Research Feature

Evaluating the Effectiveness of Independent Verification and Validation

The authors are among the first to test whether independent verification and validation have a measurable beneficial effect on the development process.

James D. Arthur
Markus K. Gröner
Virginia Tech

Kelly J. Hayhurst
C. Michael Holloway
NASA Langley Research Center

The complexity of today's software systems mandates not only a structured approach to development but a verification and validation process that will ensure both that the right product is built and that it is built right.[1] A V&V process is especially critical for high-consequence systems in which a software failure can result in the loss of life[2] and for systems—like missiles—where live testing isn't feasible.

The two terms—verification and validation—are often used in conjunction to describe a single set of activities. The terms themselves, however, imply distinctly different sets:

- *Verification* refers to the process of examining each development phase to ensure that the output of a particular phase satisfies all the pertinent requirements of the previous phase, is internally acceptable, and can support the development effort in the next phase.
- *Validation*, on the other hand, is an activity primarily concerned with software testing. During validation you execute the system and compare the test results to the requirements.

Independent V&V has many things in common with V&V—like these terms—but while V&V is generally performed by a group closely aligned with the development team, IV&V is conducted by a group that is completely independent. But whether IV&V differs from V&V in more than just the independence of its practitioners is still open to debate.[3,4] We believe, however, that it is the independence that promotes objectivity and encourages you to consider a wider range of solutions, which in turn can improve error detection.

To better understand and evaluate the impact of verification and validation on the development process, NASA Langley Research Center funded a study to examine the effectiveness of the Army's Software Engineering Evaluation System (SEES).[5] A joint investigative effort that included both Virginia Tech and NASA LaRC, the project led to a study designed to examine the benefits of using SEES as an *independent* V&V methodology. In particular, the study focused on assessing to what extent, if any, IV&V activities help detect faults earlier in the software development life cycle, reduce the time to remove those faults, and produce a more robust product.

The study consisted of two independent development groups—see the sidebar "The Experimental Groups"—each of which was given an identical set of requirements that outlined a solution to a particular problem. We asked both groups to design, code, and test their software. The results, particularly the difference in fault detection capabilities between the two methods, indicate that IV&V provided a significant value-added component to the software development process.

THE EXPERIMENT'S ARCHITECTURE

For testing the value of IV&V, we used a modified version of SEES, which is a comprehensive V&V methodology developed by the US Army Missile Command's Software Engineering Directorate. SEES is based on a five-step implementation process:

1. assessing the V&V needs for a planned software development project;
2. tailoring SEES V&V procedures to fit schedule and budget constraints;

0018-9162/99/$ 10.00 © 1999 IEEE

The Experimental Groups

The experiment described in this article employed two distinct, noninteracting groups. Group 1 was composed of two individual teams: a three-person development team and a four-person IV&V team. The IV&V team assessed the quality of the development artifacts produced by the development team. Group 2 had no IV&V contingent and was only composed of a two-person development team.

In assigning team members, common practice generally dictates the use of random assignments to minimize the potential for bias. Because of the small number of available participants (and the consequent belief that random subject assignment would not lead to development teams with equivalent levels of software engineering expertise), we elected instead to use matching assignments based on work experience histories and education levels to produce teams having similar skill levels.

While random assignment could not be used, we did apply several controls to the experiment to help minimize bias and improve rigor. We

- used a matching assignment to achieve parity in development skill levels,
- developed the requirements specification prior to the experiment and distributed it to both groups at the same time,
- used a single DEC Alpha running Unix as the development platform for both groups,
- instructed members of each group not to discuss its development effort, and
- controlled interaction between Group 1 and the IV&V team.

Finally, to minimize errors in measurement and data collection, we assigned an individual who was not a member of either treatment group the responsibility of gathering, reviewing, organizing, and confirming the validity of the data recorded by both groups.

3. executing selected Technical Assessment Procedures (TAPs), which focus on technical V&V tasks and programmatic evaluation tasks that address development process and configuration management issues;
4. measuring and analyzing the results of the V&V activities to identify software risks and deficiencies; and
5. providing analytical feedback to improve the quality and reliability of the current system.

