Designed by James Bach, Testing Consultant james@satisfice.com
Satisfice, Inc. http://www.satisfice.com

General Functionality and Stability Test Procedure

for Certified for Microsoft Windows Logo

Desktop Applications Edition

This document describes the procedure for testing the functionality and stability of a software application (hereafter referred to as "the product") for the purpose of certifying it for Windows 2000. This procedure is one part of the Windows 2000 compatibility certification process described in *Certified for Microsoft Windows Test Plan*.

This procedure employs an exploratory approach to testing, which means that the test cases are not defined in advance, but rather are defined and executed on the fly, while you learn about the product. We chose the exploratory approach because it is the best way to test a product quickly when starting from scratch.

This document consists of five sections:

- **Introduction to Exploratory Testing**
- **Working with Functions**
- **Testing Functionality and Stability**
- **Reading and Using this Procedure**
- **Test Procedure**

The first three parts explain the background and concepts involved in the test procedure. The fourth section gives advice about getting up to speed with the procedure. The fifth section contains the procedure itself.

This document is designed to be duplex printed (two sides on each page). For that reason, pages 2, 10, and 12 are intentionally blank.

Introduction to Exploratory Testing

With this procedure you will walk through the product, find out what it is, and test it. This approach to testing is called *exploratory* because you test while you explore. Exploratory testing is an interactive test process. It is a free-form process in some ways, and has much in common with informal approaches to testing that go by names like ad hoc testing, guerrilla testing, or intuitive testing. However, unlike traditional informal testing, this procedure consists of specific tasks, objectives, and deliverables that make it a systematic process.

In operational terms, exploratory testing is an interactive process of concurrent product exploration, test design, and test execution. The outcome of an exploratory testing session is a set of notes about the product, failures found, and a concise record of how the product was tested. When practiced by trained testers, it yields consistently valuable and auditable results.

The elements of exploratory testing are:

- *Product Exploration.* Discover and record the purposes and functions of the product, types of data processed, and areas of potential instability. Your ability to perform exploration depends upon your general understanding of technology, the information you have about the product and its intended users, and the amount of time you have to do the work.
- *Test Design.* Determine strategies of operating, observing, and evaluating the product.
- *Test Execution.* Operate the product, observe its behavior, and use that information to form hypotheses about how the product works.
- *Heuristics.* Heuristics are guidelines or rules of thumb that help you decide what to do. This procedure employs a number of heuristics that help you decide what should be tested and how to test it.
- *Reviewable Results.* Exploratory testing is a results-oriented process. It is finished once you have produced deliverables that meet the specified requirements. It's especially important for the test results to be reviewable and defensible for certification. As the tester, you must be prepared to explain any aspect of your work to the Test Manager, and show how it meets the requirements documented in the procedure.

Working with Functions

This procedure is organized around functions. What we call a function is anything the software is supposed to do. This includes anything that results in a display, changes internal or external data, or otherwise affects the environment. Functions often have sub-functions. For instance, in Microsoft Word, the function **`print`** includes the functions **`number of copies`** and **`page range`**.

Since we can't test everything, we must simplify the testing problem by making risk-based decisions about how much attention each function should get. For the purposes of Windows 2000 Certification, you will do this by identifying the functions in the product and dividing them into two categories: *primary* and *contributing*. For the most part, you will document and test primary functions. How functions are partitioned and grouped in the outline is a situational decision. At your discretion (although

the Test Manager makes the ultimate call) a group of contributing functions may be treated as a single primary function, or a single primary function may be divided into primary and contributing sub-functions.

Although you will test all the primary functions, if possible, you may not have enough time to do that. In that case, indicate in your notes which primary functions you tested and which ones you did not test.

It can be hard to identify some functions just by looking at the user interface. Some functions interact directly with the operating system, other programs, or modify files, yet have no effect that is visible on the screen. Be alert for important functions in the product that may be partially hidden.

The functional categories are defined as follows:

Definition	Notes
Primary Function Any function so important that, in the estimation of a normal user, its inoperability or impairment would render the product unfit for its purpose.	A function is primary if you can associate it with the purpose of the product *and* it is essential to that purpose. Primary functions define the product. For example, the function of adding text to a document in Microsoft Word is certainly so important that the product would be useless without it. Groups of functions, taken together, may constitute a primary function, too. For example, while perhaps no single function on the drawing toolbar of Word would be considered primary, the entire toolbar might be primary. If so, then most of the functions on that toolbar should be operable in order for the product to pass Certification.
Contributing Function Any function that contributes to the utility of the product, but is not a primary function.	Even though contributing functions are not primary, their inoperability *may* be grounds for refusing to grant Certification. For example, users may be technically able to do useful things with a product, even if it has an "Undo" function that never works, but most users will find that intolerable. Such a failure would violate fundamental expectations about how Windows products should work.

