Reducing the Mean Time to Remove Faults Through Early Fault Detection: An Experiment in Independent Verification and Validation*

James D. Arthur
Markus K. Groener

The Department of Computer Sciences
Virginia Tech
Blacksburg, VA 24060

arthur@vt.edu
groener@vt.edu

Sanjay Gupta
Martha Cannon
Zakia Khan

The Department of Computer Sciences
Virginia Tech Graduate Center
Falls Church, VA 22042

gupta@cyrano.cs.vt.edu
mcannon@ainur.cs.vt.edu
zkhan@aimur.cs.vt.edu

Abstract

This paper presents the results of a study investigating the extent to which Independent Verification and Validation (IV&V) impacts early fault detection, and consequently, the mean time to remove faults. Two separate development groups, one with an IV&V team, are tasked to produce a software system from the same set of requirements. For each development phase, fault detection and 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 substantially less time to remove those faults.

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

Reducing the Mean Time to Remove Faults Through Early Fault Detection: An Experiment in Independent Verification and Validation

1. INTRODUCTION

As computers continue to expand 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]. The V&V process is especially crucial for those safety critical systems where a software failure can result in the loss of life, and for systems where "live" testing is too expensive or infeasible, e.g., DoD treat/response systems.

V&V, and in particular *independent* V&V (IV&V) promotes the development of software systems that meet stated requirements through methods and activities that encourage: (a) early fault detection, (b) enhanced operational correctness and (c) improved product maintainability. The findings presented in this paper focuses (a) above. That is, it describes the results of a controlled experiment designed to *assess the impact that IV&V has on early fault detection*. As described in several studies, e.g., [Boehm, 1984] and [Radatz, 1981], the earlier a fault is detected in the development process, the less effort required to remove that fault. At the request of our sponsor, NASA - LRC, the application selected for development is a software system that computes the "ideal" weight of an aircraft, given performance and aerodynamic characteristics, and an initial "best guess" estimate of particular weight components.

The study employs two independent development groups, each of whom design, code and test the aircraft "sizing" application from the same set of requirements. One group includes an IV&V team which assists in a phase-wise evaluation of the development artifacts. The IV&V team uses a modified version of the Army's Software Engineering Evaluation System (SEES) [U.S. Army Missile Command SED, 1993] as a guide for its IV&V activities. The results, reported in Section 4, illustrate the power of IV&V when it is applied as an overlay process operating in tandem with the development effort. More specifically, the results substantiate the utility of an IV&V process in (a) detecting faults earlier in the software development lifecycle (SDLC), and (b) reducing the time required to remove those faults.

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, group composition, inter- and intra- group communication, 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

The primary objective of this section is to provide: (a) working definitions of "V&V" and "IV&V", and (b) an overview of SEES and its modified counterpart, SEES'. We view both of these as necessary items of discussion because they are integral parts of the experiment outlined in Section 3.

2.1 Verification and Validation

Verification and validation is a process whereby a system is comprehensively analyzed and tested during each phase of the SDLC. In general, the V&V group is closely linked to the development team. The purpose of V&V is to improve the quality and reliability of a system, and to ensure that a system meets the users' stated requirements.

In the context of the V&V process, the terms *verification* and *validation* imply distinctly different activities. Verification refers to the process whereby each phase in the SDLC is examined to ensure that:

- the output of a particular phase satisfies all the pertinent requirements of the previous phase,
- the output is internally complete and consistent, and
- the output can support the requirements of the next phase.

Thus, verification focuses on: (a) the intrinsic correctness of the products of a software development phase, and (b) the relation of each phase's product(s) to immediately preceding and succeeding development phases.

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 functional and performance requirements defined in the system and software requirements specifications. In effect, the correctness of the test results validates the specifications stated in the requirements document.

2.2 Independent Verification and Validation

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 from the development team provides IV&V analysts with an objectivity and wider perspective that improves error detection [Lewis, 1993]. Whether IV&V differs from V&V in more than the independence of its practitioners is still open for debate -- see, for example, [Fujii, 1994] and [Wallace, 1994].

Assuming the conventional waterfall model as a guide for software development, 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 were 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 quality characteristics 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 quality 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]. In the remainder of this section we outline the SEES approach to V&V and describe corresponding modifications to form SEES', the set of IV&V procedures used in our experiment.