SEES TAPs are the heart of the evaluation system—see the sidebar "The Software Engineering Evaluation System"—and provide the technical guidance for V&V activities.

Due to limited resources and time constraints imposed on our experiment, and because of the need for an IV&V procedure instead of the V&V procedure that SEES defines, we changed SEES in three ways. More specifically, to produce SEES′, the modified version of SEES, we

- adapted SEES to include an independent V&V component,
- selected only five of the original 10 TAPs—those that stressed fundamental V&V activities, and
- modified the reporting forms required by each TAP in order to maintain a coherent picture of development and IV&V activities.

We developed the requirements specification prior to the start of the experiment and gave it to both development groups at the same time. We instructed each group to review the requirements for understanding and to request clarification where needed. The IV&V team examined the requirements, developed a software requirements list in accordance with SEES′ guidelines, and prepared defect trouble reports for all faults detected. The IV&V team then reported its findings to the Group 1 development team, which reviewed the reports and made appropriate corrections. In this way, the IV&V team worked on each of the development stages with Group 1 and had no contact whatsoever with Group 2.

DETECTING FAULTS EARLIER

After the experiment, we gathered the data and asked this question: Do the IV&V activities prescribed by the SEES′ IV&V approach add value to the existing software development process? More specifically, do the IV&V activities defined in SEES′ support early fault detection and reduce the effort required to remove those faults?

Figure 1 provides a side-by-side comparison of the number of critical errors detected by Group 1 and Group 2 (the non-IV&V group). The figure illustrates that the number of faults detected by Group 1 peak during low-level design (LLD) and then fall off substantially through integration and testing (I&T). This trend indicates that testers detected the majority of the faults during the design phases, with detected defects decreasing steadily in the later two phases. Figure 1 also illustrates that the number of faults detected by Group 2 grows exponentially as development proceeds from requirements specification through code and unit testing, with a significant number still being found during I&T. Experience suggests that this trend is more the norm for software development efforts lacking a formal V&V process.

When we compare the fault detection trend for Group 2 to that for Group 1, we observe that Group 2

- begins finding faults a phase later,
- detects fewer faults in the earlier phases, and
- peaks a phase later.

Hence, given the assumption that we have equally proficient development teams, we must conclude that employing the IV&V activities prescribed by SEES′ *does* promote the earlier detection of faults. For readers interested in our statistics: A chi-square test comparing the two groups yields a P value of .002, indicating that these results are extremely statistically significant.

One additional observation that should be noted is

the marked difference between the number of faults reported by Group 1 and the number reported by Group 2. Why such a substantial difference? Two of the more obvious possibilities are that either

- the development team from Group 1 introduced more faults than Group 2, or
- both groups introduced basically the same number of faults, but Group 2 simply failed to detect them.

We tend to discount both of these explanations because the process by which the personnel were assigned to groups focused on maintaining equal development capabilities between them, and preliminary results from acceptance testing[6] indicate that the software produced by both groups met the basic set of stated requirements.

A detailed examination of the data reported by Group 1, however, reveals the most plausible explanation: The IV&V team from Group 1 identified an additional class of critical faults not found by Group 2. More specifically, a substantial number of the critical faults reported by Group 1 stem from the detection and recording of ambiguous or unclear statements in the requirement specifications and design documents. Those statements, if left uncorrected, could have easily been misinterpreted or misunderstood, which, in turn, could have resulted in the introduction of critical faults downstream in the development process. Group 1 reported 40 such faults: 14 in the requirements, 18 in the high-level design, and eight in the low-level design phases. Group 2 reported only one such fault.

