Orthogonal Defect Classification

Using Defect Data to Improve Software Development

by

Norm Bridge

Motorola Corporate Software Center

Schaumburg, Illinois

Corinne Miller

Motorola GSM Products Division

Arlington Heights, Illinois

ABSTRACT - *This paper will present a framework developed by IBM for classifying and analyzing defect data collected during software development. The paper will describe Orthogonal Defect Classification (ODC) and illustrate how ODC can be used to measure development progress with respect to product quality and identify process problems. Next, the paper will present the results of a feasibility study conducted by the Motorola Corporate Software Center, Software Solutions Lab (CSC/SSL) and the Cellular Infrastructure Group, GSM Products Division's Base Station Systems (GSMBSS) software development group using ODC. Finally, GSMBSS's future plans for deploying ODC are discussed.*

KEY WORDS - *Defect Prevention, Orthogonal Defect Classification, Inspections, Process Improvement, Quantitative Process Management, Root-Cause Analysis*.

1. Introduction

Formal Inspections are currently used throughout Motorola to identify software errors near the point of insertion. In 1996 nearly 85% of all product software developers within Motorola reported that they used formal inspections on their development projects. GSMBSS has been conducting Fagan Inspections since 1992. Since then they have amassed a large amount of data on software defects. While analysis of this data has resulted in improved containment, continuing to reduce defects by 10X every two years is becoming increasingly more difficult. GSMBSS, like most software development organizations, has struggled with how to better utilize data

collected from defects to measure the progress of the product quality during development, and how to use the data to identify and eliminate process problems.

Traditionally, two distinct techniques, Statistical Defect Modeling and Casual Analysis, have been used to analyze data collected from software defects. Statistical Defect Modeling considers each defect as a random sample from an ensemble to which a statistical model is fitted. This includes statistical methods such as, counting techniques, comparisons with historical data, and reliability modeling. While Casual Analysis considers each defect as a unique occurrence and attempts to find the root-cause for each defect.

Neither of these techniques are universally applied due to the difficulties associated with each of them. Analyzing the number of defects alone is too general to be useful for process improvement, while analyzing individual defects to determine their root-cause is time consuming, expensive and does not generalize well to future releases. In addition, the analysis and feedback is normally too late to benefit the development project that created the defect. To overcome this problem with casual analysis, many software organizations are instituting end of phase post mortems. However, this is costly since improvement activities are focused on the entire development process, instead of the "hot spots." Another problem often experienced with these traditional approaches is the inconsistent classification of defects as a result of unclear and overlapping defect classifications. This makes it difficult to do reliable statistical analysis. Finally, these approaches require specially trained staff to perform the analysis.

2. Orthogonal Defect Clas-sification

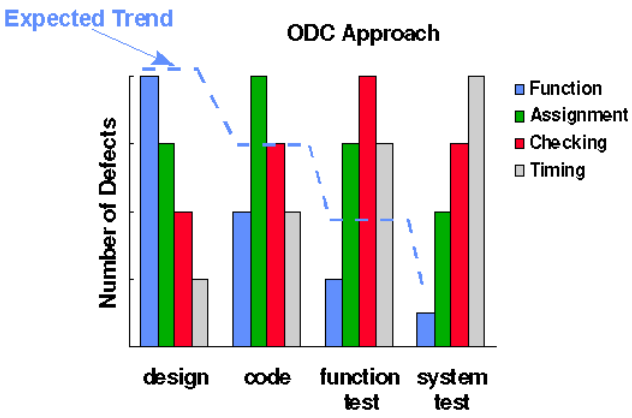
In order to model the software development process as a controllable and observable system explicit input-output or cause-effect relationships must be known. Numerous software researchers and industry experts have defined various methods for classifying software defects. Most of the methods developed to date fail to sufficiently quantify the key cause and effect relationships. It is not sufficient to collect a lot of data and hope that some subset of the data will explain the process.

A breakthrough study [5] conducted by Ram Chillarege, et. al. from IBM’s Watson Research Center showed that when defects are uniquely categorized by the semantics of the fix, it is possible to relate changes in the shape of the reliability growth curve to defects of a specific type. The IBM study showed that defects of a specific type were the result of a cause in the process that resulted in an effect on the reliability growth curve. This study forms the basis for Orthogonal Defect Classification (ODC).

ODC is a method for classifying and analyzing software defects. It bridges the gap between statistical defect modeling and casual analysis. ODC classifies each defect based upon the semantics of the defect correction and links the defect distribution to the development progress and maturity of the product. It provides an in-process measurement paradigm that extracts key properties from defects and enables measurement of cause-effect relationships as opposed to a mere taxonomy of defects for descriptive purposes.

ODC is based upon the principal that different types of defects are normally discovered during different phases of the software development life cycle and that too many defects of the wrong type discovered during a particular phase may indicate a problem. In other words, ODC uses the distribution of defect types throughout

the software development life cycle to produce a signature for the development process, see figure 1.



