A Study and Project-Based Evaluation of The Software Engineering Evaluation System (SEES):

Requirements Analysis Through Acceptance Testing
and
Statistical Analysis of Results

FINAL REPORT

Prepared for K. Hayhurst & M. Holloway

NASA Langley Research Center

by

James Arthur, Co-principal Investigator

William Frakes, Co-Principal Investigator

Sanjay Gupta, Co-principal Investigator

Martha Cannon, Graduate Research Assistant

Markus Groener, Graduate Research Assistant

Zakia Khan, Graduate Research Assistant

Department of Computer Science
Virginia Tech

Funded by NASA Contract NAS1-19610, Task 17

February 28, 1997

APPENDICES

1. INTRODUCTION

The purpose of this document is to describe the design and execution of the quasi-experiment conducted in the Department of Computer Science, Virginia Tech, in accordance with the procedures described in the document "An Experimental Design for Evaluating SEES" prepared for NASA under contract NAS1-19610, Task 17. (Hereafter, that report is called the general design document.) The quasi-experiment serves three important purposes.

- First, the quasi-experiment is the proof of concept of the general experiment design. It shows that the procedures defined in the general experiment design can be implemented.
- Second, provides details for setting up a true experiment: identifying the research hypothesis, designing the investigation, selecting various variables, procedures, and controls, measuring the variables, and evaluating the results.
- Third, any insights revealed during the course of the quasi-experiment can be incorporated in the costly true experiment, thus providing a cost-effective experimental methodology.

This report contains sufficient information so that the quasi-experiment can be replicated at an appropriate level of abstraction. We also document and interpret all the results of the quasi-experiment.

1.1 Experimental Justification

> The prime justification for this quasi-experiment is to provide a proof of concept for the general experimental design.

All the activities of the quasi-experiment are described in sufficient detail in this report so that a true experiment with statistically valid results can be easily set up, even though the sample size in the quasi-experiment is not sufficient to justify calling it a true experiment. The appropriate sample sizes to conduct a true experiment with statistically valid results for various degrees of confidence are described in detail in the general design document.

1.2 Experimental Hypothesis

> Independent verification and validation (IV&V) adds value to the existing software development process.

Setting up an experiment requires that a number of teams (depending on the sample size) develop a software product with no accompanying IV&V procedures, while another set (of the same number) of teams develop the same software with assistance from an IV&V team -- each development team is associated with a separate IV&V team. At the request of NASA, the specific IV&V method selected for the purpose of the quasi-experiment is the Software Engineering Evaluation System (SEES) developed by U.S. Army Missile Command's Software Engineering Directorate (SED). As defined by SED, SEES is a verification and validation (V&V) method. However, since SEES does not inhibit IV&V, the IV&V procedures in the quasi-experiment are guided by the SEES philosophy. Hence, the null hypothesis for this quasi-experiment states that

> no difference exists between software developed with SEES and the software developed without SEES.

1.3 Development Application

For the experiment with SEES, the software application chosen reflects a Multidisciplinary Design Optimization (MDO) approach to aircraft sizing. This application is based on the work of Mason and Malone at Virginia Tech (Malone, 1991). The software computes the design of an aircraft through iterative calculations based on several parameters such as wing area, cruise altitude, and various weights. Since the MDO application is in the aeronautical engineering domain, the results should be generalizable to aeronautical engineering domains within NASA.

1.4 Experimental Rigor

Every possible effort is made to maintain the highest possible level of experimental rigor. However, limited resources such as labor, and the fact that the experiment must be conducted in a university environment where all the team members are computer science graduate students, prevent the quasi-experiment from being a true experiment. The following issues must be addressed differently in a true experiment.

- **Sample size:** In the context of the given research hypothesis, the sample size refers to the number of different teams developing the software without IV&V, or equivalently, the number of teams developing the software with IV&V. The general design document defines various sample sizes to achieve different degrees of statistical confidence in the results.

- **Random assignment**: Once an adequate sample size has been selected, it is imperative that the various development personnel be randomly assigned to the development teams. Furthermore, all IV&V personnel should also be randomly assigned to IV&V team(s). As mentioned in the general design document, random assignment converts any biases into noise, giving a higher degree of confidence in the results.

- **No interaction between different groups**: There should not be any form of interaction between the members belonging to different teams, except, of course, the minimal appropriate interaction between the members of a development team and its associated IV&V team.

- **No knowledge of the other teams to minimize bias due to competition**: Each group developing the software should not be aware that there are other groups developing the same software product. The knowledge of the existence of other teams can make the team members competitive, thereby biasing the results.

- **No IV&V knowledge for the development group**: The members of the teams developing the software with no IV&V should not be versed in SEES, or any other IV&V procedures in general. Such knowledge can be consciously or subconsciously used by the development team and thereby could bias the results of the experiment.

1.5 Organizational Layout

Background: Section 2 provides the working definitions of the technical terms used in this report. In particular, the terms “IV&V” and “V&V”, as well as related terms required to define these concepts are presented. Formal and precise definitions of all the terms used in the report are given in the glossary (Appendix A). We also provide a brief overview of SEES as defined by SED.

Experimental Adaptation of SEES: Section 3 introduces the term SEES' (pronounced "SEES prime") to denote the modified version of SEES. These modifications described are made to adapt the labor-intensive SEES procedures to the available limited resources, and also to use SEES as an IV&V method as required by the experimental hypothesis.

Experimental Methodology: Section 4 describes in detail the methods and procedures used in the experiment, including experimental design, control, procedures, and assignments of the subjects to teams. The description is detailed enough so that the quasi-experiment can be replicated at an appropriate level of abstraction. In particular, the activities performed prior to the experiment, the experimental environment, the experimental design, experimental controls, and the experimental activities are all discussed.

Results: Section 5 summarizes the data and observations produced by the experiment, including all computations and analysis. The results are evaluated and related to the experimental hypothesis.

Summary and Conclusion: Section 6 the results are interpreted and presented in a generalizable context. In particular, we discuss how the results relate to development situations other than the experimental one.

Future Work: In Section 7 outlines several extensions to the research reported in Sections 5 and 6. More specifically, we present additional questions for exploration and propose the examination of classical statistical techniques to general experimental methods. In both cases, the data already acquired can serve as a testbed for analysis.

Appendices:

- Appendix A contains the detailed and precise definitions of the technical terms used in this report.
- Appendix B contains a detailed and annotated bibliography for future studies.
- Appendix C contains the background information for members of all the teams participating in the described experiment.
- Appendix D contains references to all the forms, and copies of the modified forms used in the SEES' experiment.
- Appendix E contains the Software Requirements Specifications document (SRS) given to both the development teams and the IV&V team at the start of the experiment.

Folder 1: The requirements, design, code, and test documents produced by each group are provided in Folder 1. The folder also contains a data summarization of all forms turned in by each the group participating in the experiment.

2. BACKGROUND

The main objective of this section is to provide working definitions for the technical terms used throughout the report. In particular, the terms Independent Verification and Validation (IV&V) and Verification and Validation (V&V) are defined. Also included is a short overview of SEES as defined by the Software Engineering Directorate (SED) of the U.S. Army Missile Command [SED,1993] . Formal and precise definitions of these concepts and related terms are included in the glossary (Appendix A).

2.1 Definitions of V&V and IV&V

IV&V and V&V are closely linked to the Software Development Lifecycle (SDLC); a short description of the SDLC and an example of a general SDLC, the Waterfall Model, follows.

2.1.1 Software Development Lifecycle

A system life cycle describes the evolutionary process that a system undergoes from its conception through its useful operational life. In the waterfall model [Royce, 1970], the life cycle is divided into a series of phases that are ordered sequentially: concept and exploration, requirements, design, implementation, testing, and maintenance. Of these, the phases from requirements through testing are considered the Software Development Life Cycle, and are the principal phases with which IV&V and V&V is concerned.

