



The Path to Software Cost Control



Dr. James R. Eckardt ■ Timothy L. Davis ■ Richard A. Stern
Dr. Cindy S. Wong ■ Richard K. Marymee ■ Arde L. Bedjanian

Many programs risk cost growth and schedule delays because of software development issues. In the 2010 Government Accountability Office (GAO) defense acquisition report, *Assessments of Selected Weapon Programs*, the programs with count growth in significant source line of code (SLOC) since development startup experienced accelerated cost increases and excessive schedule delays relative to other programs. The report asserted that collecting, tracking and containing software defects in the phase where they occur is an excellent cost-control management practice. Programs surveyed indicated that an average of 31 percent of defects corrected were detected after the development phase in which they were inserted. Capturing software defects in phase is critical because detecting defects out of phase results in expensive program rework.

Real Cost Impact

Software defects are so prevalent and detrimental that they cost the U.S. economy an estimated \$59.5 billion annually, or about 0.6 percent of the gross domestic product, according to a 2002 study commissioned by the Department of Commerce's National Institute of Standards and Technology (NIST). A more recent Cambridge University study reported that the global cost of debugging software has risen to \$312 billion annually. The research found that, on average, software developers spend 50 percent of their programming time detecting and fixing defects.

Bedjanian is founder and president of GreenDart Inc., a San Pedro, Calif., firm focused on identifying software issues early in development cycles. **Davis, Stern, Eckardt, Wong** and **Marymee** are systems engineers at GreenDart.

Some Recognized Challenges

Defect-removal efforts substantially compound several parallel factors that also result in significant program cost growth:

- Decreased Product Life Expectancy**—Due to technology advances and rapid product evolution, the life expectancy of software products has decreased dramatically over the past several years.
- Increased Program Complexity**—The size of software products no longer is measured in thousands of lines of code but in millions.
- Optimistic Software Reuse Plans**—Many programs propose aggressive software reuse in order to lower the proposed cost of the software without reducing the estimated software size.
- Requirements Growth**—A current trend toward “late binding” along with the revision of customer requirements during development risks an introduction of an unintended requirements creep. This disrupts predevelopment cost and schedule estimates.
- Curtailed Testing**—As development progresses, many programs experience a cost growth and schedule slip that result in a simplified “back-end” testing agenda to recover some schedule. This approach emphasizes test for success (verifying all requirements are met) and limits test for failure (the search for critical flaws).

These factors place early pressure on developers to maintain schedule commitments, leading to increased reliance on final product testing for defect detection. In the commercial realm, the increased use of “beta releases” is a symptom of this. However, studies have shown that optimal schedule and cost outcomes actually occur with rigorous early detection and removal of defects. This paper presents a means to move toward that optimum.

The Software Development Life Cycles (SDLC) adheres to critical phases that are essential for product development. These phases include planning, analysis, design and implementation and may include concurrent system evaluation, information gathering and feasibility studies. Traditional waterfall SDLC may be replaced by variations of the Agile/SCRUM (the later involves multiple small development teams) development methodology, due in part to today’s increased program complexity and module count. No matter which process is implemented, defect insertion can occur during the correction of the identified defect and will additionally impact program cost and schedule.

Typical Defects and Frequency

Reference data indicate that about 40 percent of defects originate in the requirements definition phase (with design accounting for 10 percent, code for 45 percent, and test for

