TechExcel

Product Guide
DevTest

Improve test productivity and product quality THROUGH TEST MANAGEMENT

Product testing is more complicated, labor-intensive, and time-consuming than ever before. Businesses are demanding greater openness, transparency, and scalability from their software investments—they looking for versatile solutions that enable them connect users and resources worldwide while leveraging their existing investments.

As a result, contemporary business applications run on many different platforms and in many different environments, support integration with emerging technologies and legacy systems, and feature add-on modules that support every imaginable business process. All this versatility has multiplied the number of variables that may affect the performance application making the task of testing software more complicated and time-consuming than over before.

But the same forces that drive the demand for these applications also require that these products be delivered to market as quickly as possible—if not sooner. QA organizations are under increasing pressure to complete their testing in ever-shorter time frames. And, as if this were not enough, these challenges are compounded by hectic development schedules that introduce problems all their own—new features may suddenly appear or disappear, the design and functionality of product features often change to meet the demands of the marketplace or to accommodate tight schedules.

To meet these challenges, QA organizations are pursuing several different strategies —beginning their test planning earlier in the development life cycle, leveraging the results of previous test efforts, improving the management of testing data, and reusing existing test coverage. And increasingly, many testing organizations are abandoning outmoded paper-based systems and turning to software test management solutions to organize, plan, and analyze their testing efforts.

Why Test Management?

In the past, testing organizations could rely on paper-based solutions to plan, design, manage, and track their testing projects. Older technologies did not require the same degree of planning and organization as do contemporary software suites—they lacked the portability, the versatility, and extensibility.

But even with simple applications, paper-based systems are merely adequate. The lack of organization, poor planning, and inefficient communication that are inherit in paper-based systems regularly jeopardize delivery schedules and, ultimately, jeopardize the quality of the product itself.

In a paper-based system all testing activities are recorded and tracked in static lists, tables, outlines, and matrices created in word processing documents or spreadsheets. Paper-based solutions are difficult to maintain, update, and track. As a result, testing strategies are necessarily straight forward and limited, because testing teams are straitjacketed from the very start of the process.

Poor or inaccessible documentation and inadequate channels for communication make it difficult for QA organizations to adjust to sudden changes in product design. Test plans are frequently based on out-dated requirements documents, and this can lead to test cases that do not adequately test product features and functionality.

Understandably, test planning is frequently left to late in the development life cycle when the product approaches some kind of stasis. But delaying test planning creates a whole new set of hurdles. Time pressures mean that QA organizations must focus on testing the application at hand and have little time to plan ahead for future release cycles. Hastily created test cases are generally not reusable.

Test planning efficiency can be improved when the test planners can base future test assignments on previous results and an analysis or test or defect data. But tight schedules mean their is little time for reflection or future planning. And even if there were time, traditional models make it difficult to review previous test data or defect data because it is often inaccessible—lost or misplaced in a stack of paper, buried in someone's inbox, or stored away somewhere in different files or applications.

Why test management? Because test management solutions enable businesses to work more intelligently and efficiently. Test management solutions provide following benefits:

- Manage knowledge and facilitate communication: A test management solution provides testers with a centralized and secure tool through which they may review control documents and research previously reported bugs. Knowledge collected by the testing group is accessible to all project members at all times.
- Enforce the standardization of test cases: A test management solution enables the testing group to define the data they wish to track and to design a standard interface for collecting and tracking that data. Standards increase efficiency.
- Promote the creation of reusable test cases: A test management solution makes it easy for testing groups to save, manage, and reuse their most effective test cases.
- Leverage previous results in test planning: A test management solution enables testing groups to plan future test assignments based on the analysis previous test results. Instant access to summary reports enables test planner to improve test efficiency.
- Respond to issues quicker: A test management solution provides real-time access to all testing data so that testing groups can immediately respond to critical test blockers or and other issues. The faster the team is able to address the critical issues the sooner the test team may resume their testing.
- Make information accessible: A test management solution provides a test planners with a complete picture of their testing project so that they better manage their project and they can make the necessary adjustments to meet testing objectives and deadlines.

Knowledge Management

One key factor to meeting the demands of contemporary software development is to ensure that all project stakeholders—management, developers, and testers— have access to the most up-todate control documents and may effectively communicate with others both within and outside their organizations and teams—in short, everyone must be on the same page at all times.

