Improving software testing via ODC: Three case studies

by M. Butcher
H. Munro
T. Kratschmer

Orthogonal Defect Classification (ODC) is a methodology used to classify software defects. When combined with a set of data analysis techniques designed to suit the software development process, ODC provides a powerful way to evaluate the development process and software product. In this paper, three case studies demonstrate the use of ODC to improve software testing. The first case study illustrates how a team developing a high-quality, mature product arrived at specific testing strategies aimed at reducing field defects. The second is a middleware project that identified the areas of system test that needed to be strengthened. The third describes how a very small team with an inadequate testing strategy recognized its risk in trying to meet the scheduled release and made the product more stable by postponing the release date and adding badly needed testing scenarios. All three case studies highlight how technical teams can use ODC data for objective feedback on their development processes and the evolution of their products. This feedback facilitates the identification of actions to increase the efficiency and effectiveness of development and test, resulting in improved resource management and enhanced software quality.

Software systems continue to grow steadily in complexity and size. The business demands for shorter development cycles have forced software development organizations to struggle to find a compromise among functionality, time to market, and quality. Lack of skills, schedule pressures, limited resources, and the highly manual nature of software development have led to problems for both large and small organizations alike. These problems include incomplete design, inefficient testing, poor quality, high development and maintenance costs, and poor customer satisfaction. As a way to prevent defects from being delivered, or "escaping," to customers, companies are investing more resources in the testing of software. In addition to improving the many other aspects of testing (e.g., the skill level of testers, test automation, development of new tools, and the testing process), it is important to have a way to assess the current testing process for its strengths and weaknesses and to highlight the risks and exposures that exist. Although it is well documented that it is less expensive to find defects earlier in the process and certainly much more expensive to fix them once they are in the field,[1] testers are not usually aware of what their specific risks and exposures are or how to strengthen testing to meet their quality goals.

In this paper, we discuss the use of Orthogonal Defect Classification (ODC),[2] a defect analysis technique, to evaluate testing processes. Three case studies are presented. The first two discuss software products that began using ODC in early 2000 when the three authors of this paper started working together to deploy ODC at the IBM Hursley site. The project manager for the product in the third case study was T. Kratschmer, one of the authors.

Developers of the product discussed in the first case study began using ODC to evaluate field defects. Al-

©Copyright 2002 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

though it is a high-quality, mature product that does not generate many defects, defects still occur in the field, where it is the most expensive to fix them. The goal of this team in using ODC was to discover how to enhance the testing process to find more of these defects in house, thereby decreasing the number of defects escaping to the field and cutting warranty costs. The second case study describes a large middleware project with a well-defined development process. Despite the mature process, this team had several problems and decided to see whether ODC could shed some light on the improvements the team needed to make. The team was especially interested in any exposures in its process that ODC could highlight and that could be addressed before releasing the product. The third case study represents a very small project. ODC was used to evaluate product stability and identify areas of test that needed to be strengthened to meet an acceptable level of test criteria before releasing the product. Because of an inadequate focus on testing, it was important to have an objective method of measurement that would convince both management and the development team to agree to the decisions that were being made.

Overview of Orthogonal Defect Classification

There have been several papers written on ODC that describe the concepts, define the ODC scheme, discuss the relationships among the ODC attributes, and include results from pilot studies.[2,3] For this reason, we include only a short description of the ODC details in this paper. ODC methodology provides both a classification scheme for software defects and a set of concepts that provides guidance in the analysis of the classified aggregate defect data. "Orthogonal" refers to the nonredundant nature of information captured by the defect attributes and their values that are used to classify defects. Much like the Cartesian coordinates in geometry, nearly a decade of research has shown that these attributes (and their values) are adequate to "span" the interesting part of the defect information space for most software development issues. Unlike other defect analysis tools used primarily by management,[4,5] ODC targets the technical team—developers, testers, and service personnel. As a result, it is the technical team that classifies defects, takes an active part in assessment of data, and decides on the actions to be implemented. The ODC attributes are listed in Appendix A and B. Appendix C lists items typically included in an assessment of ODC defects.

ODC deployment process. The process for deploying ODC has evolved over the last 10 years. However, the following basic steps are critical in order for the ODC deployment to be successful:

- Management must make a commitment to the deployment of ODC and the implementation of actions resulting from the ODC assessments.
- The defect data must be classified by the technical teams and stored in an easily accessible database.
- The classified defects are then validated on a regular basis to ensure the consistency and correctness of the classification.
- Once validation has occurred, assessment of the data must be performed on a regular basis. Typically, the assessment is done by a technical person who is familiar with the project and has the interest and skills for analyzing data. A user-friendly tool for visualizing data is needed.
- Regular feedback of the validation and assessment results to the technical teams is important. It improves the quality of the classification. It also provides the teams with the necessary information so that they can determine the appropriate actions for improvement. This feedback is also important in obtaining the necessary commitment from the technical teams. Once they see the objective, quantified data, and the reasonable and feasible actions that result, commitment of the teams to the ODC process usually increases.
- Once the feedback is given to the teams, they can then identify and prioritize actions to be implemented.

When this ODC process has been integrated into the process of the organization, the full range of benefits can be realized. The development process and the resulting product can be monitored and improved on an ongoing basis so that product quality is built in from the earliest stages of development.

Classification and validation of defects. The classification of the defects occurs at two different points in time. When a defect is first detected, or submitted, the ODC submittor attributes of activity, trigger, and impact are classified.

- *Activity* refers to the actual process step (code inspection, function test, etc.) that was being performed at the time the defect was discovered.
- *Trigger* describes the environment or condition that had to exist to expose the defects.
- *Impact* refers to either perceived or actual impact on the customer.

When the defect has been fixed, or responded to, the ODC responder attributes, which are target, defect type, qualifier, source, and age, can be classified.

- *Target* represents the high-level identity (design, code, etc.) of the entity that was fixed.
- *Defect type* refers to the specific nature of the defect fix.
- *Qualifier* specifies whether the fix that was made was due to missing, incorrect, or extraneous code or information.
- *Source* indicates whether the defect was found in code written in house, reused from a library, ported from one platform to another, or outsourced to a vendor.
- *Age* specifies whether the defect was found in new, old (base), rewritten, or refixed code.