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. There are ten separate TAPs, each of which prescribes specific V&V activities. A brief description of each TAP *used in the SEES' experiment* follows below. (We note, however, that the descriptions reflect a SEES characterization; slightly modified versions are used in SEES'.)

- *Requirements Review and Assessment TAP (RRATAP)*: The RRATAP is a requirements verification methodology. A list of software requirements (the SRL) is developed from the Software Requirements Specifications. For each identified requirement a Requirements Analysis Diagram (RAD) is used to document each input conditions, processing activities, and processing results. Checklists are then used to evaluate the completeness, traceability, nonambiguity and other verification quality factors of each software requirement. All defects are noted on the RADs, and a problem report (the Defect Trouble Report, or DTR) is generated.

- *Requirements Trace/Completeness Matrix TAP (RTMTAP)*: The RTMTAP is a verification procedure used to link system level requirements to their software counterparts and to ensure that all software requirements can be correctly traced back to a system requirement. A traceability matrix used to link system and software level requirements.

- *Design Review and Assessment TAP (DRATAP)*: The DRATAP defines a formal inspection process for verifying design documents. It describes the verification of preliminary and detailed design documents as well as system level design documents. The inspection process defined in the DRATAP is based on the Fagan Inspection Methodology [Fagan, 1976].

- *Code Review and Assessment TAP (CRATAP)*: The CRATAP includes a manual inspection process to ensure that the code is consistent with design documentation, and an automated process to compute various software quality metrics and adherence to approved programming language standards.

- *Independent Test and Evaluation TAP (ITETAP)*: The ITETAP is an optional procedure used to determine software adherence to requirements and to assess system reliability.

As previously stated SEES provides a comprehensive approach to V&V. Because of limited resources and time constraints imposed on the experiment, and because of the need for an *Independent* V&V procedure, three major changes have been made to SEES to produce SEES'.

(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 the five TAPs listed above 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.

It is important to note, however, that although SEES' is a modified version of SEES, the modifications were such that the *philosophy* and *intent* of SEES remained inviolate.

3. EXPERIMENT DESIGN

An experiment design focuses on the establishment of criteria to control biases and to protect the validity of the results. The experiment described in this paper comprises two groups, each developing the same software application. Group 1 is composed of an IV&V team and a software development team. Group 2 consists only of a development team and performs its own "V&V". The objective of the experiment is to determine the impact (if any) of IV&V on the software development effort. The application used in the experiment is an aircraft "sizing" problem that seeks to find an appropriate aircraft weight given predetermined performance, aerodynamic and initial (best guess) weight characteristics. A more extensive discussion of the experiment design is given below.

3.1 Class Material, Team Composition and Sample Size

During the summer of 1995, seven (7) students participated in a graduate class focusing on the study of IV&V activities within the SDLC. The objectives of the class were twofold: (a) to study the concepts underlying IV&V, including the benefits and costs of IV&V, and (b) through an experimental study, investigate how, and to what extent, IV&V impacts fault detection and the effort required to remove those faults. Class material was presented from two perspectives. The first perspective emphasized conventional V&V concepts as described by Lewis [Lewis, 1993]; the second focused on the SEES approach to V&V. As each development phase was introduced, conventional V&V activities were presented first, followed by the SEES approach for comparison and subsequent implementation.

The students in the class, designated as Group 1, were divided into two separate teams: a development team (comprising three students) and an IV&V team (composed of four students). The IV&V team assessed the quality of the development artifacts produced by the development team. A second "development only" team, designated as Group 2, consisted of two students who were paid wages to develop the same application. Members of Group 2 were not members of the class and did not participate in any of the class activities. Our expectations were, that by comparing the number of faults detected by each group and the mean time required to remove those faults, definitive conclusions could be made regarding the impact of IV&V on early fault detection and on reducing the mean effort required to remove those faults. In effect, Group 1 served as the experimental group, while Group 2 was viewed as the control group.

Because of the small number of participants (seven in class and two outside of class), however, there existed a low probability that random subject assignment would lead to development teams with equivalent levels of software engineering expertise. We elected, therefore, to use matching assignments according to class and work experience histories to produce teams having similar skill levels. We do recognize, that because of the small sample size and non-random assignment, no formal statistical significance can (or should) be assumed. Nonetheless, we do view the results as one additional data point relating the impact of IV&V on a software development effort.

3.2 Development and IV&V Activities

The following sections describe the development activities of the Group 1 and Group 2 development teams, and the IV&V activities of the Group 1 IV&V evaluation team.