To address this issue, TechExcel DevTest takes a "knowledge-centric" approach to test management that places the collection, management, and distribution of information at the core of all development and quality assurance processes.

Knowledge management enables all stakeholders to access, manage, and share information so that QA may make informed decisions throughout the testing process, leverage the information collected and generated by development, and learn from their past successes and failures so that they may implement more efficient and intelligent processes.

Key components of the DevTest approach to knowledge management include a centralized knowledge base, instant access to quality reports, and tools for communicating information relevant to individual test cases and the project as a whole.

Managing Project Knowledge

In DevTest, all testers may access a centralized repository of information from within the DevTest client. The DevTest knowledge view, powered by TechExcel KnowledgeWise, provides development and QA organizations with a tool for collecting and organizing the ideas that drive product development and testing. All ideas, both formal and informal are collected and managed in the same, centralized knowledge base.

TechExcel KnowledgeWise is a key component in the TechExcel DevSuite of application life cycle management products. KnowledgeWise provides project managers, designers, developers, testing groups, and sales and marketing departments with a secure repository for all project documents— the project plan, business requirements, functional and technical specifications, risk management documents, and corporate standards and guidelines may be managed in one knowledge base that is shared by the entire business.

- A centralized knowledge base increases efficiency, prevents the loss of information, helps to reduce system and maintenance costs, and facilitates collaboration by distributed teams.
- Access to knowledge items is protected by privilege-based access controls. The ability of project members to view, edit, lock, check out, and check in protected files is defined by their role in the project.
- Built-in version control tools ensure that all project development and testing teams (no matter where they are in world) always have access to the most up-to-date documents.

Leveraging Control Documents

A common knowledge base ensures that the QA organization always has access to the latest project control documents —business requirements, functional and technical specifications— and enables them to leverage this information to plan and organize their test plans early in the development processes.

In DevTest, QA organizations may access project control documents as soon as those documents are uploaded to the knowledge base—at the same time they are available to the developers themselves. Access enables the QA organization to jump-start test planning and develop testing strategies and test cases in parallel with the implementation of the designs.

Using these control documents, testing groups may estimate the scope of the project, allocate resources, and plan appropriate strategies for testing the functionality and performance of those features.

Testing groups need not wait for all of the control documents to be approved before they begin their test planning. They can create test cases and build test planning structures in a piecemeal fashion as the designed product is realized in the knowledge base. With each new control document that is added to the knowledge base they can create appropriate test cases.

Ideally, project control documents are complete and approved at the beginning of the test planning stage. However, the specifications and designs that are approved at the beginning of the development process may be sketchy, inadequate, or change significantly over time due to changes in the marketplace, technical problems, or time constraints. DevTest provides testing groups with the flexibility they need to adjust to changes in design or schedule.

QA organizations may take an evolutionary approach to test planning. By organizing their test cases by product features, they can readily adjust to changes in product design or changes to the schedule. They may create appropriate structures and test cases as the requirements and specifications are completed and update the list to accommodate changes to the application. DevTest planning structures and test cases may be easily edited and updated so that QA organizations need not pay the penalty for changes in design.

Facilitating Task-level Communication

By placing knowledge management at the core of all development processes, DevTest facilitates all project-level and task-level communication.
Just as the centralized knowledge base ensures that all stakeholders have access to project information, the complete history of every test case and all background information is always right at hand. All communication is recorded and tracked with the test case so that test task itself is the vehicle for communication. Key information cannot be lost in e-mail exchanges or buried in user inboxes.

Good notes from the prior testing enables test team members to pick up testing where others have left off. Any testing team member may pick up the work of another tester, and with a little research understand the test in its entirety.

Every test case may be directly linked to supporting materials—test plans, business requirements, schedules, and so on— that enable testers to run successful tests. Testers may read all business requirements and specifications and compare product functionality against those control documents.

Test Planning and Organization

Test planning begins with the identification and definition of product features in a feature list: a simple list of the visible functions, subfunctions, commands, menu choices, reports, and so on that need to be tested. Whereas a feature list begins as simple list of all of the product features that need to be tested, it is ultimately an outline of the testing project in which product features categorized and prioritized.