Definitions of the attributes and a more detailed discussion with examples can be found at http://www.research.ibm.com/softeng.

Typically, the ODC attributes are captured in the same tool that is used to collect other defect information with minor enhancements. Two methods are used to validate data. The individually classified defects can be reviewed for errors by a person with the appropriate skills. This may be needed only until the team members become comfortable with the classification and its use. It is also possible to use an aggregate analysis of data to help with validation. Although this method of validation is quicker, it does require skills beyond classification. In order to perform a validation using this method, the validator reviews the distribution of defect attributes. If there are internal inconsistencies in the information contained in the data or with the process used, it points to potential problems in the quality of the data, which can be addressed by a more detailed review of the subset of defects under question. Even in cases where there is a misunderstanding by a person in the classification step, it is typically limited to one or two specific aspects, which can be clarified easily. Once the team understands the basic concepts and their use, data quality is no longer a problem.

Data assessment. Once the data have been validated, the data are then ready for assessment.[6,7] When doing an assessment, the concern is not with a single defect as is done with causal analysis.[8] Rather, trends and patterns in the aggregate data are studied. Data assessment of ODC classified data is based on the relationships of the ODC attributes to one another and to non-ODC attributes such as component, severity, and defect open date. For example, to evaluate product stability, the relationships among the attributes of defect type, qualifier, open date, and severity of defects might be considered. A trend of increasing "missing function" defect type or increasing high-severity defects may indicate that the product stability is decreasing.

The following three case studies use ODC to measure test effectiveness. In each case, ODC highlighted exposures in the testing process that needed to be addressed. The assessments provided the required information for the final step in the ODC process—selecting the appropriate actions for improvements. The case studies include background information on the products, how their ODC process was implemented, and the details of the assessments, including actions that resulted.

Case Study 1: Learning from the field defects in a mature product

The product in this first case study has a mature development process and is recognized as a high-quality product that provides industrial-strength solutions for mission-critical applications. However, a few defects still escape to the field and need to be fixed under service. The cost of these defects to the customers arises both from the impact of the defect and the expense of applying service to the products. The cost incurred in our development organization from a field defect is far greater than if it had been found during our test phase, and this in turn is more expensive than finding it during the design and code phase. Irrespective of the exact cost savings, it is clearly better not to introduce defects into the design or the code, or at least to find defects early in the development phase.

The team aims to avoid injecting defects in the first place, and many of the existing quality activities do reduce the number of defects introduced into the code. Measurements used during the development process include the number and rate of defects (the defect detection model). There is also a "lessons learned" analysis at the end of each release, with changes made to improve the development process.

The current process for the development and test of this product includes the following main stages:

- High-level design, which is documented and inspected
- Low-level design, which is documented and inspected

Figure 1 Age versus activity in uncovering defects in code

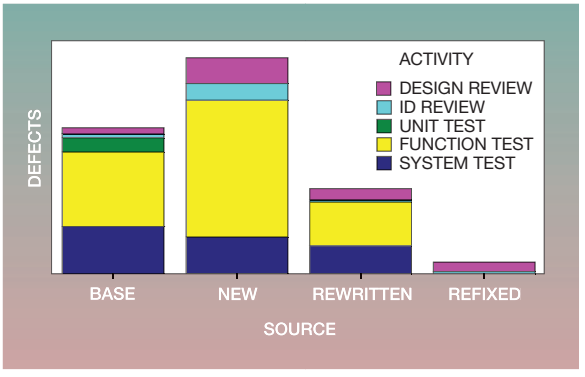
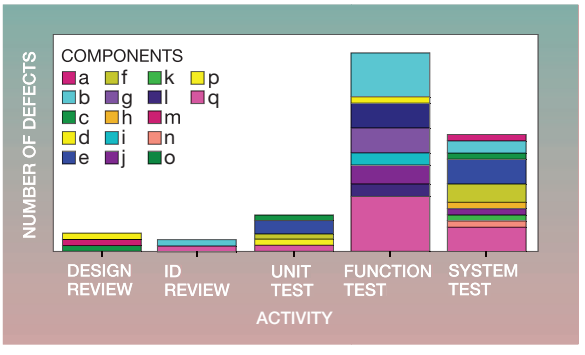


Figure 2 Frequency for component within activity (base defects only)



- Coding that is buddy-checked or inspected
- Functional verification testing (test plans are inspected)
- System testing, acting as first user (test plans are inspected)
- Package and release test (repeats existing test cases)
- Solution test (integration test with other products)

There are numerous regression test suites created during previous releases. These suites are used during both function verification test (FVT) and system test to ensure that the new release changes do not cause problems in the existing code base. In addition, causal analysis is performed on each field defect in order to understand how it escaped to the field and to prevent the escape of similar defects in the future. In spite of these numerous best practices, more focused improvements are needed. ODC is able to provide that focus by identifying the areas of greatest opportunity in testing based on customer usage.

Deployment. After the decision was made to implement ODC for the last release of the product, a team of seven people from development, FVT, system test, and solution test was formed and then educated in the use of ODC. This team met weekly to classify the unique customer-discovered defects.

When classifying field defects, first the trigger is chosen based on the actual description of the circumstances, exposing the defect and hence the activity that most likely would have caught the defect, if found in house. If it is not known what the customer was doing when the defect was found, we record how to recreate the problem in house. If code was executing when the defect occurred, it is assumed to have escaped from a test phase. In fact, the majority of defects did escape from a test phase. A few had also escaped documentation reviews.

Assessment. After classifying 12 months of ODC data, an assessment was performed. Although there is a need to be cautious in making major changes because this product has a life cycle longer than 12 months, the assessment highlighted some interesting facts.

Importance of base code. The first ODC attribute reviewed was age. Was the customer uncovering defects that indicated exposures with base, rewritten, refixed, or new code? Figure 1 shows age versus activity. Although the development team members expected most of the defects would be in the new code since defects are usually introduced in new or updated code, they were surprised to find a significant number of defects in the base and rewritten code, as well. Two functional areas that had a significant amount of rewriting without including new function accounted for those defects classified as "rewritten." However, the number of defects in the base code seemed high. The next step was to identify from which test phase those base defects had escaped and the functional components that contained the defects. The frequency for component within activity for the base defects (Figure 2) shows that the majority of escapes to the field were from FVT and that two specific components contained over 40 percent of the defects.

