will be delivered within a short focused 'sprint' by the team.

J

Java

Java is a mature, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture.

Jmeter

JMeter is a software offering that can perform load test, performance-oriented business (functional) test, regression test, etc., on different various protocols or technologies. Stefano Mazzocchi of the Apache Software Foundation was the original developer of JMeter.

K

Kanban

Kanban is a management approach that is sometimes used in agile projects. The general objective is to visualise and optimise the flow of work within a value-added chain.

Knowledge-Based Work

Work where the main capital is knowledge, such as doctors, engineers and information technology workers.

L

Lean

Lean software development is a translation of Lean manufacturing and Lean IT principles and practices to the software development domain. Adapted from the Toyota Production System and is a set of techniques and principles for delivering more values with the same or fewer resources by eliminating waste across organisations and business processes.

M

MS-DOS

MS-DOS, in full **Microsoft** Disk Operating System, the dominant operating system for the personal computer (PC) throughout the 1980s. The acquisition and marketing of **MS-DOS** were pivotal in the **Microsoft** Corporation's transition to software industry giant.

MySQL

MySQL is an open-source relational database management system (RDBMS). Its name is a combination of "My", the name of co-founder Michael Widenius's daughter, and "SQL", the abbreviation for Structured Query Language.

N

Noise

Anything that interrupts the team within an iteration/ sprint, noise causes a significant disturbance within a team and causes a lack of focus on delivery.

NUnit

NUnit is a unit testing framework for performing unit testing based on the .NET platform. It is a widely used tool for unit and automated regression testing and is preferred by many developers/testers today.

P

Pair Programming

Pair programming is an agile software development technique in which two programmers work together at one workstation. One team member will type in the code while the other reviews each line of code as it is typed in. The person typing is called the driver. The person reviewing the code is called the observer (or navigator). The two programmers switch roles frequently.

Planning Poker

Also called Scrum poker, is a consensus-based technique for estimating, mostly used to estimate effort or relative size of tasks in software development.

Postman

Postman is a software tool for testing RESTful APIs made by other developers or even test ones you have made yourself. It offers a user interface with which to make HTML requests, without the hassle of writing a bunch of code just to test an API's functionality.

Product Backlog

The product owner manages a prioritised list of planned product items (called the product backlog). The product backlog evolves from sprint to sprint (called backlog refinement).

Product Backlog Refinement

This is the activity within a Sprint through which the Product Owner and the Development Team add granularity to the Product Backlog.

Product Increment

Each sprint results in a potentially releasable/shippable product (called an increment).

Product Owner

Represents the customer, and generates, maintains, and prioritises the product backlog. This person is not the team lead.

R

Ready

A shared understanding between the Product Owner and the Development Team regarding the preferred level of description of Product Backlog items introduced at Sprint Planning.

Requirements

These are more correctly described as ‘stories’ within most Agile frameworks.

Retrospective

This is a team meeting that should happen at the end of every complete development iteration. The purpose of this meeting is to review lessons learned and to discuss how the team can be more efficient in the future. It is based on the principles of applying the learning from the previous sprint to the upcoming sprint.

RestSharp

RestSharp is a comprehensive, open-source HTTP client library that works with all kinds of DotNet technologies. It can be used to build robust applications by making it easy to interface with public APIs

Ruby

Ruby is an interpreted, high-level, general-purpose programming language. It was designed and developed in the mid-1990’s by Yukihiro Matsumoto in Japan

RubyMine

Rubymine is an integrated development environment (IDE) for Ruby and Rails.

S

Selenium

Selenium is a powerful, free (open-source) automated testing framework used to validate web applications across different browsers and platforms.

Scrum

The framework to support teams in complex product development. Scrum consists of Scrum Teams and their associated roles, events, artefacts, and rules, as defined in the Scrum Guide.

Scrum Board

A physical board to visualise information for and by the Scrum Team; often used to manage Sprint Backlog. Scrum boards are an optional implementation of Scrum to make information visible.

Scrum Guide

The definition of Scrum, written and provided by Ken Schwaber and Jeff Sutherland, the co-creators of Scrum. This definition consists of Scrum's roles, events, artefacts, and the rules that bind them together.

Scrum Master

Ensures that Scrum practices and rules are implemented and followed, and resolves any violations, resource issues, or other impediments that could prevent the team from following the practices and rules. This person is not the team leader, but a coach.

Scrum Team

A self-organising team consisting of a Product Owner, Development Team and Scrum Master.

Scrum Values

A set of fundamental values and qualities underpinning the Scrum framework; commitment, focus, openness, respect and courage.

Self-organisation

The management principle that teams autonomously organise their work. Self-organisation happens within boundaries and against given goals. Teams choose how best to accomplish their work, rather than being directed by

others outside the team.

Serverless Computing

Serverless computing is a software design pattern where applications are hosted by a third-party service. This eliminates the need for server software and hardware management by the developer. Another reason this pattern is growing in popularity is the customer only consumes what they require when they require it, and this reduces costs.

Sprint

Scrum divides a project into iterations (called sprints) of fixed length (usually two to four weeks).

