Jon T. Huber
Customer Solution Test Engineer,
Hewlett Packard Company

jon_huber@hp.com
(208) 396-6551
Metrics, 1999

A Comparison of IBM's Orthogonal Defect Classification to Hewlett Packard's Defect Origins, Types, and Modes

1.0 Abstract

The basis of this paper is actual work experience. In the last three years, the author has worked with seven Software Development teams to help them categorize defects using Hewlett Packard's Defect Origins, Types, and Modes. More recently, the author has assisted a software testing and development organization analyze the results of defects categorized using IBM's Orthogonal Defect Classification (ODC).

2.0 Introduction

A greater amount of measurable information is available about how software is created and tested in the software defect than in almost any other place. As a result, large corporations such as Hewlett Packard and IBM have developed defect categorization models. However, few direct comparisons of these different models have taken place. This paper compares a subset of actual defects categorized using IBM's Orthogonal Defect Classification model and Hewlett Packard's Defect Origins, Types, and Modes and explores the data gathered from each model. It then compares the models in terms of process, resource, and data. Finally, recommendations will be made regarding what conditions are best suited for each model.

3.0 Purpose and Parameters of the Project

The code base that the defects were logged against represented over half a million lines of code and is the firmware source code for one of the LaserJet's most popular product lines. The LaserJet product that contains the code is currently available in the marketplace.

A Software Testing group sponsored the defect analysis project. The primary objectives were to:
- Learn from existing defects to improve defect-finding methods.
- Determine what conditions had to exist to find the defects.
- Understand what defect finding methods should be applied to code areas.
- Find defects more effectively and quickly.

The project was a "one-shot approach". This means that one 3-hour workshop was all the effort that was required by the engineers who participated. In those three hours, 22 Software Test and Development Engineers categorized 93 high priority defects using IBM's Orthogonal Defect Classification (ODC) model. The decision to use ODC, as opposed to Hewlett Packard's model, was made by the Software Testing manager. It was felt that more information applicable to Software Testing improvement would be generated using ODC as compared to Hewlett Packard's model. Due to engineer time constraints, categorization of the same defects using the Hewlett Packard model was done after the workshop.

4.0 IBM's Orthogonal Defect Classification (ODC)

IBM developed the ODC model. It contains several different levels and sub-levels[1]. At the time of the workshop, a year ago, the ODC levels were:

- **ACTIVITY** - The testing activity being performed when the defect was found.
- **TRIGGER** - The environment or condition, within the testing activity, that had to exist for the defect to surface.
- **IMPACT** - The effect the defect would have had upon the customer if it had not been found.
- **DEFECT TARGET** - Represents the identity of the work product where the fix was implemented.
- **DEFECT TYPE** - For the particular defect target(s), what was the specific problem?
- **DEFECT QUALIFIER** - Indication of whether the defect type was an omission, a commission, or extraneous.
- **DEFECT SOURCE** - In terms of developmental history, what best defines the fix that was made to Requirements/Design/Code?
- **DEFECT AGE** - In terms of the age/origin of the code, where was the defect found and fixed?

Today, the definitions of ODC Defect Type and Source have changed:

- **DEFECT TYPE** – Represents the nature of the actual correction that was made.
- **DEFECT SOURCE** – Identifies the origin of the design/code, which had the defect.

Based on the ODC definitions that existed a year ago, we decided to focus primarily on Defect Activity, Trigger, Target, and Type. Had the new definition of Source been in place at the time, use of this level might have been considered.

4.1 Applying ODC

Although a base set of "Activities" is defined in the model, each organization has the flexibility, and is encouraged, to define its own testing "Activities". We defined the following "Activities" for the purpose of the workshop:

- **Product Test** - Testing the "box" (HW, FW, SW).
- **System Test** - Testing the customer solution (HW, FW, SW, Packaging, Networking, etc.).
- **Component Group Test** - Testing groups of firmware components.
- **Component Test** - Testing a single firmware component.
- **Unit Test** - Testing an element of a firmware component.
- **Inspections/Reviews** – Formal and Informal Specification, Design, and Code reviews.

The next step in applying this part of the ODC model is to align "Triggers" under each "Activity". To do this, the organization using ODC must take an unchanging set of ODC triggers and align them to their set of organization specific "Activities". We mad the following associations for the workshop:

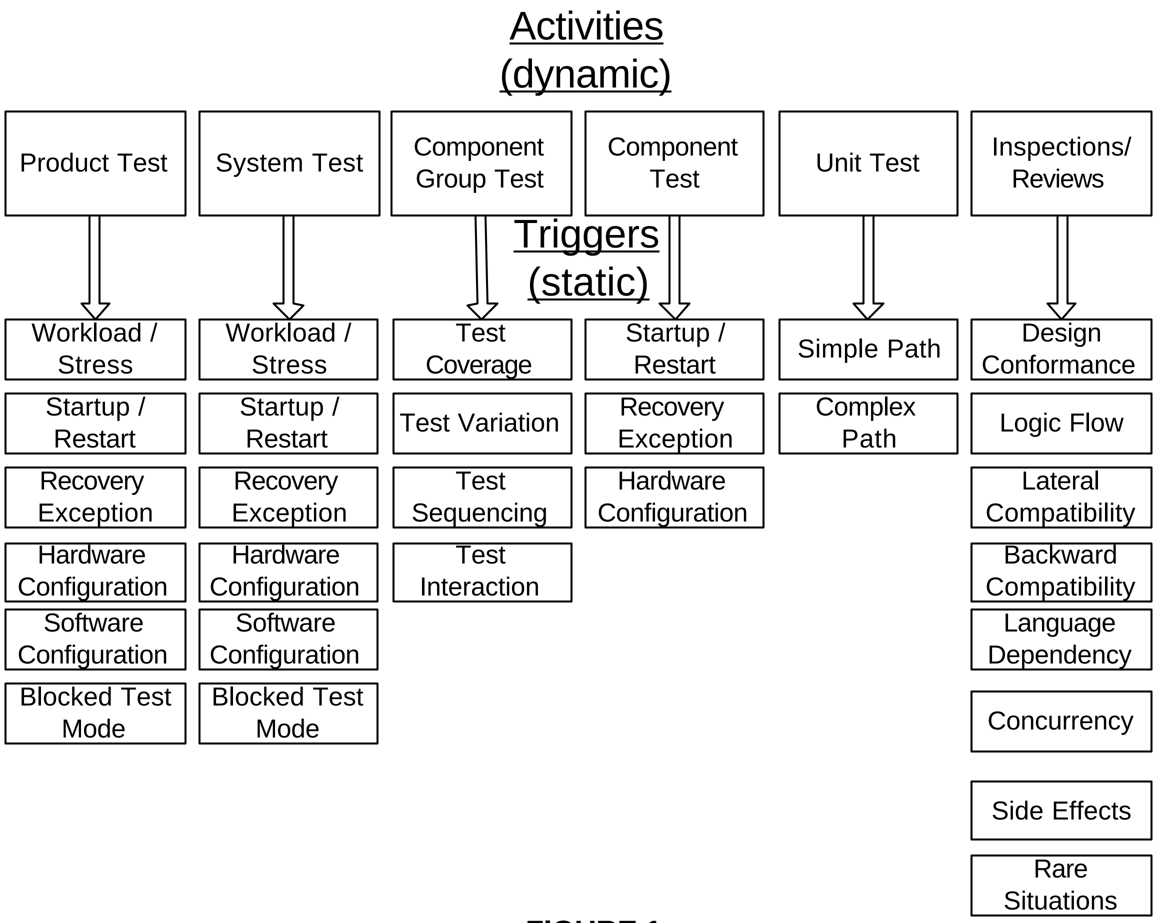

FIGURE 1

Notice that some of the Triggers are repeated under various Activities. This means that, for the organization that participated in the workshop, the condition that had to exist for the defect to surface could occur with different testing activities. It was hoped that the pairing of Defect Activity and Trigger would better help the organization understand what type of testing happened on the project.

Categorization using Defect Target and Type was more or less straightforward because organization specific categories do not need to be created. ODC Defect Targets are Requirements, Design, and Code. Possible defect Types are Algorithm/Method, Assignment/Initialization, Checking, Function/Class/Object, Interfaces/O-O Messages, Relationship, and Timing/Serialization. One difficulty in combining Target and Type is that the ODC model does not specifically tie these two levels together. Although the levels are generally tied together, in several cases there is not an alignment between the Target and the Types that applies to that Target. This determination is left to the user's discretion. As a result, the engineers who participated in the workshop came to their own conclusions regarding what Types should be grouped with particular Targets. This is especially true in the case of Requirements and Specifications as no specific Type definition refers to these work products[1].

5.0 ODC Activity/Trigger Classification Results

The results of categorizing 71 high priority defects using the ODC Activity/Trigger designations are:

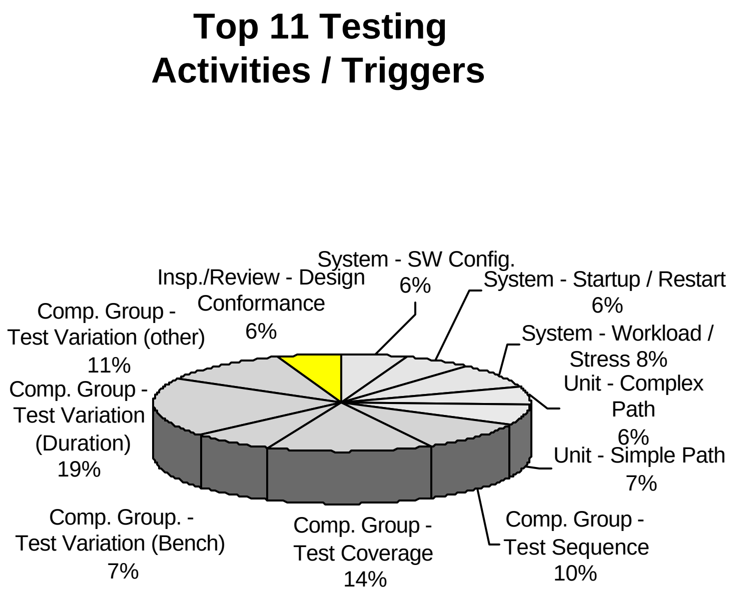

FIGURE 2

The most significant percentage of defects found by particular types of testing are Component Group Activities using various Test Variation, Coverage, and Sequencing test techniques. When presented to the team, they agreed that a significant amount of testing effort on this project was in these areas. As applied to test teams in general, the ODC Activity/Trigger analysis is beneficial in that it confirms engineering judgment in regards to what type of testing is currently finding defects. This may help teams be more open to other testing alternatives in the future. For this project, however, the ODC Activity/Trigger data did not help the team in suggesting ways they might change their approach to improve future testing.

