Adding Value to the Software Development Process: A Study in Independent Verification and Validation*

James D. Arthur

The Department of Computer Sciences
Virginia Tech
Blacksburg, VA 24061

arthur@vt.edu

Markus K. Groener

The Department of Computer Sciences
Virginia Tech
Blacksburg, VA 24061

groener@vt.edu

Kelly J. Hayhurst

NASA - Langley Research Center
Mail Stop 130
Hampton, VA 23681-0001

C. Michael Holloway

NASA - Langley Research Center
Mail Stop 130
Hampton, VA 23681-0001

Abstract

Independent Verification and Validation (IV&V) is best viewed as an overlay process supporting a software development effort. While the touted benefits of a properly managed IV&V activity are many, they specifically emphasize: (a) early fault detection, (b) reduced time to remove faults, and (c) a more robust end-product. This paper outlines a study funded by NASA-Langley Research Center to examine an existing IV&V methodology, and to confirm (or refute) the touted beneficial claims. In the study two distinct development groups are established, with only one having an IV&V contingent. Both groups are tasked to produce a software product using the same set of requirements. Within each phase of the development effort, fault detection and fault removal data are recorded. An analysis of that data reveals that the group having the IV&V contingent: (a) detected errors earlier in the software development process, and (b) on the average, required significantly less time to remove those faults. Moreover, a test for operational correctness further reveals that the system developed by the group having the IV&V component was substantially more robust than the one produced by the other development group. A statistical analysis of our results is also provide to establish significance.

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

Adding Value to the Software Development Process: A Study in Independent Verification and Validation

1. INTRODUCTION

As computers continue to increase in computational power, so too do the demands of the applications that execute on them. Today, the complexity of many software systems mandates not only a structured approach to software development, but also a verification and validation (V&V) process to ensure that: (a) the product is built right, and (b) that the right product is built [Boehm, 1984]. A V&V process is especially critical for those high-consequence systems where a software failure can result in the loss of life [Yen, 1998], and for systems where "live" testing is too expensive or infeasible, e.g., DoD threat/response systems.

Motivation for the work described in this paper stems from the desire of NASA-LaRC to study the effectiveness of the Army's Software Engineering Evaluation System (SEES) [U.S. Army Missile Command SED, 1993]. A joint investigative effort that included both Virginia Tech and NASA-LaRC personnel led to the design and implementation of a prototype experimental study that examined the benefits of SEES as an IV&V methodology. In particular, the study focused on assessing to what extent, if any, the IV&V activities help in (a) detecting faults earlier in the software development life cycle (SDLC), (b) reducing the mean time to remove those faults, and (c) producing a more robust product.

The study employs two independent development groups, each given an identical set of requirements outlining a solution to an aircraft "sizing" problem. Both groups are tasked to design, code and test their software solution. One of the groups includes an IV&V contingent whose activities are guided by a modified version of SEES. The results of this experiment is reported in Section 4, and illustrate the power of IV&V when it is applied as an overlay process working in tandem with the development effort. More specifically, the results indicate, with statistical significance, that IV&V activities promote early fault detection, reduce the time required to remove those faults, and help produce a more robust product.

The organization of the paper is as follows. Section 2 presents a discussion of V&V activities as they relate to SEES and its modified version, SEES'. Section 3 provides an overview of the experiment design, including the underlying development process, hypothesis generation and data reporting. In Section 4 we present an analysis of the collected data, and state several conclusions which can be inferred from that analysis. Finally, Section 5 provides a summary of our work.

2. VERIFICATION AND VALIDATION: A SEES' PERSPECTIVE

2.1 Verification, Validation, and Independent V&V

Verification and validation are sets of comprehensive activities that examine process and product artifacts throughout the software development lifecycle. The objectives of V&V are to improve the quality and reliability of a system, and to ensure that a system meets the users' stated requirements. Although often appearing together, the two terms imply distinctly different sets of activities.

Verification refers to the process whereby each phase in the SDLC is examined to ensure that the output of a particular phase (a) satisfies all the pertinent requirements of the previous phase, (b) is internally complete and consistent, and (c) can support the development effort in the next phase.

Validation, on the other hand, is primarily concerned with software (and system) testing. During validation, the system is executed, and test results are compared to stated functional and performance requirements. In effect, the correctness of the test results validates the system against the specifications stated in the requirements document.

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 of the development team. Independence promotes objectivity and encourages one to consider a wider range of analytical support tools and solution approaches, which in turn, improves error detection [Lewis, 1993]. Whether IV&V differs from V&V in more than the independence of its practitioners (and derived benefits thereof) is still open for debate -- see, for example, [Fujii, 1994], [Wallace, 1994] and [Arthur, 1998].

2.2 IV&V Goals and Objectives: An SDLC Perspective

Within the conventional waterfall SDLC model IV&V activities are organized according to development phases. Lewis provides a comprehensive list of phase-wise IV&V activities

[Lewis, 1993], many of which are included in our experiment. The major goals and objectives for each phase are summarized below.

- *Requirements Verification*: The goal of requirements verification is to provide the customer with a high confidence that the system is well enough defined to enter the design phase. This activity focuses on a comprehensive examination of the software and interface requirement specifications to ensure that they are consistent with system requirements, unambiguous, complete, internally consistent, testable and traceable.
- *High- and Low-Level Design Verification*: Design verification concentrates on two major objectives: (a) to ensure the necessity and sufficiency of all design components relative to the requirement specifications, and (b) to ensure completeness and consistency among and within the design documents.
- *Code Verification*: Code verification is performed during the implementation and unit testing phase. The primary goals of code verification are: (a) to catch the major portion of coding errors before coded units are placed under configuration management, (b) to ensure that the software design is accurately reflected in the code, and (c) to assess conformance to specified coding practices.
- *Product Validation*: The major portion of validation occurs during the integration and testing phase of the software development process. The goals of validation are twofold: (a) to ensure that every requirement has been qualified through demonstration, analysis or inspection, and (b) to confirm that every performance and functional requirement is adequately tested.