Requirements Phase:

The Software Requirements Specification (SRS) was developed prior to the start of classes 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 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. Defect Trouble Reports (DTRs) were prepared for each fault detected. In a controlled classroom 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 groups 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 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 fault report forms.

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, (b) duplicating selected tests performed by the development team and compared 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.

3.3 Experiment Controls

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

- To eliminate any impact that grading might have on the production of a better quality product, each student was instructed that grades in the class were not based on the quality of the product produced, nor on the number of faults detected.
- To minimize interaction between Group 1 and Group 2, members of each group were instructed not to speak 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 class instructor.
- 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 before the first class meeting, 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.
- 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.
- 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.

3.4 Advancing the Experimental Hypothesis and Supporting Computational Variables

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. In general, the model relating the two class of variables is:

$$DV's = f\ (IV's) + \varepsilon$$

This equation means that the values of the dependent variables are determined, to a large extent, by some function of the independent variables. The following subsections provide an elaboration of these experiment components.

The Hypothesis

The general question this experiment is designed to address is:

> "Does the SEES' IV&V method 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 will be 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. Two dependent variables were chosen to measure the effect of SEES' in the experiment:

- the number of faults and the phase in which each fault was 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:

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

where n is the total number of faults reported, and $F_m(I,R)$ 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

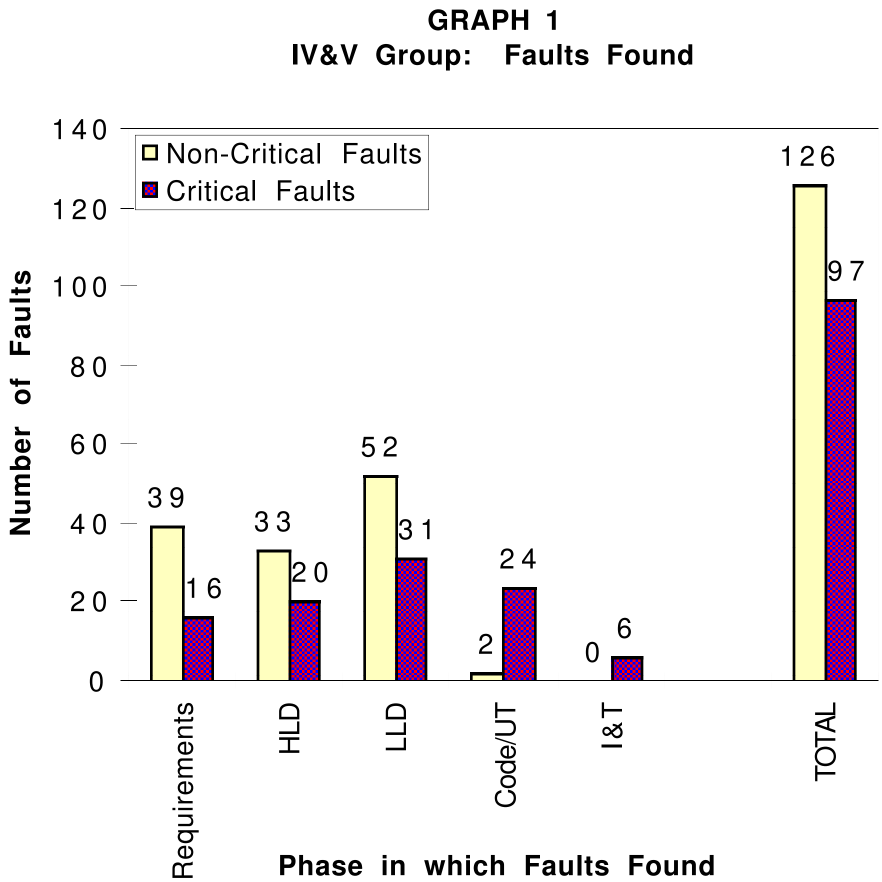
By design the experiment outlined above is focused on answering the following question:

> *"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.

Graph 1 illustrates the number of critical and non-critical faults found by Group 1. The counts are categorized by phases, beginning with the requirements phase and continuing through I&T. The faults reported during the requirements phase are attributed exclusively to the IV&V group.