In DevTest, the feature list is articulated as a hierarchical tree structure that both organizes test cases and represents the product to be tested. DevTest transforms the information that previously captured in static lists into a dynamic structure called a test library that helps testing groups to manage their testing project, improve team efficiency, and find bugs.

Test Case Organization

The TechExcel DevSuite of applications conceptualizes the product under development as a fully designed product that is articulated, managed, and communicated in DevTest project structures prior to its actual implementation—product features define the structure that is used to organize and manage the implementation and testing of those features.

The DevTest test library is sometimes called the "functional tree" because it organizes test cases by product functions. The tree structure enables the testing group to visualize the entire application and understand the relationship between every area under development. Every product feature may be represented by a unique branch and contain multiple subfolders representing feature subfunctions. Every dialog box, command, report, error message, menu option, and so on may be represented by a subfolder in the test library.

The test library defines the scope of the project, shows the relationship between product features, and manages the test templates that will be used to test those features.

- The test template tree manages the test library: The template tree structure enables testing groups to manage test templates in a hierarchical tree structure. Test templates may be organized by product, functional area, test type, or any other categories that is useful to their business.

- The template tree defines project scope: The test template tree may represent the product being tested organized into functional areas and enables testing groups to better provide full test coverage of all product features. Organizing the test library by product, feature, and function shows every area under development. Project managers may use the test template tree to estimate the resources needed for the testing project.

- The test template tree helps test designers to create better tests: Representing the product enables testing groups to visualize "the designed product" described in the control documents, analyze its design, and identify good tests for its feature. Understanding how the program features work together enables testers to develop test cases that are more likely to find bugs and combine or eliminate redundant tests.

- The test template tree makes test templates accessible: A library of tests is only useful if the test cases are accessible. The template structure enables project teams to define hierarchical structures for organizing templates and making them accessible to users.

- The test template tree shows all areas of development. Testing groups may ensure that every feature is properly tested and prioritize test coverage by module, component, or feature. The test template tree shows which functional areas have been tested and which areas have not so that testing groups can make sure that they don't do duplicate work.

Test Case Definition and Standardization

DevTest enables organizations to define custom test cases and enforce standardization through test templates. A test template is a blueprint for creating test cases that may be used to test product features and performance across multiple builds, platforms, and environments. Test templates are easily edited, copied, and duplicated. Changes in the design or functionality of a product feature can be easily accomodated.

In DevTest, each test case is displayed in the client as a form through which testers may track and record test results. Test cases typically consist of a test procedure, the expected outcome of that procedure, the prerequisite state or steps for the test, and other information that the QA team may wish to track. DevTest features a fully customizable graphical user interface (GUI) that enables QA organizations to identify the data they wish to track and to standardize the look-and-feel of test cases. QA organizations may add any number of custom fields to record and track testing data.

Test templates enable testing organizations to control the distribution of test cases and to implement a standardized look-and-feel for all test cases. Testing organizations may define their own approval process for all test templates that ensure that only those test template that have been approved may be used in test cycles.

Test case standardization ensures that test results are consistent from one group or test cycle to the next and that the data collected from different tests may be compiled and compared. The format of test procedures, expected results, and data-entry controls is consistent across all test cases enabling testers of work more intuitively and efficiently.

Test templates enable testing groups to preserve and recycle their most creative and successful tests—it makes no sense for these tests to be recreated from scratch with each new test cycle, or change in platform.

Test Templates and Test Tasks

As we have seen, the test template tree structure—the test library—both organizes test cases and articulates the product under development—the designed product—and all its features. Once the QA organization has created a library of test cases for a specific product, both the tests and the structure used to manage those tests may be used and reused with each new release cycle.

In DevTest, all test cases are represented as test templates and test tasks. The test library is a tool for organizing and managing reusable test templates that may be used to create test cases that are applicable to many different platforms and environments.

Using test templates and test tasks enables testing teams to define and manage standardized and reusable test cases (test templates) and to track the performance of program features against that test (test tasks) in many different environments.

- A test template is a blueprint for creating test cases that may be used to test product features and performance across multiple builds, platforms, and environments.

- Test tasks, on the other hand, are the instances of a test template that are actually used to test a feature in a test cycle. Every test task inherits its properties from the test template that was used to create it.

