

# Structured Testing: Do It Right or Pay the Price

## A White Paper

by **Richard Luetngen**  
ThoughtWing

---

This paper was developed to provide general background to assist clients in decisions related to organizing and performing testing activities of reengineered business processes and of transaction processing software, especially enterprise-wide suites of packages, or ERPs (Enterprise Resource Planning), as well as SCMs (Supply Chain Management) and CRMs (Customer Relationship Management).

Please note that this paper presents professional opinions intended to apply generally and that clients must take appropriate care to evaluate them in light of their specific needs. ThoughtWing makes no representations, warranties or guarantees of any sort as to the applicability of the opinions presented in this paper to the specific needs of any client. Please also note that the techniques and materials presented in this paper are representative of a testing methodology developed and used by ThoughtWing, and that other techniques and methodologies are commercially available.

The paper is organized as follows:

1. Overview
2. Test Planning and Organization
3. Test Execution.

## 1. Overview

Companies employ various means to test reengineered business processes and supporting software, from very little, which risks a great deal, to quite structured tests that deliver high levels of confidence that processes and software operate as intended and are secure. This paper presents a framework for testing as well as techniques that support different levels of testing confidence, in order to provide clients with options that meet their needs.

Testing optimally occurs at various levels and is conducted by several groups of people with different objectives. The different types of testing include the following:

- *Installation Testing.* Such testing occurs on installation of packaged software. The software vendor, who seeks to determine that the installed, unmodified and unconfigured software operates as intended, usually performs it.
- *Unit and String Testing.* Technicians such as programmers perform these activities to determine that modifications made to existing processes and software, or entirely new processes and software, satisfy the specifications that drove development. Each affected process task or software module is subjected to relatively simple input/output tests (unit tests), while related process tasks and software modules (strings) are subjected to tests that determine that the interactions among the related tasks or modules operate as intended.
- *Configuration Testing.* This major category of testing seeks to determine during the prototyping phase of package implementation whether reengineered business processes and configured (or modified) packaged software operate as intended. It is an iterative process, performed each time reconfiguration (or new modification) to a process or software module takes place, until satisfactory results are obtained.
- *System and Acceptance Testing.* This major category of testing consists of a number of different tests, and seeks to determine that a set of processes and supporting software is ready for production use. Its different components include the following:
  - *Functional* - Determines that all process/system (*combined*) objectives have been met and that the combined system correctly performs all functions as required.
  - *Communications* - Determines that interfaces between components operate properly. This includes communication links between remote devices as well as software links between different implementation units contained in the same computer. Man/machine interfaces are tested as well.
  - *Performance* - Measures response time and throughput rates under a variety of conditions.
  - *Volume* - Subjects the combined system to heavy volumes of data, similar to the loads expected to be encountered in the course of normal operations.

- *Stress* - Examines the behavior of the combined system at, and beyond, the limits of its resources by subjecting it to excessively high loads. Stress testing should attempt to break the system to determine the extreme points at which it can operate beyond the stated requirements.
- *Recovery* - Demonstrates that the combined system can recover from hardware and software failures without extensive loss of data.
- *Usability* –Completeness and accuracy of data values transmitted across man/machine boundaries are examined as part of functional and interface testing, while response times are the subject of performance testing. However, the ability of users to correctly enter data and understand and make use of the automated system's responses must also be tested. The users' comfort level must be evaluated in two ways: (1) to ensure that the new procedures match the way they are accustomed to working and (2) to determine if they are physically comfortable in their environment.
- *Operations* – Specifically tests computer operations procedures, including start-up and shutdown procedures and the ability of the operators to perform them.
- *Environment* - Determines how the combined system interacts with other systems in the same environment (e.g., several systems share the same DBMS, or multiple downstream combined systems are dependent on input created by an upstream combined system).
- *Security* – Determines that security mechanisms cannot be breached either maliciously or accidentally.

\* \* \* \* \*

This paper deals only with *Configuration Testing* and *System and Acceptance Testing*, the most complex of the various types of testing.

## 2. Test Planning and Organization

The primary deliverable of the test planning process is the *Test Plan*, which consists of the following:

- An organizational structure and set of rules to administer testing
- The *Test Specification*, which sets out the processes and automated system components to be tested, and how they will be tested, as well as test schedules and budgets
- A physical environment to support testing.

