Metrics to evaluate vendor-developed software based on test case execution results

by K. Bassin
S. Biyani
P. Santhanam

Various business considerations have led a growing number of organizations to rely on external vendors to develop software for their needs. Much of the day-to-day data from vendors are not available to the vendee, and typically the vendee organization ends up with its own system or acceptance test to validate the software. The 2000 Summer Olympics in Sydney was one such project in which IBM evaluated vendor-delivered code to ensure that all elements of a highly complex system could be integrated successfully. The readiness of the vendor-delivered code was evaluated based primarily on the actual test execution results. New metrics were derived to measure the degree of risk associated with a variety of test case failures such as functionality not enabled, bad fixes, and defects not fixed during successive iterations. The relationship of these metrics to the actual cause was validated through explicit communications with the vendor and the subsequent actions to improve the quality and completeness of the delivered code. This paper describes how these metrics can be derived from the execution data and used in a software project execution environment. Even though we have applied these metrics in a vendor-related project, the underlying concepts are useful to many software projects.

Appropriate use of software metrics is vital to any software project.[1–3] From the software release management point of view, there have been examples of metrics used in various companies.[1,4–6] These metrics help track aspects of an ongoing software project, such as changing requirements, rates of finding and fixing defects, and growth in size and complexity of code. From the testing point of view,[6,7] the metrics typically focus on the quantity and quality of software, the progress of testing, and the effectiveness of testing. Examples of typical metrics include product or release size over time, defect discovery rate over time, defect backlog over time, test progress over time (plan, attempted, actual), and percentage of test cases attempted. Although each of these metrics has merit, overall they were insufficient to address the special requirements imposed by the IBM project for the 2000 Summer Olympics, held in Sydney, Australia, particularly with regard to evaluating components developed by an external vendor.

The status of these metrics is typically reported either as measured against expected goals in the schedule or against a previous release with similar content for comparison. In the case of the 2000 Summer Olympics, data from prior Olympics were either not available or not sufficiently similar in content to be useful. Thus, the assessment of progress being made, the stability of the product in terms of functional content and code quality, and the effectiveness of test activities had to depend on being measured against expected goals in the current project rather than on historical evidence.

The trend toward increased outsourcing of software development demands the regular use of new metrics and methodology to assure adequate quality and schedule integrity. One of the challenges a vendee

Table 1 The scope and complexity of Olympics events

	Winter Games	Summer Games
Number of medal events	68	300
Different sports	7	28
Competition venues	12	39
Accreditation venues	25	30
Concurrent events	9	7
CIS (Commentator Information System) screen formats	535	800
Languages supported	English, French, Japanese	English, French
Timing vendor	Seiko (Nagano)	Swiss T (Sydney)
Timing performance	Subsecond	Subsecond
Report types	2,000	3,000
INFO users	84,000	260,000
INFO terminals	1,500	2,000
News records	4,000	10,000
Biographical records	8,000	35,000
Historical records	500,000	1,500,000
Average INFO requests per day	710,000	6,500,000
Internet hits per day	56M peak	874.5M peak

organization faces is the evaluation of the delivered code in terms of functionality, performance, etc. Because of the implicit risks in software projects, the contractual commitments for quality and completeness are generally at a high level, and typically the vendee organization defines and executes its own system or acceptance test to validate expectations. Often the code is delivered in an incremental fashion through multiple iterations, and many of the detailed operational data from the vendor are not available to the vendee.

This paper presents a set of metrics that can be derived by the vendee organization based on the evaluation of its own test cases on code delivered to it, without depending on a sparse amount of information typically provided by the vendors. These metrics can significantly enhance the strength of vendee organizations in accepting vendor code by giving them objective measures of the vendor's performance. Though problems typical to vending software development are known, there appears to be nothing in the literature describing the derivations and validation of quantitative metrics from actual test case execution records.

