Orthogonal Defect Classification v 5.2
for
Software Design and Code

September 12, 2013

1. INTRODUCTION

This document gives the details related to the actual steps involved in the classification of defects using ODC. When the defects are collected and analyzed in-process during an ongoing software development, information on defects is available at two specific points in time. When a defect is opened, the circumstances leading to the exposure of a defect and the likely impact to the user are typically known. When a defect is closed after the fix is applied, the exact nature of the defect and the scope of the fix are known. ODC categories capture the semantics of a defect from these two perspectives. By defining the Activities during a development process and their mapping to the ODC Triggers, an organization customizes the generic scheme to the local process..

2. Overview of the ODC Schema for analyzing defects related to Design and Code

Three attributes (Activity, Trigger and Impact) are collected when the defect is opened and five attributes (Target, Defect Type, Qualifier, Age and Source) are collected when the defect is closed. The attributes and their values are defined in the rest of this document with examples.

Opener Section (These attributes are usually available when the defect is opened.)			**Closer Section** (These attributes are usually available when the defect is fixed.)				
Defect Removal Activities	*Triggers*	*Impact*	*Target*	*Defect Type*	*Qualifier*	*Age*	*Source*
Design Rev, Code Inspection, Unit test, Function Test, System Test.	• Design Conformance • Logic/Flow • Backward Compatibility • Lateral Compatibility • Concurrency • Internal Document • Language Dependency • Side Effect • Rare Situations • Simple Path • Complex Path • Test Coverage • Test Variation • Test Sequencing • Test Interaction • Workload/Stress • Recovery/Exception • Start up/Restart • Hardware Configuration • Software Configuration • Blocked Test (Previously Normal Mode)	• Installability • Serviceability • Standards • Integrity/Security • Migration • Reliability • Performance • Documentation • Requirements • Maintenance • Usability • Accessibility • Capability	Design/ Code	• Assignment /Initialization • Checking • Algorithm/Method • Function/Class/Object • Timing/Serialization • Interface/O-O Messages • Relationship	• Missing • Incorrect • Extraneous	• Base • New • Rewritten • ReFixed	• Developed In-House • Reused From Library • Outsourced • Ported

2.1 Activity to Trigger Mapping

The table below gives a generic example of an Activity to Trigger mapping and is not intended to be the actual mapping used by all organizations. One of the first things an organization must do once they have decided to implement ODC is to define the activities they perform and map the triggers to those activities. **Note that although the organization defines their activities, they do not define or redefine the triggers. The only triggers to choose from are those listed above**.

Activity	Triggers
Design review	• Design Conformance • Logic/Flow • Backward Compatibility • Lateral Compatibility • Concurrency • Internal Document • Language Dependency • Side Effect • Rare Situations
Code Inspection	• Design Conformance • Logic/Flow • Backward Compatibility • Lateral Compatibility • Concurrency • Internal Document • Language Dependency • Side Effect • Rare Situations
Unit Test	• Simple Path • Complex Path
Function Test	• Test Coverage • Test Variation • Test Sequencing • Test Interaction
System Test	• Workload/Stress • Recovery/Exception • Start up/Restart • Hardware Configuration • Software Configuration • Blocked Test (Previously called Normal Mode)

3. Opener Section: (These are attributes you can classify when you find a defect.)

3.1 Activity

This is the actual activity that was being performed at the time the defect was discovered. For example, during function test phase, you might decide to do a code inspection. The phase would be function test but the activity is code inspection.

3.1.1 Design Review

You are reviewing design or comparing the documented design against known requirements.

Examples:

1) All background colors should be blue

2) Requirement specifies support for top six printers, but one popular printer was omitted in the design document.

3.1.2 Code Inspection

You are examining code or comparing code against the documented design.

Examples:

1) The code has only 3 choices for the city but design document lists 4.

2) Design standard calls for checking all return code values, but they have not been coded.

3.1.3 Unit Test

White box testing or execution based on detailed knowledge of the code internals.

Examples:

A specific path is expected, but a trace shows a different path was taken.

3.1.4 Function Test

Black box execution based on external specifications of functionality.

Examples:

1) I clicked on print and nothing happened.