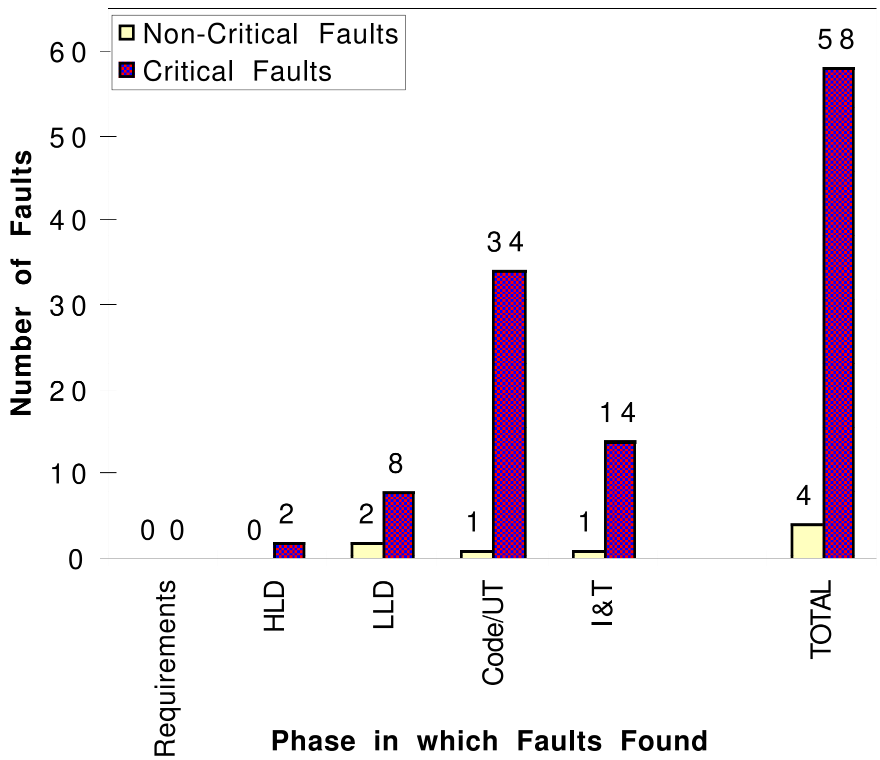
The primary observation to be made from Graph 1 is that the number of faults peak during Low-Level Design and then fall off substantially through I&T. This trend indicates that the majority of the faults are being detected during the design phases, and then steadily decrease in the later two phases. This trend holds for both critical and non-critical faults.



Graph 2 depicts the pattern of fault detection for Group 2, the non-IV&V development group. It, too, categorizes the fault counts within development phases beginning with Requirements and continuing through I&T.

As illustrated in Graph 2, and in particular for critical faults, the number of faults detected grow exponentially as development proceeds from Requirements through I&T, with a significant number still being found during I&T. When compared to Graph 1, faults are, in fact, being detected later in development process, and at a progressively higher rate. Experience suggests that the trend illustrated in Graph 2 is more the norm for software development efforts lacking a formal V&V process.

