Process Metamorphosis in IBM SSD-SSD

Thomas P. Messmore
Advisory Software Engineer
IBM Tucson

Prologue:

Process change does not come easily, regardless of whether it is needed immediately or not. In most cases, change is very slow. Most new technologies take seven years to become standard practice, and some processes take even longer. In the fast-paced setting of today's computer environment, this is not acceptable. Change is inevitable. It just depends on whether you end up with good change or bad change. Likewise, a process improvement strategy is not without its highs and lows, as will be detailed in this writing. In the last two years, The Systems Storage Development - Systems Storage Division (SSD-SSD) of IBM has used a method for process change that has seen some quantifiable results toward product quality, even though the program is still in its infancy. This methodology, called Orthogonal Defect Classification, results in defect reduction and preventive actions that are quantifiable. That is, the effect a particular action has on the quality can be measured by the percentage of defects that will be reduced or eliminated by that action. This may be the first process methodology that can project improvement with certainty. This paper details what it took to implement ODC in a large programming organization, starting with the basics and evolving to coverage of 35 projects in SSD-SSD.

I. Previous Programs

Historical Perspective:

Through the years, IBM created or adopted a litany of quality programs that came ever closer to being the "silver bullet"; that is, zero defect programming. In the author's twenty six (26) years with IBM, there was exposure to quality circles, kickoff meetings at the beginning of projects, and postmortems at the end; the "team" concept, consisting of mutual goals, mission statement, and negotiating to resolution, Market Driven Quality, 10X and 100X goals, Malcolm Baldridge and Carnegie Mellon Software Process Maturity, Reuse and all it's common reuse repositories, and Defect Prevention Process. Point in fact, my job was to convince programmers that each of these in turn was the answer to process improvement. This meant developing an in-depth study of how each of these worked and the best way to implement each one. Each of the "programs" or "methods" had both good and bad points; therefore the success in implementing any other process improvement technique in our organization was to learn from those shortcomings, and build upon the strengths.

We think we have done that with ODC, but it took experimenting with and experiencing other methods before a successful implementation strategy could be found.

Putting all of these lessons learned from previous programs yields the following: Whatever method for process improvement is chosen, the investment up front must result in an easy to implement, non time-consuming program that has a realistic goal of implementing actions based on timely analysis of a selected group of problems. Thus far, ODC matches all these requirements

II. ODC - What it is and How It Works in Theory

What it is

Beside the esthetic characteristics, what is the ODC at SSD-SSD really?

ODC was developed by The Center Of Software Excellence, at The IBM Thomas J. Watson Research Center, Hawthorne, New York. It is the defect classification methodology, used for all software and microcode projects in SSD-SSD that have enough defects to make a valid assessment. It measures review and test effectiveness. ODC compares the defect escapes from stage to stage to gauge the overall process efficiency. In addition, ODC is the method to categorize and, analyze field defects to develop projections for future releases and point out what defects escaped the development and test process.

How It Works

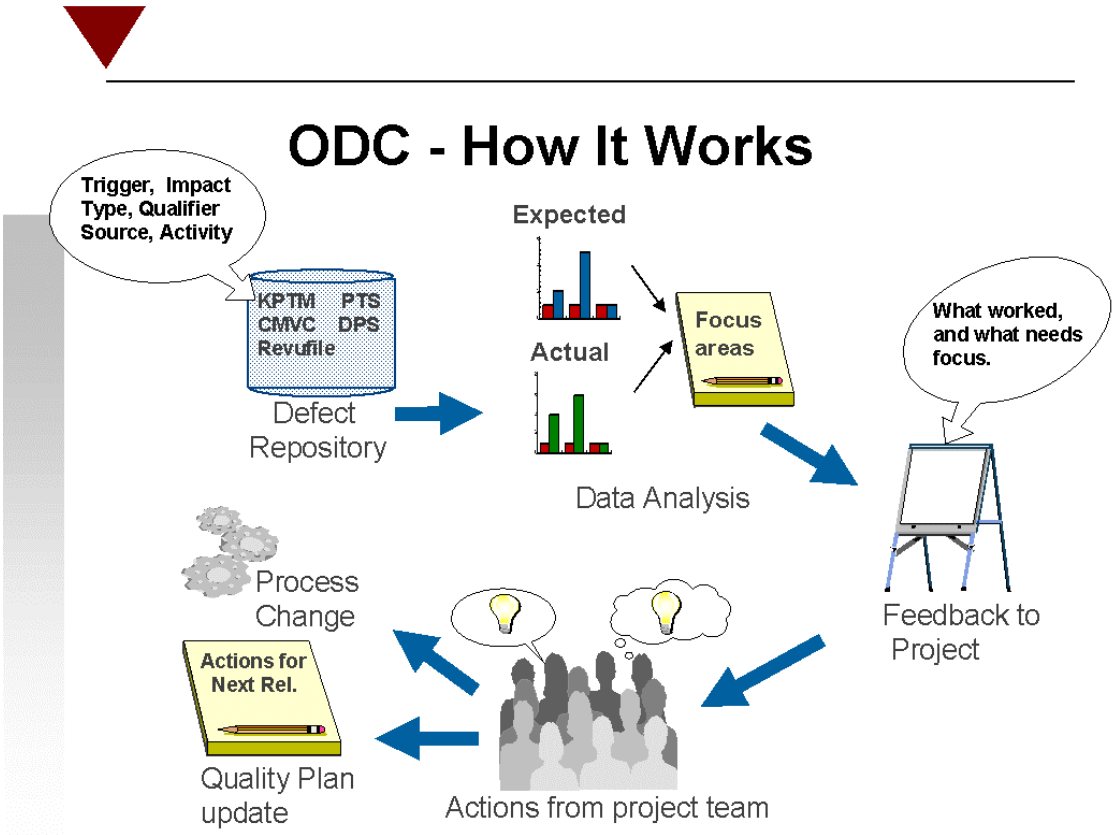


Diagram Explanation follows:

1.Defect Repository

A. Tools

You need defect data, and lots of it. To gather the data, the programmers and testers use existing defect tracking tools (CMVC, Revufile, KPTM, PTS, and DPS.) This is an important point that the tools used for gathering ODC are ones that the programming community is already familiar with. As a result, the learning curve is much less.

Some projects use a manually filled in template, but a tool that allows the data to be modified and accessible by all is the recommended method.

The tools were modified to include six additional pieces of information called classifiers for each defect record. Some of the classifiers were kept the same as those defined in the original technical papers by IBM Research. The idea is that each of the classifiers contains a list of choices that is orthogonal in nature. That is, each of the choices are mutually exclusive of one another. For any given defect and each classifier, there should be only one choice that best applies. While keeping the concept of orthogonal, the classifiers Trigger, Impact, and Source were modified to match the environment and terminology of the SSD-SSD programming organization.

B. Data

Classifiers at Open Time - The first are those classifiers that are entered by the person who discovers the problem and opens up the defect, either the reviewer at specification, design, or code stage, or, the tester at test stages. The person supplying the fix fills in the classifiers for defects discovered after the product is made available to the customer. There are three classifiers in this group:

1) Activity (also called Stage), what design or test cycle was being done when the defect occurred.
2) Trigger, which can be thought of as the type of review or test that uncovered the defect.
3) Impact, which is the effect the defect would have on the customer, regardless of what stage the defect is discovered in.