2) When I selected "save", the data was lost.

3.1.5 System Test

Testing or execution of the complete system, in the real environment, requiring all resources

Examples:

1) When attempting to print 1000 jobs, the system hung.

2) When trying to read data from the word processor, got a message saying "unknown format".

3.1.6 Classifying Field Defects:

ODC can be used to classify customer reported defects as well as in-process discoveries, with two adaptations, regarding Activity and Trigger. Customer reported defects which share a database with in-process discoveries, should be identifiable as such. One such suggestion, for example, is to specify under a category such as "Phase Found", a value of

"Field", "Customer", or "BETA". In terms of ODC, where the ability to associate customer reported defects with specific in-process activities is key, the definition of activity should not refer to the task which was being done by the customer at the time the defect was uncovered, but rather, the in-process activity which was most likely to have caught the defect (but didn't). In order to minimize subjectivity, we recommend the following steps be followed:

1) Select (from the entire list) the trigger which most closely matches the environment, special conditions, or catalyst which was required for the defect to surface.

2) Using the mapping of activity to trigger which was defined internally, select the internal activity which has primary responsibility for targeting the selected trigger, and capture this in a field. This step could be automated, based on the internal mapping.

3.2 Triggers:

A trigger represents the environment or condition that had to exist for the defect to surface. What is needed to reproduce the defect? During Review and Inspection activities, choose the selection which best describes what you were thinking about when you discovered the defect. For other defects, match the description with the environment or condition which was the catalyst for the failure.

3.2.1 Design Review/Code Inspection Triggers

3.2.1.1 Design Conformance

The document reviewer or the code inspector detects the defect while comparing the design element or code segment being inspected with its specification in the preceding stage(s). This would include design documents, code, development practices and standards, or to ensure design requirements aren't missing or ambiguous.

Examples:

1) The code didn't implement the case when no data exists.

2) On one screen, all fonts are in bold while in another they are not.

3.2.1.2 Logic/Flow

The inspector uses knowledge of basic programming practices and standards to examine the flow of logic or data to ensure they are correct and complete.

Examples:

1) Memory in this subroutine needs to be allocated before storing these values.

2) A value assignment is spelled incorrectly.

3.2.1.3 Backward Compatibility

The inspector uses extensive product/component experience to identify an incompatibility between the function described by the design document or the code, and that of earlier versions of the same product or component. From a field perspective, the customer's application, which ran successfully on the prior release, fails on the current release.

Examples:

1) The install option in the current product hard codes the drive for the install whereas the previous version just used the current drive.

2) The previous version defaulted to 0 decimal places for numeric data but the current defaults to 2 decimal places.

3.2.1.4 Internal Document

There is incorrect information, inconsistency, or incompleteness within internal documentation. Prologues and code comments represent some examples of documentation which would fall under this category.

Examples:

1) The prologue for subroutine A lists only 3 parameters when there are actually 4.

3.2.1.5 Lateral Compatibility

The inspector with broad-based experience, detects an incompatibility between the function described by the design document or the code, and the other systems, products, services, components, or modules with which it must interface.

Examples:

1) A graphics application was not able to read a gif file put out by some other application package it is supposed to work with.

3.2.1.6 Concurrency

The inspector is considering the serialization necessary for controlling a shared resource when the defect is discovered. This would include the serialization of multiple functions, threads, processes, or kernel contexts as well as obtaining and releasing locks.

Examples:

1) Routine A didn't release the lock before calling routine B which requests the lock.

2) This variable was incorrectly reset to 0 by routine A after routine B had already incremented it.

3.2.1.7 Internal Document

There is incorrect information, inconsistency, or incompleteness within internal documentation. Prologues, code comments, and test plans represent some examples of documentation which would fall under this category.

Examples:

1) The prologue for subroutine A lists only 3 parameters when there are actually 4.

3.2.1.8 Language Dependency

The developer detects the defect while checking the language specific details of the implementation of a component or a function. Language standards, compilation concerns, and language specific efficiencies are examples of potential areas of concern.

Examples:

1) The inspector is checking C code and sees a "=" that should be "==".