6.0 ODC Target/Type Classification Results

The results of categorizing 93 high priority defects using the ODC Target/Type designations are:

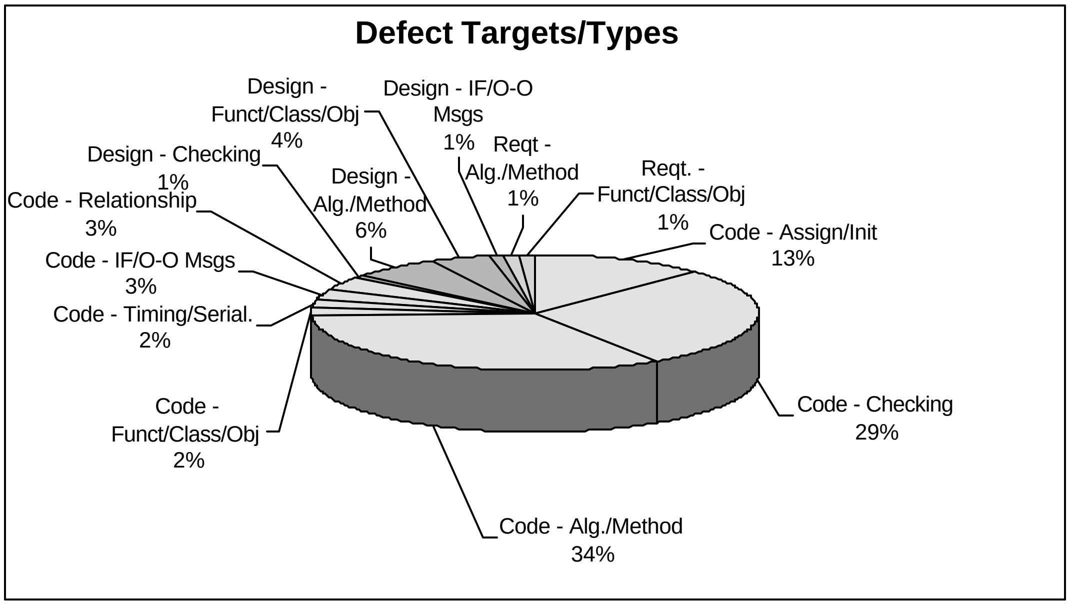

FIGURE 3

On this project, engineers were asked to use the ODC model but there was no effort made to stop them from classifying defects based on their best engineering judgment. This led them to combine defect Targets and Types although the ODC definitions and examples do not indicate an overlap between these two levels. Once again, the new definition of Defect Source was not available at the time of the workshop.

7.0 Hewlett Packard's Defect Origins and Types

The Hewlett Packard Software Metrics Council developed the following model in 1986[2].

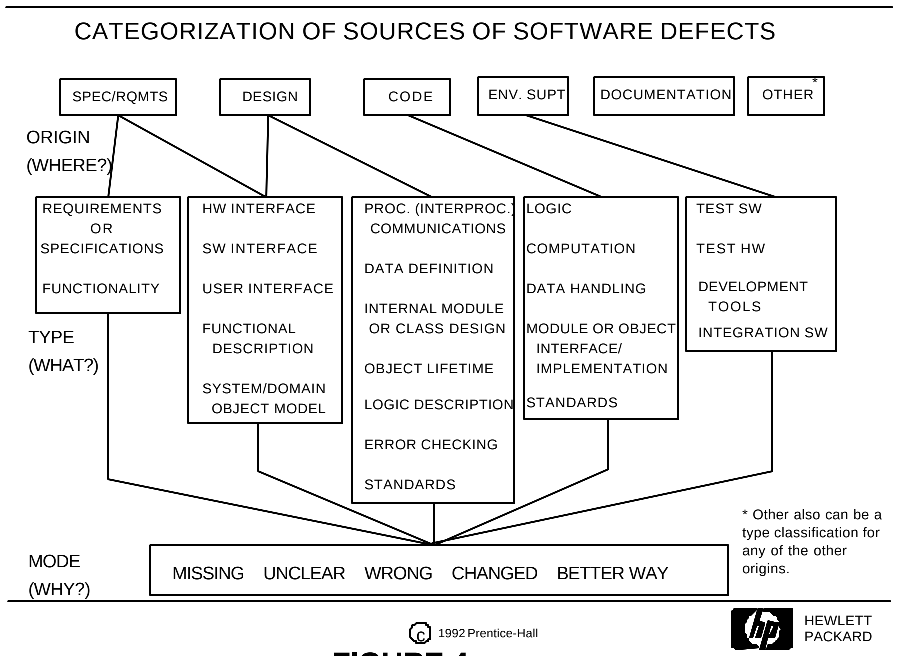

FIGURE 4

- **Origin** – The first activity in the lifecycle where the defect could have been prevented (not where it was found).
- **Type** – The area, within a particular origin, that is responsible for the defect.
- **Mode** – Designator of why the defect occurred (this level of the model is not widely used within HP).

