

# ODC Experiment

## Applying ODC to Mobile Platform Testing

A SIPTECH WHITE PAPER

Version 1.04

---

### Inside this document ...

- 01. Introduction
- 02. Background
- 03. Approach
- 04. ODC Definition
- 05. ODC Pitfalls
- 06. Necessary Conditions
- 07. Sufficient Conditions
- 08. ODC Implementation
- 09. Analyze Defect Data
- 10. Next Steps
- 11. Appendix



---

© 2005 All rights reserved. SIP Technologies and Exports Ltd. The information contained in this document is CONFIDENTIAL and PROPRIETARY in nature, and subject to the rights and ownership of SIPTECH. Any and all unauthorized copying or use of the contents hereof is prohibited.

## Acronyms, Abbreviations and Symbols

Acronym	Definition
ODC	Orthogonal Defects Classification

## References

1. Orthogonal Defect Classification - A Concept for In-Process Measurements Ram Chillarege, Inderpal S. Bhandari, Jarir K. Chaar, Michael J. Halliday, Diane S. Moebus, Bonnie K. Ray, Man-Yuen Wong, IEEE Transactions on Software Engineering, Vol 18, No. 11, Nov 1992. <http://www.chillarege.com/odc/articles/odcconcept/odc.html>
2. Statistical Software Engineering. Panel on Statistical Methods in Software Engineering Committee on Applied and Theoretical Statistics Board on Mathematical Sciences Commission on Physical Sciences, Mathematics, and Applications National Research Council, 1996. <http://www.nap.edu/readingroom/books/statsoft/index.html>
3. "Software Triggers as a function of time - ODC on field faults", Ram Chillarege and Kathryn A. Bassin, DCCA-5: Fifth IFIP Working Conference on Dependable Computing for Critical Applications, Sept 1995. <http://www.chillarege.com/odc/articles/trig/trig.html>

## 1. Introduction

SIPTECH's stated vision is to become the preferred service testing services provider to companies that focus on product and technology verticals. As part of the efforts to realize this vision several initiatives have been undertaken or planned. One set of such initiatives are to identify best practices in the industry, evaluate them and assimilate them to provide better value to customers in terms of quality, timeliness and cost effectiveness by continually improving the processes. SIPTECH has identified Orthogonal Defect Classification as one of the significant practices that need to be evaluated and integrated into SIPTECH's testing practices.

## 2. Background

The primary focus of software engineering is to monitor a software development process to improve quality and productivity.

Quality can be improved by two different approaches. One approach considers each defect as unique and tries to identify a cause. The second approach considers a defect as a sample from a collection to which a formal statistical reliability model is applied.

Chillarege et al. (1992) proposed a new methodology that strikes a balance between these two ends of the spectrum. This method, called orthogonal defect classification, is based on exploratory data analysis techniques and has been found to be quite useful at IBM. It recognizes that the key to improving a process is to quantify various cause-and-effect relationships involving defects.

The basic approach is as follows. First, classify defects into various types. Then, obtain a distribution of the types across different development phases. Finally, having created these reference distributions and the relationships among them, compare them with the distributions observed in a new product or release. If there are discrepancies, take corrective actions.

Operationally, the defects are classified according to eight "orthogonal" (mutually exclusive) defect types:

- ❖ functional
- ❖ assignment
- ❖ interface
- ❖ checking
- ❖ timing
- ❖ build/package/merge
- ❖ data structures and algorithms and

- ❖ documentation.

Further, development phases are divided into four basic stages (where defects can be observed):

- ❖ design
- ❖ unit test
- ❖ function test and
- ❖ system test.

For each stage and each defect type, a range of acceptable baseline defect rates is defined by experience. This information is used to improve the quality of a new product or release. Towards this end, for a given defect type, defect distributions across development stages are compared with the baseline rates. For each chain of results-say, too high early on, lower later, and high at the end-an implication is derived. For example, the implication may be that function testing should be revamped.