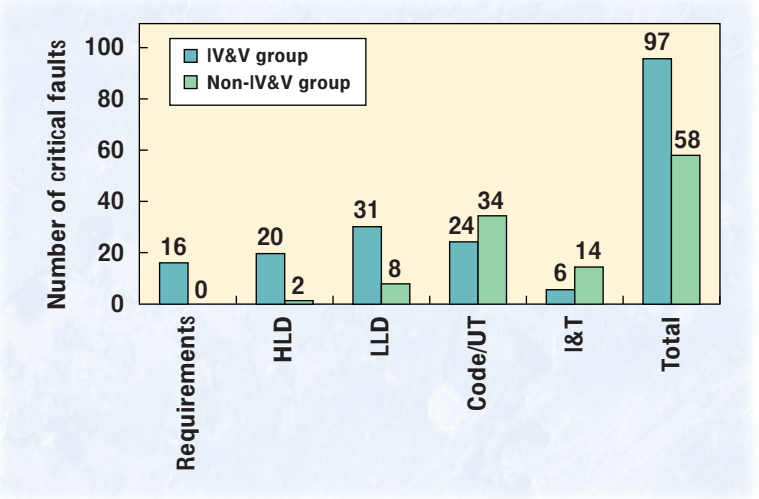

Figure 1. This figure illustrates both teams' fault counts by phase, beginning with the requirements phase and continuing through integration and testing.

REDUCING TIME AND EFFORT

Figure 2 provides a side-by-side comparison of the mean effort required (MER) to remove those faults detected during the development process. The MER being reported is based on observed times recorded in

The Software Engineering Evaluation System

We adapted the following information about SEES from the SEES Executive Summary.[1]

The Software Engineering Directorate within the US Army Missile Command's Research, Development, and Engineering Center developed SEES to define the verification and validation (V&V) tasks that can be performed to evaluate software-intensive systems. It defines a software engineering approach that ensures effective use of the available V&V resources focused on the mitigation of risks inherent to the software development process.

These analytical methods and practices are documented in Technical Assessment Procedures (TAPs) to ensure consistent and repeatable execution by the V&V practitioner. Each TAP focuses on a particular aspect of the software development process and identifies those activities and measures that best support an evaluation of the objectives of that focus. Here is a brief description of each TAP:

- The Requirements Review and Assessment TAP examines input conditions, processing activities, and processing results.
- The Requirements Trace/Completeness Matrix TAP analyzes the completeness of the system requirements allocation to the software requirements.
- The Design Review and Assessment TAP establishes completeness and correctness of the design.
- The Code Review and Assessment TAP assesses the adequacy of the Ada code.
- The Test Plan Assessment TAP determines the adequacy of the software test plan.
- The Test Description Assessment TAP determines the adequacy of the formal qualification test (FQT) description and procedures in accomplishing the tests defined in the STPs.
- The Test Witness and Assessment TAP assists in determining the adequacy of the FQT execution and data results.
- The Independent Test and Evaluation TAP determines the software adherence to requirements and the reliability inherent in the software.
- The Functional Configuration Audit TAP determines whether the testing and test results have been accurately validated.
- The Physical Configuration Audit TAP verifies the adequacy and accuracy of the documentation that establishes the product baseline for configuration management.

Reference

1. US Army Missile Command, *SEES Executive Summary, SED-SES-IES-001*, Software Engineering Directorate, Redstone Arsenal, Ala., 1993.

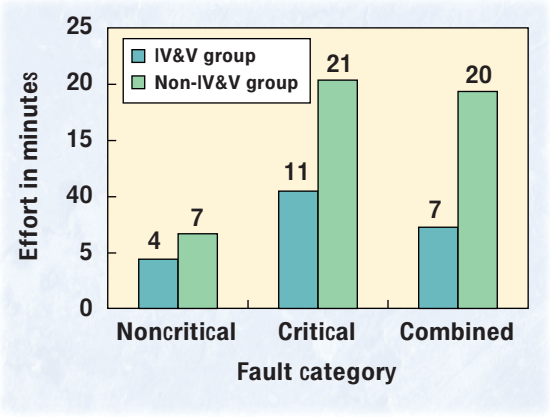

Figure 2. Side-by-side comparison of the mean effort required (MER) to remove faults detected during the development process. We computed average values by dividing the sum of observed values by the number of observations.