ODC essentially means that software defects are categorized into classes that collectively point to the part of the development process that needs attention. It is used to characterize defects and identify process problems that result in product defects in the same way as x,y,z coordinates are used to identify a point in a Cartesian system of orthogonal axis. "Orthogonal" simply means that the defect classification categories used to characterize a defect don't overlap and are statistically. independent of each other.

[1] defines the following necessary and sufficient conditions for ODC:

Necessary Condition - There exists a semantic classification of defects such that its distribution, as a function of process activities, changes as the product advances through the process.

Sufficient condition - The set of all values of defect attributes must form a spanning set over the development process.

3. ODC Defect Attributes

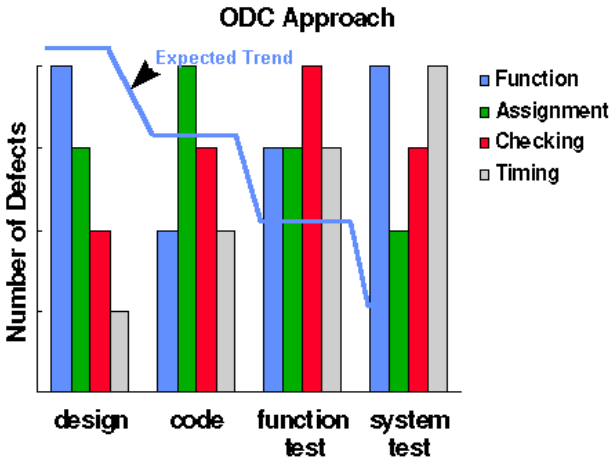
The challenge for any software defect measurement scheme is to identify a minimal set of defect attributes, in order to keep the classification simple and the overhead added to the development process minimal, while completely mapping all activities of the development process. ODC uses two attributes, *Defect Type* and *Defect Trigger*, to provide measurement instruments of the casual relationship of software defects. The Defect Type characterizes the defect based upon the nature of the change to fix the defect. It provides a measurement of the progress of the product through the development process. The Defect Trigger characterizes the defect based upon the catalyst that caused the defect to surface and result in a failure. It provides a measurement of the verification process. A third attribute, *Defect Impact*, is used to meter the impact of the defect in terms of the effect of the failure on the customer.

ODC does not imply use of only these three attributes. These attributes should be collected in addition to the traditional defect attributes such as phase found, severity, and missing, incorrect, extra. Any attribute-value set that satisfies the necessary and sufficient conditions can be considered part of ODC. Additional attributes such as the source of the defect, which characterizes the type of code corrected (new, old, reused, vendor, etc.) should be collected along with software size, team size, experience, etc., to characterize the development environment. The key is to create a measurement framework from a minimum set of attributes, that can be sliced various ways to provide visibility into the fundamental issues effecting the software development process, and that is broadly applicable and easily expandable.

3.1 Defect Type

ODC uses eight categories for defect type, *Interface*, *Function*, *Build/Package/Merge*, *Assignment*, *Documentation*, *Checking*, *Algorithm*, and *Timing/Serialization* (see Appendix I). These defect type categories defined by IBM [1] apply to any software development project, independent of the process or product. The defect type is based upon the semantics of the defect correction. The categories for defect type are independent of the software product or development process used. The categories span all software development life cycle phases, while at the same time each category is associated with a particular development activity. As illustrated by figure 1, Function defects are typically associated with requirements or high-level design inspections since they constitute the majority of defects found during these phases. While, Timing/Serialization are normally associated with system testing activities.

The relative trend from phase to phase for each defect type category indicates the progress of the product through the development process. For example, see figure 2, for a typical waterfall development process, a project may be chronologically in the function test phase, but an increase in the percentage of Function defects found relative to coding or design may indicate that the product in actuality should still be in coding or design. This provides an early indication of a process problem only one phase removed from the point in the process that needs attention. With traditional methods the problem would not be identified until integration and system testing or later.



3.2 Defect Trigger

ODC uses three sets of defect triggers, *Review and Inspection Triggers*, *Unit and Function Test Triggers*, and *System and Field Test Triggers*, (see Appendix I). For defects found after the software is released to the customer, the trigger is selected from the set of triggers for the testing activity that should have most appropriately detected the defect prior to release. The trigger for a defect is assigned based upon the testing activity that caused the defect to manifest itself as a failure. For example, if during an inspection a defect is discovered as a result of examining the design for compatibility with previous versions of the system the defect trigger would be Design Compatibility. The defect trigger provides a measure of the thoroughness of the verification process.

The defect trigger should not be confused with the symptom of the defect which is the visible effect of a defect that results in a failure. For example, if during an inspection while considering compatibility of the new software with previous versions of the system, a reviewer discovers an assignment error that would result in a particular icon not being properly displayed, the symptom is the icon not being displayed, the trigger is backward compatibility, and the defect type is assignment.

The defect trigger can be used either alone or in combination with other defect attributes. Alone, defect triggers can be used to improve the effectiveness of system testing. For example, the distribution of defect triggers from system test and field/beta testing should be similar. Any difference in the distribution indicates areas of weakness in the system testing. The system testing should be enhanced to provide additional testing using the triggers that resulted in a higher percentage of field/beta testing failures.