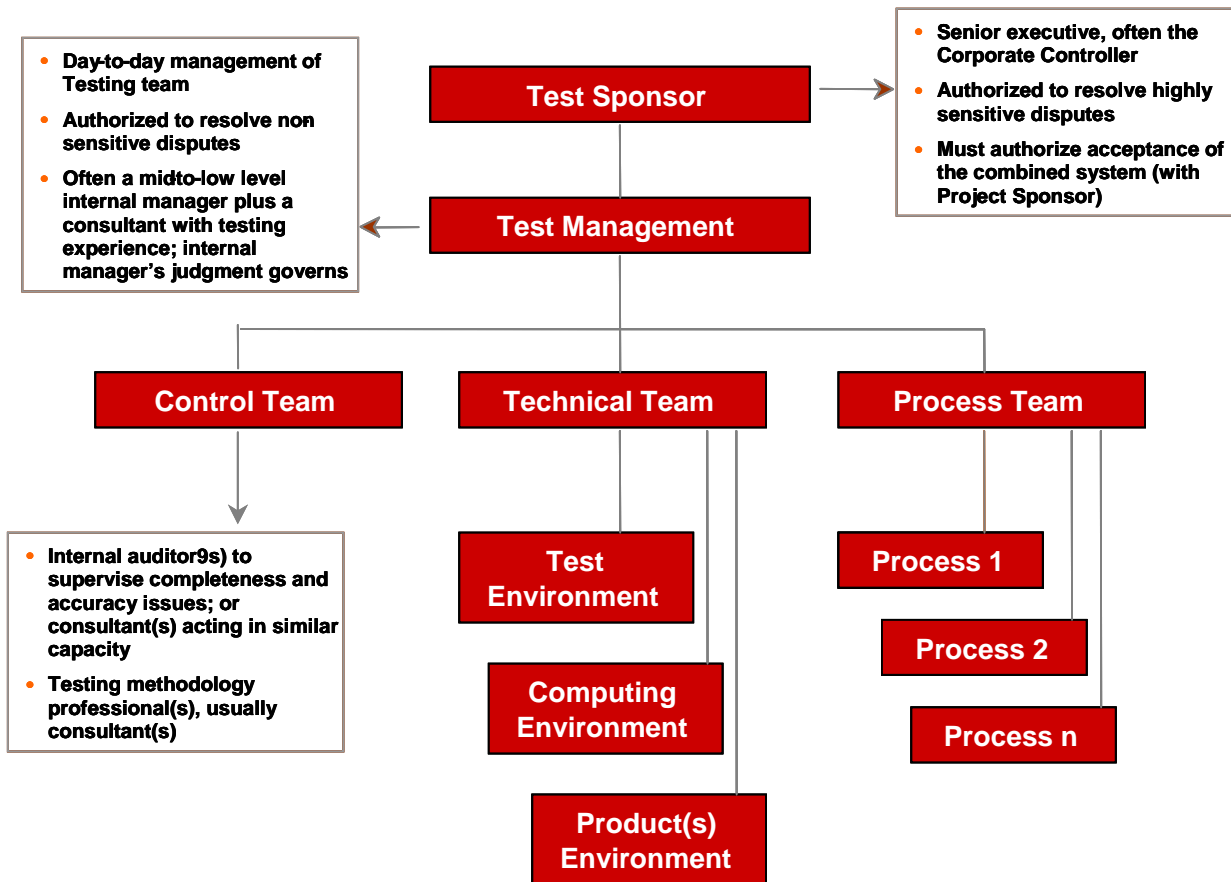
**Organization and Administration**

An organizational structure supports effective command and control over a complex set of processes. If the environment undergoing testing is not complex, such as in a small package implementation with little or no business process reengineering, then the organizational structure supporting testing need not be complex, either.

However, in a large organization undergoing a complex implementation with extensive business process reengineering, particularly one where multiple divisions will be supported, opportunities will abound for misunderstanding, mistakes and political maneuvering. In such cases, undue and costly delay can result unless managed effectively by a structured team with clear lines of authority and accountability, guided by coherent rules.

The testing organization depicted below is a workable alternative for large, complex tests.

**Exhibit 3: Sample Testing Organization**



The *Test Sponsor* approves testing policy and rules, which are usually proposed by testing methodology professionals in the *Control Team*, who work with *Technical Team* and *Process Team* members to plan the test.

*Test Management*, often an internal manager, sometimes paired with a consulting manager, administers the test according to approved policy and rules, and works with testing professionals in the *Control Team* to plan the test.

The *Control Team* consists of professionals who perform audit functions, assuring completeness and accuracy of the test by observing its conduct and by reviewing results. It also provides specialized test expertise through one or more testing specialists, usually consultants.

The *Technical Team* consists of technicians who establish and maintain the computing and test environments, as well as the software products themselves. *Test Environment* and *Computing Environment* team members are usually internal Information Technology/MIS personnel, while *Product Environment* team members are usually primarily consultants, with some internal personnel undergoing training.

*Process Team* members are internal operational personnel in charge of actually developing most test materials and of performing the tests and validating results.

Policies and rules can vary enormously depending on the priorities of management, but one simple scheme for dealing with test results is as follows:

- *Result 1* --- Test part has delivered expected results and is successful. Move on to next part of the test.
- *Result 2* --- Test has *not* delivered expected results and action proceeds according to the circumstances, determined as follows:
  - *Severity 1 Exception* --- a testing exception has occurred (actual results do not match expected results), and on examination it is determined that expected results are correct and that the test cannot proceed without first successfully passing the part that encountered this Severity 1 Exception.
 

*Action: Test halts until problem is analyzed and corrected, then the condition is re-tested until expected results are obtained.*
  - *Severity 2 Exception* --- a testing exception has occurred, and on examination it is determined that expected results are correct, but the exception does not require that the test halt before the underlying problem is resolved.
 

*Action: The underlying problem must be resolved and the condition causing the exception successfully re-tested before **all** parts of the test are completed.*
  - *Severity 3 Exception* --- a testing exception has occurred, and on examination it is determined that expected results are correct. However, on examination it is determined that the exception is insufficiently serious to warrant interrupting or causing a delay in concluding the overall test.
 

