Creating Relevant, Achievable Test Strategies

This is an excerpt from my book, Expert Test Manager, *written with James Rommens and Leo van der Aalst. I hope it helps you think more clearly about the test strategies you use.* – Rex Black, RBCS, Inc.

A test policy contains the mission and objectives of testing along with metrics and goals associated with the effectiveness, efficiency, and satisfaction with which we achieve those objectives. In short, the policy defines *why* we test. While it might also include some high-level description of the fundamental test process, in general the test policy does not talk about *how* we test.

The document that describes *how* we test is the test strategy. In the test strategy, the test group explains how the test policy will be implemented. This document should be a general description that spans multiple projects. While the test strategy can describe how testing is done for all projects, organizations might choose to have separate documents for various types of projects. For example, an organization might have a sequential lifecycle test strategy, an Agile test strategy, and a maintenance test strategy.

The Contents of the Test Strategy

Because the test strategy describes how testing proceeds, independent of any particular project, the document can include any information that is not project specific. (While we use the word *document*, the test strategy could be a set of pages on the company intranet or an internal wiki.) For example, the test strategy may include material on the following topics:

- **Test specification techniques.** This material should discuss the process by which the test basis is analyzed and documented and then decomposed into specific tests. In other words, how does the test team gather information about the project and product, generate a set of test conditions to be covered, generate sufficient tests from those conditions, maintain traceability, and assure the quality of those work products (e.g., with reviews)? It can also include links to various templates used for documenting tests, guidelines on the level of detail required in tests, rules for documenting references to the test basis and test oracle, and procedures for updating tests when the test basis or test oracle change or a defect is detected in a test.

- **Scope and independence of testing.** This material can describe how to coordinate across the different levels of testing, which are typically carried out by different participants who have different degrees of independence. For example, the test strategy may specify that unit testing is performed by developers, integration testing by developers and technical test analysts, system testing by business-oriented test analysts and technical test analysts, and user acceptance testing by senior users of the application being developed. It may further specify how these different groups coordinate their efforts to eliminate any gaps or overlap that might occur.

- **Test environments.** This material should describe the hardware, software, infrastructure, facilities, networking capabilities, and other equipment needed for testing. To the extent that the test environments differ from the production or customer environments, the test strategy should describe the impact of those differences on the testing results. If, for unavoidable reasons, test environments must be shared across test levels or production hardware used for testing, the test strategy should describe how to coordinate that sharing and how to manage the attendant risks.

- **Test data.** This material should include information on how test data is to be obtained, created, maintained, and managed during testing. If production or actual customer data is to be used, the security of sensitive information should be considered and procedures around both access and use of such data should be put in place.

- **Test automation.** This material should describe the use of automation in the various test levels. In organizations where extensive test automation is used—for example, to automate functional regression tests or unit tests—this section could consist of links to other documents that describe the tools to be used, their acceptable usage, documentation requirements, and so on.

- **Confirmation testing and regression testing.** This material should discuss how to carry out a proper confirmation test, which typically involves at least repeating the steps to reproduce a failure from a defect report and may also involve rerunning the test(s) that previously failed due to the defect. It should also cover how to carry out regression testing for single changes or defect fixes as well as for larger groups of changes; for example, it might discuss how frequently to run a functional regression test or how to perform impact analysis to determine which manual regression tests to run.

- **Non-functional testing.** This material should describe how the test organization addresses activities such as security testing, reliability testing, and load and performance testing. This section could include links to other documents that

describe tools to be used, corporate or external standards to be followed, and regulatory information, if necessary.

- **Reusability and configuration management.** This material should address which software work products and test work products (i.e., all the testware) can be reused and how these work products should be specified, named, stored, managed, and maintained. It should include a discussion on version and change control for test work products as well as information (or links to information) about how the configuration management tools are used.

- **Test control.** This material should cover the ways in which the test manager monitors progress against the test plan throughout the test process and exercises control actions when needed to redirect efforts when necessary. This includes regular meetings with test team members (e.g., daily test team debriefs during test execution) and other project participants (e.g., weekly project status meetings). This also includes executing risk mitigation and, when necessary, implementing contingency actions.