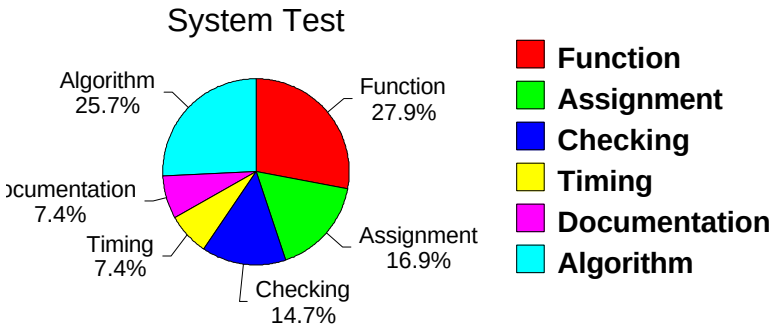
Classifiers at Close Time -The last 3 classifiers are entered by the person supplying the fix when the defect is closed. They are:

1) Defect Type, the type of code causing the problem.
2) Qualifier, whether the design or code was missing or incorrect.
3) Source, the location within the code that was corrected, such as in new code added for the release, or discovered in code that previously existed.

2. Analysis

The ODC team statistically compares actual distributions of classifiers with expected distributions and uses the differences as the areas of focus. For example, Chart 1 represents the various choices for defect type. When mapped against actual data, there is a different distribution of each of the types. Depending on the stage, these distributions can point to a potential process inefficiency. For example, if the type of 'function' was larger than expected and the stage was system test, this would indicate a problem with incomplete or missing design, because function type of defects should be found in the earlier review stages.

Chart 1.



3. Feedback to the Project

The ODC team provides feedback to the project team. This step takes the findings of the analysis and presents them to the team to show them what process steps worked well and which need review or actions to improve. See examples in section IV for the type of information that is provided to the teams.

4. Actions

The project team uses the ODC feedback to formulate quantifiable actions. They also document actions in the quality plan of the next release. Process documents may also be changed at this point. The project team creates actions that are validated by the ODC team

III. Implementation Strategy

1. Start with the Basics

The IBM Center of Software Excellence (hence called CSE) contracts with a site to provide the initial ODC education. This is most effective if the site has someone responsible for the implementation of ODC. That person is the site coordinator, who can arrange for the facilities and attendees. The education team at research provides the materials and usually requests some actual defect examples to use as practical illustrations in the presentation. The initial education is high level and is given to all members of projects that will use ODC. In addition to this overview education for all team members, CSE provides education to the focal points for the site. These are the people responsible for the implementation of ODC on a project level. They work with the site coordinator for subsequent education, materials, analyses and correction of classification data, scheduling feedback sessions, and to coordinate the development of actions.

The initial tool support was contracted by CSE with the various sites responsible for the maintenance of defect repository tools. For example, IBM Raleigh maintained the DPS tool. As a result, the tools are modified with ODC classifier definitions provided by CSE. If there are no individual site modifications of the ODC classifiers then the tools can be used as supplied.

2. Get Project Specific

The education provided by CSE for the overview is general in nature. A lot of terminology and a lot of concepts are presented. The mistake of some sites is not to add to this initial education. The terminology applies more to software than it does for microcode, and the data gathered is presented for large systems projects. Here are some hurdles the site coordinator and focal points (henceforth called the A-Team) must rectify early in establishing ODC.

Terminology: all of the classifiers for trigger, impact, type, qualifier, and source use terms that may have multiple interpretations. For example, the term "usability" as an impact in ODC is defined to be "ease of use," "human factors," "esthetics." The problem is that some programmers see the term to mean "usable," "functioning," "state of working," from previous definitions they developed for the term. Without added emphasis in the form of reference cards, examples, or tips on correct usage, misclassification is certain to occur. This means that initially the A-team should make sure that the proper choice of classifier is used. The coordinators developed a guide to help in making the right choice.

Practical examples: In addition to the terminology, the examples should be something to which members of the project can relate. Although some microcode is more like software than hardware, there is still enough of a difference that the example may not be meaningful. For example, the term "abend" or "crash" in software can be translated to "panic" for microcode. If the example does not use the word "panic," then some microcoders may not know that the example applies to them. To the extent possible, reference materials should be developed separately for software, microcode, and hardware projects.

Tool Specific: With the flexibility of adding ODC classifiers to a wide variety of tools, it may mean that due to restrictions on name length, help text capability, and panel structures, the look and feel of the definitions may vary from tool to tool. As a result, each tool should have its own reference cards, education, and tailoring. This tailoring is usually specific to the project. Some projects may want to have the ODC fields be mandatory, others may want them mandatory for certain classes of defects, such as those resulting in code hits, but not for others: integration, user error, etc. These tool changes make it easier for project members to accept the responsibility to enter the data and to do so correctly. Table 1. shows an example of some of the tool specific features for the corresponding ODC support that the A-team or the projects requested.

Table 1.

TOOL	FEATURE	MODIFICATION
PTS 2	Fields Mandatory	Special reports created
CMVC for OS/2	Trigger dependent on Activity	Field level help
CMVC for AIX	All ODC fields mandatory	Default values added
KPTM	All choices shown, accepts only one selection.	ODC panels displayed only for defects with code hits.
DPS	Invoked when APAR in tool SPA is closed	A project identifier determines if ODC required

Project Identifier: In preparation for later analysis, the focal point should work with the site coordinator to determine some characteristics about the process used for a project. To provide consistency on the type of information needed, the questions are provided on two forms. The first set of questions is referred to as the Product Difficulty Index (PDI). Information concerning interfaces, timing, new function, and documentation level are recorded in the PDI.