3.2.1.9 Side Effects

The inspector uses extensive experience or product knowledge to foresee some system, product, function, or component behavior which may result from the design or code under review. The side effects would be characterized as a result of common usage or configurations, but outside of the scope of the component or function with which the

design or code under review is associated.

Examples:

1) While looking at a pointer that accesses data by byte, the inspector realizes in another part of the code, the bytes could be in reverse order, so that the pointer won't be accessing them correctly.

3.2.1.10 Rare Situation

The inspector uses extensive experience or product knowledge to foresee some system behavior which is not considered or addressed by the documented design or code under review, and would typically be associated with unusual configurations or usage. Missing or incomplete error recovery would NOT, in general, be classified with a trigger of Rare Situation, but would most likely fall under Design Conformance if detected during Review/Inspection.

Examples:

1) Requirements state that a network can handle 250 machines. A bug occurs when a customer has 1000 machines. This is not workload stress because the requirements state that only 250 machines are currently able to be accommodated.

3.2.2 Unit Test Triggers

3.2.2.1 Simple Path

The test case was motivated by the knowledge of specific branches in the code and not by the external knowledge of the functionality. This trigger would not typically be selected for field reported defects, unless the customer is very knowledgeable of the code and design internals, and is specifically invoking a specific path (as is sometimes the case when the customer is a business partner or vendor).

Examples:

1) Tried executing the "default" path of a case statement but since it didn't exist, the test failed.

3.2.2.2 Complex Path

In White/Gray Box testing, the test case that found the defect was executing some contrived combinations of code paths. In other words, the tester attempted to invoke execution of several branches under several different conditions. This trigger would only be selected for field reported defects under the same circumstances as those described under Simple Path.

Examples:

1) Path failed because one part of subroutine released memory that subsequently was used in another part of that subroutine.

3.2.3 Function Test Triggers

3.2.3.1 Coverage

During Black Box testing, the test case that found the defect was a straightforward attempt to exercise code for a single function, using no parameters or a single set of parameters.

Examples:

1) The tester tried to delete a city from the database but it couldn't be deleted.

2) Every latch set gets messages 'MSGISG101141' Formatting incomplete. Code='10'. This message is issued incorrectly because an error condition does not really exist.

3.2.3.2 Variation

During Black Box testing, the test case that found the defect was a straightforward attempt to exercise code for a single function but using a variety of inputs and parameters. These might include invalid parameters, extreme values, boundary conditions, and combinations of parameters.

Examples:

1) When the tester tried to add one more character than the maximum allowable, the software hung.

2) If an assignment of an Operation to a work position is made, and on the Operation Details screen no Work Rule is specified, upon saving Details, the client application crashes.

3.2.3.3 Sequencing

During Black Box testing, the test case that found the defect executed multiple functions in a very specific sequence. This trigger is only chosen when each function executes successfully when run independently, but fails in this specific sequence. It may also be possible to execute a different sequence successfully.

Examples:

1) The test case first added a record, then deleted it, and finally tried to add it again but got the message "You cannot add a record that already exists."

2) The "+" key was pressed twice and the program crashed.

3.2.3.4 Interaction

During Black Box testing, the test case that found the defect initiated an interaction among two or more bodies of code. This trigger is only chosen when each function executes successfully when run independently, but fails in this specific combination. The interaction was more involved than a simple serial sequence of the executions.

Examples:

1) Created a UWIP of type 'Serialized Unit'. Saved the UWIP without assigning a Serial to the part. Reopened the UWIP from the UWIP fern view, went to Parts and selected 'Edit'. Now the list of available serials comes up empty. Can't assign any serial to that part anymore.

3.2.4 System Test Triggers

3.2.4.1 Workload/Stress

The system is operating at or near some resource limit, either upper or lower. These resource limits can be created by means of a variety of mechanisms, including running small or large loads, running a few or many products at a time, letting the system run for an extended period of time.

Examples:

1) When the system is idle for 10 minutes, it hangs.

2) ISGGRP00 should not hold lock for so long that it causes the rest of complex to hang. After processing a certain number of requests it should release and then re-obtain the lock in order to give other units of work, specifically ring processing, a chance to execute.