Table 1. Typical Software Development Life Cycles (SDLC) Phase-Related Defects

SDLC Phase	Typical Defect
Requirements Definition	<ul style="list-style-type: none"> Requirements, and associated data, are not traced correctly, are missing or aren’t stated clearly. Software requirements specifications, interface requirements specifications, test approaches/data, algorithms are incorrect and/or inconsistent. Inadequate and/or incorrect user interface as input from user groups.
Design	<ul style="list-style-type: none"> Incorrect or inconsistent interface traceability between documents. Requirements are not satisfied by the software design. Critical functions and/or algorithms have been identified but not correctly described. Design risk and risk mitigations have been incorrectly identified.
Code	<ul style="list-style-type: none"> Incomplete source code, unused or unreachable code. Incorporation of “buggy” reuse code and ineffective integration of commercial off-the-shelf (COTS) and government-furnished equipment (GFE) software. Failure to track code corrections, uncompleted code and code-completion schedules. Failure to systematically identify critical and hazardous components of the code for additional risk management. Inadequate/incorrect/misleading or missing comments in the source code. Standards and project-related design/requirements/coding standards not followed.
Test	<ul style="list-style-type: none"> Failure to track code corrections, incomplete code and code-completion testing schedules. Failure to ensure that hazardous and otherwise critical components of the code are thoroughly tested. Limited test data used in component development and testing. Incomplete developer test plans, test procedures or test execution results. Limited testing and review of results do not adequately demonstrate that the software supports mission requirements and capabilities.



... More than a third of these costs ... could be eliminated by a more rigorous software assessment process that enables earlier and more effective detection and correction of software defects.

5 percent). Of these defects, the requirements phase only detects and corrects about 15 percent (design corrects 10 percent, code 45 percent and test 30 percent). Table 1 depicts a list of typical phase-related defects independent of SDLC process model used.

Cost of Latent (Out-of-Phase) Defects

Defects not removed in their respective creation phase are subject to a substantial—and escalating—repair cost penalty when corrected later. For example, a requirement defect detected in operations resulted in a cost 368 times greater than it should have been, according to NASA's study of return on investment (ROI) for software independent verification and validation (IV&V). Delayed defect correction increases rework (cost/schedule impact) required to correct the defect. Delayed defect correction typically involves making numerous changes to both the original and now related software, to intermediate work products (such as test procedures) and more extensive regression testing. More change activity also increases the opportunity to introduce new defects during the delayed corrections.

Figure 1, Latent Defect Cost Escalation, compiled from this NASA study illustrates the relative cost escalation of correcting an out-of-phase defect. In this figure, an in-phase corrected defect receives no cost impact but, if the detection and correction occur in a subsequent phase, the costs increase exponentially. This cost penalty creates a great incentive to identify and correct the defect in phase.

According to the 2002 NIST study, not all defects can be corrected in a cost-effective time span. However, more than a third of these costs, or an estimated \$22.2 billion, could be eliminated by a more rigorous software assessment process that would enable earlier and more effective detection and correction of software defects.

Addressing Developmental Program Latent Defects

Major cost savings at the total program level are

achievable by systematically containing most software defects in or near the phases where they are introduced. Detecting latent defects as early as possible is best, specifically if corrected in the phase where they are introduced rather than detected later. Current defect-detection strategies include: (1) independent testing; (2) developer verification and validation (V&V); and (3) IV&V. As will be shown, only one of these approaches is effective for identifying potential latent defects within the phase where the defect is introduced.

Independent Tests to Identify System Defects

Independent testing brings significant value to the final acceptance of software systems. These tests typically are executed on completed systems by an organization (or separate company) independent of the development effort—which increases system assessment objectivity. The problem with addressing latent defect costs using this approach is timing—the testing occurs much too late in the SDLC to reduce latent defect impacts. Therefore, independent testing is not a mechanism for reducing latent defect costs.

Why an “Independent” Effort Is More Effective

Development organizations address V&V in two ways: (1) employing a product review process at the end of each phase of the development by the developers themselves; and (2) using a separate team to V&V the developed products. While developer V&V may encompass many forms of development

