Improving the Software Testing Process in NASA's Software Engineering Laboratory

Sharon Waligora
Computer Sciences Corporation
10110 Aerospace Road
Lanham-Seabrook, MD 20706
(301) 794-1744
swaligor@csc.com

Richard Coon
Computer Sciences Corporation
10110 Aerospace Road
Lanham-Seabrook, MD 20706
(301) 794-1979
rcoon1@csc.com

In 1992, the Software Engineering Laboratory (SEL) introduced a major change to the system testing process used to develop mission ground support systems in NASA Goddard's Flight Dynamics Division. This process change replaced two sequential functional testing phases (system testing and acceptance testing) with a single functional testing phase performed by an independent test team; functional system testing by the software developers was eliminated. To date, nine projects have been completed using the new independent testing process. The SEL recently conducted a study to determine the value of the new testing process, by assessing the impact of the testing process change on the delivered products and overall project performance. This paper reports the results of this study; it presents quantitative evidence that this streamlined independent testing approach has improved testing efficiency.

This paper presents the results of a study that the authors conducted in 1995 to assess the value of a major system testing process change that was introduced in 1992 in the Flight Dynamics Division at NASA Goddard. It first presents background information to provide the context for the study. This includes information about the SEL and its project environment and highlights some key improvements that had taken place before the testing change was introduced. It then describes the testing process change and our goals and expectations for the new process. The bulk of the paper presents quantitative results that clearly show the impact of the new testing process on recent projects. These results are followed by the conclusions drawn from this study.

Background

The Software Engineering Laboratory (SEL) is a partnership of NASA Goddard's Flight Dynamics Division, its major software contractor, Computer Sciences Corporation, and the University of Maryland's Department of Computer Science. The SEL is responsible for the management and continuous improvement of the software engineering processes used on Flight Dynamics Division projects.

The SEL process improvement approach [1] is based on the Quality Improvement Paradigm [2], in which process changes and new technologies are 1) selected based on a solid understanding of organization characteristics, needs, and goals, 2) piloted and assessed using the scientific method to identify those that add value, and 3) packaged for broader use throughout the organization. Using this approach, the SEL has successfully facilitated and measured significant improvement in project performance and product quality [1].

The Flight Dynamics Division primarily builds software systems that provide ground-based flight dynamics support for scientific satellites. The projects included in this study cover the period from 1985 through 1995 and fall into two sets: ground systems and simulators. Ground systems are midsize systems that average around 250 thousand lines of code (KSLOC). Ground system projects typically last approximately 2 years. Most of the systems have been built in FORTRAN on mainframes, but recent projects contain some subsystems written in C and C++ on workstations. The simulators are smaller systems averaging around 60 KSLOC. These are much smaller projects that last between 1 and 1.5 years. Most of them have been built in Ada on a VAX computer. Simulators provide the test data for the ground systems. The project characteristics of these systems are shown in Table 1.

Table 1. Characteristics of Flight Dynamics Division Projects

Characteristics	Applications	
	Ground Systems	Simulators
System Size	150 - 400 KSLOC	40 - 80 KSLOC
Project Duration	1.5 - 2.5 years	1 -1.5 years
Staffing (technical)	10 - 35 staff-years	1 - 7 staff-years
Language	FORTRAN, C, C++	Ada, FORTRAN
Hardware	IBM Mainframes, Workstations	VAX

Before delving into the testing process change and its results, it is important to understand where the SEL was in its improvement process when the testing improvement initiative began. Previously, the SEL had largely focused its efforts on activities of the design and implementation phases of the life cycle. Experimentation with object-oriented analysis and design had led to a threefold increase in software reuse [3]. Projects were regularly achieving 60% to 90% reuse in the early 1990s. This, in turn, had significantly reduced the cost to deliver systems by reducing the amount of code that needed to be developed. However, it is interesting to note that the cost to develop new code had remained relatively constant. The SEL had also done extensive studies of unit testing techniques and code inspections that led to reduced development error rates (75% reduction).