Each of the major life cycle products, or documents, is developed in sequence, beginning with the most abstract and ending with the concrete system. The waterfall analogy derives from the fact that, as each phase completes, its products or outputs flow as inputs "down to" the next development phase; each phase is viewed as a discrete entity, with no concurrent phase activity. At a minimum, the SDLC of the waterfall model should produce the following life cycle products for IV&V or V&V:

1. An overall software development plan,
2. Requirements documents such as a Software Requirements Specification (SRS),
3. Design specifications, such as an Interface Design Document (IDD) and a Software Design Document (SDD),
4. Documented code, and
5. A complete software test plan, including a description of test cases. Such documents include the high-level Software Test Plan (STP) and the more detailed test descriptions contained in the Software Test Description (STD).

2.1.2 Verification and Validation

Verification and Validation is a process whereby a system is comprehensively analyzed and tested during all phases of the development life cycle by a group either identical or 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. V&V is generally used for the development of critical software systems, where safety and/or financial risks are high. V&V can be applied to hardware and software systems; only software V&V is addressed here.

In the context of V&V and IV&V, the terms *verification* and *validation* are two distinct terms. 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 concerns both the intrinsic correctness of the products of a software development phase, as well as the relation of each phase's products to the preceding and succeeding development phases.

Validation essentially refers to software (and system) testing. During validation, the system is executed and test results are compared to performance requirements, namely, the system and software requirements specifications. The correctness of the test results therefore validates the specifications originally verified in the requirements phase.

2.1.3 Independent Verification & Validation

While V&V is generally performed by a group closely linked to the development team, IV&V is conducted by a group 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]. There is disagreement in the literature over whether IV&V differs from V&V in more than the independence of its practitioners. In general, the difference varies depending on the project, and the perspective of the practitioners. See, for example, [Fujii, 1994].

2.1.4 IV&V Activities by SDLC Phase

- *Requirements phase IV&V* uses mostly static analysis techniques such as review and inspection techniques, process and flow analysis, interface analysis, and traceability of system requirements to software requirements. Formal algorithmic analysis can be started during the requirements phase IV&V, however no dynamic analysis (see glossary) is used during this phase.
- *Design phase IV&V* activities are similar in type to those used in the requirements phase. Review and inspection, process and flow analysis, interface analysis, and traceability analysis are used in the design phase. Software requirements are traced to design specifications. Data structures analysis is a static technique that must start in the design phase. Formal algorithmic analysis is conducted in depth during the design phase; as with requirements, no dynamic analysis is used during this phase.
- *Implementation phase IV&V* activities can include both static and formal analysis techniques. Review and inspection, process and flow analysis, and data structure analysis may all be used again. In addition, the following verification techniques can be used: assumptions analysis; hidden path analysis; real time interruptibility analysis; timing analysis; computational accuracy analysis; constraints analysis; and programming syntax analysis. No dynamic analysis is used during the implementation phase IV&V.
- *The Integration & Testing phase* of IV&V contains some static review of test documentation, however, the emphasis is on dynamic analysis, namely, testing. Some of the type of tests performed during this phase include: performance testing; abnormal cases testing; timing dependencies testing; transaction analysis testing; error recovery testing; and system interface (integration) testing.

2.1.5 Verification Quality Criteria

Operational definitions of quality criteria used to verify requirements, design, and code documents are given below.

- ***Completeness:*** Each requirement should be complete in the sense that the requirement describes in detail all functions, features, capabilities, constraints, and other properties of the target system (the MDO system in this case), [Frakes, 1991].
- ***Traceability:*** A requirement that is traceable in the design phase, means that the requirement has been fulfilled, or realized in the design specifications. Each requirement

should be able to be traced to one or more design specifications, one or more code implementations, and to one or more tests verifying that the requirement has been met and is functional. Numbering individual requirements and referring to them by number in design documents, source code, and the test plan helps to determine whether a requirement is traceable through the entire development lifecycle [Frakes, 1991].

- ***Testability:*** Each requirement must be stated in such a way that the requirement is testable. If a requirement cannot be tested, there is no way to verify and validate it. For example, the statement “the system must respond to queries in a reasonable amount of time” is not testable. To be testable, a requirement must describe a precise amount of time, such as, “the system must respond to any query within 10 seconds” [Frakes , 1991].
- ***Consistency:*** This quality criterion refers to the fact that requirements must not conflict with one another, and that terms are used consistently throughout the development documentation. Design specifications must also be consistent.
- ***Unambiguous:*** A requirement, a design specification, and code documentation should all be unambiguous. That is, there must be one, and only one interpretation of the meaning of a specification. If requirements are ambiguous, the design will be ambiguous, and the implementation may or may not match what the customer intended.
- ***Correctness:*** The output of a particular phase is correct if it 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. For example, a requirements document is correct if it accurately describes all needed functions and properties of a product and easily leads to development of a design document. A design specification is correct if it accurately describes a design which satisfies the requirements and can easily be implemented as code.
- ***Adherence to Standards:*** Adherence to Standards is really correctness of requirements, design, and code artifacts as they relate to outside standards. Standards can include language standards, i.e., the code must adhere to ANSI C standards. Standards can also refer to standards imposed by the customer. For example, the Department of Defense has very detailed software development standards, with requirements for particular documents in a particular format.

2.1.6 Validation Quality Criteria

Operational definitions of quality criteria used to validate the integration and testing phase are given below.

- ***Testing:*** Testing is the exercising or evaluation of a system or system component by manual or automated means to demonstrate that it satisfies specified requirements, or to identify differences between expected and actual results [ANSI/IEEE, 1983].
- ***Test Coverage:*** Test coverage refers to how many of the myriad possible execution paths are “covered”, or tested, by the total tests run on the software during. In general, it is impossible to test every single possible execution path, so tests are designed to cover the most critical execution paths. Automated tools are available to determine test coverage. Also, automated metrics which test frequency of execution, cyclomatic complexity, branching complexity, etc., can be used to determine the most critical execution paths. Further information on critical paths is obtained from the developers’ experience during integration and testing.
- ***Test Accuracy:*** Test accuracy refers to whether or not a test designed to a particular requirement accurately tests that requirement. For example, a test may be designed by developers to examine query response time, but because of a logic flaw in the test, the test

may actually reflect a query response time plus some unknown time factor. This test would therefore, not be considered accurate.

- ***Requirements Validation:*** As the final step in a full phase IV&V effort, requirements are validated through testing. In other word, each test is traced back to a requirement which the test validates.

2.2 SEES OVERVIEW

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 (SED). Comprehensive is used to mean that SEES is a *Full Cycle In-Phase* approach to V&V [see Glossary, Appendix A]. SEES was originally developed to evaluate critical Battlefield Automation Systems (BAS). SEES is based on a five step implementation, performed by a trained team from SED and in concert with the development organization [U.S. Army Missile Command SED, 1993a].

- First, a planned software development project is assessed for its V&V needs.
- Second, the various SEES V&V procedures are tailored to fit schedule and budget constraints, while ensuring that critical software issues are addressed.
- Third, the tailored V&V program is executed on two fronts: selected Technical Assessment Procedures (TAPS), which concentrate on technical V&V evaluation tasks, (i.e., verifying and validating lifecycle objects), and programmatic evaluation tasks (i.e., development process issues and configuration management issues).
- Fourth, the results of the V&V are measured and analyzed to identify software risks and deficiencies.
- Fifth, the results are used to improve the quality and reliability of the system, and to ensure that the system meets the users' stated requirements. SEES also provides a feedback mechanism whereby V&V results for each evaluated system are reported to a data depository known as the "SEES Metrics Lab". Data from the Metrics Lab is then used to focus and plan future V&V efforts for similar projects.