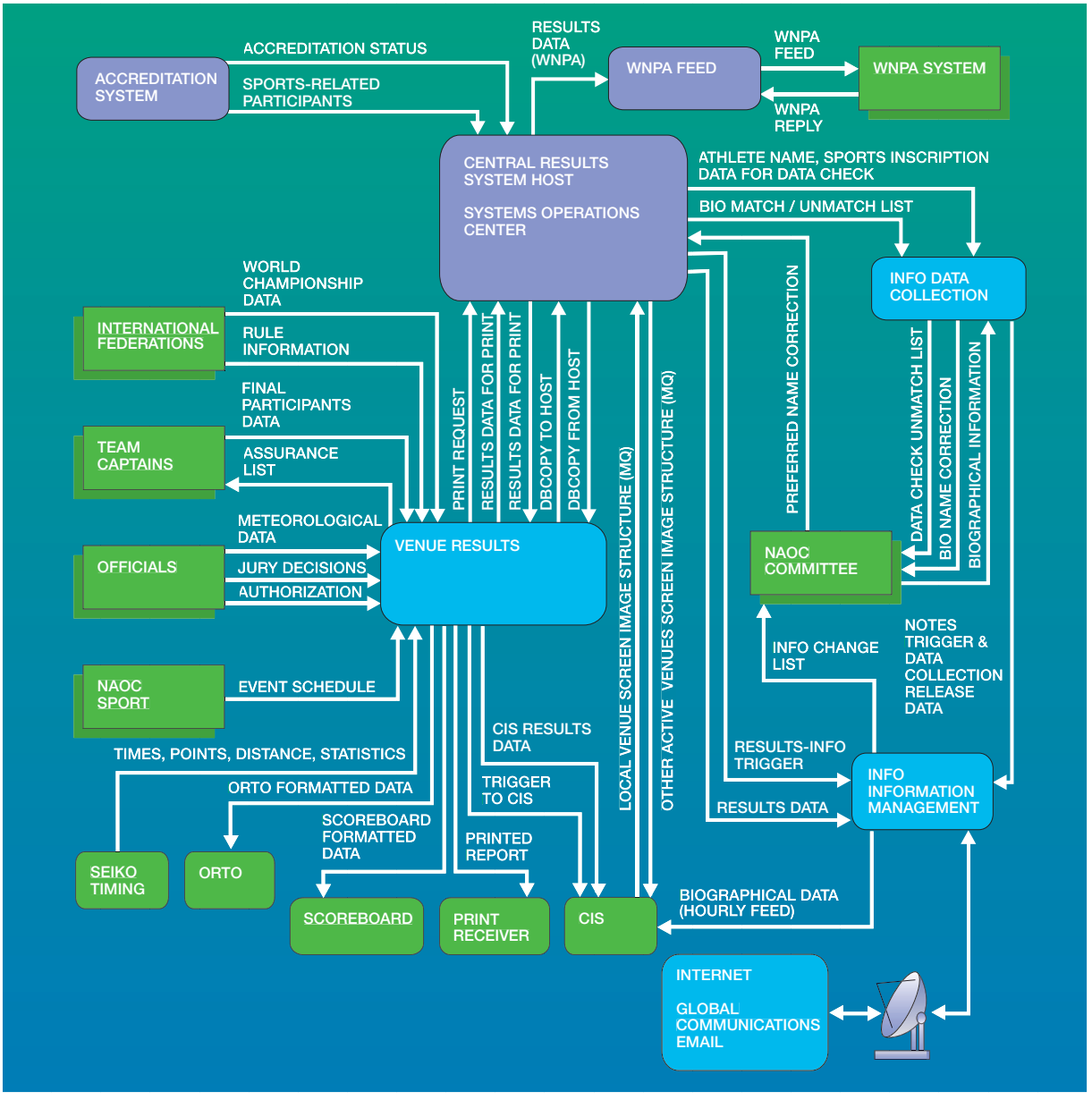
Background of project

The 2000 Summer Olympic Games were considered to be the largest single sporting event in the world. Table 1 shows some of the facts that reflect the scope of this endeavor. In addition to capturing and reporting event results, IBM's role in those Summer Olympics had expanded to include managing and monitoring the information technology (IT) infrastructure and many other aspects of a total of 39 venues. This role included everything from keeping track of each of the 15836 athletes, coaches, and officials to managing food service for the 5.5 million spectators at the games. Preparing for the Olympic Games has been described as building a billion dollar company with 200000 employees in less than four years, then tearing it down in 90 days. One critical challenge was the development, test, and integration of the venue results systems. The diagram in Figure 1 is a representation of the overall architecture and the complex data flows and interfaces to and from venue results. These systems are responsible for capturing and collecting the results of each sporting event and reporting these results to 15000 accredited media personnel (journalists and broadcasters), sport officials, coaches, participants, and 3.7 billion remote spectators.

Project description. The focus of this paper is on the development and testing of the venue results components. These components were developed by a vendor in Spain, and the testing and integration of these and other components were executed by the IBM Games System Center (GSC) in Spain. Specific responsibilities were defined at the beginning as follows:

- Design of the venue results components was primarily the responsibility of the vendor, with IBM providing input and participating in design reviews.
- The vendor was responsible for coding and early

Figure 1 Major applications and critical data flows and timing requirements for 2000 Summer Olympics (trigger in this figure not related to ODC trigger)



testing (including code inspections, unit test, and simple functional testing for each sport).

- GSC was responsible for multiple test activities: verification test (straightforward tests intended to verify that the component was sufficiently stable to proceed to subsequent test activities), function test (thorough testing within each component in terms of basic functional requirements), function integration test (testing of integrated components to ensure functional interaction and interfaces meet requirements), and system integration test (testing of fully integrated system).

- Acceptance testing, performance test, and a more extensive and final system integration test were conducted by the team in Sydney, Australia.

Late in 1998 it was decided that advanced analysis and decision support, which included the use of Orthogonal Defect Classification (ODC),[8–10] was required for the evaluation of progress and risk associated with testing and integrating components.

Metrics used prior to work reported in this paper. Many tools and methodologies to evaluate quality were already in place. Quality plans had been developed to define acceptable outcomes. Inspections of a wide variety of artifacts, including requirements, design, code, use cases, and object and class definitions, were defined as key process steps. Status reviews were an integral part of all major checkpoints during the test and integration activities. Many process and product evaluation metrics were already being used by the vendor and GSC. The vendor responsible for developing the venue results components was highly respected and experienced in delivering these types of applications, having participated in such high-profile (albeit single venue) international events as the Wimbledon tennis competition. The vendor had implemented a set of metrics based on Rational Rose** technology, including such object-oriented measurements[11] as attributes per class, associations per class, and number of children per class. In addition, they were systematically measuring code complexity primarily by means of cyclomatic complexity.[12] Given these considerations, why was there a perceived need for additional decision support?

Need for additional metrics. One concern was the cyclomatic complexity metrics used by the vendor. These metrics were able to detect variances in complexity and implied risk; however, the measurements could not be taken until after the design and code were complete, resulting in the risk that there would be insufficient time to address exposures and still meet the demanding schedule. Of greater significance was the fact that applying these measurements to the venue results components independently from the rest of the system precluded an evaluation of how the components would behave when integrated. This fact was particularly relevant with regard to the multivenue Olympics system, whose architecture and components consisted of many platforms, components, and technology.

A key inhibiting factor was that the plan called for not only incremental deliveries of the vendor product overall but for individual functions for each sport component to be delivered incrementally on independent schedules. This delivery routine had implications in terms of defining and executing a test plan based not on the full functional content of a sport product upon delivery, but rather on those functional elements that were presumed to be part of the delivery. Thus, it was extremely difficult to define an expectation in terms of conventional metrics with regard to the overall product.

The extent to which defect backlog over time was a concern had implications with regard to both product stability and test effectiveness, and it ultimately could inhibit progress. Unfortunately, it was difficult to identify a trend for a model that called for defect fixes to be held for relatively long, inconsistent intervals and delivered in large batches, as was the case in the 2000 Summer Olympics. The ability to utilize an S-curve model to track progress was also impacted by the frequency and interval inconsistency of incremental code and fix deliveries. An S-curve model is ideal for an environment in which defect discovery is random and test effort is steadily executed. In the 2000 Summer Olympics, testing for each sport would be halted for long periods of time until the new increment had been received.

Organizational churn was also a factor. Up to 80 percent of the test team members changed every few weeks as attention was directed to each individual sport and corresponding specialists were brought in. Thus there was a relatively small core team of test experts able to provide continuity in terms of assessing status, identifying weaknesses, and putting actions in place to address them.

Because of the severe schedule stress, interaction between the technical team and the analyst was kept at an absolute minimum. This limitation meant that the evaluations had to be based almost entirely on what could be derived by available data. Fortunately, the collection of data by the test team was very thorough and, with a few exceptions, complete. The following section describes the data that were collected during the project and identifies those elements that were available for the specific analysis described in this paper.

Data

Although there were gaps in terms of defect data, as previously described, there was nevertheless a great deal of valuable data collected throughout the

project. These data originated from three sources: vendor, GSC, and test activities.

Vendor metrics, data, and measurements. The vendor captured measurements that enabled its analyst to assess productivity and complexity, and to project quality for each sport. The vendor initially provided GSC with complexity values based on Rational Rose as well as a projection of quality for each sport, although not the underlying data. The measurements and projections were helpful in the initial planning stages. As time progressed, however, decisions were made to deviate from the original plan in terms of technology and function, creating a dynamic environment difficult to size and measure accurately. Not any of the actual results for defect detection and removal activities, the defect record counts, nor the defect records themselves were made accessible to GSC. Additionally, information regarding the required fixes for defects that were discovered and reported by IBM (GSC and Australia) was not available to IBM.

GSC metrics, data, and measurements. The quality plan of GSC described many process- and product-related metrics designed to ensure a high standard of quality in terms of all deliverables as well as maintaining the planned schedule.

The process-related metrics were primarily targeting the assessment of current status and, to some degree, anticipating changing needs for resource and skill. Because of the nature of the Olympics project (i.e., it was a single engagement, executed once), continuous improvement over multiple generations of a product was not a priority in terms of the process-related metrics. It was, however, important to be able to identify process weaknesses and exposures over the series of incremental product deliveries. Process metrics included summary-level evaluations of effort, staffing, schedule, successful completion of an activity, cost, and productivity. These metrics could be used early in the project to anticipate resource and skill requirements, and during the project they could be used to provide executive-level views of status. They proved to be inadequate, however, at more granular views in isolating cause and effect and providing sufficient insight to enable management and technical teams to address deficiencies quickly.