This methodology has been extended to a study of the distribution of triggers, that is, the conditions that allow a defect to surface. First, it is implicit in this approach that there is no substitute for a good data analysis and there is a stable environment with similar projects. Thus, it may be necessary to create classes of reference distributions and classes of similar projects.

Next, although the defect types are mutually exclusive, it is possible that a fault may result in many defects, and vice versa. This multiple-spawning may cause serious implementation difficulties. Proper measurement protocols may diminish such multi-propagation.

Finally, given good-quality data, it may be possible to extend orthogonal defect classification to efforts to identify risks in the production of software, perhaps using data to provide early indicators of product quality and potential problems concerning scheduling. The potential of this line of inquiry should be carefully investigated, since it could open up an exciting new area in software engineering.

It is this last aspect constitute the focus of SIPTECH's pilot of the application of ODC to an on going project to assess the feasibility of analyzing the available defect data to come-up with in process improvements to the testing and development process.

### 3. Approach

#### **In-Process ODC Feedback Loops**

Orthogonal defect classification is a measurement method that uses the defect data to provide precise measurability into the process. Given the measurement, a variety of analysis techniques have been developed to assist management and decision-making on a range of software engineering activities.

One of the uses of ODC has been the ability to close feedback loops in a software development process, which has traditionally been a difficult task. While ODC can be used for a variety of other software management methods, closing of feedback loops has been found over the past few years to be a much needed process improvement and cost control mechanism.

### 4. ODC Definition

Orthogonal Defect Classification (ODC) essentially means that we categorize a defect into classes that collectively point to the part of the process which needs attention, much like characterizing a point in a Cartesian system of orthogonal axes by its (x, y, z) coordinates.

In the software development process although activities are broadly divided into design, code, and test, each organization can have its variations. It is also the case that the process stages in several instances may overlap while different releases may be developed in parallel. Process stages can be carried out by different people and sometimes, different organizations.

Therefore, for classification to be widely applicable, the classification scheme must have consistency between the stages. Without consistency it is almost impossible to look at trends across stages. Ideally, the classification should also be quite independent of the specifics of a product or organization. If the classification is both consistent across phases and independent of the product, it tends to be fairly process invariant and can eventually yield relationships and models that are very useful.

Thus, a good measurement system, which allows learning from experience and provides a means of communicating experiences between projects, has at least three requirements:

- ❖ Orthogonality,
- ❖ Consistency across phases, and
- ❖ Uniformity across products.

## 5. ODC Pitfalls

One of the pitfalls in classifying defects is that it is a human process, and is subject to the usual problems of human error, confusion, and a general disinterest if the use of the data is not well understood.

However, each of these concerns can be handled if the classification process is simple, has less room for confusion or possibility of mistakes, and if the data can be interpreted easily.

If the number of classes is small, there is a greater chance that the human mind can accurately resolve between them. Having a small set to choose from makes classification easier and less error prone. When orthogonal (I.e. mutually exclusive), the choices also have higher probability of being uniquely identified and easily classified.

## 6. Necessary Conditions

There exists a semantic classification of defects, from a product, such that the defect classes can be related to the process, which can explain the progress of the product through this process.

## 7. Sufficient Conditions

The set of all values of defect attributes must form a spanning set over the process sub-space.

## 8. ODC Implementation

The goal was to take up a module or component and apply ODC as a pilot study to assess the usefulness of ODC in providing in-process feedback on test effectiveness to help make necessary process changes as early as possible.

Another goal was to use this experience to come-up with a detailed process that can be applied to a different project or different organization helping in the integration of ODC into the existing processes and toolset.

The following sections discuss the process followed in the Pilot in implementing ODC, analysis results and inferences drawn.

### **Map Organizational to ODC Defect types**

Each organization will have its own processes and tools for collecting, tracking and managing the defect data and these would have evolved over a period of time. This process would also be well

integrated with the software engineering processes that are followed in that organization depending on the maturity level of the organization.