8.0 Hewlett Packard Classification Results

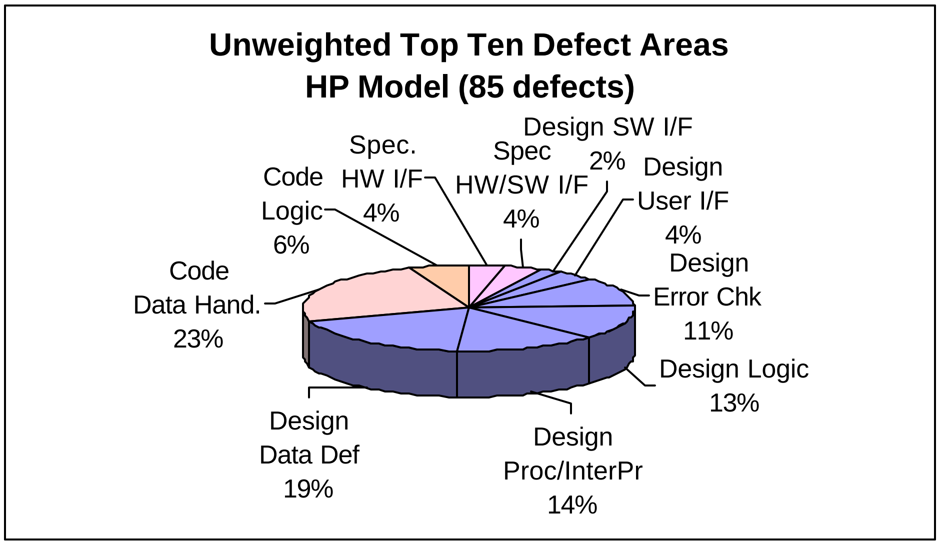

FIGURE 5

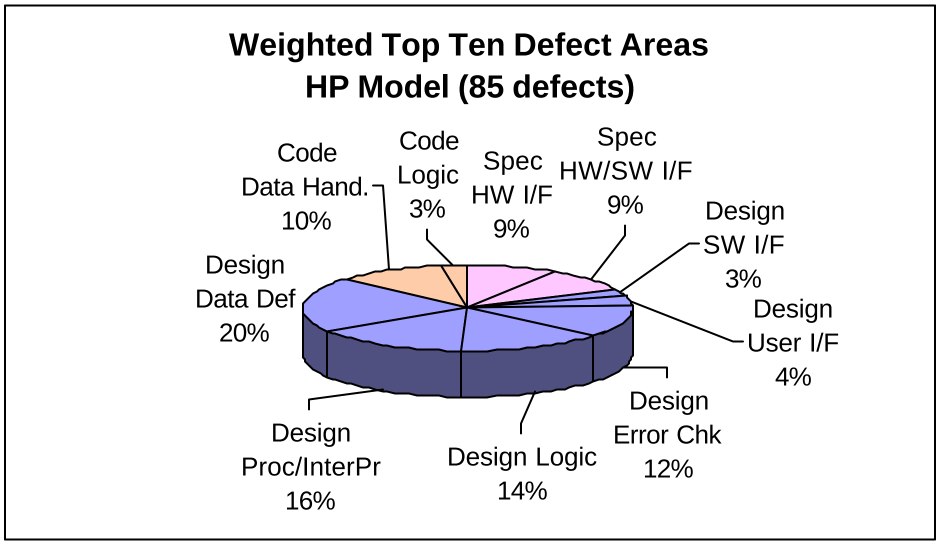

FIGURE 6

These pie charts show 85 defects from the same set as those categorized with the ODC model. The first chart (Figure 5) shows the non-weighted distribution of defects. The second chart (Figure 6) applies weighting factors, derived from industry data, to the distribution[3]. Weighting factors vary from one software development project to another. The important thing to remember is that weighting factors exist for every software development project. If a project does not agree with the weighting factors listed above, they are encouraged to adjust the weighting factors based on data gathered from their environment.

The weighting factors used are:

- Specification = 14.25
- Design = 6.25
- Code = 2.5

One of the first questions asked is why a weighted calculation is needed? The answer is that all defects are not created equal. In other words, it will cost a development team more to fix a defect whose origin is Specifications than it will to fix a defect whose origin is Code. The main reason for this is the number of work products that will need to be altered to fix a Specification defect as compared to a Code defect. For example, a developer will need to alter the specification, change the design, and potentially modify multiple sections of the code in order to fix one specification defect. On the other hand, a simple coding defect, such as a missing operator, requires a change to one work product, the source code, in one place. This cost difference in fixing particular types of defects becomes even more noticeable after the product is released. For example, if a design defect is found after release, the potential cost could be millions of dollars to redesign the product. In comparison, a simple coding error might be fixed by releasing another version of the Software to the customer.

Based on the HP categorization, 63% of the non-weighted defects were attributed to the Design Origin. This indicates that improved design techniques may lead to significant improvements for this Origin. For example, small Design improvement strategies such as the creation of data structure checklists for each type of data structure and requiring process interface documents for each process that communicates with another process can lead to significant improvements in the percentage of Design defects. Although the distribution of defects will always equal one hundred percent, improvement steps like these can help prevent defects in the front end of the lifecycle.

9.0 SW Testing Extension to the HP Model

The following model (Figure 7) was introduced at Quality Week Europe in 1998[4]. This model helps teams use defect trends from Hewlett Packard classification efforts to consider what kinds of testing might be appropriate in the future.