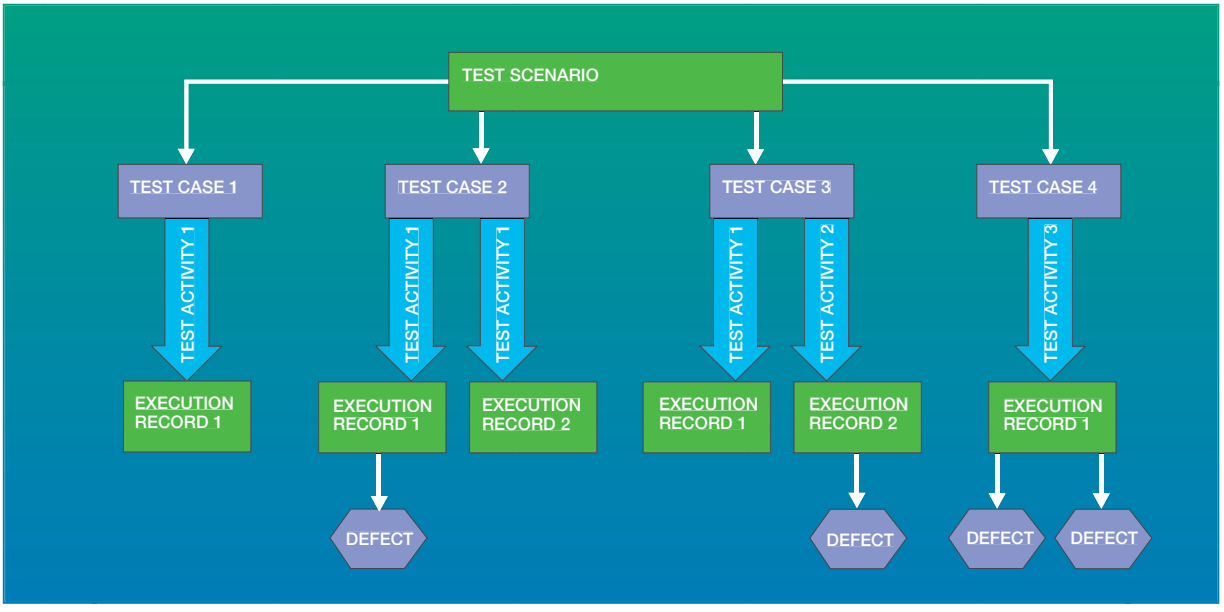
Product-related metrics included many that are typical of software development activities such as system size, defect density, defect removal rate, incoming defect rate, and defect closure rate.

The success of these metrics was, by definition, greatly affected by the availability of accurate information regarding component and increment sizes. As the project content and design evolved, the original form of these metrics was inadequate for keeping pace with the changes. The set of metrics was enhanced through the implementation of ODC,[8–10] which enables evaluation based on distributions of semantic attributes rather than defect rates, and as such are not as dependent on the accuracy of product size measurements as metrics based on defect rates per kloc (thousands of lines of code). Unfortunately, since defect data were not complete, it was necessary to find other measurements that would "fill in the gaps" and improve the accuracy of the analysis.

Test data. Fortunately, the data collected in planning and executing test activities were extensive—much more so than a typical project, and more thoroughly than many large, complex projects. Considerable effort was invested in the early stages to plan and evaluate the comprehensiveness of the various test activities. ODC was used innovatively[13] in classifying and analyzing test cases. This use made it possible to evaluate the completeness of the planned effort across many dimensions (for example, depth and variety of testing relative to each sport, event reporting with regard to media and content requirements, and adherence to test strategy) and to set expectations for test execution. The investment in terms of capturing data in test cases and their associated execution and defect records proved to be invaluable in terms of analysis that could be performed throughout the project.

Categorization. All data were organized first by a main category based on 39 competition venues plus 12 general information categories. The main categories were subdivided into *scenario groups*. Typically, these groups represented different events within a sport (such as men's singles within tennis), as well as certain additional categories, such as installation, configuration, and dynamic information. Another categorization, called *function type*, was used to describe the functionality under test, such as different types of data entry, and reports before, during, and after the event. Many scenarios and test cases were executed during multiple test activities (i.e., verification test, function test, etc.) to ensure that functions could be successfully invoked as a component became more complete, the configurations increased in complexity, and the environment evolved to be more realistic.

Figure 2 Hierarchy and relationship of test scenarios, test cases, execution records, and defects



Hierarchical structure. The planned test effort, as well as the current status of successful execution, can be understood and evaluated by examining the hierarchy of records representing it. Figure 2 depicts this hierarchy and illustrates some typical sequences. Test Case 1 shows a successful execution under Activity 1 with no defects found. Test Case 2 shows an unsuccessful execution under Activity 1 with its associated defect. A subsequent execution under the same activity verifies the fix. Test Case 3 shows that the first execution under Activity 1 was successful, whereas the execution in Activity 2 failed. This failure may be caused by a more complex environment during Activity 2 or by regression of the code. Test Case 4 was executed for the first time in Activity 3 and revealed two defects. In Test Case 3 and Test Case 4, for simplicity we have not shown the execution records associated with the successful retesting of the fix.

Test case data. The data collected for each test case included the categorization outlined above, the test activities during which the test case was to be executed, and some ancillary information such as the author and last modification time. In addition, an ODC trigger[8] was assigned to each test case. ODC defines a trigger as the environment, catalyst, or special conditions that must exist in order for a defect to surface. The description or instruction normally associated with a test case includes this information. For example, the test setup would specify a certain hardware or software configuration, or an explicit condition such as simulation of a heavy workload. These descriptions map to specific definitions of ODC triggers. Thus, the focus or intention of the test case is understood when it is defined and can be represented by an ODC trigger at that time. Later during test execution, if a defect was revealed, the trigger would be automatically copied to any defect records associated with that test case. The trigger values in the defect records were validated against the description of the failure for accuracy. Reference 13 describes this use of ODC in the Olympics project, providing details of the method, validation, and results achieved.

Execution data. When a test case execution was attempted for a given release (e.g., increment delivery), an execution record was created. Each execution record included an execution history consisting of an entry for each time the test case was attempted within the same release and the result of each attempt. These entries would reflect the date and time of the attempt and the status (pass, completed with errors, fail, not implemented, or blocked). A status of "failed" or "completed with errors" would result

in the generation of an associated defect record. A status of "not implemented" indicated that the test case attempt did not succeed because the targeted function had not yet been implemented. "Blocked" status was used when the test case attempt did not succeed because access to the targeted area was blocked by code that was not functioning correctly. Defect records would not be recorded for these latter two statuses. Additional information in the execution record included the test activity, pointers to any defects found during execution, and other ancillary information.

Defect data. For each defect found during testing, the associated execution and test case information was recorded along with defect description, severity, priority, status (open, closed, canceled, etc.), open date, close date, etc. In addition, defects were classified according to ODC, providing information about the semantics of the defect, including test activity, trigger, and impact. Trigger has been previously defined. Activity refers to the task being performed when the defect was uncovered (for example, function test, performance test, or system test). Impact reflects the expected affect the defect would have on an end user should it have escaped the test (for example, reliability or usability). Aside from these attributes known when the defect was uncovered, additional defect attributes were classified based on the fix information: target, defect-type, and qualifier. ODC target reflects the high-level view of what needed to be fixed (such as design, code, or documentation). Defect-type expresses the complexity and scope of the fix, as well as its characteristics (for example, simple defects such as initializing a variable or complex defects such as timing or serialization). Qualifier describes the defect-type in terms of whether the defect was an incorrect, missing, or extraneous element. Since fix information was not provided by the vendor, it was necessary to classify fix attributes using the discovery information and a relationship model based on comparable projects and experience.