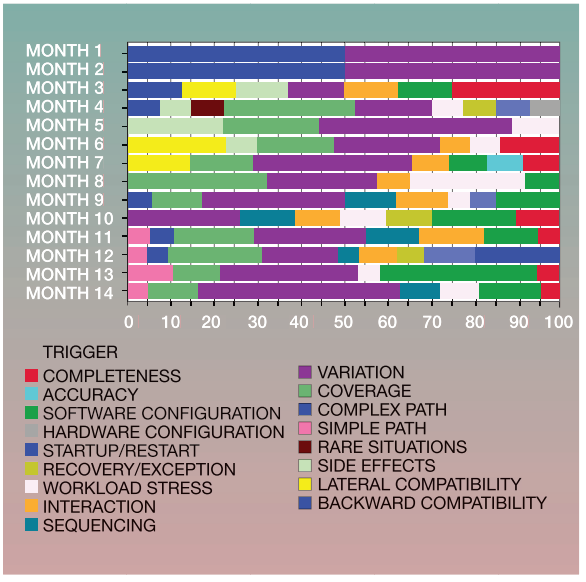
The ODC assessment identified the areas that required further investigation. Causal analysis techniques were then used on the base defects within these specific components. This assessment showed:

1. Component "q" required more regression testing, specifically near the end of the test phase. This testing is being included in the next release.
2. There had been a significant amount of new code added in component "b," so these defects were being uncovered as the new code was exercising some aspects of the base code that had not been used before in that way. The defects in this area are still being investigated, but an obvious improvement is to include more testing of the base function for components that are being affected by the new code.

When looking at *all* defects, it is clear that component b had the greatest number of defects. This functional area had undergone significant change during the project. The team knew that some of the changes had not been fully documented because of schedule constraints, and the risk of less documentation had been accepted. However, this ODC assessment identifies the impact of that decision. In future releases, the process for documenting changes will be enforced.

Improvements to function test. Ideally, the number of defects uncovered by the customer decreases over time. In Figure 3, which shows percent view of the trigger classification for field defects over time, it can be seen that the number of defects actually increased. However, that increase was expected since the number of licenses is still increasing. More important than observing the trend in the total number of defects is the assessment of customer usage. This assessment will indicate how the customer is uncovering the defects. Figure 3 shows that in the first few months the majority of defects were uncovered through execution of single functions as indicated by the triggers of coverage and variation. Then, in month nine, there is an increase in the defects uncovered through more complex testing scenarios as evidenced by the triggers of interaction, sequencing, and software configuration. However, variation is the trigger that predominates in most months. Variation describes finding defects by varying the input parameters of a single function. Figure 2, which graphs activity and components, shows that many of the defects have escaped FVT as illustrated by the presence of the triggers coverage, variation, interaction, and sequencing. These observations point out that improvements must be made in the FVT process to ensure that fewer defects escape. Specifically, the test plans must include more variation testing. This will be done through a combination of education for the testers and an emphasis on this point during inspections of the test plans.

Figure 3 Trigger classification for field defects over time



Summary. ODC has helped us identify the following deficiencies in our process and the actions for improvement:

- The majority of defects were escapes from FVT. The trigger of variation is the most common. Therefore, the action the team will take is to educate testers on the types of testing in FVT and ensure that there are enough variation test cases in the FVT test plans.
- Component q had too many base defects. The team must increase regression testing in this area.
- Component b had the majority of defects. The team will ensure that all changes are clearly documented and also ensure that where code is being changed there is greater emphasis on testing any base code that may be impacted.

Future plans. The team will continue to classify the field defects and use this information to improve the development and test process. The ODC field defects provide the foundation for defining a customer usage profile that can be reviewed by the test teams. The team is also starting to classify all in-process defects found during the development cycle on the new release and compare this classification with the clas-

sified field defects. This comparison will provide more accurate information on the current testing process and provide a focus for future test effort.

Case Study 2: A large middleware project

This case study involves a member of a family of IBM middleware products, available on many operating system platforms. The products are developed by a few hundred people on three continents. They are typically developed by following a waterfall process, with decision checkpoints between the various phases to assess whether the product would move on to the next phase.

The use of ODC by the product teams began at the initiative of the test teams. The teams were in the process of reorganizing themselves to overcome a number of problems. They needed a set of metrics that would allow them to objectively assess their progress and confront some of their problems, which were:

- Overlapping test phases of unit test, functional verification (FV), and system test. In theory, these phases are supposed to be sequential, but often they overlap, sometimes completely. It is the objective of the system test entry checkpoint to determine whether the previous test phases have progressed enough to allow system test to make acceptable progress. Metrics are required to enable this decision to be made based on quantifiable, objective data rather than personal opinions.
- The test teams had developed very large test suites, but had no idea how effective the suites were. They had only two metrics: (1) defects—raising and closure rates, and (2) test cases—execution rates and test case results, i.e., success or failure.

These metrics measured test progress but gave no indication as to the test effectiveness. ODC was selected to provide the objective, quantifiable data necessary for decision support that the team was looking for.

ODC process. The ODC process was piloted by two of the project teams, starting with a series of education sessions. Members of the teams attended a one-hour overview and two hours of detailed education on how to classify defects. A subset of the testers and developers also attended a two-hour class on how to validate the data. A month later a number of individuals received two hours of assessment education, learning how to review the data and how to begin to make use of the data.

The data were validated on a weekly basis by the validation team to ensure that the classification was done correctly. Any mistakes were corrected. The data were subsequently reviewed by the team of assessors and used to assess the state of the project and determine whether any specific actions needed to be taken. The assessments took place at points appropriate to the individual project.

Assessment. During assessment, it was determined that system test could be started and what improvements could be made to system test and function test.