This poses a significant challenge in terms of assessing the feasibility of implementation of ODC, planning the implementation process and establishing the long-term benefits that may accrue by the deployment of ODC into the existing process framework. A study of attributes that are captured in organizational defect tracking system is conducted and the results are documented.

As the next step, an assessment of whether there are any attributes that has a match with ODC defect types is checked and the results are recorded. (Appendix I)

If there are no existing attributes that represent the ODC types then a new attribute to capture ODC defect type's information is introduced.

#### **Map Organizational to ODC triggers**

A study of attributes that are captured in organizational defect tracking system is conducted and the results are documented.

As the next step, as assessment of whether there are any attributes that has a match with ODC trigger is checked and the results are recorded. (Appendix II)

If there are no existing attributes that represent the ODC triggers then a new attribute to capture ODC trigger information is introduced.

#### **Defect data classification**

During this pilot, the defect data is offloaded to a spreadsheet with important attributes as captured in the defect tracking system. This data is augmented with the additional fields to capture ODC specific attributes like type, trigger, etc.

Given the fact that the development team's participation will not be available, the focus was restricted to the opener section of defect classification as per ODC method.

Since multiple members were involved in the classification process, it was necessary to review the classifications and discussions between the members to minimize issues like multi-propagation and bring in uniformity in the classification. The learnings from this exercise have been incorporated into the ODC implementation process documentation.

## 9. Analyze Defect Data

The primary objective in this detailed quantitative analysis is to gain a clearer understanding of the distribution of Opener section attributes and to see what inferences that can be drawn from that analysis. These inferences are expected to guide the development and testing process improvement steps in process.

Some of the questions that were formulated to draw inferences from were:

- ❖ Was the test process independently responsible for the delays?
- ❖ Were there elements of development that could have helped the test cycle?
- ❖ What would make the test process more effective?

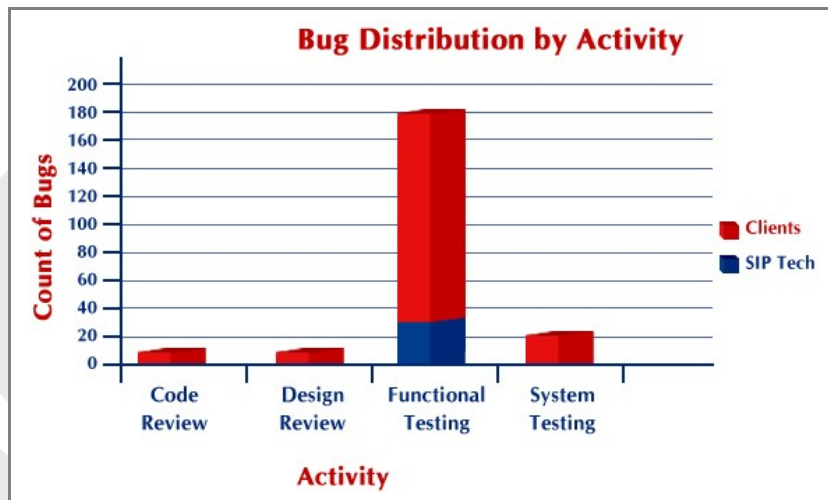
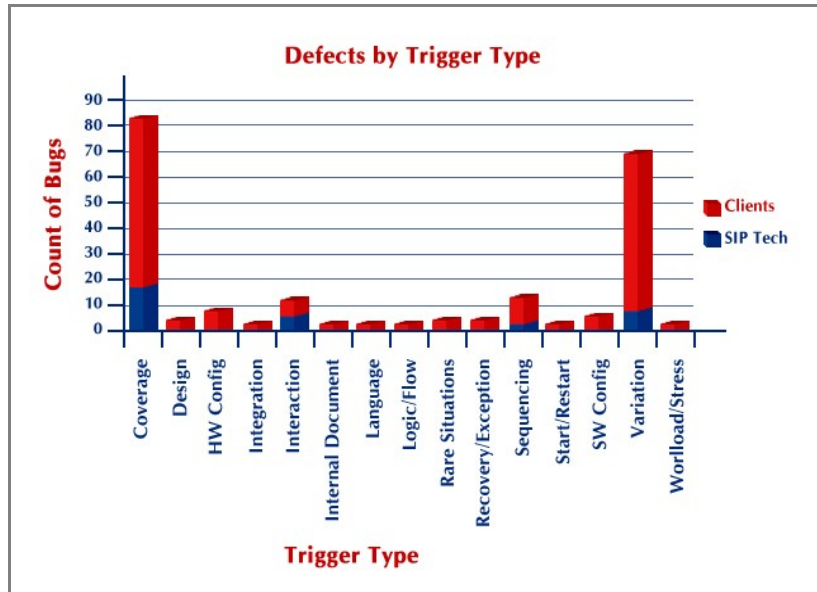
The answers to these questions need to be substantiated with detail. This and next sections analyze the data and summarize the inferences. Then these inferences based on analysis are related back to ODC pilot goals.