By 1992, system testing had become the largest part of the software development job, as shown in Figure 1. With higher reuse, less project effort was required in design and coding. In fact, more time was now being spent doing testing than doing design and coding combined. From the business point of view, testing was a natural target for the next major improvement initiative. Also, the testers had pointed out that although the system testing process was effective, in that it produced high-quality systems, they felt it was somewhat inefficient. The SEL had also learned quite a bit about the value of independent testing from experimentation with the Cleanroom methodology. Thus, the SEL turned its attention to improving the system testing process.

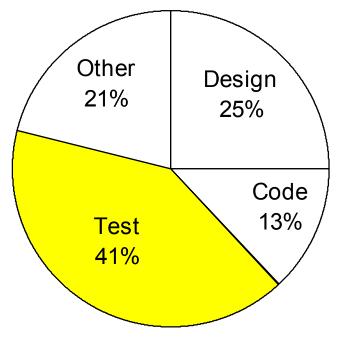

Figure 1. Project Effort By Activity

System Testing Process Changes

So, in 1992, the SEL began its initiative to improve the system testing process. The primary goals of this initiative were to reduce the cost of testing and to shorten the project cycle time without degrading the quality of our delivered systems. Software developers, managers, and acceptance testers worked together to identify inefficiencies and weaknesses in the standard SEL testing approach and to propose improvements.

This group proposed a series of process changes and corresponding organizational changes to support them. The changes focused on eliminating redundancy in functional testing and identifying operational deficiencies (deficient requirements or software) earlier in the life cycle. The following subsections provide a high level description and comparison of the standard SEL testing approach (old testing process) and the new combined independent testing approach (new testing process).

Standard SEL Testing Process

Figure 2 presents the standard life cycle that had been used in the SEL for many years [4]. It is a typical waterfall life cycle in which the system is fully implemented before any system testing begins. There are two sequential functional testing phases. In the system testing phase, the developers test the system against their interpretation of the written requirements. When they are satisfied, they hand it over to a separate group of testers representing the users, who perform another round of functional testing on the system, called acceptance testing. These testers have flight dynamics application expertise and operational experience. They test the system to be sure that it meets operational requirements (i.e., that it can successfully support the mission). Thus, two very similar sets of functional tests are run, with the second set being most realistic for the mission. When the system passes acceptance testing, it is delivered to the users.

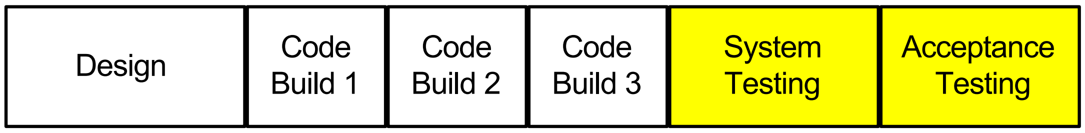

Figure 2. Standard SEL Testing Approach

This approach has several disadvantages. First, operational deficiencies are not uncovered until very late in the development life cycle. Second, the separate, sequential functional testing phases are time consuming and somewhat redundant. Thus, the value of separate functional testing phases is questionable.

New Combined Independent Testing Approach

The new testing approach, shown in Figure 3, involved both an organizational change and a process change. First, we combined system testing and acceptance testing into a single independent testing phase. Second, we created an independent test group within the software engineering organization to do all functional testing and staffed it with people who have flight dynamics expertise and operational experience. Third, we began independent testing earlier, as soon as the first build is completed. The testers test the most recently completed build while the developers move on and implement the next build. The developers are responsible for integration testing of each build before it is delivered to the independent testers. When the system is complete, the testers perform end-to-end testing of the full system for a short period.