3.3 Defect Impact

Defect Impact provides a mechanism to relate the impact of software defects to customer satisfaction. This allows quality improvement efforts to be focused on reducing the defects that most significantly impact customer satisfaction as opposed to blindly reducing the total number of defects. The defect impact is used in addition to defect severity. Severity assesses the magnitude of the failure while impact assesses the capability of the product effected by the defect.

IBM [15] defines nine defect impact types, *Capability*, *Usability*, *Performance*, *Reliability*, *Installability*, *Maintainability*, *Documentation*, *Migration*, *Standards*, and *Integrity/Sequirity* (see Appendix I). These categories were established by IBM based upon customer surveys of what product attributes were most important to their users. Each defect is assigned an impact type based upon the effect that a failure would have on the customer had the defect escaped to the field.

4. GSMBSS ODC Feasibility Study

The GSMBSS Process Improvement Coordination Team (PIC Team) was in search of additional defect prevention and quantitative process management technologies to supplement the Fagan inspection controls already utilized. It was very important that any new technology be piloted and phased into the organization's existing process without a revolutionary upheaval. Thus, the possible synergy of ODC with GSMBSS's Fagan inspection data was especially attractive to the team.

GSMBSS Fault Code	Name	ODC Type	Requirements	Design	Code
OT	Other	ALL	X	X	X
ST	Standards	Documentation?	X	X	X
PE	Performance	Algorithm	X	X	X
CL	Clarification	Function	X	X	
TD	To much detail	Documentation?	X	X	
TR	Tracibility	Documentation?	X	X	
TY	Typo	Documentation	X	X	
PO	Product Objectives	Function	X		
UE	User Environment	Function	X		
IE	Interface Error	Interface	X		
HW	Hardware	Function	X		
SA	Software Architectur	Algorithm, Timing/Serializatio n,	X		
CS	Constraints	Checking, Timing/Serializatio	X		
EH	Error Handling	Function		X	X
ID	Internal Design Interf	Interface		X	
LO	Logic	Timing/Serializatio n, Algorithm,		X	X
MN	Maintainability	Algorithm		X	X
DA	Data Area	Algorithm		X	X
RY	Reliability	Function		X	
RQ	Requirements	Function		X	X
SI	System Interface	Interface		X	X
UY	Usability/Human Fac	Function		X	
DE	Design Error	ALL			X
RU	RegisterUsage	Assignment, Checking, Algorithm			X
PR	Prologue/Header	Assignment			X
CC	Code Comments	Documentation			X
IN	Initialization	Assignment			X
BO	Boolean Logic	Algorithm, Checking			X
PL	Program Looping	Checking			X
DU	DecisionUsage	Checking			X

In addition, GSMBSS had been performing statistical correlations between their quality metrics and customer satisfaction scores for over a year. They also had recently introduced a customer Cost Of Ownership Leader (COOL) metric to measure product attributes and services that directly impact the customer's ability to profit from using GSMBSS's products. The customer focus aspect of ODC appeared to provide a vital method to link internal defect prevention activities with the cost of ownership metric and onto external customer satisfaction. The timing was right to integrate ODC into the customer-focused defect prevention strategy if the ODC feasibility study proved successful.

The objective of the feasibility study was to determine:

1. If ODC concepts could utilize GSMBSS's existing Fagan defect classification categories.

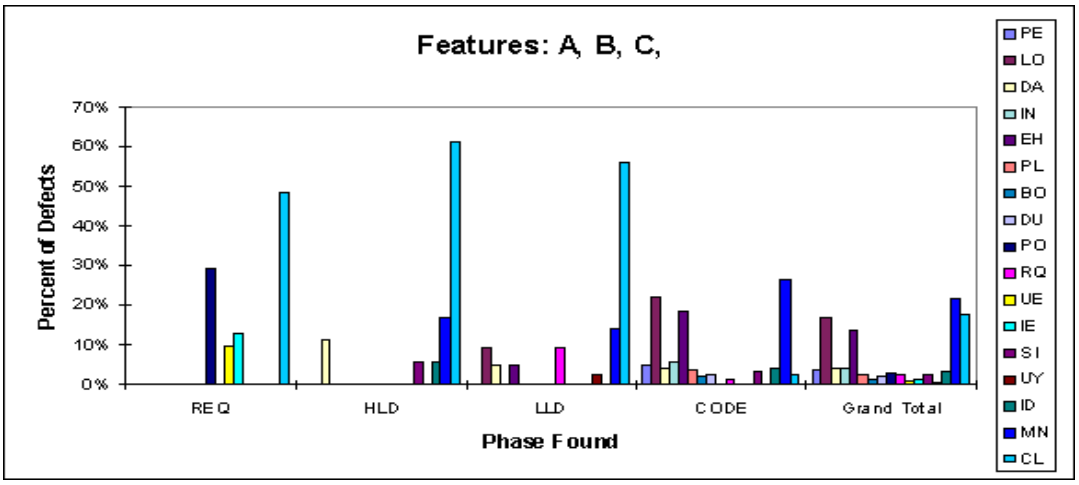
2. Confirm that ODC did in fact result in unique process signatures.