3.2.4.2 Recovery/Exception

The system is being tested with the intent of invoking an exception handler or some type of recovery code. The defect would not have surfaced if some earlier exception had not caused exception or recovery processing to be invoked. From a field perspective, this trigger would be selected if the defect is in the system's or product's ability to recover from a failure, not the failure itself.

Examples:

1) ISGQSCAN recovery routine ISGQSCNR converts unexpected error in ISGQSCAN to ABEND09A and RCA220, instead of ABEND322.

2) After an abend, a DISPLAY GRS Contention command is entered, the MASID/MTCB issuer has JOBNAME/*UNKNOWN in the MSGISG020I output. The bit RIBESDIV has been invalidly set.

3.2.4.3 Startup/Restart

The system or subsystem was being initialized or restarted following some earlier shutdown or complete system or subsystem failure.

Examples:

1) After pulling out the plug on the CPU, the software was not able to start up again until we cleaned out some files left in one of the directories.

2) During auto restart processing, one system fails with MSGISG0151 and abend. Further attempts to restart system result in more error messages and abend dumps.

3.2.4.4 Hardware Configuration

The system is being tested to ensure functions execute correctly under specific hardware configurations.

Examples:

1) Found a defect when sending any job to a particular brand printer.

3.2.4.5 Software Configuration

The system is being tested to ensure functions execute correctly under specific software configurations.

Examples:

1)TRYJOIN does not work in a non-sysplex environment.

3.2.4.6 Blocked Test (previously called Normal Mode)

The product is operating well within resource limits and the defect surfaced while attempting to execute a system test scenario. This trigger would be used when the scenarios could not be run because there are basic problems which prevent their execution. This trigger must not be used in customer reported defects.

Examples:

1)During system test, wanted to check for workload stress by printing 1000 jobs. However, when Print was clicked, nothing happened. No screen appeared to prompt for input.

3.3 IMPACT

For in-process defects, select the impact which you judge the defect would have had upon the customer if it had escaped to the field. For field reported defects, select the impact the failure had on the customer.

3.3.1 Installability

The ability of the customer to prepare and place the software in position for use. (Does not include Usability).

Examples:

1) During automated installation, got an error message saying installation failed because a file was missing.

3.3.2 Integrity/Security

The protection of systems, programs, and data from inadvertent or malicious destruction, alteration, or disclosure.

Examples:

1) Logged in as Read Only, Profiles enabled. Was able to save changes from the System Component Assignment Panel. Was also able to delete a component.

3.3.3 Performance

The speed of the software as perceived by the customer and the customer's end users, in terms of their ability to perform their tasks.

Examples:

1) Module ISGGRP00 should not hold the GRS local lock for so long that it causes the rest of the complex to hang. After processing a certain number of requests it should release and then re-obtain the lock in order to give other units of work a chance to execute.

3.3.4 Maintenance

The ease of applying preventive or corrective fixes to the software. An example would be that the fixes can not be applied due to a bad medium. Another example might be that the application of maintenance requires a great deal of manual effort, or is calling many pre- or co-requisite requisite maintenance.

Examples:

1) Fixes can not be applied due to a bad medium.

2) Maintenance requires a great deal of manual effort.

3.3.5 Serviceability

The ability to diagnose failures easily and quickly, with minimal impact to the customer

Examples:

1) The diagnostics software number error messages rather than indicating where the problem actually occurred.

3.3.6 Migration

The ease of upgrading to a current release, particularly in terms of the impact on existing customer data and operations. This would include planning for migration, where a lack of adequate documentation makes this task difficult. It would also apply in those situations where a new release of an existing product introduces changes effecting the external interfaces between the product and the customer's applications.

Examples:

1) Co-requisite information with regard to other products is not made available to customers.

2) When migrating to a new level, the customer's applications fail because the external interface has been changed to no longer accepts blanks. This backward compatibility forces the customer to rewrite 36 applications.

3.3.7 Documentation

The degree to which the publication aids provided for understanding the structure and intended uses of the software are correct and complete.

Examples:
1) MSGISG015I RCAAE78 is not documented in the system messages manual.