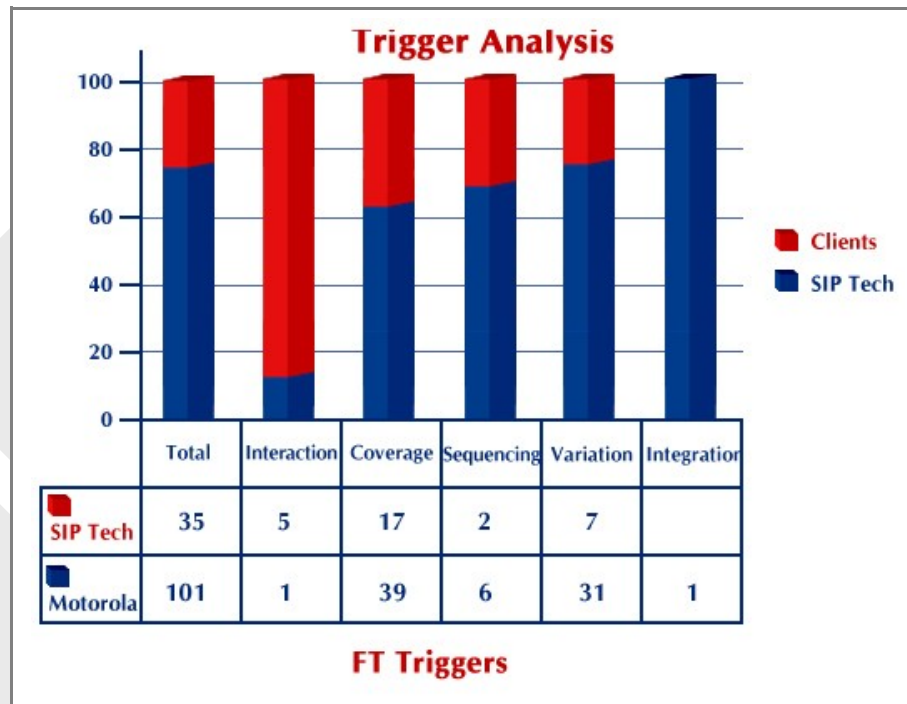
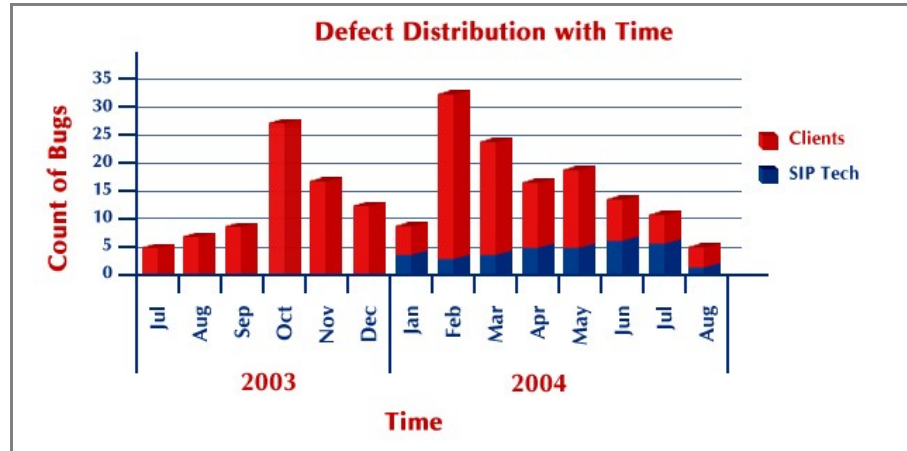
Provides three trigger distributions as a function of time, attributable to escapes to the field from each of the verification activities: review, function test and system test.