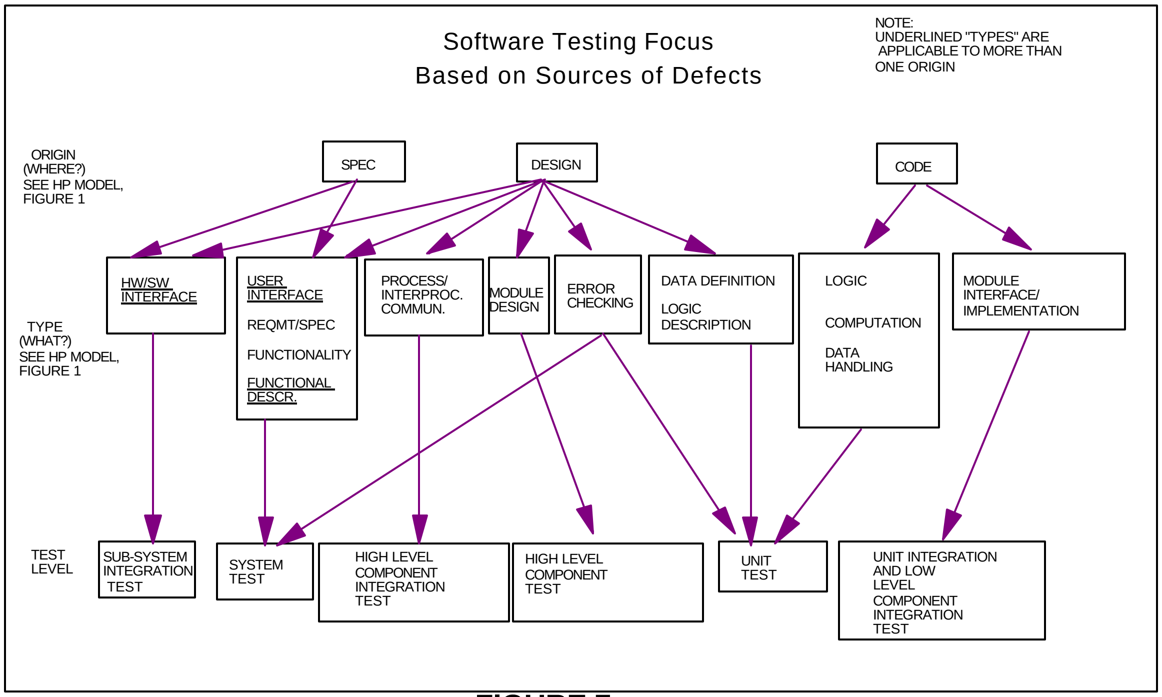
Software Testing Focus
Based on Sources of Defects
NOTE: UNDERLINED "TYPES" ARE APPLICABLE TO MORE THAN ONE ORIGIN
ORIGIN (WHERE?) SEE HP MODEL, FIGURE 1
SPEC
DESIGN
CODE
TYPE (WHAT?) SEE HP MODEL, FIGURE 1
HW/SW INTERFACE
USER INTERFACE
REQMT/SPEC
FUNCTIONALITY
FUNCTIONAL DESCR.
PROCESS/ INTERPROC. COMMUN.
MODULE DESIGN
ERROR CHECKING
DATA DEFINITION
LOGIC DESCRIPTION
LOGIC
COMPUTATION
DATA HANDLING
MODULE INTERFACE/ IMPLEMENTATION
TEST LEVEL
SUB-SYSTEM INTEGRATION TEST
SYSTEM TEST
HIGH LEVEL COMPONENT INTEGRATION TEST
HIGH LEVEL COMPONENT TEST
UNIT TEST
UNIT INTEGRATION AND LOW LEVEL COMPONENT INTEGRATION TEST

FIGURE 7

The testing focus model applied to the non-weighted HP categorization shows the following:

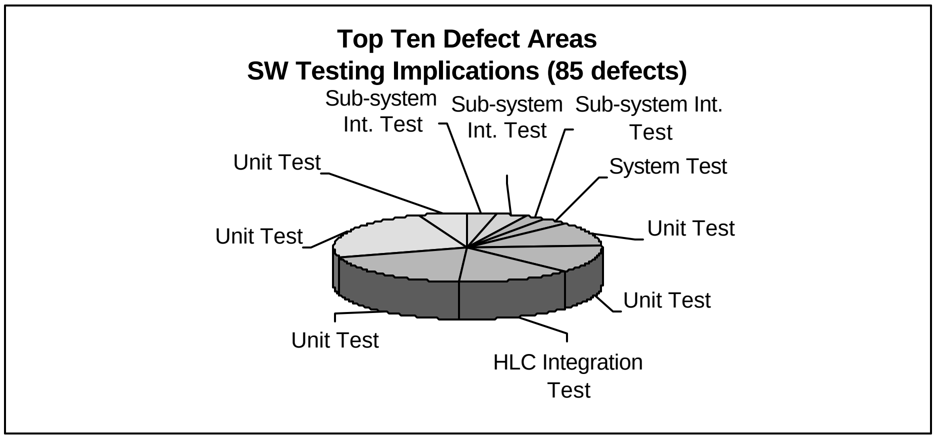

FIGURE 8

As compared to Figure 5, Figure 8 shows the same pie wedges as the HP defect categorization. For each wedge, the “Software Testing Focus” model (Figure 7) was applied. The result is a representation of what a test team would consider as future opportunity areas. It does not show what type of testing actually found particular defects. The testing team would use the “Software Testing Focus” model to generate a pie chart such as this and then analyze the defect Type definitions in the HP classification model. As a result, they would apply the suggested “Software Testing Focus” technique in the area of the work product indicated by the HP defect Type definition.