- **Test measurements, metrics, and reporting.** This material should include detailed information about the various data gathered by the test team during the test process; how that data is used to generate project, product, and process metrics; and how those metrics are used to report status for the project, including product quality and test progress. (Note that the process metrics that measure testing capability were discussed in the test policy.)

- **Defect management.** This material should give test team members and other project participants detailed insight into the essential topic of defect management. It should describe the defect lifecycle, including the various states in the lifecycle, the roles of the various project participants as each defect report goes through its lifecycle, and the expected response time. It should describe the various data and classifications gathered throughout the defect lifecycle, including enough information for project participants to properly and completely document each defect.

- **Tools.** This material should include basic information on the tools used by the testing organization. In addition to general descriptions, links to tool documentation and information on where and how to access the tools should be available in this section as well. This list may include the test case management tool, defect management tool, automation and performance testing tools, database querying tools, proprietary tools, and the test organization's wiki, just to name a few.

- **Standards.** If applicable, this material should explain and reference any mandatory and optional standards that testers should or must adhere to during the test process.
- **Integration procedures.** To the extent that component integration testing and/or system integration testing are applicable, this material should describe how integration proceeds and the way in which appropriate integration tests are created and executed.

You probably noticed that many of the preceding elements refer to coordination across different groups and test levels. Ideally, a single test strategy spans the different test activities across each project to maximize the effectiveness and efficiency of these activities. This requires the author of the test strategy, often a test director or senior test manager in the independent test organization, to work closely with managers in the other groups that participate in testing. This is similar to the interaction required in the definition of the test policy but in this case involves a smaller cadre of people and a focus on the work to be done.

It is a best practice for test organizations to create training material for new testers that covers the test policy and, especially, the test strategy. Simply having these documents available to testers or even assigning new testers the task of reading these documents will generally not ensure adequate compliance. There is no point in spending hours and hours creating test policy and strategy documents if those documents do not guide behavior during testing.

While the test strategy should guide behavior on each project, there is usually a need to provide more specific information for each project. This is either to instantiate a general procedure given in the test strategy or to handle unique properties of the project. These details about the implementation of a test strategy on a particular project are referred to as the test approach and should be captured in the test plan for the project. For example, if the test strategy calls for risk-based testing, the test approach might specify the particular people involved in the risk analysis on a project.

In addition to the content of the test strategy, it's important for the test manager or test director writing a test strategy to understand the common types of test strategies; their benefits, success factors, and risks; and how these types of test strategies relate to key stakeholders. In the following subsections, we'll review these types of test strategies and their associated details. Remember that you can and typically should mix these types of strategies to maximize the benefits while minimizing the risks.

Analytical Strategies

Any time a test team analyzes the project, the product, and the documents describing the project and the product as a way of determining the test conditions to be tested, they are following an analytical test strategy. The test conditions are then elaborated

into tests during test design, using black box, white box, and experience-based test design techniques. During test implementation, the test data and test environment necessary to run the tests are defined and created.

In all analytical test strategies, the analysis takes place before test execution. Often, this analysis can be done early enough so that the design and implementation of the tests can occur before test execution begins. However, because this is not always possible or practical, a test charter may be defined that blends reactive and analytical test strategies, stipulating the analysis to occur prior to test execution while specifying the design and implementation of tests to occur during test execution. Alternatively, logical tests can be written early, with the concrete tests defined during test execution, or concrete tests can be written during test implementation, with the testers given the latitude to respond to observed behavior in a reactive fashion during the execution of these concrete tests.

Two of the most common analytical test strategies are requirements-based testing and risk-based testing. In requirements-based testing, one or more test conditions are identified for each requirement element. In risk-based testing, each quality risk item is a test condition. In either case, one or more tests are elaborated for each test condition. In the case of requirements-based testing, the number of tests is sometimes determined by factors such as the specified priority of the requirement, by the details of the specification itself, and by the test design technique(s) used. In the case of risk-based testing, the number of tests is determined by the level of risk associated with the quality risk item.