The first key to determining whether a function is primary is to know the purpose of the product, and that, in turn, requires that you have some sufficiently authoritative source of information from which to deduce or infer that purpose. The second key is knowing that a function is essential. That depends on your knowledge of the normal user, how the function works, and how other functions in the product work.

Testing Functionality and Stability

Your mission—in other words the reason for doing all this— is to discover if there are any reasons why the product should not be granted Certification, and to observe positive evidence in favor of granting Certification. In order to be Certified for Windows 2000, the product must be basically functional and stable. To evaluate this, you must apply specific criteria of functionality and stability.

These criteria are defined as follows:

Definition	Pass Criteria	Fail Criteria
Functionality The ability of the product to function.	1. Each primary function tested is observed to operate in a manner apparently consistent with its purpose, regardless of the correctness of its output.	At least one primary function appears incapable of operating in a manner consistent with its purpose.
	2. Any incorrect behavior observed in the product does not seriously impair it for normal use.	The product is observed to work incorrectly in a manner that seriously impairs it for normal use.
Stability The ability of the product to continue to function, over time and over its full range of use, without failing or causing failure.	3. The product is not observed to disrupt Windows.	The product is observed to disrupt Windows.
	4. The product is not observed to hang, crash, or lose data.	The product is observed to hang, crash, or lose data.
	5. No primary function is observed to become inoperable or obstructed in the course of testing.	At least one primary function is observed to become inoperable or obstructed in the course of testing.

The functionality standard is crafted to be the most demanding standard that can reasonably be verified by independent testers who have no prior familiarity with the product, and only a few days to complete the work. The word "apparently" means "apparent to a tester with ordinary computer skills". As the tester, you will not necessarily be able to tell that the program is functioning "correctly", but if you are able to tell that the product is *not* behaving correctly in a manner that seriously impairs it, the product fails the Certification.

In order to know if the product is seriously impaired for normal use, you must have a notion of what the normal user is like, and what is normal use. In many cases, the normal user can be assumed to be a person with basic computer skills; in other words, someone a lot like the normal tester. In some cases, however, the normal user will be a person with attributes, skills, or expectations that are specialized in some way. You may then have to study the product domain, or consult with the Vendor, in order to make a case that the product should be failed.

In order to perform the stability part of the test, you must also identify and outline the basic kinds of data that can be processed by the product. When testing potential areas of instability, you'll need to use that knowledge to design tests that use challenging input.

Test Coverage

Test coverage means "what is tested." The following test coverage is required under this procedure:

- *Test all the primary functions that can reasonably be tested in the time available.* Make sure the Test Manager is aware of any primary functions that you don't have the time or the ability to test.
- *Test a sample of interesting contributing functions.* You'll probably touch many contributing functions while exploring and testing primary functions.

- *Test selected areas of potential instability.* As a general rule, choose five to ten areas of the product (an area could be a function or a set of functions) and test with data that seems likely to cause each area to become unstable.

The Test Manager will decide how much time is available for the General Functionality and Stability Test. You have to fit all of your test coverage and reporting into that time slot. As a general rule, you should spend 80% of your time focusing on primary functions, 10% on contributing, and 10% on areas of instability.

Products that interact extensively with the operating system will be tested more intensively than other products. More time will be made available for testing in these cases.

Sources and Oracles

How do you know what the product is supposed to do? How do you recognize when it isn't working? These are difficult questions to answer outright. But here are two concepts you'll need in order to answer them to the satisfaction of the Test Manager: sources and oracles.

- *Sources.* Sources are where your information comes from. Sources are also what justifies your beliefs about the product. Sometimes your source will be your own intuition or experience. Hopefully, you will have access to at least some product documentation or will have some relevant experience. In some cases, you may need to consult with the Vendor to determine the purposes and functions of the product.
- *Oracles.* An oracle is a strategy for determining whether an observed behavior of the product is or is not correct. An oracle is some device that knows the "right answer." An oracle is the answer to the question "How do you *know* it works?" It takes practice to get good at identifying and reasoning about oracles. The significance of oracles is that they control what kinds of problems you are able to see and report.