Since GSMBSS was already collecting defect data, they did not want to abandon their large database of historical defect data and introduce an entirely new defect classification scheme to their development staff. The plan was to see if it was possible to map GSMBSS's Fagan inspection fault types to the ODC defect types and determine if the resulting process signatures were similar. If this proved plausible, ODC would be considered by GSMBSS for deployment. The final decision rested with the Defect Prevention and PIC Team's review of the technology evaluation report.

4.1 ODC Feasibility Study Results

The first step of the feasibility study was to map GSMBSS's defect categories to the ODC defect categories. Table 1 shows the resulting mapping. Many of the GSMBSS defect categories satisfied ODC's sufficient condition, that the set of all defect categories must form a spanning set over the process subspace. The question was, did they also satisfy the necessary condition, that the distribution, as a function of process activities, changes as the product advances through the development process?

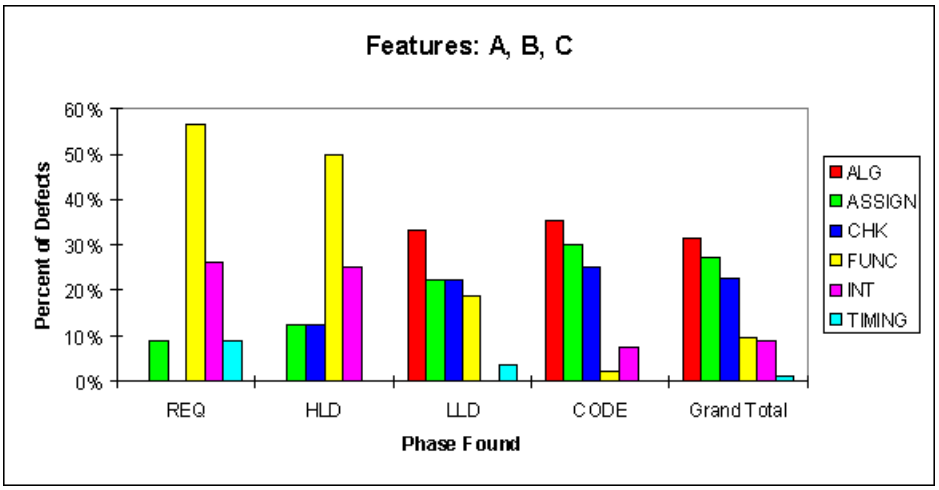
To test whether or not the GSMBSS Fagan Inspection fault types satisfied the necessary condition, the authors chose three software features from completed GSMBSS projects and classified the defects using the ODC fault types. Next we plotted a histogram of the ODC and GSMBSS defects for each development phase and compared the results. Figure 3 shows the ODC defect categories for the three features selected, while Figures 4 shows the GSMBSS fault categories for the corresponding features.



Figures 3 shows how the distribution for each ODC defect type changes as the feature passes through the

development process and how the distribution is a function of process activities. For example in figure 1, the percent of assignment type defects found increases from requirements (REQ) to high-level design (HLD) to low-level design (LLD) to code. While, the percent of interface defects found decreases from REQ to HLD to LLD to code. This is the expected distribution or signature for a typical waterfall development process.

The features used for this study were small. As a result, for an individual feature some phases only had a small number of defects of a particular type, or none at all were discovered. This caused some anomalies in the signatures for the individual features when plotted in terms of percent. For example for feature C, only three defects were found during HLD causing the trend for checking defects to decrease from HLD to LLD. Normally this would indicate a process problem, since it is logical to expect a higher percent of checking errors to be found during LLD and Code than HLD. However in this case, the small number of defects found during HLD causes the trend to be skewed. Therefore, for small projects or features it is important to look at both the percent and absolute number of defects.



Note that even though the trend or signature for each defect type was similar for each feature the type and number of defects discovered in each phase was different. This is because the number of defects is a function of the complexity and functionality of the product. GSMBSS plans to use historical defect data and product and project attributes in combination with ODC to help calibrate the magnitude of each defect type from phase to phase.

Figure 4 shows the defect data for features A, B, and C using GSMBSS's Fagan Inspection fault types. Two problems with the Fagan Inspection fault types make it difficult to identify defect trends from phase to phase. First, some of the Fagan Inspection fault types only apply during a particular development phase. Secondly, some of the Fagan Inspection fault types are not orthogonal, i.e. they are not independent and they overlap. This along with the larger number of categories, makes it difficult to identify a process defect signature in figure 4 using GSMBSS's Fagan Inspection fault types.

GSMBSS did not want to completely abandon the wealth of historical data they had compiled using their Fagan Inspection fault types. So for the features studied, the authors attempted to map the GSMBSS Fagan Inspection fault types to the single, most appropriate ODC defect type. The resulting ODC defect type distribution is shown in figure 5.

The expected process signature is more clearly visible in figure 5 than it was from using the GSMBSS Fagan

Inspection fault types, figure 4. The authors next looked at the actual mapping of the Fagan Inspection fault types to ODC defect types for the three features studied to see if some minor adjustments could be made to the GSMBSS defect types in order to more closely adhere to the ODC principles (Table 2).