An analytical test strategy has a number of benefits:

- **Alignment of testing against a clearly defined test basis.** In analytical strategies, the test basis is well documented. In properly performed analytical test strategies, traceability of the tests back to the test conditions, and from the test conditions to the test basis, is defined and maintained during test design, implementation, and execution.

- **Measurability of testing against the test basis.** During test execution, the traceability of the tests and their results back to the test conditions allows the test team to deliver to the project participants and stakeholders a clear and data-based presentation of what has been tested and what has not been tested, what works and what does not work.

- **Defect prevention through early analysis of the test basis.** The analysis of requirements, risks, and other project documents often results in the discovery of defects and project risks. These defects and risks can then be eliminated or managed early in the project, improving the efficiency of the achievement of quality goals and the overall execution of the project.

Ideally, project and product stakeholders participate in the analysis sessions and define the test conditions to be tested. This has the added benefit of providing these stakeholders with transparency and insight into what is to be tested (including to what depth), what is not to be tested, and the way the testing process will proceed and will help in their understanding of the test results.

To succeed with an analytical strategy, the following factors are required:

- For document-focused analytical strategies such as requirements-based, design-based, or contractual-based testing, the test team must receive the underlying documents prior to test analysis. These documents need not be final because they can undergo revisions or iterations (e.g., based on defects or risks discovered during analysis), but they should be sufficient for the discovery of the test conditions.

- For stakeholder-focused analytical strategies such as risk-based testing, the test team must be able to get the technical and business stakeholders to participate in the analysis of the project and product to identify the test conditions and the level of risk associated with each condition. The test conditions need not be finalized during this stakeholder-driven analysis.

In either case, while the test conditions will change during test design, implementation, and execution, a significant amount of change in these conditions can reduce the efficiency of testing. Sequential lifecycle models attempt to limit the amount of change over the long time period from analysis to execution. Iterative and especially Agile lifecycle models limit the number of test conditions that will change by limiting the number of test conditions defined for each iteration.

For analytical test strategies to succeed, the following risks must be managed:

- For document-focused analytical strategies, any unmanaged change to the documents used as a test basis will result in incongruity between the tests and the test basis. In addition, this strategy will fail when these test basis or test oracle documents are not delivered or are extremely vague or ambiguous. In this case, the test conditions or test cases will be incomplete or incorrect.

- For stakeholder-focused strategies, if the test team cannot engage the relevant stakeholders in the process of identifying the test conditions (e.g., the quality risk items), the test conditions or test cases will be incomplete or incorrect. Even if the list of risk items is complete, if the assessment process is incorrect due to insufficient stakeholder involvement or a lack of stakeholder consensus, the depth of testing will be incorrect. Either outcome will reduce the effectiveness and efficiency of the strategy.

Analytical test strategies are well described in a number of books on testing, which makes these strategies one of the best-defined and most common types of test strategies.[1]

Model-Based Strategies

When a test team generates a model of system behavior and then uses that model to design and implement tests, they are following a model-based testing strategy. The model can be mathematical, statistical, graphical, or tabular. The model can include the environment, the inputs processed by the system, the conditions in which the system operates, and the expected behavior of the system under those collective circumstances. The model is often based on some analysis of actual situations or of predicted situations. It can address functional behavior, as when the analysis is based on UML models, or non-functional behaviors.

For example, in performance and reliability testing, technical test analysts will examine the proposed or actual production environment and the test environment. Predicted or actual load is analyzed, including the variations of those load conditions over time and background loads constantly or periodically present on the system. This analysis results in a characterization of the system's operating conditions sometimes referred to as an operational profile. For these various operating conditions, maximum allowable response time, maximum acceptable resource utilization, and/or minimum allowable throughput are defined.