Your ability to reason about and report sources and oracles has a lot to do with your qualifications to perform this test procedure. It also helps the Test Manager do his or her job. That's because a poor oracle strategy could cause you to assume that a product works, when in fact it isn't working very well at all. In many cases, you will not have a detailed specification of the product. Even if you had one, you wouldn't have time to read and absorb it all. Still, you and the Test Manager must determine if you can discover enough about the product to access and observe its primary functions. If your sources and oracles aren't good enough, then the Test Manager will have to get the Vendor to assist the test process.

A simple example of an oracle is a principle like this: "12 point print is larger than 8 point print." Or "Text in WordPad is formatted correctly if the text looks the same in Microsoft Word."

One generic pattern for an oracle is what we call the Consistency Heuristics, which are as follows:

- *Consistence with Purpose*: Function behavior is consistent with its apparent purpose.
- *Consistence within Product:* Function behavior is consistent with behavior of comparable functions or functional patterns within the product.
- *Consistence with History:* Present function behavior is consistent with past behavior.

- *Consistence with Comparable Products:* Function behavior is consistent with that of similar functions in comparable products.

Even if you don't have certain knowledge of correct behavior, you may be able to make a case for incorrect behavior based on inconsistencies in the product.

Reading and Using this Procedure

This procedure follows the pattern of a "forward-backward" process, as opposed to a step-by-step process. What that means is that you will go back and forth among the five different tasks until all of them are complete. Each task influences the others to some degree; thus, each task is more or less concurrent with the others. When all tasks are complete, the whole procedure is complete.

Forward-backward processes are useful in control or search situations. For example, a forward-backward process we're all familiar with is driving a car. When driving, the task of checking the speedometer isn't a sequential step in the process, it's a concurrent task with other tasks such as steering. When driving somewhere, the driver does not just think forward from where he is, but backwards from where he wants to go. Exploratory testing is, in a sense, like driving. Also, like driving, it takes some time, training and practice to develop the skill.

Task Sheets

This procedure consists of five tasks, which are documented in the Test Procedure section, below. Each task is described by a task sheet with the following elements:

- *Task Description.* Located at the top of each sheet, the task description is a concise description of what you are supposed to do.
- *Heuristics.* In the middle of each sheet is one or more lists of ideas. We call them heuristics. Heuristics are guidelines or rules of thumb that help you decide what to do. They are not sub-tasks that must be "completed." Instead, they are meant to both provoke and focus your thinking. The way to use to them is to visit each idea briefly, and consider its implication for the product you are testing. For example, in the Identify Purposes task, there is a list of potential purpose verbs. One of the ideas on that list is "solve, calculate." When you see that, think about whether one of the purposes of the product is to solve or calculate something. If the product has such a purpose, you might write a purpose statement that includes "Perform various mathematical calculations." If the product has no such purpose, just shrug and move on.
- *Results.* Located at the bottom left of each sheet is a list of what you are expected to deliver as a result of that task.
- *You can say you're done when…* An important issue in a procedure like this is: How do you know when you're done? So, in the bottom right of each task sheet is a list of things that must be true in order for you to be done. In other words, it's not enough simply to produce something that you call a result according to the list at the bottom left. You also have to be prepared to defend the truth of the statements on the right. Most of those statements will require some subjective judgment, but none of them is totally subjective.
- *Frequently Asked Questions.* On the opposite side of each page (this document is designed to be printed two-sided), you'll find a list of answers to questions that testers generally have when first encountering that task.

The Role of the Test Manager

The Test Manager has ultimate responsibility for the quality of the test process. If any questions are raised about how you tested, the Test Manager must be prepared to vouch for your work. For that reason, escalating issues and questions to the Test Manager is an important part of *your* role.

Issues and Questions

Issues and questions will pop up during the course of your work. If you can't immediately resolve them without interrupting the flow of your work, then note them and try to resolve them later. These include specific questions, general questions, decisions that must be made, as well any events or situations that have arisen that have adversely impacted your ability to test.

It's important to write down issues and questions you encounter. Your notes may be revisited by another tester, months later, who will be testing the next version of the product. By seeing your issues, that tester may get a better start on the testing. Writing down the issues also gives the Test Manager, or anyone else who reviews your notes, a better ability to understand how the testing was done.