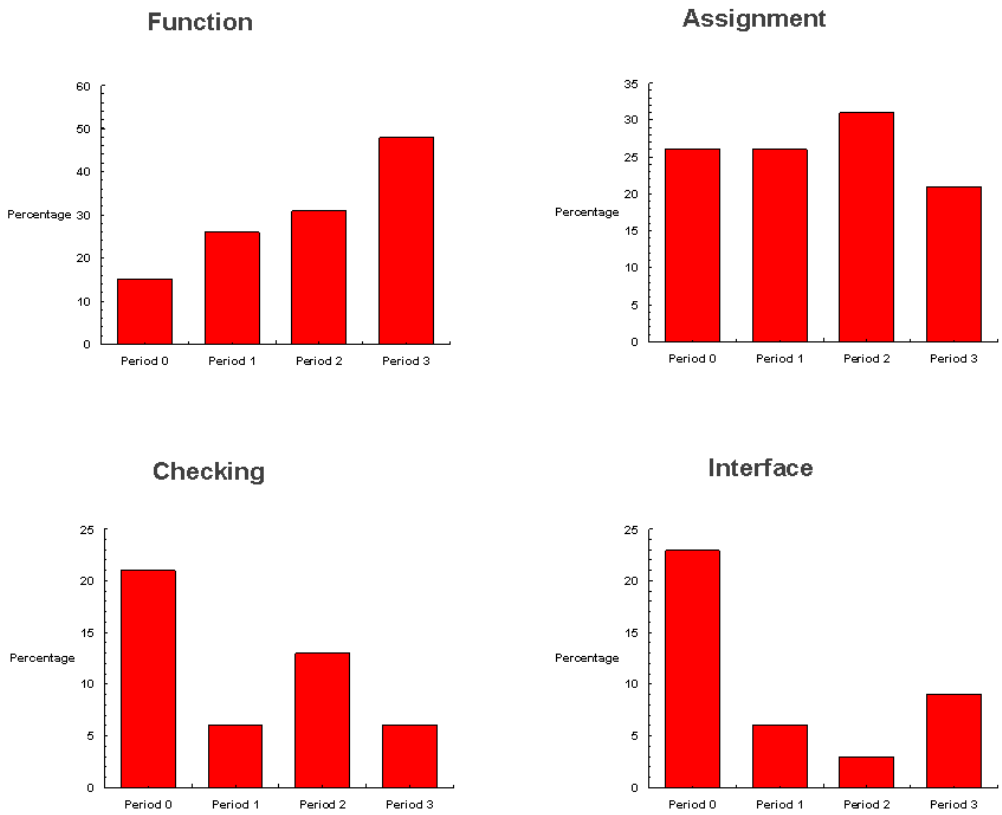
The next set of questions is on a process signature form. They determine characteristics about each stage in the process to be used in terms of percent of the total effort, what triggers are projected to be used to uncover defects, and the percentage of each trigger for each of the stages. This information provides the projected discovery rate and will be used when compared to the actual data. The information is used to determine whether a process stage worked as intended, or where modifications are needed. In addition, percentages are graphed to create what is

known as the 'signature' for the process. A signature is done for the triggers, and then algorithms are applied to determine the corresponding signature for the defect types at each of the stages. This relationship between trigger and type is affected by high or low PDI factors; therefore adjustments are made according to tables that describe the PDI and defect type relationship to get the appropriate signature.

Bar Chart 1 shows various trends that make up the overall type signature for a project. The four stages (shown in the chart as period 0 - 3) are design review, code review, function test and system test. Note that function type defects continue to rise, assignment defects are discovered during review and then again during system test. Note: A project would also have a signature for defect trigger, but it is not shown here.

Bar Chart 1

Type Signature Revealed by ODC Analysis



It is obvious from this type signature that the steady increase in function type of defects, the number of assignment type defects in system test, and the increase in checking defects during function test are potential process focus areas. Interface problems appear to be found at the right stages, so this would not be a focus item.

Keep in mind that this signature should be compared with what was expected. It may well be that new function was going to be introduced throughout the development cycle and that resulting simple errors, such as assignment, would continue to show up as a result in each of the later stages. The project focal point should work with the coordinator to update projections based on adjustment to the plan. Otherwise, the projection and actuals will always be different, even before analyzing the data.

An example of deriving the trigger signature using intended percentage breakout by stage is shown Table 2. Note that the first set of triggers is for review stages and the second set is for the testing stages. Each column adds up to 100%. .

Table 2.

Stages -> vs. Triggers	High Level Design	Low Level Design	Code	Function Test	System Test
Design	0.6	0.2	0.4		
Flow	0.15	0.6	0.5		
Concurrency	0.1	0.1			
Lateral Compatibility	0.05	0.5	0.1		
Backward Compatibility	0.05	0.5			
Special Review	0.05				
Single Function				0.7	0.15
Sequencing				0.2	0.15
Recovery				0.05	0.6
Hardware					0.1
Software				0.05	

Although these are just projected percentages, it does point out several things about the process that makes it unique from others. First, it identifies the stages to be used and how much emphasis is placed on each stage. If this percentage by stage is graphed, it gives the projected defect removal curve, independent of any calculation based on numbers of defects. If a project has quality goals, such as "70% of the defects will be uncovered prior to formal testing," then the projected effort by stage should support that type of goal. Second, by displaying the percentage of each trigger to be used by stage, it shows what defect removal activities are being considered, and likewise, which were not. In the example, 60% of the defects are to be removed by reviewing that the high level design corresponds to the specification and objectives, 15% due to errors while reviewing the flow, and a small percentage found by reviewing for compatibility with other products or previous release of the same product. In essence, the review for high level design is really a number of different types of reviews, because one reviewer will be looking for design conformance, another for flow, and so on. It is equally important to ensure that all the right types of "reviews" are taking place. Table 2 does not show a review trigger of document consistency, effects of language used, or review for any rare situations. So, even before a stage is executed, the plan for what is to be done to remove defects is laid out, and such omissions can be corrected. The results are quite different than from a review where everyone is reviewing for the same type of error.

Now examine the stages using test triggers. In this case, the triggers to some degree reflect the test plan for that stage. Table 2 the test triggers for Function Test indicate that 70% of the testing at this stage will be variations of main path command streams, invoking single functions, or executing simple scenarios, and 20% will be more complicated scenarios, or will require some sequencing of events. These two types of tests are expected to be executed at this stage. What the projection is saying about other tests is that only 5% of the effort is devoted to error recovery and nothing to volume workload or stress. The information supplied in the PDI along with any history of problems in the field will indicate whether this is a good plan for testing or not. The intent of doing the PDI and this projection of triggers is to encourage some thinking about the process that is going to be used to develop the product, and not just focus on the product itself. If the project turns out successful, the signature that was created for the process can be used as a model for other projects intent on using the same process.

With all of these project-specific factors employed, such as education, training, examples, tools, and a unique way to identify the process, ODC becomes a personalized process improvement methodology for that project. The key to

make sure all of these steps are taken is to have active site coordinators and active project focal points. In SSD-SSD, the only responsibility the site coordinators have is ODC, but the focal points are usually volunteers that take on ODC as additional workload. As a result, it is important that the focal point understands all the benefits ODC has to offer, and that those benefits can be realized only with their help in personalizing ODC for their project. It is an up-front investment that is justified when teams create actions that help improve their project.

3. Tie to Quality Management System

It makes sense for a site to experiment with ODC on a pilot basis. Once the decision is made to make ODC the site-wide process improvement methodology, as it is in SSD-SSD, it should be integrated into the Quality Management System (QMS) for that site. ODC is part of three elements of our QMS: Product Quality Plans, Quality Certification at Announce and before the product is generally available, and the Field Defect Escape Analysis Process.

Quality Plans - each quality plan has goals to reduce the number of defects, usually based on a directive from a higher level. Instead of just projecting numbers of defects, a project can use the ODC data to indicate which types of defects can be reduced and at what stages. For example, if a project discovered too many checking or simple assignment type of defects during function testing, several actions can be taken. The team could perform better design and code reviews to look for these specific types of problems, and unit test can be improved to make sure all decision points are executed. In the quality plan, the actions would be reflected as a goal to reduce checking and assignment defects in formal test by a certain percentage so the actions could be quantified. Examples are shown In Table 3.

