Click to verify

A Test Plan Template is an essential tool in software testing that outlines the strategy, scope, and objectives of the testing process. It provides a structured approach to ensure that all aspects of the application are thoroughly evaluated. By using a well-defined test plan template, you can streamline the testing process, improve coverage, and enhance the quality of your software. This guide will help you understand how to create an effective test plan that meets your project's needs and ensures reliable results.What is a Test Plan template?A Test Plan is a descriptive document that describes the test approach, purpose, plan, estimation, results, and resources required for testing. It assists us in regulating the effort required to verify the quality of the application under test. The Test Plan acts as a layout to implement Software Testing activities as a defined process that is closely observed and supervised by the Test Manager.Test Plan Template FormatBelow is a simple and easy-to-understand test plan template format that can be customized according to the project requirements.Test Plan Template FormatWho Prepares the Test Plan Template?The Test Lead or Test Manager prepares the Test Plan template. Testers are also involved in the procedure of writing test plan templates. After the test plan is written, the testers will write test scenarios and test cases depending on the test plan document.Steps to Create an Efficient Test PlanStep 1: Define the Release ScopeClearly outline the features, functionalities, and components that need to be tested in this release.Identify any features that are out of scope to avoid unnecessary testing.Step 2: Schedule TimelinesSet realistic deadlines for each phase of the testing process.Include buffer time for unexpected delays or issues.Step 3: Define Test ObjectivesSpecify the goals of the testing process, such as verifying functionality, performance, and security.Align these objectives with the overall project requirements.Step 4: Determine Test DeliverablesBefore Testing: Create and review the test plan, test cases, and test environment setup.During Testing: Execute test cases, log defects, and monitor progress.After Testing: Provide test reports, defect logs, and sign-off documents.Step 5: Design the Test StrategyChoose the appropriate testing methodologies, such as manual testing, automation, or performance testing.Outline the tools and techniques to be used for each type of testing.Step 6: Plan the Test Environment and Test DataDefine the hardware, software, and network requirements for the test environment.Prepare the necessary test data, including both valid and invalid data sets, to ensure comprehensive testing.One-Page Test Plan TemplateFor a quick overview, a one-page test plan template can include:Project Name: [Software Name]Test Manager: [Name]Test Objectives: [List Key Objectives]Scope: [In Scope/Out of Scope]Test Strategy: [Testing Types, Tools, etc.]Schedule: [Start Date, End Date, Milestones]Resources: [Testers, Developers, Tools]Risks: [Identify Potential Risks]Deliverables: [Test Cases, Reports, etc.]Example of Test Plan TemplateProject OverviewThe project involves developing a new e-commerce platform for a retail company. The platform includes features such as user registration, product browsing, shopping cart management, payment processing, and order management. The goal is to ensure that the platform is fully functional, secure, and user-friendly before its public launch.Test Plan DetailsProject Name: E-Commerce Platform DevelopmentTest Manager: John DoeTesting TeamMary Smith (Lead Tester)Raj Patel (Automation Specialist)Jane Liu (Performance Tester)Ahmed Hassan (Security Tester)1. Define the Release ScopeIn-Scope Features:User Registration and Login: Verify that users can register, log in, and manage their accounts securely.Product Browsing and Search: Ensure that users can browse products, use filters, and search functionalities.Shopping Cart Management: Test adding, updating, and removing items from the shopping cart.Payment Processing: Validate the payment gateway integration, including credit card processing, PayPal, and other methods.Order Management: Confirm that users can place orders, track them, and receive notifications.Out-of-Scope Features:Social Media Integration: Will be included in a future release.Advanced Analytics: Only basic tracking will be tested in this release.2. Schedule TimelinesTesting Phases:Test Planning: Aug 1, 2024 - Aug 7, 2024Test Case Design: Aug 8, 2024 - Aug 14, 2024Test Environment Setup: Aug 15, 2024 - Aug 17, 2024Test Execution: Aug 18, 2024 - Sept 5, 2024Defect Resolution: Sept 6, 2024 - Sept 12, 2024Test Closure: Sept 13, 2024 - Sept 14, 2024Milestones:Test Plan Sign-Off: Aug 7, 2024Test Case Review: Aug 14, 2024First Test Execution Cycle Complete: Aug 25, 2024Final Test Report Submission: Sept 14, 20243. Define Test ObjectivesPrimary Objectives:Functional Testing: Verify that all in-scope features work as expected under various conditions.Performance Testing: Ensure the platform can handle high traffic volumes, especially during sales and promotions.Security Testing: Identify and fix vulnerabilities that could expose user data or allow unauthorized access.Usability Testing: Assess the platform's ease of use and user experience, ensuring that customers can navigate and complete purchases effortlessly.Secondary Objectives:Compatibility Testing: Test the platform on different browsers (Chrome, Firefox, Safari) and devices (desktop, mobile, tablet).Regression Testing: Re-test previously tested functionalities to ensure that new changes have not introduced defects.4. Determine Test DeliverablesBefore Testing:Test Plan Document: Detailed test plan, including scope, objectives, strategy, and resources.Test Cases: Specific test cases for each feature, outlining the steps to be executed and the expected results.Test Data: Prepared data sets for testing various scenarios, such as valid/invalid user inputs, different payment methods, etc.Test Environment Setup: Configuration of servers, databases, and other necessary components.During Testing:Execution Logs: Records of all test cases executed, including pass/fail status.Defect Reports: Documentation of any issues found, including steps to reproduce, screenshots, and severity.Daily/Weekly Status Reports: Updates on testing progress, including completed tests, open defects, and risks.After Testing:Final Test Report: Summary of the testing activities, including overall test coverage, defect trends, and final recommendations.Defect Log: Comprehensive list of all identified defects, including their status (open, fixed, closed) and resolution.Test Closure Report: Document indicating that all planned tests have been completed, and the software is ready for release.5. Design the Test StrategyFunctional Testing:Manual Testing: Testers will manually execute test cases for critical functionalities like user registration, payment processing, and order management.Automation Testing: Automation scripts will be created using Selenium WebDriver to handle repetitive tasks like login/logout, adding items to the cart, and performing search operations.Performance Testing:Load Testing: Simulate a large number of users accessing the platform simultaneously to assess performance under normal and peak conditions.Stress Testing: Test the platform's behavior under extreme conditions to identify breaking points and ensure graceful degradation.Security Testing:Vulnerability Scanning: Use automated tools to scan for common vulnerabilities like SQL injection, cross-site scripting (XSS), and insecure configurations.Penetration Testing: Perform manual tests to identify potential security threats that automated tools might miss.Usability Testing:User Interviews and Surveys: Collect feedback from a sample of users to understand their experience with the platform.A/B Testing: Compare different versions of certain features (e.g., checkout flow) to determine which performs better in terms of user satisfaction and conversion rates.6. Plan the Test Environment and Test DataTest Environment Setup:Servers: Ensure that the test servers mimic the production environment, including load balancers, web servers, and databases.Software: Install necessary software, such as the web server (Apache), database management system (MySQL), and the application itself.Network Configuration: Set up the network to replicate typical user conditions, including firewalls, VPNs, and internet speeds.Test Data Preparation:Valid Data: Create realistic data sets, such as user profiles, product listings, and payment information, that reflect typical use cases.Invalid Data: Generate data to test edge cases and error handling, such as incorrect user inputs, expired credit cards, and unavailable products.Purpose & Importance of Test Plan TemplateThe key purpose of a Test Plan is to build documentation that defines how the tester will prove that the software works as it should. A test plan template is essential because it acts like a roadmap for testing, guiding what tests need to be done, how to do them, and when. Its primary purposes are to ensure organized and thorough testing by outlining the testing scope, objectives, and resources. It also serves as a communication tool among testers, developers, and stakeholders, helping everyone stay aligned. Additionally, it helps monitor progress, adapt to changes, and manage the entire testing process effectivelyThe document should detail what is required to be tested, how it will be tested, and who is in charge of testing.It acts as a layout to implement software testing activities as a defined process.By putting up a test plan, all team persons can work together and communicate their parts to each other.How To Prepare an Effective Test Plan?1. Define ObjectivesObjective Setting: Clearly outline the goals of your testing. Determine what you aim to achieve, such as:Finding Bugs: Identifying and documenting defects or issues within the application.Assessing Performance: Evaluating how well the application performs under various conditions.Verifying Functionality: Ensuring that the application works as intended and meets specified requirements.Why It Matters: Clear objectives help focus the testing efforts and align them with the project's goals, ensuring that all critical aspects are covered.2. Break Down the ApplicationComponent Analysis: Divide the application into manageable components or modules. This involves:Identifying Key Areas: Determine which parts of the application are critical and need thorough testing.Prioritizing Components: Focus on high-risk areas that could significantly impact the application's performance or functionality.Why It Matters: Breaking down the application helps in organizing the testing process and ensures that all crucial parts are thoroughly tested.3. Design Test ScenariosScenario Creation: Develop test scenarios that simulate real-world use cases and edge cases, including:User Interactions: Create scenarios based on typical user behaviors and interactions with the application.Edge Cases: Include less common but possible scenarios that could uncover hidden issues.Prioritization: Rank scenarios based on their importance and impact on the application to optimize testing time and resources.Why It Matters: Comprehensive and well-prioritized test scenarios help in effectively evaluating the application's functionality and robustness.4. Select Testing TechniquesMethodology Choice: Choose the appropriate testing techniques and methodologies based on the project's requirements. This may include:Manual Testing: Performing tests manually to evaluate functionality and user experience.Automated Testing: Using tools and scripts to perform repetitive and complex test cases efficiently.Performance Testing: Assessing how the application handles load and stress.Why It Matters: Selecting the right techniques ensures that the testing is efficient and aligned with the project's specific needs and constraints.5. Create a Master ScheduleScheduling: Develop a detailed schedule that includes:Testing Phases: Outline all phases from preparation (e.g., setting up test environments) to execution (running the tests) and reporting (documenting results).Timelines: Set realistic timelines for each activity to manage the testing process effectively.Why It Matters: A well-structured schedule helps in managing time and resources, ensuring that testing activities are completed systematically and on time.By following these steps, you can prepare a comprehensive and effective test plan that helps ensure thorough testing and contributes to the successful delivery of a high-quality application.Test Planning in a Test Management ToolUsing a test management tool can streamline the test planning process by allowing teams to:Milestones: Track important deadlines and progress.Collaboration: Share the test plan with all stakeholders and receive feedback in real-time.Reporting: Automatically generate reports on testing progress, defects, and coverage.ConclusionA well-structured test plan is essential for ensuring that all aspects of a software application are thoroughly tested and that the testing process is efficient and effective. By following the steps outlined above, teams can create a comprehensive test plan that covers all necessary areas and aligns with the project's goals. This leads to higher quality software and a smoother release process. , the free encyclopedia that anyone can edit. 109,638 active editors 7,014,808 articles in English HMS Neptune was a dreadnought battleship built for the Royal Navy in the first decade of the 20th century, the sole ship of her class. Laid down at HM Dockyard, Portsmouth, in January 1909, she was the first British battleship to be built with superfiring guns. Shortly after her completion in 1911, she carried out trials of an experimental fire-control director and then became the flagship of the Home Fleet. Neptune became a private ship in early 1914 and was assigned to the 1st Battle Squadron. The ship became part of the Grand Fleet when it was formed shortly after the beginning of the First World War in August 1914. Aside from participating in the Battle of Jutland in May 1916, and the inconclusive action of 19 August several months later, her service during the war generally consisted of routine patrols and training in the North Sea. Neptune was deemed obsolete after the war and was reduced to reserve before being sold for scrap in 1922 and subsequently broken up. (Full article...) Recently featured: Nominative determinism Donkey Kong Land History of education in Wales (1701–1870) Archive By email More featured articles About Wreckage of Thai Airways International Flight 114 ... that Thai prime minister Thaksin Shinawatra was minutes away from boarding an aircraft that exploded (wreckage pictured)? ... that L. Whitney Watkins was given the Bull Moose Party's nomination in a 1912 election despite his own opposition? ... that a 1915 film about Florence Nightingale was criticised for not mentioning her pet parrot? ... that the statue Receiver was repainted in 2013 to match the likeness of NFL player Donald Driver after his retirement? ... that actress Jennifer Metcalfe used the experience of her father's cancer in Episode 6465 of the British soap opera Hollyoaks? ... that economist Roger A. Freeman questioned the value of college and favored limiting access to it to a select few? ... that the children's novel Queenie portrays the early years of the NHS in England? ... that painter Nicolino Calyo left Naples after participating in a failed uprising against King Ferdinand IV, then fled Spain following the outbreak of the First Carlist War? ... that Class War was held responsible for the poll tax riots? Archive Start a new article Nominate an article Trifid and Lagoon nebulae The Vera C. Rubin Observatory in Chile releases the first light images (example shown) from its new 8.4-metre (28 ft) telescope. In basketball, the Oklahoma City Thunder defeat the Indiana Pacers to win the NBA Finals. An attack on a Greek Orthodox church in Damascus, Syria, kills at least 25 people. The United States conducts military strikes on three nuclear facilities in Iran. In rugby union, the Crusaders defeat the Chiefs to win the Super Rugby Pacific final. Ongoing: Gaza war Iran–Israel war Russian invasion of Ukraine timeline Sudanese civil war timeline Recent deaths: John R. Casani Richard Gerald Jordan Franco Testa Raymond Laflamme Gertrud Leutenegger Maria Voce Nominate an article June 28: Vidovdan in Serbia Ned Kelly 1880 – Police captured Australian bank robber and cultural icon Ned Kelly (pictured) after a gun battle in Glenrowan, Victoria. 