When to Escalate

In the following situations, ask the Test Manager how to proceed:

- You encounter an obstacle that prevents you from completing one or more of the test tasks.
- You feel lost or confused due to the complexity of the product.
- You feel that you can't learn enough about the product to test it well, within the timeframe you've been given.
- You encounter a problem with the product that appears to violate the functionality or stability standards.
- You feel that the complexity of the product warrants more time for testing than was originally allotted.

Testing Under Time Pressure

The amount of time allotted to test the product will vary with its complexity, but it will be on the order of hours, not days. Your challenge will be to complete all five tasks in the time allotted. Here are some ideas for meeting that challenge:

- *The first question is whether testing is possible.* Some products are just so complex or unusual that you will not be able to succeed without substantial help from the Vendor. In order to do a good job completing this test procedure on a tight schedule, you first must determine that the job can be done at all.

- *Make a quick pass through all five tasks.* Visit each one and get a sense of where the bulk of the problems and complexities will be. In general, the most challenging part of this process will be identifying and categorizing the product functions.
- *Pause every 20 or 30 minutes*. Assess your progress, organize your notes, and get some of your questions answered.
- *If you feel stuck in one task, try another*. Sometimes working on the second task will help clear up the first one. For instance, walking through the menus of the product often sheds light on the purpose of the product.
- *Tackle hard problems first.* Sometimes clearing up the hard parts makes everything else go faster. Besides, if there's a problem that is going to stop you cold, it's good to find out quickly.
- *Tackle hard problems last.* Alternatively, you could leave some hard problems until later, on the hope that doing an easier task will help you make progress while getting ready to do the rest.
- *Set aside time to clean up your notes.* The final thirty minutes or so of the exploratory test should be set aside for preparing your notes and conclusions for delivery, and doing a final check for any loose ends in your testing.
- *Keep going.* Unless you encounter severe problems or obstacles, keep the process moving. Stay in the flow of it. Write down your questions and issues and deal with them in batches, rather than as each one pops up.

The Prime Directive: Be Thoughtful and Methodical

Throughout the test procedure, as you complete the tasks, you have lots of freedom about how you do the work. But you must work *methodically*, and follow the procedure. In the course of creating the result for each task, you'll find that you have to make a lot of guesses, and some of them will be wrong. But you must *think*. If you find yourself making wild and uneducated guesses about how the product works, areas of instability, or anything else, stop and talk to the Test Manager.

Test Procedure

Complete these five tasks:

- ❑ Identify the purpose of the product.
- ❑ Identify functions.
- ❑ Identify areas of potential instability.
- ❑ Test each function and record problems.
- ❑ Design and record a consistency verification test.

Things to Deliver	You can say you're done when...
❑ Purpose statement ❑ Function outline ❑ List of potential instabilities and challenging data ❑ Product failures and notes ❑ Consistency verification test	❑ Each task is complete. ❑ Each question and issue is either resolved or accepted by the Test Manager. ❑ Each task deliverable is accepted by the Test Manager. ❑ You know enough about the product to determine whether it should or shouldn't receive Certification according to the functionality and stability criteria.

➢ Identify the purpose of the product.

1. Review the product and determine what fundamental service it's supposed to provide. To the extent feasible, define the audience for the product.

2. Write (or edit) a paragraph that briefly explains the purpose of the product and the intended audience.

Some Potential Purpose Verbs for Use in the Statement

- Create, Edit
- View, Analyze, Report
- Print
- Solve, Calculate
- Manage, Administer, Control
- Communicate, Interoperate
- Serve Data, Provide Access, Search
- Support, Protect, Maintain
- Clean, Fix, Optimize
- Read, Filter, Translate, Convert
- Entertain

Some Attributes of Users That May be Worth Discussing in the Statement

- Special skills,knowledge, abilities or disabilities
- Troubleshooting ability
- Expectations or needs
- Limitations (who will *not* be a user of this product)

Things to Deliver	You can say you're done when...
❑ Purpose statement ❑ Issues/questions	❑ You have performed the task as described above. ❑ The purpose statement is based on explicit or implicit claims made by the Vendor. ❑ All aspects of the product's purpose that are important to a normal user are identified. ❑ The purpose statement is fundamental (if it couldn't be fulfilled, the product wouldn't be fit for use).