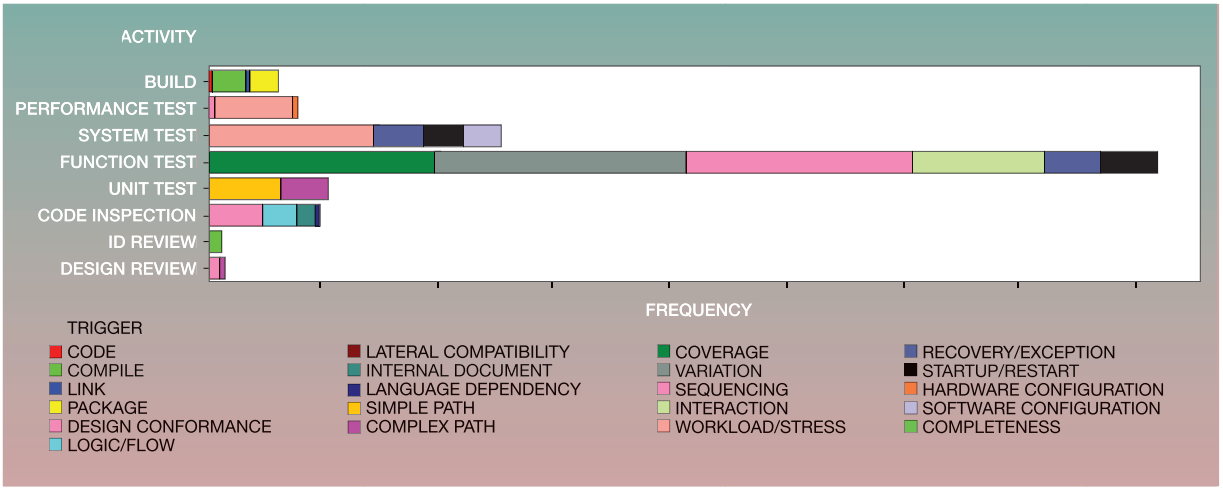
A case for early entry into system test. The use of ODC in this product began approximately three-quarters of the way through its FV phase. Within seven weeks, the project reached the decision checkpoint for system test entry. Although the ODC data were not officially part of the decision-making process at this time, the team was interested in reviewing the data. The data proved very useful in supporting the initiation of system test.[9]

One of the criteria for system test entry was "FV more than 80 percent attempted and not less than 50 percent complete (executed cleanly through to completion)." FV had not met the 80 percent attempted target, but was over 70 percent complete—a very good pass rate. In addition, the FV team believed that the code was very stable. Figure 4, which shows activities and triggers, highlights:

- The fact that functional testing had achieved a broad range of ODC triggers. Coverage, variation, sequence, and interaction testing had all exposed defects. In fact, defects had even been uncovered during this time using the more complex system test triggers such as recovery/exception and startup/restart.
- The fact that functional testing had progressed to the more complex triggers implied that the basic function of the product was solid and ready to progress into system test. This implication supported the "gut feeling" of the FV team.

Despite not having reached the official FV criteria, this information from ODC led the team to believe that the project was ready to start system test. Subsequently, the system test team was able to make rapid progress.

Figure 4 Each activity uses a broad range of triggers to expose defects

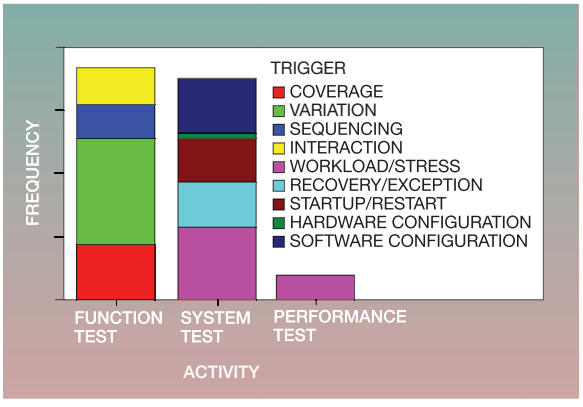


Improvements to system test from analyzing field defects. Figure 4 also illustrates that the majority of system testing is focused on workload/stress. Over 50 percent of the defects were found by workload/stress testing. The chart in Figure 5 was generated after subsequent analysis of the field data from the previous release. The activities and triggers on this chart clearly show customers exposing defects through a broader spread of system triggers than in-process testing had exposed, most notably, software configuration. As a result, the system test team will review its testing in order to broaden its test scenarios, particularly in the area of software configuration.

Improvements to function test from analyzing field and in-process defects. Following the completion of FVT, the test team reviewed the ODC data in order to identify potential areas for improvement. Analysis of Figures 6, 7, and 8 highlighted a potential problem with missing checks.

Figure 6 shows defect type and qualifier for field defects from the previous release of the product. Although the checking defects are not the largest group, a significant proportion of them are caused by missing code. Figure 7 shows the same chart, but for the in-process defects found for the current release. This time, checking defects are in the majority, again with a significant proportion caused by missing elements. Figure 8 shows the defect type and trigger for those in-process defects that were caused by missing elements, indicating that the majority of the missing checks were found by variation and sequencing tests.

Figure 5 Frequency of activities with triggers



This highlights a problem with low-level design stability.

As a result of this assessment, members of the FV team decided to review their test suite with the goal of improving their ability to discover missing checks in the product. Specifically the test team will:

1. Use a code coverage tool to measure the statement and branch coverage achieved by the function test suite. The output will be used by the testers, with the help of the developers, to identify gaps and duplication in the test suites.
2. Classify and then analyze the function test cases in terms of which ODC triggers they represent to