In summary, the criteria often used to *verify* requirements, design and code documents are: completeness, forward and backward traceability, testability, consistency, non-ambiguity, correctness and adherence to standards. The criteria used during *validation* are: testing, test coverage, test accuracy, and conformance to requirements. A more detailed examination of these quality criteria can be found in [Wallace, 1992], [IEEE1012, 1986] and [IEEE1028, 1989] .

2.3 SEES and SEES': A Definition of the IV&V Process

The Software Engineering Evaluation System (SEES) is a *comprehensive* software V&V methodology developed by the U.S. Army Missile Command's Software Engineering Directorate. Comprehensive means that SEES implements a *Full Cycle In-Phase* approach to V&V [Lewis, 1993]. SEES was originally developed to evaluate critical Battlefield Automation Systems and is based on a five step implementation process. In abbreviated form, those steps are:

(1) Assessing the V&V needs for a planned software development project,
(2) Tailoring SEES V&V procedures to fit schedule and budget constraints,

(3) Executing: (a) selected Technical Assessment Procedures (TAPS) which focus on the technical V&V tasks, (i.e., verifying and validating lifecycle objects), and (b) programmatic evaluation tasks that address development process and configuration management issues,
(4) Measure and analyze the results of the V&V activities to identify software risks and deficiencies, and
(5) Provide analytical feedback to improve the quality and reliability of the current system.

SEES TAPs are the "heart" of the evaluation system and provide the technical guidance for V&V activities.

In defining SEES', our implemented version of SEES, several factors were considered. Due to limited resources and time constraints imposed on the experiment, and because of the need for an *Independent* V&V procedure, three major changes to SEES were identified:

(1) First, we adapted SEES, which embraces a V&V philosophy, to include an *independent* component for V&V. We then designed the experiment to include an *independent* V&V team whose only responsibility was to perform V&V on development artifacts produced by one of the two development groups.
(2) Only five of the original ten TAPs were selected for use in the experiment, and only the more critical activities within each TAP were performed.
(3) The reporting forms required by each TAP were appropriately modified to maintain a coherent picture of development and IV&V activities.

Although SEES' is a modified version of SEES, we emphasize that the modifications are such that the *philosophy* and *intent* of SEES remain inviolate.

3. EXPERIMENT DESIGN

3.1 Development and IV&V Activities

The following sections, outlined by development phase, describe the activities of both development teams and the IV&V team.

<u>*Requirements Phase*</u>: The Software Requirements Specification (SRS) was developed prior to the start of the experiment and given to both development groups at the same time. Each group was instructed to review the SRS for understanding and to request clarification where needed. Fault reports were filed for erroneous requirement specifications.

The IV&V team (associated with development Group 1) examined the SRS and developed a Software Requirements List (SRL) in accordance with SEES' guidelines. Each requirement was analyzed by the IV&V team using the SEES' Valid Requirements Criteria and SRS checklist. A Defect Trouble Reports (DTR) were prepared for all fault detected. In a controlled environment, the IV&V team reported its findings to the Group 1 development team. The Group 1 development team then reviewed the DTRs and made appropriate corrections.

Design Phase: The design phase was divided into two sub-phases: high-level design (HLD) and low-level design (LLD). In the HLD phase, both development teams produced a high-level design document that included a structure chart and data flow diagram of major system components and the interfaces among those components. In the LLD phase the two development teams added detail to the structure chart, data flow diagram, and interface design. Each software component was further decomposed into smaller operational units. The resulting artifact became the "code to document". Software Test Descriptions (STDs) were also developed during the LLD. Fault Report Forms (FRFs) were filed by each development team if any of their previously accepted design items had to be changed.

The IV&V team conducted two formal inspections on the documents produced by the Group 1 development team: one on the HLD document, and the other on the LLD document. As stipulated in SEES' the inspection process followed the Fagan Inspection Methodology [Fagan, 1976]. During the inspection meetings each IV&V member presented their individual list of perceived defects. The defects were discussed and, based on group consensus, entered into an Inspection Defect List (IDL). A corresponding set of DTRs was developed. The IDL and DTRs were then given to the Group 1 development team for subsequent action.

Implementation (Code and Unit Test) Phase: During the implementation phase, each development team coded the software components specified in their respective LLD document. Both groups were instructed to use structured programming techniques and to code according to ANSI C standards. Unit testing was also to be performed on each software components during this phase. Any change to design documents or subsequent changes to "baselined" code modules were recorded on FRFs.

The IV&V team examined the code developed by the Group 1 development team to determine (a) the extent to which structured programming practices were followed, (b) how well the code conformed to ANSI-C Coding standards, and (c) the presence (or absence) of code quality characteristics (*vis-à-vis* PC-METRIC [SET Laboratories, 1994]). DTRs were written for deviations from specified standards and practices, and for quality measures falling outside of established threshold values.

Integration and Testing Phase: Both development teams had multiple persons developing software. During integration and testing (I&T) all coded units were integrated to form a complete system. At selected intervals, tests were performed by each development team to ensure that each higher-level function behaved according to expectations. After all units had been integrated into a single software system, additional tests were performed by the development teams to substantiate conformance to stated requirements. As faults were encountered, they were corrected, and a corresponding fault report was produced.