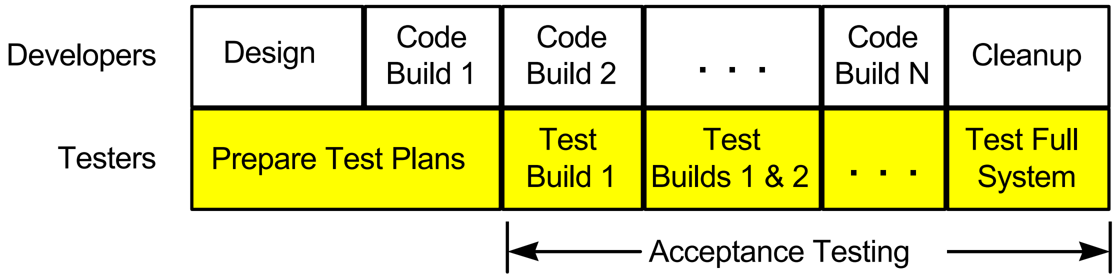

Figure 3. New Combined Independent Testing Approach

Comparison of Testing Approaches

Figure 4 highlights the key differences between the two approaches. The new approach is shown on the top and the standard or old testing approach is shown on the bottom. Key differences are as follows:

- In the new approach, system testing begins much earlier, about halfway through the development life cycle. This allows more calendar time for testing the integrated system and enables the testers to identify operational deficiencies earlier in the project.
- In the new approach, the system is completed very late in the life cycle. This allows the developers more time to resolve incomplete requirements and to respond to the changing requirements that are inevitable in our business.

- In both cases, the software is placed under configuration control near the completion of build 1. However, in the new approach, system testing begins right after the start of configuration control. Therefore, errors found in the configured code would be reported during the testing phase rather than distributed over the code and testing phases as they were with standard approach.

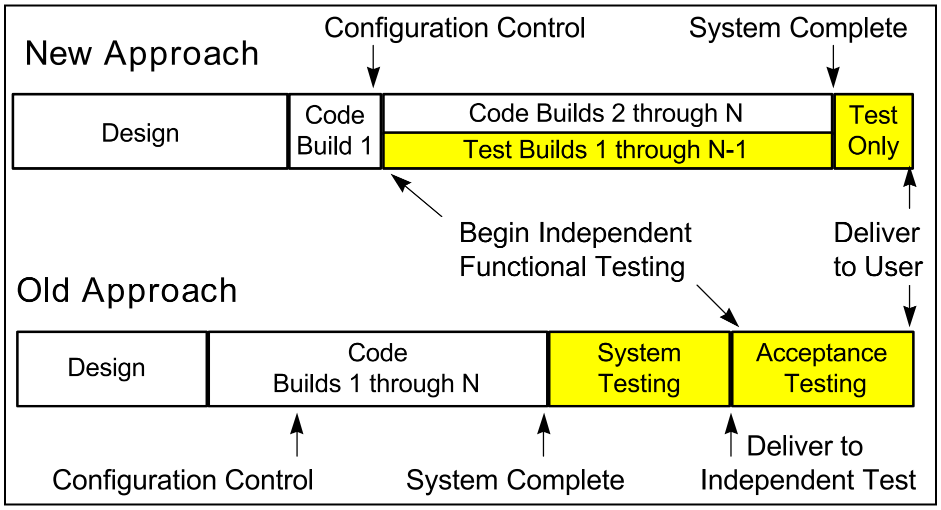

Figure 4. Comparison of Testing Approaches

Quantitative Analysis

To assess the overall impact, or value, of the new testing approach, we did a quantitative analysis of key process and product measures from projects before and after the introduction of the new testing process. We computed new baseline measures[1] for projects that used the new testing approach and compared them with baseline measures of comparable earlier projects.

Sova used a different approach in a similar study [5] in which he compared three testing approaches used in the Flight Dynamics Division, namely the standard SEL approach, the modified or combined independent testing approach, and the Cleanroom statistical testing approach. His selection criteria carefully screened the projects to be sure that the project samples were highly comparable except for the testing process used. His study focused on one subsystem common to all ground systems that was minimally affected by reuse. Six projects were included in his study; two projects for each testing approach. In contrast, our analysis included ground systems and simulator projects completed during each baseline period, excluding only those projects where special circumstances caused them to be unrepresentative samples.