1895 – The U.S. Court of Private Land Claims ruled that James Reavis's claim to 18,600 sq mi (48,000 km2) of land in present-day Arizona and New Mexico was "wholly fictitious and fraudulent". 1904 – In the worst maritime disaster involving a Danish merchant ship, SS Norge ran aground on Hasselwood Rock and sank in the North Atlantic, resulting in more than 635 deaths. 1950 – Korean War: South Korean forces began the Bodo League massacre, summarily executing tens of thousands of suspected North Korean sympathizers. 1969 – In response to a police raid at the Stonewall Inn in New York City, groups of gay and transgender people began demonstrations, a watershed event for the worldwide gay rights movement. Charles Cruft (b. 1852)Olga Sapphire (b. 1907)Meralda Warren (b. 1959)Aparna Rao (d. 2005) More anniversaries: June 27 June 28 June 29 Archive By email List of days of the year About Myosotis scorpioides, the water forget-me-not, is a herbaceous perennial flowering plant in the borage family, Boraginaceae. It is native to Europe and Asia, but is widely distributed elsewhere, including much of North America, as an introduced species and sometimes a noxious weed. It is an erect to ascending plant of up to 70 cm, bearing small (8–12 mm) flowers that become blue when fully open and have yellow centers. It is usually found in damp or wet habitats, such as bogs, ponds, streams, ditches, fen, and rivers. This focus-stacked photograph shows a water forget-me-not growing in Niitvälja bog, Estonia. Photograph credit: Ivar Leidus Recently featured: Whitehead's trogon Atacamite Turban Head eagle Archive More featured pictures Community portal – The central hub for editors, with resources, links, tasks, and announcements. Village pump – Forum for discussions about Wikipedia itself, including policies and technical issues. Site news – Sources of news about Wikipedia and the broader Wikimedia movement. Teahouse – Ask basic questions about using or editing Wikipedia. Help desk – Ask questions about using or editing Wikipedia. Reference desk – Ask research questions about encyclopedic topics. Content portals – A unique way to navigate the encyclopedia. Wikipedia is written by volunteer editors and hosted by the Wikimedia Foundation, a non-profit organization that also hosts a range of other volunteer projects: CommonsFree media repository MediaWikiWiki software development Meta-WikiWikimedia project coordination WikibooksFree textbooks and manuals WikidataFree knowledge base WikinewsFree-content news WikiquoteCollection of quotations WikisourceFree-content library WikispeciesDirectory of species WikiversityFree learning tools WikivoyageFree travel guide WiktionaryDictionary and thesaurus This Wikipedia is written in English. Many other Wikipedias are available; some of the largest are listed below. 1,000,000+ articles العربية Deutsch Español فارسی Français Italiano Nederlands 日本語 Polski Português Русский Svenska Українська Tiếng Việt 中文 250,000+ articles Bahasa Indonesia Bahasa Melayu Bân-lâm-gú Български Català Čeština Dansk Eesti Ελληνικά Esperanto Euskara עברית Հայերեն Magyar Norsk bokmål Română Simple English Slovenčina Srpski Srpskohrvatski Suomi Türkçe Oʻzbekcha 50,000+ articles Asturianu Azərbaycanca Bosanski کوردی Frysk Gaeilge Galego Hrvatski ქართული Kurdî Latviešu Lietuvių Македонски Norsk nynorsk Shqip Slovenščina اردو Retrieved from " 2Battleship formation of the Royal Navy For the German counterpart during World War I, see I Battle Squadron. 1st Battle SquadronThe 1st Battle Squadron at sea, April 1915Active1912–1945Country United KingdomBranch Royal NavyTypeSquadronSize8 x BattleshipsPart ofGrand FleetMilitary unit The 1st Battle Squadron was a naval squadron of the British Royal Navy consisting of battleships. The 1st Battle Squadron was initially part of the Royal Navy's Grand Fleet. After World War I the Grand Fleet was reverted to its original name, the Atlantic Fleet. The squadron changed composition often as ships were damaged, retired or transferred. As an element in the Grand Fleet, the Squadron participated in the Battle of Jutland.[1] On 5 August 1914, the squadron was constituted as follows:[2] HMS Marlborough HMS Collingwood HMS Colossus HMS Hercules HMS Neptune HMS St. Vincent HMS Superb HMS Vanguard Revenge and Hercules en route to Jutland with the sixth division. During the Battle of Jutland, the composition of the 1st Battle Squadron was as follows:[1] Sixth Division HMS Marlborough Flagship of Vice-Admiral Sir Cecil Burney; Captain G. P. Ross; HMS Revenge Captain E. B. Kiddle; HMS Hercules Captain L. Clinton-Baker; HMS Agincourt Captain H. M. Doughty; Fifth Division HMS Colossus Flagship of Rear Admiral E. F. A. Gaunt; Captain A. D. P. R. Pound; HMS Collingwood Captain J. C. Ley; HMS St. Vincent Captain W. W. Fisher; HMS Neptune Captain V. H. G. Bernard; HMS Revenge Following the Battle of Jutland, the 1st Battle Squadron was reorganized, with Colossus, Hercules, St. Vincent, Collingwood and Neptune all transferred to the 4th Battle Squadron. In January 1917, the squadron was constituted as follows:[3] HMS Marlborough HMS Agincourt HMS Benbow – joined July, 1916 HMS Canada HMS Emperor of India – joined July, 1916 HMS Revenge HMS Royal Oak – joined May, 1916 HMS Royal Sovereign – joined June, 1916 By 1918, Agincourt had been transferred to the 2nd Battle Squadron, and Resolution, Ramillies and Iron Duke had joined the squadron on completion.[4] For many years the squadron served in the Mediterranean as the main British battle force there. On 3 September 1939 the 1st Battle Squadron, serving in the Mediterranean Fleet, consisted of Barham, Warspite and Malaya, with headquarters at Alexandria, Egypt, under the command of Vice-Admiral Geoffrey Layton.[5] In December 1943 the Squadron was under the command of Vice Admiral Arthur Power. In January 1944 the Eastern Fleet was reinforced by HMS Queen Elizabeth, HMS Renown, HMS Valiant, HMS Illustrious, HMS Unicorn and seven destroyers. The Admiralty sent this force out to India under the title of the First Battle Squadron.[6] From November 1944, the squadron served in the British Pacific Fleet under the command of Vice-Admiral Henry Rawlings, who also served as Second-in-Command of the Fleet. It consisted of HMS King George V, HMS Howe, HMS Duke of York and HMS Anson at various times. Commanders were as follows:[7] Vice-Admiral Sir Stanley Colville (1912–14) Vice-Admiral Sir Lewis Bayly (June–December 1914) Admiral Sir Cecil Burney (1914–16) Vice-Admiral Sir Charles Madden (1916–19) Vice-Admiral Sir Sydney Fremantle (1919–21) Vice-Admiral Sir William Nicholson (1921–22) Vice-Admiral Sir Edwyn Alexander-Sinclair (1922–24) Rear-Admiral William Fisher (1924–25) Rear-Admiral Cecil Staveley (1925–26) Vice-Admiral Sir Michael Hodges (1926–27) Vice-Admiral Sir John Kelly (1927–29) Vice-Admiral Howard Kelly (1929–30) Vice-Admiral Sir William Fisher (1930–32) Vice-Admiral Sir Roger Backhouse (1932–34) Vice-Admiral Sir Charles Forbes (1934–36) Vice-Admiral Hugh Binney (1936–38) Rear-Admiral Ralph Leatham (1938–39) Vice-Admiral Geoffrey Layton (January–November 1939) Rear-Admiral Henry Pridham-Wippell (July–October 1940) Vice-Admiral John Tovey (October–December 1940) Rear-Admiral Bernard Rawlings (1940–41) Vice-Admiral Sir Henry Pridham-Wippell (1941–42) Vice-Admiral Sir Arthur Power (1943–44) Vice-Admiral Sir Bernard Rawlings (1944–45) Post holders included:[8] Rear-Admiral Charles E. Madden, 5 January 1912 – 10 November 1912 Rear-Admiral The Hon. Somerset A. Gough-Calthorpe, 10 December 1912 – 10 December 1913 Rear-Admiral Hugh Evan-Thomas, 10 December 1913 – 25 August 1915 Rear-Admiral Ernest Gaunt, 25 August 1915 – 12 June 1916 Rear-Admiral Alexander L. Duff, 12 June 1916 – 30 November 1916 Rear-Admiral Sir William C. M. Nicholson, 1 December 1916 – 20 March 1919 Rear-Admiral The Hon. Victor A. Stanley, 1 April 1919 – 1 April 1920 Rear-Admiral Henry M. Doughty, 24 March 1920 – 14 April 1921 Rear-Admiral Sir Rudolf W. Bentinck, 3 May 1921 – 3 May 1922 Rear-Admiral Arthur A. M. Duff, 3 May 1922 Rear-Admiral William A. H. Kelly, 3 May 1923 Rear-Admiral William H. D. Boyle, 3 May 1924 – 3 May 1924 Rear-Admiral William W. Fisher, 14 October 1924 – 7 September 1925 Rear-Admiral Cecil M. Staveley, 15 October 1925 – 1 October 1926 Rear-Admiral David T. Norris, 1 October 1926 Rear-Admiral Bernard St. G. Collard, 1 October 1927 Rear-Admiral William M. Kerr, 20 March 1928 – 5 April 1929 Rear-Admiral The Hon. Reginald A. R. P.-E.-E.-Drax, 12 April 1929 – 26 April 1930 Rear-Admiral Henry D. Pridham-Wippell, 8 May 1940 – 24 October, 1941 ^ a b Macintyre, Donald. Jutland Evans Brothers Ltd. 1957; ISBN 0-330-20142-5 ^ Dittmar, F.J & Colledge J.J., British Warships 1914–1919 Ian Allan, London. 1972; ISBN 0-7110-0380-7 ^ Dittmar, F.J & Colledge J.J., British Warships 1914–1919 Ian Allan, London. 1972; ISBN 0-7110-0380-7 pp20 ^ Dittmar, F.J & Colledge J.J., British Warships 1914–1919 Ian Allan, London. 1972; ISBN 0-7110-0380-7 pp24 ^ Orbat.com/Niehorster, Mediterranean Fleet, 3 September 1939, accessed May 2008 ^ Jackson, Ashley (2006). The British Empire and the Second World War. Continuum International Publishing Group. p. 301. ISBN 1-85285-417-0. ^ "Royal Navy Senior Appointments" (PDF). Archived from the original (PDF) on 11 July 2011. Retrieved 4 October 2014. ^ Harley, Simon; Lovell, Tony. "First Battle Squadron (Royal Navy) - The Dreadnought Project". www.dreadnoughtproject.org. Harley and Lovell, 27 December 2016. Retrieved 15 February 2018. First Battle Squadron at DreadnoughtProject.org Royal Navy History Composition of the Grand Fleet Retrieved from " 3 The following pages link to 1st Battle Squadron External tools (link count transclusion count sorted list) · See help page for transcluding these entries Showing 50 items. View (previous 50 | next 50) (20 | 50 | 100 | 250 | 500)List of dreadnought battleships of the Royal Navy (links | edit) Revenge-class battleship (links | edit) HMS King Edward VII (links | edit) HMS Glorious (links | edit) HMS Revenge (06) (links | edit) HMS Resolution (09) (links | edit) HMS Royal Oak (08) (links | edit) Renown-class battlecruiser (links | edit) HMS Renown (1916) (links | edit) HMS Repulse (1916) (links | edit) HMS Emperor of India (links | edit) British Pacific Fleet (links | edit) HMS New Zealand (1911) (links | edit) Courageous-class battlecruiser (links | edit) 5th Battle Squadron (links | edit) HMS Agincourt (1913) (links | edit) HMS Africa (1905) (links | edit) Mediterranean Fleet (links | edit) Hugh Evan-Thomas (links | edit) Henry Harwood (links | edit) HMS Britannia (1904) (links | edit) List of fleets and major commands of the Royal Navy (links | edit) HMS Hibernia (1905) (links | edit) Reginald Drax (links | edit) HMS Southampton (1912) (links | edit) Atlantic Fleet (United Kingdom) (links | edit) HMS Commonwealth (links | edit) Boadicea-class cruiser (links | edit) Sir Charles Madden, 1st Baronet (links | edit) HMS Hindustan (1903) (links | edit) Tondern raid (links | edit) 6th Battle Squadron (links | edit) British 1st Battle Squadron (redirect page) (links | edit) Henry Pridham-Wippell (links | edit) HMS Gloucester (1909) (links | edit) HMS Dominion (links | edit) HMS Zealandia (links | edit) HMS Bellona (1909) (links | edit) Main Page (links | edit) 2nd Battle Squadron (links | edit) 3rd Battle Squadron (links | edit) 4th Battle Squadron (links | edit) Action off Cape Passero (links | edit) 701 Naval Air Squadron (links | edit) Henry Bruce (Royal Navy officer) (links | edit) 9th Battle Squadron (links | edit) David Norris (Royal Navy officer) (links | edit) Ralph Leatham (links | edit) Rudolph Bentinck (links | edit) Bernard Rawlings (Royal Navy officer) (links | edit) View (previous 50 | next 50) (20 | 50 | 100 | 250 | 500) Retrieved from " WhatLinksHere/1st_Battle_Squadron" Are you a Journalist, Influencer, or a Product Owner? Get free products to review on your Social Media/Blogs. In the last blog post we looked how Azure DevOps Test Plans is used to structure testing and how we can run the regression testing. In this post we are going to look how Test Plans can be used to handle different test environments and how we can improve our test cases with parameters. Configurations is a Test Plan feature that can be used to define different environments that we should run our tests in. For example we can define configurations for operating systems, browsers and devices. If we are developing IoT devices we can create configuration for every device type (or generation) we are currently supporting and the run test cases against all those configurations. This is simple way to duplicate test cases for different devices. Configurations are split into two different things: Configuration variables and Test Configurations. The Configuration variable is for example an operating system or a browser. It is variable that will have multiple different values and the test configuration is a set of multiple configuration variables. In my example I'm having browser configuration variable with following values: Firefox, Chrome and Edge. I also have operating system with Windows 10, Windows 11 and Windows 8. Now I will combine these two test variables to create a test configuration: Operating system Windows 10 and browser Firefox, operating system Windows 10 and browser Edge etc. I don't have to create configurations for all the combinations of variables, just the ones that I'm going to need. Remember to push Save button at top when doing changes After creating the configurations, we can assign them into test suites or test cases by right-clicking on top of test suite/case. Assigning configurations into test suite When the configuration is assigned into test suite, the Azure DevOps will duplicate all the test cases for selected configurations. In this example I have three different configurations set for test suite, so all the tests are tripled into test run (one time for each configuration). Test cases are multiplied by selected number of configurations into test run We can also filter our test progress report with configurations to monitor how tests are succeeding in different configuration sets Progress report also supports configurations Parameters is a way to share information between test cases. For example if we have username and password that we are using in multiple different use cases (registration, logins etc.). We don't want to write down those into every single test case, because if we need to change them, it is hard to go through all the test cases. We can use the parameters to store the username and password combinations and then use them in test cases. Username, password and groupname as parameter Here I have created parameter set called Usernames that contains usernames, passwords and groupnames. I have two different users in this parameter set. Now when I'm writing the test case I can use the parameters with @Username, @Password and @Groupname notation. The Azure DevOps will recognize the syntax and link the parameter set into test case. We can see that there is "Pameter values" at the bottom that means we have successfully linked test case into parameter set. Now when we go to execute this test the Azure DevOps will add iteration for each parameters. We had two rows in our example, so we will get two different iterations per linked test case. Parameters will create an iteration into test run Note that you can link only one parameter set per test case. If you need two, you have to combine the parameter sets into one and just use the parameters that you need from the set. Configurations can be used to create different kind of test environments. It can be used to track testing against different devices or different browsers. We can easily filter test results based on their run configurations so we can get good overview of test progress per configuration. Parameters are simple way to