During I&T the primary objective of the IV&V team was to validate the system software produced by the Group 1 development team. Validation was guided by the requirements stated in the SRS. IV&V team activities included: (a) monitoring selected tests being performed by the development team to ensure conformance to specified test procedures outlined in the STD, (b) duplicating selected tests performed by the development team and comparing their results to those reported by the development team, and (c) based on critical path analysis and observation, performing an additional set of tests to assess general system performance and conformance to stated requirements.

3.2 Advancing the Experimental Hypothesis

An experimental hypothesis states a question that the experiment is designed to answer. Essential to confirming or refuting a hypothesis is the identification of the independent variable(s), the dependent variable(s), and a defining (or computational) relationships among them.

The Hypothesis

The general question our experiment is designed to address is:

> "Do the IV&V activities prescribed by the SEES' IV&V approach add value to the existing software development process?"

From this question, a null hypothesis is formulated. In particular, the null hypothesis for this experiment states that no difference exists between software developed with SEES' and software developed without SEES', or more formally:

> "The values of the dependent variables will be the same for the software products developed using SEES' as for the software products developed NOT using SEES' "

The Independent Variables

Independent variables are those variables thought to produce an effect on the dependent variables. The independent variable in the SEES' experiment is *the IV&V method*. The value that an independent variable can assume in an experiment is referred to as the "treatment level" or "treatment condition" of that variable. For the SEES' experiment only two treatment levels or values are assigned to the independent variable. One value is "Use SEES'", while the other value is "Not Use SEES'".

The Dependent Variables

A dependent variable, or response variable, is a variable measured to indicate the effect of the independent variables. Three dependent variables have been chosen to measure the effect of SEES' in the experiment:

- the number of faults,
- the phases in which each fault was introduced/detected, and
- the *Mean Effort to Remove Faults (MER)*.

Hence, to confirm our hypothesis, the use of SEES' IV&V methods by Group 1 in the development process should result in: (a) errors being detected *earlier* in the development lifecycle, and consequently, (b) a *reduced* effort to remove those errors.

For the purpose of our experiment, *MER* is calculated two ways: one is based on a direct observation of effort, and the other on estimated effort values provided by Boehm [Boehm, 1981]. For the direct observation case, i.e. $MER_{observed}$, the formula is as follows:

$$MER_{observed} = \frac{\sum_{m=1}^{n} F_m^{Obs}}{n}$$

where n is the total number of faults reported, and F_m^{Obs} is the observed effort to remove fault F_m. Similarly, the formula for the estimate case, i.e. $MER_{estimated}$ is:

$$MER_{estimated} = \frac{\sum_{m=1}^{n} F_m^{Est}(I, R)}{n}$$

where n is the total number of faults reported, and $F_m^{Est}(I, R)$ is the estimated effort to remove fault F_m inserted during life cycle phase I and detected/removed during phase R.

In support of these computations several data values were recorded for each fault reported. Most pertinent among them are: (a) the phase in which the fault was detected/removed, (b) the phase in which the fault was introduced, (c) the amount of time (effort) required to fix the fault, and (d) the severity category, i.e., critical or non-critical. Using an augmented version of Boehm's table, the effort values used to compute $MER_{estimated}$ are:

- 1.3 for any faults whose insertion and removal phases are the same,
- 4.0 for any faults whose insertion and removal phases are adjacent in the SDLC,
- 10.0 for any faults whose insertion and removal phases are separated by one other phase, and
- 40.0 for all faults whose insertion and removal phases are separated by two phases.

For a more detailed explanation of the derivation of the augmented table see Section 3.1.4, Table-1, page 14 of the *Experimental Design for Evaluating SEES* [Frakes, 1995].

4.0 DATA ANALYSIS AND CONCLUSIONS

By design the experiment outlined above is focused on answering the following question:

> *"Do the IV&V activities defined in SEES' support early fault detection, and consequently, reduce the effort required to remove those faults?"*

The following discussion and supporting graphs attempt to answer that question.

4.1 Detecting Faults Earlier in the Software Development Lifecycle

Graph 1 provides a side-by-side comparison of the number of critical errors detected by Group 1 and Group 2. The counts are categorized by phases, beginning with the requirements phase and continuing through I&T.

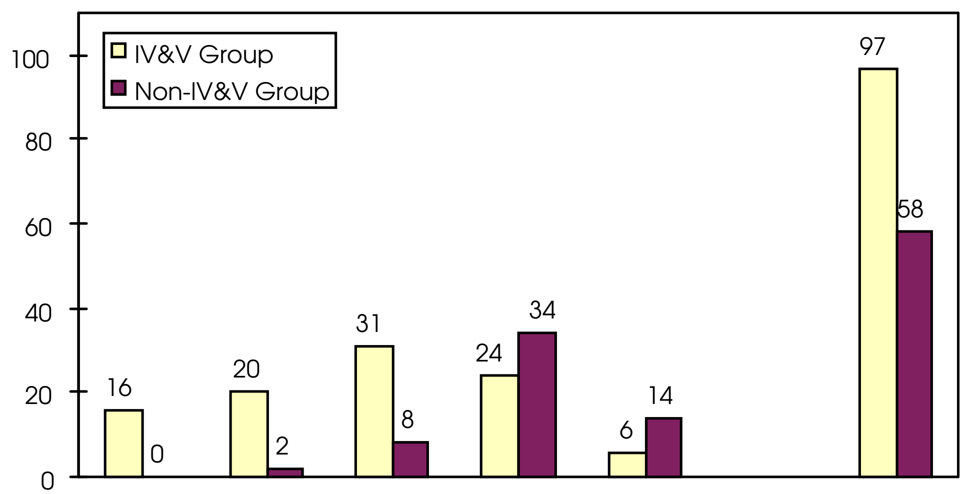
Three observations can be drawn from Graph 1:

(1) The number of faults detected by Group 1 (having the IV&V contingent) peak during Low-Level Design and then fall off substantially through I&T. This trend indicates that the majority of the faults are being detected during the design phases, and then steadily decrease in the later two phases.