*Action: Proceed with the test until completed, but schedule analysis at a later time to resolve the exception.*

- *Severity 4 Exception* --- a testing exception has occurred, and on examination it is determined that expected results are *not* correct.

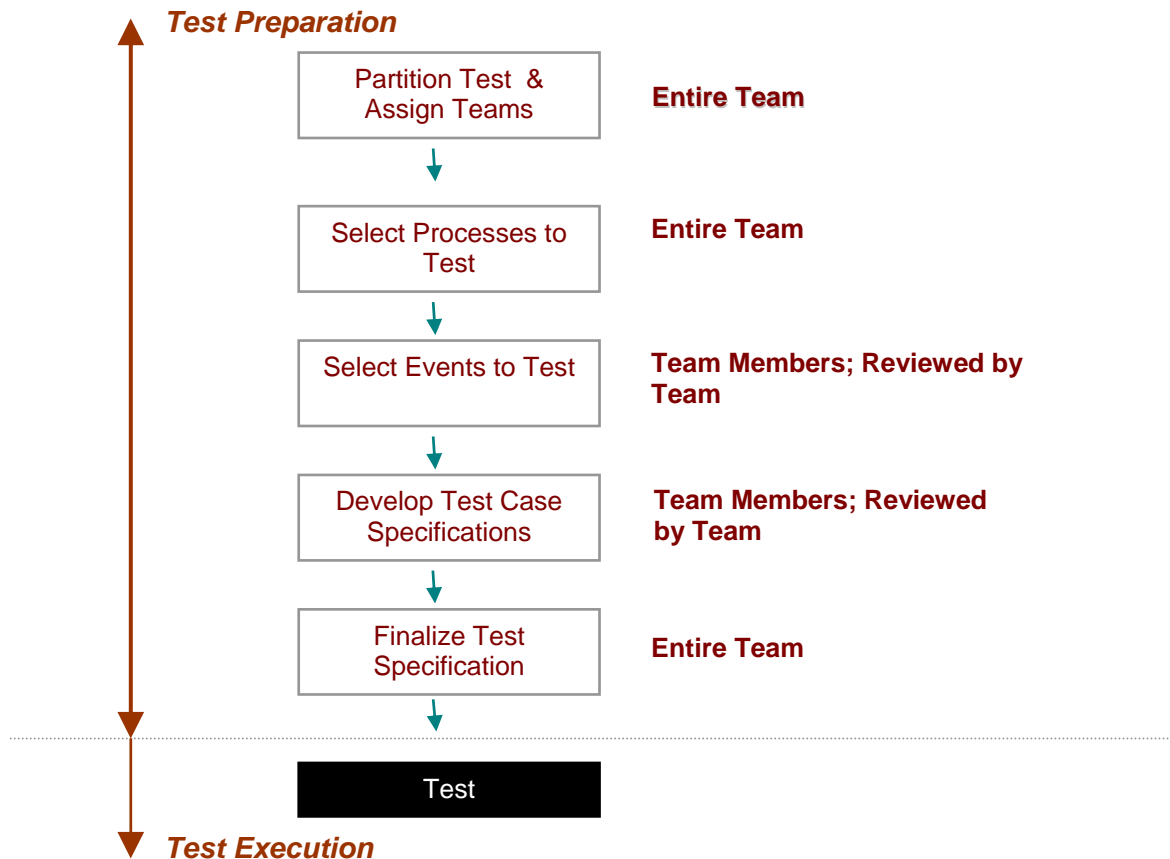
*Action: if the actual results are determined to be correct, note the fact and proceed with the remainder of the test. If both expected results and actual results are determined to be incorrect, correct the expected results and resolve the underlying problem with the actual results if appropriate (by severity of exception), then re-test.*