share information between multiple test cases. Parameter row will generate iteration per test case into test run where the configuration will duplicate the test case itself per assigned configurations. We might have seen earlier about how what is meant TestPlans, TestSuites & TestCases. Let us see how how to create TestPlans in Azure DevOps. A Project can have one or more TestPlans but generally if it's more than 10 or 20 it's difficult to maintain so any project team should have minimum TestPlans and create more folders for different Releases under TestPlan. You can have different TestPlans for different teams within a project and it all depends upon which areapath you are creating a TestPlan. let us see what are the steps to create a TestPlan Step 1: Click on "Test Plans" on the left menu & then Click on "New Test Plan" Step 2: Enter the name of the TestPlan you like to create , then choose the AreaPath, Iteration for the TestPlan & then click on "Create" button Step 3: After the "Create" button is clicked, then the "TestPlan" will be created as shown. Each and every TestPlan will have a default "Test Suite" with the name of the "TestPlan" itself. Like this you can create multiple TestPlans as shown If you're looking to streamline your software development process and ensure better quality for your applications, integrating test plans into Azure DevOps can be a game-changer. It allows you to manage, track, and execute tests efficiently, helping you identify and fix bugs early in the development cycle. In this guide, we'll show you how to enable test plans in Azure DevOps and get the most out of this powerful feature. What are Test Plans in Azure DevOps? Test plans in Azure DevOps are a set of tools and functionalities that allow you to create, manage, and track tests for your software projects. By defining test suites, test cases, and test configurations, you can organize your testing efforts and ensure comprehensive coverage for your applications. Test plans help you validate the functionality, performance, and security of your software, leading to higher customer satisfaction and reduced maintenance costs. Enabling Test Plans in Azure DevOps Enabling test plans in Azure DevOps is a straightforward process that can be done within the Azure DevOps portal. Follow these steps to enable test plans for your projects: Log in to your Azure DevOps account. Select the project for which you want to enable test plans. Go to the "Test Plans" tab in the project settings. Click on the "Enable test plans" button. Follow the on-screen instructions to set up your test plans. Once you've enabled test plans for your project, you can start creating test suites, test cases, and test configurations to kickstart your testing efforts. Key Features of Azure DevOps Test Plans Azure DevOps test plans offer a wide range of features to help you streamline your testing process and ensure better software quality. Some key features include: Test Suites: Create and manage test suites to group related test cases together. Test Cases: Define individual test cases with detailed steps, expected results, and assigned testers. Test Configurations: Define different configurations for running tests, such as browser versions or operating systems. Test Runs: Execute test cases and track the results in real-time. Test Reports: Generate comprehensive test reports to analyze test results and identify issues. Enhancing Software Testing with Azure DevOps By enabling test plans in Azure DevOps, you can elevate your software testing process and ensure a higher quality for your applications. With robust testing capabilities and seamless integration with your development workflow, Azure DevOps test plans empower teams to deliver bug-free software efficiently. Start leveraging Azure DevOps test plans today and transform the way you test your applications! azuredevops #testplans #softwaretesting #qualityassurance #devops #testautomation What is a Test Plan? It is undeniable that one of the many aspects that sets a company apart from others is the quality of the products and services they offer. One of the many reasons why quality products are essential in business is because they satisfy your clients, and good customer reviews establish your business' reputation. Whether you are managing software and application programming business or a start-up toy company, testing your product is immensely necessary. However, before you start engaging in a testing process, you need to layout a test plan first. A test plan is like a guide book. It is a detailed document that outlines the test strategies, objectives, schedules, and deliverables required for software project testing. The test plan also serves as the roadmap of your testing process, whose goal action plan is to make sure that there are no problems with the project before providing it to your clients. It also serves as documentation for future references. Did you know that due to a poor test plan, the recall of Takata airbags in 2016 has been the costliest at 26 billion dollars affecting up to 19 million cars? Aside from vehicle recalls, the most expensive products ever recalled due to a lack or poor test planning were Merck Vioxx ($8.9 billion losses), Volkswagen emissions ($7.3 billion loss), Pfizer Bextra ($3.3 billion loss), and the Toyota vehicle recall ($3.2 billion loss). With increased emphasis on product safety and efficacy testing, the United States Consumer Product Safety Commission reported only 214 product recalls in 2019, a 7% decrease from the previous year's list. The Benefits of Software System Testing It is important to make sure that your product is functioning correctly. Take Apple as an example. It will not be called the most successful company in history if not with their quality products such as iPhones. According to BBC News, one of the big things that made Apple is the revolution of the iPhone since it was launched in 2007. With its big hit in the market, it is more likely guaranteed that Apple products are of excellent quality. With this, it is more likely guaranteed that Apple ensured that all features in the product work at its best. Regardless of what type of business you are running, and established or start-up one, it is always necessary to ensure that all software system products that you provide are in great shape. It is an excellent tool for business optimization; hence it should not be neglected. So, whether you have a new software or product, it is essential to get it tested first. Here are some of the reasons why software system testing is necessary. Excellent Quality Product: The first advantage or benefit of software system testing is that it allows your business to produce excellent and reliable quality products. And, high-quality products attract the market.Satisfied Clients and Customers: It is frustrating to have customers knocking at your door asking for a refund because they are unsatisfied with the product. With software or product testing, you will have an excellent quality product that will satisfy the needs of clients. And satisfied and happy clients will more likely to avail of your products again.Increases Sales: When your clients find your products reliable and worthy of the price, they may recommend it to their family or friends. Hence, you will gain more customers, as well as increase your business sales plan.Cut Costs: Although software system testing will cost you some money, it will still save you some in the long run. Testing your software system products before launching and marketing plan it out to the public ensures that they function well and do not need constant fixing.Enhances User Experience: Following a testing process for all your company software and application products enhances user experience. This means that your products should be entirely free from errors that may cause inconvenience. How To Make a Test Plan A test plan does not only serve as a roadmap of the testing process, but it also serves as a means of communication of the team. It also sets forth standards about the process, such as what needs to be tested and what does not, the different testing strategies and approaches, etc. A professionally made test plan ensures that all features of the software product are covered and tested. If making a test plan from scratch is inconvenient for you, you can always make use of a ready-made template that you can find in this post. Simply download the sample template that suits your needs. Nonetheless, here are some simple steps on how to make an accurate test plan. Step 1: Analyze the Product Begin making a test plan, you must first analyze what the product is all about. It is important to state an overview of the product in the test plan. Software product analysis is necessary because it allows you to understand the significance of the product thoroughly. It allows you to know who are the end-users of the product, what it is used for, how it works, and what elements the product uses. Make sure that you have this information in your test plan. Step 2: Define the Testing Scope and Type Once you have already analyzed the software project, the next thing you need to do is to identify the testing scope and the testing type. The testing scope should include information about what components of the project to be tested. Doing this allows the team to have a clear understanding of what is tested and what is not. As for the testing type, it refers to a standard test procedure formulated to determine specific areas of the software project. The testing type depends on what part of the project is being tested. But, all testing types have the same goal, which is to ensure that the project is good before releasing it to clients. Step 3: Identify Testing Activities and Schedule Now that you have determined the testing scope and type, it is now time to identify each event in the testing process. In the test plan, list down all the testing activities to be performed with its corresponding schedule and estimation. Indicate also the person who is responsible for the task. Doing this is essential so that the testing team will be guided about the process. Nevertheless, make sure that your testing activities match your testing scope. This is to establish clarity and accuracy. Step 4: Incorporate Testing Strategies Once you have already identified the testing activities and its corresponding schedule, the next thing you need to have is testing strategies. In the test plan, list down the different approaches that need to be followed to execute the testing process. Remember, a good strategy makes you achieve all the product objectives. Make sure that the strategic plan is not only evident in the test plan but also for the whole testing team so that they will be guided on how to conduct the software project testing properly. Include also the testing resources and tools to be used for the testing process. Step 5: Generate a Test Criteria After that, the next thing you need to do is to generate and place test criteria on the test plan. In every testing process, test criteria are necessary because this is where the success of the project will be based. The test criteria will serve as your guide, whether each component of the project has met the required description and percentage to pass. The test criteria for the software project should be discussed with the team making sure that the criteria are accurate for the project. Step 6: Write a Test Result Finalize your test plan while writing the test results. By this, document the summary of the testing process and the project itself. If there are any features or issues that need to be fixed before the final approval, make sure to record them along with its action plan. Also, include the staff or member of the team that is responsible for doing it. Make sure that this section of the test plan is clearly stated and well-defined to avoid any confusion and misunderstanding that will jeopardize the system. FAQs The test plan is a document that outlines the testing process standard for a project prepared by the test lead or manager. A test plan is a detailed document that sets the testing information about a software product. It contains the guidelines for testing, tasks and schedule, testing resources, and testing environment. The four main stages of testing that need to be performed before the product launch are unit testing, integration testing, system testing, and acceptance testing. A test Execution strategy is a document that outlines the different approaches or styles in conducting software testing. On the other hand, a test plan is a written document that provides a concrete plan for the testing process. A Software Testing Life Cycle or STLC is a process with different tasks performed to improve the quality of a software system product. It has seven phases, which include the requirement phase, planning phase, analysis phase, design phase, implementation phase, execution phase, conclusion phase, and closure phase. It is no doubt that testing software, mobile application, or consumer products before making it available for the public is essential because it gives quality assurance and customer satisfaction, which are two of the many aspects of a successful business. However, product testing needs to follow a clear and accurate process to ensure a great result. That is why it is important to establish a well-made test plan that will serve as the testing team's guide throughout the process. A test plan can be both high-level or low-level, depending on your needs. But rest assured it is one of the best sample plans for the business that is necessary to ensure that your products function well and does what it promised. Type of document A test plan is a document detailing the objectives, resources, and processes for a specific test session for a software or hardware product. The plan typically contains a detailed understanding of the eventual workflow. A test plan documents the strategy that will be used to verify and ensure that a product or system meets its design specifications and other requirements. A test plan is usually prepared by or with significant input from test engineers.[1] Depending on the product and the responsibility of the organization to which the test plan applies, a test plan may include a strategy for one or more of the following: Design verification or compliance test – to be performed during the development or approval stages of the product, typically on a small sample of units. Manufacturing test or production test – to be performed during preparation or assembly of the product in an ongoing manner for purposes of performance verification and quality control. Acceptance test or commissioning test – to be performed at the time of delivery or installation of the product. Service and repair test – to be performed as required over the service life of the product. Regression test – to be performed on an existing operational product, to verify that existing functionality was not negatively affected when other aspects of the environment were changed (e.g., upgrading the platform on which an existing application runs). A complex system may have a high-level test plan to address the overall requirements and supporting test plans to address the design details of subsystems and components. Test plan document formats can be as varied as the products and organizations to which they apply. There are three major elements that should be described in the test plan: test coverage, test methods, and test responsibilities. These are also used in a formal test strategy.