(2) The number of faults detected by Group 2 (the Non-IV&V Group) grow exponentially as development proceeds from Requirements through Code&UT, with a significant number still being found during I&T. Experience suggests that the trend illustrated in Graph 1 is more the norm for software development efforts lacking a formal V&V process.

(3) When we compare the fault detection trend for Group 2 to that of Group 1, we observe that Group 2 (a) begins finding faults a phase later, (b) detects fewer faults in the earlier phases, and (c) peaks a phase later.

GRAPH 1
Critical Faults: A Comparison Between Groups



Phase in which Critical Faults Found

Hence, given the assumption that we have equally proficient development teams, the above observations leads us to a preliminary conjecture that: *employing the IV&V activities prescribed by SEES' does promote the earlier detection of faults.*

It is also natural to expect that if a fault is detected earlier in a software development effort, then the time it remains in development artifact (i.e., the time between when it is introduced and detected) is shorter. If, as conjectured above, SEES' does promote earlier fault detection, then we should expect to find that the average time a fault detected by Group 1 has remained in the development artifact is less that the average time for a fault found by Group 2. Figure 1a depicts a categorization of the faults reflecting the frequency which faults were (a) introduced and detected in the *same* phase, (b) introduced and found in *adjacent* phases, (c) had *one* intervening phase between introduction and discovery, or (d) had *two* intervening phases between introduction and discovery. If we let phases be our unit of time, we can compute the average number of phases that a fault remains in the development artifact. Doing so, we find that faults detected by Group 1 are found (on the average) 0.2 phases after the phase in which they were introduced; we find that faults detected by Group 2 are found 0.6 phases after they are introduced. *These averages imply that faults introduced by Group 1 are, one the average, detected sooner than those introduced by Group 2*, and hence, further substantiate the above conjecture.

Establishing Statistical Significance

The question that now comes to mind is: "Is that difference observed above statistically significant?" Or more specifically, we ask: "*Does a statistically significant relationship exist between the use/non-use of IV&V activities and the length of time to find a fault?*"

To answer this question we apply the Chi-Square Test to the observed frequencies listed in Figure 1a. The expected frequency values are provided in Figure 1b; the Chi-Square and Chi Square P-Value are shown in Figure 1d. Is the overall difference statistically significant? The Chi-Square P-Value of .002 indicates "Yes", with extremely high (> 99%) probability. Therefore, we reject a null hypothesis stating that "no significant relationship exists between the use/non-use of IV&V activities and the length of time to find a fault", in favor of the hypothesis stating that "a statistically significant relationship exists between the use/non-use of IV&V activities and the length of time to find a fault."

	Same	Adj	-1-	-2-	Totals
IVV	79	16	0	2	97
NIVV	32	21	3	2	58
Totals	111	37	3	4	155

Figure 1a:
Observed Frequencies by Category

	Same	Adj	-1-	-2-	Totals
IVV	69.465	23.155	1.877	2.503	97.0
NIVV	41.535	13.845	1.123	1.497	58.0
Totals	111.000	37.000	3.000	4.000	155.0

Figure 1b:
Expected Values For Each Category

	Same	Adj	-1-	-2-
IVV	1.309	2.211	1.877	.101
NIVV	2.189	3.697	3.140	.169

Figure 1c:
Cell Chi Squares By Category

Chi Square	14.694
Chi Square P-Value	.0021

Figure 1d:
Summary Table for Category

Based on results described above, we offer the following conclusion:

Given the assumption that we have equally proficient development teams, SEES' (and its prescribed IV&V activities) do assist in the detection of faults earlier in the development process than does a process devoid of IV&V activities.

Moreover, the reduction in time to find those faults is shown to be statistically significant.

One additional observation that the reader might discern from Graph 1, and one on which we would like to comment is:

Why is there such a marked difference between the number of faults reported by Group 1 than the number reported by Group 2?

Two obvious possibilities for the observed phenomenon are:

(1) the development team from Group 1 introduced more faults than Group 2, or

(2) both groups introduce basically the same number of faults, but Group 2 simply failed to detect them.

We tend to discount both (1) and (2) above because:

- the process by which personnel were assigned to groups focused on maintaining equal development capabilities between the groups (see Appendix A), and
- preliminary results from acceptance testing [Arthur, 1995] indicates that the software produced by *both* groups meet the basic set of stated requirements.