Figure 6 Frequency of defect type with qualifiers in previous release

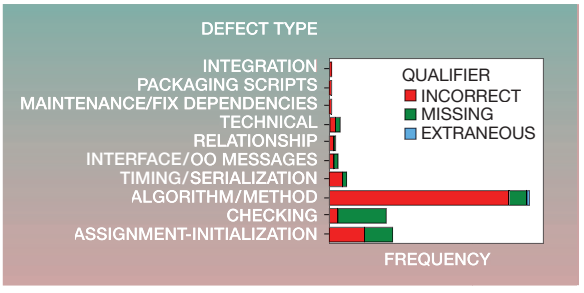


Figure 7 Frequency of defect type with qualifiers for current release

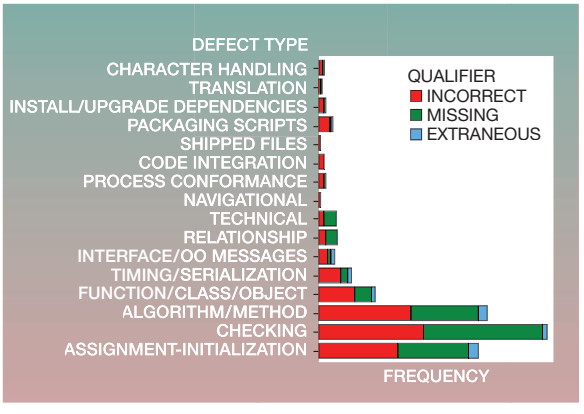
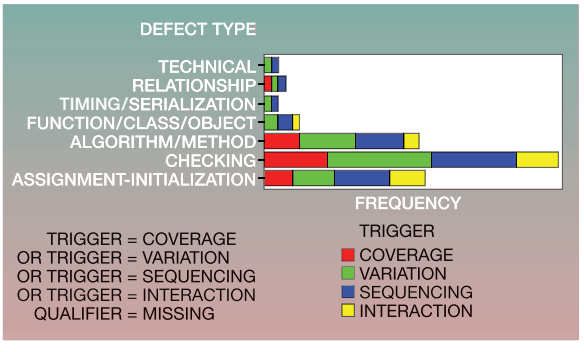


Figure 8 Defect frequency for trigger within defect type



ensure good coverage. The IBM team for the 2000 Summer Olympics in Sydney used this technique as a way of identifying, in advance, whether the test suites were using a broad range of triggers.[10]

It is vital to obtain an assessment of the effectiveness of the FV test suite of this product, especially since the number of test cases has increased significantly over time.

Actions resulting from assessment. Several actions aimed at improving the process have resulted from the ODC assessments. During a typical release cycle, it was only during the final weeks or months that any attempt at prioritizing defect fixes was made, often resulting in areas of testing being blocked for longer than was necessary. The data presented in Figure 9 indicate that many of the defects were of severity 3, generally considered severe enough to require fixing, but not so severe as to actually bring a customer's system down. Although these defects comprised 80 percent of the defects raised, it was impossible to prioritize them. As a result, a new field, "importance" has been added to the defect description. The raiser can specify the importance of obtaining a fix quickly, based upon how much work is impacted. The values for "importance" are:

- 1 meaning high importance: Blocking significant amounts of testing; fix is urgently required.
- 2 meaning medium importance: Blocking some areas of testing, but some progress still possible in the meantime; fix is required as soon as possible.
- 3 meaning low importance: Blocking one or two tests at most; able to make progress easily; fix when feasible.

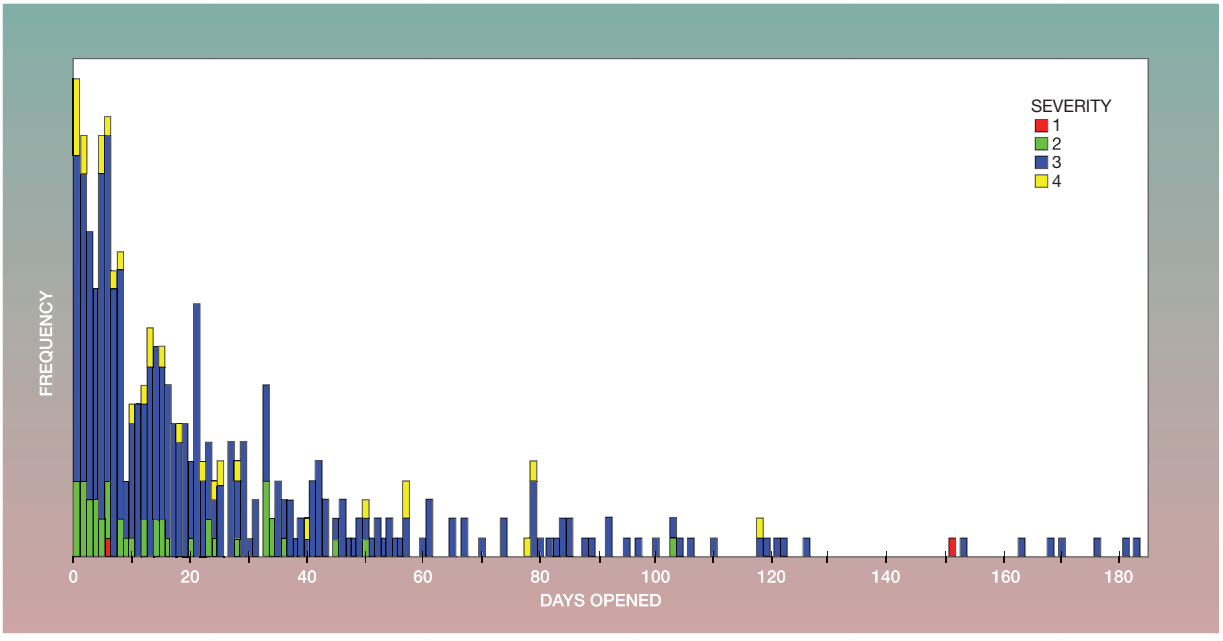
This field has enabled the teams to focus on those defects that are having the most significant impact on their progress.

Another action that resulted from the ODC assessment is that more information is being included in the defects themselves to make it easier for the ODC classified defects to be validated. This action has also been of use to those team members classifying defects. Clearer explanations are provided, resulting in the defects being dealt with more effectively.

Summary. ODC has been in use by this project for just over a year. Within a matter of weeks it proved valuable, helping to assess the risks of passing through a decision checkpoint when the traditional criteria had not been met.

The teams have been able to look at their own data objectively and quickly identify actions to improve their processes and ultimately their product. The actions taken have been reasonable, not requiring huge

Figure 9 Severity of defects



investments in time, money, or effort. The test teams are achieving their goal of improving test effectiveness through the use of ODC. The value of the technology has become evident to the teams and, as a result, use of ODC has been expanded to other projects.