To date, five ground systems and four simulators have been completed using the new independent testing process. The study compared software engineering measures for this

[1]A baseline measurement is the average of the projects (measurements) in a particular baseline time period; it represents the typical project measurement from that time period.

recent project set (projects completed between 1993 and 1995) with those from two earlier baseline time periods. The first includes projects between 1985 and 1989, when a fairly traditional waterfall process was used and projects averaged 20% reuse. The second period covers 1990 through 1992, when projects were regularly achieving between 60% and 90% reuse and the process had been tailored to accommodate high levels of verbatim reuse. All projects in both of the earlier baseline periods used the standard SEL (system and acceptance) testing process described above. We separated the earlier projects into two baseline sets according to reuse levels so that we could more clearly see the effect of the testing process change from one high reuse period to another. Table 2 shows the characteristics and number of projects included in each baseline period.

Table 2. Characteristics of Baseline Periods

Time Period	Testing Approach	Reuse Level	Systems Included
1985 - 1989	Standard Testing (ST)	Low Reuse (LR)	4 Ground Systems 5 Simulators
1990 - 1993	Standard Testing (ST)	High Reuse (HR)	3 Ground Systems 3 Simulators
1993 - 1995	Independent Testing (IT)	High Reuse (HR)	5 Ground Systems 4 Simulators

Process Change Confirmed

Software development activity data, shown in Figure 5, clearly confirms that a process change has taken place. Displayed here are two donut charts. In each case, the inside donut represents the standard testing approach and the outside one represents the new testing approach. The left chart shows the relative percent of effort performed by the developer organization vs. the independent testing organization. The chart on the right shows the percentage of project effort spent doing development activities (design and coding) vs. testing activities regardless of organization. In both charts the testing portion is shaded.

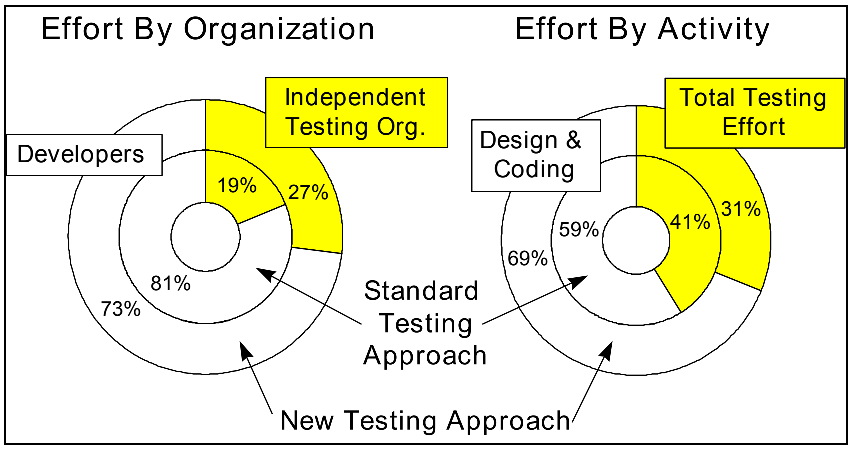

Figure 5. Distribution of Project Effort

Overall system testing effort declined from 41% to 31% of the total project effort (a 25% reduction). Although recent projects spent a smaller portion of the total effort doing testing overall than earlier projects did, the independent testing organization now contributes a larger part of the total effort (an increase from 19% to 27%). The difference between the total testing effort and the amount spent by the independent testers represents the amount of testing done by the software developers. This data indicates that currently software developers are spending only 4% of the total effort on integration testing as compared with 22% on system testing activities(including integration testing) for the earlier systems. This confirms a major shift in the process used and the responsibility for testing systems.

Cost To Deliver a Line of Code

Figure 6 shows the average cost to deliver a line of code[2] for each of the three baseline periods. The labels under the bars indicate the testing approach and the level of reuse during each period (see Table 2). Each bar is divided into three parts showing the portion of the cost attributable to the various software engineering activities. The bottom portion represents design and coding, the middle indicates the amount of system or integration testing done by the developers, and the top section indicates the portion of the effort spent doing independent or acceptance testing.

As you can see, there was a significant reduction in the overall cost to deliver a line of code with the increase in reuse. The introduction of the new testing approach in the third baseline period had a smaller impact (15% reduction) on the cost to deliver.