2.2.1 SEES Technical Assessment Procedures (TAPs)

The SEES TAPs form the nucleus of the SEES methodology. There are ten separate TAPs, each of which describes a specific methodology to accomplish a particular V&V technical evaluation task [U.S. Army Missile Command SED, 1993b]. A brief description of each TAP used in the SEES' quasi-experiment follows below. (The descriptions reflect a SEES characterization, not SEES'.)

- Requirements Review and Assessment TAP (RRATAP: 1-1)

  The RRATAP is a requirements verification methodology. A list of software requirements (the SRL) is developed based on the SRS. Each requirement is then subjected to the Requirements Analysis Diagram (RAD) methodology and a checklist review. Under the RAD methodology, input conditions, processing activities, and processing results are documented for each requirement on the SRL. New RADs are generated for derived requirements. In conjunction with the RADs, checklists are used to evaluate the completeness, traceability, nonambiguity and other verification quality factors of software requirements. Any requirement defects are noted on the RADs, and a problem report (the Document Trouble Report, or DTR) is generated.

- Requirements Trace/Completeness Matrix TAP (RTMTAP: 2-1)

  The RTMTAP is a verification procedure to determine that system level requirements have been completely allocated to software level requirements, and that all software requirements can be correctly traced back to a system requirement. In the RTMTAP a matrix between system and software level requirements is built to accomplish this task.

- Design Review and Assessment TAP (DRATAP: 3-1 & 3-2)

  The DRATAP is a formal inspection methodology to verify design documents. DRATAP: 3-1 describes the verification of preliminary and detailed design documents, while DRATAP: 3-2 focuses on system level design documents. The inspection methodology used in DRATAP is based on the Fagan Inspection Methodology [Fagan, 1976].

- Code Review and Assessment TAP (CRATAP: 4-1)

  The CRATAP is a dynamic analysis tool to verify Ada code. A manual analysis ensures that the code is consistent with design documentation, while an automated tool (Adamat) analyzes various software metrics to ensure that the code adheres to approved programming language standards.

3. EXPERIMENTAL ADAPTATION OF SEES

For the purposes of this experiment, modifications have been made to SEES in order to accommodate limited resources and the use of SEES as an IV&V method. These changes are discussed below.

3.1 Adaptation of SEES to SEES'

Two major changes have been made to SEES as originally defined by the SED. First, we have adapted SEES, a V&V method, to be used as an IV&V method in the experiment. Second, resource limitations mean that we have concentrated mainly on the Technical Assessment Procedures (TAPs), which are the heart of SEES. We have also had to make some modifications to the individual TAPs, and time limitations meant that not all TAPs could be completed during this experiment. It is important to note that while we are not following the SEES methodology verbatim, we are following the <u>*intent*</u> of the SEES methodology. Henceforth, in order to distinguish between SEES as promulgated by the SED and the modified methodology adapted for this experiment, we will refer to the adapted methodology as SEES'.

3.2 SEES' as an IV&V Process

Per NASA's request, we are applying SEES as an IV&V process, rather than as a V&V process. In general, there is nothing to preclude using a V&V methodology as an IV&V methodology; in SEES' the analysts performing the V&V are not part of the development team. Moreover, certain IV&V procedures have been hidden from the development team. SEES is particularly amenable to such a change, since SEES requires the SED to send in a separate team to perform the V&V with assistance from the developers. Thus, while SEES emphasizes V&V, it does not inhibit IV&V.

One example of how SEES has been adapted from a V&V to an IV&V methodology concerns the development of requirements. SEES contains detailed instructions on how to develop software requirements from system requirements and perform V&V on the requirements while developing them. In the SEES' experiment, software requirements were developed independently of the experiment, and given to development teams as a baseline. Since SEES' tests IV&V, development of software requirements was not included in the SEES' experiment. Nonetheless, IV&V was conducted on the completed and baselined requirements in accordance with SEES methodology.

3.3 Adaptations Due to Limited Resources

Adaptations have had to be made both because of limited human resources, and limited time resources. The SEES' experiment was structured around a university graduate course on IV&V which ran for two summer terms, or approximately twelve weeks. Seven students registered for the class, while two additional subjects were employed to participate in the experiment. With nine subjects participating, only two development teams, one experimental and one control, could be formed, along with one IV&V evaluation team (See Section 4.4.1 for detailed assignments). The average size of each team was three students. Obviously, the SEES methodology had to be scaled back to accommodate these small numbers.

The twelve week time restriction meant that the full software development lifecycle could not be completed, and that some of the TAPs that were completed had to be scaled back. Finally, the fact that the subjects were graduate students, and not professional software developers may have further contributed to the time limitation.

3.3.1 General Experimental Adaptations

As noted in the Introduction (Section 1), some of the adaptations have meant that the SEES' experiment is actually a quasi-experiment. The primary reason that this experiment is not a true experiment, is that, due to limited human resources, the sample size is too to impart statistical significance. To achieve reasonable results within the constraints, we have used matching to achieve skill level parity in the assignment of subjects to teams (See Section 4.4.1). Another general adaptation was made both because of the limited time and skill of the subjects. Instead of using a full-scale software product with multiple Computer Software Configuration Items (CSCIs), and system level requirements, a scaled back version of the MDO application was chosen (See Section 4.2.4). The MDO application as developed under the SEES' experiment comprises one CSCI, has no system level requirements, and specifies single-parameter optimization rather than multi-parameter optimization.

3.3.2 Adaptations of Technical Assessment Procedures

With only twelve weeks for the SEES' experiment, software development could be completed only to the Code and Unit Testing phase. Thus IV&V activities used procedures from the following four TAPs:

- Requirements Review and Assessment TAP (RRATAP: 1-1)
- Requirements Trace/Completeness Matrix TAP (RTMTAP: 2-1)
- Design Review and Assessment TAP (DRATAP: 3-1)
- Code Review and Assessment TAP (CRATAP: 4-1)

The following TAPs were not applied, since the Testing and Validation phase could not be completed during this particular phase of the experiment:

- Test Plan Assessment TAP (TPATAP: 5-1)
- Test Description Assessment TAP (TDATAP: 6-1)
- Test Witness and Assessment TAP (TWATAP: 7-1)
- Functional Configuration Audit TAP (FCATAP: 9-1)
- Physical Configuration Audit TAP (PCATAP: 10-1)

Of the TAPs that were used, certain tasks not deemed applicable to the experiment were never executed. For example, each TAP contains tasks to create a Procedure Assessment Report (PAR) to be submitted to the Department of Defense Project Management Office (PMO). Each TAP also contains a task to report all data utilized or generated by the TAP to the SEES Metrics Laboratory where the data is to be used for quality improvement of future projects. Since such activities, were not applicable to the SEES' experiment, these activities were systematically dropped. Instead, in the SEES' experiment, IV&V information about software defects was passed to the development team through carefully controlled classroom presentations and the use of DTRs.

In another adaptation of SEES TAPs to SEES', labor intensive SEES procedures were not followed literally. For example, SEES clearly intends that every single specification be literally checked against every checklist item; we deemed this task impossible considering our time and personnel limitations. Instead, TAP checklists were distributed to IV&V team members, who then read them, and used the checklist as references when evaluating requirements specifications, design specifications and code. Thus we preserved the intent and philosophy of the SEES checklists.

Finally, of the four TAPs that were used in SEES', certain tasks were not executed, or were executed slightly differently because of resource constraints or inapplicability. For example, in the RRATAP:1-1, requirements were assessed and verified according to the criteria laid out in the TAP, but task 2, the Requirements Analysis Diagram (RAD) methodology was not performed. In the RTMTAP:2-1, a Software Requirements List (SRL) was generated in accordance with task 1, but a formal Requirements Trace Matrix (RTM) that traces software requirements back to system requirements was not developed. Since the MDO had no system requirements, this task was deemed not applicable to SEES'. Instead, traceability to software requirements was considered as a verification criteria during IV&V on the design specifications.