During test design and implementation, this model is typically automated using a load testing tool. These automated reliability and/or performance tests are then run during system test and/or system integration test. During this period, however, it is very difficult to resolve the types of fundamental design defects that underlie performance and reliability failure. Therefore, it is a best practice for this model to be developed during system design and implemented using a spreadsheet or an automated performance simulation tool. This allows static performance testing of the system design before any effort or money is expended on implementing a (possibly flawed) approach to building the system.[2]

---

1 For example, for risk-based testing, see Rex Black's book *Managing the Testing Process*.

2 Rex has personal, first-hand knowledge of two multiyear, multimillion-dollar, person-century projects that failed during system integration testing due to the discovery of irresolvable performance-related design defects. You can read about the right way to manage performance risks in Rex Black and Barton Layne's article, "The Right Stuff," found at www.rbcs-us.com/images/documents/Risk-Mitigation-and-Performance-Testing.pdf.

A model-based test strategy has the major benefit of realistic testing of the system. The generation of the model focuses on the real-world environment, conditions, inputs, and other usage characteristics of the system. For non-functional tests such as performance and reliability testing, this realism of environment, load conditions, and usage is often critical to meaningful results. If the production environment and test environment are not the same (e.g., memory, disk space, load balancing), the model must be adjusted, though the adjustment entails significant risk that the test results will not reflect production results.

As you might expect given the foregoing, one critical success factor for model-based testing is an accurate model of real-world usage. Another factor that applies for many non-functional forms of model-based testing is the availability of a tool to support the testing. Most often, we think of load testing tools that automate test execution for performance and reliability testing, and such tools certainly are irreplaceable for those test types. However, performance simulation tools as mentioned earlier can play a critical role too.

For model-based test strategies to succeed, the following risks must be managed:

- The technical test analysts may find, during analysis, that they do not have all the data they need to create an accurate model. This can lead to a scramble to find answers to hard questions such as, "How many concurrent users can we expect during peak system load?" In some cases, these questions remain unanswered even during test execution, resulting in significant erosion in the confidence building aspect of testing.

- Even if the data is available, the model constructed by the analysts can be inaccurate. Even skilled analysts can make such mistakes, and the likelihood of this risk occurring becomes almost certain if inexpert testers are used. For example, the tests for performance and reliability testing are some of the most difficult to create, and unless someone with five or more years' experience is involved in the process, mistakes will probably occur.3 If the model is wrong, the tests will be wrong and severe defects can escape into production.

- Even given an accurate model, model-based testing can still go awry if the wrong tools are selected. A tool can be wrong not only because it is incapable of implementing the model, but also because the testers themselves are unable to use the tool to implement the tests, to execute the tests, or to interpret the results reported by the tool. The most frequent outcome in these situations is a level of false confidence when the tests don't reveal failures or the failures that are revealed are mistaken for correct behavior.

---

3 The five-year figure is quoted in Malcolm Gladwell's book, *Outliers*, and he cites numerous examples where this rule applies.

- Another mistake that can endanger model-based testing is a natural tendency to test what the model tells you should happen rather than testing for what should not happen. This preponderance of positive path tests also tends to result in false confidence.

As with analytical risk-based testing, technical test analysts performing model-based testing rely on key test stakeholders, both business and technical. These stakeholders can help provide and explain data for the model, construct the model, and validate the model.[4]

Methodical Strategies

In analytical strategies, analysis produces the test conditions; in model-based strategies, the model determines the test conditions; in methodical strategies, the test team follows a standard set of test conditions. This set of test conditions can come from a variety of sources. Some test teams use a quality standard such as ISO 9126. Other test teams assemble a checklist of test conditions over time, based on their experience. The checklist can arise from the business domain of the system, the characteristics of the system itself, and the type of testing being performed. For example, security testing often relies on checklists of common vulnerabilities.

Sometimes test managers become confused about the difference between analytical test strategies and methodical test strategies. For example, the result of a risk analysis is a set of risk items and their associated risk levels, which serve as the test conditions, and this can look a lot like a checklist. The difference is that, with analytical strategies, analysis occurs at the beginning of each iteration (for iterative and Agile lifecycles) or at the beginning of each project (for sequential lifecycles), generating a new set of test conditions. In methodical strategies, the set of test conditions does not vary from iteration or project, though periodic updates to the test conditions can occur.