Features: A, B, C,

	ODC Defect Type						
GMBSS FI Fault Code	ALG	ASSIGN	CHK	DOC	FUNC	INT	TIMING
PE	73%	18%	9%	0%	0%	0%	0%
LO	**44%**	**33%**	**20%**	**0%**	**0%**	**2%**	**0%**
DA	8%	69%	15%	0%	8%	0%	0%
IN	0%	100%	0%	0%	0%	0%	0%
EH	**43%**	**7%**	**48%**	**0%**	**0%**	**2%**	**0%**
PL	0%	0%	100%	0%	0%	0%	0%
BO	25%	0%	75%	0%	0%	0%	0%
DU	0%	0%	100%	0%	0%	0%	0%
PO	0%	11%	0%	0%	89%	0%	0%
RQ	0%	14%	0%	0%	86%	0%	0%
UE	0%	0%	0%	33%	33%	33%	0%
IE	0%	0%	0%	0%	0%	100%	0%
SI	0%	0%	0%	0%	0%	100%	0%
UY	0%	0%	0%	0%	100%	0%	0%
ID	0%	10%	0%	0%	0%	90%	0%
MN	**38%**	**30%**	**10%**	**14%**	**3%**	**0%**	**1%**
CL	**9%**	**13%**	**9%**	**52%**	**13%**	**2%**	**4%**
CC	0%	0%	0%	100%	0%	0%	0%
PR	0%	0%	0%	100%	0%	0%	0%
ST	5%	5%	0%	91%	0%	0%	0%
TY	0%	0%	0%	100%	0%	0%	0%
TD	100%	0%	0%	0%	0%	0%	0%
TR	0%	0%	0%	100%	0%	0%	0%
Grand Total	18%	16%	13%	41%	5%	5%	1%

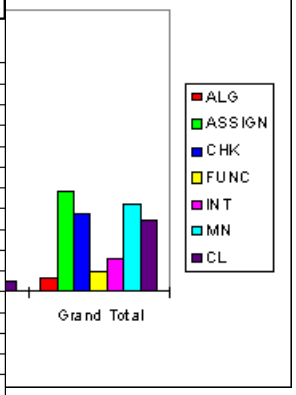


Table 2 shows that four Fagan Inspection fault types map to multiple ODC defect types. These four fault types, CL - Clarification, EH - error handling, LO - logic, MN - Maintainability, account for nearly 50% of the total defects found during development for the features investigated. We recognized that these four Fagan Inspection fault types are actually additional defect attributes that qualify the nature and trigger of the defect type. Clarification (CL) qualifies the nature of the defect and can be added as an additional enumeration to missing/wrong/extra. While, Error Handling (EH), Logic (LO), and Maintainability (MN) are defect trigger attributes which identify the catalyst that causes a defect to be discovered. Error Handling and Logic are the same as the ODC triggers Recovery/Exception and Operational Semantics respectively. While Maintainability is an additional defect trigger type not defined by IBM.

Thus, we concluded that with only a few minor modifications, GSMBSS's Fagan Inspection fault types can be mapped directly into the ODC defect types. This will allow GSMBSS's PI team to map the Fagan Inspection fault types to the ODC defect types without introducing a completely new defect classification scheme to the GSMBSS software developers. The resulting process signatures from the ODC defect types can be reviewed during development to provide an early indication of the quality of the product. While, the finer level of granularity that the Fagan defect types provide can be maintained to aid in performing root-cause analysis.

4.2 ODC Feasibility Study Findings

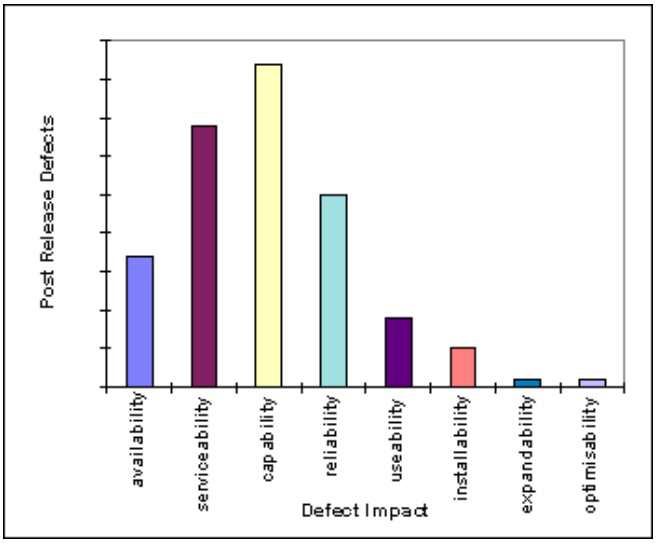
The GSMBSS ODC feasibility study successfully demonstrated that ODC does result in a defect distribution that is a function of the development process and that deviations from the expected "process signature" can be used to provide an early indication of product/process quality problems. The feasibility study also showed that with only a few minor modifications, the GSMBSS's Fagan Inspection fault types can be easily mapped directly into the ODC defect types.