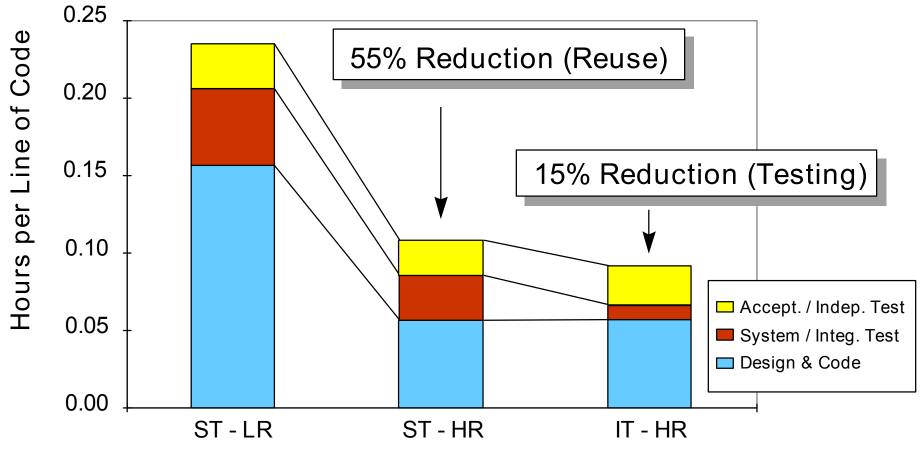

Figure 6. Cost to Deliver A Line of Code

Cost To Develop a New Line of Code

By removing the effects of reuse, we see a different picture. Figure 7 is similar to Figure 6 except that it shows only the cost to develop new code[3]. As you can see, the cost to develop a

[2] Cost to Deliver = Total Project Effort / Total Lines of Code
[3] Cost to Develop a New Line of Code = Total Project Effort / New and Modified Lines of Code

new line of code increased somewhat in the middle time period, along with a fairly large increase in the cost of testing a new line of code.

After the introduction of independent testing, effort measures show a 23% reduction in actual cost to develop a new line of code. A closer look shows that most of the savings have occurred in testing activities, where a reduction in developer test effort (78%) and a 33% rise in independent testing effort (33%) together net a 40% reduction in overall testing cost per new line of code. This is significant because the SEL had previously never seen a decrease in the cost to develop new code despite large reductions in the cost to deliver systems as a result of high reuse.

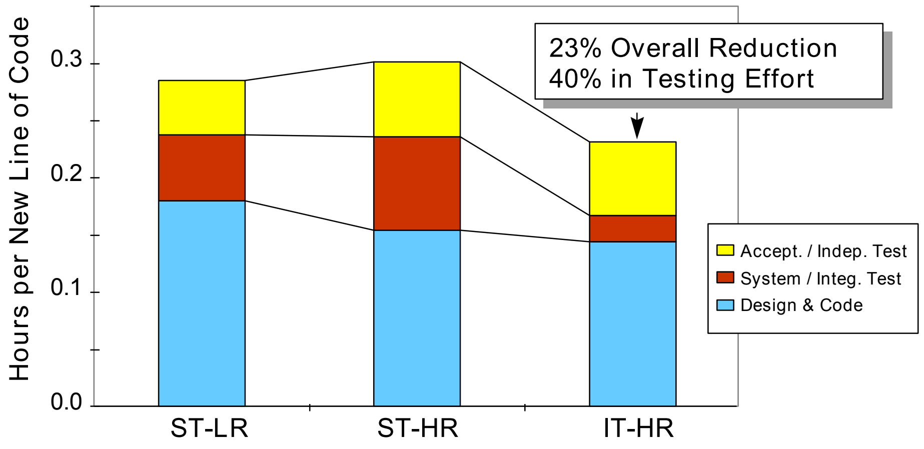

Figure 7. Cost To Develop A New Line of Code

Separating the data shown in Figure 7 by project type uncovered a difference in cost savings. As shown in Figure 8, ground system development projects reaped a significant 35% savings, almost entirely due to a reduction in testing cost, while simulators experienced only a 10% overall reduction in cost with minimal savings in testing. Close examination of project histories revealed that the simulators tended to have fewer builds and a smaller overlap between coding and testing. Also, simulators are now tested under more schedule pressure because they are needed earlier to begin testing the ground systems. This data seems to indicate that cost may be negatively affected by schedule pressure. Further analysis is needed to clarify this.