Sprint Backlog

At the start of each sprint, the Scrum team selects a set of highest priority items (called the sprint backlog) from the product backlog. Since the Scrum team, not the product owner, selects the items to be realised within the sprint, the selection is referred to as being on the pull principle rather than the push principle.

Sprint Goal

A short expression of the purpose of a Sprint, often a business problem that is addressed. Functionality might be adjusted during the Sprint in order to achieve the Sprint Goal.

Sprint Planning

Time-boxed event of 1 day, or less, to start a Sprint. It serves as an important meeting for the Scrum Team to inspect the work from the Product Backlog that's most valuable to be done next and design that work into Sprint backlog.

Sprint Retrospective

Time-boxed event of 3 hours, or less, to end a Sprint. It serves as a valuable meeting the Scrum Team to inspect the past Sprint and plan for improvements to be enacted during the next Sprint.

Sprint Review

A time-boxed event of 4 hours for a monthly sprint, less if the sprint was shorter. The review is designed to conclude the development work of a

Sprint. This meeting the Scrum Team and the stakeholders to inspect the Increment of the product resulting from the Sprint and assess the impact of the work performed on overall progress and update the Product Backlog to maximise the value of the next period.

SQL

SQL stands for Structured Query Language, SQL lets you access and manipulate databases of any size or complexity. SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987.

Stakeholder

Any person or group who can help or hinder the team. They are external to the Scrum Team and they have a specific interest in and knowledge of a product that is required for incremental discovery. The Stakeholder(s) are represented by the Product Owner and they should actively engage with the Scrum Team at Sprint Review.

Stored Procedures

A stored procedure is a prepared SQL code segment that you can save so the code can be reused over and over again. You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value's that is passed.

Story

A requirement or feature that may be delivered at some point; a story is a token to remind everyone that something may need to be delivered. Stories reside on the backlog.

T

TDD

Test Driven Development (TDD) is software development approach in which test cases are developed to specify and validate what the code will do.

Time-box

A fixed period within which delivery is made and stories are prioritised within a time-box. With Agile projects, releases and iterations/ sprints are all time-boxes.

Transparency

The development team reports and updates sprint status daily at a meeting called the daily scrum. This makes the content and progress of the current sprint, including test results, visible to the team, management, and all interested parties. For example, the development team can show sprint status on a whiteboard.

U

Unit Tests

A unit test is a way of testing a unit - the smallest piece of code that can be logically isolated in a system.

User

People who will use the product, known as 'Agile personas' within Agile.

User Stories

A user story is an informal, general explanation of a software feature written from the perspective of the end user or customer.

V

Velocity

An optional, but often used, indication of the average amount of Product Backlog turned into an Increment of the product during a Sprint by a Scrum Team, tracked by the Development Team for use within the Scrum Team.

Visual Studio

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft.

W

Watir

Watir stands for Web Application Testing in Ruby. It facilitates the writing of automated tests by mimicking the behaviour of a user interacting with a website.

Working Software

Software that works, it has all the elements associated with the 'Definition of Done' and is ready to deploy into an environment which should be the live production environment.

Appendix 1

Further Reading

Further Reading

Eclipse

- Eclipse: A Java Developer's Guide Paperback by Steve Holzner
- Eclipse IDE: Eclipse IDE based on Eclipse 4.2 and 4.3

Jmeter

- <https://jmeter.apache.org/usermanual/get-started.html>
- <https://www.amazon.co.uk/s?k=jmeter>

Jira

- <https://docs.atlassian.com/software/jira/docs/api/REST/6.1.7/>

Selenium

- Selenium Testing Tools Cookbook by Unmesh Gundecha
- Mastering Selenium WebDriver by Mark Collin
- Selenium 2 Testing Tools: Beginners Guide by David Burns

Team Foundation Server

- Professional Team Foundation Server 2013 by Steven St. Jean and Damian Brady
- Microsoft Team Foundation Server 2015 Cookbook by Tarun Arora

Testing

- Foundations of Software Testing ISTQB Certification by Dorothy Graham and Erik Van Veenendaal

- Testing in Scrum: A Guide for Software Quality Assurance in the Agile World by Tilo Linz

Trello

- <https://developer.atlassian.com/cloud/trello/guides/rest-api/api-introduction/>

Watir

- Cucumber and Cheese by Jeff Morgan

Appendix 2

Download URL's

Download URL's

Chai

<https://www.chaijs.com/api/bdd/>

Eclipse

<http://www.eclipse.org/downloads/>

Java JDK -

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

JMeter

<https://jmeter.apache.org/>

JIRA Download

<https://www.atlassian.com/software/jira/download>

Junit

<http://junit.org/>

Postman

<https://www.postman.com/downloads/>

IntelliJ IDEA

<https://www.jetbrains.com/idea/download>

Nunit

<https://nunit.org/>

RubyMine

<https://www.jetbrains.com/ruby/>

Selenium Chrome Driver

<https://sites.google.com/a/chromium.org/chromedriver/downloads>

Selenium

<http://www.seleniumhq.org/download/>

SpecFlow

<http://specflow.org/>

Visual Studio Community Edition

<https://www.visualstudio.com/en-us/products/visual-studio-community-vs.aspx>

Watir

<http://watir.com/>