### The Test Specification

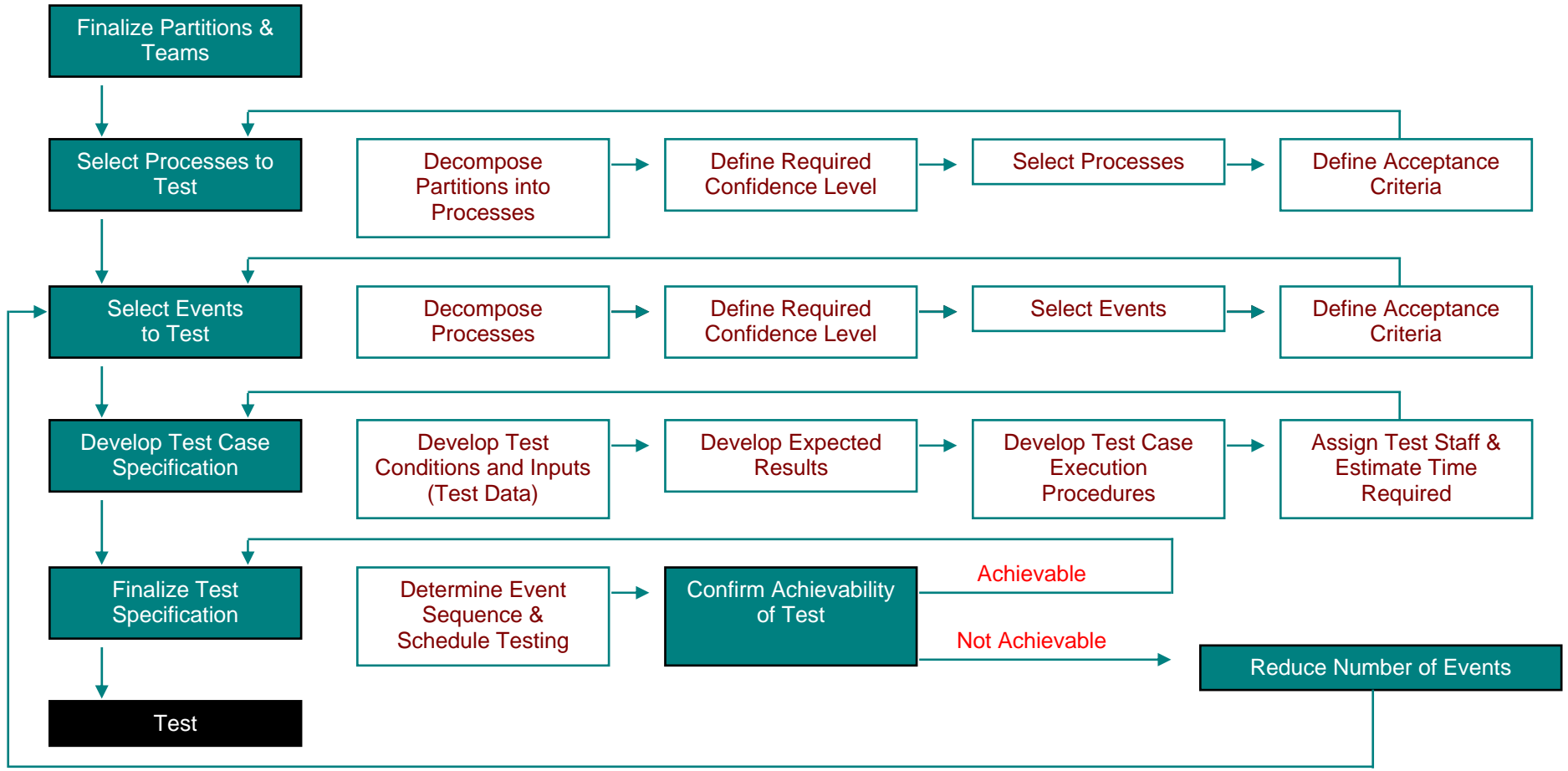
The *Test Specification* is the set of materials that primarily defines the test, and most of the actual work involved with test planning is consumed in producing this major deliverable.

The graphic below depicts the sequence of major activities required to develop *the Test Specification*. Beside each major activity appears the worker type associated with performing the activity. On the following page is a graphical depiction (Exhibit 3) of the tasks that comprise each major activity. Following is a description of each task.

### Exhibit 2: Major Activities In Test Specification



**Exhibit 4.8-2: Activities Associated with Performing Test Specification Development**



### Partition Test & Assign Teams

This activity seeks to decompose the test at the highest level into *partitions*, which are sets of related processes, in order to allow adequate scope of test control. An example of a *partition* might be the order-to-cash cycle.

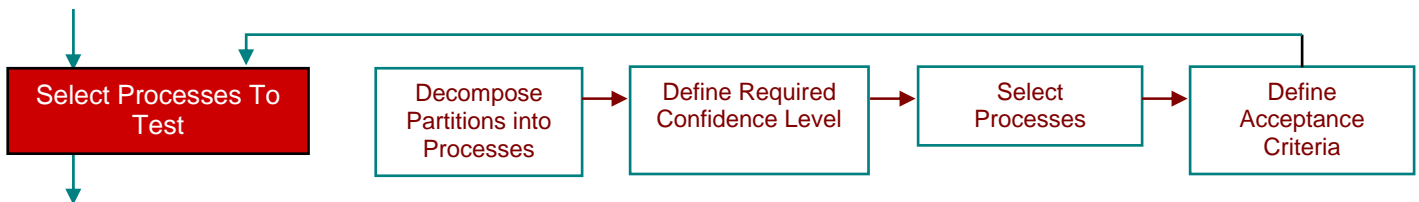
Each partition is assigned a team whose members understand the processes which form the partition, and a team captain who leads the team and who is authorized to set priorities, assign work to team members and resolve most disputes.

The key questions that need to be answered in finalizing test partitions are:

1. Does each partition represent a related process stream that is sufficiently end-to-end to minimize the need for participation by team members on multiple teams?
2. Taken together, do the partitions fully satisfy the scope of the business operations and support capabilities to be tested?

### Select Processes to Test

This major activity results in selection of the business processes within each partition that will be tested. The activity consists of the following more detailed activities:



The first step is decomposition of each partition into its constituent processes. The step identifies significant-risk processes by test partition, and associates with each those automated system components that support the business process.

Next, for each process within each partition, a *confidence level* needs to be established. This represents the level of relative importance of the process to the test objectives.

A company should determine the desired level of testing confidence early, since the higher the desired confidence level, the higher the likely testing related cost. Often, management will direct that in all cases the highest level of confidence will be planned, yet the cost of satisfying such direction is always very high and often unnecessary. Moreover, testing confidence can vary within a test, imposing high confidence for some processes and lower confidence for others.