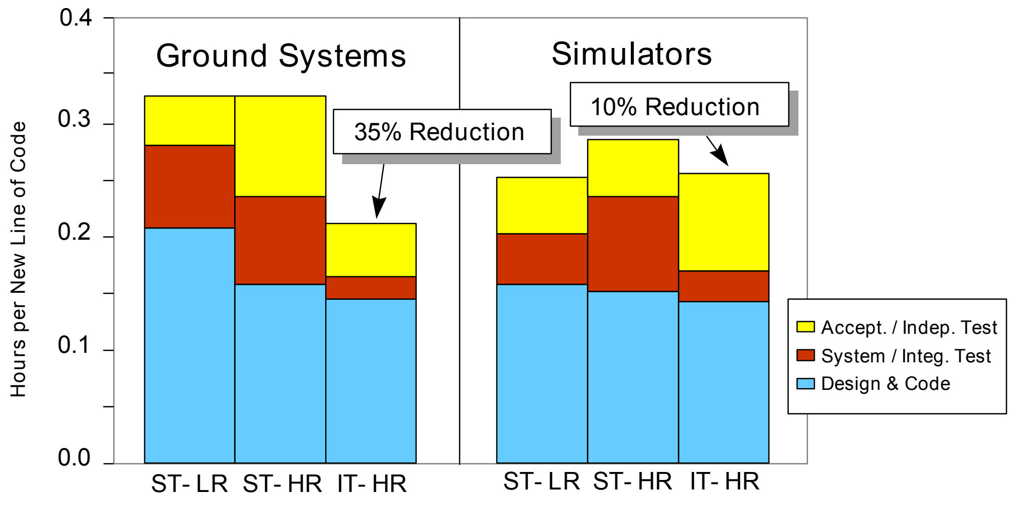

Figure 8. Cost To Develop A New Line of Code by Project Type

Shorter Cycle Time to Delivery

Schedule data for the recent projects, presented in Figure 9, show a slight reduction in average project duration for ground systems and a 29% reduction for simulators, when compared to the middle (high reuse) time period. Notice that the schedule impact on each type of project is the opposite of the impact on cost (shown in Figure 8). Simulators saved time, but not cost; while ground systems saved cost, but not time. It appears that the ground systems projects have used the productivity gain from the new testing process to reduce staff, while the simulator projects have used it to reduce schedule. Further analysis is required to fully understand the trade-off between cost and schedule and its relationship to the testing process, but is clear that the new testing process has helped projects meet their various objectives.