Purposes: Frequently Asked Questions

Why does this task matter?

Without an understanding of the purposes of the product, you can't defend the distinctions you make between primary and contributing functions. And those distinctions are key, since most of your testing effort will focus on the primary functions. You don't need to write an essay, but you do need to include enough detail so that any function that you think is important enough to call primary can be traced to that statement.

How do I write a purpose statement?

If the Vendor supplies a product description with the Vendor Questionnaire, start with that and flesh it out as needed. If you have to write it yourself, start with a verb and follow with a noun, as in "edit simple text documents", or "produce legal documents based on input from a user who has no legal training." Also, if there are any special attributes that characterize a normal user of the product, be sure to mention them.

The list of purpose verbs comes from all the purposes gleaned from a review of software on the racks of a large retail software store. It may help you notice purposes of the product that you may otherwise have missed. Similar purpose verbs are grouped together on the list to save space (e.g. calculate, solve), and not because you're supposed to use them together.

How are purposes different from functions?

Purpose relates to the needs of users. Functions relate to something concrete that is produced or performed by the product.

Sometimes the purpose of a function and the name of the function are the same, as in "print": printing is the purpose of the print function. Most of the time, a function serves a more general goal that you can identify. For instance, the purpose of a word processor is not to search for and replace text; instead search and replace are part of editing a document. Editing is the real purpose. On the other hand, in a product we could imagine called "Super Search and Replace Pro," the search and replace function presumably *is* the purpose of the product.

➢ Identify functions.

1. Walk through the product and discover what it does.
2. Make an outline of all primary functions.
3. Record contributing functions that are interesting or borderline primary.
4. Escalate any functions to the Test Manager that you do not know how to categorize, or that you are unable to test.

Some Ways to Look for Functions

- Check online help.
- Check the Vendor Questionnaire.
- Check all programs that comprise the product.
- Check all product menus.
- Check all windows.
- Check toolbars.
- Check all dialog boxes and wizards.
- Right-click on all data objects, interface elements, and window panes (this might reveal context menus).
- Double-click on all data objects, interface elements, and window panes (this might trigger hidden functions).
- Check product options settings for functions that are dormant unless switched on (e.g., automatic grammar checking in Microsoft Word).
- Check for functions that are triggered only by certain input (e.g., saving a JPEG image might trigger a JPEG Save wizard).
- Examine sample data provided with the product.
- Check for error handling and recovery functions that are embedded in other functions.

Function Classification

- **Primary:** Any function so important that, in the estimation of a normal user, its inoperability or impairment would render the product unfit for its purpose.
- **Contributing:** Any function that contributes to the utility of the product, but is not a primary function.

Things to Deliver	You can say you're done when...
❑ Function outline ❑ Issues/questions	❑ You have performed enough of the *Identify the Purpose of the Product* task to enable you to correctly categorize functions of the product. ❑ You have performed the task as described above. ❑ Each primary function you identified is *essential* to the fulfillment of the purpose of the product. ❑ You have explored enough to reasonably conclude that all interesting functions of the product are accounted for.

Functions: Frequently Asked Questions

Why does this task matter?

By listing the functions that comprise the operation of the product, you are making an outline of what could be tested. When you complete the testing, this outline is an indicator of what you understood the product to be, and what you might have tested. This outline is an important record for use by the Test Manager or the Vendor as a reference in case they want to question you about what you did and did not do, or by other testers who may test this product in the future.

What if I'm totally confused as to what are the primary functions?

Escalate to the Test Manager. Do not simply choose arbitrarily. The Test Manager will contact the Vendor for information, locate documentation, or otherwise advise you what to do.

In what format should I record the functions?

Keep it simple. Use a two- or three-level outline. Record a one-line bullet for each function or functional area. Sometimes a function will not have an official name or label. In that case, make up a name and put it in square brackets to indicate that you invented the name. If there are a hundred functions that all belong one family, list the name of the group as in "Drawing functions," rather than listing each by itself.

If you identify contributing functions, clearly distinguish them from the primary functions.

For example: here is a portion of the function outline for Microsoft Bookshelf:

```
Note...
      Add Current Article
      Delete
      Goto
      Annotation
Search All...
      [Result Outline]
      Articles About...
      Articles Containing the Words...
Find Media...
      All Media...
      Audio...
      Images...
      Animations...
      (Result List)
Go Online...
      BookShelf Premier Search
      BookShelf Premier News
      Encarta Online Library
Advanced Search...
      Books
      Media
      Articles
      [Search Hit Highlighting]
```

➢ Identify areas of potential instability.