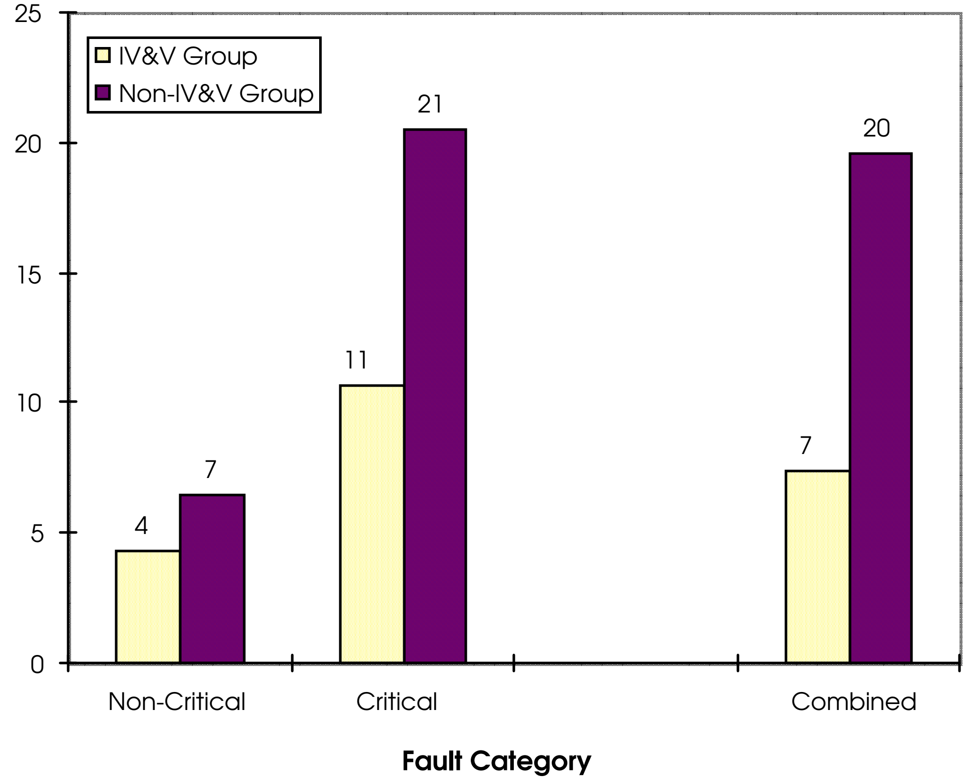
A detailed examination of the data reported by Group 1, however, reveals a plausible explanation: *the IV&V Team from Group 1 identified an additional class of "critical" faults not found by Group 2*. More specifically, a substantial number of the (critical) faults reported by Group 1 stems from the detection and recording of ambiguous or unclear statements in the requirement specifications and design documents which, if misinterpreted or misunderstood, *could* result in the introduction of critical faults downstream in the development process. 40 such faults are reported by Group 1 (14 requirements, 18 High-level Design, and 8 Low-level Design); only one such fault is reported by Group 2.

4.2 Reducing the Time and Effort to Remove Faults

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

Based on our assumption that IV&V enables faults to be detected earlier in the development process, and therefore reduces the average time to fix faults, one would expect that over similar periods of time the $MER_{observed}$ for Group 1 would be less than that of Group 2. *In each case depicted in Graph 1 this is exactly what happens*! In particular, if one examines the combined category, we observe that *the MER for Group 2 is more than double that of Group 1.*

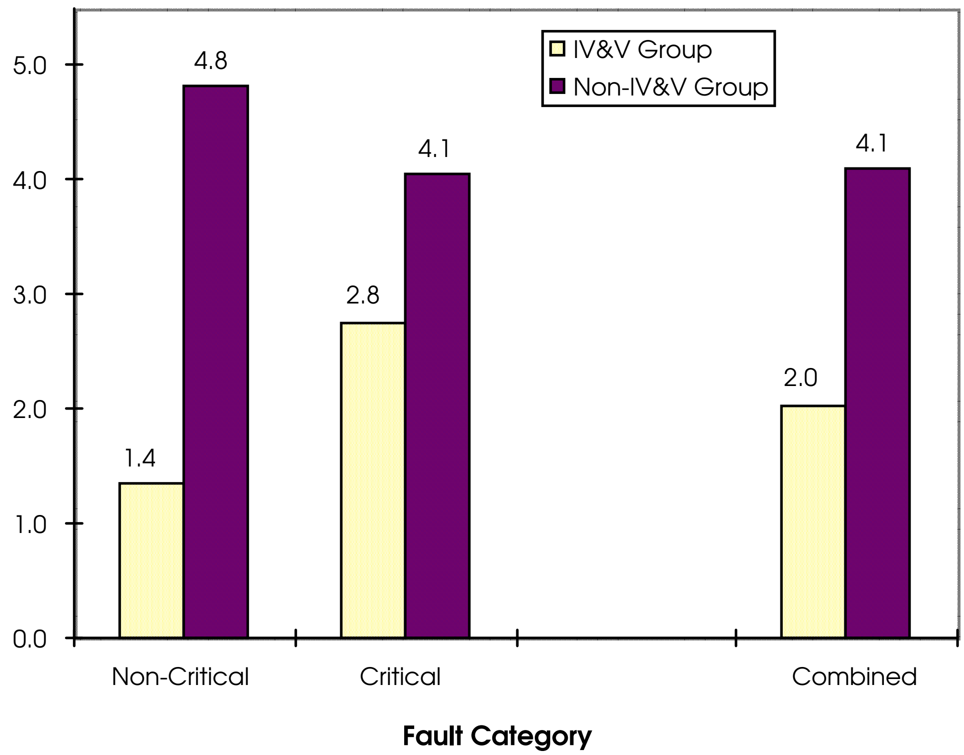
GRAPH 2
Mean Effort to Remove Faults
-MER Observed-



In Graph 2 the average (mean) values are computed by dividing the sum of observed values by the number of observations. Major fluctuations in either the observed values or in the number of observations, however, can have a significant impact on the reported average. In [Boehm, 1981], Boehm reports an estimated effort value for correcting a fault. That estimate is derived from empirical studies relating effort values to: (a) when the fault is introduced in the software development process, and (b) 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 outlier values can have in computing a mean. The following graph illustrates the results of using Boehm's *estimated* effort-to-remove values as an alternate computation for the mean effort to remove faults. This alternate computation is appropriately termed: $MER_{estimated}$.