Case Study 3: A small team project

This case study pertains to a small project in which a few people developed a Java**-based software tool that allows users to visualize data related to software development. This tool supports ODC analysis. The primary users of this product are testers, developers, and service personnel in software development laboratories throughout IBM. The challenges faced by this development team were:

- Lack of development resources and limited experience—The team was inexperienced with object-oriented programming, with the Java language, and with the development tools. In addition, there was little experience with testing. The team members relied on a checklist of functional areas and their own sense of what they thought should be tested.
- Schedule pressures—Program bug fixes and increasing functionality were always in demand, which led to an aggressive schedule.
- Customer need—This tool was required for the successful widescale deployment of ODC. Therefore, it was imperative that a high-quality, stable product be developed in order to maximize the success of ODC.

ODC was crucial for effectively managing resources, gauging the stability of the product, and guiding the team in efficient testing of the product.

ODC process. Although ODC has been practiced since the first release of this product, this paper will focus on the improvements made in release 1.3. By the time this release was available, the team was experienced at ODC classification. Since the project manager had been an ODC consultant for the past five years and had extensive experience performing validation and assessments, there was no learning curve for performing the assessment.

Assessment. Initially, version 1.3 was to be released in the August–September 2000 time frame. It was primarily a release to fix a software bug with little or no additional functionality. A month before it was

Figure 10 Frequency versus activity with triggers

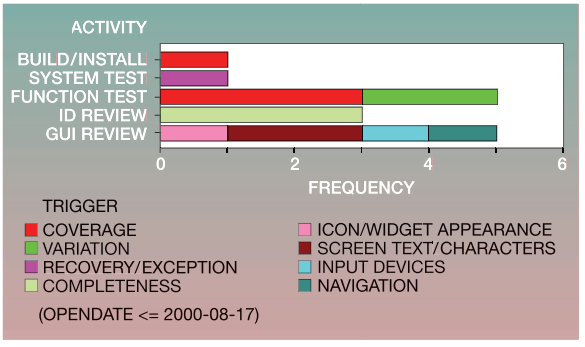
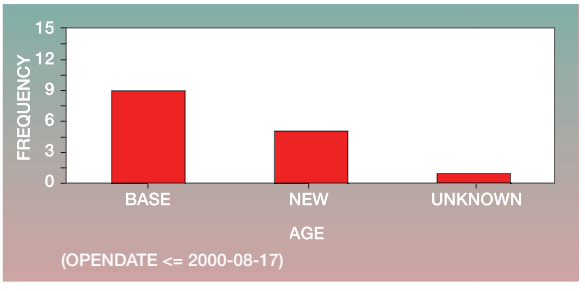


Figure 11 Number of defects by age



scheduled to be released, team members voiced their confidence that the product was ready for release. They believed that since these were only bug fixes, basic functionality was unchanged. They had already spent several weeks testing new code, so they felt there should be no schedule slip, and the product could be released as originally planned. However, when the ODC assessment was performed, it was evident the product was *not* ready for release.

Figure 10 shows activity and triggers. The first point to note is that there are only 15 defects. Even for this small project, it was expected that function and system test would have uncovered more. Second, the number of defects found during function testing was equal to the number found during graphical user interface (GUI) review. Historically in this product, significantly more defects are found in function testing than in GUI review. Third, the defects found during function testing were exposed through execution of single functions, as evidenced by the triggers of coverage and variation. It was easy to uncover these defects through the click of a button or execution of a simple command. In general, a product is well-tested when defects have been uncovered through a wide range of triggers for each activity. Although GUI review had uncovered defects through a variety of triggers, function and system test had not. Coverage and variation represent only two of the four possible triggers available for this activity. Before releasing the product, testing needed to progress to uncover defects through the more complex triggers. Specifically, more defects were expected to be uncovered through sequencing in function test and through workload stress and software configuration in system test.

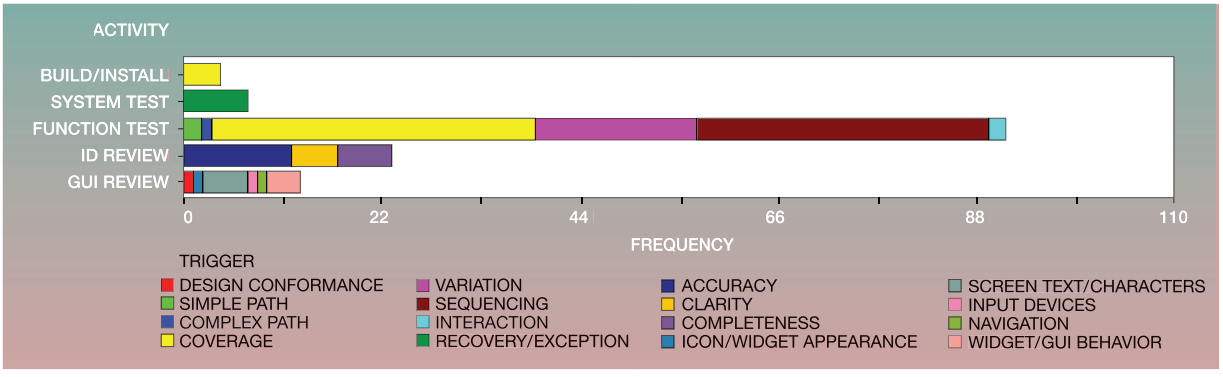
Figure 11, which shows the number of defects for the ODC attribute age, further supported the conclusion that the product was not ready for release. It shows that nine defects (60 percent) were found in base code—defects that existed prior to this current release. These are defects that had been dormant. They should have been caught in the previous release but were not. Therefore, not only have defects been uncovered simply by executing single functions, but the majority of those defects were found in old code, not new. This information further supported the team's suspicions that the code had not been adequately tested. Therefore, the team took the only possible acceptable action under the circumstances. The testing efforts were increased and the release date was postponed.

In order to improve test effectiveness, several additional test cases were created in the areas of variation, sequencing, interaction, software configuration, and workload stress. The result of the additional testing is shown in Figure 12, which shows ODC activity by triggers.