Data quality. The overall quality of the data was very good. As is often the case with large data sets, not all records were "clean." Some invalid values crept in, and it was necessary to apply some filters to eliminate them. Defect records were validated for accuracy. Inconsistency in date formats used was another problem area. Although the time stamps were generated automatically by the test environment, variation in local settings caused a mix of two-digit and four-digit years as well as a mixture of U.S. format (month-day-year) and European format (day-month-year) often within the same sport. Although this problem could have been avoided with better input controls, avoidance is not always easy in a multinational environment, and careful handling of date data after the fact solved most of the problems.

Metrics

The collection of data is critical, but the value the data provide can only be achieved through use of the data in analysis and feedback. In addition to overall project goals (for example, quality and delivery requirements), the depth of captured data made it possible to define goals and associated metrics to support technical decisions.

Goals associated with the metrics. Through analysis of test case, execution, and defect records it was possible to define a new set of metrics that could successfully relate cause and effect relationships and isolate exposures and opportunities. The set of metrics was applied to the overall project but also to individual sports or test activities, making it possible to identify the critical opportunities or exposures by team. Goals that could be supported included:

- Establish clear and precise checkpoint exit criteria for each sport increment.
- Quickly identify actions that could be put in place to target exposures and mitigate risk.
- Track and evaluate the degree to which the executed actions were successful with regard to schedule and effectiveness.
- Anticipate the impact of projected product weaknesses and rework on the current and subsequent test activities.
- Enhance the team's ability to successfully negotiate the resolution of exposures with the vendor.

Metrics based on test data. To capture a good overview of the test execution process, both planned and executed, we extracted information for each main category, subcategorized two ways: by scenario group and by function type. The following discussion describes the basic and derived metrics that were used.

Metrics based on counts. The first group of metrics, presented below, represents a sample of basic record counts used. Some were useful in the initial evaluation of the planned test effort, as well as during test to evaluate progress and risk. The related metrics on which we focused included:

- Number of scenarios
- Number of scenarios by function type
- Number of test cases defined
- Number of test cases executed
- Total number of execution records (over multiple releases)
- Number of defect records
- Number of test cases with failures but no associated defect records

The *percentage of test cases attempted* was used as an indicator of progress relative to the completeness of the planned test effort.

The *number of defects per executed test case* was an indicator of the quality of code as it progressed through the series of test activities. Since the comprehensiveness of the planned test cases had already been validated,[13] this metric could also be used to evaluate the effectiveness of the test effort associated with each activity in uncovering defects.

The *number of failing test cases without defect records* was an indicator of the completeness of the defect recording process.

Metrics derived from execution history. The availability of execution results for each test case and the time sequence gave us a unique opportunity to consider some aspects not often scrutinized. Each of these aspects and the preceding list of metrics were input for the evaluation of risk at critical intervals in terms of product stability as well as test progress and effectiveness. Specifically, we considered the following:

Success rate: The percentage of test cases that passed at the last execution was an important indicator of code quality and stability.

Persistent failure rate: The percentage of test cases that consistently failed or completed with errors (but did not change from fail to error or vice versa) was an indicator of code quality. It also enabled the identification of areas that represented obstacles to progress through test activities.

Defect injection rate: We used the percentage of test cases whose states went from pass to fail or error, fail to error, or error to fail, as an indicator of the degree to which inadequate or incorrect code changes were being made.

Code completeness: The percentage of test executions whose status remained "not implemented" or "blocked" throughout was important in terms of evaluating the completeness of the coding of component design elements. It was also useful to measure progress.

Statistically significant cases. A critical requirement we were asked to address was decision support for management, project management, and technical teams. Since it is not easy to make sense from a table with a large number of metrics for each subcategory, we highlighted the statistically "extreme" values of the following metrics as possibly needing attention. Some of the key, statistically significant cases are identified in this list.

- Low percentage of test cases executed
- High number of defects per test case
- High unrecorded defects
- Low success rate
- High persistent failure rate
- High defect injection rate
- Low code completeness

Identifying values as "significant" was based on the statistical distributions appropriate to the respective metrics. For the number of defects per test case, a Poisson model was used, with the expected number of defects in a subcategory proportional to the number of test cases. A subcategory was flagged as significant if the actual number of defects was within the upper five percent of the Poisson distribution. For all other metrics (based on percentages), the significance calculation was based on the hyper geometric distribution of the counts within a subcategory. For metrics where a high value would be of concern, values in the upper five percent of the distribution were flagged as significant. For metrics with undesirable low values, the values in the lower five percent of the distribution were flagged as significant.

Defect data summaries. We generated the following summaries for each sport using the ODC data.

Summary of defect state by severity: The focus of attention here is the possible presence of a large number of open defects of high severity. Priority in fixing and retesting was to have been given according to the severity of the defects, and this metric was one mechanism by which exceptions could be identified.

Summary of defects by test activity: Having first established an expectation for each activity based on

the test case classification and analysis, these summaries were used as one element of evaluating the effectiveness of each test activity in terms of defect removal.

Summary of defects by trigger: This metric provides evidence of how comprehensive the test effort associated with a particular activity has been. A broad range of triggers would indicate that the code has been exercised in a diverse manner, both simple and complex.

Summary of defects by impact: This summary allows an assessment of the nature of the effect the defects are likely to have had if they would have escaped discovery by testing. A broad range of impacts reflected in the defect data suggests that the effort was comprehensive in terms of testing for such areas as reliability, usability, and capability.

Summary of defect type versus qualifier: Analysis of the relationships between defect type and qualifier uncovers weaknesses in explicit areas of requirements, design, and coding activities. These exposures can be targeted in subsequent increments by the developers. In addition, this information helps test management understand the implications of a retesting effort in terms of schedule and resource constraints based on the implied scope of the fault.

The defect data summaries, along with the new metrics based on the execution data, provided detailed insight into various parts of the system. This insight was used to guide the test teams toward more effective defect discovery and evaluation of risk to the overall plan. In addition, the analysis pinpointed focus areas for early prevention or removal of defects by the vendor development team and provided accurate evaluations of progress.

Application of metrics

The decision of where the metrics should be applied to the ongoing project was dictated by when and how critical decisions would be made. Some of these decision points had been long established as part of the original test plan. Others evolved as the project came closer to the completion date and any exposure could mean the difference between success and failure. The comprehensiveness of the planned test effort had been previously verified, as alluded to earlier in this paper and described more fully in Reference 13. This verification was a key element in the analysis of progress, completeness, and effectiveness, and formed the baseline and expectation against which results could be measured. We will begin with a discussion of the formal checkpoints and how the new metrics were integrated into decision support.

Exit criteria. The project had several major checkpoints when progress and status were to be evaluated in terms of four criteria:

The analysis pinpointed focus areas for early prevention or removal of defects by the vendor development team and provided accurate evaluations of progress.

1. There may not be any open Severity 1 problems.
2. There should not be any open Severity 2 problems.
3. Analysis must demonstrate that the test effort has been sufficiently effective and comprehensive (against the planned objectives for this checkpoint).
4. Analysis must demonstrate that the component has reached a sufficient stability level in terms of design (functional content) and code (reliability).

Criteria 1 and 2—Severities. The first two criteria would appear to be quite clear: there either are, or are not, Severity 1 and Severity 2 defects. However, since the checkpoint represents a moment in time, but the analysis of risk does not, it was necessary to look beyond the current open defects. One obvious solution would be to build a metric that captures defect trends over time by severity. Unfortunately, the incremental delivery schedule was such that there would be relatively long periods of inactivity with regard to defects, followed much later by a substantial code drop including fixes for many defects. Standard growth models were simply not effective in this environment. Taking a series of "snapshots," however, using the applicable metrics outlined in the previous section, subset by severity, enabled specific trends to be identified. The relevant metrics and their application to these criteria are described in the following subsections.