[2] Test coverage in the test plan states what requirements will be verified during what stages of the product life. Test coverage is derived from design specifications and other requirements, such as safety standards or regulatory codes, where each requirement or specification of the design ideally will have one or more corresponding means of verification. Test coverage for different product life stages may overlap but will not necessarily be exactly the same for all stages. For example, some requirements may be verified during design verification test, but not repeated during acceptance test. Test coverage also feeds back into the design process, since the product may have to be designed to allow test access. Test methods in the test plan state how test coverage will be implemented. Test methods may be determined by standards, regulatory agencies, or contractual agreement, or may have to be created new. Test methods also specify test equipment to be used in the performance of the tests and establish pass/fail criteria. Test methods used to verify hardware design requirements can range from very simple steps, such as visual inspection, to elaborate test procedures that are documented separately. Test responsibilities include what organizations will perform the test methods and at each stage of the product life. This allows test organizations to plan, acquire or develop test equipment and other resources necessary to implement the test methods for which they are responsible. Test responsibilities also include what data will be collected and how that data will be stored and reported (often referred to as "deliverables"). One outcome of a successful test plan should be a record or report of the verification of all design specifications and requirements as agreed upon by all parties. IEEE 829-2008, also known as the 829 Standard for Software Test Documentation, is an IEEE standard that specifies the form of a set of documents for use in defined stages of software testing, each stage potentially producing its own separate type of document.[3] These stages are: Test plan identifier Introduction Test items Features to be tested Features not to be tested Approach Item pass/fail criteria Suspension criteria and resumption requirements Test deliverables Testing tasks Environmental needs Responsibilities Staffing and training needs Schedule Risks and contingencies Approvals The IEEE documents that suggest what should be contained in a test plan are: 829-2008 IEEE Standard for Software and System Test Documentation[3] 829-1998 IEEE Standard for Software Test Documentation (superseded by 829-2008)[4] 829-1983 IEEE Standard for Software Test Documentation (superseded by 829-1998)[5] 1008-1987 IEEE Standard for Software Unit Testing[6] 1012-2004 IEEE Standard for Software Verification and Validation[7] 1012-1998 IEEE Standard for Software Verification and Validation (superseded by 1012-2004)[8] 1012-1986 IEEE Standard for Software Verification and Validation Plans (superseded by 1012-1998)[9] 1059-1993 IEEE Guide for Software Verification & Validation Plans (withdrawn)[10] Software testing Test suite Test case Test script Scenario testing Session-based testing IEEE 829 Ad hoc testing ^ Dale, Nell; Weems, Chip; Richards, Tim (2022-07-15). Programming and Problem Solving with C++. Jones & Bartlett Learning. ISBN 978-1-284-15732-1. ^ Laganà, Antonio; Gavrilova, Marina L.; Kumar, Vipin; Mun, Youngsong; Gervasi, Osvaldo; Tan, C. J. Kenneth (2004-05-07). Computational Science and Its Applications -- ICCSA 2004: International Conference, Assisi, Italy, May 14-17, 2004, Proceedings. Springer Science & Business Media. ISBN 978-3-540-22054-1. ^ a b 829-2008 — IEEE Standard for Software and System Test Documentation. 2008. doi:10.1109/IEEESTD.2008.4578383. ISBN 978-0-7381-5747-4. ^ 829-1998 — IEEE Standard for Software Test Documentation. 1998. doi:10.1109/IEEESTD.1998.88820. ISBN 0-7381-1443-X. ^ 829-1983 — IEEE Standard for Software Test Documentation. 1983. doi:10.1109/IEEESTD.1983.81615. ISBN 0-7381-1444-8. ^ 1008-1987 - IEEE Standard for Software Unit Testing. 1986. doi:10.1109/IEEESTD.1986.81001. ISBN 0-7381-0400-0. ^ 1012-2004 - IEEE Standard for Software Verification and Validation. 2005. doi:10.1109/IEEESTD.2005.96278. ISBN 978-0-7381-4642-3. ^ 1012-1998 - IEEE Standard for Software Verification and Validation. 1998. doi:10.1109/IEEESTD.1998.87820. ISBN 0-7381-0196-6. ^ 1012-1986 - IEEE Standard for Software Verification and Validation Plans. 1986. doi:10.1109/IEEESTD.1986.79647. ISBN 0-7381-0401-9. ^ 1059-1993 - IEEE Guide for Software Verification and Validation Plans. 1994. doi:10.1109/IEEESTD.1994.121430. ISBN 0-7381-2379-X. Public domain RUP test plan template at Sourceforge (templates are currently inaccessible but sample documents can be seen here: DBV Samples) Retrieved from " Within Microsoft Azure DevOps, Azure Test Plans is a potent solution that simplifies software testing through both automated and manual test management. Azure Test Plans helps teams increase test efficiency, regardless of the size of the project—from small project goals to enterprise-level applications.Do not pass this up! Examine the most important lessons you will learn:Makes Test Management Easier – Easily plan, carry out, and monitor automated and manual tests in one location.Smooth CI/CD Integration – To expedite software delivery, run tests automatically throughout your DevOps pipelines.Improved Tracking & Visibility – Easily track faults, get real-time information, and guarantee complete test coverage.Designed for Teams & Growth – It facilitates team collaboration, making it ideal for projects of all sizes.Increases Efficiency with Automation – Use Azure service Automation to execute dependable, repeatable tests and minimize manual labor. To begin, arrange test cases according to features or requirements and create a Software Test Suite in Azure Test Plans. Reduce human labor and guarantee consistency by using Azure Automation to run automated test suites. Software delivery is accelerated by continuous testing within CI/CD pipelines made possible by a smooth interaction with Microsoft Azure DevOps.Teams can increase test coverage, effectively track defects, and guarantee high-quality releases with less risk by using Azure Test Plans. What Is Azure?With features including networking, storage, processing power, and artificial intelligence, the Microsoft Azure cloud platform makes it possible to develop and operate apps effectively while maintaining scalability and flexibility. With Azure DevOps, organizations can manage test cases, run tests, and guarantee product quality with Azure Test Plans. While Azure Automation increases the productivity of the CI/CD pipeline by automating tasks, test suites aid in test organization. Azure services offer modern software development tools that are safe, scalable, and flexible for companies of all sizes. What Is Azure Test Plans?Software testing is much easier with Azure Test Plans. As a component of Azure DevOps, it provides teams with all the tools they need to manage automated and manual testing without becoming overwhelmed.The ability to construct and arrange your test cases is what makes Test Plans so awesome. Related tests can be bundled into suites, and you can observe the results as they are generated. You can have your tests run automatically whenever you push new code because it's built into Azure DevOps.Azure Test Plans can help you whether you're experimenting with exploratory testing or performing your standard suite of regression tests. Key Features of Azure Test Plans in Azure DevOpsDetailed Test Management: Develop, run, and monitor exploratory, automated, and manual test cases.Smooth CI/CD Integration: This feature allows agile processes to seamlessly integrate with pipelines for continuous testing.Compatibility: Across many environments is ensured by cross-browser and cross-platform testing.Test steps: These are parameterized and shared to improve test execution efficiency and reusability.Real-Time Reporting & Analytics: Offers information on defect tracking and test coverage.Bug tracking and traceability: help to improve defect resolution by connecting tests to work items.Collaboration and Scalability: Facilitates team-based testing for both startups and large corporations.Why Use Azure Test Plans for DevOps Testing?Within Azure DevOps, Azure Test Plans offer a smooth method for managing and carrying out tests. They guarantee the delivery of high-quality software by supporting automated, exploratory testing tools, and manual testing features. Teams may increase productivity and agility by integrating testing into Azure DevOps CI/CD pipelines using Test Plans. The platform makes it simpler to find and address problems by providing real-time reporting, defect tracking, and traceability. Additionally, Azure Test Plans facilitate cross-platform and cross-browser testing, guaranteeing a seamless user experience. Using Azure Test Plans improves cooperation, expedites releases, and guarantees dependable, error-free software for DevOps testing teams.Creating and Managing Test Cases in Azure DevOpsAzure DevOps Test Plans offer an organized method for effectively developing and maintaining test cases. Teams can better execute their work by defining test scenarios, setting parameters, and organizing test suites with Azure Test Plans. To ensure full traceability, the platform enables testers to connect test cases with requirements. For quicker validation, teams can automate tests or run them manually. In order to improve issue resolution, Azure DevOps Test Plans also include real-time reporting and defect tracking. Azure Test Plans may help organizations increase test coverage, optimize workflows, and confidently deliver high-quality software.Procedure for Developing Test Cases in Azure DevOps:Structured and effective testing is ensured by creating test cases in Azure DevOps Test Plans. Take these actions:Step 1: Open the Azure DevOps Test Plans page. Navigate to the Test Plans section after opening Azure DevOps.Step 2: Click "New Test Plan" and specify test suites to create a test plan.Step 3: Add Test Cases. Type in the parameters, intended outcomes, steps, and test title.Step 4: Assign Testers: Assign team members to test cases.Step 5: Conduct Tests: Carry out automated or manual tests.Step 6: Track & Analyze: To keep an eye on outcomes and fix bugs, use reports in Azure Test Plans.Teams can guarantee software testing quality and traceability using Azure DevOps Test Plans.How to Handle Test Case Management in Azure DevOps:Better traceability and efficient testing are guaranteed by effective test case management in Azure DevOps Test Plans. Take these actions:Access Azure Test Plans: Go to Azure DevOps and select Test Plans.Sort Test Suites: Assemble test cases into suites that are static, requirement-based, or query-based.Edit and Update Test Cases: Adjust procedures, anticipated outcomes, and parameters as necessary.Assign & Prioritize: Assign test cases to testers and establish priorities for their execution.Execute & Monitor Progress: For efficiency, run tests manually or automate them.Analyze Findings: To monitor errors and enhance quality, use Azure Test Plans' dashboards and reports.How to Link Test Cases with User Stories in Azure DevOpsImproved requirement validation and traceability are guaranteed when test cases and user stories are connected in Azure DevOps Test Plans. Take these actions:Login Azure Portal: Open Azure DevOps from the Azure Dashboard after logging into the Azure Portal.Navigate to Test Plans: After choosing your project, select Azure Test Plans from the left-hand menu.To open a test case: either click on an already-existing one or start from scratch.Link to a User Story: Look for and associate the user story in the "Linked Work Items" column.Save and Monitor Your Progress: To track test coverage and fix bugs, use the Azure Admin Portal.Look for updates in Azure Files if Azure is unavailable.Executing Test Runs in Azure DevOps: A Step-by-Step GuideEffective software functionality validation is ensured by doing out test runs in Azure DevOps Test Plans. Take these actions:Open the Azure Test Plans page: Navigate to the Test Plans section after launching Azure DevOps.Choose a Test Suite: Decide which test suite or test plan includes the test cases.Start a Test Run: To run automatic or manual tests, click "Run."Log Results & Defects: Report errors and mark test cases as blocked, failed, or succeeded.Examine Reports: Gain insight into execution progress and failures by utilizing Azure Test Plans analytics.Teams can easily monitor and improve software quality with Azure DevOps Test Plans.How to Integrate Azure Test Plans with CI/CD PipelinesAzure Test Plans can be integrated with Azure Pipelines to improve testing and expedite release cycles. This is how you do it:To set up: an Azure DevOps pipeline, go to the Azure Pipelines "Pipelines" section and create or edit your pipeline.Add the Test Execution Task: Add tasks from Azure Test Plans or Visual Studio Test to run test cases from your test suite.Turn on Azure Automation to optimize your testing efforts by setting it up to run tests automatically following builds or deployments.Track and Evaluate Outcomes: Watch test results, spot mistakes, and improve quality with Azure Test Plans.For more dependable software delivery, teams can conduct testing tasks constantly by integrating Azure Test Plans with Azure Pipelines.How to Track and Analyze Test Results in Azure Test PlansAzure Test Plans' tracking and analysis of test results aid in ensuring software quality and spotting problems early. Take these actions:Execute Test Suite: Use automated pipelines in Azure DevOps or Azure Test Plans to run tests.View Test Results: To view test execution information, including pass/fail status, select the "Runs" tab in Azure Test Plans.Examine Defects: To improve defect management, connect unsuccessful tests to work items.Make use of Azure Automation to automate test reporting and execution for ongoing feedback.Make better decisions by using Azure's Reports and Dashboards to visualize test coverage and trends.Teams may improve software dependability by effectively tracking and analyzing test results with Azure Test Plans.Manual vs. Automated Testing in Azure Test PlansBoth automated and manual testing are essential components of Azure Test Plans that guarantee program quality. For exploratory testing and situations needing human intuition, a manual test plan is perfect. It enables direct interaction between testers and the application, yielding insightful feedback. However, for repeatable and regression testing, automated testing with Azure DevOps services and Azure automation is effective. Teams can perform tests more quickly and consistently by incorporating automated tests into a test suite. Both approaches are supported by Azure Test Plans, enabling teams to select the most appropriate strategy depending on project requirements and test requirements.Best Practices for Using Azure Test Plans EffectivelyUse these best practices to get the most out of Azure Test Plans:Organize Test Suites: To improve management, arrange your tests into logical test suites.Use Azure DevOps Automation: To boost productivity and cut down on manual labor, use Azure DevOps to automate repeated tests.Link Work Items: To guarantee traceability and improve teamwork, link tests to user stories and problems.Employ exploratory testing to address various testing requirements by combining automated and manual testing.Track Test Results: To find and fix problems early, examine test results frequently in Azure Test Plans.Teams may improve software quality in Azure, increase test coverage, and optimize workflows by implementing these best practices.Common Challenges and How to Overcome ThemTeams may run into issues like inconsistent test coverage, delayed test execution, and test suite management when combining Azure Test Plans with Azure DevOps. Take into account the following tactics to deal with these problems:Optimize Test Suites: To enhance management and expedite execution, group tests into requirement-based, query-based test suites.Automate Testing: To minimize human error and automate repetitive testing tasks, use Azure DevOps Automation.Track Progress: You can keep an eye on the real-time status of your tests by using the Execute Tab, Visual Studio Test, and picture action logs.Improve Cooperation: Enhance teamwork for more effective testing by connecting Azure DevOps services with other technologies to promote better communication.Through the implementation of these testing techniques, teams may guarantee improved quality assurance, more seamless user acceptability testing, and faster software delivery.Enhancing Security & Compliance in Azure Test PlansAzure Test Plans must provide security and compliance in order to safeguard sensitive data and satisfy legal obligations. First, you can limit access to test plans and test suites with Azure DevOps' strong