Figure 1. Latent Defect Cost Escalation

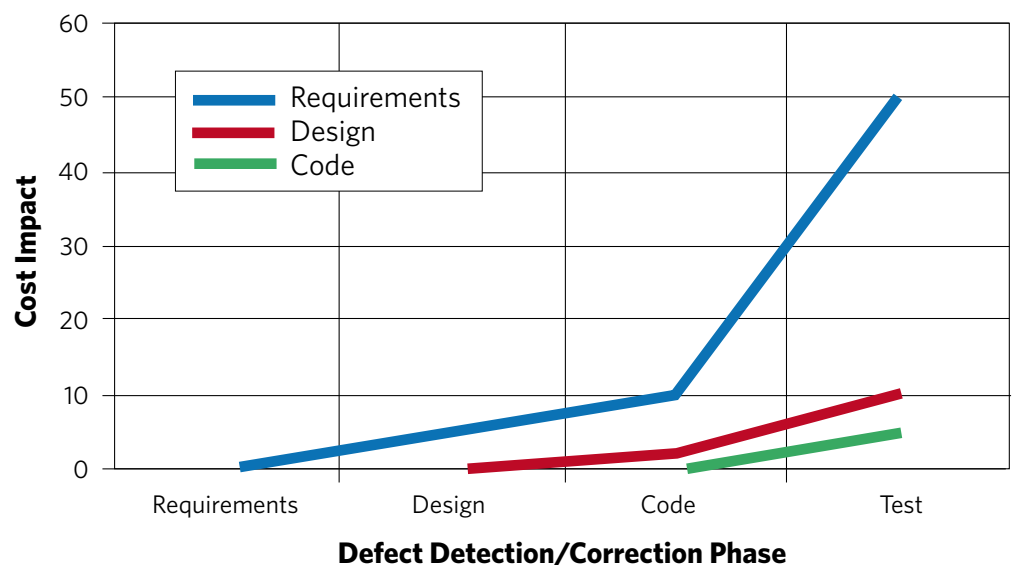
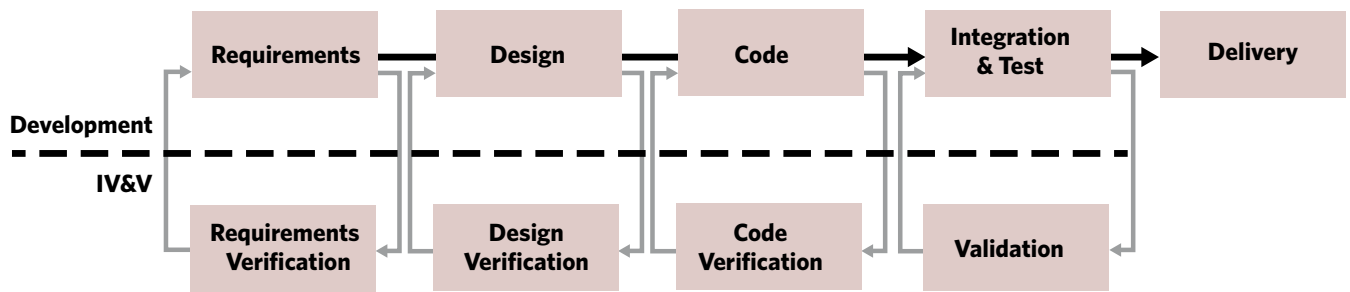


Figure 2. IV&V Process Tied to SDLC Phases



testing, the developer’s primary focus is requirements sell-off “test for success” verification activities. However, a significant portion of the defects identified in Table 1 are not detectable by this strategy. To capture these types of defects, the approach must include a “test for failure” focus (e.g., limit checking, off-nominal condition analysis, etc.). These are not typical requirements sell-off strategies and, therefore, are not activities performed by the developer’s V&V team. They are, however, key strategies of an effective IV&V effort.

Reducing the Latent Defect Impact

IV&V is a software assessment technique that integrates with the developer’s process to capture, assess and report on defects in developed products. A sample IV&V program, linked to developer activities, shown in Figure 2, integrates the developer’s waterfall SDLC process with IV&V assessments and feedback loop responses. The outputs for each developer phase are assessed, and feedback (e.g., identified defects) is provided to the development team in phase. IV&V maximizes