Table 2 Results of three metrics intended to represent progress shown relative to two components when approximately 60 percent of testing should have been completed

	Component A (%)	Component B (%)
Percent of unique test cases attempted	43	30
Percent by activity		
Percent of DVT attempted	95	100
Percent of FT attempted	100	27
Percent of FVT attempted	6	32
Percent of test execute = pass	24	14

Success rate. This rate is the percentage of test cases that passed at the last execution. In a mature organization, a typical set of metrics for performing the analysis associated with measuring test progress would likely include one that calculates the percentage of the planned test activity that has been completed. Although this calculation may appear to be satisfied by a straightforward, simple metric, there are mitigating factors that must be considered. In this project, for example, the same test cases were frequently used in two and sometimes three test activities. It was not sufficient to count each unique test case only once. Even basing the same metric on each test activity was not sufficient to provide an accurate view of progress.

In the example reflected in Table 2, both Component A and Component B should have progressed at least 60 percent through the overall test effort at the time the data were analyzed. Using the first metric (percent of unique test cases attempted), we can see that both components are at risk, with Component A having only achieved 43 percent, while Component B is only 30 percent of the overall test effort. The second metric (percent of test cases attempted by activity) provides additional insight but leaves the observer perhaps even more confused as to the risk associated with each component. It does seem clear that Component A has progressed further than Component B and that risk is apparently only associated with the last test activity, function verification test (FVT). For Component B, it seems that both function test (FT) and FVT are at risk, perhaps FVT less so than FT, though this seems counterintuitive. When we use the metric of test case attempts that have successfully passed, however, we see that the risk associated with both components is actually significantly higher than the first two metrics would suggest.

Defect injection rate. The percentage of test cases that showed an execution state sequence of pass to fail, fail to error, or error to fail was interpreted as the extent to which defects were injected while attempting to fix a defect or the attempt to fix a defect was either incomplete or incorrect. The fluctuations in this rate and fluctuations in the volumes of Severity 1 or Severity 2 defects were studied to determine whether there was a correlation. If a relationship did exist, it was used to project the change in volumes for these severities in future test case executions.

Defects per test case, by severity. The rates at which test cases revealed defects and how those rates changed in the course of multiple "snapshots" was an indicator of whether the product was becoming more stable. Measuring the rates, subset by severity, enabled this insight to be used in projecting future risk relative to each severity. Figure 3A and Figure 3B show both the cumulative rates and current period rates relative to Component A and Component B, respectively. We can see that overall, the defects of Component A are primarily low severity, although the most recent period revealed a few Severity 2 defects. It is a very different picture with Component B, where we see a predominance of Severity 2 defects, and the most recent period is continuing this distribution.

Defect state, by severity—summary. This view is mainly concerned with revealing the possible presence of a significant volume of open defects of high severity.

Criterion 3—Test effectiveness. Test effectiveness requires measurements across many dimensions. It is not only necessary to evaluate the extent to which the planned effort has been successfully executed, but it is also important to validate that the plan was sufficiently comprehensive and that the actual results matched the plan. The validation of the plan was performed[13] initially and on an ongoing basis as changes were made to product content and associated test

efforts. Thus, the validated plan was used as the expectation against which the results were compared. Since the plan could be translated into aggregates of domain-specific test cases and their associated execution and defect records, it was possible to track progress and evaluate the effectiveness of a test relative to each domain. In addition to the question of test effectiveness, it is imperative that the test manager understand whether the test resource is being used efficiently, whether there are obstacles that will inhibit or delay the plan execution, and how these obstacles can be managed.

The metrics described in the following subsections were used to analyze these areas of concern.

Success rate. This metric is as critical, if not more so, for evaluating the effectiveness of a test as it was for assessing risk associated with severity. The implication for both Component A and Component B at the checkpoint reflected in Table 2 is that very little progress had been made and that there was still a long way to go.

Persistent failure rate. This metric, as a percentage of test cases that consistently failed, provides insight into one reason why the success rate is reflecting exposures for both components. Excessive and persistent failure rates result in rework for testers and wasted effort. These discoveries are not new but rather are old defects not yet addressed. In comparing our two components in Table 3, we can see that this situation was a very real concern for both components and continues to be a factor for Component B.

Defect injection rate. This metric is also relevant to test effectiveness and provides additional evidence of wasted effort. Both the persistent failure rate and defect injection rate were used to calculate the degree to which the schedule had to be inflated and, in conjunction with other metrics, was used to assess the risk of not retesting specific areas.

Code completeness. This metric captures another aspect of wasted effort on the part of testers, that is, the extent to which test cases fail because the function has not yet been implemented or the successful execution of the test case is blocked because other pertinent function is not available. Table 4 shows that as of the later period, this was not an exposure for either component. In previous periods, it was a consideration for Component A but had not ever been an exposure for Component B.

Figure 3A Results of the execution of test cases in exposing defects at two different stages during development for Component A

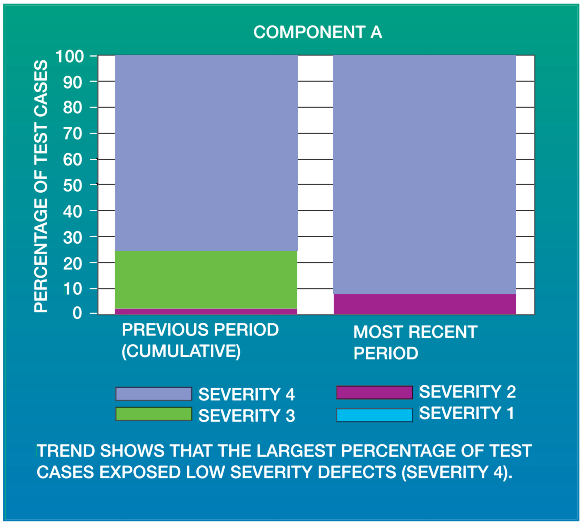
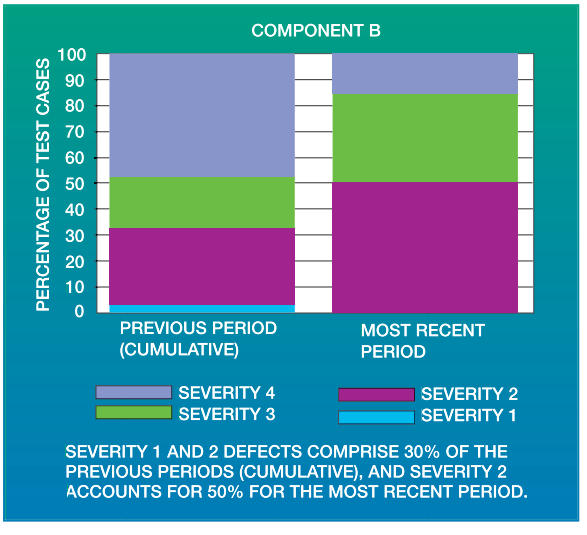


Figure 3B Results of the execution of test cases in exposing defects at two different stages during development for Component B



Percentage of test cases attempted overall, percentage of test cases attempted by subset. These two simple metrics, although not reflecting as accurate a view of progress as other metrics, do serve to indicate the

Table 3 Percentage of test cases having a pattern of multiple "fail" or "complete with errors" and the most current status is not "pass"

	Component A (%)	Component B (%)
Previous periods (cumulative)	25	70
Most recent period	0	52

Table 4 Percentage of test cases that failed because the function had not yet been implemented or the execution is blocked since another function is not available

	Component A (%)	Component B (%)
Previous periods (cumulative)	9	2
Most recent period	0	0

Table 5 Counts of unrecorded (duplicate) defects potentially camouflage wasted effort on the part of testers in rediscoveries

	Component A	Component B
Previous periods (cumulative)		
Count of total defects	201 (100%)	356 (100%)
Count of recorded defects	106 (53%)	317 (89%)
Count of unrecorded defects	95 (47%)	39 (11%)
Most recent period		
Count of total defects	29 (100%)	101 (100%)
Count of recorded defects	15 (52%)	101 (100%)
Count with unrecorded defects	14 (48%)	0 (0%)

changing focus and to what extent the component has been verified. The percentage of test cases attempted by test activity, scenario, and schedule reflect to what extent each is being targeted, and the number of test cases organized by these subsets allows the overall plan to be scrutinized in terms of its comprehensiveness.