role-based access control. To ensure that security measures are applied consistently across all environments, implement Azure DevOps automation for consistent test execution. Maintain traceability by connecting test cases to pertinent work items and auditing test results on a regular basis. Use secure pipelines for automated testing and enforce data protection regulations within Azure Test Plans to improve compliance. You can guarantee a safe and legal testing procedure in Azure by adhering to these guidelines.SummaryThis blog has explained how Azure Test Plans' extensive testing architecture enhances the software development process. Teams may efficiently manage testing efforts and guarantee sufficient test coverage by employing requirement-based, query-based test suites, and static suites. Functional tests and user acceptability testing are among the testing methodologies that Azure Test Plans provide, whether automated or manual. We looked at how integrating Azure DevOps automation and DevOps tools may improve testing procedures and speed up software delivery. The software offers useful insights into test execution by enabling real-time reporting and tracking of test results. Furthermore, tools like exploratory testing services and browser-based test management solutions facilitate testing workflows, guaranteeing a smooth software development lifecycle and better software.People Also AskWhat are the different types of test plans?Different types of test plans include Requirement-Based Test Suites, user acceptance testing, basic test suites, and testing strategies for various testing activities.What is the difference between a test suite and a test plan?A test plan outlines overall testing activities, while a test suite groups related query-based suites, including user acceptance testing and planned testing.What is a test execution plan?A test execution plan outlines testing activities, individual test steps, and testing environments for running tests using Visual Studio Test and testing tools.Is Azure free for testing?Azure offers free tiers for testing, providing basic test suites and tools for modern software development teams to manage current status and image action logs.How do I delete test plans in Azure Devops?To delete test plans in Azure DevOps, go to the Test Plans section, select the plan, and click Delete using the browser extension. Azure DevOps is a powerful suite of tools that streamlines the planning, development, testing, and delivery of software. It seamlessly integrates into DevOps workflows, helping to maintain software quality and reliability. To ensure the updates and the workflow work well

and don't cause any issues in the future, performing Azure DevOps testing is important for maintaining software quality throughout the DevOps lifecycle. It ensures that software updates function as intended, remain defect-free, and meet business and customer requirements. What Is Azure Test Plans? Azure Test Plans is a modern test management module integrated with the Azure DevOps ecosystem. It is a powerful Azure DevOps testing tool that enables teams to manage test plans, test suites, and test cases throughout the Software Development Life Cycle (SDLC). This Azure DevOps testing solution supports both manual and automated testing needs, making it an invaluable tool for modern software development teams. At its core, Azure Test Plans is designed to be a centralized testing environment where development teams, Quality Assurance (QA) professionals, and business analysts can collaborate effectively. It acts as a centralized repository for all testing activities, ensuring that everyone involved in the project has access to the same testing information and can work collaboratively towards quality objectives. Azure Test Plans offer a robust set of tools designed to streamline testing workflows and ensure comprehensive test coverage. Below are some of its core key features: Test Planning: Streamline the management test plans, suites, and cases to align with project objectives. Test Execution: Perform manual and automated tests with real-time result tracking. Test Data and Parameters: Simplify test data management for parameterized testing scenarios. Exploratory Testing: Utilize browser extensions for ad-hoc and exploratory testing. Test Automation: Integrate automated testing seamlessly into CI/CD pipelines. Bug Tracking: Capture detailed diagnostics, including screenshots and videos, to resolve issues efficiently. Security and Permissions: Implement role-based access and secure testing environments. Note Create complete test plans, ensure software quality, and streamline your testing process. Try LambdaTest Today! How Azure Test Plans Work? Azure Test Plans simplifies test management with three core artifacts, namely, Test Plan, Test Suite, and Test Case. As part of the Azure DevOps testing ecosystem, it helps enhance the reusability via the Shared Steps and Shared Parameters are work items stored in Azure DevOps. These features enable efficient management of common test procedures and variable test data, helping you streamline the phases of the DevOps lifecycle. Here is how they work: Test Plan: A container for test suites and test cases, including configurations, shared test steps, and parameters. It allows the grouping of test suites or individual test cases and can use static, requirement-based, or query-based test suites. Test Suite: A collection of test cases grouped under a common scenario within a test plan. It simplifies tracking the status of related test cases. Test Case: A set of steps designed to validate specific aspects of code or deployment. Test cases can exist independently within a test plan or be part of the suite. Shared Steps: Reusable test procedures that can be referenced across multiple test cases to avoid duplication. Shared Parameters: Parameters (marked with "@" in test steps) that allow reusing test data across multiple test cases with different data sets. How to Create a Test Plan in Azure DevOps? Creating a test plan in Azure DevOps involves several steps, from defining test cases to assigning testers and organizing test suites to ensure complete test coverage. As part of Azure DevOps testing workflows, this helps streamline the testing workflow and track the progress effectiveness. To effectively create and manage test plans in Azure DevOps, it's essential to meet specific prerequisites concerning access and project setup. Access Requirements: Access Level: Users must have at least a Basic access level to execute tests and mark outcomes. To create and manage test plans, test suites, and test cases, a Basic + Test Plans access level is required. Alternatively, subscriptions like Visual Studio Enterprise, Visual Studio Test Professional, or MSDN Platforms also grant equivalent access. Permissions: Ensure that permissions are configured appropriately: Edit work items in this node: Allows adding or modifying test plans, test suites, test cases, and related work items. Manage test plans: Enables modification of test plan properties, including build and test settings. Manage test suites: Permits creation, deletion, and modification of test suites, as well as managing test cases within them. Project Setup: Project Access: Users should be added to the relevant Azure DevOps project to gain access to its resources. Area Path Configuration: Set the appropriate area paths to organize work items effectively. Configure permissions under these area paths to control access to test artifacts. Iteration Path Configuration: Define iteration paths to manage sprints or release cycles, ensuring that test plans align with project timelines. Creating a New Test Plan Follow the below steps to access the test plan module: Log in to your Azure DevOps account and navigate to the project where you want to manage test plans. In the left-hand menu, locate and click on Test Plans. This module serves as the central hub for managing all testing activities. If the Test Plans module isn't visible, ensure that your access level includes the Test Plans extension. Inside the Test Plans hub, click on the New Test Plan button. Provide the following details: Name: A descriptive title for the test plan. Area Path: Select the relevant project area to which this test plan belongs. Iteration Path: Specify the sprint or release cycle this test plan is associated with. Click on the Create button to create a test plan. With the above steps, you have now created a test plan under which you can add various test suites and test cases. Adding Test Suites Test suites allow you to organize test cases within your test plan. There are three types of test suites in Azure DevOps: Static Test Suites: It is used to group manual test cases often used for nested suites or exploratory testing. To create, click New Suite → Static suite. Name the suite appropriately and save it. Requirement-Based Test Suites: Automatically include all test cases linked to a specific requirement, enabling traceability and alignment with acceptance criteria. They support collaboration across roles like QA, developers, and BAs by creating a shared source of truth. To create, click New Suite → Requirement based suite. Select a requirement to associate this creates traceability between requirements and test cases. Query-Based Test Suites: Use ad-hoc queries to group test cases based on specific criteria, such as tags or iteration cycles. These suites are ideal for flexible, requirement-independent test scenarios. To create, click New Suite → Query based suite. Define a query to filter and include test cases meeting those conditions. After creating a test suite, teams can execute the associated test cases, ensuring requirements with development goals are met. Creating Test Cases Once test suites are in place, create the test cases under them to organize and group the test cases. Here's how to create test cases: Select a test suite under your test plan. Click New Test Case. Fill in the details: Title: Provide a concise, descriptive title for the test case. Steps: List the steps to perform the test and include expected outcomes for each step. Priority: Set the priority level (High, Medium, Low) based on the test case's importance. Tags: Use tags to group or categorize test cases. Save the test case. You can create additional test cases as needed. Assigning Configurations and Testers Test configuration and tester assignment are crucial steps in Azure DevOps test management that ensure proper test coverage and responsibility allocation. Here's how to manage these settings: Assigning Configurations: Select a test suite or specific test cases. Open the context menu and choose the Assign configuration option. Select configurations (e.g., OS, browsers, devices) to ensure the test plan covers diverse scenarios. Save the configuration assignments. Assigning Testers: Select the test suite or test cases you want to assign. Open the context menu and select Assign testers to run all tests. Add team members responsible for executing these tests. Save the assignments. How to Execute Test Cases on Azure DevOps Test Plan? Efficient execution of test cases is crucial for maintaining software quality. Azure DevOps testing provides a robust solution to facilitate this process, ensuring comprehensive test coverage and streamlined issue tracking. Running Azure DevOps testing can be done in two ways: Manual Testing: Access Test Runner: Navigate to the Test Plans module and select the desired test case. Click on Run for a web application to launch the Test Runner. Execute Steps: Follow each test step as outlined, marking them as pass or fail based on the outcomes. Capture Observations: Utilize the Test Runner to record any observations, take screenshots, or note comments during execution. Automated Testing: Integration: Configure automated tests to run as part of your build or release pipelines. Execution: Upon triggering the pipeline, tests execute automatically, and results are captured within Azure DevOps. How to Manage and Analyze Test Results? After executing tests, analyzing the results is essential for assessing application quality and identifying areas for improvement. Viewing Test Reports: Test Results Overview: Access the Runs tab within the Test Plans module to view the status of test executions, including pass rates and failure details. Detailed Reports: For automated tests, navigate to the Pipelines section and select the relevant build or release to review detailed test reports. Analyzing Metrics Progress Reports: Utilize the Progress report feature to track the status of planned tests, monitor testing progress, and analyze metrics such as pass/fail rates and test execution trends. Test Analytics: Leverage Test Analytics to gain near real-time visibility into your test data, helping to improve pipeline efficiency and identify bottlenecks. Continuous Improvement Identify Patterns: Analyze test results to detect recurring issues or patterns that may indicate underlying problems in the codebase. Refine Test Cases: Based on insights gained, update and improve test cases to enhance coverage and effectiveness. Feedback Loop: Establish a feedback loop with development teams to address defects promptly and iteratively improve the software. Common Challenges and Troubleshooting Azure DevOps testing helps in managing and executing tests within the SDLC. However, teams often encounter challenges such as test flakiness, managing test data, and resolving permission issues. Addressing these challenges is crucial for maintaining efficient and reliable Azure DevOps testing processes. Challenge 1: Handling Test Flakiness Tests produce inconsistent results, passing or failing intermittently without any changes to the code or environment. Solution: Flaky Test Management: Use Azure DevOps to detect and manage flaky tests, tagging them to prevent build failures. Test Rerun Configuration: Configure test reruns to isolate genuine issues from flaky behaviors. Challenge 2: Managing Test Data Ensuring data consistency, handling sensitive information, and maintaining the availability of data across different environments. Solution: Test Configurations: Define variables like OS, browsers, etc., for comprehensive test coverage. Data Management Tools: Use tools to create, mask, and provision test data reflecting production environments. Challenge 3: Resolving Permission Issues Inadequate access control leads to unauthorized access, inability to edit/view test artifacts, and collaborative challenges. Solution: Access Levels and Permissions: Configure permissions for specific tasks like managing test plans and editing work items. Stakeholder Access: Ensure users have appropriate access levels (e.g., Basic or Basic + Test Plans). To overcome such challenges, integrating Azure DevOps Pipelines with cloud-based platforms helps streamline the DevOps and testing workflows by providing scalable infrastructure and enabling automated, cross-environment testing. A cloud testing platform allows you to integrate with CI/CD tools to enhance and accelerate issue detection and resolution, ensuring seamless application performance and reducing time-to-market. Azure DevOps helps to automatically build, test, and deploy your projects with Continuous Integration in various production environments. So your automation test scripts can run successfully on Azure Pipeline. However, executing test scripts in Azure DevOps testing can sometimes be time-consuming and hectic. This is where the LambdaTest plugin can be used in the Azure pipeline to ease the workflow and automate the tests. LambdaTest is an AI-native test execution platform that lets you perform manual and automated tests at scale across 5000+ real devices, browsers and OS combinations. This platform not only allows you to run tests but also integrates with Azure and other DevOps tools to ensure a smooth feedback loop, testing, and DevOps workflow. Check LambdaTest Extension On Azure DevOps Marketplace There are a few benefits mentioned below when using the LambdaTest plugin for Azure DevOps testing. Simple Account Integration: You can easily set up your LambdaTest account in Azure Pipeline using your account credentials, enabling a smooth and quick integration process. Enhanced Environment Testing: Use the LambdaTest Tunnel