Illustrates that each trigger peaks at a different time from date of release. This is a key finding with significant implications in several aspects of software dependability and software engineering.

### TRIGGER DISTRIBUTION AND ANALYSIS

- ❖ Trigger Distribution by type
- ❖ Trigger Distribution by Phase
- ❖ Trigger vs Time
- ❖ Trigger Break-up Analysis (SIPTECH vs Client)





**Strength**

- ❖ Interaction
- ❖ Coverage

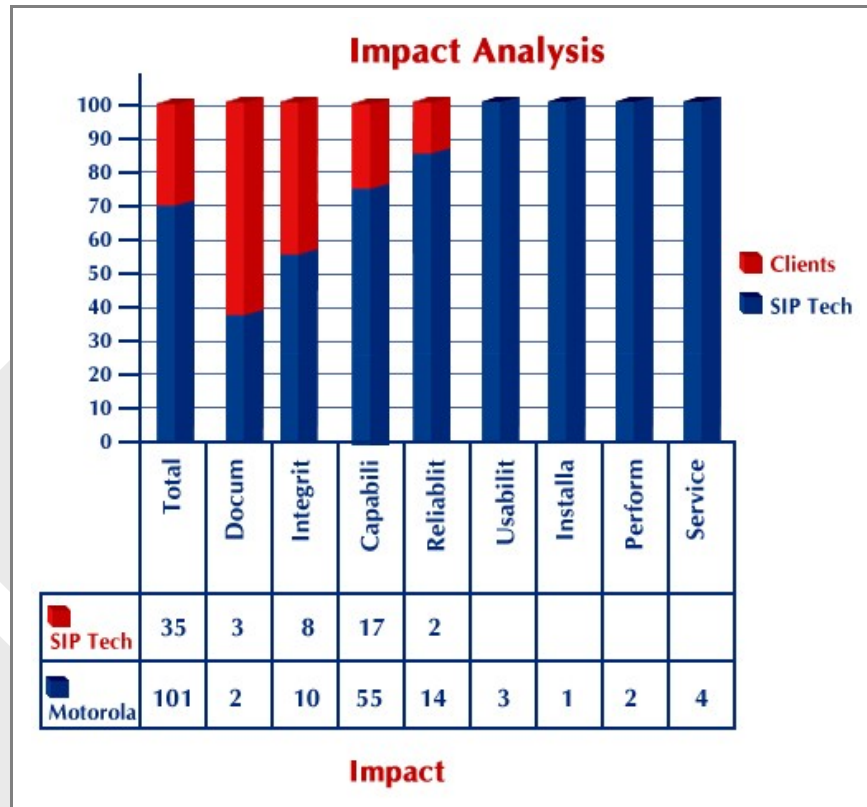
**Average**

- ❖ Sequencing

**Area Of Improvement**

- ❖ Integration
- ❖ Variation

**IMPACT ANALYSIS BREAK-UP (SIPTECH VS CLIENT)**



**Strength**

- ❖ Documentation
- ❖ Integrity/Security

#### **Average**

- ❖ Capability

#### **Area Of Improvement**

- ❖ Installability
- ❖ Performance
- ❖ Reliability
- ❖ Serviceability

## **10. Next Steps**

Take this effort forward to classify the defects as per closer section attributes but that will require the commitment from the development team of the client.