Graph 3 illustrates the mean estimated effort required to remove a single fault. For both groups the estimated effort value for each recorded fault is determined by: (1) noting during which phase the fault was introduced and subsequently detected, and then (2) using those phases to select an estimated effort value from an augmented version of Boehm's table. $MER_{estimated}$ is computed

GRAPH 3
Mean Effort to Remove Faults
-MER Estimated-



by summing the estimated effort values for each reported fault and then dividing the sum by the total number of faults. In effect, Graph 3 reflects an attempt to remove analytical perturbation introduced by the sensitivity of *MER* computations to the fluctuations of observed effort-to-remove values. More stability is introduced through the use of fixed (estimated) values.

Using estimated effort values instead of observed values, we note that *the MER computations still reflect the same trends depicted in Graph 2*, and hence, provide a substantiation of the validity of those trends. That is, through I&T the average relative effort to remove a fault found by Group 1 (IV&V) is 2.03 units. The average relative effort for Group 2 (Non-IV&V) during the same time frame is 4.11 (again, more than double that of Group 1).

Guided by the above observations and by the results depicted in Graph 2 and Graph 3, we now offer a second conjecture: *SEES' (and it's attendant IV&V activities) does have a beneficial impact in reducing the effort to fix faults*.

Establishing Statistical Significance

Graphs 2 and 3 support a reasoned conjecture that IV&V has a positive impact on reducing the time to repair faults. More specifically, the mean time (in minutes) to repair faults for the IV&V group was 10.7; the mean time for the Non-IV&V group was 20.5. The question we ask again is: *Is that result statistically significant?*

To address this question we would like to use the T-Test to compare the mean time to repair faults for the two groups. In particular we examine the difference between the means and determine the probability that the observed difference could have occurred by chance. Assumptions for the T-Test are (a) the distribution samples are normally distributed (but this can be relaxed), and (b) the variances do not differ significantly. An analysis of variance (reported later in this paper) indicates, unfortunately, that they do differ significantly. Hence, we cannot use the T-Test to test significance in the difference of the means.

An alternative approach is to employ a non-parametric, distribution free statistical method to examine significance. The Wilcoxon Rank Sum Test is ideal for such scenarios, and approaches the significance question by examining distribution values based on assigned ranks. It assigns ranks to values in the combined distribution and the compares the medians of the ranks of two distributions. One benefit of the Wilcoxon Rank Sum Test is that because it is based on ranks rather than actual data values, it is not sensitive to outliers.

Like the T-Test, the Wilcoxon Rank Sum Test also assumes that the variances do not differ significantly. To remove the variability, each value in the two distributions was divided by its respective standard deviation - effectively normalizing the variances by making the standard deviation equal to one.

In applying the Wilcoxon Rank Sum Test, our null hypothesis is: "Given the actual times to repair faults for the two development efforts, there is no significant difference between their medians (or means). To reject this hypothesis at the .05 level (alpha) we need a Z value <= -1.96 or >= 1.96. The Wilcoxon Rank Sum Test provided a Z = -4.25. Therefore, *with better than 95% assurity, we can conclude that the actual time to repair faults between the two sets of data is statistically significant, and subsequently, can reject the null hypothesis*.

We now offer a second conclusion:

As illustrated in Graphs 2 and 3, the IV&V activities prescribed by SEES' do have a positive impact in reducing the effort to remove faults.

Furthermore, that reduction in effort (or time) is shown to be statistically significant.

4.3 Assessing Operational Correctness

In addition to the higher-profile development artifacts like the requirements specifications, design documents and code, IV&V activities also stress the verification of the software test plan and software test description, and a rigorous validation of the final software product. Consequently, an additional expected benefit of IV&V is *enhanced operational correctness*. Typically, Acceptance Testing is the time when a product is judged relative to its operational correctness.

Prior to Acceptance Testing, both systems were subjected to validation activities during the I&T phase. Based on the set of tests employed by each group, both systems were judged to meet the *original* set of requirement specifications. We stress "original" because each group independently refined their respective set of specifications during separate question and answer sessions.

The test suite described below focuses on many aspects of the software system. As is often the case, and as is reflected in the test suite, those aspects include both stated and assumed customer requirements. For example, the original set of requirements says nothing about checking for invalid input data. Nonetheless, through question and answer periods both development groups evolved their own set of input validation criteria.

4.3.1 The Test Suite

The test suite consists of 36 independent tests comprising eight distinct test classes. Each class is described below:

- Duplicate IV&V Tests: Test Class One consisted of five (5) tests that are duplicates of those used by the IV&V team in Group 1. While such a set might appear to be biased toward Group 1, they still reflected the expectations of the customer.
- Maximum Weight Limit: Test Class Two is composed of four (4) tests that focused on various parameters which causes the aircraft weight limit to be exceeded. This situation occurs when an unrealistic set of aircraft parameters is provided as the initial conditions. In each case our expectations were that the software system would terminate execution with a "weight exceeded" message.

- Maximum Iteration Limit: Test Class Three includes input parameters that are expected to result in a non-convergent process by exceeding the maximum number of allowed iterations. Again, our expectations (and those stated in the requirements) are that the software system will successfully terminate with an "iteration limit exceeded" message. Two (2) such tests were defined.
- Boundary Condition Tests: Test Class Four exploits prior knowledge about the algorithms that had to be used in the software product. The five (5) tests included in this class focus on conditions that tests the boundaries of those algorithms.
- Missing or Incorrectly Formatted Input Data: Test Class Five is composed of five (5) tests that have missing input data elements or input data that are not in the expected format, e.g., a real number is provided when an integer is expected.
- Input Out Of Bounds: Test Class Six also determines the ability of each system to handle input errors, but restricts test cases to input data that fall outside of a specified range. This test class is composed of five (5) separate test.