4.3. GSMBSS ODC Implementation Plans

Once GSMBSS's Defect Prevention and PIC Teams were satisfied that the organization's Fagan Inspection fault types could be mapped to the ODC defect types with only a few minor modifications, they were sold on the benefits of deploying ODC. A four phased implementation plan was developed to deploy ODC, to improve defect analysis and prevention throughout GSMBSS's development life cycle.

The plan was designed to integrate the ODC methodology into the software development life cycle from the beginning and ending phases first and then work inwards. Even though GSMBSS was already doing root cause analysis of post release defects, the plan was to use ODC to quantitatively focus this analysis on the defects most impacting the customer. While at the same time, the plan also included integrating ODC into the front end of the development process to produce in-process defect signatures from inspection data. GSMBSS added ODC in-process defect signatures to their data warehouse plans as a component of feature characterization and modified their end of phase exit criteria to include a review of the in-process defect signatures.

Phase 1 of the implementation plan is currently underway. It involves three parallel efforts: The analysis of post released defects from 1996 in terms of defect type, trigger, and impact for causal analysis; use of in-process defect signatures by the feature development teams; and use of in-process defect signatures by the Systems Integration and Test teams.



A team of five experienced development engineers representing all GSMBSS product development functional areas was used to categorize all post release defects from 1996 by Defect Impact. It took them approximately six minute per defect. The results are shown in figure 6. The team used a subset of IBM's ODC impact types since the impact types cover all system problem areas, not just software. Except for a few minor changes, migration was renamed to expandability and optimisability was added, the ODC impact types defined by IBM mapped directly to GSMBSS's COOL metric.

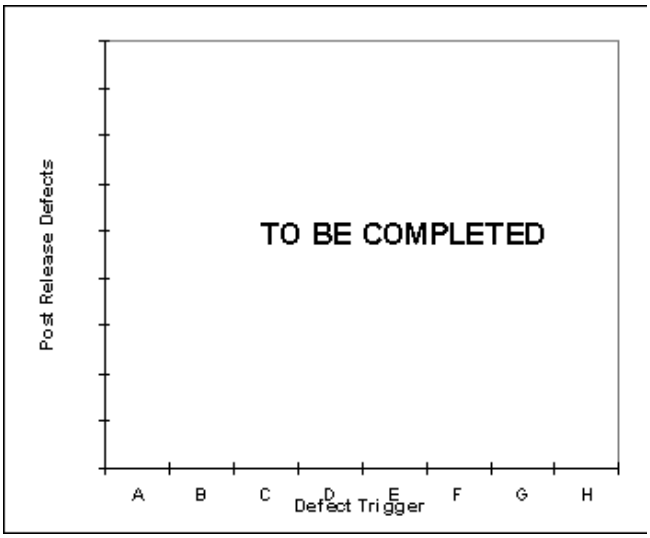
Another team made up of experienced System Integration and Test (SITG) engineers was assembled to determine the defect trigger and type for the set of post release defects and analyze the results to identify test related corrective actions. The team required less than 3 minutes per defect to assign the defect trigger. The results are shown in figure 7. The next task for the team is to determine the Defect Type for each defect. An on-line defect reporting form is currently being enhanced to collect the Impact and Trigger information in the future.

GSMBSS is currently determining appropriate Fagan Inspection fault type defect signatures commensurate with those published by IBM for ODC. The signatures will be reviewed by the leads of the various feature development tasks to direct in-process corrective actions throughout the design, code, and development test life cycle phases. The PI Team will analyze the results to determine process improvement efforts and to evolve the GSMBSS signatures over time.

The results of the combined defect signatures and Fagan inspection control charts will be reviewed by the SITG prior to the onset of system integration and testing to identify potential problem areas that may require changes to the SITG testing strategy. This may include testing higher risk features sooner or more robust testing of certain features. As escaped defects for each feature are correlated back to the offending feature, the ability to characterize the quality of similar features in the future will evolve.

5. Conclusions

This paper has presented an overview of Orthogonal Defect Classification (ODC). In it we showed how ODC can be used to provide process improvement feedback to developers. We also showed how ODC is used to measure the progress of development and to focus improvement activities on the areas that most impact the customer. The principals behind ODC were demonstrated by the results of GSMBSS's ODC feasibility study.



The authors believe that ODC can be easily applied by Motorola software development organizations striving to achieve continuous quality and customer satisfaction improvement. ODC forms an excellent foundation for the development of defect prevention and quantitative process management techniques, similar to those that have been used for years by Motorola's manufacturing operations.

6. Acknowledgments

The authors wish to acknowledge the support received from GSMBSS in conducting the feasibility study. Barbara Hirsh, GSMBSS metrics champion, was instrumental in supporting this effort. She collected and assisted with the classification and analysis of GSMBSS's inspection data for the feasibility study. Without her help this paper would not have been possible. The authors also acknowledge the strong support by Christine Ioriatti and Vernon Hamlin, who supported the study and fostered follow-on actions to make positive changes based on the results.