One set of guidelines for determining optimal testing confidence is as follows:

- Required test confidence should be **HIGH** when all events and all associated conditions related to the process must be tested because:
  - the business impact of failure of the process is very high, and the failure of any event or condition could cause the process to fail;

- or, because the risk of failure of *all* events and conditions is considered high-to-moderate.
- Confidence is **MODERATE** when least important events and associated conditions related to the key activity are not tested because:
  - while the business impact of failure may be high or moderate, not all events or conditions associated with the process may cause the process to fail;
  - or, because the risk of failure of some events or conditions may be considered low.
- Test Confidence is **LOW** when only the most important events and associated conditions related to the key activity are tested, because the business impact of failure is low.

Once confidence level is established, processes are actually selected for the test. All High confidence processes must be selected. However, if the test must be conducted according to a very aggressive schedule, it might be advisable to eliminate from this selection all Low confidence processes.

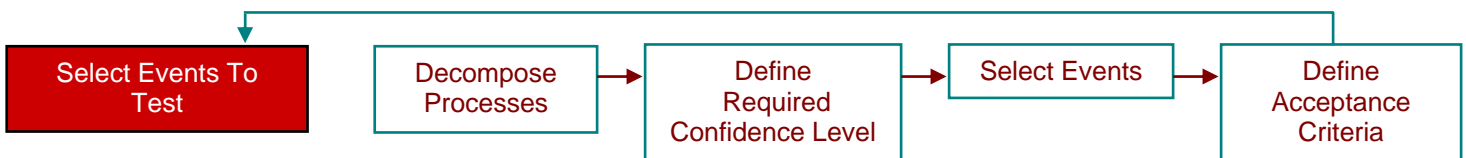
The Moderate confidence level processes usually represent the battleground. Initially, they should all be selected for testing. Later in the overall process, when achievability of the *Test Specification* is determined, teams should be prepared to eliminate some. Accordingly, Moderate confidence level processes should be ranked in descending order of importance, to serve as a basis for such elimination later.

Then, *Acceptance Criteria* are defined. For each selected process, identify the conditions that must be produced for the process to be regarded as operating acceptably. Examples could include a fully satisfied customer inquiry, or a shipment that is timely and complete when received by a customer, or an accurate bill.

These *Acceptance Criteria* serve as the basis for development of *Expected Results* later in the process. Actual test execution will involve determining whether the criteria have been satisfied for all processes tested, and team members will be required to assert that they have been satisfied before the test can be successfully completed.

The *Select Processes To Test* major activity is finalized by drafting the *Test Script*. One *Test Script* is developed for each process to be tested. Begin the draft at this point by briefly defining the process, its reasons for selection, and its *Acceptance Criteria*.

### Select Events to Test



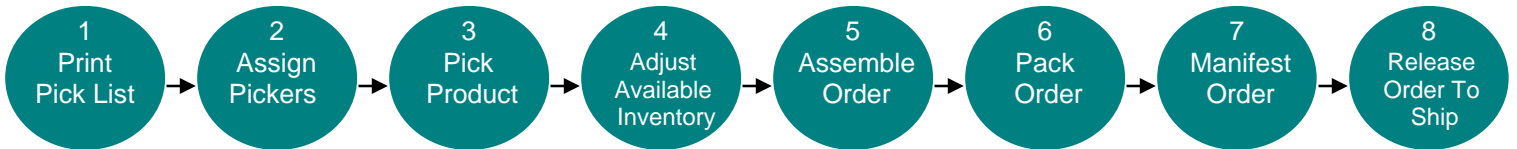
The first level of decomposition was achieved by taking partitions down to processes.

Normally, a series of subsequent, increasingly detailed decompositions results in a set of events, or sub-processes where each is sufficiently discrete to serve as the basis for a *test case*, which has a limited set of inputs and expected results. However, such an exhaustive modeling exercise is usually beyond the time available for *Test Specification* preparation.

Accordingly, teams often must drive directly from processes to events. This is the first activity in which individual team members can be leveraged by the team captain. Once processes have been selected for testing, they can be assigned to team members for decomposition into events, which can then be reviewed and confirmed by the entire team.