A methodical test strategy has the benefit of consistent testing. Once a particular set of attributes is put on the checklist, we can be sure that those will be tested. For example, if usability characteristics such as learnability, understandability, operability, and attractiveness are placed on the checklist, we can be confident that these behaviors have been evaluated during testing. The checklist is typically defined at a logical level, allowing all stakeholders to evaluate it for completeness.

To succeed with a methodical strategy, the following factors are required:

---

4 For more information on model-based testing, see *Practical Model-Based Testing: A Tools Approach*.

- The set of test conditions, checklists, or test cases used by the testers must reflect what is required for the system to satisfy customers and users. It must be accurate, complete, and current. Therefore, some periodic maintenance, perhaps as part of the test closure process, must occur.
- Since the maintenance of the test conditions will be retrospective and thus slow moving (as compared to an analytical strategy), methodical test strategies work best when the test object is stable, varying only slowly over time. This is often the case with successful and mature applications in the enterprise, mass-market, and IT context.

The main risk of methodical test strategies, obviously enough, is the possibility that the test conditions, test cases, or system attributes are incomplete, incorrect, or obsolete. Such defects in the checklist will result in significant gaps in test coverage, and therefore testers will likely miss important defects.

In situations where methodical strategies work well, such as very stable, slowly evolving systems, these strategies can be very successful. The use of a standard set of tests makes practical economical approaches such as test technicians or outsourcing to a stable test team. The stakeholders, after helping to define the set of test conditions, are required only to review the periodic updates, making this a lightweight strategy in terms of stakeholder workload.

Process- or Standard-Compliant Strategies

In methodical strategies, the test team follows a standard set of test conditions; in process- and standard-compliant strategies, the test team follows a set of processes that are defined by people outside the test team. The process typically does not predetermine the conditions, but it does determine the extent and type of test documentation gathered, the way in which the test basis and test oracles are identified and used, and how the test team is organized relative to the other parts of the project team.

For example, in teams following Agile methodologies, there is an emphasis on lightweight documentation, including in testing. Logical test cases rather than concrete test cases are more typical. The tests are derived from user stories, and these user stories and acceptance criteria are the test oracles. Individual testers are embedded in development teams, though the best practice is to matrix these testers into a central test team, with test managers and even a test director, to preserve independence of testing.

The primary benefit of a process- or standard-compliant test strategy is that the test team can take advantage of the skills and experience of the people behind the standard or process. For example, very experienced people were involved in defining the ISTQB

syllabi, the IEEE 829 test standard, and the Agile methodologies. If the problems being solved by these processes or standards are similar to the problems facing the test team, the processes or standards provide out-of-the-box solutions, allowing the test team to be more efficient (by not having to generate its own solutions) and more effective (by not having to arrive at good solutions through a painful period of trial and error).

Of course, that requirement for similarity of problems is a critical success factor for these strategies: The processes or standards were created in response to a particular set of problems that the creators regularly faced in their consulting work, dealt with in their employment as practitioners, or studied in their capacity as academics. If those problems are also your problems, then these strategies can work, if you select the right processes or standards. Agile methodologies are very different than the IEEE 829 standard, and you have to select the right one, given your specific context.

For process- and standard-compliant test strategies to succeed, the following risks must be managed:

- If the test team or broader project team does not understand the process or standard they are implementing, they will make mistakes. This risk can be mitigated through good training and proper use of consultants during the implementation process, but this mitigation is imperfect in that many trainers and consultants also have an imperfect understanding of the process or standard they are advocating, promoting, and promulgating and, furthermore, these trainers and consultants often have a vested interest in minimizing the weaknesses and fallibility of the process or standard.