Failing test cases without defect records. Initially the results of this metric caused concern, especially in the area of severity and product stability since it appeared that defect records were not being written against all defects. Eventually it was verified that these unrecorded defects were, in fact, duplicates of other defects. The metric is not without value, however, since it shows the extent to which the test team has wasted effort attempting to test a function that has known defects not yet fixed. As we see in Table 5, this was a greater concern for Component A than Component B in the previous periods and continues to be for the most recent period.

Defect data summaries. ODC-based analysis of the defect data was another critical element of the risk assessments. A summary by each ODC attribute revealed specific weaknesses or strengths pertinent to testing. For example, examining the relative distributions of the "trigger" attribute showed whether the majority of the effort had been focused on simple, basic testing, or whether test cases spanning simple to complex use cases had been attempted. The implication of distributions in which simple cases dominate is that the component has not been pervasively tested, it may not be stable, and the test effort to date has not yet been effective. The implication of distributions that show populations across all triggers relevant to the test activity being performed (as defined by the test team at the beginning of the project) is that testing has been comprehensive and the product is relatively stable at least with regard to executing basic function. Another important contribution of the ODC-based analysis was to verify the extent to which added value was provided

by those test cases executed across multiple test activities. Were these test cases serving as little more than a regression test, i.e., an insurance policy that the same failures were not recurring? If this were the case, we would expect to see similar relationships between trigger values and defect-types in any test activity in which the test case was attempted. Or, as the product became more stable and the underlying environment became more complex, would the same test case be able to reveal additional faults? Results for the majority of test case attempts showed that the behavior resembled a regression test—not much that was new was revealed. However there was, in fact, some evidence in specific areas that the relationship of triggers to defect types did change somewhat with regard to the redundant test cases from early test activities to later ones. Additional investigating should be performed to understand the implications more fully.

Criterion 4—Product stability. Metrics discussed in the previous section were found to be sufficient for evaluating the stability of each component. Those that were pertinent included the following:

Success rate: The degree to which the execution attempts were completing without failure or error and how this rate changed across the checkpoints was an indicator of code quality and design stability.

Persistent failure rate, defect injection rate: Test cases that consistently failed, when viewed from the perspective of functional scenarios or test activity, enabled the identification of specific focus areas that may not have been well understood by the development team.

Code completeness: This metric represents the extent to which the design of the product is incomplete ("not implemented" or "blocked"). Although some omissions were planned for and expected (since the plan called for the incremental delivery of function within each component), an unusually high rate or one that continued beyond the date of code drops could be easily identified and scrutinized.

Defects per test case, summaries by defect-type, qualifier, and impact: The rate of defects per test was expected to decrease, and exceptions were easily identified. The combinations of the ODC attributes defect-type, qualifier, and impact revealed a great deal with regard to high-level, detailed, and coding exposures associated with a particular component. Components with a large proportion of function, interface, timing or serialization, or relationship defect-types, particularly if they were "missing" elements, would be considered high risk as the delivery date drew near. If algorithms were the highest priority, that suggested detail design was the weakness. In these cases, further investigation was performed to identify other characteristics of the design concerns. Since fixes associated with algorithm defects are relatively small, risk is considered to be less both in terms of product stability and the impact to testing. Assignments and checking, particularly if incorrect, are usually associated with coding oversights. These have the lowest degree of risk since the fixes are very small, and there would be much less impact to test in terms of delays.

Analysis not only quantified and defined risk at critical checkpoints but provided important information for resource, schedule, and product decisions.

An overall risk value, as well as values for each individual criterion, were included in the assessments. These values were on a scale of 0 (no risk) to 5 (highest risk). The individual criterion risk values were derived by calculating the risk indicated by each of the measurements associated with that criterion, weighted by the importance of each measurement relative to that criterion. For example, the defect injection rate would only be important as a risk measurement relative to Severity 1 criteria if there was a correlation between the rate of defect injection and the volume of Severity 1 defects. In contrast, the defect injection rate would always be critical in evaluating risk associated with product stability. The overall risk value was calculated by a weighted average of the criteria values. The two criteria, based on severity since that exposure could be addressed relatively swiftly, were weighted as less significant than the criteria of test effectiveness and product stability. The primary purpose of the overall value was to provide a consistent mechanism for comparing risk across all of the components and to measure progress from one checkpoint to the next.

Decision support

The analysis guided by exit criteria was a key element of decision support—at a summary level for executives and management, and at a more detailed level for the test manager. This analysis not only quantified and defined risk at these critical checkpoints, but the details of the analysis provided important information for resource, schedule, and product decisions.

The checkpoint analysis was also used on several occasions as the basis for negotiation between IBM and the vendor to resolve issues. A situation arose with regard to one component in which the vendor believed significant progress had been made but the test team did not agree. The vendor felt that much of the risk associated with the component had already been mitigated but had not yet been reflected in the test results because of the time needed to integrate the code increments. Analysis based on these metrics was able to show that even after allowing for the progress reported by the vendor, explicit exposures remained. As a result of the negotiations, the developer for the component worked side by side with the testers to resolve the issues, and the component was stabilized very quickly.

In addition to the in-depth checkpoint analysis, assessments and outlook reports were provided on a weekly basis. The reports included defect data summaries, defect projections, delivery outlooks, and statistically significant values that represented potential concerns. The reports were used to check status and progress by executives, management, and solution managers. Solution managers were responsible for ensuring that any issues or concerns relative to one or more sports were resolved expediently.

In fact, this analysis was instrumental in verifying the need for additional testing for certain components before their delivery to IBM for formal testing. An agreement was reached for IBM testers to participate in "joint tests" at the vendor's site, ensuring that the components were adequately stable before delivery to the IBM test organization. The impact of the early tests was incorporated into the measurements, and the success of these efforts was evaluated in terms of subsequent test results. In each case, the components were shown to be significantly more stable than their predecessors.

Validation

In any commercial software development environment, a controlled experiment on establishing cause and effect is not feasible. In the Olympics project, which involved hundreds of people in many locations across at least three continents, there was enormous complexity in execution, making any attempt for a controlled experiment impossible. Nevertheless, validation of the approach described in this paper was performed, to the extent possible, in terms of two key aspects: verification of the correctness and completeness of data and the evaluation of the usefulness of the metrics.