Except for the PAR and the SEES Metrics Report discussed above, the formal design inspections of the DRATAP:3-1 were executed precisely as directed in the TAP. Finally, the Code Review and Assessment TAP performed in the SEES' experiment differs from the SEES CRATAP:4-1 only in the fact that the ANSI C programming language was used for SEES' while the SEES is designed for Ada. This meant that an ANSI C automated code metrics analyzers had to be used instead of Adamat, which measures Ada software metrics.

3.3.3 Modification of Forms

The SEES data collection forms were also appropriately modified for SEES'. As an example, the changes made to the SEES Software Requirements List (SRL) form for the SEES' experiment are discussed below.

The SEES SRL contains the following fields (see Appendix E for a copy):

1. *Program ID*
2. *Date*
3. *Reference #*
4. *Requirement ID #*
5. *Description*
6. *S/D*
7. *DTR #*
8. *RAD #*

The SEES' SRL contains the same fields, except for field numbers 3, 6, and 8, which are not included in the SEES' SRL (See Appendix D for a copy). The SEES' SRL contains an additional field, the *Requirement Type* field. In SEES, the *Reference #* is mainly used for constructing the RTM, the traceability matrix between system level and software level requirements. Since SEES' did not execute an RTM, there was no need for a *Reference #* field on the SEES' SRL. In SEES, the *S/D* field refers to whether a requirement is stated or derived. In SEES' this information was recorded in the *Requirement Type* field. As for the *RAD #* field in the SEES SRL, this field was unnecessary in SEES', as the RAD methodology was not executed. The only other difference between the two SRLs is concerned with how the requirement is categorized. In SEES, each requirement is classified as follows, with the abbreviation noted in the *Description* field of the SRL:

- (PERF) Performance
- (IN) Input
- (OUT) Output
- (HMI) Human-Machine Interface
- (PROC) Processing
- (DSGN) Design
- (DOC) Documentation

In SEES', a different classification scheme more appropriate for the MDO application was developed. The SEES classification scheme accounts for much larger development applications, with system and hardware requirements. In SEES' the *Requirement Type* field was added to make it easy for IV&V analysts to classify a requirement. The following classification scheme was developed for the SEES' SRL:

- Functional
- Performance
- Interface
- Design/Implementation Constraint
- Inverse
- Process
- Derived

In fact, the two classification schemes are very similar. Both schemes have Performance, Design, and Process categories; the Input and Output categories of SEES correspond to the Interface category of SEES'; and the SEES *stated/derived* field corresponds to the Derived category of SEES'.

To review modifications made to other SEES forms for the SEES' experiment, refer to Appendix D for the copies of the modified forms. Also, two additional forms, the "Activity Log", and the "Fault Report Form", have been developed solely for data collection in the SEES' experiment.

4. EXPERIMENTAL METHODOLOGY

An experimental methodology specifies how an experiment is to be run, including the research design. A complete methodology permits the experiment to be replicated. The complete experimental methodology for the SEES' experiment is presented below.

4.1 Experimental Preparation

As part of the preparation for the SEES' experiment, an independent study was conducted prior to the beginning of the experiment. In conjunction with the SEES' experiment, Professor Arthur conducted a class on IV&V.

4.1.1 Independent Study

During the 1995 Spring Semester Dr. Arthur offered a study on IV&V techniques through an Independent Study program. Several students participated in the Independent Study, one of whom later served as the Group 1 IV&V team leader. The study consisted of performing IV&V on a simple software metrics program called *ccount*. *Ccount* is an example software metrics program with supporting documentation used to illustrate software engineering principles in [Frakes et al, 1991]. *Ccount* counts the number of non-commentary source code lines (NCSL) and commentary source code lines (CSL) in a C program. NCSL, CSL and the ratio of NCSL to CSL are computed for each C function, in an effort to determine one measure of program complexity.

IV&V performed on *ccount* followed the SEES methodology as much as possible. While no data was collected, this study served to help structure Professor Arthur's IV&V class, and served as a vehicle for better understanding SEES.

4.1.2 The IV&V Course and the SEES' Experiment

Professor Arthur prepared a course on IV&V for the two summer terms of 1995. The main text for the course was Robert Lewis' *Independent Verification and Validation* [Lewis, 1993]. The seven students who took the course were required to participate in the SEES' IV&V experiment, with the students in the class comprising Group 1 of the experiment. Three students comprised the Group 1 development team, while the remaining four students in Group 1 were members of the SEES' IV&V team. The Group 1 IV&V team performed IV&V on the Group 1 development team's products. The control group, referred to as Group 2, was a development team consisting of two graduate students who did not take the IV&V class, and who were paid wages for participation in the SEES' experiment. No IV&V was performed on the development effort of the control group.

During the IV&V class, IV&V concepts were presented from two perspectives. First, a generic perspective was taught, with material largely drawn from Lewis [Lewis, 1993]. Second, for each development phase, elements from the SEES methodology were presented for comparison with more general IV&V methods. To control potential biases, the Group 1 development team was directed not to use knowledge of any IV&V techniques gained during the course in their development efforts. Also all SEES' IV&V forms were withheld from the development team until such time as formal IV&V feedback was given.

4.2 Experimental Environment

The experimental environment for a software development project includes the following: development methodology, programming language, hardware platform and development support tools, and programming application. Each item is addressed below.

4.2.1 Development Method

The waterfall software development method was used by both the IV&V development team (Group 1) in the IV&V class, and by the non-IV&V development team (Group 2) outside of the class. The waterfall method is described in Section 2.1.1.

4.2.2 Programming Language

At present, there is no evidence to indicate whether or not the programming language chosen is a significant factor in the evaluation of IV&V methods. Without such evidence, a language should be chosen that is generalizable to the target population. SEES' was originally designed for Ada, while FORTRAN has traditionally been the language of choice for MDO applications. However, the student subjects are most familiar with C, thus ANSI C was chosen to implement the MDO application. Some of the more mathematically complex modules of the MDO application were provided in completed form to the developers.

4.2.3 Hardware Platform

Both teams used the same machine for development, a DEC Alpha, running the OSF-1 Unix operating system. Developers could use software engineering tools available in the Unix environment, such as profilers, coverage analyzers, and configuration management tools. The tools used were the C compiler, the vi editor tool, and the symbolic debugger.

4.2.4 MDO Application

For this experiment, the Multidisciplinary Design Optimization (MDO) application was chosen. Because the MDO application is in the aeronautical engineering domain, the results of the SEES' experiment should be generalizable to aeronautical engineering domains within NASA.

The MDO application chosen for the experiment is based on work by Mason and Malone at Virginia Tech (Malone, 1991). As originally conceived, this application computes the design of an aircraft through calculations that factor in parameters such as wing area, cruise altitude, and different weights. In performing the design calculations, an initial set of input parameters is given, and a baseline design determined. Further iterative computations are performed to refine the baseline design by increasing the precision of the initial and computed parameters. There are four major computational areas in this application: initialization, design calculations, derivative calculations, and optimization calculations. For purposes of the SEES' experiment, the MDO application was scaled back to use a single parameter, weight, as the optimization variable. For more detailed information on the MDO application see Appendix A of the *Experimental Design for Evaluating SEES* [Frakes et al, 1995] .