Table 3

Goal	Method
50% reduction of checking defects in test	Error recovery paths reviewed separately at design/code. Error inject plan from previous release used as a model.
15% reduction of unitialized variables	Use of tool to validate all variables are set before used.
50% reduction of ported code defects in formal test	Review step added for ported code. Documentation requested prior to test from originator.

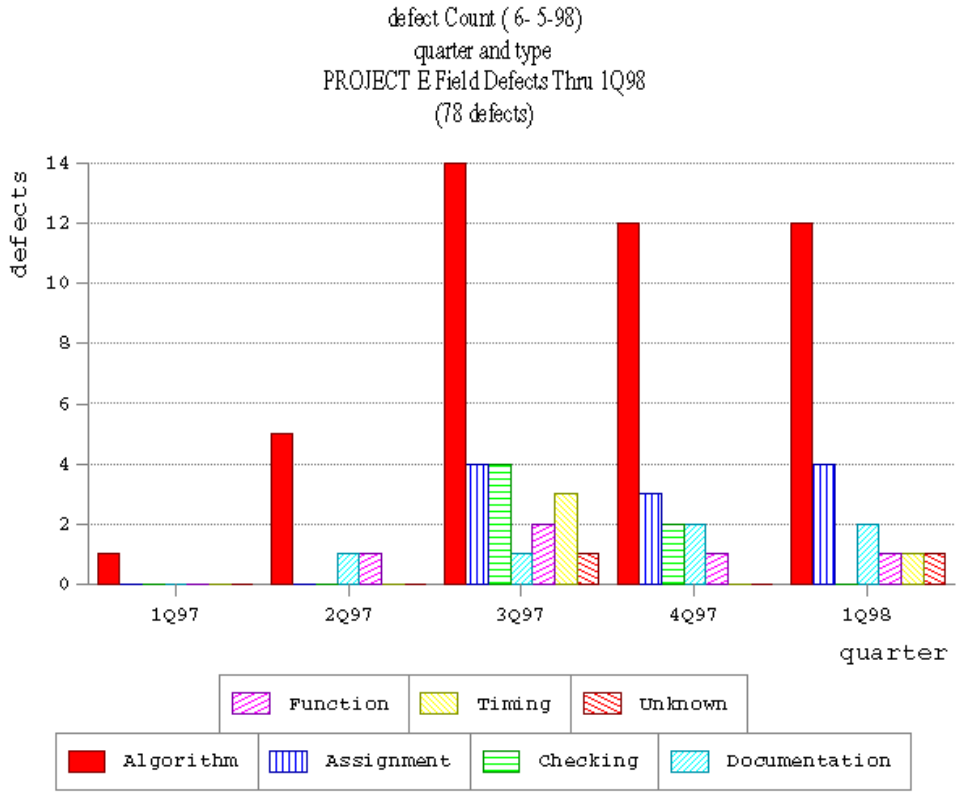
Another feature of the quality plan is a list of key checkpoints. The dates for phase level ODC analysis and feedback is listed as a checkpoint, which helps to integrate ODC as part of the product checkpoints.

Quality Certification - ODC is a checklist item for certification that a product is ready for announce or general availability. Each product must show evidence that ODC analysis was performed on the defects and what actions are being implemented as a result to improve the product quality. The goals reflected in the quality plan are validated using ODC data. Because ODC looks at percentages and classifications of defects rather than technical content and severity of defects, it can be used as an independent assessment of quality.

Field Defect Escape Analysis - ODC is listed as the first step in this process. Instead of analyzing every field defect, ODC is used to determine which defects, based on the way they are classified, would most likely be escape candidates. Defects that arise from special customer environments or conditions would not have been found by the testing lab because it did not have the same environment, and therefore are not good escape candidates. Simple assignment or checking defects found by issuing a command are better candidates. The ODC analysis provides the

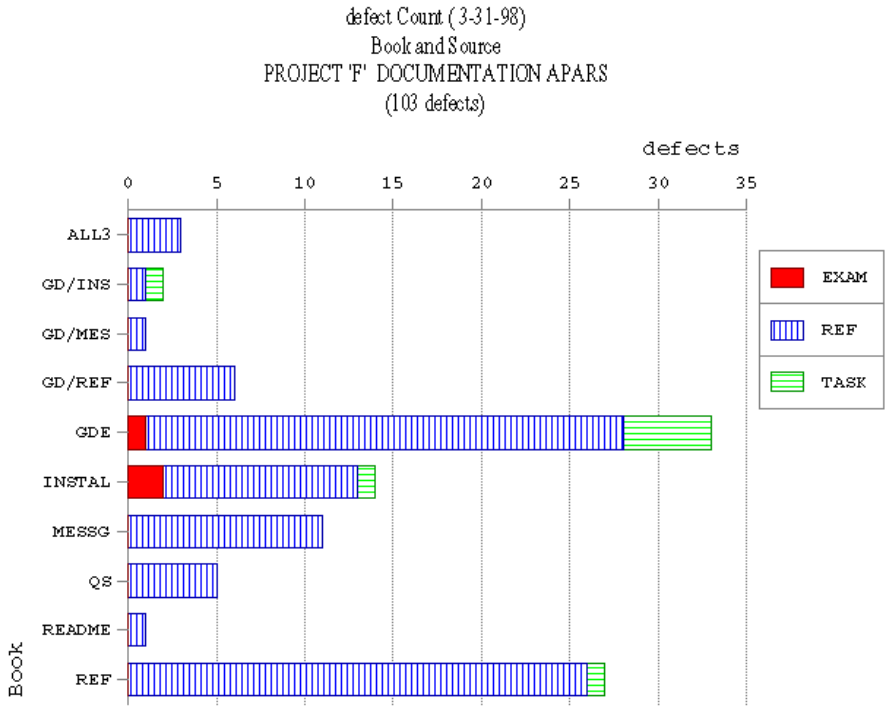
list of these type of defects and others that fall into focus areas, so that the escape analysis teams can concentrate on these areas first. The value of ODC for escape analysis is shown by Example 1and Example 2.

Example 1: Microcode supported tape product in the Field



Example 1 shows where the greatest number of defects is occurring (in 3Q97) and that the defect type of algorithm accounts for most field defects. Comparison of this chart with other views yields which of these defects were exposed by simple testing to determine the true escape candidates. An additional view using customer names revealed some accounts that had set ups not considered during development testing. The second release of this product has added these set-ups.

Example 2: Field Defects for Product 'F' publications



In Example 2, defect source was modified to represent terms used by Information Development to point out whether customers are experiencing more problems with examples, references, or task descriptions in this product's publications. The data shows that examples are in very good shape, the guide book (GDE) has some problems with tasks, and all the books need to focus on the reference materials, particularly the guide and the reference books, which have the most defects.

4. Develop a Good Feedback Mechanism

Consistent Source of Analysis - the first step to providing the proper feedback is to use the same tool. For SSD-SSD, the tool of choice is MYSTIQ, developed for internal use by CSE. It interfaces with the VISUALIZER tool to build one-dimensional views of a classifier, or many dimensional views of one classifier vs. another. Although there are many combinations of combining phase, activity, trigger, type, qualifier, source, and impact, a project may want to include other defect data fields, such as open and close date, severity, component name, and so on, to compare against one another or against the ODC classifiers, as illustrated by Example 2. This is at the discretion of the project focal point, but the more variety of data, the better the insight. All the charts generated have the same look and feel. This allows for multiple projects to compare charts, or to roll up charts to a higher level view.