For example, applying the “Software Testing Focus” model to the Design (Origin) / Data Definition (Type) indicates a stronger focus on Unit Testing. The designation of the Data Definition Type is “Incorrect design of the data structures to be used in the module/product”. Based on this definition, the test team would examine the detailed defect reports for those defects classified as Design / Data Definition. This would help them to understand what parts of the product contains large concentrations of data structures. Tests would then be written to further exercise this code with Unit Testing. Although this technique is not the only way to determine where to focus testing, it is a way to analyze current trends to help plan for the future. More importantly, the test team can work with developers to help them develop better Unit Tests and encourage them to find defects earlier with Design and Specification Inspections and Reviews.

10.0 Comparing ODC to Hewlett Packard’s Model

10.1 Data Comparisons

10.1.1 ODC Target/Types & HP Origin/Types

This discussion compares Figure 3 to Figure 5. There is a difference between ODC's Target and HP's Origin. At the time of the workshop, Target was defined as “the identity of the work product where the fix was implemented.” Origin is defined as “The first activity in the lifecycle where the defect could have been prevented (not where it was found).” The difference between where the defect was fixed and how it could have been prevented may account for why a large percentage of ODC Targets were Code and HP Origins were Design. The actual fixes for these defects may have been in the source code but the defects should have been prevented with better design techniques.

The data in Figure 3 supports the Hewlett Packard categorization model where there is overlap of defect Types across Origins. Although ODC does not call out an overlap between Target and Type, the engineers naturally classified different Types across multiple Targets. A good example is the Algorithm/Method Type, which is represented in all Targets. In the HP model, this overlap is called out between Defect Origins and Types. For example, there is a defect Type of “User Interface” that exists for both the “Design” and “Specifications / Requirements” Origins. Although the Target/Type combination in ODC might be an appropriate place for this type of overlap, no documented correlation is made between the two levels. With the new definition of

Source in the ODC model, a new correlation between Source and Type is possible and seems very similar to Origin and Type in the HP Model.

10.1.2 HP Testing Focus & ODC Activity / Triggers

This section compares Figure 2 to Figure 8. For this project, ODC Activities and Triggers showed the engineers what kind of testing currently finds defects. On the other hand, the HP SW Testing Focus model looks at what might be done in future testing to find the defects that exist within particular Origins and Types. Based on the data, only about 13% of the Activity/Trigger combinations are attributed to Unit Testing. This data by itself probably indicates that defects found in Unit Test are not being logged consistently. However, when combined with the HP SW Testing Focus data, a different conclusion is possible. The HP SW Testing Focus model indicates that additional emphasis on Unit Testing might be appropriate when applied to particular HP Origin and Type definitions and areas of code. For this project, the data indicates that if a testing organization wants evidence to back up what type of testing is currently finding defects, ODC Activities and Triggers may provide some insight. If the organization is interested in improving the testing process, the HP models provide some options.

10.2 Process Comparison

The HP model is specifically designed to maximize the possibility for process improvement and incremental change. There are only a few levels and the definitions within the levels do not typically change (although a few Types have been added in the last decade to account for Object Oriented systems). The process to use the HP model is well documented and intuitive. In a nutshell, the process is:

- Categorize defects using the HP model.
- Choose specific, high risk, defect areas to target for process improvement.
- Engage in Root Cause analysis to determine the causes of these categories of defects.
- Generate a defect prevention plan, with action items, for major defect areas.
- Apply the SW Testing Focus model as described to determine testing focus on the next project.
- Determine action plans and assign owners.
- Execute the plans on the next project.
- At the end of the next project, categorize again and re-evaluate the defect trends.
- Document changes.

A primary focus of the HP model is that all defects are not created equally. Therefore, the Hewlett Packard process puts emphasis on defect prevention and early defect detection in the front end of development (requirements and design). The emphasis is on what can be done to improve and better engineer Software Development and Testing in the future.

Unlike the HP model, the ODC model is designed for use throughout the lifecycle of a product. On this project, it was difficult to determine a clear, consistent process for using ODC to achieve the objectives we had outlined.

10.3 Specification / Requirements Comparison

Requirements or Specifications are tailored to consider what problem the product will solve. Software Design, on the other hand, refers to what solution exists to solve the problem. For example, a simple Requirement for a LaserJet printer might be "The printer must print at twelve pages per minute." The design would then specify what features must be in the Hardware, Firmware, and Software to cause this to happen. The HP defect model clearly delineates the difference between a Specification/Requirement and a Design defect. The ODC model, on the other hand, has few references to the Specifications or Requirements. It is interesting to note that the current definition for ODC Defect Source, as listed on their web page, is "Identifies the origin of the design/code which had the defect". No mention of Specifications or Requirements is made in the definition. In digging deeper into the web page, Requirements are mentioned as one possible Source, but are not discussed in detail. Another example of the lack of Requirements or Specifications focus in the ODC model is at the Type level. The ODC Type is currently defined as "The nature of the actual correction that was made", but no Type definitions contain a reference to Requirements or Specifications. Requirements or Specifications are a necessary work product of any software system. Because of this, a defect categorization model should include buckets to handle defects occurring in Requirements or Specifications.

10.4 ODC Type and HP Type

Based on the ODC definitions that were available at the time of the workshop, one of the most logical comparisons is between ODC Type and HP Type. When the workshop was conducted the definition for ODC Type was "For the particular defect target(s), what was the specific problem?" The most recent definition for ODC Type is "The nature of the actual correction that was made."[1] The definition for HP Type is "The area, within a particular origin, that is responsible for the defect." The focus of ODC Type seems to be on the actual correction to fix the defect. The emphasis of the HP model is different and does not necessarily reflect where the correction was made to fix the defect. Rather, it focuses on what action plans are put in place that improve the process around the category of defects represented by the Type. For example, in the case of defects with a Code Origin, the actual correction to fix the defect is likely made in the source code. For the Design and Specification/Requirement Origins, the fix is more likely to improve a process that is eventually used to change the work product. Keeping in mind these differences between ODC Type and the HP Type, a further comparison is possible.