What makes this all possible are environmental variables. Test templates do not merely define test procedures and expected results, they also define the environments in which an application may be tested. An environmental variable represents the various environmental factors that may affect the performance of an application during a test cycle. Environmental factors include things like system platforms, hardware, network infrastructure, browsers types, and other factors.

A single test template that includes one environmental variable may be used to generate multiple test tasks – one for each environmental variable value – or, if it includes many environmental variables – one test task for each permutation of environmental variable values. For example, the web browser environmental variable may represent three environmental variable values: Internet Explorer, Firefox, and Safari. A test template that includes the web browser environmental variable may be used to generate three test tasks: a Internet Explorer test task, a Firefox test task, and a Safari test task.

Scheduling and Assigning Testing

DevTest simplifies the management, assignment, and scheduling of test cycles by organizing testing in distinct release cycles and test cycles. This two-tiered approach simplifies test management by leveraging the power of test templates and environmental variables to create test cycles for different environments and to provide instant access to summary statistics.

In DevTest, all test planning – the definition of the scope and objectives—is managed in release cycles. A release cycle defines the scope and objectives of a group of test cycles—the test schedules, test templates, environments, and testing teams that are applicable to the test cycles within that release cycle.

DevTest test planning is managed in the planning tree, a hierarchical tree structure that represents products, releases, and test cycles as a series of folders and subfolders. Just as the test template tree organizes the test library (test templates) by functional areas of development, the planning tree organizes the test tasks based on those templates into products, releases, and test cycles.

The DevTest two-tiered approach to test planning provides the following benefits:

- View the test plan: The planning tree structure defines and shows the relationships between products, releases, and test cycles.

- View test depth: The planning tree structure shows which functional areas have been tested and which areas have not so that testing groups may see which features have been tested and which areas have not been tested so that they can avoid unnecessary duplication of effort.

- Reports by release cycle: Testing groups may define and run reports that enable them to view the effectiveness of tests for every test cycle in a release cycle.
- Reports by test cycle: Testing groups may define and run reports that enable them to view the effectiveness of individual test cycles so that they may make adjustments to subsequent test cycles within a release cycle.

The scope and depth of each test cycle is determined by the applicable test templates, environments, and project members that are defined at the release cycle level. Every test cycle is the child of a release cycle and test cycle options are limited to those options defined as applicable to that release.

Planning Test Cycles

Test cycle planning is the act of choosing appropriate test cases based on the results of previous test cycles within a release cycle. Test cycle planning is an iterative process that relies heavily on the results of previous test cycles within the current release cycle. After the completion of each test cycle, test planners may view summary reports and make adjustments based on the results of those tests. Subsequent test cycles may target features or environments that are buggy or stress tests that successfully discover bugs. Selecting appropriate test cases can be based on many different factors including the stability of the program, the results of previous test cycles, the availability of testing groups, or the testing objectives defined in the parent release cycle.

- Smoke tests: The first test cycle in a release cycle is frequently an automated smoke test of basic product functionality. If the product cannot pass a smoke test there is no need to assign more complex test tasks to the testing group until the basic bugs are fixed.
- Assign tasks for multiple environments: In each test cycle testing groups may determine which environments are tested in that cycle of testing. The applicable test template is used to generate test tasks specifically targeted at a selected group of environments and are assigned to a specific tester or testing group.
- Testing and retesting environments: Testing teams may generate and regenerate test tasks for specific environments in multiple test cycles. In each test cycle the test tasks may be assigned to the same applicable owners or to any other applicable owner or applicable testing group.
- Advance coverage selection by query: At the beginning of each test cycle the testing group may run a query to see which tests passed and which tests failed in previous test cycles. They may generate new test tasks based on the same test template to retest the feature.
- Meeting testing objectives: Testing groups may define targets for each release cycle of testing including targets for the number of tasks performed, the number of product defects found, the effective time, the ineffective time, and the total time spend testing.

Running Tests and Tracking Results

Test tasks give testers the information that they need to set up and execute a test (the environment, test procedure, and expected result), they enable the testing team to track and analyze the results of the test, and they are foundation for any bug reports based on that test.

DevTest reports enable testing teams to measure the performance and efficiency of their testing teams, identify their successes and failures, and to adjust test plans, test cases, schedules, or test assignment accordingly.
DevTest summary reports show the number of test tasks assigned, the number run, the percentage of the test tasks that passed or failed, the time required to execute the tests compared to the time estimated in the test plan. Summary reports may be grouped by release or test cycle or by tester within each test cycle.