Consistent Type of Analysis - with the many varieties of charts that can be created, it is important that the various views have a purpose. To this end, there are a number of key charts that are the basis for most analysis and feedback reports. Each has a specific function in building the proper feedback. These are listed in Table 4.

Table 4

View	What it shows
Stage vs. Trigger	**Distribution of triggers over time. Compare this with the projection to determine review or test effectiveness. This view provides the trigger signature if shown as percentages.**
Stage vs. Type	**Distribution of defect types over time. Look for continued growth of certain types, such as function, as a focus area. This view provides the type signature when shown in percentage of type within a stage.**
Stage vs. Impact	**Distribution of the impact over time. Look for decrease in reliability towards end of testing cycle. High number of usability is usually a classification issue.**
Stage vs. Source	**Distribution of location of defect over time. Usually an increase in the number of base code defects in the later stages. Significant percentage of categories other than new are potential focus items.**
Stage vs. Qualifier	**Distribution of missing or incorrect over time. An increase in missing usually points to an inadequate design.**
Type vs. Qualifier	**Breakout of type by whether it is missing or incorrect "code." Look for missing function or missing assignment and checking in later testing cycles as focus areas. In general, the missing defects should be less than 30% of each type.**
Type vs. Trigger	**What reviews/tests are effective at uncovering what types. Look for distributions that seem out of place - either high or low.**
Type vs. Impact	**Which defect types regardless of phase create the biggest impact to the customer**

Knowing the functions that each of the classifiers provides in the analysis add to the importance of properly classifying the data for each defect.

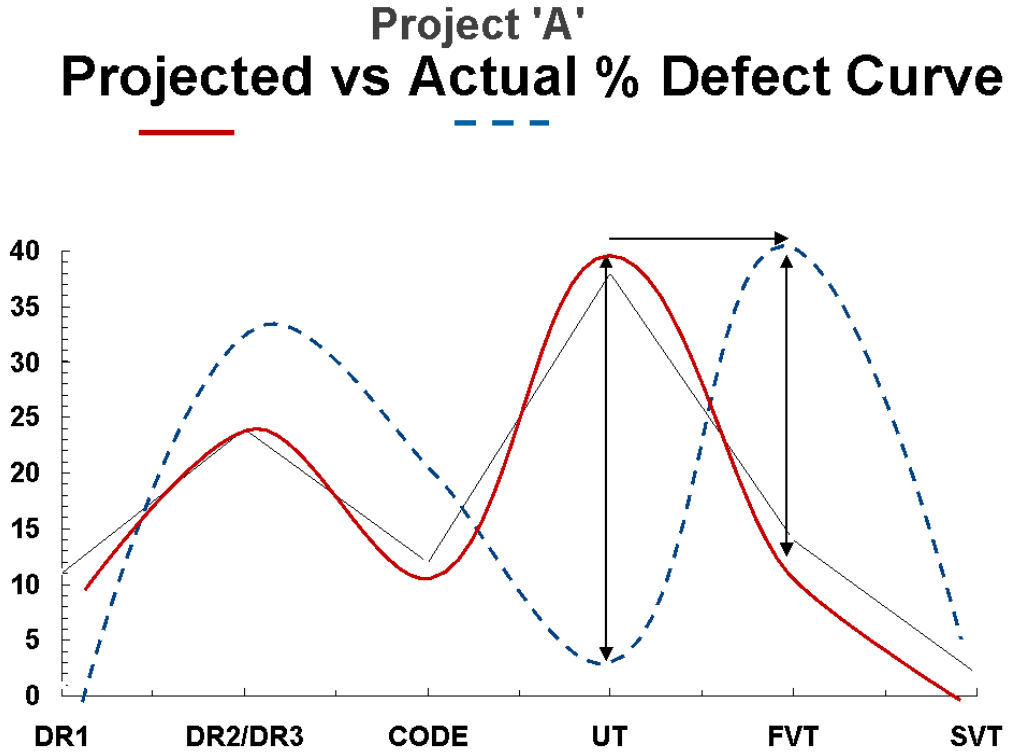
Consistent Format of the Feedback - Still evolving, but through experience, the projects want to know the following items: Effectiveness of Reviews, Effectiveness of Test, glaring process holes, defect escapes, specific recommendations, and effects of actions previously implemented.

Consistent place to find feedback: A repository in Lotus Notes was set up with universal access so that all project members can refer to the data presented during a formal feedback meeting. The forms feature of Lotus Notes allows for the data to be searched through many differing views including "by project," "by organization," "by person." Lotus Notes provides features to create document links or, copy and paste the feedback to any location, such as status report repositories, presentation packages, and department folders. Appropriate adjustments are made for data presented externally, as was the case for examples extracted from feedback reports for this presentation.

IV. Examples of ODC Analysis on Real Data

This section contains examples using actual data that demonstrate our approach to showing the many benefits of ODC. The examples were chosen to show different development processes, and include a project that modified its plan after interim feedback and, as a result, delivered very clean code to formal test.

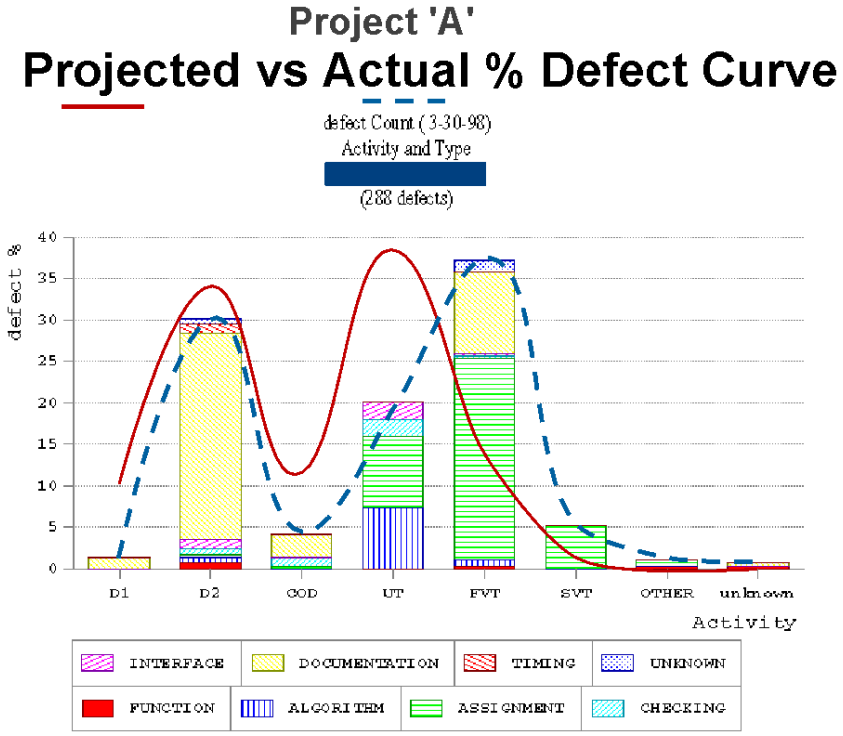
Example A: Complex line item with numerous interfaces.