3.3.8 Usability

The degree to which the software and publication aids enable the product to be easily understood and conveniently employed by its end user.
Examples:
1) In some situations, the date field is not filled in.
2) When running several jobs in system test, system was flooded with messages. They scrolled by so quickly they couldn't be read.
3) In order to perform a specific migration task, the customer must enter many commands, some with parameters which contain information difficult to find and understand.

3.3.9 Standards

The degree to which the software complies with established pertinent standards.
Examples:
1) Command menu occurs on bottom of screen instead of at top which is the industry standard.
2) Protocol specifications for participating in an exchange across heterogeneous systems are not being followed.

3.3.10 Reliability

The ability of the software to consistently perform its intended function without unplanned interruption. Severe interruptions, such as ABEND and WAIT would always be considered reliability.
Examples:
1) While invoking modem software, system crashed and had to be rebooted.

3.3.11 Requirements

A customer expectation, with regard to capability, which was not known, understood, or prioritized as a requirement for the current product or release. This value should be chosen during development for additions to the plan of record. It should also be selected, after a product is made generally available, when customers report that the capability of the function or product does not meet their expectation.
Examples:
1) Customer needs the software to have the ability to take input either from the keyboard, mouse, OR flat files.

3.3.12 Accessibility

Ensuring that successful access to information and use of information technology is provided to people who have disabilities.

3.3.13 Capability

The ability of the software to perform its intended functions, and satisfy KNOWN requirements, where the customer is not impacted in any of the previous categories.
Examples:
1) On an unconditional Latch Obtain request for an SRB, the code in ISGLRTR does not check the return code from SUSPEND SPTOKEN. If there is a user or system error, this could result in the requester thinking the latch had been obtained when in fact, it hasn't.
2) When save was clicked on, nothing happened.

4. Closer Section: These are attributes you can classify when you know how the defect was fixed.

4.1 Target : Represents the high level identity of the entity that was fixed.

4.1.1 Requirements

In order to fix the defect, it was necessary to change the requirements document.

Examples:

1) Scope of function must be comparable to competitor

2) Must be able to isolate source of failure in an open environment.

3) Must provide support for new software configuration.

4.1.2 Design

In order to fix the defect, it was necessary to change the design specification document.

Examples:

1) Interface with the new DASD was not included in the design.

2) Excessive paging will result in Performance impact.

4.1.3 Code

In order to fix the defect, it was necessary to change the code.

Examples:

1) Failure to check for return codes.

2) Incorrect assignment.

4.1.4 Build/Package

In order to fix a defect uncovered in-process, changes were necessary in the driver build process, library systems, or with management of change or version control. With regard to customer reported defects, the fix involved replacement of media, packaging scripts, missing parts, etc.

Examples:

1) Part not defined to build process. Mapping macros required to assemble application programs missing from a public library.

2) Module compiled using a back-level version of a source code part.

3) Customer could not fix, due to media error.

4.1.5 Information Development

The problem is with the written description contained in user guides, installation manuals, on-line help, user messages.

Examples:

1) An installation problem is experienced because there was no user documentation of the data sets to be allocated before the installation was attempted.

4.1.6 National Language Support

The fixes were made to the implementation of the product functions in languages other than English.

Examples:

1) Message contains garbage for French only

2) Cannot print German specific characters

4.2 Defect Type: Represents the actual correction that was made.

4.2.1 Defect Type for Target=Design/Code

4.2.1.1 Assignment/Initialization

Value(s) assigned incorrectly or not assigned at all; but note that a fix involving multiple assignment corrections may be of type Algorithm.

Examples:

1) Internal variable or variable within a control block did not have correct value, or did not have any value at all.

2) Initialization of parameters

3) Resetting a variable's value.

4) The instance variable capturing a characteristic of an object (e.g., the color of a car) is omitted.

5) The instance variables that capture the state of an object are not correctly initialized.

4.2.1.2 Checking

Errors caused by missing or incorrect validation of parameters or data in conditional statements. It might be expected that a consequence of checking for a value would require additional code such as a do while loop or branch. If the missing or incorrect check is the critical error, checking would still be the type chosen.

Examples:

1) Value greater than 100 is not valid, but the check to make sure that the value was less than 100 was missing.