1. As you explore the product, notice functions that seem more likely than most to violate the stability standards.

2. Select five to ten functions or groups of functions for focused instability testing. You may select contributing functions, if they seem especially likely to fail, but instability in primary functions is more important.

3. Determine what you could do with those functions that would potentially destabilize them. Think of large, complex, or otherwise challenging input.

4. List the areas of instability you selected, along with the kind of data or strategies you'll use to test them.

Some Areas of Potential Instability

- Functions that interoperate with other products (e.g. object linking and embedding, file conversion).
- Functions that handle events external to the application (e.g. wake up a sleeping computer when a fax arrives).
- Functions that make intensive use of memory.
- Functions that interact extensively with the operating system.
- Functions of unusual complexity.
- Functions that change operating parameters (e.g. preference settings)
- Functions that manipulate operating system configuration.
- Functions that intercept or recover from errors.
- Functions that replace basic operating system functions (undelete files or process user logon).
- Any function or set of functions that involve multiple simultaneous processes.
- Functions that manipulate multiple files at once.
- Functions that open files over a network.

Some Ideas About Challenging Data

- **Documents:** Long documents; a lot of documents open at once; or documents containing lots of different objects.
- **Records:** Long records; large numbers of records, or complex records.
- **Lists:** Long lists; empty lists; multicolumn lists.
- **Fields:** Enter lots of characters; very large values.
- **Objects:** Lots of objects; too many characters; large objects; compound objects.
- **Changes:** Add and delete things; edit without saving or exiting.
- **Loads:** Get a lot of processes going at once; batch processing with large batches; do lots of things in a very short time.
- **Non sequiturs:** Click randomly around windows; type randomly on keys; enter unexpected input.
- **Exceptions and Escapes:** Interrupt processes over and over again; cancel operations; give erroneous data to trigger error handling.

Things to Deliver	You can say you're done when...
❑ List of potential instabilities and challenging data ❑ Issues/questions	❑ You have completed exploring the product and looking for areas of potential instability. ❑ You have performed the task as described above. ❑ Everything you identify represents something you will test or have tested. ❑ For each potential instability you identify, you can state your reasoning and your sources.

Instabilities: Frequently Asked Questions

Why does this task matter?

When testing for stability, it's a good idea to focus your efforts on areas that are more likely to become unstable. Some input data you give to a product is more likely than others to trigger instability.

What is instability?

Any behavior that violates the stability standard. Obvious instabilities are crashes. The basic difference between functional failures and instabilities is that, with the latter, the function *can* work but sometimes doesn't. The function is unreliable, but not completely inoperable. It is also often called instability when a function works correctly in some ways, but has negative side effects, such as corrupting some other function or product.

How do I know what is potentially unstable?

You can't know for sure. The heuristics we provide are general hints. As you explore the product, you may get a feeling about what parts of the product may be unstable. Corroborate your initial suspicions with quick tests. Let's say you suspect that a particular function may harbor instabilities because it's complex and seems to make intensive use of memory. You could corroborate your hypothesis about its complexity just by looking at the complexity of its visible inputs and outputs, and the varieties of its behavior. You could corroborate your hypothesis about memory use by using the Task Manager to watch how that product uses memory as it executes that function.

Once you have a definite idea that a function might be unstable, or at least has attributes that are often associated with instability, design a few tests to overwhelm or "stress" the function. When testing for instability, you don't need to restrict yourself to normal input patterns. However, instabilities exhibited with normal input are certainly very interesting.

➢ Test each function and record results.

1. Test all the primary functions you can in the time available.
2. Test all the areas of potential instability you identified.
3. Test a sample of interesting contributing functions.
4. Record any failures you encounter.
5. Record any product notes you encounter. Notes are comments about quirky, annoying, erroneous, or otherwise concerning behavior exhibited by the product that are not failures.

Grounds for Refusing to Certify

- At least one primary function appears incapable of operating in a manner consistent with its purpose.
- The product is observed to work incorrectly in a manner that seriously impairs it for normal use.
- The product is observed to disrupt Windows.
- The product is observed to hang, crash, or lose data.
- A primary function is observed to become inoperable or obstructed in the course of testing.