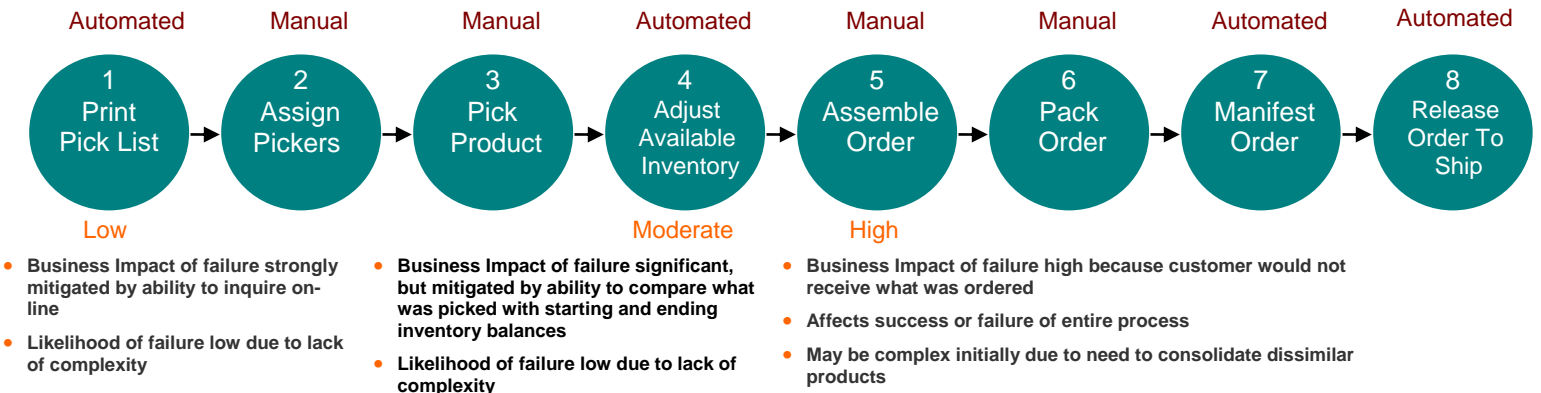
A process is a set of related, usually sequential, events or sub-processes that accomplish a coherent outcome, such as *Take an Order*, *Prepare an Order*, *Transport an Order*, or *Apply Cash*. An event can be a manual procedure, an external dependency that must be satisfied, or an automated system component. Examples of events for *Prepare an Order* could include *Print Pick List*, *Assign Pickers*, *Pick Product*, *Adjust Available Inventory*, *Assemble Order*, *Pack Order*, *Manifest Order*, and *Release Order to Ship*. The first, fourth, seventh and eighth of the example events would probably be automated system components while the rest would probably be manual procedures. The events may be represented as follows for the process as a series of sequential bubbles:

**Illustrative**



Once the events have been identified, level of confidence must be determined for each event, in a manner similar to that used with processes. Where is there a high business impact of failure? To what extent does the failure of the event cause the failure of the process? How easily can a failure be recovered? Given the complexity of the event and the known state of preparations, what is the likelihood of failure? The answers, and their impact on required confidence of events, may require a revisit of the required confidence assigned to the overall process.

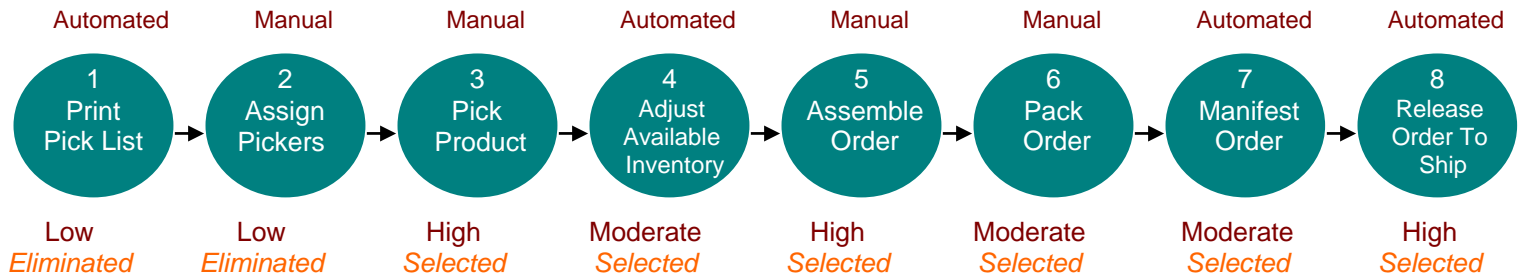
**Illustrative**



Once the confidence levels of the events have been identified, select those that are high and moderate, remembering that some moderates may need to be eliminated later if the test is unachievable with them included. Accordingly, rank the moderates in descending order of importance to serve as a basis for later elimination, if needed.

It may be that events assigned a low confidence level may become external dependencies, and may become a part of the initial test conditions, i.e., you specify their presence as a starting condition of the test. In the example below, *Print Pick List* and *Assign Pickers* might be eliminated, but a printed pick list by picker could be one of the starting conditions to test *Pick Product*.

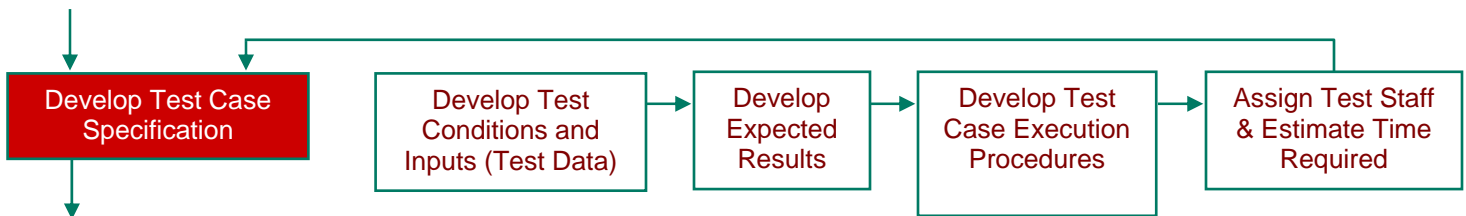
*Illustrative*



For each selected event, identify the conditions that must be produced for the event to be regarded as operating acceptably. Examples could include smooth performance of a procedure and satisfaction of specific control mechanisms that verify that information is processed completely and accurately.