Tests cases comprising the final two classes reflect knowledge gained through an examination of each group's code. The examinations revealed both desirable and undesirable characteristics in both code sets. Based on such observations, two additional sets of tests were independently developed.

- IV&V Code: Test Class Seven includes five (5) tests derived from an examination of the code produced by Group 1.
- Non-IV&V Code: Test Class Eight includes five (5) test that reflected particulars of the code developed by Group 2.

4.3.2 A Test of Operational Correctness

Table 1 summarizes the results of applying each of the 36 tests (outlined above) to the two software systems.

We reiterate that upon completion of I&T both products appeared to meet the specifications stated in the requirements. Nonetheless, an examination of Table 1 reveals that the software system produced by the Group 1 passed 33 of the tests and failed 3. On the other hand, the system produced by Group 2 passed only 11 of the additional tests and failed 25. We attribute the results to the following two causes:

- Because the development process followed by Group 1 included a formal *verification* process

(a) the evolved set of requirements were substantially better than those used by Group 2, and more accurately reflected the expectations of the customer, and

(b) the design artifacts consisted of only the necessary and sufficient components, and left little room for misinterpreting them during code development.

- Because the development process followed by Group 1 included a formal *validation* process

(a) the software system produced by Group 1 was subjected to a more comprehensive set of tests that better covered *even the original set of stated requirements*, and

(b) the objectivity of the IV&V team reduced the tendency to overlook crucial test procedures and to be misguided by preconceived perceptions normally acquired by participating directly in the development process.

Test Class	**Number of Tests**	**Group 1 (IV&V)**		**Group 2 (Non-IV&V)**	
		Passed	Failed	Passed	Failed
Duplicate IV&V Tests	5	5	0	3	2
Exceeding Weigh Limit	4	4	0	0	4
Exceeding Maximum Number of Iterations	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 Cod Particulars	5	5	0	0	5
Group 2 Cod Particulars	5	4	1	2	3
	36 **Total**	33 **Passed**	3 **Failed**	11 **Passed**	25 **Failed**

Table 1: Operational Correctness

We now offer a third conclusion, substantiated through an examination of the data presented in Table 1:

Applying the IV&V activities prescribed by SEES' results in a product whose operational correctness (and robustness) substantially exceeds that of the product developed by the parallel effort having no IV&V contingent.

4.4 Controlling the Software Development Process

During the course of our experiment, we observed several ancillary benefits IV&V activities, i.e., better development documentation and a more controlled development process. In this particular subsection we examine the latter benefit and, using statistical analysis provide a fourth conclusion.

One approach to assessing process control is based on measuring the variability of the process itself. From an IV&V and software engineering perspective we have elected to examine the variability in times to repair faults. In general, we would expect less variance in the time to repair faults for a controlled development process than for one where control is ineffective or lacking.

Correspondingly, our null hypothesis is: "Relative to IV&V and non-IV&V development efforts, no significant difference exists in the variability of the time it takes to repair faults." To test the null hypothesis we use the F-Test and examine the ratio of the variances. An analysis of the data reveals that the assumption of a normally distributed data space does not hold. This assumption, however, can be (and is) relaxed. The F-Test assumes that the ratio of the variances is 1, i.e., equal variances, and then uses the F-Distribution to compute the probability that the ratio actually observed occurs by chance. Figure 2 summarizes the descriptive statistics related to the time (in minutes) recorded by each group to repair faults. Note that the variance for the Non-IV&V group is almost eight (8) times higher than that for the IV&V group. Could this disparity have occurred by chance?

	IV&V	Non-IV&V
Mean	10.667	20.517
Std. Dev.	24.325	64.098
Count	81	58
Minimum	1.000	1.000
Maximum	180.000	450.000
Variance	591.700	4108.605

Figure 2: Descriptive Statistics

Figure 3 provides the results of applying the F-Test to the two sets of recorded times to repair faults. The P-Value reveals that probability of such disparity occurring by chance is less than 0.0001.

Hypothesized Ratio = 1

	Var. Ratio	Num. DF	Den. DF	F-Value	P-Value	99% Lower	99% Upper
IVV, Non-IVV	.144	80	57	.144	<.0001	.080	.253

Figure 3: Variance Comparison for Actual (Minutes)

Hence, we reject the null hypothesis in favor of the hypothesis which states that the difference between the IV&V and non-IV&V development efforts is significant and does not occur simply by chance. Because, (a) the variability in time to fix faults is an indicator of control (or lack thereof) in a development effort, (b) the difference in variability between the two activities is statistically significant, and (c) the presence of an IV&V component is the major distinguishing characteristic between the two development efforts, we conclude the following:

The development effort having an IV&V process shows less variability in the time to fix faults than does the Non-IV&V effort. That reduction is statistically significant and leads to the conclusion that the IV&V development effort exhibits better control over the development process.

5. Summary and Final Observations

The experiment described above focuses on answering the question:

Does SEES', with its prescribed set of IV&V activities, add value to a software development process?

The results outlined in Section 4 indicate that SEES' and its prescribed IV&V activities *do* have a beneficial impact on the software development process. As argued above those benefits include

- the detection of faults earlier in the software development lifecycle,
- the reduction in the effort required to repair individual faults, and
- enhanced operational correctness.