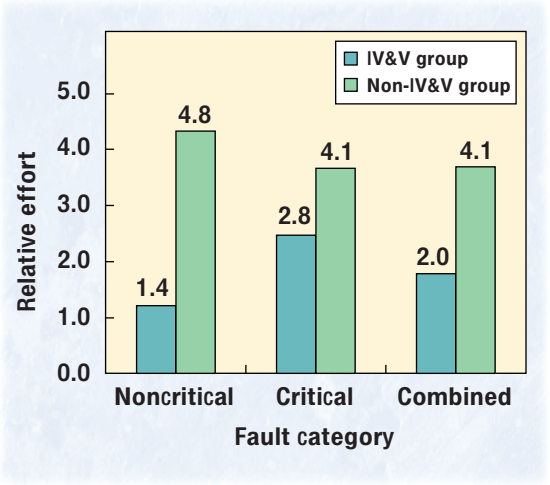

Figure 3. The mean estimated effort required to remove a single fault.

minutes. We assume that IV&V enables faults to be detected earlier in the development process and also reduces the average time to fix faults, so you would expect that over similar periods of time the MER observed for Group 1 would be less than that for Group 2. In each case depicted in Figure 2 this is exactly what happens. In particular, the MER for Group 2 is more than double that of Group 1.

Barry Boehm reports an *estimated* effort value for correcting a fault.[7] He derives that estimate from empirical studies relating effort values to when the fault is introduced in the software development process and when it is subsequently found. The use of Boehm's estimated effort values (in lieu of observed values) tends to negate the adverse impact that observed errant or disproportionate values can have in computing a mean. Figure 3 shows the results of using Boehm's estimated effort-to-remove values as an alternate computation for the mean effort to remove faults.

For both groups the estimated effort value for each recorded fault is determined by noting during which phase the fault was introduced and subsequently detected, and then using those phases to select an estimated effort value from an augmented version of Boehm's effort values. For the figure, we computed an estimation of MER by summing the estimated effort values for each reported fault and then dividing the sum by the total number of faults. In effect, Figure 3 reflects an attempt to remove analytical perturbation introduced by the sensitivity of MER computations to the fluctuations of observed effort-to-remove values.

When using estimated effort values instead of observed values, the MER computations still reflect the same trends depicted in Figure 2 and therefore provide a substantiation of the validity of those trends. That is, through the I&T phase, the average relative effort to remove a fault found by Group 1 is 2.03 units. The average relative effort for Group 2 during the same time frame is 4.11 units, more than double that of Group 1. Hence, we can conclude that SEES′ (and its attendant IV&V activities) did have a beneficial effect in reducing the effort to fix faults.

Readers interested in our statistics should note the following. By using an alpha value of .05, we applied the Wilcoxon rank sum test to the recorded fault removal times; the resulting Z value (–4.25) indicates that the difference in the time to repair faults between the two groups is statistically significant.

GAINING OPERATIONAL CONTROL

In addition to stressing verification of the higher-profile development artifacts like requirements specifications and design documents, IV&V activities also stress verification of the software test plan and software test description, and a rigorous validation of the final software product. Consequently, IV&V also enhances operational correctness.

Acceptance testing is the time when a product is judged relative to its operational correctness. During the I&T phase (and prior to acceptance testing), both systems were subjected to standard validation activities. Each group judged its software system as meeting the original set of requirement specifications. During acceptance testing, however, the application of our independently developed test suite—made up of 36 tests and focused on key aspects of the software system—showed that there was a significant difference between developing with and without IV&V. Table 1 summarizes the results of applying the tests to the software systems.

The software system produced by Group 1 passed 33 of the tests and failed three. The system produced by Group 2 passed only 11 of the tests and failed 25. We attribute this difference to IV&V.

The other claimed benefits of using IV&V include better development documentation and a more controlled development process. One approach to assessing process control is based on measuring the variability of the process itself. In general, we would expect less variance in the time to repair faults for a controlled development process than for one in which control is ineffective or lacking.

Descriptive statistics indicated that the variance in time to fix faults for Group 1 was one eighth that of Group 2 (the non-IV&V group). Furthermore, for those interested in our statistics, an application of the

Table 1. The results of 36 tests applied to Groups 1 and 2.