As with processes, event based *Acceptance Criteria* serve as the basis for development of *Expected Results* later in the process, and will need to be verified during testing by team members.

*Develop Test Case Specification*



Each selected event represents the basis for developing a *Test Case Specification*. The first step in developing that specification is to identify the starting conditions that must be present for the test to be executed. Then, the controlled data that serves as input to the test must be identified and prepared.

Starting conditions may include external dependencies that must be present for the test to be executed, such as a printed pick list by picker, as in the example above, or fully trained personnel. In the case of fully trained personnel, evidence that the condition has been satisfied should be collected prior to test execution.

This could include a signed statement by supervisory personnel that training materials were provided and that, using those materials, personnel were trained in the procedure at question and are ready to be tested.

Other conditions that must be defined have frequency-and-volume, and pass-fail, related characteristics. If an event performed at one time, such as daily, has different characteristics when performed monthly, those conditions must be identified.

For example, if an automated procedure when performed daily needs to be performed in five minutes but can be performed in two hours when done monthly, that will affect *Expected Results*. Often, a distinguishing factor here will be volume of input. Similarly, the impact of failure of an event should be assessed when failure has a significant business impact, which requires introducing operating conditions or input that will force failure, to observe its recoverability and its broad impact. These distinctions become important later when *Test Case* execution is scheduled, because the same *Test Case* may be executed in different *Test Cycles*, where volume of input and pass-fail may distinguish cycles.

*Test data* must be developed which will be used as input to the execution of the *Test Case*. It must be of known content, to facilitate preparation of *Expected Results*. *Test Data* must be prepared for each frequency-and-volume and pass-fail condition identified above.

*Expected Results* must be developed for each *Test Condition* (other than starting conditions) within each *Test Case*. Materials that are most useful in developing *Expected Results* include written policies and procedures for performance of operational tasks and functional specifications and user manuals for automated procedures.

Given the known content of the input driving the test, and knowledge of how the task being tested is intended to operate for the stipulated condition, identify the expected outcomes of the task. Include those outcomes that span selected processes, even those that cross test partitions, but *not* those that affect matters outside all partitions. Examples of *Expected Results* include specific data values that appear in specific databases as a result of an automated event and condition, or clearly defined hand-offs from one manual procedure to another. The rule is that they be clearly observable as having been satisfied or not satisfied at the end of the *Test Case* execution.

Note that in pass-fail conditions *Expected Results* should also be defined, even though a major objective of the test is to *determine* the impact of a failure to operations. Here, team members should focus on anticipating and documenting the results that are *recoverable* within an acceptable timeframe. Actual results that are unanticipated should therefore represent real threats and can serve as a basis for prioritization of remedial action.

*Test Case Execution procedures* must be developed for each *Test Condition* (other than starting conditions) within each *Test Case*. These are simply the steps that must be taken to execute the *Test Case* for the stipulated condition. They must be sensitive to the testing environment that has been created, must assure that starting conditions obtain at the outset of the execution, must document the steps that must be taken to process *Test Data*, and must identify how *Expected Results* are to be verified. Each set of procedures defined by condition represents an occurrence of the *Test Case* in a *Test Cycle*.

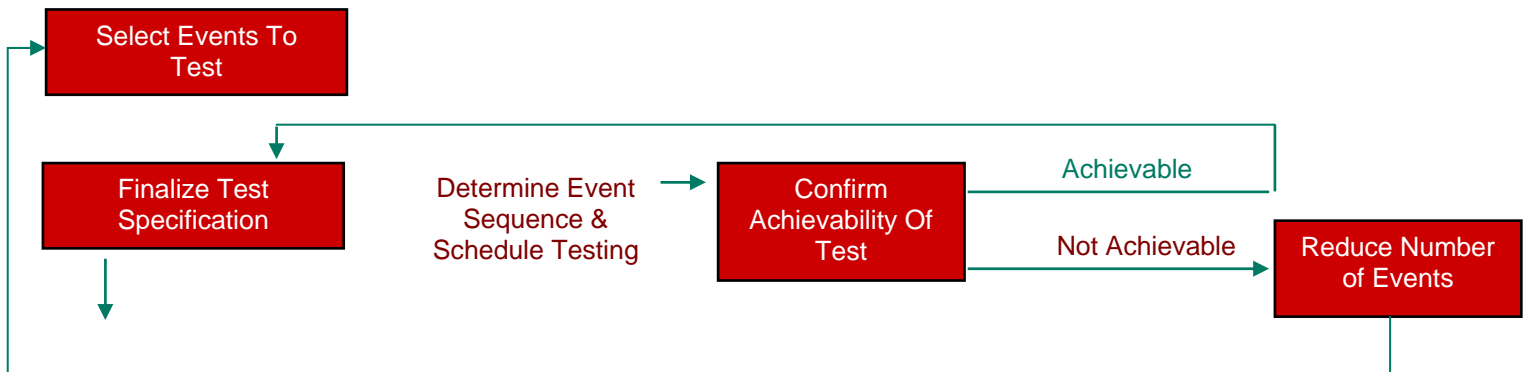
Assign one or more team members to the *Test Condition* within each *Test Case*. These individuals will be responsible for actual test execution and verification of *Expected Results*. Then, identify the operational staff that must participate in the test execution.