4.3 Experimental Design

The experimental, or research, design is the part of the experimental methodology that ensures that the criteria for an experiment are met. Through the research design, the researcher attempts to control all biases, and protect the validity of the experiment. All experiments must be concerned with two fundamental issues, internal validity and external validity. Internal validity refers to the certainty that the observed changes in the dependent variables are due to the independent variables. External validity refers to whether the observed results can be validly extrapolated to a target population, in this case assumed to be NASA software development groups.

Even though the SEES' experiment is considered a quasi-experiment because of the lack of appropriate sample size, extensive efforts have been made to avoid bias and to protect internal validity. These efforts are discussed below in Section 4.3.3.

Selecting an experimental design involves selecting the dependent variables (DV's), the independent variables (IV's), and a structure that relates them. The general model can be described using the functional equation:

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

This equation means that the values of the dependent variables are determined to some degree by some function of the independent variables. The independent and dependent variables are described in detail below.

4.3.1 Advance Hypothesis

The general question this experiment is designed to answer is:

> "Does the SEES' IV&V method add value to the existing software development process?"

From this question, a null hypothesis was formulated. The null hypothesis for this experiment states that no difference exists between software developed with SEES' and software developed without SEES'. A more formal statement of the null hypothesis for the SEES' experiment is:

> "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'"

4.3.2 Unit of Analysis

The unit of analysis refers to the object being studied, whether it is an individual or a group. For the SEES' experiment, the unit of analysis is a software development team developing the MDO application. In the case of this experiment, there are two units, the development team in Group 1, to which SEES' was applied, and the Group 2 development team, to which SEES' was not applied.

4.3.3 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***'. As noted above, each treatment level has a single unit of analysis.

4.3.4 Dependent Variables

A dependent variable, or response variable, is a variable measured to indicate the effect of the independent variables. The primary dependent variables which are chosen to measure the effect of SEES' in the experiment are:

(1) Operational Correctness

Operational correctness is a measurement of the time that the final software product operates correctly. Operational correctness can be measured only after the MDO application has been completely developed and has undergone IV&V (depending on the treatment group). There are two ways of measuring operational correctness depending on whether the application developed is a continuous execution system or a system that executes in discrete mode.

For a discrete system such as the MDO application, frequency of failure can be measured to determine the degree of operational correctness. Sets of input data are developed, including input data sets that mimic expected operational values, as well as data sets that are at the boundaries of expected data value ranges. These data sets are then uniformly executed on both the Group 1 system and the Group 2 system, and the number of operational failures in each are noted. The failure frequency of a system is inversely related to the system's degree of operational correctness.

Operational Correctness is not measured as part of this experiment, since time limitations meant that the development groups could not complete the Integration and Testing phase. Without testing data, operational correctness could not be measured before the end of the experiment. Since the code has been completed, however, tests can be developed and executed at a later date to determine the level of operational correctness of the code produced in this experiment. We conjecture that the software developed by Group 1 (using SEES') will be found to have an operational correctness level greater than or equal to the operational correctness of the Group 2 software.

(2) Mean Effort to Remove Faults (MER)

MER , the variable of primary interest to NASA for the evaluation of SEES (or SEES' in this case) reflects the improvement of one IV&V method in removing faults over another (or no) IV&V method, i.e., the "value added" by a specified IV&V method. This summary variable is complex and requires component variables to calculate. Component variables are described below in Section 4.3.5. Two ways of calculating *MER*, one based on estimation using a table, and one based on direct observations of values have been used.

$$MER = \frac{\sum_{i=1}^{n} F_i(I,R)}{n}$$

The above equation for *Mean Effort to Remove Faults* (*MER*) gives the average effort to detect faults $\boldsymbol{F}$ inserted in life cycle phase $\boldsymbol{I}$, and detected and removed in life cycle phase $\boldsymbol{R}$. The variable $\boldsymbol{n}$ is the total number of faults found over all (completed) phases of the experiment. Values for a fault $\boldsymbol{F_i}$ can be either estimated or observed. Observed data was collected by observing in which phase a fault is found and removed (the variable $\boldsymbol{R}$), by inferring in which phase the fault originally occurred (the variable $\boldsymbol{I}$), and by measuring the time effort taken to correct the fault (the value of the function $\boldsymbol{F_i(I,R)}$).

The estimation method for *MER* uses an augmented table derived from Boehm [Boehm, 1981 (fig 4-2)]. This table contains the relative cost-to-fix a fault introduced in phase *I* , and discovered in phase *R*. See Section 3.1.4, Table-1, page 14 of the *Experimental Design for Evaluating SEES* [Frakes et al, 1995] for a copy of the augmented table and for a more detailed explanation of its derivation.

4.3.5 Component Variables Used to Calculate MER

(1) Number of faults found, categorized by the severity of the fault

This variable is measured for each completed development phase and for the total number of life cycle phases completed (requirements through code). This variable requires a human subject to measure the number of faults and classify each fault according to a severity scale. The human subject also indicates in which phase the fault is detected and corrected, as well as the probable phase in which the fault is introduced.

In the case of the Group 1 development team (the "use SEES'" treatment group), the number of faults reported equals the total number detected by the development team and the Group 1 IV&V team. In the case of the Group 2 development team (the "not use SEES'" treatment group), the number of faults equals only those reported by the development team.

Two data collection forms are used to report faults. The two development teams used a Fault Reporting Form (FRF) which was created by the Group 1 development team in accordance with requirements stated by Professor Arthur. The Group 1 IV&V team, on the other hand, used a modified version of the SEES Documentation Trouble Report (DTR) to record all faults found during IV&V evaluation (the modified version is called a "Defect Trouble Report"). The FRFs are used solely by the development teams. The DTRs are filled out by the Group 1 IV&V team, and are passed on to the Group 1 development team for action, once IV&V on a phase is completed. The FRF and the DTR are similar except for the following items:

- The DTR uses a severity scale of *Critical, Non-Critical,* and *Nominal,* while the FRF expands severity categories to *Critical, Severe, Normal, and Inconvenience. Critical* and *Severe* on the FRF correspond to *Critical* on the DTR; *Normal* on the FRF corresponds to *Non-Critical* on the DTR; and *Inconvenience* corresponds to *Nominal* on the DTR.

- The FRF requires the subject to report *Time taken to fix fault*. The DTR does not report this information; however, for faults reported via the DTR this information can be obtained from the Activity Log Form, a non-SEES form developed for this experiment which records activity, time taken, and if applicable, DTR number.

See Appendix D for data collection forms used in the SEES' experiment.

(2) Time effort expended for given tasks

The time effort variable requires a human subject to measure the number of person-hours worked on a particular development activity. For the SEES' treatment group (Group 1), all time effort measurements include time spent developing reports or performing tasks as required by SEES'. Time spent reporting measurements of dependent variables for this experiment is not included. Reporting forms called "Activity Logs" are used by subjects in both the Group 1 and Group 2 development teams to report time effort. For each member of a development team, start and end times are recorded for each development activity during each development phase. For faults discovered by the IV&V evaluation team, the DTR identification number was reported as well as the development phase during which the fault is fixed. For faults that do not correspond to a DTR, effort to fix a fault is measured using the Activity Log and the FRF.

(3) Difference between fault insertion and detection

This component variable of *MER* is a measure of how long it takes to find a fault. The difference between the two phases, (when the fault occurs, and when it is found and removed), provides a measure of IV&V effectiveness, assuming that the longer it takes to detect a fault the more ineffective an IV&V method probably is. Whenever a fault is discovered, a human subject must determine in which life cycle phase the fault was most likely introduced. This can usually be determined from the type of fault. The subject reports the "fault insertion phase" as well as the "fault detection phase" on the FRF. For faults reported by the IV&V team, the fault detection phase is known. If a fault reported on a DTR is thought to be inserted in an earlier development phase, a comment to this effect is made in the *Comments* field of the DTR.