to test various production environments, such as internal, development, and staging. Streamlined Test Results Fetching: Embed or fetch LambdaTest test results in your project's job results, as it allows easy access to key test data for improved tracking and reporting. Detailed Test Insights: Get in-depth visibility of your test execution, capture screenshots, mark bugs, plot graphs, and more; you can view everything from the LambdaTest automation dashboard. Install LambdaTest Extension from the Azure DevOps Marketplace Follow the below steps to install the LambdaTest extension: Go to the Azure DevOps marketplace and, under the Azure DevOps tab, search for LambdaTest in the search box. In the results window, you will see an option showing the LambdaTest extension. Open the found result and click on the Get it free button to get the LambdaTest Extension for free in your organization. Select an Azure DevOps organization and click "Install." The LambdaTest extension will be installed in this organization. Once finished, you will see Done at the top, indicating that the LambdaTest Extension has been successfully installed. You can proceed to your organization to see the extension. By integrating LambdaTest into Azure DevOps testing pipelines, teams can significantly improve test efficiency, enhance collaboration, and reduce time-to-market. To integrate this plugin, you can also follow the detailed guide on integrating LambdaTest with Azure Pipelines. This support document helps you set up the plugin and provides a thorough guide on how to view results on LambdaTest. Best Practices for Azure DevOps Test Plans When creating a test plan in Azure DevOps, consider the following best practices to ensure effective testing and successful software releases: Define clear test objectives and scope, including which features need testing and specific quality goals you want to achieve through your testing efforts. Organize test cases into logical test suites based on functionality, linking them directly to requirements and user stories for traceability. Establish comprehensive test environments (dev, staging, prod) and prepare diverse test data sets covering valid, invalid, and boundary cases. Choose appropriate testing strategies combining manual, automated, and exploratory testing approaches based on project needs and risks. Assign roles and responsibilities within the team, specifying who will create, execute, and review test cases. Integrate the test plan seamlessly with your Agile development process, aligning with sprints and releases while regularly reviewing and updating based on project progress. Conclusion Azure DevOps Test Plans offers a comprehensive and integrated test management solution that empowers teams to plan, execute, and monitor their testing activities effectively throughout the software development lifecycle. As organizations adopt Agile and DevOps methodologies, the relevance of Azure Test Plans has grown, helping teams maintain high standards of software quality while adapting to fast-paced development environments. Prominent controversies surrounding Azure Test Plans often relate to its integration complexities and the learning curve associated with its comprehensive feature set, which some users may find overwhelming. Additionally, the platform's reliance on Azure DevOps necessitates a commitment to the Microsoft ecosystem, which can be a consideration for organizations evaluating alternative solutions. The future development of Azure Test Plans is focused on enhancing the capabilities of test management and integrating more advanced features to support the evolving needs of software development teams. One of the key areas of improvement is the continued emphasis on combining manual and automated testing processes, fostering a collaborative environment where all team members contribute to quality assurance efforts. This shift reflects the modern understanding that quality is a shared responsibility across development, testing, and product management roles. Nevertheless, Azure Test Plans continue to evolve, addressing user feedback and expanding its functionalities to meet the changing landscape of software testing. As part of a broader movement toward continuous improvement in testing practices, Azure Test Plans is positioned as a vital tool for teams aiming to enhance their software delivery processes, ensure compliance with business requirements, and foster a culture of collaboration and feedback across all stages of development. You must select a test suite from the Azure Test Plans and run the test suites. Yes, it has a test management tool, named Azure Test Plans. Azure DevOps offers Requirement-Based Test Suites (linked to specific requirements), Query-Based Test Suites (grouped by specific criteria), and Static Test Suites (manually selected test cases). When a test fails, Test Runner can automatically create bugs with pre-populated details, including test steps, system information, and diagnostic data, while maintaining traceability with test cases and requirements. The three core artifacts are the Test Plan (container for suites and cases), Test Suite (collection of related test cases), and Test Case (set of steps to validate specific aspects). Citations Software development keeps changing consistently, and we often need to update our software quickly. To make sure these updates work well and don't cause problems in the future, we need to use good testing methods and strategies. One of these strategies is to create test plan in Azure DevOps, a powerful tool designed to simplify the development process and ensure the delivery of robust and error-free applications.In this article, we'll teach you how to use Azure DevOps, how to create a test plan in Azure DevOps, and how they work. Whether you are a seasoned software developer or a newcomer to Azure DevOps, you'll find practical tips in this guide to optimize your testing procedures and boost the quality of your software releases.Azure test plans: how to create and use themAzure DevOps is a comprehensive platform that supports the entire software development lifecycle and equips the development team with tools to plan, develop, test, and deliver software efficiently. Among its many features, Azure DevOps test plans emerge as a critical aspect for managing and executing testing activities.What is the Azure Test Plan?Azure Test Plan is a test management and execution toolset offered by Microsoft as part of its Azure DevOps services. It assists software developers and testers in managing testing efforts in the software development lifecycle.Azure Test Plan is an effective tool for development teams that follows Agile and DevOps development methodologies, as it helps to ensure the quality of software through effective test management and execution. It offers flexibility in terms of manual and automated testing, making it suitable for a wide range of testing scenarios.Below are the key capabilities of the Azure Test Plan.With Azure test Plan, you can create test plans, define test suites, and organize test cases. Working collaboratively, testers and test managers can create test cases based on specific requirements.Azure Test plan allows testers to execute test cases and store their results. On top of that, the tester can mark the test case as fail, pass, blocked, or not executed.It supports data-driven testing where the users can run the same test case with multiple sets of input data to verify how an application behaves under various conditions.It allows users to perform exploratory testing sessions where testers, instead of following scripted test cases, explore the application dynamically. It's an unscripted testing method where testers rely on their domain knowledge and experience to detect usability issues and other problems within the software.It integrates with automation tools and frameworks and allows users to automate the execution of test cases and include automation tests in their test plans.You can track bugs and issues directly from the Azure DevOps series, making it easier to track and follow up on bugs in your application ahead of the deployment.You can manage access and permissions for different team members to control who can create, execute, and manage tests.How do they work?Azure DevOps test plans provide an efficient way to manage testing activities within the Azure DevOps environment. Here is how they work:Test Plans in Azure DevOps allow you to plan your testing efforts. You can create test suites and organize test cases within these suites. These test cases can be designed to cover different aspects of your application, such as functionality, integration, or regression testing.You can create test cases within the Azure DevOps test plans. Each test case includes details about what needs to be tested, steps to follow, and expected results. These test cases act as a roadmap for your testing activities.When it's time to start testing, you can assign test cases to testers and the testing environment. Testers can then execute the test cases, follow the defined steps, and provide feedback on the results.Azure DevOps Test Plans offer traceability features. This means you can link test cases to user stories, features, or requirements, ensuring testing meets your project goals. On top of that, this traceability helps in tracking progress and ensures comprehensive coverage.Test results and reportingAs testers execute test cases, they can mark them as pass, fail, or blocked, and provide comments. Azure DevOps captures these results and provides real-time reporting on the testing progress. This reporting is useful for testers and team managers as it enables them to monitor the quality of the application.Bugs and defect managementIf a tester identifies a bug or issue during the testing, they can create a bug directly from the test case. This integrates defect management into your testing process, ensuring that issues are properly tackled and addressed.Integration with Azure pipelineAzure DevOps test plans can be easily integrated with Azure CI/CD pipelines. This integration allows you to automate the execution of test cases as part of your build and release pipeline, making testing an integral part of your development process.Azure DevOps promotes collaboration among team members. Testers, developers, and project managers can work together within the same environment, making communication and issue resolution more efficient.Azure DevOps retains historical data, allowing you to analyze trends, track improvements, and make data-driven decisions for future releases.How to use Azure DevOps test plans?Learn how to use Azure DevOps Test Plans to assist test objectives. You can use the below-mentioned sections of the Microsoft Azure DevOps "Test Plans" page to complete the associated tasks.The Mine PageThe Mine Page shows a list of test plans that are important to you, including both team-related plans and your favorite selections. You have the option to expand or hide plans for each team, simplifying the process of finding and reviewing particular team plans. Besides, you can use the shortcut menu for tasks like editing or removing the chosen plan.The Filter option allows you to sort and locate test plans, specifically when dealing with a long list of plans. You can apply filters based on the plan's name, associated team, current status, or the specific development iteration it belongs to. These filters help you narrow down your search and find the plans you're interested in more effectively.The "All" pageThe All page displays a list of all available test plans. You can apply filters, edit, and delete plans through the shortcut menu, just as you would on the Mine page. This page also allows you to include any test plan in your list of favorites.While viewing a plan, you have the option to include it in your list of favorites.Test plan headerThe test plan header allows you to perform one of these tasks:You can choose to mark or unmark a test plan as a favorite.Access the test plan you like the most.Check the test plan iteration path to see if it's marked as current or past.Click on the view report link to go to the test progress report.Return to the All/Mine test plans page by clicking on the "All test plans" option.Test plan Context MenuSelect "more" option to access a test plan's context menu which, which allows you to perform one of these actions:Copy test plan: Immediately create a copy of an existing test plan.Edit test plan: Customize the test plan form by adding details to the description or discussion.Test plan settings: You can configure the test run settings which helps to connect them with the build or release properties, and set up the test outcome settings.Adjust settings for the test planSelect the test plan parameters for setting up test executions or recording test results.Copy test planIt's a good practice to make a fresh test plan for each sprint or release. You can duplicate the test plan from the previous cycle, and with just a few adjustments, the duplicated plan is set for the new sprint. You can perform this task with the Copy test plan menu option. Eventually, you can create a copy of an existing test plan within the same project.Test suite header tasksThe Test suite header provides users the capability to perform the following tasks.Expand/Collapse: This toolbar provides options to expand all or collapse all, which can be used to reveal or hide the suite hierarchy tree.Show test nodes from child suites: Select the Show test points from the child suites option in the toolbar, which is available on the execute tab. This option enables you to see all the test points associated with the current suite and its sub-suites in a single view. This simplifies the process of managing test points as you won't need to navigate individual suites one by one.Order suites: You can rearrange the sequence of suites or transfer them between different suite hierarchies in the test plan by simply dragging and dropping them.Test suites Context Menu optionChoose the "more" option menu to access a test suite's context menu. Then proceed with one of the actions below:Create new test suites: You have the option to create new suites and choose from three different types:Choose Static Suite when you want to structure your tests in a suite that resembles a folder.Select the Requirement-based suite for a direct combination of requirementsChoose a Query-based suite to manage test cases based on specific query criteria.Assign configurations: Specify browser configurations for the suite, like Chrome and Firefox. These configurations will be used for all test cases in the suite, whether they already exist or added in the future.Export: You can export the properties of the test plan, the properties of test suites, and other details as either an email attachment or by saving them as a PDF document.Open and access test suite work item: Allows you to customize the work item form for the test suite, granting the user complete control over the work item fields.Assign a user (developer/QA) to run all tests: This feature is highly valuable for User Acceptance Testing (UAT) use cases in which multiple testers from various departments can execute identical tests.Rename/Delete: You can use these options to manage the suite's name or to remove both the suite and its contents from the test plan.Import test suites: This option helps to bring in test cases located in different suites within the same project or even from other projects.Export Test Suite DialogYou can have the option to reutilize the suites you've generated and include them in your current test suite. Moreover, you can bring it into the project, test plan, and test suite that contains the test you want to import.Once you select the test suite, the structure of the chosen suite and its associated test cases will be imported into the current plan.It is pertinent to mention that test cases are added as references, not duplicates or copies. Besides, it's not possible to import test suites from within the identical test plan.Execute testsUse the "Execute" tab to allocate test points or run tests.This leads to a question what exactly constitutes a "Test point"? Test cases by themselves cannot be directly executed. When you include a test case in a test point, it generates a