- Even with a good theoretical understanding, a team can make mistakes in implementation of the process or standard. This is especially the case because there is no truly out-of-the-box way to implement a process or standard in an organization; some tailoring is always required. In this tailoring, the team (or the trainers or consultants) can make the mistake of being too compliant with the approach, not modifying those segments that will impede success, or too loose with the approach, modifying or omitting parts of the process or standard that enable its success. We've observed both mistakes with clients, though the most frequent is the dogmatic application of a process in spite of clear warning signs that it won't work. We have seen a number of good testing ideas executed by the three-word firing squad: "That's not Agile."

- Even the perfect understanding and implementation of a process or standard will fail when it is implemented in the wrong situation. For example, it is a best practice to take aspirin at the first signs of a heart attack, but for people who are allergic to aspirin or who are at an extreme risk of stomach bleeding, this can kill rather than cure. If the problems facing your organization are dramatically

different than those facing the creators of the process or standard, you will be effectively trying to win a chess game by applying a backgammon strategy.

The degree of stakeholder involvement varies considerably in these strategies. Agile methodologies require daily interaction between the business and technical stakeholders and the test team, though this practice is frequently minimized. IEEE 829 standards generally recommend reviews of test work products by stakeholders but not daily meetings.[5]

Reactive Strategies

A test team follows a reactive test strategy when the focus is on reacting to the system or software that is delivered for testing. Reactive test strategies are characterized by the following:

- While test analysis to identify the test conditions (often captured in test charters) may occur prior to test execution, the test design and implementation occur primarily in parallel with test execution.
- The test plan and approach are dynamic, evolving rapidly as the needs of the project change and as the true state of the product being tested become clear.
- The test cases are designed and implemented using primarily experience-based and defect-based techniques.
- Aids such as heuristics, software attacks, and defect taxonomies are used during test design and implementation.[6]
- Processes and documentation are lightweight and informal.

Reactive strategies can be used in any lifecycle, including sequential and iterative lifecycles.

A reactive test strategy has a number of benefits:

- Finding a complementary set of defects. Reactive strategies tend to find some defects that other strategies would miss (and vice versa).
- Locating defects efficiently. The cost to find a defect is lower because of the lightweight documentation and rapid evolution of the tests.

---

5 For more information on process-compliant strategies, see Lisa Crispin's book *Agile Testing*, and Rodger Drabick's book, *Best Practices for the Formal Software Testing Process*.

6 See, for example, James Whittaker's *How to Break Software*.

- Refocusing continuously. Information gained from testing, especially about emerging defect clusters, allows the test team to redirect their efforts on the most productive test areas.

- Responding robustly to challenging situations. Unlike some strategies, such as analytical requirements-based testing, reactive test strategies can handle incomplete or even entirely missing test basis documents.

That said, not all teams that try reactive strategies succeed with them. Success requires highly skilled and experienced testers. These testers must understand the business problem that the application solves and the technologies involved in the solution, and they must be proficient in a wide variety of testing skills and techniques.

For reactive test strategies to succeed, the following risks must be managed:

- If the testers executing have insufficient skills, the tests will be superficial at best and often just plain wrong. Testing will reveal less important defects at the expense of more important ones, and the rate of false positives will be high. Reactive testers, to succeed, must have skills and experience with testing techniques, the business domain of the software under test, and the technology used to implement the software.

- By default, reactive testing does not produce documentation of what was tested or how the expected results of the tests were determined. This means that test coverage is unknown and arguments about test results ensue. Test charters, with traceability back to the test basis and a definition of what test oracles should be used, must be provided to the testers.

To enjoy the benefits of reactive strategies while reducing the risks, the best practice is to use a reactive strategy in combination with other, more formal and systematic strategies. This allows the test team to achieve a high level of measurable coverage while at the same time taking advantage of the skills and experience of the test team to find defects that the other strategies often miss.

Consultative Strategies

A *consultative test strategy* is one where the test team asks a few of the key testing stakeholders to select the test conditions that the test team should cover. The test team will often define the specific tests to cover those conditions—which is what makes the strategy consultative—but test coverage is determined by the stakeholders. This differs from an analytical risk-based strategy in that the test team does not build a consensus about the test conditions but simply responds to the conditions provided by the stakeholders.