Submitting Bug Reports

DevTest-DevTrack integration enables organizations define processes for testing, fixing, and retesting product defects, streamlines the submission of bug reports, and facilitates the sharing of information between the development and testing groups. Bugs discovered in DevTest may be submitted to DevTrack development projects for resolution, and the fixes may be retested in subsequent DevTest regression testing.

The TechExcel DevTest-DevTrack solution enables testers to submit bug reports to an integrated DevTrack project from within the DevTest client. Testers do not need to switch back and forth from DevTest and DevTrack to test and report product defects and so can report bugs immediately while they are still fresh in their minds. The quicker the tester creates the bug report the less likely the tester is to forget key details that will enable developers to reproduce the bug.

Organizations may streamline the bug reporting process even further by mapping DevTest test task fields to DevTrack development issue fields. Key information such as the test procedure, the version of the software, and the environment tested may be automatically copied from the test task to the newly created development issue. Test case-bug report field mapping reduces the amount of time that testers need to spend typing, and enables them to get back to their primary business of testing.

Linking Test Cases to Bug Reports

During the course of product testing, QA engineers may encounter bugs that are responsible for malfunctions in many different product features. Without the means to effectively communicate with one another, QA engineers may submit multiple identical bug reports to developers. The submission and re-submission of what are essentially identical bug reports only creates more work for the developers.

In an integrated DevTest-DevTrack system, every DevTrack development issue may be linked to one or more test tasks using defect links. Defect links enable QA engineers to associate development issues with the test tasks that exposed them and enables testers and developers to share information and work more effectively.

Moreover, every test task that is linked to a DevTrack development issue provides testers with the opportunity to draw from and add to the collected knowledge of the entire testing team. Each tester may add key details to the DevTrack issue enabling developers to understand the scope of a bug and how that bug affects other product features.

Researching Existing Bug Reports

DevTest encourages QA engineers to identify and research existing development issues before they submit new bug reports to the DevTrack project. Researching existing product defects provides the following benefits:

- Enables testers to familiarize themselves with development issues and draw on the experience of other QA engineers.

- Enables testers to fully understand product features and expected behavior. Access to DevTrack development issues enables QA engineers to read all linked control documents— business requirements, functional specifications, and technical specifications.

- Reduces the number of duplicate bugs reported to the development team and enables them to work more effectively. The submission and re-submission of what are essentially identical bug reports only creates more work for the developers.

In fact, DevTest enables administrators may define workflow rules that require testers to search the knowledge base for existing issues before they can submit a new bug report.

Sharing Experiences and Information

The DevTest knowledge-centric approach facilitates communication between testing teams and developers so that all project members may work together more efficiently and effectively.

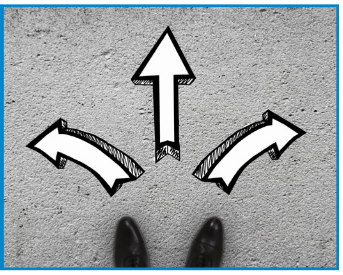
As noted previously, submitting a bug report from within a DevTest project automatically creates a link between the DevTrack development issue and the originating test task. All test tasks notes are automatically copied over to the development issue providing the developers with key information that may enable them to identify the source of the bug.

DevTest provides testing groups within many other tools for communicating key information to developers.

- DevTest HTML memo field controls enable testers to format the text of bug reports so that they can describe the steps required to reproduce the bug more effectively. Text formatting tools enable testers to create bulleted and numbered lists, tables, headers, and other formats that highlight key information.

- DevTest enables testers to attach files to bug reports such as screen captures of error messages and typos, keystroke captures, macros that generate the test case, printouts, memory dumps, and documents that define steps required to reproduce the bug in greater detail than that found in the test case.
- The mapping of DevTest test task fields to DevTrack development issue fields ensures that all key information is copied into the bug report. Developers need not question in which version or environment the bug was discovered.

Conclusion



TechExcel DevTest is a test management solution that enables test organizations to manage every stage of testing life cycle—from test case design, to test execution, to test analysis. DevTest provides testing groups with the tools they need work more effectively and efficiently, hold down costs, and to deliver higher quality products.