Traditionally, the projected vs. the actual percentage of defects has been tracked, albeit in a one-dimensional view. The previous example shows that the anticipated discovery of defects in code review did not actually happen until the function test (FVT), and that no low level design review activity (DR1) was actually done. Regardless, more defects were discovered prior to unit test than was expected. The project, even with these phase shifts of discovery, appears to be in control by the start of the system test (SVT). Does this indicate that the project had an effective process?

Without additional data, there is no way to tell whether this is a problem in projecting the defects or a shift in more emphasis in test vs. in code reviews. Based on this amount of data, teams usually take no actions to correct anything in their process, especially if the project appears to be in control by the end of system test.

Example A' - Project A with ODC added



Example A' shows that the same type of projected vs. actual curve can be enhanced with ODC data to help answer the question of whether a particular stage was effective, and to some extent, whether the projections were accurate or the underlying process was efficient.

When the actual data curve is overlaid with the various ODC defect types for each stage, it reveals the fact that most of the defects in FVT were the result of simple assignment errors, and this trend continued into SVT. Note that most of the timing issues and algorithm type defects were addressed during unit test. The following analysis concerning the chart in Example A' was provided to the team:

> *There were 288 defects recorded from phase DR1 to phase SVT. The corrections to the specification document are reflected in the high number of documentation defects at D2 (DR2 timeframe), and also during FVT when the actual mappings of the various volume records were displayed, and determined to be in error. Unit test flushed out most of the algorithm type of errors and the few interface errors found. The observation is that checking type problems are low for the FVT phase and that, based on the notion that this is API code, there would be more interface type of defects at this stage as well. SVT was mainly a regression test of FVT, so it would be expected that more types of defects would be uncovered than just the assignment ones reported.*

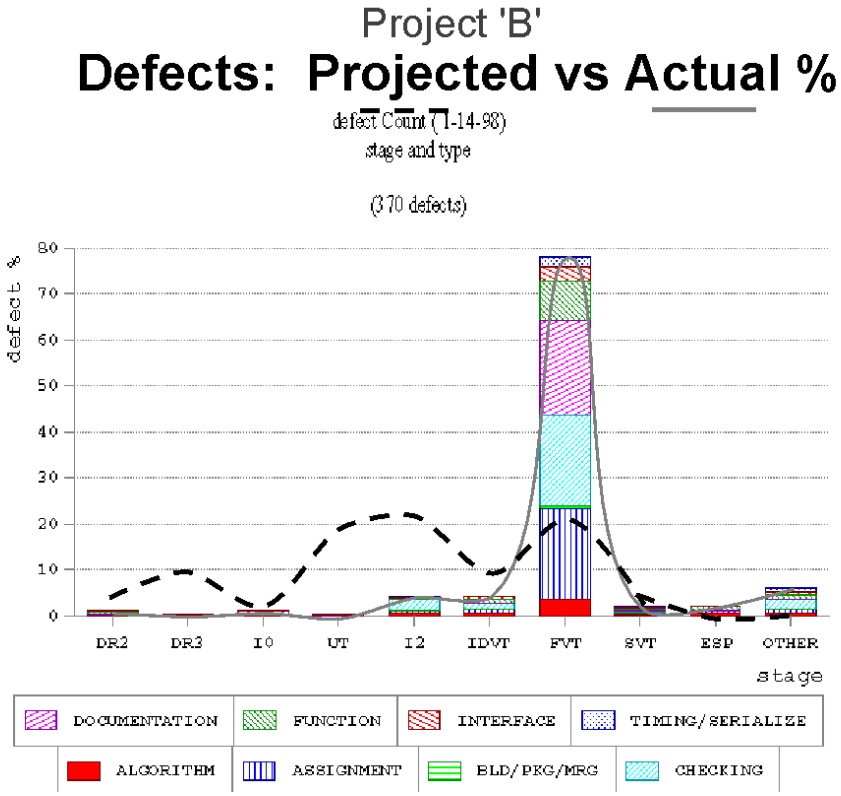
The conclusion is that larger pieces of code were well tested at unit test time, but simple items like variable settings were only partially checked, and it was not until FVT that these assignment type of defects were uncovered. Assignment type analysis showed that they could have been discovered by a simple code review. The recommendation to the team based on this analysis was:

Recommendations:
Continue the effort toward the 6 "Cs" (completeness, clarity, coherent, etc) to produce the specification. Based on the defects discovered in FVT, this appears to be successful in developing a thorough design. The iterative process appears to be successful as well. To improve review effectiveness (percentage of defects found prior to formal test) to a higher ratio than 59/41 determine why the algorithm defects and the missing assignment defects could not have been discovered via review. It appears that most of the review activity was towards design conformance. Different perspectives, such as reviewing the flow for logical holes, and separate reviews on the algorithms is suggested if dealing with this type of data conversion code again. Finally, the fact that a test team, not previously familiar with the complexities of Product 'A', was able to find so many defects at FVT timeframe using a compare tool suggests that this tool could have been used in unit test resulting in earlier defect discovery. Its use for this purpose should be explored by the development team.
The next step is to take these recommendations and those from the team and formulate action plans on improving this hybrid, but apparently effective, process. The proof of its effectiveness will be validated by the field data.

Now for an interesting discovery! If the defects that were found during FVT are mapped to the appropriate stage in which they should have been found, it shows that the projection for code review through SVT are more accurate than first thought. Defects such as missing assignments that are discovered by simple command testing are probably escape candidates from earlier test or code review stages. By moving the assignment type of defects that were found by simple type of command testing into the column of the defects that were found during code review, the number in the code review stage would increase closer to the projection, and at the same time, the number of defects in FVT would decrease, also closer to its projection. So, in this case, it is not the projection, but the execution of the process in finding these types of defects in earlier stages that needs focus.

This project has started another release with basically the same team and the same overall process. To emphasize the capture of simple mistakes at code review timeframe, the team has added additional review time to the schedule. To ensure this, the quality plan for the release emphasizes this.