4.4 Experimental Controls and Randomization

Experimental control of potential bias, or error, determines how rigorous an experiment is. In turn, experimental rigor determines the strength of an experiment's results. Biases introduce errors that are always in the same direction. Bias can be introduced into an experiment in many ways; for example, improper control of communication between subjects, or incorrect use of statistical inference can introduce bias into an experiment and weaken the validity of results. The most important means for controlling bias and ensuring experimental rigor is to use random assignment.

4.4.1 Assignment of Subjects to Teams

Ideally, random assignment should have been used first to assign subjects to development teams or IV&V teams, and then to assign development teams to treatment groups. Random assignment means that each subject has an equal chance of being assigned to an IV&V team, a development team whose products will not undergo any IV&V, or a development team whose products will undergo IV&V using SEES'. Random assignment could not be used, however, because of the extremely small sample size (see below Section 4.4.2). A total of nine human subjects were available for assignment to the Group 1 development team or IV&V team and the Group 2 development team. With so few subjects, all with varying abilities, there existed a high probability that, using random assignment, subjects with equivalent software engineering abilities would not be equally distributed throughout the teams. For example, using random assignment, the Group 1 development team could easily have ended up with all the experienced programmers, while Group 2 ended up with only inexperienced programmers. With a large enough sample size, the probability of this type of disparity between treatment groups is extremely low. Moreover, since the sample size for this experiment was so small, any statistical results obtained from this experiment will have low significance. Therefore, it was elected to use matching assignment rather than random assignment.

Since randomness could not be relied on to create un-biased subject combinations, matching assignment was used to assign subjects to teams so that subject-combinations on teams were equivalent in skill levels. Using subject experience histories, subjects were assigned to teams by Dr. Arthur so that, as nearly as possible, no team would have a skill advantage over another team.

4.4.2 Sample Size and Team Size

Only nine students were available for assignment to a development team or an IV&V team. Seven of the students were participants in Dr. Arthur's IV&V class. Of these seven, four were assigned to the Group 1 IV&V team, and 3 were assigned to the Group 1 development team. The two remaining students comprised the Group 2 development team, and participated in the experiment in return for wages. As noted above, (Section 4.3.2) for the SEES' experiment, the unit of analysis is a software development team. In the case of this experiment, then, there are two units, the development team in Group 1, to which SEES' was applied, and the Group 2 development team, to which SEES' was not applied. Therefore the sample size for this experiment is actually two -- for the two development groups. The IV&V team is part of the treatment conditions, and not part of the sample.

The sample size (N) required for a given experiment is a function of the alpha level (α) required, the power (1- b), and the effect size (g) required. For further information on the relationship of the alpha level, power, and effect size to sample size, see Section 3.2.2 of the *Experimental Design for Evaluating SEES* [Frakes et al, 1995] . In general, error decreases and power increases as the sample size is increased; so, the larger the sample size, the greater the probability of finding a real difference between treatment conditions. With a sample size of two, results cannot be considered to have statistical validity.

4.4.3 Experimental Controls

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

- One possible bias that was eliminated is the effect of grading on the Group 1 development team and IV&V team. Since Group 1 was also participating in a class on IV&V, it was important to eliminate any incentive to produce better work that might occur from grading. Therefore, grades in the class were not based on the quality of the software product developed, or the number of faults detected. Students were carefully instructed as to this fact.

- Interaction between the two development groups could cause multiple biases, including a build-up of competition between the two groups, or the passing of development information between groups. To avoid either of these problems, members of the groups were told specifically not to speak to members in another group. Further, password codes specific to each development group meant that neither group could access the work of the other group on the DEC-alpha machine.

- Interaction between the Group 1 development team and the Group 1 IV&V team was also carefully controlled. The IV&V team imparted IV&V information to the development team only under the supervision of Dr. Arthur.

- Both in an attempt to save time, and as a way to baseline the initial development document used in the experiment, the Software Requirements Specification (SRS) for the MDO application was developed by Dr. Arthur and Markus Groener. Thus, the two development groups received the same baselined SRS.

- Every effort was made to ensure that both groups developed the MDO application under an identical development environment. As noted above, a single DEC-alpha with a Unix environment was used for all development.

- Matching assignment was used to achieve parity, thereby avoiding the high probability of biased subject combinations due to the small sample size.

4.5 Activities of Each Experimental Group for Each Software Development Phase

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.

4.5.1 Requirements Phase

- **Development Activities**

  A Software Requirements Specification was developed by Dr. Arthur and Markus Groener in order to save time and to establish a single baseline for the two development groups. No System Requirements Specification was necessary, as System Requirements were not applicable to the MDO application. The Software Requirements Specification (SRS) was given to both development groups at the same time. Each group was instructed to review the SRS for understanding. The Group 1 development team had no comments, while Group 2 filed several FRFs.

- **IV&V Activities**

  Since SEES is primarily a V&V methodology as far as the requirements phase is concerned, the Group 1 IV&V team conducted a limited IV&V evaluation on the SRS. The IV&V team developed a Software Requirements List (SRL) in accordance with Task 1 of the RTMTAP:2-1. Next, each requirement was analyzed by the IV&V team using the SEES Valid Requirements Criteria of Table 2-1, Task 1, RRATAP:1-1. The checklists from Appendix A of the RRATAP:1-1 were distributed to members of the IV&V team to serve as a guide for reviewing and assessing requirements. Finally, the IV&V team prepared DTRs for each fault it had found with a requirement in the SRL. In a controlled classroom environment, the IV&V team reported its findings to the Group 1 development team, and gave the Group 1 development team a copy of all DTRs created in the requirements phase. The Group 1 development team then reviewed the DTRs and made appropriate corrections.

4.5.2 Design Phase

- **Development Activities**

The design phase was divided into two sub-phases: high-level design (HLD) and low-level design (LLD). In the HLD phase, both the Group 1 and the Group 2 development teams produced a High Level Design document, which broke down the single MDO Computer Software Configuration Item (CSCI) into intermediate Computer Software Components (CSCs). The HLD encompassed the elements of an Interface Design Document (IDD) and a Software Design Document (SDD). Specifically, the HLD contained a Top-Down Modular Structure Chart, a Data Flow Diagram, and an IDD.

In the LLD phase the two development groups added detail to the structure chart, data flow diagram, and interface design, in order to produce the “code to document”, that is, the actual design document from which the code was developed. The CSCs were further divided into Computer Software Units (CSUs) descriptions of routines to be coded. Also in the LLD phase, the development groups produced a Software Test Document (STD). In the STD, a test was developed to test at the intermediate level of each Computer Software Component (CSC).

- **IV&V Activities**

The IV&V team conducted two formal inspections, one of the HLD documents, and the other of the LLD documents. The formal inspection methodology as described in the Design Review and Assessment TAP (DRATAP:3-1) was used. The inspection methodology used in DRATAP is based on the Fagan Inspection Methodology [Fagan, 1976]. For each inspection, a moderator, recorder, and reader were selected. The moderator chaired the inspection process, while the recorder recorded any changes to the design documents, including DTRs; the reader formally read the design documents during the formal inspection. The inspection for both the HLD and the LLD documents took place in four parts.

First, an overview meeting was held to ensure that team members understood the design material to be reviewed, and that the individual inspection procedure was understood. Second, each member acted as an inspector, inspecting the design documents on his/her own before meeting for the inspection session. Individual inspectors reviewed the design documents for defects, clarity problems, or general concerns, listing these concerns on a SEES Individual Preparation Log (IPL). As in the requirements IV&V, checklists from Appendix A of the DRATAP:3-1 were distributed to IV&V inspectors to serve as a reference guide in evaluating the design documents. Dr. Arthur developed additional criteria to focus the design inspection, such as traceability to requirements, and completeness and correctness of the interface, algorithms, and dataflow.