Examples of such personnel include distribution center pickers, administrative personnel who approve decisions included in the test and financial analysts who use reports that are outcomes of the test.

Finally, estimate the time required by all test participants to perform assigned roles and sequence the steps (*Test Procedures*) they must take, by individual, to form a critical path for the *Test Condition* (or *Test Cycle*) within the *Test Case*.

In order to finalize *Develop Test Case Specification*, all *Test Case Specification* documents must be completed and the individual schedules and critical paths for each *Test Condition* within each *Test Case* must be summarized into a critical path at the *Test Script*, or process, level.

### Finalize Test Specification



The script-level summaries of critical paths must be synchronized to determine achievability of the test in its planned approach. *Determine Event Sequence & Schedule Testing* accomplishes that at an overall test level, across processes and partitions.

*Confirm Achievability of Test* determines whether the overall test is *achievable*, given the resource requirements, against elapsed time constraints and availability of personnel. In the event that it is not achievable in its planned approach, the task also identifies the areas (processes) that should be revisited to modify the approach so as to accomplish achievability.

In the case where the overall test is not achievable, further elimination of events or perhaps of whole processes may be required to accomplish achievability. Based on the rankings developed in prior activities, such eliminations would take place, new critical paths developed at the script level and the whole subjected to iterative achievability tests until the overall test is pronounced achievable. At that time events proceed to actual *Test Execution*, based on *The Test Scripts* and *Test Case Specifications*.

### The Physical Environment Supporting Testing

The physical environment supporting testing should be separate from any other environment, such as production, training or development. This may be accomplished most simply through partitioning a computer or by use of a separate computer. In any event, access to test data, software and relevant tools undergoing or used in testing must be controlled.

Tools, such as automated facilities supporting test data generation, program planning and management, and business process reengineering, can dramatically enhance productivity in large, complex tests.

With respect to test data generation, care should be taken to retain all test data, in order to support *Regression Testing*. Put simply, *Regression Testing* re-tests any modified process or software component against all conditions every time it undergoes a test. This is done to assure that modifications performed against processes and software do not inadvertently cause problems with other elements of operation of the process or software. The assumption is that, if all conditions that determine correct processing are tested each time a modification occurs, then the likelihood of unintended and undiscovered consequences arising out of a modification are minimized.

Automated program planning and management tools can be very helpful in developing work breakdown structures (the actual hierarchy of activities, tasks and deliverables), schedules and budgets, then in tracking performance against them. Care should be taken with such tools, as they are expensive to maintain in terms of required skills and time: the largest, most complex testing efforts represent sufficient investment and risk to justify the expense involved in extensive use of these tools.

Business Process Reengineering (BPR) tools are varied, and include the following:

- A BPR planning tool
- An organizational entity analysis tool
- A process modeling analysis tool
- An activity based costing tool
- A graphical simulation tool
- A business metrics tool
- A benchmarking analysis tool.

As with automated program planning and management tools, care should be taken to be sure that the investment and risk associated with the testing process justifies the expense associated with using BPR tools. Normally, the practice of BPR with sufficient rigor to require these tools also requires engagement of professional management consultants familiar in BPR techniques and the use of the tools.

### 3. Test Execution

There are two major types of tests, the *Iterative Test* and the *Fixed Test*.

*Configuration Testing*, which is the iterative process of testing packaged software and the business processes supported during the prototyping and configuration exercise, is an example of an Iterative Test. Its primary characteristic is that it is performed in a disjointed way, its schedules slaved to the schedule for configuring the software, which is done by business process specific teams, often operating concurrently.

*System and Acceptance Testing* is performed with all software and processes complete in the opinion of developers and others, and is an example of a Fixed Test. Its primary characteristic is that it is performed end-to-end until satisfactorily completed.

A comprehensive test plan can encompass both major types of testing (and should), by employing the *Test Cycle* concept discussed above. Separate test cycles can be developed for Configuration Testing and for the different components of System and Acceptance Testing. In either use, extensive preparation is called for during test planning.

Typically, test teams operate out of segregated facilities, sometime called a *War Room*. The primary objectives of segregation are to minimize disruption, to support a team environment and to allow effective and efficient control over the testing process.

Execution of test case specifications proceeds for both major types of testing in very similar ways:

- Test materials are assembled
- Starting conditions are assured, including participation by all required personnel
- Test data (inputs) are delivered to the process or software module, and test procedures are executed
- Actual results of the operation are reviewed to determine conformance with expected results
- If actual results do not conform to expected results a *Severity Determination* is made and further required action is specified by policy governing the severity associated with the exception condition (see above, under 2. Test Planning and Organization – *Organization and Administration*).
- Test results are documented and, through the test organization, reported to management.

The results also are analyzed for the presence of patterns, which can reveal weaknesses in process or software preparation, or in test planning. For instance, a pattern could be revealed that excessive test failures are obtained in re-testing of modified software, determined through regression testing; in other words, while primary objectives of the modification might have been obtained, the modification caused unintended problems in other areas of the process or software operation. This could indicate inadequate analysis and design, or inadequate unit and string testing, by the technician(s) performing the modification. This in turn could induce tightening of inadequate procedures, or better supervision of the modification process, which could bring significant productivity gains to the testing process, and perhaps to the underlying prototyping process.