GRAPH 2
Non-IV&V Group: Faults Found

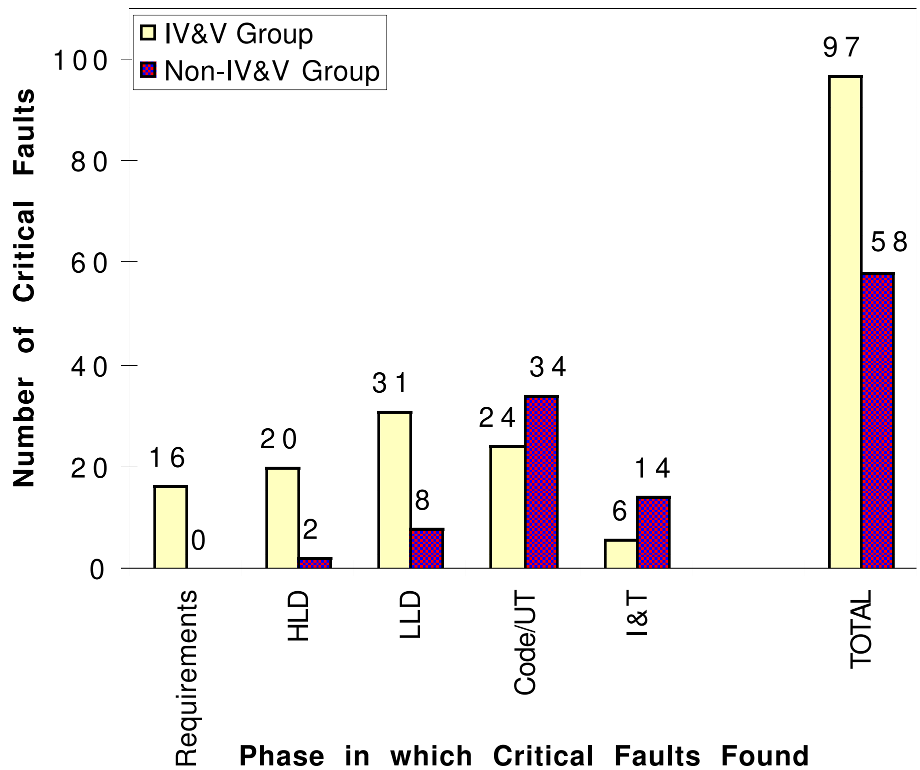


Conclusion 1:	**Given the assumption that we have equally proficient development teams, a comparison of Graph 1 and Graph 2, indicates that employing SEES' during the development process has a beneficial impact of enabling the earlier detection of faults**.

Graph 3 focuses attention on critical errors only, and provides a side-by-side comparison of fault detection for both groups. This pictorial comparison better illustrates the Conclusion 1.

What Graph 3 also illustrates, however, is that Group 1 detected almost double the number of faults reported by Group 2.

GRAPH 3
Critical Faults: A Comparison Between Groups



Given the above observation, we must ask:

> *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 that

(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:

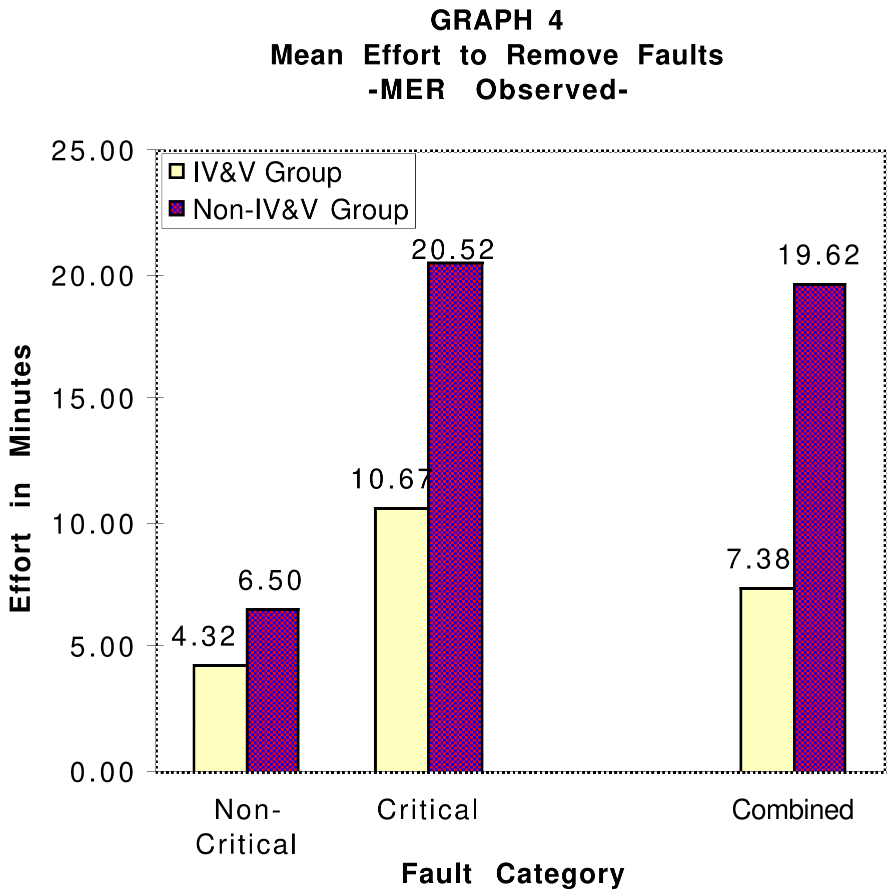
- both development groups are assumed to be equally capable in their development capabilities, and

- preliminary results from acceptance testing 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 more plausible explanation, that is, *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.