Third, formal inspection meetings were held, in which all the inspectors met, and a formal reading of the design documents ensued. For each design specification, individual participants checked their IPLs for any potential defects; if any were found, the defects were discussed and a group decision was made whether or not to include the defect in the SEES Inspection Defect List (IDL). After an inspection session was completed, the

Recorder prepared a DTR for each defect. Finally, the Moderator prepared a SEES Inspection Summary Report (ISR) which detailed the total person hours spent on an inspection, the total number of major defects, the total number of minor defects, and the total number of DTRs produced. At this point, the DTRs and the IDL were given to the Group 1 development team, under the supervision of Dr. Arthur.

4.5.3 Implementation (Code) Phase

- **Development Activities**

  During the implementation phase, the Group 1 and Group 2 development teams coded each CSU as specified in the respective LLD document. Coding was done in ANSI C, and both groups were directed to use ANSI C coding standards, as well as structured programming standards. Unit testing was also performed on all CSUs during this phase.

- **IV&V Activities**

  The Group 1 IV&V team performed code review and assessment activities on the Group 1 team's code. Whenever a defect was found, a DTR was prepared. First, all code was reviewed for conformance to ANSI C Standards [Kernighan and Ritchie, 1988]. Second, the IV&V team reviewed the code for conformance to standard coding practices, such as structured programming, proper indentation, complete documentation, etc. [Schaeffer, Coding Standards, Dept. of Computer Science, Virginia Tech]. Finally, an automated software metrics tool, "PC-Metric", was applied to the code to measure code quality. PC-Metric reports such common software metrics as N-path, the Halstead Software Science Measures, and McCabe's Complexity Metric. The main purpose of using the metrics analyzer was to give the IV&V analysts enough information to determine which code modules should be further investigated based on non-conformance to established metrics thresholds. Although running the metrics analyzer was automated, the IV&V team had to manually analyze the results. DTRs were prepared based on this further analysis. The products of this IV&V phase included DTRs and a printout of the metrics analysis with threshold values.

4.5.4 Integration and Testing Phase (IV&V Validation)

Our intent was to perform complete system validation, yet time constraints meant that the Group 1 development team was not able to begin integration and testing. Group 2 did complete integration and testing; but lacking Group 1's integration and testing results, Group 2's integration and testing results have not been included in this report. Instead, Group 2's integration and testing results will be used as a baseline for future experiments.

4.6 Data Collection Process

The data collection process for the SEES' experiment has been designed to minimize bias in measurement. An individual who was not part of either treatment group was responsible for gathering and organizing the data collected by both Group 1 and Group 2. Although team individuals were still responsible for recording correct values on data collection forms, the independent data collector for each group reviewed the reported data for possible errors. First, the independent data collector checked each form for completeness before accepting it; second, the independent data collector ensured that all data was collected in an orderly manner, catalogued by phase. See Appendix D for all data collection forms used in this experiment.

4.6.1 Measurement of Variables

Methods for measuring the dependent variables in the SEES' experiment are discussed in Section 4.3.4. However, there are several other aspects to measurement that the researcher must address, including measuring instruments, construct validity, reliability, precision, level of measurement, and survey instruments. These aspects are discussed in detail in Section 3.3 of the *Experimental Design for Evaluating SEES* [Frakes et al, 1995] .

5. RESULTS

As described earlier in this report, the data represents a characterization of two independent software development efforts and is focused on assessing the impact of IV&V during product development. The data reported below spans a development effort that begins with the Requirements Analysis and continues through Integration and Testing (I&T). More specifically, Section 5.1 addresses the impact of IV&V on early faults detection and the mean time to remove those faults. Section 5.2 describes the impact of IV&V on operational correctness. Finally, Section 5.3 presents a stastical analysis of the results relative to their significance.

5.1 Detecting Faults Earlier and Reducing the Mean Time to Remove those Faults

The results reported in this section are conveyed through two instruments: (1) a set of tables summarizing fault data collected during the experiment, and (2) a set of graphs from which observations and suppositions are made, and conclusions are drawn. The data, graphs, and discussions focus on the impact that IV&V has on supporting the detection of faults earlier in the software development process and, as a consequence, the reduction of the mean time to remove those faults.

5.1.1 Data Summarization for Group 1 - The In-Class IV&V and Development Teams

The following five tables summarize the fault counts and time/effort (observed and estimated) required by the Group 1 development team to fix the faults. To facilitate understanding, comprehension and analysis, the count and effort data are categorized based on the phase in which the faults are detected. The following data is reported for Group 1, the in-class IV&V and development teams.

Table 1: Faults found by both the IV&V and development teams are reported. As expected, the IV&V team played the more crucial role during requirements analysis. Consequently, most (and in fact, all) faults found during requirements analysis were found by the IV&V team. All faults were noted and corrected before high-level design began.

Faults reported in those phases following Requirements Analysis (i.e., High-Level design, Low-Level Design, Code and Unit Testing, and I&T) represent those that (a) were identified by either the IV&V or development team and (b) had correction times logged against them by the development team.

Phase Fault Found	Non-Critical Faults	Critical Faults	TOTAL
Reqmnts - IVV	39	16	55
Reqmnts - DEV	0	0	0
HLD	33	20	53
LLD	52	31	83
Code/UT	2	24	26
I&T	0	6	6
TOTAL	126	97	223

Table 1: Fault Count for Group 1 (In-Class)

Table 2: The times reported in Table 2 reflect the effort required by the (in-class) development team to correct the faults summarized in Table 1. Note that no time is reported for the correction of requirement faults. These were corrected by the “requirements engineers” (Arthur and Groener).

Phase Fault Found	Non-Critical Faults	Critical Faults	TOTAL
Requirements	00:00:00	00:00:00	00:00:00
HLD	02:20:00	00:45:00	03:05:00
LLD	03:40:00	01:30:00	05:10:00
Code/UT	00:16:00	07:37:00	07:53:00
I&T	00:00:00	04:32:00	04:32:00
TOTAL	06:16:00	14:24:00	20:40:00

Table 2: Effort (Observed) Reported To Remove Faults

Table 3: $MER_{observed}$ is defined to be the mean observed time to remove a given fault. As illustrated in the following formula, $MER_{observed}$ is computed using the number of faults reported in Table 1 and the times it took to remove those faults (shown in Table 2). Table 3 summarizes $MER_{observed}$ for this experiment. $MER_{observed}$ is computed as followed:

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

where n is the number of reported faults and
F_m^{Obs} is the observed effort to fix $fault_m$.

Phase Fault Found	**Non-Critical Faults**	**Critical Faults**	**TOTAL**
Requirements	00:00:00	00:00:00	00:00:00
HLD	00:04:15	00:02:15	00:03:29
LLD	00:04:14	00:02:54	00:03:44
Code/UT	00:08:00	00:19:03	00:18:12
I&T	00:00:00	00:45:20	00:45:20
TOTAL	00:04:19	00:10:40	00:07:23

Table 3: Mean Effort (Observed) to Remove Faults
-- MER Observed--

Table 4: Table 4 provides an alternate measure of effort expended by Group 1 to correct those faults summarized in Table 1. The effort measures shown in Table 4 are derived from an augmented set of effort estimates provided by Boehm [Boehm, 1981 (fig 4-2)].

More specifically, Boehm's model provides an effort estimate reflecting the "distance" between the phase in which the fault is introduced and the phase in which the fault is detected/removed. Associated with each fault logged by Group 1 (and Group 2) is the phase in which the fault was found and the (conjectured) phase in which it was introduced. Using the augmented set of Boehm's estimates, the data reported in Table 4 reflects a simple sum of the estimated effort to remove each fault.