Alternatively, another initiative can be planned where the development and testing teams are from SIPTECH have better control over the experiment to help perform more detailed analysis and draw more insightful inferences.

## **11. About us**

SIPTECH provides world class software testing and quality assurance services based on proven methodologies and processes. We focus on companies that produce software products and technology platforms. With SIPTECH you will reduce your costs, improve your quality, and benefit from our independence. We care about only one thing: helping you release and maintain a high quality product.

For more information about SIPTECH and its capabilities visit the World Wide Web at <http://www.siptech.com> and email us at [info@siptech.co.in](mailto:info@siptech.co.in) and [sales@siptech.co.in](mailto:sales@siptech.co.in).

## Appendix I - Defect types

The design of the defect type attribute measures the progress of a product through the process. Defect type identifies what is corrected and can be associated with the different stages of the process. Thus, a set of defects from different stages in the process, classified according to an orthogonal set of attributes, should bear the signature of this stage in its distribution.

Moreover, changes in the distribution can meter the progress of the product through the process. The departure from the expected distribution alerts us by pointing to the stage of the process that requires attention. Thus, the defect type provides feedback on the development process.

A programmer making the correction usually chooses the defect type. The selection of defect type is implied by the eventual correction. These types are simple, in that they should be obvious to a programmer, without much room for confusion. In each case a distinction is made between something missing or something incorrect.

### DEFECT TYPE DEFINITIONS

#### Function/Class/Object

A function error is one that affects significant capability, end-user interfaces, product interfaces, interface with hardware architecture, or global datastructure(s) and should require a formal design change.

#### Assignment/Initialization

An assignment error indicates a few lines of code, such as the initialization of control blocks or data structure.

#### Interface/O-O Messages

Interface corresponds to errors in interacting with other components, modules or device drivers via macros, call statements, control blocks or parameter lists.

#### Checking

Checking addresses program logic, which has failed to properly validate data and values before they are used. Timing/serialization errors are those which are corrected by improved management of shared and real-time resources.

#### Build/package/merge

Build/package/merge describe errors that occur due to mistakes in library systems, management of changes, or version control.

### **Documentation**

Documentation errors can affect both publications and maintenance notes.

### **Algorithm**

Algorithm errors include efficiency or correctness problems that affect the task and can be fixed by (re)implementing an algorithm or local data structure without the need for requesting a design change.

## **Appendix II - Defect triggers**

The design of the defect trigger attribute (not covered in this short overview paper, but available in the original reference) provides a measure of the effectiveness of a verification stage. Defect triggers capture the circumstance that allowed the defect to surface.

The information that yields the trigger measures aspects of completeness of a verification stage. The verification stages could be the testing of code or the inspection and review of a design. These data can eventually provide feedback on the verification process. Taken together with the defect type, the cross-product of defect type and trigger provides information that can estimate the effectiveness of the process.

A defect trigger is a condition that allows a defect to surface. For instance, when a product is shipped it is assumed that all the functions and operations are tested. However, in the field a series of circumstances may allow a defect to surface that otherwise would not occur in the test environment.

It may be that the system had to get into recovery to uncover a checking defect type or a checking defect type does not occur until the software is run under a new hardware platform. Thus, although the defect type is the same, it might take different triggers to work as a catalyst for the defect to surface.

In the field, the trigger can potentially be identified by the customer engineer, or someone experienced in problem diagnosis. Thus, triggers, unlike defect types, are identified early in the life cycle of a defect.