7. References

[1] Chillarege, R., Bhandari, I.S., Chaar, J.K., Halliday, M.J., Moebus, D.S., Ray, B.K., and Wong, M.-Y., "Orthogonal Defect Classification - A Concept for In-Process Measurements," IEEE Transactions on Software Engineering, vol. 18, no. 11, November 1992, pp.943-956.

[2] Chillarege, R., and Bassin, K., "Software Triggers as a function of time- ODC on field faults," Fifth IFIP Working Conference on Dependable Computing for Critical Applications, September 1995.

[3] Chillarege, R., "ODC for Process Measurement, Analysis and Control," Fourth International Conference on Software Quality, October 1994.

[4] Bhandari, I.S., Halliday, M.J., Tarver, E.D., Brown, D.D., Chaar, J.K., Chillarege, R., "A Case Study of Software Process Improvement During Development," IEEE Transactions on Software Engineering, vol. 19, no. 12, December 1993, pp. 1157-1170.

[5] Chillarege, R., Kao, W.-L, Condit, R.G., "Defect type and its impact on the growth curve," Proc. 13th Int. Conf. Software Engineering, 1991.

[6] Bhandari, I., Halliday, M.J., Chaar, J., Chillarege, R., Jones, K., Atkinson, J.S., Lepori-Costello, C., Jasper, P.Y., Tarver, E.D., Lewis, C.C., Yonezawa, M., "In-process improvement through defect data interpretation," IBM Systems Journal, vol. 33, No. 1, pp182-214, 1994.

[7] Bhandari, I.,Wong, M.Y., Chillarege, R., Ray, B., Choi, D., "An Inference Structure for Process Feedback: Technique and Implementation," Software Quality Journal, Vol. 3, No. 3, pp167-189, September 1994.

[8] Chillarege, R., Biyani, S., "Identifying Risk Using ODC Based Growth Models," Fifth International Symposium on Software Reliability Engineering, November 1994.

[9] Sullivan, M., Chillarege, R., "A Comparison of Software Defects in Database Management Systems and Operating Systems," 22nd International Symposium on Fault-Tolerant Computing, July 1992.

[10] Chaar, J.K., Halliday, M.J., Bhandari, I.S., Chillarege, R., "On The Evaluation of Software Inspections and Tests," International Test Conference, 1993.

[11] Halliday, M.., Bhandari, I., Chaar, J., Chillarege, R., "Experiences in Transferring a Software Process Improvement Methodology to Production Laboratories," Second International Conference on Achieving Quality in Software, October 1993.

[12] Chaar, J.K., Halliday, M.J., Bhandari, I.S., Chillarege, R., "In-Process Evaluation for Software Inspection and Test," IEEE Transactions of Software Engineering, vol. 19, No. 11, November 1993, pp. 1055-1070.

[13] Sullivan, M., Chillarege, R., "Software Defects and their Impact on System Availability - A Study of Field Failures in Operating Systems," 21st International Symposium on Fault-Tolerant Computing, June 1991.

[14] Lyu, M.R., editor, Handbook of Software Reliability Engineering, McGraw-Hill, New York, 1995, pp.359-400.

[15] Kaplan, C., Clark, R., Tang, V., Secrets of Software Quality : 40 Innovations from IBM, McGraw Hill, New York, 1994, pp.319-328.

Appendix I: ODC Definitions

Defect Impact - The Defect Impact provides a mechanism to relate the impact of software defects to customer satisfaction. For defects found prior to release, the Defect Impact captures information on the effect of a defect on the end user in the event that a defect had escaped to the field and resulted in a failure. For customer reported defects, the Defect Impact captures information on the actual customer impact.

Capability

The ability of the product/system to perform its intended functions and satisfy the customer's functional requirements.

Usability

The ease with which the product/system can be easily understood and utilized by the customer to perform its intended function.

Performance

The speed and responsiveness of the product/system as perceived by the customer.

Reliability

The ability of the product/system to consistently perform its intended functions without unplanned interruption.

Installibility

The ease with which the product/system can be easily prepared and placed into service.

Maintainability

The ease with which a failure can be diagnosed and the product/system can be upgraded to apply corrective fixes without impacting the customer's data and operations.

Documentation

The degree to which the product/system documentation and user's manuals are correct and aid in the customer's understanding and use of the product/system.

Migration

The ease and degree to which the product/system can be upgraded to the newer release without impacting the customer's data and/or operations.

Standards

The degree to which the product/system conforms to established pertinent standards.

Integrity/Security

The degree to which the product/system is protected from inadvertent or malicious destruction, modification, or disclosure.

Defect Trigger - The Defect Trigger is the catalyst that caused a defect to manifest itself as a failure. It provides a measure of the effectiveness of the verification process. Separate triggers are defined for reviews and inspections, unit and function test, and system and field test. Customer reported defects are assigned triggers from all three categories.

Triggers during Review and Inspection Activities

(Note: *Indicates may also be used for field defects.)

Design Conformance* - The error was detected while comparing the work product being inspected with the corresponding specification from the prior development phase(s).