Consultative test strategies are also called *directed test strategies*, a name that is particularly appropriate when clients direct testing service providers to perform particular tests. If the stakeholders provide the tests themselves, that is also a consultative strategy, though the name *directed strategy* is probably more appropriate.

When properly executed by the test team, a consultative test strategy has the benefit of providing the consulted stakeholders with the exact test coverage they requested. Proper execution means that the test team designs, implements, and executes the right tests, based on the specified conditions. This assumption actually applies regardless of the test strategy because the failure to test the test conditions identified by the test strategy is a well-known breakdown in testing.

The benefit of consultative testing is realized only when the stakeholders ask for the right areas of test coverage. The accuracy of the stakeholders' requested test conditions is a critical success factor. If the stakeholders do not know what conditions should be tested, the depth of testing required for each condition, and the order in which to cover each condition, the test team may do exactly what it was asked but not what is needed.

For consultative test strategies to succeed, the following risks must be managed:

- When two or more stakeholders are consulted, they may have conflicting opinions about what should be tested, in what order, and to what depth. This places the test team in a bind that it will find difficult to resolve, especially since consultative strategies often occur in situations in which the test team is disempowered relative to the stakeholders.
- Even if the stakeholders agree about the test conditions, they might hold conflicting expectations about what testing can accomplish. While not a problem unique to consultative strategies, it is particularly acute in such situations.
- As mentioned earlier, if the stakeholders provide an incorrect definition of the test conditions to cover, the test effort will be misdirected. Many implementations of consultative strategies do not include a way of self-checking after the analysis phase (where the test conditions are provided to the test team).
- If some of the key stakeholders are not consulted, the neglected stakeholders might disparage and even disrupt the test effort. Trying to enlist the support of these disgruntled stakeholders after the fact is generally harder than including them beforehand.

To mitigate some of these risks, some test teams blend a consultative strategy with other test strategies. This blending will tend to reduce coverage gaps. However, in the case of testing services providers, it is often difficult to use other strategies.

Regression-Averse and Test Automation Strategies

The last of our list of common test strategies are the regression-averse and test automation strategies. These involve various techniques to mitigate a particular risk to the quality of the system—namely, the possibility that existing features of the system will regress or cease to work as well as they currently do. The most effective and efficient technique for managing regression risk is the extensive automation of existing tests. This automation can and ideally should occur at multiple test levels. Testing through application programming interfaces (APIs) at the unit test and component integration test level often prove easier to create and maintain compared with the system test and system integration test levels. However, many defects cannot be detected at these levels, so some degree of regression risk mitigation at the system and system integration levels is necessary.

Sequential models, such as the V-model, tend to have relatively stable code bases once system testing begins. The regression risks are relatively low and are focused on those features already delivered to customers. However, regression risk management is critical for iterative lifecycles. In such lifecycles, the existing code base, with a stable, tested set of features, is repeatedly modified and new code is added. In Agile lifecycles, there is a need for regression testing that is not only thorough but also very quick. The test team should work closely with developers to ensure the design of testable software and to develop a maintainable, multilevel approach to the automation architecture and the automated test scripts.

If properly carried out, a regression-averse test strategy has the benefit of reducing the risk of regression in key feature areas without excessively impeding the release of new versions of the system or product. Because complete regression risk mitigation is not possible in a finite period of time, proper execution of this test strategy implies reaching an agreement on how much regression risk mitigation is required to make people comfortable.

A critical success factor for regression-averse strategies is successful test automation. Manual regression testing of all but the simplest systems is too time consuming, tedious, and error prone to be effective and efficient. Successful test automation implementation involves the ability to create, execute, and maintain a large set of automated tests.

For regression-averse test strategies, especially automation-based strategies, to succeed, the following risks must be managed:

- Depending on the interface being tested, the test team might find that suitable automated test execution tools are insufficient or even nonexistent. This is especially true at the system and system integration test levels, when testing using a graphical user interface other than Windows or via a browser.