The concept of the trigger provides insight not on the development process directly, but on the verification process. Ideally, the defect trigger distribution for field defects should be similar to the

defect trigger distribution found during system test. If there is a significant discrepancy between the two distributions, it identifies potential holes in the system test environment.

This is particularly useful when a product is sent out to an early ship customer prior to general availability. The difference in trigger distribution between early ship and system test could be used to enhance the test plans in order to cut the potential field defect exposure.

### **Trigger definitions**

Triggers are defined for the three primary verification activities: software review and inspection function test and system test.

### **Review and Inspection Triggers**

Triggers associated with design review and code inspection activities during development deal primarily with requirements and design. The focus of design review is on ensuring that the functional requirements for the product are complete, understood, and incorporated into the design. Code inspection is undertaken in order to ensure that the design has been interpreted accurately, and coded correctly.

### **Backward Compatibility**

Backward Compatibility is an area of great concern. As customers migrate to the most recent products, they take with them a body of applications that were used successfully in the previous environment. Customers want these applications to run successfully in the new environment with minimal, if any, required changes.

### **Lateral Compatibility**

Lateral Compatibility addresses another critical requirement of customers, that is, that products are able to function with other products of the same generation. Lateral compatibility becomes an increasing challenge throughout later periods of a product's life cycle as more and more products are introduced. Thus, a particular aspect of this area is anticipating accurately the interface requirements of products, which do not yet exist.

### **Design Conformance**

Design Conformance related faults are targeted by reviewers or inspectors who attempt to ensure that the interpretation of requirements, structure, or logic matches the corresponding document. These triggers are manifested in the field by customer reported problems which describe situations where the product did not function in a manner that meets the customer's requirements or expectations.

### **Concurrency**

Concurrency is one of three triggers, which are associated with the importance of understanding the details beneath the overall design. This trigger applies to simultaneous use of the same resources, and has implications of security, locking mechanisms, and sometimes performance. For customers, these often appear as long, indefinite waits for a command or function to complete.

### **Logic Flow**

Logic Flow is the second of these triggers, and relates to those occasions when the operational semantics are in question. From a customer viewpoint, the result of a fault in this area is often manifested when a command or program completes, but the returned information is inconsistent or incorrect, but there are many other possibilities as well.

### **Side Effects**

Side Effects is the third trigger which is best tackled by an experienced developer, well versed in not only the details of the product under development, but also aware of potential impacts on another function or product. When these faults surface in the field, they are characterized by seemingly unrelated symptoms, often difficult to diagnose.

### **Documentation**

Documentation is a trigger associated with both the internal information, such as prologues and code comments, and external such as user guides and installation manuals. Customers rely heavily on accompanying documentation, whether hard copy or on-line, for information about the abilities of the function or product, how to invoke the capabilities, and the results which can be expected. The extent to which the logic flow and interfaces are documented in the code has a significant influence on the prevention of future faults, as well as impacting the degree to which a reported fault can be diagnosed and corrected quickly.

### **Rare Situation**

Rare situation triggers are also best tackled by experienced developers. They are, by definition, unusual sets of circumstances, for the majority of customers. It is possible, however, that a rare situation for many is a common problem for a particular set of customers. Thus, understanding and interpreting these triggers could become an important element in accommodating particular environments.

### **Function Test Triggers**

There are several terms used to describe the testing of the functional aspect of a product. Depending on the size and scope of the project, any or all could be applied under the heading of function test. Unit test, for example, is an effort to validate the ability of the code written to execute successfully, independently from other influences such as interfaces with other products or functions. Function test takes a broader view, ensuring not only that the function executes

successfully, but that interfaces are handled correctly, and that the function provides expected results. Component test is a term applicable to a large product which consists of multiple elements (components). This additional 'function test' ensures that all of the functions within a component perform satisfactorily, and the components of a product interface correctly with each other.