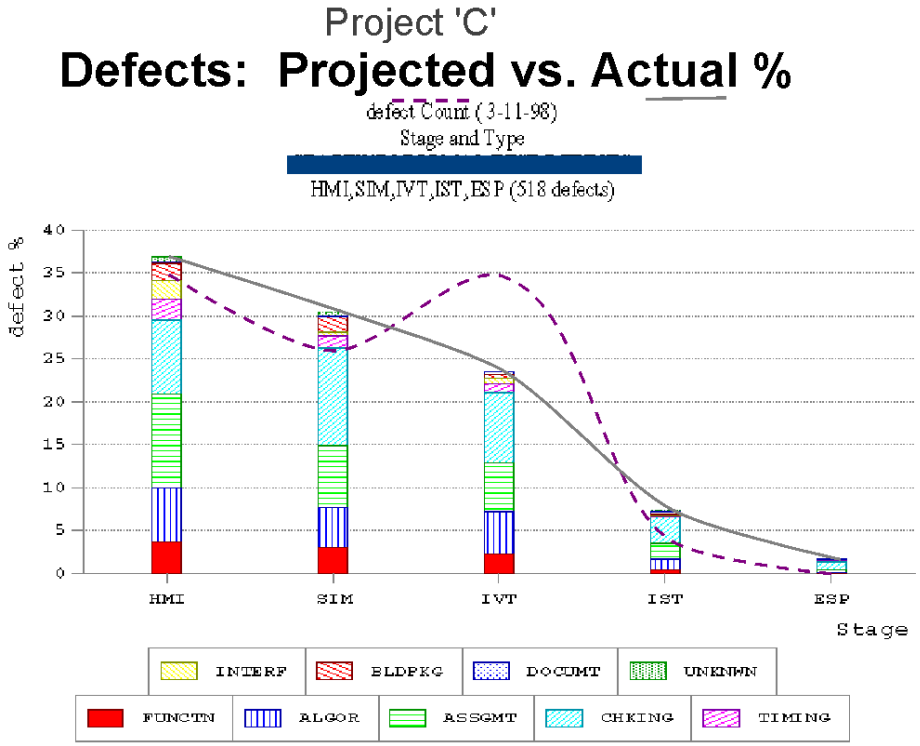
Example B: Well defined, fast paced project with experienced team



Example B is a mapping of the ODC data on the projection vs. actual defect curve for Project B. The projection was based on the way this team of experienced programmers went after defects in the past. The actual defects reflect issues with available reviewers and team members, who were much better at testing than reviewing, after the project began. Clearly, there was no effort placed in performing earlier reviews as there had been in the past. Because the code was complex and little was found in the earlier stages, the team developed a very robust function test matrix. As with Example A, the project appeared to be in control by the end of SVT. The curve after SVT is a result of new enhancements requested by the customer. These additional, not previously planned, code drops late in the cycle resulted in an additional testing cycle to flush out the same type of errors found during the FVT.

Mapping the defect types back to the stages where they should have been found shows that the projections for design and code reviews would have been closer to what was projected.. In the case of unit test projections, the team was far too optimistic, and the project required more complicated testing to create the conditions necessary to show the function was performing as desired.

Example C: Microcode project using Simulation Testing



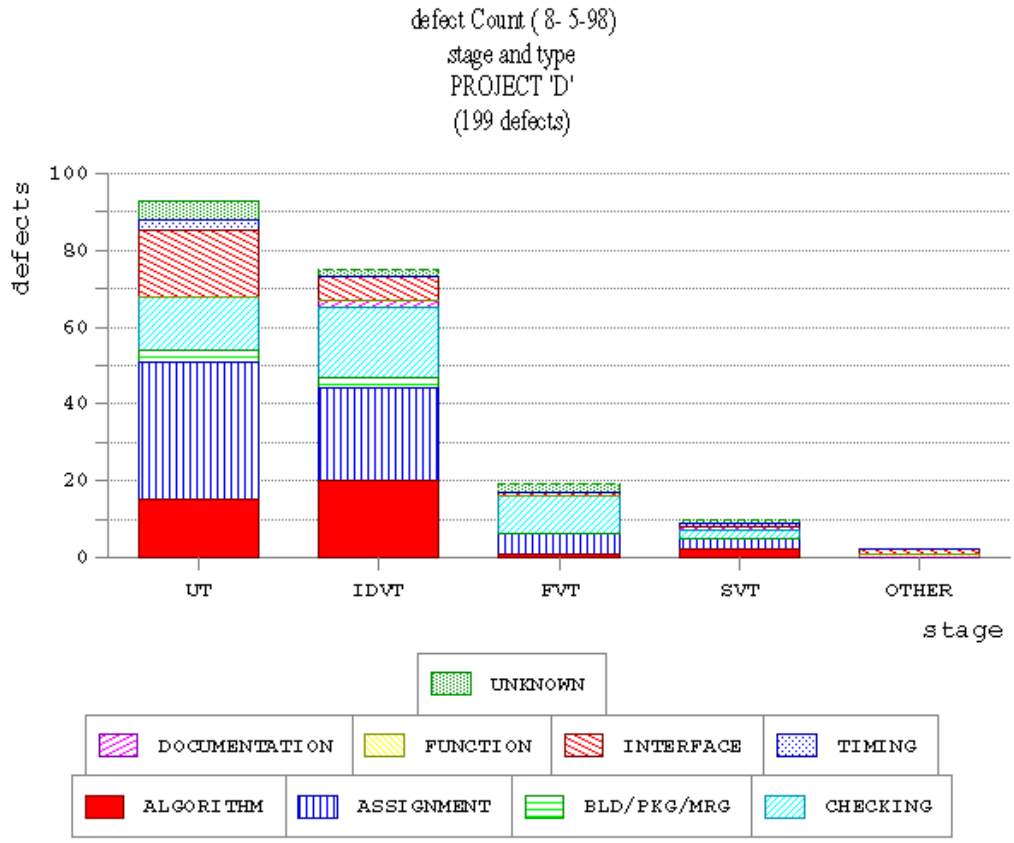
Example C shows a project where the effects of adding simulation testing was thought to have influence on finding defects, but the amount was unknown. The ODC data shows the effect simulation had on the other formal tests, and the type of defects uncovered. The data shows a large portion of the checking defects are removed by the simulation testing, which is less expensive than running on the actual hardware. The graph also shows the tests in the order of execution, with the exception of simulation testing, which runs during the entire testing cycle. Note that the less expensive simulation testing can find the same type of defects and is not limited to just simple errors like assignment and checking. The feedback to the team was summarized as follows, and included data from earlier feedback of the design stages:

There is a recommendation for needing better design reviews; it comes from several of the orthogonal views. First, the percentage of missing defects throughout the cycle suggests many different areas, such as the "teach" function and error recovery paths that did not have thorough design reviews. Secondly, the distribution of the "types" of defects mapped against those missing also indicate which are design review holes, especially of type function or algorithm. As the chart shows, two areas are prevalent: new function added throughout the cycle and missing conditionals on decision statements (type=checking). The amount of base code defects suggest that code reviews did not validate the code extracted from the xxxx library source. Also, regression testing, especially in the area of error recovery from hardware error scenarios should be added to the next plan. The developers should spend some time reviewing the formal test plans. Some of the defects occurred by simply following a prepared script. Developers could have followed those paths in a code review and discovered some of those defects. Even though the testing stopped, the defect distribution shows that SIM testing found about the same number of problems month to month. There is no indication that it would not have found more had the test continued.

Summary:

If you are looking to reduce the number of function defects, the number of missing defects, the number of checking defects, or the number of base code defects, all of these goals can be quantified by the ODC data for Project 'C'. If you want to enhance SIM testing, that too can be quantified.

Example D: Software project, early change to process



The data in Example D represents a project that had multiple line items, one of which experienced some early problems and was subsequently pulled from the release. For the most part, every defect was formally captured regardless of whether it was design or code review, unit testing or any of the formal test cycles. Early analysis of the data was provided after the low level design, after code and unit test, and after function testing. The curve of the projected and actual defect percentages at each of the stages is shown in the graph for project 'D'. It points out that this project realized it had some design issues, especially in error recovery, confirmed by the ODC analysis, and it took some extra steps to ensure that they were covered. The result was an increase in the development testing until they were satisfied that all the defects were removed. When the code was turned over to formal test, very few defects were found. This type of defect removal curve is similar to one for a clean room process, but this project used a traditional waterfall approach adhering to stage exit criteria.