- In some cases, testers find that tools are available but that automation of tests is not an efficient solution. Wide-ranging system test and system integration test automation through a graphical user interface can prove difficult, particularly if the features and interface are changing rapidly (requiring significant maintenance and revision of existing scripts), if tests require human intervention to keep them running, or if there is no way to automate the comparison of actual and expected results.

- Test automation, the best technique for mitigating regression risk, is a very complex endeavor, in spite of the efforts of the last 20 years to make it simpler. Anecdotal evidence indicates that about half of all test automation efforts fail, often because these efforts are undertaken by people with insufficient skills. We have seen situations in which people with no experience in testing, not to mention test automation, were placed in senior positions in test automation efforts, resulting in a sad but predictable failure. As with automated performance and reliability testing, mentioned earlier, five or more years of experience with test automation is required to achieve expertise, and at least one person with such expertise should be included in any automated testing effort.

- While regression is a significant risk for software, there are other quality risks to consider. One problem that can occur is that the test team becomes overly fixated on regression testing and automating the regression tests. This results in stable existing features but poorly tested new features. This can be very dangerous for test teams in organizations that rely on aggressive release schedules of highly touted new features because testing will be misaligned with the overall corporate strategy for software.

- If the scope, technology, or features for the product are constantly changing, it is difficult to predict what tests will be needed in the long term. Extensive investments in automated testing for features that are dropped from the project or retired from the application reduce the efficiency of testing, especially when other areas that remain unchanged are not tested. The lack of a clear road map for the product's features or technology is a key reason for test automation failure.

In spite of the risks and challenges, most test teams should consider at least some type of regression-averse test strategy as part of their overall test strategy blend. Test stakeholders will tend to demand that currently working features continue to work in subsequent releases; regression failures are generally seen as unacceptable.[7]

---

[7] Dorothy Graham and Mark Fewster's book *Software Test Automation* is a valuable resource for those trying to design, implement, execute, and maintain automated tests.

Developing a Workable Test Strategy

In the previous sections, we have surveyed a variety of test strategies. All of these strategies have succeeded in certain test situations, and all have also failed when misapplied. One of the most common test strategy failures occurs when a single strategy is selected, especially under the misapprehension that it and the strategies not selected represent mutually exclusive "schools" of thought.

Even worse occurs when a single strategy is selected and that strategy is inapt for the testing needs of the test stakeholders and test team. Even the perfect execution of the wrong strategy will result in the failure of the test team. That error is sometimes fatally compounded when the strategy continues to be applied in the face of evidence of its inadequacy. An expert test manager must be able to recognize when a test strategy has failed. Further, they must be able to identify reasons a strategy might have failed.

To avoid these failures, the expert test manager must be able to analyze the actual needs of the test team and test stakeholders. They must discard the dogma that says that one must choose between competing schools of testing methodologies and instead remain open to the use of all applicable test strategies, based on a cold-eyed evaluation of the strategies, their benefits, and which ones are most likely to work. The best test strategy is often a blend of two, three, or more of the canonical test strategies discussed earlier. In addition, other, less frequently observed test strategies exist, and the test manager should consider those as well.

Part of selecting the right test strategy involves considering the skills and resources available to the test team specifically and the organization in general. For example, selecting an analytical requirements-based testing strategy in an organization that does not write requirements specifications is a recipe for failure. The level of formality of the test strategy can vary as well, and implementing an overly formal test strategy in an organization that is highly informal is another common cause of test strategy failure. Successful test strategies are aligned not only with the needs of the organization but also with the practical realities.

Having selected test strategies and blended them into a coherent test approach, the test manager must be able to explain the benefits of this strategy to key testing stakeholders. While Bismarck's famous saying that those who enjoy sausage and respect the law should never watch either being made is a clever one, it does not apply to testing. Test stakeholders need not understand all the details of what we do, but they should understand how we select what to test, the general set of activities involved in testing, and most important, how they will benefit.