Although often overlooked, the experiment reveals two additional benefits of a development process including IV&V activities:

- the detection of ambiguous or unclear specifications that can lead to the introduction of critical faults downstream in the development process, and
- better control over the software development process.

SEES, as defined by the US Army, describes one particular approach to implementing V&V activities. SEES', a minimally modified subset of SEES, is the vehicle through which our particular IV&V activities were implemented. There are, however, elements of SEES that tend to reduce its effectiveness [Arthur, 1997]. We view several of the modification to SEES (which are embodied in SEES') as enhancements that have helped streamline the enunciated IV&V process.

As a final note we make the following conjecture:

Our experiment illustrates the beneficial impact that IV&V activities can have on the development process. SEES is designed to enforce the implementation of many of those IV&V activities. Nonetheless, we conjecture that *any* process which stresses the systematic overlay of well designed IV&V activities onto a structured software development process will bring benefits similar to those outlined in this report.

Data supporting the conclusion stated in this paper can be found in [Arthur, 1995].

REFERENCES

Arthur, J.D. and Groener, M.K. (1997), "An Assessment of Sees Based on Operational Experiences, " Technical Report 97-07, Department of Computer Science, Virginia Tech, Blacksburg, VA.

Arthur, J.D., Groener, M.K., Gupta, S., Cannon, M., and Z. Khan (1995), "Report On Quasi-Experiment For Evaluating SEES," Technical Report TR-95-24, The Department of Computer Science, Virginia Tech, Blacksburg VA.

Boehm, B.W. (1984). "Verifying and Validating Software Requirements and Design Specifications," *IEEE Software*, Vol. 1, No. 1, pp. 75-88.

Boehm, B. W. (1981). *Software Engineering Economics*. Englewood Cliff, N.J.: Prentice-Hall.

Dabney, J.B. and Arthur, J.D. (1998), "Anticipating and Mitigating the Professional Challenge to Independent Verification and Validation," Technical Report TR 98-04, Computer Science department, Virginia Tech.

Fagan, M.E. (1976), "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal*, Vol. 15, No. 3, pp. 182-211.

Frakes, W.B., Arthur, J.D., Cannon, M., & Gupta, S. (1995). *An Experimental Design for Evaluating SEES (NAS1-19610, No. Task 17)*, The Department of Computer Science, Virginia Tech Graduate Center, Falls Church, VA.

Fujii, R.U. (1994). "Independent Verification and Validation," In J. J. Marciniak (Ed.), *Encyclopedia of Software Engineering*, New York: John Wiley & Sons, Inc., pp. 568-572.

IEEE1012 (1986). *IEEE Standard for Software Verification and Validation Plans*, ANSI/IEEE Std. 1012-1986, IEEE, Inc., November 1984.

IEEE1028 (1989). *IEEE Standards for Software reviews and Audits*, ANSI/IEEE Std. 1028-1988, IEEE, Inc., 1989.

Lewis, R.O. (1993). *Independent Verification and Validation: A Life Cycle Engineering Process for Quality Software*. New York, NY: John Wiley & Sons, Inc.

Radatz, J.R. (1981). *Analysis of IV&V Data*, RADC-TR-81-147, Rome Air Development Center, Griffiss AFB, N.Y., June 1981.

SET Laboratories (1994). *The PC-METRIC Analysis System*, Set Laboratories, Inc. Mulino, OR.

U.S. Army Missile Command (1993). *SEES Executive Summary*, SED-SES-IES-001. Redstone Arsenal, AL: Software Engineering Directorate.

Wallace, D.R. and L.M. Ippolito (1994). *A Framework for the Development and Assurance of High Integrity Software*, NIST Special Publication 500-223, Computer Systems Laboratory, NIST, Gathersburg MD., December 1994.

Wallace, D.R., Peng, W.W. and L.M. Ippolito (1992). *Software Quality Assurance: Documentation and Reviews*, NISTIR 4909, Computer Systems Laboratory, NIST, Gathersburg MD., September 1992.

Yen, I., Pail, R. and Kinji Mori (1998). "Toward Integrated Methods for High-Assurance Systems", *IEEE Computer*, Vol. 31, No. 4, pp. 32 - 34.

Appendix A

Group Composition and Sample Size

The experiment described in this paper employs two distinct, non-interacting groups. Group 1, itself, is composed of two individual teams: a 3-person development team and a 4-person IV&V team. The IV&V team assesses the quality of the development artifacts produced by the development team. Group 2 has no IV&V contingent and is compose only of a 2-person development team.

In assigning team members one generally prefers to use 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.

Experiment Controls

Although random assignment could not be used, several controls were applied to the experiment to help minimize bias and improve rigor.

- Matching assignment was used to achieve parity in development skill levels, thereby avoiding the high probability of biased subject combinations due to small sample size.
- To ensure that both groups were "singing to the same sheet of music", the baseline development document (i.e., the Software Requirements Specification) was developed prior to the experiment, and was distributed to and discussed with both groups at the same time.
- To ensure that both groups developed the application under similar conditions, a single DEC-Alpha running UNIX was used as the development platform by both groups.
- To minimize interaction between Group 1 and Group 2, members of each group were instructed not to discuss its development effort with members of the other group. Furthermore, separate password protected accounts were set up for each group.
- Interaction between Group 1's development and IV&V teams was also carefully controlled. The IV&V team conveyed its findings to the development team only in the presence of the experiment designer.
- Finally, to minimize errors in measurement and data collection, an individual who was not a member of either treatment group was assigned the responsibility to gather, review, organize, and confirm the validity of the data recorded by Groups 1 and 2.