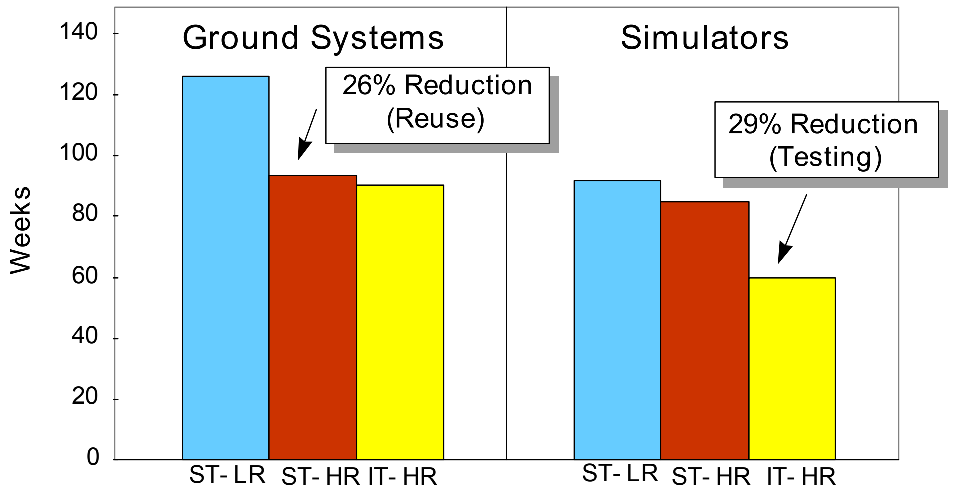

Figure 9. Average Project Duration by Project Type

System Quality

Development error rates of the recent systems, shown in Figure 10, also decreased when compared with the earlier systems. The average error rate on the recent projects is 1.5 errors per KSLOC. This is down from 4.3 errors per KSLOC and 2.8 errors per KSLOC for the two earlier baseline periods. The distribution of errors by phase (indicated by the shaded subsections of each bar in Figure 10) reveals that very few errors are now reported by the software developers (during the implementation phase). Although more errors are uncovered by the independent test team using the new approach than when doing standard acceptance testing, fewer errors were reported altogether during the system testing portions of the life cycle. However, it is unclear from this data whether the product quality is improving or whether the independent testers are not finding all of the errors. A better indicator of system quality would be the error rate during the first year of operational use. Unfortunately, records of operational errors were not kept for systems in the early baseline period and the recent systems, tested using the new testing method, are now just beginning to be used operationally. Thus, it is too early to judge whether the new testing process maintains quality.

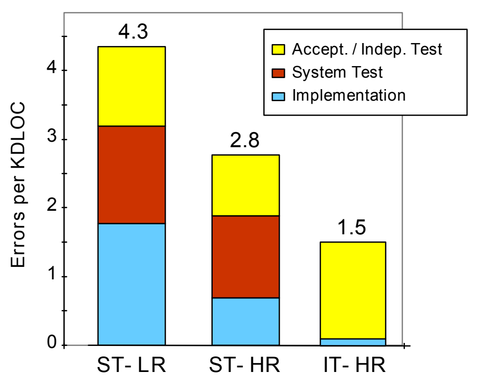

Figure 10. Development Error Rates

Conclusions

Based on our quantitative analysis, we conclude that the combined independent testing approach is beneficial and should be adopted as the standard testing approach to be used on future projects in the Flight Dynamics Division. Our analysis revealed that project performance improved without sacrificing quality on projects using the new testing approach. Specific performance factors are addressed below.

- Cost: We conclude that the new testing approach reduces project cost. There was significantly reduced testing effort on projects using the new approach, which consistently contributed to overall project cost savings.
- Cycle Time: Because cycle time improvement varied by project type, we conclude that shorter cycle times are possible, but not guaranteed when using the new approach. There appears to be a tradeoff between cycle time and cost; i.e., the shorter the test phase, the more it will cost.
- Quality: It is too early to determine the impact of the new testing approach on product quality. Although project data continue to show a decline in the development error rates, operational error rates are not yet available for most of the recent systems.

One additional benefit of the new testing approach is the longer overlapped periods of implementation and testing. This allows more calendar time for both development and testing. This stretched-out development time period provides the flexibility needed to deal efficiently with late-coming requirements, thereby reducing rework. The stretched-out testing time period allows for a smaller team of testers to test each system, thereby reducing the learning curve and capitalizing on growing mission knowledge as testing proceeds.

The results of this study are similar to those found by Sova [5]. His study found the independent testing approach (referred to as the modified approach) to be most efficient and produce the lowest fault density. Thus, both the broad brush analysis of baseline comparisons and the detailed comparison of carefully selected samples confirm that the new independent system testing approach is beneficial and should be adopted as the Flight Dynamics Division standard testing process.

References

1. McGarry, F., G. Page, V. Basili, et al., An Overview of the Software Engineering Laboratory, Software Engineering Laboratory, SEL-94-005, December 1994
2. Basili, V., “Quantitative Evaluation of a Software Engineering Methodology,” Proceedings of the First Pan Pacific Computer Conference, Melborne, Australia, September 1985
3. Waligora, S., J. Bailey, M. Stark, Impact of Ada and Object-Oriented Design in the Flight Dynamics Division at Goddard Space Flight Center, Software Engineering Laboratory, March 1995
4. Landis, L., S. Waligora, F. McGarry, et al., Recommended Approach to Software Development (Revision 3), Software Engineering Laboratory, June 1992
5. Sova Jr., D., “A Study of Software Testing Methodologies Within the Software Engineering Laboratory”, M.S. Thesis, University of Maryland, 1995