A benefit of performing analysis at such a detailed level was that any specific problem with the data was quickly exposed. The introduction of the metric, number of failing test cases without defect records, is one example. The process required that a test case that failed or completed with errors had to have at least one associated defect record. It became clear during the analysis that this process was not always being followed and that it was necessary to understand the degree to which that was the case as well as to interpret the broader implications. Although the metric revealed that the situation was pervasive in some areas, discussions of the results with the technical teams led to the understanding that the unrecorded defects should be interpreted as duplicates of known defects. Thus, the usefulness of the metric shifted from evaluating the accuracy of the data to exposing an important concern with regard to wasted test effort and its impact on an already compressed schedule. It could be argued that in-depth analysis often serves the dual purpose of verifying the data and producing meaningful interpretations of the results achieved. In the case of the Olympics project, when inconsistencies or gaps in the data were revealed, explicit actions were taken to address the inconsistencies and fill the gaps with alternative metrics.

Validation of the metrics was accomplished throughout the course of the project in terms of understanding the value of the metric in decision-making, of verifying the results against expectation, and of ongoing discussions with management and technical leaders. The selection, adaptation, and introduction of metrics were made based on their usefulness in critical decision support. The earliest metrics were a key factor in defining the plan and expectations and were verified against the documented plans and interviews with management and technical leads. The

early metrics were followed by metrics that were mainly intended to show progress in terms of product content and stability, to show thoroughness of test, and to identify specific exposures.

Initially the metrics were reviewed with management and technical leads to ensure that they addressed specific requirements. As the project progressed, the analysis was reviewed at critical checkpoints with the key participants. Checkpoint risk assessments were validated through subsequent results and through confirmation from the technical leaders and managers based on their observations and hands-on experiences. For the most part, the assessments matched the perception of the key participants, and on occasion areas of risk not previously known were revealed. The metrics in the last six months of the project, while continuing to focus on product content and stability, also targeted schedule and delivery dates. Status reports, defect projections, delivery date outlook, and associated risk assessments were all provided and tracked on a weekly basis. The analysis was done independently based almost entirely on available data, with as little interaction with the teams as possible (given the severe schedule pressure on them). The weekly reviews were important not only to demonstrate progress and risk, but to identify and validate the analysis and address any anomalies that surfaced.

Results. Where actions were taken to target either product stability or test effectiveness concerns, the impact of those actions was estimated, and corresponding actual results were tracked. The assessments were consistent with the results observed at the time the assessment was made and until the next code drop took place. Since these metrics were available on a weekly basis, it was possible to anticipate—with only a few anomalies—any evidence of mitigated or increased risk. Where there were anomalies between the assessment and apparent actual results, it was discovered that additional test effort had taken place in partnership with the vendor, with defects being recorded only at the vendor site. Test case and related execution records were not captured in the IBM database for these special, joint tests. However, defect counts were provided and accounted for the difference between predicted and actual results.

In addition to the checkpoint risk assessments, weekly component reports were generated for the solution managers. These reports provided information relevant to each sport and also reported exceptions such as the statistically significant cases described earlier. The Appendix contains an example of this report.

In the previous section examples were provided of decisions that were supported by the metrics described in this paper.

The collective set of metrics was an integral part of management decision support at all levels, and as such it is not possible to enumerate all of the ways in which the analysis was used. However, we can report some of the key contributions:

- They formed the set of metrics against which every activity and checkpoint exit was evaluated for each sport component and increment.
- They were the key factor in delivering defect and schedule projections as well as ongoing weekly status and risk assessments.
- They were cited as important tools in negotiations with the vendor, particularly in terms of risk mitigation actions.

Reaction. The executive and management team responded very favorably to the assessments and evaluations. They were satisfied with both the accuracy and timeliness of the assessments and were able to use the reports in regular status meetings to review status with the vendor and to demonstrate the ability of GSC to manage the project to the Olympic Committee.

The test manager in particular expressed appreciation for receiving insight not available to him by any other means. He indicated that the code completeness, defect injection rate, and persistent failure rates were especially useful in managing schedule risks and negotiating with the vendor for additional early testing.

Feedback received from solution managers indicated that they felt the summary reports and prioritized exposure lists enabled them to quickly target or verify areas of concern and mitigated to at least some extent the complexity of their responsibilities. They also appreciated the fact that the assessment was derived independently from the key participants in the project.

The test leaders were pleased to have quantified, irrefutable confirmation of their perceptions. In addition, they acknowledged that the assessments revealed exposures that were previously not known to them. They also expressed appreciation for the fact

that they considered this method to be minimally intrusive and that it had not imposed additional overhead of any significance.

Value

The Summer Olympic Games in Sydney were considered to be a resounding success as an information technology project. According to GSC management, the analysis and metrics described in this paper were instrumental in helping them successfully manage one of the world's most demanding projects and were tremendously helpful in allowing them to gain insight to process and schedule corrections that were required. In addition, there are implications to the industry as a whole. Any project that has been defined with developers in one organization and testers in another, whether or not it is a vendor-vendee relationship, could benefit from the execution record metrics as a means of verifying the quality levels after delivery. If acceptance test is an activity that is performed, these metrics provide a mechanism for performing such an evaluation in a quantifiable manner.

The collective set of metrics was an integral part of management decision support at all levels.

In the case of the Olympics project, the tester population was extraordinarily dynamic because experts had to be brought in for each sport as each increment was being tested. Many software organizations can be characterized as having a good deal of movement within the test organization for a variety of reasons. Distributed development and test also share similar concerns in terms of distant and diversified populations. These metrics were perceived as nonobtrusive by the test leaders, and consistency of data capture was not an issue in spite of the cultural diversity across the team. Each component was delivered incrementally through many builds as planned. Even when planned for, evaluating product stability and test effectiveness in conjunction with these types of development models is extremely complex. It becomes increasingly difficult with each decision to change functional or content deliveries. The use of these metrics at regular intervals throughout the test activities makes it possible to evaluate the risk associated with these characteristics in spite of the complexity inherent in an iterative model.

We described a means by which the value of redundant test cases was measured. Many software development teams are attempting to strike a balance between defining adequate but not excessive regression test suites and ensuring that the test cases are comprehensively testing the product across pertinent environments. Examining the ODC defect types revealed by regression suites or redundant test cases provides a means of identifying the value of the investment and suggests whether additional test cases are needed.

It bears noting that the investment in collecting and validating data during planning and execution of the Olympics project was not trivial and not typical of most projects. The classification of test cases was necessary in order to evaluate the completeness of the planned test effort and establish the baseline against which results could be measured. The metrics described in this paper require a database containing scenarios, test cases, execution records, and defect records, with their hierarchical associations. In addition to the collection of data, a significant commitment was also made in terms of validating and analyzing the data. Although the scope and complexity of the Olympics project justified this investment, it would not be feasible or appropriate for all projects.

Conclusions

The 2000 Summer Olympics was an extraordinarily challenging software project due to many circumstances that characterized it. A highly complex software architecture was required in order to address many demands inherent in such an enormous undertaking, including tracking results of competition venues and events, extracting and incorporating biographical information on athletes, coaches, and officials, and delivering these results and information in many formats, on demand, to multitudes of media representatives, athletes, and spectators.

The specialized knowledge required to test each of the sports components resulted in a high degree of movement in terms of the test personnel, leaving only a small core team of testers to ensure continuity and understand the status of the test effort. Having access to reliable metrics to support their determinations was a valuable ingredient to their success. The

delivery schedule was aggressive and final, and the risk associated with exceptions had to be clearly understood and addressed swiftly. Portions of the project were outsourced to vendors who delivered their components following an iterative model in many increments—a model not easily managed using standard software metrics. These and many other considerations resulted in the need for a new set of metrics that could quickly, but effectively, evaluate and measure risk, progress, product stability, and test effectiveness, without requiring extensive input and involvement from the vendor or the testers. The test metrics described in this paper were highly successful in addressing these needs and were found to be minimally intrusive.