Each of these types of test effort would have a characteristic trigger signature, for example, the triggers most often associated with Unit test would be those at the simple end of the scale, while those associated with Component test would be predominantly more complex. One consideration is that, in general, the more complex the test scenario, the greater it's ability to test not only the code, but the underlying design. Whether these various approaches are appropriate in all cases, or are combined in an organization, is irrelevant to the topic at hand, however, and therefore we have chosen to combine all of these activities under one heading, 'function test'.

#### **Test Coverage**

Test Coverage refers to the use of an obvious test case of an external function. From a customer viewpoint, this would correspond to entering a single command, with a single set of parameter values.

#### **Test Sequencing**

Test Sequencing is another relatively simple scenario, which examines the sequential effect of executing more than one body of code. This trigger reflects a customer scenario in which a simple series of commands is entered, such as 'create file', 'open file', 'close file'.

#### **Test Interaction**

Test interaction begins to introduce more complex conditions, with multiple commands and multiple parameters interacting with each other. Customer environments in which databases are shared among multiple end users, and each command is dependent to a large degree on the successful execution, stored information, and interfaces of other functions, will often result in the surfacing of these types of faults.

#### **Test Variation**

Test variation is another trigger, which requires more complex scenarios. It involves an attempt to exercise a single function in many different ways by entering a combination of parameters and values, sometimes invalid ones. These faults often show up as situations in which the customer inadvertently enters a parameter incorrectly, and instead of handling the situation and returning a meaningful message to the customer, the program abends.

#### **Simple Path and Combination Path**

Simple path and variation path are two additional triggers associated with function test, which require some knowledge of how the code is designed and executed, as well as the logic flow. While this is not purely an external viewpoint, many customers apply some knowledge of these internals to the task of using the function in ways, which are of particular interest to them, sometimes in an unexpected manner. The distinction between the two triggers is one of complexity: Simple path is a straightforward attempt to force specific branches to be executed, while complex path attempts to exercise these branches under varying conditions.

### **System Test Triggers**

System test activities deal with system wide and cross-system issues, including hardware and software environment implications as well as cyclic and sometimes demanding workload volumes.

### **Recovery**

Recovery corresponds with validating the product's ability to recover from error conditions. Customers prefer that failures never occur. If a fault does become a failure, however, it is highly desirable that the system or product is able to recover from the defect and carry on with as little impact to the customer as possible. One of the tasks of system test is to inject faults in order to evaluate the system or product's ability to recover gracefully.

### **Start (initialize) and Restart**

Start and Restart trigger can be related to Recovery, since they are associated with starting or restarting a system following an earlier shutdown or complete system or product failure. It may have appeared that the system or product recovered successfully from a failure, only to find later that critical information was lost or control areas overlaid.

### **Workload Volume/Stress**

Workload Volume/Stress trigger is exemplified by the ability (or inability) of a system or product to operate effectively and without failure at or near a resource limit. The mechanisms which system test uses to emulate to these scenarios could include heavy network traffic, many users, or a large number of system transactions, among others.

### **Hardware Configuration and Software Configuration**

Hardware Configuration and Software Configuration are triggers which cause failures to surface that are related to unusual or unanticipated configurations. The industry is volatile with regard to software and hardware products. The need for these products to interact and interface successfully is of utmost concern. Attempting to ensure this is an important system test activity.

### **Normal Mode**

Normal mode is a trigger, which says, in effect, that no special conditions needed to exist in order for the failure to surface. There is strong evidence to suggest that the real trigger is, in fact, associated with an earlier activity. When classifying field reported failures, if the environment description sounded like 'Normal Mode' there would almost always clearly be an associated inspection Trigger. Aside from the broader implications, there is one very specific application of this phenomenon. A product, against which a high percentage of defects classified as Normal Mode surfaces during system test, would most certainly benefit from revisiting earlier activities before it is released to the public.