Operational Semantics (Understanding flow) - The error was detected while considering the logic flow needed to implement the function under review.

Concurrency - The error was detected while considering the synchronization needed between tasks to control a shared resource.

Backward Compatibility* - The error was detected while considering the compatibility between the function described by the work product under review and that of prior versions of the same product. Note, this requires that the inspector have extensive product experience and familiarity with the function under review.

Lateral Compatibility* - The error was detected while considering the compatibility between the function described by the work product under review and other systems/subsystems or functions that it must interface with. Note, this requires that the inspector have broad-based knowledge of the system under development.

Rare Situation - The error was detected while considering some abnormal system behavior that is not specified by the requirements for the function under review. Note, this requires that the inspector have extensive experience and/or product knowledge.

Side Effects - The error was detected while considering some system, product, function, or behavior that may occur that is beyond the scope of the work product under review. However, the side effects would be characterized as the result of normal usage or configurations of the system. Note, this requires the inspector to have extensive experience and/or product knowledge.

Document Consistency/Completeness (Internal Document) - The error was detected while reviewing the work product for consistency/completeness and conformance to documentation standards.

Language Dependencies - The error was detected while reviewing the work product for implementation language specific details.

Triggers during Unit and Function Test Activities

(Note: *Indicates may also be used for field defects.)

Simple Path Coverage - The error was detected by using White/Gray Box testing to execute simple code paths related to a single function.

Combinatorial Path Coverage (Complex Path) - The error was detected by using White/Gray Box testing to execute combinations of code paths related to multiple functions. the test case that found the defect was executing combinations of code paths.

Test Coverage* - The error was detected by using Black Box testing to exercise a single function with either no or a single set of input parameters.

Test Variation* - The error was detected by using Black Box testing to exercise a single function using multiple sets of input parameters, such as illegal values, boundary conditions, and various combinations of parameters.

Test Sequencing* - The error was detected by using Black Box testing to execute multiple functions in a very specific sequence. This trigger applies only if the functions operate correctly when tested independently, but fail when executed in a particular sequence.

Test Interaction* - The error was detected by using Black Box testing to execute multiple functions in combination. The interaction involves more than a simple sequencing. This trigger applies only if the functions operate correctly when tested independently but fail when executed in together.

Triggers during System and Field Test Activities

(Note: *Indicates may also be used for field defects.)

Workload Volume/Stress* - The error was detected while operating the system at or near some resource limit, either upper or lower.

Normal Mode - The error was encountered under normal operating conditions without exercising any particular test suit and with the system operating well within resource constants. This trigger should only be used when system test scenarios can not be executed because of basic problems that block their execution.

Recovery/Exception* - The error was detected as a result of executing an exception hander or recovery code. The error would not have be discovered if some earlier event had not caused error handling or recovery processing to be invoked. Note, this trigger is selected only when the error is in the systems ability to recover from a failure, not the failure itself.

Startup/Restart* - The error was detected while the system/subsystem was being initialized or restarted following an earlier shutdown or complete system or subsystem failure.

Hardware Configuration* - The error was detected as a result of testing a particular hardware configuration.

Software Configuration* - The error was detected as a result of testing a particular software configuration.

Triggers for Customer Reported Failures In the case of field reported defects, select a trigger from any of the trigger categories based on what the customer was attempting to do or that best matches the condition that was the catalyst for the failure. For example, if a customer entered a single command with no parameters, the trigger would be Test Coverage. Normally only those triggers marked with an asterisk should be used for field defects.

Defect Type - is based upon the semantics of the defect correction, i.e. it is assigned based upon the description of the action taken to fix the problem. The Defect Type attributes are independent of the development process and span all live cycle phases. The Defect Type provides a measure of the products maturity.

Interface - The defect was the result of a communication problem between subsystems, modules, components, operating system, or device drivers, requiring a change, for example, to macros, call statements, control blocks, parameter lists, or shared memory.

Note: A defect that is the result of passing the wrong type of variable is an interface defect, while a defect that is the result of passing the wrong value is an assignment.

Function - The defect was the result of the omission or incorrect implementation of significant capability, end-user interfaces, product interfaces, interface with hardware architecture, or global data structure(s).

Build/Package/Merge - The defect was encountered during the system build process, and was the result of the library systems, or with management of change or version control.

Assignment - The defect was the result of a value incorrectly assigned or not assigned at all. Note that a failure correction involving multiple assignment corrections may be of type Algorithm.

Documentation - The problem was the result of an error in the written description contained in user guides, installation manuals, prologues, and code comments. Note this should not be confused with an error or omission in the requirements or design, that might be a Function or Interface defect type.

Checking - The defect is the result of the omission or incorrect validation of parameters or data in conditional statements. Note a fix involving the correction of multiple checking statements may be of type Algorithm.

Algorithm - The defect is the result of efficiency or correctness problems that affect the task and can be fixed by (re)implementing an algorithm or local data structure.

Timing/Serialization - The defect is the result of a timing error between systems/subsystems, modules, software/hardware, et. or is the result of the omission or an incorrect use of serialization for controlling access to a shared resource.