Table 2. IV&V Tasks to Eliminate Latent Defects

Requirements Verification	<ul style="list-style-type: none"> • Validate that the requirements are complete, concise, understandable, testable and that they satisfy the user’s needs. • Verify that the developer requirements are traced accurately to software components and back to the system and interface requirements. • Evaluate risks associated with the requirements and with the concepts and plans for testing. • Review software requirements specifications, higher-level requirements and interface requirements specifications for consistency. • Ensure that test approaches and test data are correct and consistent. • Ensure algorithms are consistent with requirements and test planning and that the algorithm test plans are sufficient.
Design Verification	<ul style="list-style-type: none"> • Verify that the interfaces are correct and consistent in all documents. • Validate that the requirements are satisfactorily implemented in the design and that the design satisfies all of the requirements. • Review the reuse code and the reuse plan to ensure the feasibility of reuse as planned. • Verify that the critical functions and algorithms have been identified and prototyped and are addressed in the design. • Ensure that the developers have correctly identified design risk and security issues and appropriate mitigations. • Ensure that test procedures and test data are correct and consistent.
Code Verification	<ul style="list-style-type: none"> • Analyze supplied code with code analysis tool(s), identifying any code debug/violations. • Track code corrections, incomplete code and code completion schedules. • Ensure that critical and hazardous components of the code are identified. • Monitor code development performing design through code trace analysis. • Evaluate unit test artifacts for completeness, addressing relevant requirements and off-nominal testing.
Validation	<ul style="list-style-type: none"> • Validate that test results address the user’s needs and system requirements. Validate test results against expected results in test plans. • Identify and track retest of corrections, incomplete testing, and retest/regression test completion schedules. • If developer cost and/or schedule overruns occur, identify and evaluate mitigation options.

development insight, identifies weaknesses, assesses failure conditions and uncovers defects as they are introduced into the system—thereby reducing the potential for latent defect propagation into later phases.

Other nonwaterfall developer processes (e.g., Agile, etc.) are also accommodated by an IV&V integration strategy.

The application of IV&V is unique to each development effort, based on such factors as customer’s priorities (where to focus), developer strategy, developer processes and products and the application of IV&V tools unique to the particular development effort. Typical IV&V tasks include those listed in Table 2. When the tasks referenced in Table 2 are executed successfully, critical defect detections are accelerated, thereby saving program costs through minimized rework, reduced development schedule and decreased operational maintenance costs.

Identifying defects early and, hence, saving program costs requires an investment in IV&V tasking. So we come to the real question: “Is the price of the IV&V effort justified by the program cost savings?”

IV&V Return on Investment

In 2012, GreenDart, along with the NASA IV&V Facility in West Virginia, conducted a study into the long-term effects of IV&V on program development costs. Based upon the NASA-provided development and IV&V defect-identification information for 31 programs, the paper concluded the ROI for IV&V ranged from a conservative 85 percent to a maximum 294 percent above the cost of performing the IV&V.

Therefore, an investment in IV&V returns at least 85 percent program savings beyond the cost of the IV&V effort. In the most extreme cases, IV&V returned 294 percent program savings. In short, the investment in IV&V is justified.

Computed IV&V Cost Savings

The example in Figure 4 illustrates the impact of including IV&V in a software program’s development. The results of the GreenDart-NASA and NASA IV&V ROI studies show that, for a program with an initial development cost of \$90 million, latent defects are estimated to raise the project’s actual cost to \$115 million. The customer can reduce some of this cost by adding IV&V. Using the conservative ROI of 85 percent, the following calculation shows that \$6 million spent on IV&V reduces the cost of latent errors by about \$11 million:

Figure 3. IV&V Return



* Return on investment, excluding investment costs

$(\$6 \text{ million IV\&V}) \times 1.85 = \$11 \text{ million latent defect savings.}$

Subtracting the cost of IV&V from this gives a software development savings (excluding schedule savings) of:


$\$11 \text{ million} - \$6 \text{ million} = \$5 \text{ million net savings.}$

It is important to note the final program costs are still in excess of the proposed \$90 million price:

TOTAL COST: $\$90 \text{ million} + \$25 \text{ million (latent defects)} - \$11 \text{ million (latent defect savings)} + \$6 \text{ million (IV\&V costs)} = \110 million.

Additional program measures must be employed to sustain a \$90 million cost profile (reduce program requirements, etc.).

Conclusion

Many factors contribute to software development cost overruns. One major cost impact is latent defects. Significant study results, presented in this paper, identify the latent defect cost impacts and the positive cost savings of an effective IV&V program. 

The authors can be contacted through arde.bedjanian@greendart.aero.

Figure 4. Anticipated IV&V Results