10.4.1 Types with Reference to Data

The HP Model has two clear Origin/Type combinations that are focused on problems with data, which are:
Design / Data Definition: Incorrect design of the data structures to be used in the module/product.

Code / Data Handling Problems: Initialized data incorrectly, accessed or stored data incorrectly, scaling or units of data incorrect, dimensioned data incorrectly, or scope of data incorrect.

These definitions and the application of them are very specific and straightforward. Based on experience in using the model, once an engineer is accustomed to using clear Type definitions such as those found in the HP Model, it typically takes them between 1-2 minutes to categorize a defect.

The ODC Model has several Types that mention data, which are:

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.
Checking
Errors caused by missing or incorrect validation of parameters or data in conditional statements.
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)
Relationship
Problems related to associations among procedures, data structures and objects.

Each of these definitions suggests data problems of some kind. Algorithm/Method mentions that this Type is likely located in the code. Function/Class/Object mentions that a formal design change is necessary to fix this Type of defect. Relationship and Checking make no mention of whether this Type applies to Requirement, Code, or Design. When Type is not associated to Requirements, Design, or Code it is difficult for engineers to understand and it takes more time to categorize defects.

Engineers have limited time to classify defects. It is important that critical elements of the model are clearly defined and understood so that categorization can occur as quickly as possible.

10.5 Resource Comparison

When using the ODC model it is often assumed that at least one full time resource per project is often needed to fully understand, explain, and implement the model. This resource is involved in defect analysis throughout the life of the project and is responsible for communicating and sometimes implementing needed adjustments, based on defect trends, with the development team. The Hewlett Packard model is purposely designed to have a minimal amount of resources required to implement the model. No full-time resources are needed. Often, the model is implemented as part of post-project retrospective. The resources required to implement the HP model are:

- A facilitator and a local improvement champion, who choose a set of defects, organize the workshop, and generate/report post workshop results (8-10 hours per).
- The development and/or test engineers who worked on the project and attend the workshop (3-4 hours per).

For example, if a total of ten development/test engineers worked on a project, it would cost approximately 60 engineering hours to implement the HP model. That's it!

11.0 Comparison Question and Answers – What was learned?

Although all possible comparisons could not be presented in this paper, following are few questions and answers that help compare the HP model with ODC:

- **Question:** Which model is better for me, under what conditions?
  Answer: If your goal is to examine defect trends on past products and use those trends to initiate process improvement, early defect detection, and defect prevention on future products, then the HP model and process is specifically designed to help you achieve this. ODC seems to be applied more when the primary objective is to closely examine defect trends throughout the life of the project.
- **Question:** What does each model cover that the other doesn't?
  Answer: ODC looks closely at what happened in the Code and Design of the product. In addition to Code and Design, the HP model provides clear definitions for Requirements and Specifications defects. The HP model links together Defect Origins and Types so that it is clear which Types apply to which Origins. After the initial trends are determined, the HP model looks closely at root cause analysis, using fishbone cause-effect diagrams, to determine what could be done differently to improve future processes.
- **Question:** When should each model be used?
  Answer: The ODC model can be used throughout the product lifecycle. The HP model is typically used after development and testing is complete as part of a project retrospective.
- **Question:** What are the results from using each model?
  Answer: HP model Origin/Type results shows major defect areas where process improvement should be applied to prevent defects in the future. The HP SW Testing Focus model takes the Origin/Type information and points to areas where future testing should be improved. The HP results are focused on future improvement. The ODC Activity/Trigger results showed the Test team what they already knew in regards to how they tested. The Target/Type results showed the source code areas where defect fixes actually occurred. For this project, the ODC results did not help the team with future process improvement.
- **Question:** What does it cost to use each model?
  Answer: ODC often takes a full-time resource or resources to fully implement. The HP model, on the other hand, can be implemented on a project for a cost of 60 engineering hours. This assumes one facilitator (10 hours), one product champion (10 hours), and 10 engineers (4 hours each to attend a workshop).
- **Question:** Where can I find out more?
  Answer: The ODC model has a web site[1]. This web site contains a definition of the model and experiences with ODC. Chapter 9 of the "Handbook of Software Reliability Engineering" contains a discussion of ODC[5]. The HP model is documented in "Practical Software Metrics for Project Management and Process

Improvement"[2] and "Successful Software Process Improvement"[6]. There are also many papers published on each model.

- **Question:** Are the models mutually exclusive or complementary?
  Answer: This really depends on the goals for defect analysis and which parts of the models are used. If a user is trying to improve SW Testing they might classify using ODC Activities/Triggers to get an idea about what type of testing currently finds defects. They would also categorize using the HP Model and apply the SW Testing Focus extension to understand what future testing improvements might be appropriate. Another example is when a project is just starting defect analysis. In this case, the HP model is used to generate excitement about process improvement and defect analysis because large improvements often occur with a minimum investment. After the excitement and sponsorship for defect analysis is present, applying parts of the ODC model may be appropriate at different times in the lifecycle.
- **Question:** Which model works better for whom?
  Answer: The ODC model may be more appropriate for engineers who are convinced that defect analysis will benefit their project and sponsors are willing to invest in the effort it takes to implement ODC successfully throughout their lifecycle. In a three to four-hour workshop, the HP Model can be taught and used by engineers who have never done defect analysis. This is largely due to the tailored process that, after each step, clearly suggests the next step. This makes the HP model attractive to the defect champion who wants to quickly get people new to defect analysis excited about positive change. With a relatively small investment, using the HP model, a project can get an idea about their defect trends and identify focus areas for future improvement.