V. ODC Benefits for IBM SSD-SSD

As our implementation strategy steadily evolves, more benefits of ODC continue to emerge. The following are the highlights of those benefits:

Timely Feedback

There is more analysis and opportunity for correction due to the feedback mechanism. ODC analysis and feedback can occur at predetermined checkpoints, such as the end of High Level Design, Low level Design, Code and Unit Test, and so on, or can be done when there is enough data that makes the results statistically valid. Timing for

analysis and feedback is at the discretion of each team, but more teams are requesting early stage data to make sure they are on the right path. Many teams have put actions in place in the quality plans to cover defects for these early stages.

Quantifiable Actions

Goals, such as finding seventy (or greater) percentage of the total defects prior to formal test, can be accomplished with the right set of actions. As the examples illustrated, the breakout by ODC data of the later stages shows the probable escapes from earlier design and code stages. The feedback provided through ODC points to specific areas that can address these escape issues. The reduction of a certain class of defects or a percentage number of defects moved to an earlier phase is a quantifiable process improvement.

Efficiency and Effectiveness Measures

Through the use of trigger, stage or activity, and type analysis we can show both efficiency and effectiveness of reviews and tests. Given the breakout of activities and triggers for a given phase, efficiency can be thought of as the measure of the percentage of the planned activities and triggers against those that were not expected. For example, if during a system test phase there was review activity for a big fix, function test activity for a new function added late, and some documentation activity, the system test would be less efficient because not all of the timeframe was spent performing what would be considered the planned system test activity.
Effectiveness is based on how many of the proper triggers actually uncovered defects and whether they found the right type of defects for the phase. For example, if the system test is designed to have long running test cases or large workloads and it finds mainly assignment type of defects and the triggers are single function, this test is doing little more than a unit test. In other words, it was not effective in finding the errors normally associated with system test.

Use of Unit Test Data

Prior to our implementation of ODC, the defect types uncovered by the Unit Test phase were not documented, so it was unclear whether it was this phase or an earlier one that was responsible for certain escapes into formal testing. With the capture of ODC unit test data for some projects, we know more about what to expect from unit test, how to more accurately project code review defects if unit test is performed before it, and what steps can be taken to make unit test more effective.

Alternative Defect Projection Method

A by-product of a PDI and signature is a defect projection curve based on the complexity of the product and the process used to develop it, and not just on lines of code. As the examples showed, the defect projections can be validated with the actual ODC data.

New Focus on Design Reviews

The information gained by analyzing the defects resulting from missing code helps improve design reviews. It is easier to spot a defect for the design that is documented. Therefore, the ODC testing data that points out missing code is the same data that can be further analyzed to improve methods of reviewing to look for these same type of errors, only earlier in the development cycle.

Improved Code Reviews

Likewise, code reviews can be improved by analyzing ODC test data for the incorrect code, because that usually points to an implementation rather than a design issue.

New Field Data Use

Another benefit yielded by our ODC implementation is the extended use of APAR (field) data. As mentioned in our strategy, ODC data is collected for the field defects to initiate the escape analysis process, so that not every defect has to be analyzed. Also, the field data helps to determine whether enough emphasis is being placed on these types of errors in-process to prevent their occurrence in the customer environment for the subsequent release. For example, if field data showed a sufficient number of defects found by recovery exception testing, it is beneficial for the project to ensure they have sufficient recovery testing as part of their plan.

VI. Summary

The implementation of ODC takes a dedicated site coordinators and area focal points that can tailor each project so its team members can have the education, tools, references, on-site help, and necessary feedback to make it successful. In SSD-SSD, we have chosen some unique approaches on a team-by-team basis, because it really is a methodology that should be tailored. These approaches highlighted in the following letter to another site interested in
jump-starting their ODC effort. Good luck on your implementation strategy, and by all means, have fun doing it!

Spend one weekend correcting the data for a project that showed little buy-in. It is a sacrifice, but the data won't be meaningless and you can defuse the part about it being too complex by taking one or two examples to those that think so and show them how easy it is. This will also help you to get to the crux of the problem. Is it really too complex, or is it just one or two classifications giving them problem like capability vs. reliability? Perhaps a little extra documentation is all that is needed.

More importantly, if you believe in the power of ODC, then the analysis that you do on the data might provide that group some guidance. If you would like a second opinion of what the charts are saying, feel free to share them with me. I caught one group off guard with just how powerful the combined orthogonal views could be. Show your data to management and tell them with the help from development you can get down to the needed process changes. In that meeting, ask them how they demonstrate continuous process improvement and you need to find out why ODC would not be a better, more auditable, less time consuming approach than what they are using.

Think of ODC as reducing the task of looking at all the defects individually down to looking at only a few focus areas and the defects within those. When you present it as reducing work on their part, that seems to get their attention. That, and also focusing on what went right with their process, based on the data.

If you do not have a coordinator in each development and test area, then request one. You need an advocate in each group, and that is the best way to keep ODC on their minds for all those meetings you are not invited to.

Offer noon-time refresher sessions, and have them bring their data in to do the classifications right there. This strategy does not take away from their regular work, it illustrates with data they understand, and it gets some of the data filled in. Also it shows them that it is not this all time consuming task that they make it out to be.

Strike a deal with one group that you will do classification on a daily basis for several weeks if they will monitor how you are doing and then take over at a prescribed time, with the caveat that you will confer on any classification issues when they take over. On a daily basis, this task is not daunting; waiting a week can make it very time consuming.

See what you can do to get ODC part of your Quality checkpoints (even though this a process improvement methodology). If you don't have management buy-in it is because presently your ODC is not part of any approval cycle or part of the certification, so their product can go out the door with or without ODC. We delayed a ship date because it was obvious from the ODC data that they were not through testing

yet. That was a FIRST. It won't be the LAST.

Find out from the director what he thinks the level of deployment for ODC in his organization is, who has what responsibility, and then show a scorecard on education, classification, actions, where would each group rate. You need to do this, so you can guage your own progress as well.

All these things makes the job meaningful , and fun. In some places it may be a requirement; developers have a job to do, but so do you. Get them involved in the success in ODC, just pick a couple of developers and ask them for suggestions on the definitions, what would make it easier for them. Get many different views. You would be surprised that somehow they find the time to provide help even with a busy schedule.

********** Document End **********

Tom Messmore

Tom Messmore is an advisory software engineer at IBM Tucson in the Storage Systems Development-Storage Systems Division, with 26 years of programming, simulation, testing, and management experience. For the past nine of the 11 years, Mr. Messmore has been responsible for various quality programs for the programming organization. Previously Mr. Messmore worked as both a test team leader and test manager for both system 390 and distributed products in SSD-SSD. Prior to IBM Tucson, Tom worked as liason to NASA on the Space Shuttle Program and As a programmer for the Enroute Air Traffic Control Program in the old Federal Systems Division.