By December, 138 defects had been uncovered, in comparison to the 15 detected previously. The increased testing effort had paid off. Next, the trigger chart was reviewed. Previously, the single function triggers made up 100 percent of the testing. By extending the schedule, however, the multiple function triggers of sequencing and interaction grew to 37 percent of the scenarios that exposed defects. In addition, we found seven more defects uncovered through recovery/exception—a system test trigger. The testing had progressed from uncovering defects through simple scenarios to the more complex.

Two exposures still existed in workload stress and software configuration. Limited testing had been performed but had not uncovered any new defects. How-

Figure 12 ODC activity by triggers



ever, because of a lack of resources, it was decided that this exposure was acceptable.

Summary. At the end of the extended test cycle, the product was much more stable than it had been in August. Most of the defects had been found, and few escapes to the field were expected. In fact, that is what the team found. Four defects were reported in January, and one of those was a build problem. However, if the ODC assessment had not been done in August and the product had been released as originally planned, the customers clearly would have been affected as they uncovered many of those defects exposed between September and December. Customer satisfaction would have been low as they lost confidence in the team's ability to develop a quality product. In addition, the development team would have been put into the "fire-fighting" mode. Instead, the team was able to assess the effectiveness of testing through exploiting ODC classified defect data. Test plans were then created to specifically target the multiple function scenarios in sequencing that had been lacking. These steps made it possible to improve testing efficiency while mitigating weaknesses, allowing the team to deliver a stable, high-quality product to customers.

It is important to note that as mentioned previously, ODC provides data-based decision support. The assessment, which took an hour to do, and the subsequent decision to extend testing, was based on objective data. Once the team reviewed the ODC attributes displayed in the charts, there was no debate about the readiness of the product. The manager did not have to agonize over two opposing views while trying to decide the best course of action. The state of the software was clearly shown in the charts, and the actions needed were obvious to the team.

This experience convinced the development team that it did not take much time or effort to accurately measure progress during testing and highlight the risks. Despite the fact that the team was small, assessment of the classified defect data was done quickly, and specific actions were taken to avoid a potentially disastrous situation. Best of all, the feedback received from customers is that this release is the most stable yet!

Conclusion

We have described the experience of teams using ODC in projects for three different products. Each team was able to reach its objective of improving test effectiveness with a minimum impact on organizational resources by using ODC. In all three projects, defects had been collected and used prior to adopting ODC. However, ODC enabled the teams to utilize this rich collection of defect information to improve the quality of the respective product in an objective manner.

The first case study is a high-quality product with a goal of decreasing customer-reported defects in order to increase customer satisfaction while decreasing warranty costs. Test effectiveness was improved by identifying the underlying problems that led to customer-reported problems. Classifying triggers from defects that escaped to the field provided the link necessary to improve specific areas of test and

development. The actions this team adopted focused on strengthening test plans, improving FVT education, increasing regression testing for the identified components, and improving documentation.

The second case study used defects found in function and system test data to improve their effectiveness. This study illustrated an example of relieving schedule pressures when the ODC-classified defects indicated the product had progressed sufficiently enough to advance to the next step in testing. The team was able to initiate system testing earlier than scheduled because the ODC assessment provided the information that the exit criteria for earlier activities had been reached. As in the first case study, this team also identified areas of testing that were weak. The team then adopted specific actions to target those areas. The actions they implemented included use of a code coverage tool and ODC classification of function test cases to make sure they had the broad range of scenarios necessary.

The third case study illustrates how a small team with few resources can benefit by identifying specific weaknesses in testing. The team used defects found in house shortly before the project was to be released. ODC was used to identify the multiple function scenarios in function test needing improvement before releasing the product. In this case, the team was able to quickly evaluate test effectiveness and implement correct actions to strengthen the targeted triggers.

ODC data collection can be initiated at any point in the software development process. All three case studies provide examples of benefits from analyzing the ODC-classified defects at different points in the software life cycle. The actions implemented were reasonable and feasible and did not require major investments in resources to realize the goal of improving test effectiveness.

In addition to the positive impact of ODC that has been shown here, there are many results that are not easy to measure but nonetheless provide major gains. These are the subtle changes that occur in the testers and developers themselves as their skills improve, because they are now able to view their work in a new and objective manner. Testers learn to consider test cases in terms of complexity of triggers. They also learn about the trends that should be seen as they progress through the scheduled phases and become more alert to any deviations in those trends. They become adept at comparing triggers exposing defects found by customers with triggers exposing defects in their process. Testers have raised their level of knowledge in testing effectiveness which results in better quality in the product and a stronger process. Ultimately, this increased knowledge not only produces software that is higher in quality but is less expensive to develop—a goal all software organizations strive to attain.

Appendix A: Scheme version 5.11 for design and code

Please see http://www.research.ibm.com/softeng for more information and examples.

Attributes classified when a defect is opened:

Activity—This is the actual activity that exposed the defect. For example, during the scheduled phase of system test, a defect occurs when you click on a button to select a printer. The phase is system test but the activity is function test because the defect surfaced by performing a function test-type activity.

Triggers—These are the environment or condition that had to exist for the defect to surface. Triggers describe what you need to do to reproduce the defect.

Impact—The impact refers to the effect the defect had on the customer if it had escaped to the field or the effect it would have had if not found during development.

Mapping of the activities to triggers shown in Table 1 is for illustrative purposes only. In general, for defects found in process, pick the activities in the process being used and map the appropriate triggers from the list. For field defects, select the triggers first, which then automatically maps back to the activity that most likely would have found the defect if it had been discovered in house. The list of triggers represents an adequately complete set and should not be combined, have additions, or deletions.

Attributes available when the defect fix is known:

Target—What is being fixed: design, code, documentation, etc.