Although the 2000 Summer Olympics could not be considered an average or typical project, the metrics developed to address specific needs in that project are applicable to a wide range of projects across the software industry. Projects in which various roles (such as developers and testers) are managed in separate organizations, including those that have outsourced portions of their product to third-party vendors, are often characterized by incomplete data and information. It has been shown that test-based metrics can be used with a high degree of success when little if any preceding data are available. Projects characterized as dynamic, in terms of a significantly changing workforce, would benefit from these metrics, which rely on input that is standard and easy to understand, requiring little, if any, specialized training. These metrics are particularly useful in projects that have defined an iterative model with incremental code deliveries, where a standard set of metrics such as S-curve are often inadequate for measuring progress. They were also shown to be useful in measuring the value or deficiencies inherent in using the same test case across multiple test activities.

The combination of ODC-based metrics and a new set of test execution-based metrics collectively address the measurement and evaluation of complex, dynamic projects, particularly those with vendor-provided components delivered incrementally. Specifically, the metrics provide decision support in quantified terms with regard to assessing progress, analyzing test effectiveness and product stability, and calculating the degree of risk associated with each of these topics.

Acknowledgments

We want to thank Tom Furey, Patricia Cronin, and Lois Dimpfel of the Olympics executive team for the opportunity to participate in such an exciting endeavor. We appreciate the management support from Toni Plana Castillo and Joe Ryan of the GSC, Spain, for close partnership during the critical times of the testing activities. We thank David Perez Corral and Fernando Dominguez Celorrio of the GSC for integrating ODC in the Quality Management Framework System and for the data capture and enablement of the Lotus Notes*-based infrastructure that was critical to the success of this project. We thank Roy Bauer for the use of Figure 1 and the data for Table 1.

Appendix: Solution manager report for Sport S

Summary:

The overall assessment places Sport S at high risk.

Criteria	High Risk	Medium Risk	Low Risk
There may be no open Severity 1 defects.			
Volume of open Severity 1 defects	X		
Trend	X		
There may be no open Severity 2 defects.			
Volume	X		
Trend	X		
Is test progressing sufficiently?			
Test plan progress (against schedule)	X		
Test effectiveness			
Spread of ODC trigger values		X	
Spread of ODC defect types	X		
Is product adequately stable?			
Defect type volumes	X		
Specific trends	X		

Overall

* DVT = 100% complete, FT = 27% complete, FIT = 32% complete
* 65% of attempted executions concluded successfully

Detail

- Event: Woman's Qualification (551) has a high defect rate—167/88 (number of defects per number of test cases)
- Event: Woman's Qualification (551) has a high repeated failure pattern—170/239
- Test Case Title: After-Event Reports (90) has a high repeated failure pattern—139/159
- Test Case Title: Sports Level TV Graphics (51) has a very low percentage attempted test cases—7/136

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Rational Software Corporation.

Cited references

1. R. B. Grady and D. L. Caswell, *Software Metrics: Establishing a Company-wide Program*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1987).
2. K. H. Moeller and D. J. Paulish, *Software Metrics*, Chapman and Hall, London (1993).
3. N. E. Fenton and S. L. Pfleeger, *Software Metrics—A Rigorous and Practical Approach*, PWS Publishing Co., Boston, MA (1997).
4. M. K. Daskalantonakis, "A Practical View of Software Measurement and Implementation Experiences Within Motorola," *IEEE Transactions on Software Engineering* **18**, No. 11, 998–1010 (1992).
5. G. Stark, R. C. Durst, and C. W. Vowell, "Using Metrics in Management Decision Making," *Computer* **27**, No. 9, 42–48 (September 1994).
6. D. M. Marks, *Testing Very Big Systems*, McGraw-Hill, New York (1992).
7. S. H. Kan, J. Parrish, and D. Manlove, "In-Process Metrics for Software Testing," *IBM Systems Journal* **40**, No. 1, 220–241 (2001).
8. R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong, "Orthogonal Defect Classification: A Concept for In-Process Measurements," *IEEE Transactions on Software Engineering* **18**, No. 11, 943–956 (1992).
9. IBM Research, Center for Software Engineering, http://www.research.ibm.com/softeng.
10. K. Bassin, T. Kratschmer, and P. Santhanam, "Evaluating Software Development Objectively," *IEEE Software* **15**, No. 6, 66–74 (1998).
11. M. Lorenz and J. Kidd, *Object-Oriented Software Metrics: A Practical Guide*, Prentice Hall, Englewood Cliffs, NJ (1994).
12. T. J. McCabe, *Structured Testing*, IEEE Computer Society, Los Alamitos, CA (1983).
13. K. Bassin, S. Biyani, and P. Santhanam, "Evaluating the Software Test Strategy for the 2000 Sydney Olympics," *Proceedings of the IEEE 12th International Symposium on Software Reliability Engineering* (November 2001).

Accepted for publication October 1, 2001.

Kathryn Bassin *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (electronic mail: bassinka@us.ibm.com).* Ms. Bassin holds a B.S. degree from SUNY Brockport and an M.B.A. degree from Binghamton University. She is a senior software engineer with the Center for Software Engineering at the Watson Research Center. She has been with IBM since 1981 and, prior to joining IBM Research in 1993, worked on a wide range of products, performing in a variety of capacities, including management, development, test, and service. Her research interests, likewise, span many areas, most notably software metrics and modeling, and the application of scientific methods to define relationships and influences across the life of a software product. Ms. Bassin developed the Butterfly Model, which provides a comprehensive view of software development, linking design, development, test, and customer usage. Named after the popular analogy associated with chaos theory, the Butterfly Model exploits the use of categorical data including ODC to make rational, objective assessments of the software process and product.

Shriram (Ram) Biyani Dr. Biyani passed away since the original submission of this paper, in a hiking accident. He obtained a B.Sc. degree from Nagpur University, an M.S. degree from the Indian Agricultural Research Institute, a Ph.D. degree in statistics from Iowa State University and an M.S. degree in computer science from North Carolina State University. He joined IBM Research in 1992 and was involved in statistical consulting, research, and tools development. His primary interests in software engineering were in software quality and test design. Earlier, he had provided statistical support to the quality assurance area at IBM East Fishkill and contributed to the development of ALORS2 (A Library of Reliability Specifications). He had also contributed to AGSS (A Graphical Statistical System), developed by the Watson Research Center. Before joining IBM, Dr. Biyani also served as a faculty member at the University of Minnesota and East Carolina University.

Padmanabhan Santhanam *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (electronic mail: pasanth@us.ibm.com).* Dr. Santhanam holds a B.Sc. degree from the University of Madras, India, an M.Sc. degree from the Indian Institute of Technology, Madras, an M.A. degree from Hunter College, The City University of New York, and a Ph.D. degree in applied physics from Yale University. He joined IBM Research in 1985 and has been with the Center for Software Engineering, which he currently manages, since 1993. He has worked on deploying Orthogonal Defect Classification across IBM software laboratories and with external customers. His interests include software metrics, structure-based testing algorithms, automation of test generation, and realistic modeling of processes in software development and service. Dr. Santhanam is a member of the ACM and a senior member of the IEEE.