test point. A test point contains a unique combination of test suites, test cases, test suites, configuration, and a tester.For instance, if you have a test case named "Test checkout process" and you add two configurations for Microsoft Edge and Chrome web browsers, you have two test points. You can easily execute each one of these test points. Upon the execution of these test points, results are generated.In the "Execute" tab, you'll find the most recent information for the test point. The same test cases can be used again for multiple purposes. By including them in a test plan or suite, test points are generated. By executing test points, you can determine the quality of a product or service that is currently in the development phase.How to choose a test plan?Below are the steps you can follow to choose a test plan in Azure DevOps.Sign in to your Azure DevOps account by providing your credentials.Go to the "Test Plans" section of your Azure DevOps project. This section is usually found in the left-hand menu or through the "Test" tab at the top of the page.In the "Test Plans" section, you'll see a list of all the available test plans. These plans are organized by folders. You can click on these folders to view the test plans contained within.Click on the test plan you want to work with. This will open the test plan and allow you to view its details and associated test cases.Inside the test plan, you can navigate through the various sections like test suites, test cases, test runs, and results to manage and execute your tests.Depending on your role and permissions, you can create, edit, or execute test cases and test runs within the selected test plan.How to create a test plan in Azure DevOpsYou can create test plans and test suites to monitor manual testing during sprints. This way, you can easily determine when the testing for a particular sprint or milestone has been finished. Below are the practical steps to create test plansFrom the web portal, open your project and navigate to the "Test Plans". If you already have a test plan, select "Test Plans" to access the page that contains a list of all the test plans.2. Once you gain access to the "Test plans" page, click on "New Test Plan" to create a new test plan for your current sprint.3. In the New Test Plan, enter a name for the test plan. Ensure that the area path and iteration are configured accurately, then click the "Create" button.ConclusionCreate test plan in azure devops is a fundamental process for ensuring the quality and reliability of your software projects. By following the steps discussed in this article, you can easily create test plans in Azure DevOps that meet your project requirements and goalsBy using Azure DevOps to create test suites, arrange your test cases, and carry out tests in an organized way, you'll have valuable tools at your hand that help you monitor the quality of your software and make it better.However, it is important to realize that the effectiveness of your Azure Test Plans not only relies on the tools but also on your testing strategy, collaboration, and the quality of your test cases.To ensure the success of your testing efforts, it's essential to regularly review and update your test plans and test cases to cater to constantly changing project needs and requirements. In this lab, you will learn how to use Azure DevOps to manage your project's testing lifecycle. This project will guide you through creating test plans designed to efficiently validate your software milestones. You will also create and execute manual tests that can be consistently reproduced over the course of each release.In this exercise, you will learn how to create and manage test plans, test suites and test cases. Navigate to your team project on Azure DevOps. Select Test Plans to navigate to the Test Hub. The test hub provides a central place for all test planning, execution, and analysis. In general, every major milestone in a project should have its own test plan. Within each test plan are test suites, which are collections of test cases (and optionally other test suites) designed to validate a work item, such as a feature implementation or bug fix. Each test case is designed to confirm a specific behavior and may belong to one or more test suites. The Parts Unlimited project has one test plan, which is under the Parts Unlimited Team and called Parts Unlimited_TestPlan1. Select the Parts Unlimited_TestPlan1 test plan. Select the suite of tests for the story As a customer, I would like to store my credit card details securely. This suite of tests focuses on that work item, which happens to be a feature. Note that the work item numbers will vary every time you generate demo data for a lab. On the right side you can see that this test suite has three test cases designed to confirm expected behavior of the feature implementation. Double-click the Verify that user is allowed to save his credit card detail test case. This dialog provides all the info you need on this test case. Locate the Related Work panel and note that this test case is linked to the suite it belongs to. Click the work item to navigate to it. In the test suite, we can see all of the linked work items, which happen to be the test cases. However, it's not yet associated with the feature it's designed to test, which we can link now. Click Add link | Existing item. Set the Link type to Parent and search for "credit card". Select the Feature for Credit Card Purchase. Click OK. The parent feature is now associated with the suite that tests it and anyone can navigate between them to view their relationship relative to the other work items involved. Click Save & Close. Dismiss the original test case dialog. Sometimes a set of test cases should be run in a specific order to maximize efficiency. Click Order tests to specify the order these test cases should be run. While these test cases could be run separately to confirm the behavior, it probably makes more sense to run the test case that rejects invalid cards first. Then, the tester can confirm that a valid card can be saved, followed by the test case for editing a saved card. Drag and drop the second test case above the first and click Done. You can now see that the Order has been updated and that the list is now sorted by it. Another significant aspect of testing has to do with the environment each test is run in. For this web app, the browser and operating system are key considerations. Right now all the tests only use one configuration: Windows 10. Select the Configurations tab. Note that there is one existing configuration for Windows 10. Each test configuration includes a name and a description, as well as a set of customizable Configuration variables. This project has one configuration variable set for Operating System. You can easily add more and/or edit the available entries for each. Click Add configuration variable. Select the Browser variable and set it to Microsoft Edge. Click Save to save the configuration. Now let's suppose the test team has acquired an iPhone X and wants to add it into the test matrix. It's really easy to register this environment as a new configuration so that test cases can specify it. However, before adding it, we'll need an Operating System option for iOS 10. Click the Operating System configuration variable. Click Add new value and add an entry for iOS 12. Click Save. Now we have everything we need to add the iPhone X. Click the Add dropdown and select New test configuration. Set the Name to "iPhone X". Click Add configuration variable twice and set the Browser to Safari and Operating System to iOS 12. Click Save to save the new configuration. Return to the Test Plans tab. Click the dropdown next to the test suite we've been working with so far and select Assign configurations to test suite. Check the iPhone X option and click Save. Notice that each test case has been duplicated with an additional configuration for iPhone X. Now each environment can be tested and tracked separately. Expand the dropdown next to the test plan and select New static suite. A static suite of test cases is a suite where the cases have been manually assigned. You can also create suites based on common requirements (requirement-based suite) or a query of test cases and/or work items (query-based suite). Set the name of the new suite to "Shipping tests". These tests will all focus on functionality related to shipping. Remember that you can easily share test cases across suites, so there's minimal redundancy when having a lot of overlapping suites. Expand the dropdown next to the newly created suite and select New requirement-based suite. You could customize the query used to specify which requirements are retrieved, but just leave the defaults and click Run query. Locate and select the three product backlog items related to shipping. Click Create suites to create a test suite for each. Select one of the newly created suites, such as the one associated with tracking package status. While you can create test cases one at a time, it's sometimes easier to use a grid layout to quickly add many test cases. In the test cases panel, select New | New test case using grid. Enter a few test cases and click the Save All button. The Title will be the eventual title of the test case. Step Action will be the first (and possibly only) step of the test. If that step has an expected result, you can specify it as Step Expected Result. You can optionally continue to add and edit work items in the grid view. When satisfied, return back to the list view by clicking the View: Grid toggle. The list view shows the same data, but in a different view. Another option to create suites is via work item query. Expand the dropdown next to the Shipping tests suite and select new query-based suite. Let's say you wanted to create a test suite out of test cases related to shipping in the project. Change the Work Item Type to Microsoft.TestCaseCategory to search for test cases and click Run query. You now have a list of test cases that you can select to create suites from, if you choose. Press Esc to close the dialog. In this exercise, you will learn how to create a manual test plan and populate it with steps. The plan can later be run to confirm the expected behavior of your software. In this lab, we're going to focus on creating a new manual test case and running it. Install Google Chrome from . The rest of this exercise will use Chrome as its browser. If you're already using Chrome, just open a new instance for the next set of steps. Navigate to the Azure DevOps Marketplace at . Select the Azure DevOps tab. Search for "feedback" and click the Test & Feedback extension. Click on Install button on the details page. Click Install for the Chrome extension. In the Chrome Web Store, click Add to Chrome. Confirm the installation when asked. To open the extension, click the extension icon that will appear on the right of the address bar. Select the Connection Settings tab. Enter the URL of your Azure DevOps instance, such as " ", as the Server URL and click Next. The extension can be used in two modes: Connected and Standalone mode. If you have Azure DevOps or Team Foundation Server (TFS), select Connected mode. Standalone mode is for users who don't have Azure DevOps or TFS and want to use the extension to file bugs and share the report with their team. After connecting to Azure DevOps, you will need to select the team you want these efforts associated with. Select the Parts Unlimited Team under the Parts Unlimited project and click Save to continue. In Chrome, navigate to your Parts Unlimited project. As before, navigate to the Test Plans hub. Expand the dropdown next to the test plan and select New static suite. Name the new suite "End-to-end tests" and press Enter. From the Tests tab, select New | New test case to create a new test case. In the Title box, type "Confirm that order number appears after successful order" as the name of the new test case. At this point, we're ready to add steps to this manual test. Each step includes an Action, which describes the action the tester needs to perform. Optionally, a step can include an Expected Result, which describes the expected result of the given action. In the Steps panel, create a step for each of the following Actions, only one of which has an Expected Result. Action Expected Result Open project site Click Brakes Click Disk and Pad Combo Click Add to Cart Click Checkout Enter @Email, @Password Enter @Name, @Phone, @Email, @Address, @City, @State, @PostalCode, @Country, @Promo Click Submit Order Confirm order page has order # Order # should appear on order confirmation page Log out Close browser Note: If you end up with an extra empty step, delete it. At this point, the Steps panel should look similar to the following: Note the "Enter @Email, @Password" and "Enter @Name, @Phone, @Email, @Address, @City, @State, @PostalCode, @Country, @Promo" steps. In these steps, we used the @ sign to indicate that there were iteration-specific variables to be used during the manual test pass. We can define which variables to use by scrolling down to the Parameter Values section of this form and entering them for each iteration. Note that you may need to use the scroll bar on the far right side of the test case dialog to view this section. Use the following table to set up values for two iterations. Fields Iteration 1 Iteration 2 Email admin@test.com sachin@test.com Password P@ssw0rd P@ssw0rd Name Admin User Sachin Raj Phone 425-555-1234 555-555-5555 Address One Microsoft Way Two Tailspin Trail City Redmond Springfield State WA IL PostalCode 98052 11135 Country USA USA Promo FREE FREE The Parameter Values section should now look like this. Note that you can enter as many iterations as you need to fully test the breadth of the scenario. Click Save & Close to save the test case. In this task, you will learn how to run the manual test plan that we created earlier. Note that the process for triggering an automated test run follows a similar workflow. You can learn more about that in the documentation. Right-click the test case created earlier and select Run with options to begin a manual test run. There are a few options that you can use to customize each test run. The first option is to select a Runner, which will be the browser in this scenario. Next, you may have the option to specify which kinds of data to collect. Finally, you may optionally specify which build is being tested to make it easier to associate the results with the build they were from. Click OK to continue. If the Test Runner window does not appear, check to see if it was blocked by the pop-up blocker. If so, click the Pop-up blocker button, select Always allow pop-ups..., and then click Done. You can then launch the test run again with success. In the Test Runner window, expand the Test 1 of 1: Iteration 1 dropdown. Note that there are two iterations: one for each set of parameters specified in the test case. In the first iteration, the admin@test.com account is used. In the second, sachin@test.com will be used. The first step in the test is to open the project site. To do this, switch to the Visual Studio instance that has the Parts Unlimited solution loaded. From the IIS Express target dropdown, select Browse With.... Select Google Chrome and click Browse. If you're working on a large screen, it may be easier to resize the new window to fit next to the Test Runner window. Otherwise you can just switch back and forth. Once the site loads, return to the Test Runner and click the Pass test step button. As you complete the next steps of this test, be sure to check the Pass test step buttons for them as well. The next step is to click the Brakes menu item. Then click the Disk and Pad Combo product. The next step is to click Add to cart. On the next page, click Checkout. Log in using the credentials specified in the next step. Unfortunately, this will fail because there isn't an admin@test.com account. The Test Runner provides three valuable ways to record media from a test run. The first option is to take screenshots. The second is to capture each user action in an image action log. The final is to record the screen as a video. Click the Capture screenshot button to take a screenshot. Crop the screen down to show the login form and error