Test class	Number of tests	Group 1 (IV&V) Passed / Failed	Group 2 Passed / Failed
Duplicate IV&V tests	5	5 / 0	3 / 2
Exceeding weigh limit	4	4 / 0	0 / 4
Exceeding maximum number of interactions	2	2 / 0	1 / 1
Boundary condition tests	5	5 / 0	2 / 3
Input missing or wrong format	5	3 / 2	3 / 2
Input out of bounds	5	5 / 0	0 / 5
Group 1 code particulars	5	5 / 0	0 / 5
Group 2 code particulars	5	4 / 1	2 / 3
Total	**36**	**33 / 3**	**11 / 25**

F-test (with an alpha of .01 and resulting P value of less than .0001) indicated that the difference in variances between the two groups was statistically significant.

Because the development effort having an IV&V process showed less variability in the time it takes to fix faults, and because that reduction is statistically significant, we offer the conjecture that an IV&V development effort can exhibit better control over the development process than a development effort having no IV&V contingent.

Our experiment illustrates the beneficial impact that IV&V activities can have on a development process. We believe that any process stressing the systematic overlay of well-designed IV&V activities onto a structured software development process will bring benefits similar to those outlined here. Data supporting the conclusions stated in this article can be found in an Excel spreadsheet accessible through anonymous ftp to arthur.cs.vt.edu. ❖

Acknowledgments

This work was funded by NASA Langley Research Center under contract NAS1-19610, Task 17.

References

1. B.W. Boehm, "Verifying and Validating Software Requirements and Design Specifications," *IEEE Software*, Jan./Feb. 1984, pp. 75-88.
2. L. Yen, R. Pail, and K. Mori, "Toward Integrated Methods for High-Assurance Systems," *Computer*, Apr. 1998, pp. 32-34.
3. D.R. Wallace and L.M. Ippolito, *A Framework for the Development and Assurance of High Integrity Software*, NIST Special Publication 500-223, Computer Systems Laboratory, NIST, Gaithersburg, Md., 1994.
4. J.B. Dabney and J.D. Arthur, "Anticipating and Mitigating the Professional Challenge to Independent Verification and Validation," *Proc. 16th Ann. Pacific Northwest Software Quality Conf.*, IEEE CS Press, Los Alamitos, Calif., 1998, pp. 84-92.
5. US Army Missile Command, *SEES Executive Summary, SED-SES-IES-001*, Software Engineering Directorate, Redstone Arsenal, Ala., 1993.
6. J.D. Arthur et al., *Report on Quasi-Experiment for Evaluating SEES*, Tech. Report TR-95-24, Dept. Computer Science, Virginia Tech, Blacksburg, Va., 1995.
7. B.W. Boehm, *Software Engineering Economics*, Prentice Hall, Englewood Cliffs, N.J., 1981.

James D. Arthur is an associate professor of computer science at Virginia Tech. His research interests include software engineering, parallel computation, and user support environments. He received a BS and an MA in mathematics from the University of North Carolina at Greensboro and an MS and a PhD in computer science from Purdue University. Contact him at arthur@vt.edu.

Markus K. Gröner is currently pursuing a PhD at Virginia Tech. His research interests include establishing an error-resistant communication framework between clients and software engineers and improving software requirement specifications through better IV&V methods. He received a BS and an MS in computer and information sciences from the University of South Alabama. Contact him at groener@vt.edu.

Kelly J. Hayhurst is a research scientist in the formal methods group at the NASA Langley Research Center, Hampton, Va. Her research interests include reliability analysis, software engineering, and software standards for avionics systems. She received a BS in mathematics from Virginia Tech and an MS in mathematics from the College of William and Mary. Contact her at k.j.hayhurst@larc.nasa.gov.

C. Michael Holloway is a research engineer at the NASA Langley Research Center, Hampton, Va. His research interests include accident analysis, programming language theory, and high-integrity software development techniques. He received a BS in computer science from the University of Virginia, and he did graduate work at the University of Illinois at Champaign-Urbana. Contact him at c.m.holloway@larc.nasa.gov.