2) The conditional loop should have stopped on the ninth iteration. But it kept looping while the counter was <= 10.

4.2.1.3 Algorithm/Method

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. Problem in the procedure, template, or overloaded function

that describes a service offered by an object.

Examples:

1) The low-level design called for the use of an algorithm that improves throughput over the link by delaying transmission of some messages, but the implementation transmitted all messages as soon as they arrived. The algorithm that delayed transmission was missing.

2) The algorithm for searching a chain of control blocks was corrected to use a linear-linked list instead of a circular-linked list.

3) The number and/or types of parameters of a method or an operation are incorrectly specified.

4) A method or an operation is not made public in the specification of a class.

4.2.1.4 Function/Class/Object

The error should require a formal design change, as it affects significant capability, end-user interfaces, product interfaces, interface with hardware architecture, or global data structure(s); The error occurred when implementing the state and capabilities of a real or an abstract entity.

Examples:

1) A database did not include a field for street address, although the requirements specified it.

2) A database included a field for postal zip code, but it was too small to contain international postal codes as specified in the requirements.

3) A C++ or SmallTalk class was omitted during system design.

4.2.1.5 Timing/Serialization

Necessary serialization of shared resource was missing, the wrong resource was serialized, or the wrong serialization technique was employed.

Examples:

1) Serialization is missing when making updates to a shared control block.

2) A hierarchical locking scheme is in use, but the defective code failed to acquire the locks in the prescribed sequence.

4.2.1.6 Interface/O-O Messages

Communication problems between:

1) modules

2) components

3) device drivers

4) objects

5) functions

via

1)macros

2)call statements

3)control blocks

4)parameter lists

Examples:

1) A database implements both insertion and deletion functions, but the deletion interface was not made callable.

2) The interface specifies a pointer to a number, but the implementation is expecting a pointer to a character.

3) The OO-message incorrectly specifies the name of a service.

4) The number and/or types of parameters of the OO-message do not conform with the signature of the requested service.

4.2.1.7 Relationship

Problems related to associations among procedures, data structures and objects. Such associations may be conditional.

Examples:

1) The structure of code/data in one place assumes a certain structure of code/data in another. Without appropriate consideration of their relationship, program will not execute or it executes incorrectly.

2) The inheritance relationship between two classes is missing or incorrectly specified.

3) The limit on the number of objects that may be instantiated from a given class is incorrect and causes performance degradation of the system.

4.2.2 QUALIFIER (applies to Defect Type): Qualifier captures the element of a nonexistent, wrong or irrelevant implementation.

4.2.2.1 Missing

The defect was due to an omission (e.g.) an assignment statement was missing.

4.2.2.2 Incorrect

The defect was due to a commission (e.g.) a checking statement used the incorrect values.

4.2.2.3 Extraneous

The defect was due to something not relevant or pertinent to the document or code (e.g.) there is a section of the design document which is not pertinent to the current product and should be removed.

4.2.3 SOURCE: Choose the selection which best defines the fixed Requirements/Design/Code in terms of its developmental history.

4.2.3.1 Developed In-House

The defect was found in an area that was developed in-house by the organization's own development team.

4.2.3.2 Reused From Library
A defect is encountered using a part of a standard reuse library. The problem could be that the reused part was incorrectly used or that there is a problem within the reused part itself.

4.2.3.3 Outsourced
A defect is in a part provided by a vendor.

4.2.3.4 Ported
The defect has to do with the use of a part that was validated for a different environment.

4.2.4 AGE

4.2.4.1 Base
The defect is in part of the product which has not been modified by the current project and is not part of a standard reuse library. The defect was not injected by the current project, and was therefore a latent defect.

4.2.4.2 New
The defect is in a function which was created by and for the current project and which introduces new function.

4.2.4.3 Rewritten
The defect was introduced as a direct result of redesign and/or rewrite of old function in an attempt to improve its design or quality.