Failure Investigation

- Note symptoms of the problem, and justify why it's severe enough to cause failure to Certify.
- Reproduce the problem, if feasible. Provide an estimated percentage of reproducibility.
- Check for additional failures of a similar kind. Determine if this is an isolated case.
- Report to the Test Manager for confirmation.

Things to Deliver	You can say you're done when...
❑ Product failures ❑ Product notes ❑ Issues/questions	❑ You have completed enough of the *Identify Functions* task to know the primary and contributing functions to test, and you have completed enough of the *Identify Areas of Potential Instability* task to know what stability testing to perform. ❑ You have performed the task as described above. ❑ You have alerted the Test Manager about any primary functions you could not test. ❑ You have recorded failures in enough detail to allow the Vendor to reproduce them or otherwise get a clear idea of the symptoms.

Test and Record Problems: Frequently Asked Questions

Why does this task matter?

This is the heart of the whole process. This is the actual testing. The other tasks help you perform this one.

Wouldn't the process be better if this were the last task to be done?

Only in theory. In practice, testing itself almost always reveals important information about the other tasks that you could not reasonably have discovered any other way. You may think you've completed the other tasks, and then feel the need to revisit them when you're actually testing the functions.

Why shouldn't I write down all the tests I design and execute?

Although it's a common tenet of good testing to write down tests, the problem is that it takes too much time and interrupts the flow of the testing. If you stop to write down the details of each test, you will end up writing a lot and running very few tests. Besides, it isn't necessary, as long as you can give an overview of what you tested and how, on demand. All the other notes you deliver in this process will help you prepare to do that.

The only test you write down, in this procedure, is the consistency verification test, which represents a small subset of the testing you did.

➢ Design and record a consistency verification test.

1. Record a procedure for exercising the most important primary functions of the product to assure that the product behaves consistently on other Windows platforms and configurations.

Consistency Verification Test Requirements

- The test must be specific enough that it can be repeated on the same Windows platform with the same system configuration by different testers, and all testers will get the same results.
- Cover each of the most important primary functions with a simple test.
- Include steps to manipulate graphical objects created within the product.
- Include steps that select objects, drag and drop them.
- Include steps that render and repaint windows across multiple monitors.
- Specify and archive any data needed for the test.
- Specify some complex data for use in the test.
- Use specific file names and path names.
- Make the test as short and simple as you reasonably can, while meeting these requirements.

Things to Deliver	You can say you're done when...
❑ Consistency verification test ❑ Issues/questions	❑ You have completed enough of the *Identify Functions* task to know which primary functions to include in the consistency verification test. ❑ You have performed the task as described above. ❑ The test meets all of the requirements listed above. ❑ You have executed the test, from beginning to end, exactly as specified.

Consistency Verification Test: Frequently Asked Questions

Why does this task matter?

After the general functionality and stability test is complete, during the rest of the test process there will be an occasional need to perform a simple re-test of functionality and stability. The consistency verification test defines that activity. It's important that this test be precisely defined, because its purpose is to see if changes in Windows platforms or configurations reveal incompatibilities within the product.

Is this like a "smoke test"?

The term "smoke test" comes from the electronics industry. After a repair, a technician would turn on the device, a television set for example, and look for smoke. The presence of smoke drifting up from the circuit boards told the technician that some parts were getting too much current. If no smoke appeared immediately, the technician would try some simple operations, such as changing the selected channel and volume settings. If the television did those basic functions without any smoke appearing, the technician felt confident to proceed with more specific tests.

The consistency verification test is just like a smoke test, except it's important to define the test with sufficient precision enough that substantially the same test is executed every time.

How deep should the test be?

Notice what the television technician found out quickly from the smoke test:

- The television set turned on. Picture and sound appeared.
- The basic stuff seemed to work. The user could change channels and turn the volume up and down.
- Nothing burned up.

Notice the detailed tests the technician did not run:

- No attempt to change brightness, contrast or color settings.
- No tests for all possible channels.
- No tests using alternate inputs or outputs.
- No tests using alternate user interfaces (the technician used either controls on the set or the hand-held remote control, but not both).

The consistency verification test you design for the product should verify it at the same level that the technician's smoke tests verified the television set. You should test one example of each major primary function in at least one normal usage.

Another way to think about the test is that it's the set of things you can do with the product that will give the most accurate impression possible of the quality of the product in a few minutes of test time.