message. Specify the name "No admin account" and click the Confirm button. Right-click the failed step and select Add comment. Enter a comment of "Admin account does not exist by default" and fail the test using the Fail test step button. Click Create bug to log a new bug. Enter the bug title of "Admin account does not exist by default" and click Save & Close to log the bug. Since the test cannot be completed due to a bug not directly related to the functionality being tested, expand the Mark test case result dropdown and select Block test. Click Save and close to save the test run. Close the test browser windows. In this task, you will learn how to review the results of a manual test run. Return to the browser window hosting the Test Hub. Select the Runs tab. Double-click the most recent test run to open it. You may need to refresh the data to see it. The Run summary tab provides an overview of the test run, as well as high-level details on the results of all tests included as part of the run. Select the Test results tab. This tab lists the results of each individual test case included in the run along with their results. Since there was only one test case included here, double-click it to open. You can review all details for this particular test case run from here. Scroll to the bottom to locate the iterations. Expand the first iteration. Review the results of each step in this iteration, as well as the failed login step, which shows the screenshot attached during the test run. In this task, you will learn how to create shared steps. A shared step combines multiple steps that are commonly performed in sequence into a single logical step, which can be shared across tests. If the process defined by the shared steps ever changes in the future, you can update the shared step in one place and it will be reflected in all tests that reference it. Click the test case link in the Summary section. Double-click the case to open it in the test case editor. Select steps 2-4 (use Shift+Click) and click the Create shared steps button. Set the name of these shared steps to "Add Disk and Pad Combo to cart" and click Create. Now you can see the previous steps replaced with the shared steps. Double-click the shared steps to open. If necessary, you can revisit these steps later on to update them for new requirements. Press Esc to close the Shared Steps dialog. Click Save & Close to save the test case. When a configuration is assigned to a Test Suite, all the TestCases inside the suite will have the same configuration. Let us see how to assign a configuration to a Test suite in the below steps Step 1: Go to TestPlans in Azure DevOps Step 2: Click on the TestPlan for which you have to assign a configuration Step 3: And then click on the 3 dots next to it as shown Step 4: Select the configuration which you have to assign in the next window and then click on the save button Step 5: Selected configuration will be assigned to all the testcases in the TestPlan You can also assign a configuration to individual TestSuite as well to the TestCases Let's suppose, you want to create a test plan document for your web application, mobile or any other software. You search "sample test plan document" on the internet and come across numerous test plan samples. By looking at the sample test plans, you understand that a software test plan document is a guide book for testing process. It is required for the successful execution of testing process for a project. It contains comprehensive information to carry out the testing activities. A software test plan document is divided into different sections such as introduction, objectives, scope, test items, features to be tested, and environmental needs. There are several test plan samples, each with different sections. Are you wondering what the 10 attributes every sample test plan document must have? No problem! We discuss them in detail here: Introduction A software test plan document begins with the introduction of the project and the product being tested. Include the following details in the introduction of your test plan: Project Background: Explain a brief overview of the project and its background. Purpose of Document: The purpose of test plan document is to provide details on how testing process will be conducted for a given project. Objectives and Tasks: This section contains your testing objectives and tasks. Scope: In this section of test plan document, the scope of testing is identified at high level. You might also need to explicitly mention some features which are out of scope. Test Items: List down the test items with release version and module details that are targeted for testing. References: In the references section of test plan document, list down the documents that are and can be referred to during execution of testing process. Features to be tested In this section of the test plan document, list down the features and functions in details that you have planned to test. These features should fall under the testing scope which has already been identified in the introduction section. For each feature to be tested, define the references of requirement with requirements ID so that the quality assurance team can refer to it. Describe any special consideration or details about the particular feature, if required. Features not to be tested There can be some features or functionalities which are neither clearly out of scope nor could be tested due to any reason. These features should be mentioned in your software test plan document in the 'Features not to be tested' section. Also, define the reason why a certain feature or functionality cannot be tested. Item Pass/Fail Criteria Define the success criteria of your tests in the test plan document. You can encounter three situations while executing the test cases – normal, suspension, resumption. Let us have a look at the item pass/fail criteria from a sample test plan document of web application: Suspension Criteria: Any situation which impedes the ability to continue testing or value in performing testing lead to suspend testing activities. Resumption Criteria: When the problem that caused the suspension had been resolved, testing activities can be resumed. Approval Criteria: An item will be considered as 'Pass' if it meets the 'Expected Outcome' defined in the corresponding test case. Approach Test approach is the backbone of the entire testing process. Hence, it is one of the important attribute that every test plan document has. In test approach, it is clearly stated what testing techniques will be applied during the testing process. Your testing approach may combine more than one testing technique such as exploratory testing, functional testing, regression testing, user interface testing, component testing, integration testing, penetration testing. Specify the tools and required human resource to perform the testing activity. The approach should be described in such a manner that major testing tasks could be identified. You need to see 'Features to be tested' section to adequately define the testing approach in your test plan. Test deliverables At the end of every testing activity, there is a deliverable. Include the list of test deliverables in your test plan document. Test deliverables might include test plan document, test cases, issues report, and performance report. Environmental Needs Software test plan document contains details of the specifications needed to set up test environment. There can be hardware and software needs for your product. Hardware Needs Hardware needs might include the device specifications such as desktop computer, laptop, tablet PC, smartphone. It can also include a specific screen size, memory requirement or processor speed. Moreover, there can be some requirement related to your internet connection – whether the device should have Wi-Fi or LAN connection, what should be the download and upload speed of your Internet. You might also need extra hardware for requirements where you might need to simulate the load of concurrent users. Software Needs Software needs include the operating system specifications such as Windows, Mac, Linux or Android. There can be further detail of version for the operating system. If you are testing a web application, you need to list down the browsers in your test plan on which you will perform the testing. You might need to use any tools or software to perform testing or to set up the test environment. List down all required software and make sure you procure the required software on time so you can proceed with the testing process as per schedule. Roles and Responsibilities A test plan document contains resource requirements. It also contains details of roles and the associated responsibilities of the individuals. If you have a big team, you can define roles and responsibilities in the form of a table. We are sharing 'Roles and Responsibilities' section from a sample test plan document: S. No Role Responsibilities Name 1 QA Manager Review test cases, review and approve the issues 2 Senior SQA Assigns tasks, tracks the testing progress 3 QA Prepare test cases, set up test environment 4 Tester Execute test cases, reports the issues Schedule A test plan document is a guide book to your testing process. Schedule is the essential attribute that defines the timelines for your testing activities. Make sure that you plan your schedule in accordance with the development schedule. Remember that you cannot test a feature or module, unless it is developed. This develops a high dependency of quality assurance team over development team. If development team lags behind the schedule, your testing schedule will be badly disturbed. It is recommended to isolate your testing activities and continue completion of your tasks, while the product is being developed. For example, you can gain business understanding and prepare test cases before any artefact is available for testing. We have shared a schedule included in the sample test plan of web application. You might add or remove columns in the schedule table as needed. S. No Task Dependency Deliverable Week 1 Business understanding 0-2 2 Prepare test cases Task 1 Test cases 3-4 3 Execute test cases Task 2 4-7 4 Report issues Task 3 Issues log 4-7 5 Test fixes 8-9 6 Report the findings Test report 10 Risks and Contingencies Risk is the uncertainty which is associated with a future event which may or may not occur and a corresponding potential for loss. Being the project manager, it is very important that you identify the risks in your test plan. Schedule Risk Meeting the planned deadlines plays a vital role in the successful completion of a project. Before you prepare your risk mitigation strategy, it is important to understand the reasons that increases the likelihood of risk occurrence. You might lag behind your schedule because of any of the following reasons: S. No Risk Mitigation Techniques 1 Inaccurate time and effort estimation Use PERT techniques Use expert judgment techniques to assure the accuracy of estimates 2 Inability to foresee the total scope Create work breakdown structure for your project Analyse the 'Features to be tested' thoroughly 3 Unexpected expansion of scope Include contingencies in your schedule 4 Inability to complete tasks at the estimated time Track the progress of individuals on daily basis Address any issues that are hindering the completion of tasks Train resources to upgrade their skill set Provide a realistic estimate while making schedule Budget Risks Budget is a crucial element in any project. It not only affects the success of your project, but it also affects your relationship with your client. It is very important to control and mitigate any risks associated with budget to prevent any unpleasant happening that might strain your reputation. The more accurate your budget is, the better your will be able to manage your project and stakeholders. We have listed a few budget risks below: S. No Risk Mitigation Techniques 1 Wrong estimation Prepare a rough order magnitude estimate initially Prepare detailed budget estimate when tasks and activities are clearly defined 2 Resource budget over run Track and control that resources do not take more than the planned time for completion of tasks 3 Cost overrun due to scope Control the scope of the project Define a process for approval of 'Change Requests' along with their costs 4 Indirect costs Include the estimates for overhead costs, general and administrative costs Operational Risks Operational risks are associated with the day to day activities of the project. Operational risks may eventually lead to improper process implementation or a failed system. Operational activities are performed repetitively; this means that operational risks can be mitigated by following company's standard procedures on regular basis. Quality control team plays a vital role in overall improvement of the software development process. While including operational risks in your test plan, consider the following risks: S. No Risk Mitigation Techniques 1 Failure to address priority conflicts Clearly prioritize the requirements with stakeholders Use adaptive planning approach to accommodate priorities 2 Insufficient resources Control and track whether the project activities are progressing as planned 3 Insufficient resources Estimate the required resources and procure them 4 No proper subject training Conduct staff trainings if needed Use the right people for the job Outsource resources 5 No resource planning Prepare human resource plan 6 No communication in team Conduct staff trainings if needed Use the right people for the job Outsource resources Technical Risks Technical risks can still exist even if you have planned everything flawlessly. There is an increased likelihood of technical risks when the technology is new. Other technical risks include the following: S. No Risk Mitigation Techniques 1 Changing requirements Use agile software development method 2 No advanced technology available or the existing technology is in initial stages Conduct trainings to build expertise Build a relationship of trust with client and prepare his mind for the technological risks Give time, effort and budget estimate keeping this point in mind 3 Complex product Use experienced people for the job with the required skill set Break the problem into smaller parts 4 Difficult integration of project modules Perform impact analysis Exhaustively perform regression testing Test Plan Example Here is a sample test plan that gives a list of items you should include in your test plan. Introduction Objectives & Tasks Scope Test Strategy Alpha Testing (Unit Testing) System & Integration Testing Performance & Stress Testing User Acceptance Testing Batch Testing Automated Regression Testing Beta Testing Hardware Requirements Environment Requirements Test Schedule Control Procedures Features to Be Tested Features Not to Be Tested Roles & Responsibilities Schedules Dependencies Risks/Assumptions Tools Approvals Recap Let's have a quick review of what we have understood till now. We know a test plan document is vital for the successful execution, tracking and controlling of testing activities in a project. It contains all necessary information to guide the testing process. A software test plan document is divided into various sections. We had a detailed look on the top 10 attributes every sample test plan document must have. First part is the introduction which provides a brief overview of the project background, scope, testing objectives and references. Then, we define a list of features that should be tested and the features that should not be tested, along with success criteria. This enables us to scheme a detailed testing approach for the identified features to be tested. The output of testing activities is the test deliverables. We also need to include any specific environmental needs to set up the test environment. Moving on to the resource and task planning, we define the roles along with schedule of tasks. Lastly, we include the important part of risks and contingencies. Next time you make a test plan document, do not forget to include the top 10 attributes in your test plan document. If you know of other important attributes that should be included in a test plan, share them in the comments below.