4.2.4.4 ReFixed
The defect was introduced by the solution provided to fix a previous defect.

5. Usage Guidelines

5.1 General

Typically, when a defect is opened, the way in which the defect was exposed, as well as the impact to the customer, are known. Therefore, the ODC attributes of Activity, Trigger, and Impact can be classified. In some cases, it is possible that not all this information is known when the defect is opened and the user may therefore select Unknown for the value of that attribute. However, that data must then be filled in when that information is known. Similarly, when a defect is diagnosed and fixed, the details of the fix are typically known. At this time, the ODC attributes of Defect Target, Defect Type, Qualifier, Source , and Age can be classified. Additional non-ODC attributes (e.g. phase-found, severity, open date, component, etc...) that are captured in any defect tracking system can be used in conjunction with ODC-based analysis. Information on components, line items, subsystems, department, or team to represent the granularity used locally is a necessary part of the data. ODC does not impose a specific structure, it uses the fields maintained locally regarding product, in order to organize the analysis in a manner which is most effective at producing technical actions.

5.2 Usage specific to Classifying In-Process Defects during Development and Test

The following details refer to defects found in-process. For a discussion of defects found in the field/production, please see here.

1) Select the actual activity you were doing when you found the defect. For example, the calendar phase may be system test but you were actually inspecting the code when when you found the defect. Therefore, the activity would be design review/code inspection.

2) Choose the appropriate trigger from the corresponding subset.

Phase Found:Although not an ODC attribute, "Phase Found" is included in this document for clarification purposes. This attribute captures the schedule phase (e.g. Design Review, UT, Function Test, etc.) during which the defect is found.

If the defect was uncovered via a pre-general availability internal or external customer test program (including, but not limited to, Early Ship Program, Field Test, and BETA), the phase should reflect the calendar phase in progress when the defect was uncovered. The phase chosen for customer reported defects which surface after the product was made generally available should reflect that fact (i.e. Field or Production).

Activity: When classifying in-process defects, select the Activity first. You must then select one of the triggers which is associated with that activity.

Early Ship Programs, Field Tests, Beta Tests:

When classifying customer reported defects which surface prior to releasing the product, choose from the full range of triggers which represent potential customer usage. Although these activities generally occur parallel to System Test, it is likely that customer usage will invoke triggers outside the range typically associated with System Test. See usage notes under "Usage specific to Classifying Customer Reported Defects" below for recommendations.

5.3 Usage specific to Classifying Customer Reported Defects

5.3.1 Use of Triggers When Classifying Customer Reported Defects:

In contrast to the defined process (code and test activities) to find defects, the customer exposes defects by normal usage of the product. Typically, they do not have access to the internals of the design documents and code. Consequently, the following triggers are not commonly associated with customer use:

a) Design Conformance

b) Logic/Flow

c) Concurrency

d) Internal Document

e) Language Dependency

f) Simple Path

g) Complex Path

In those cases where it is clear that the customer has uncovered a defect, not based on the execution of externals, but through inspection, conformance with product announcement information, or access to source code, these triggers are appropriate.

5.3.2 Steps for Classifying Customer Reported Defects

a) Select the trigger from the list of appropriate field choices for either code defects or documentation defects, based on what the customer was actually doing or the environment or condition that was required in order for the previously dormant defect to surface. Base your choice on the description of trigger. For example, if the customer entered a single command with no parameters, you would choose Coverage, even though the customer was in production mode, rather than test.

b) Select the Activity that would most likely have found the defect based on how you have defined your software development process. If a site or product specific mapping of trigger to activity is not provided to you, use the generic mapping provided by Research (as described in this document).

5.3.3 Note concerning Blocked Test trigger:

The trigger Blocked Test is not available for customer reported defects because the typical use of the product by a customer is not based on a specific test strategy.

6. REFERENCES

[1] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S.Moebus, B. K. Ray, and M.-Y. Wong, “Orthogonal Defect Classification: A Concept for In-Process Measurements,” IEEE Transactions on Software Engineering 18, No. 11, 943–956 (1992).

[2] K. A. Bassin, T. Kratschmer and P. Santhanam, "An Objective Approach to Evaluate Software Development", IEEE Software Magazine, November/December, Vol.15 No.6 pp. 66-74 (1998)

[3] M. Butcher, H. Munro & M. Butcher, "Improving software testing via ODC: Three case studies", IBM Systems Journal, Vol. 41, No. 1, pp.31-44 (2002).