Conclusion 2:	**Applying SEES' to the development process enables the early identification and correction of ambiguous and unclear requirement and design specifications that have the potential for evolving into critical faults during later stages of the development process.**

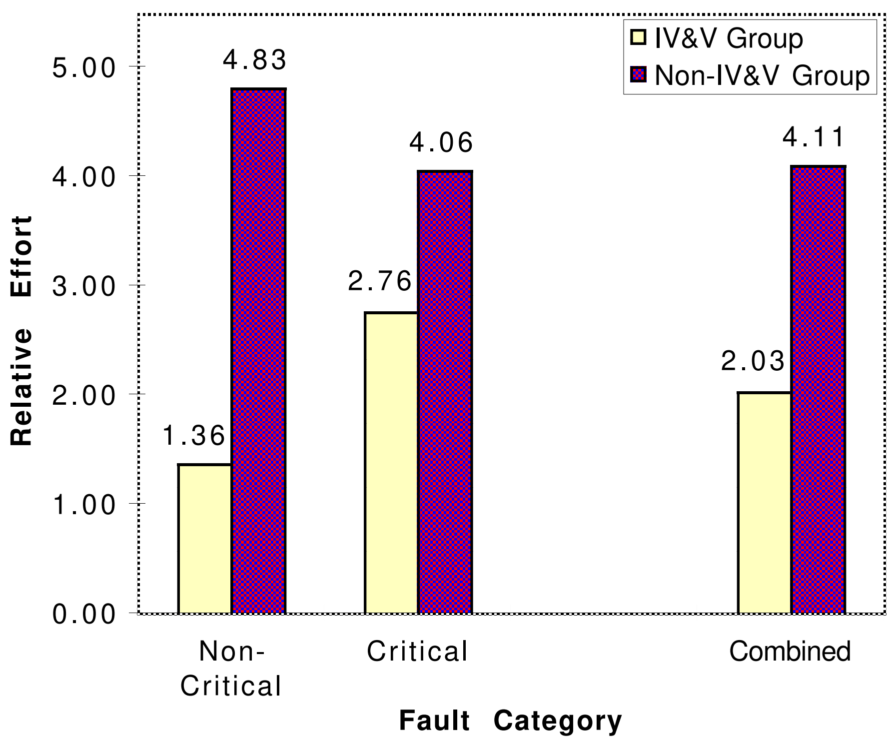
Graph 4 provides a group-by-group comparison of the mean effort required to remove faults (*MER*) detected during the development process. The *MER* being reported is based on observed times, and reports effort measures in minutes.



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 4 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.*

In Graph 4 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 effort values can have in computing a mean. The following graph employs Boehm's estimated effort-to-remove values to compute $MER_{estimated}$.

GRAPH 5
Mean Effort to Remove Faults
-MER Estimated-



Graph 5 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 what phases that 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 by summing the estimated effort values (given in Section 3.4) for each reported fault and then dividing the sum by the total number of faults.

Using estimated effort values instead of observed values, *the MER computations still reflect what we would expect*. 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 (more than double that of Group 1)

Graph 5 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. We now offer a third conclusion:

Conclusion 3: Guided by the results depicted in Graph 4 and Graph 5, SEES' has had a beneficial impact in reducing the effort to fix faults.

5. SUMMARY AND CONCLUSIONS

This paper provides an overview of an experiment which focuses on answering the question:

Do the IV&V activities dictated by SEES (or SEES') add value to a software development process?

That experiment entailed:

- the modification of SEES to reflect the pragmatic constraints of our experiment,
- the formation of two independent development groups: Group 1 utilizing SEES' during the development process, Group 2 following the same software development process but without the benefit of SEES',
- the education of Group 1 relative to the benefits and activities associated with IV&V, and in particular, those espoused by SEES',

- parallel development efforts that spanned Requirements Analysis through Integration and Testing,
- the collection of data ranging from activity logs and fault reports to development artifacts, and finally
- an analysis of the reported data.

In summary, the results outlined in this paper provides one additional data point confirming the beneficial impact of IV&V on the software development process. As argued above those benefits include

- the detection of faults earlier in the software development lifecycle,
- the detection of ambiguous or unclear specifications that can lead to the introduction of critical faults downstream in the development process, and
- the reduction in the effort required to fix individual faults.

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

As a final note, the authors would like to thank Ms. Kelly Hayhurst and Mr. Mike Holloway at NASA, Langley Research Center, for their guidance and insightful comments throughout this experiment.

REFERENCES

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.

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.