Defect type—The nature of the actual correction made

Defect qualifier (applies to defect type)—Captures the element of nonexistent, wrong, or irrelevant implementation

Table 1 Attributes identified when a defect is uncovered

Activity and Triggers				Customer Impact
Inspection	**Unit Test**	**Function Test**	**System Test**	
Design conformance	Simple path	Coverage	Workload/stress	Installability
Logic/data flow	Complex path	Variation	Startup/restart	Serviceability
Lateral compatibility		Sequencing	Recovery/exception	Standards
Backward compatibility		Interaction	Hardware configuration	Integrity/security
Language dependency			Software configuration	Migration
Concurrency			Blocked test (formerly Normal Mode)	Reliability
Internal document				Performance
Side effects				Documentation
Rare situations				Requirements
				Maintenance
				Usability
				Accessibility
				Capability

Table 2 Attributes identified when a design or code defect is fixed

Target	Defect Type	Qualifier	Source	Age
Design/code	Assignment/initialization	Missing	Developed in house	Base
	Checking	Incorrect	Reused from library	New
	Algorithm/method	Extraneous	Outsourced	Rewritten
	Function/class/object		Ported	Refixed
	Timing/serialization			
	Interface/OO messages			
	Relationship			

Source—The origin of the design/code that had the defect

Age—The history of the design/code that had the defect

The attributes identified when a design or code defect is fixed are shown in Table 2.

In this paper, we have focused on defects found in design and code and so have not included the target of information development, build/packaging, National Language Support, and their values.

Appendix B: Triggers for graphical user interface review activity

Table 3 shows the list of triggers that can expose defects while reviewing a graphical user interface consisting of graphical elements such as buttons, labels, and scrollbars. Please see http://www.research.ibm.com/softeng for more information.

Table 3 List of triggers in GUI review

Triggers
1. Design conformance
2. Icon/widget appearance
3. Screen text/characters
4. Input devices
6. Navigation
7. Widget/GUI behavior

Appendix C: Some typical assessment topics and associated defect attributes

Table 4 is a list of typical topics of interest to a software development organization and the defect attributes that are useful in addressing them. An assessment would usually consist of looking at the defect distribution across these attributes as well as evaluating the relationships of the attributes to each other.

Table 4 Typical topics and defect attributes

Topic	ODC Attributes	Non-ODC Attributes
Measuring test effectiveness	Activity, trigger, qualifier	Open date, severity, component, phase
Evaluating product stability	Defect type, impact, qualifier, source, age	Open date, severity, close date, component
Identifying strengths and weaknesses in design and code	Type, qualifier	Component
Evaluating customer usage	Impact, trigger, defect type, qualifier	Open date, severity, component
Measuring progress	Activity, trigger	Severity, component, phase

**Trademark or registered trademark of Sun Microsystems, Inc.

Cited references

1. M. Fewster and D. Graham, *Software Test Automation*, ACM Press Books, New York (1999), pp. 203–208.
2. R. Chillarege, I. Bhandari, J. Chaar, M. Halliday, D. Moebus, B. Ray, and M. Y. Wong, "Orthogonal Defect Classification—A Concept for In-Process Measurement," *IEEE Transactions on Software Engineering* **18**, 943–956 (November 1992).
3. J. K. Chaar, M. J. Halliday, I. S. Bhandari, and R. Chillarege, "In-Process Evaluation for Software Inspection and Test," *IEEE Transactions on Software Engineering* **19**, 1055–1069 (1993).
4. B. Casey, E. Kaldon, J. Sun, and W. Watters, "Application of Defect Analysis Techniques to Achieve Continuous Quality and Productivity Improvements," *ICC '91*, pp. 1450–1454.
5. R. B. Grady and D. L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Chapter 8.1, Metrics Worth Collecting, Prentice-Hall, Inc., Englewood Cliffs, NJ (1987), pp. 96–104.
6. K. Bassin, T. Kratschmer, and P. Santhanam, "Evaluating Software Development Objectively," *IEEE Software* **15**, 66–74 (November/December 1998).
7. K. Bassin and P. Santhanam, "Use of Software Triggers to Evaluate Software Process Effectiveness and Capture Customer Usage Profiles," *Proceedings of the 8th International Symposium on Software Reliability Engineering*, Case Studies, IEEE Computer Society Press, Los Alamitos, CA (1997), pp. 103–114.
8. A. Sharp, *Software Quality and Productivity*, Van Nostrand Reinhold, New York (1993), pp. 347–361.
9. N. Bridge, *Software Quality Orthogonal Defect Classification Using Defect Data to Improve Software Development*, American Society for Quality, Software Division No. 3 (1997–1998), pp. 1–8.
10. K. Bassin, R. Biyani, and P. Santhanam, "Evaluating the Software Test Strategy for the 2000 Sydney Olympics," *Proceedings of the Twelfth International Symposium on Software Reliability Engineering*, Hong Kong (November 2001).

Accepted for publication August 30, 2001.

Mark Butcher *IBM United Kingdom Laboratories, Hursley Park, Winchester, Hampshire, SO21 2JN, United Kingdom (electronic mail: Mark_Butcher@uk.ibm.com).* Mr. Butcher is a senior software engineer at the IBM Hursley Development Laboratory. For nearly 16 years, he has worked on a number of large software projects including Graphical Data Display Manager (GDDM®), OS/2®, and MQSeries®, in a variety of test, development, and build roles. In addition, he is active in the ODC and Code Coverage process improvement activities at Hursley. He studied computer science at Thames Polytechnic (now Greenwich University), graduating in 1985.

Hilora Munro *IBM United Kingdom Laboratories, Hursley Park, Winchester, Hampshire, SO21 2JN, United Kingdom (electronic mail: hilora@uk.ibm.com).* Ms. Munro has 20 years experience in the development of computer software, the last four being specifically within software testing. She received a B.Sc. degree in computer science from the University of Strathclyde, Scotland, and has subsequently been involved with a variety of software solutions and has worked in all areas of software development including design, development, test, and support. During the last few years, Ms. Munro has been actively involved in the drive to improve testing within the IBM test community. This has included the use of metrics, the deployment of ODC, and improving knowledge sharing within the test community.

Theresa Kratschmer *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (electronic mail: theresak@us.ibm.com).* Ms. Kratschmer is a software engineer with 16 years of experience developing software for real-time, graphics, and database applications. She has been with the Center for Software Engineering at IBM Research since 1996. Currently, she does research, deployment, and tool development related to defect analysis and testing technologies. She earned a B.S. degree from Cornell University in 1981 and an M.S. degree in advanced technology/computer science from the State University of New York at Binghamton in 1985.