12.0 Project Specific Conclusions

We live in a world of efficiency and effectiveness. These words are used so much, in fact, that they often seem overloaded. In the world of business, however, efficiency and effectiveness are essential for survival. If a product, service, or process doesn't perform to extremely high standards (effectiveness) with the least amount of resource necessary (efficiency), it will be replaced with something that does. Using the ODC model developed by IBM, the original objectives of this project were to:

- Learn from existing defects to improve defect-finding methods.
- Determine what conditions had to exist to find the defects.
- Understand what defect finding methods should be applied to code areas.
- Find defects more effectively and quickly.

Almost everyone involved in the project, especially the project sponsor, felt that the most important project objectives were not met using the ODC model. There were a few positive outcomes. Based on a post-workshop survey, 80% of the engineers like the fact that development and test engineers were in the same room analyzing the same defects. This communication between those who write and those who test code was seen as a major positive factor. It was also felt that the workshop confirmed how the test engineers knew they were testing. This was an important realization, but did not help with what should be done to improve future testing. Finally, the sponsoring manager did say she would support some form of defect analysis in the future, although not necessarily in the way it was implemented for this project. This paper provides some insights for a project considering defect analysis. Both ODC and the HP model represent options. Based on the analysis of both models as applied to this project, it is the position of the author that the HP defect model with the Software Testing Focus extension should have been used for the following reasons:

- The engineers using the process were new to defect analysis.
- The process is well established, proven successful, and documented.
- The models are centered on future process improvement and are change oriented.
- Process improvement is directed toward the front end of the lifecycle (requirements/specifications, design) where the defects cost less to find and fix.
- The implementation strategy was purposely designed to use the least amount of resources possible for analysis of defect trends.

When an organization engages in a defect analysis activity such as this one, the question “what will the people on the project do differently?” should be asked. For this project, the answer is not ideal. Based on conversations with the project’s sponsor, very little will be done differently based on the ODC results. It is felt that if the HP defect model with the Software Testing Focus extension had been used, more efficiency and effectiveness would have resulted, the customer would have been more satisfied, and the project objectives would have been realized. Although the results of this project were not ideal, it is hoped that what was learned from this paper is beneficial to those involved in defect analysis. If these individuals will pause to reflect, compare, and carefully consider the options available to them before they start defect analysis, then this paper will prove its worth.

References

1. IBM Research Group. “Details on Orthogonal Defect Classification for Design and Code”. IBM Center for Software Engineering (1999), http://www.research.ibm.com/softeng/ODC/DETODC.HTM
2. Grady, Robert B., “Practical Software Metrics for Project Management and Process Improvement”. Prentice Hall, Inc., (1992), pp. 122-137, 223-227.
3. Boehm, B., “Software Engineering Economics”. Englewood Cliffs, NJ: Prentice-Hall, Inc., (1981), p. 40.
4. Huber, Jon T., “Software Defect Analysis: Real World Testing Implications & A Simple Model for Test Process Defect Analysis”. Software Research Institute’s Quality Week Europe, November 1998.
5. Lyu, Michael R., “Handbook of Software Reliability Engineering”. McGraw-Hill Companies, Inc., (1996), pp. 367-399.

6. Grady, Robert B., "Successful Software Process Improvement". Prentice Hall, Inc., (1997), pp. 53, 64, 110, 158, 160, 189, 196, 198, 215, 236, 260, 272.

Acknowledgements

I would like to thank Fran McKain for sponsoring the project that allowed this comparison, Michele Moore who helped organize it, and Felix Silva for helping facilitate the workshop.

Thanks go to my section manager Raul Fuentes and team lead Becky Swanson. It's great to have leaders that believe people do a better job when they are given the opportunity to pursue their passion.

I would especially like to thank Bob Grady, author of three books and numerous articles, for introducing me to Software Measurement, Defect and Root Cause Analysis. His experience, comments, insights, and suggestions always prove to be accurate and important.

Jon Huber

Jon Huber has worked in the software industry for over a decade. After graduating from The University of Arizona, he went to work as a software engineer for IBM in Austin, Texas. After a few years, Jon took a software engineering job with Micron Semiconductor, Inc., in Boise, Idaho to move closer to family. At IBM and Micron, he worked primarily with object-oriented software systems.

Jon now works for Hewlett-Packard in the LaserJet Business Products division in Boise, Idaho. For more than three years at HP, he led the LaserJet Common Software Metrics Initiative which provides measurement consulting and a metrics tool for over 34 past and current LaserJet products. Jon also champions defect and root cause analysis and has facilitated eight defect analysis projects for HP. While working at Hewlett-Packard, he has authored several papers for Quality Week, Quality Week Europe, and Applications of Software Measurement/Management conferences. Jon recently has been asked to help lead a task force to implement customer solution testing in the Hewlett-Packard LaserJet Test Lab.

Outside of work, Jon enjoys family activities with his wife and four children, community service in a local youth organization, writing, jogging, and lifting weights.