Phase Fault Found	**Non-Critical Faults**	**Critical Faults**	**TOTAL**
Requirements	0	0	0
HLD	42.9	26	68.9
LLD	67.6	40.3	107.9
Code/UT	8	61	69
I&T	0	96	96
TOTAL	118.5	223.3	341.8

Table 4: Estimated Effort to Remove Faults

Table 5: $MER_{estimated}$ uses Boehm's estimated effort values to compute the mean time required to remove the faults. The formula for $MER_{estimated}$ is:

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

where n is the number of reported faults and

$F_m\left(I,R\right)$ is the estimated effort to remove fault_m introduced during phase I and removed during phase R.

For the purpose of this experiment $MER_{estimated}$ is computed from the fault count reported in Table 1 and the estimated times required to remove those faults (reported in Table 4).

Phase Fault Found	Non-Critical Faults	Critical Faults	TOTAL
Requirements	0	0	0
HLD	1.3	1.3	1.3
LLD	1.3	1.3	1.3
Code/UT	4	2.5	2.7
I&T	0	16	16
TOTAL	1.4	2.8	2.0

Table 5: Mean Time (Estimated) to Remove Faults
--MER Estimated--

5.1.2 Data Summarization for Group 2 - The Independent Development Team

The data reported in this section reflects the activities of Group 2, the independent development team. It summarizes the fault counts and time/effort (observed and estimated) to fix the faults. As in the previous section the counts and effort estimates have been categorized based on the phase in which the faults were detected.

Table 6: Table 6 categorizes and summarizes the number of faults found by Group 2. Note that no faults are reported during the requirements phase. Although Group 2 read the requirements for understanding, they found no obvious faults or ambiguities. They did not, however, perform any formal analysis of the requirements.

Phase Fault Found	Non-Critical Faults	Critical Faults	TOTAL
Requirements	0	0	0
HLD	0	2	2
LLD	2	8	10
Code/UT	1	34	35
I&T	1	14	15
TOTAL	4	58	62

Table 6: Fault Count for Group 2 (Independent)

Table 7: The data recorded in Table 7 are the observed times reported by Group 2 to correct the faults summarized in Table 6.

Phase Fault Found	Non-Critical Faults	Critical Faults	TOTAL
Requirements	00:00:00	00:00:00	00:00:00
HLD	00:00:00	01:01:00	01:01:00
LLD	00:06:00	00:40:00	00:46:00
Code/UT	00:10:00	01:30:00	01:40:00
I&T	00:10:00	16:39:00	16:49:00
TOTAL	00:26:00	19:50:00	20:16:00

Table 7: Effort (Observed) Reported to Remove Faults

Table 8: $MER_{observed}$ is the mean time required by Group 2 to remove the faults reported in Table 6. Like Group 1, $MER_{observed}$ is computed by:

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

where n is the number of reported faults and
F_m^{Obs} is the observed effort to fix $fault_m$.

Phase Fault Found	Non-Critical Faults	Critical Faults	TOTAL
Requirements	00:00:00	00:00:00	00:00:00
HLD	00:00:00	00:30:30	00:30:30
LLD	00:03:00	00:05:00	00:04:36
Code/UT	00:10:00	00:02:39	00:02:51
I&T	00:10:00	01:11:21	01:07:16
TOTAL	00:06:30	00:20:31	00:19:37

Table 8: Mean Effort (Observed) to remove Faults
--MER Observed--

Table 9: Similar to Table 4 the data provided in Table 9 is derived from an augmented set of effort estimates provided by Boehm [Boehm, 1981 (fig 4-2)]. More specifically, Boehm's values provide and estimated effort to remove a fault introduced in Phase I and corrected in Phase R. The wider the "distance" (in phases) between Phase I and Phase R, the larger the estimated effort value. Correspondingly, when Group 2 (and Group 1) reported a fault, they included information indicating during which phase it was found and in which phase it was introduced (the latter phase is putative).

The data reported in Table 9 is a sum (by indicated categories) of the estimated effort to remove corresponding set of faults reported in Table 6.

Phase Fault Found	Non-Critical Faults	Critical Faults	TOTAL
Requirements	0	0	0
HLD	0	2.6	2.6
LLD	5.3	15.8	21.1
Code/UT	4	71.2	75.2
I&T	10	146	156
TOTAL	19.3	235.6	254.9

Table 9: Estimated Effort to Remove Faults

Table 10: $\mathrm{MER}_{\mathrm{estimated}}$ is the estimated effort required to remove faults. In computing $\mathrm{MER}_{\mathrm{estimated}}$ the fault count reported in Table 6 and the estimated times shown in Table 9 are related through the following formula:

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

where n is the number of reported faults and
$F_m(I, R)$ is the estimated effort to remove fault_m introduced during phase I and removed during phase R.

Phase Fault Found	Non-Critical Faults	Critical Faults	TOTAL
Requirements	0	0	0
HLD	0	1.3	1.3
LLD	2.7	2.0	2.1
Code/UT	4	2.1	2.1
I&T	10	10.4	10.4
TOTAL	4.8	4.1	4.1

Table 10: Mean Effort (Estimated) to Remove Faults
--MER Estimated--

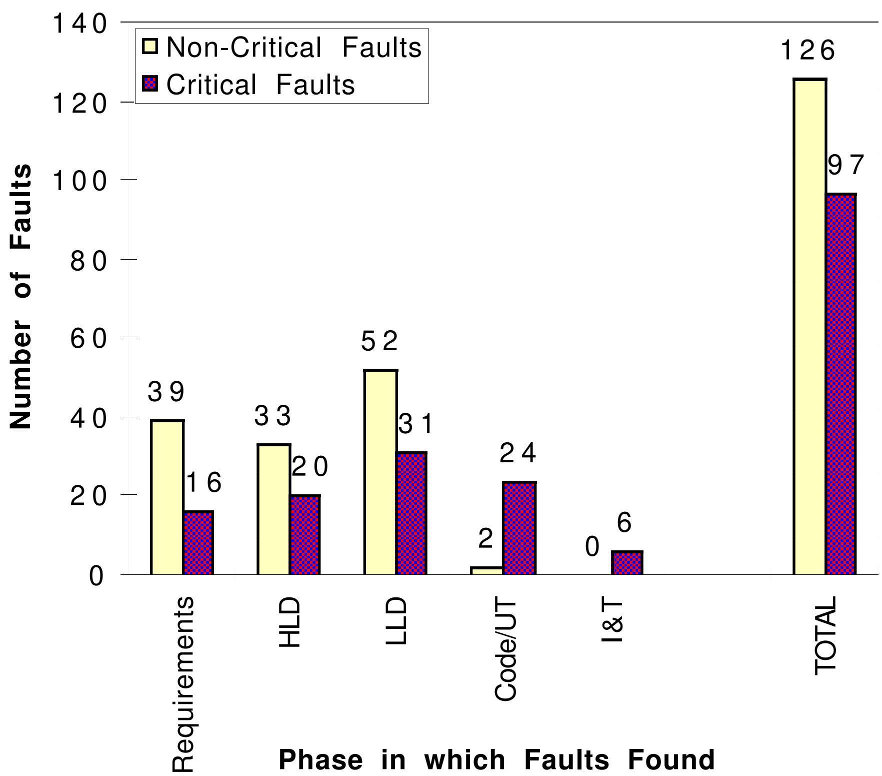
5.1.3 Analysis of Fault Detection and Removal Data

Given a formal V&V process that operates in tandem with development activities, development documents are examined and verified as they are produced. Consequently, faults are expected to be detected early in the software development lifecycle. In turn, early fault detection leads to a reduction in the effort (cost) to remove the faults. Within the confines of our experiment, the graphs shown below illustrate the impact that IV&V has had on both early fault detection and the effort required to remove those faults

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 1
IV&V Group: Faults Found

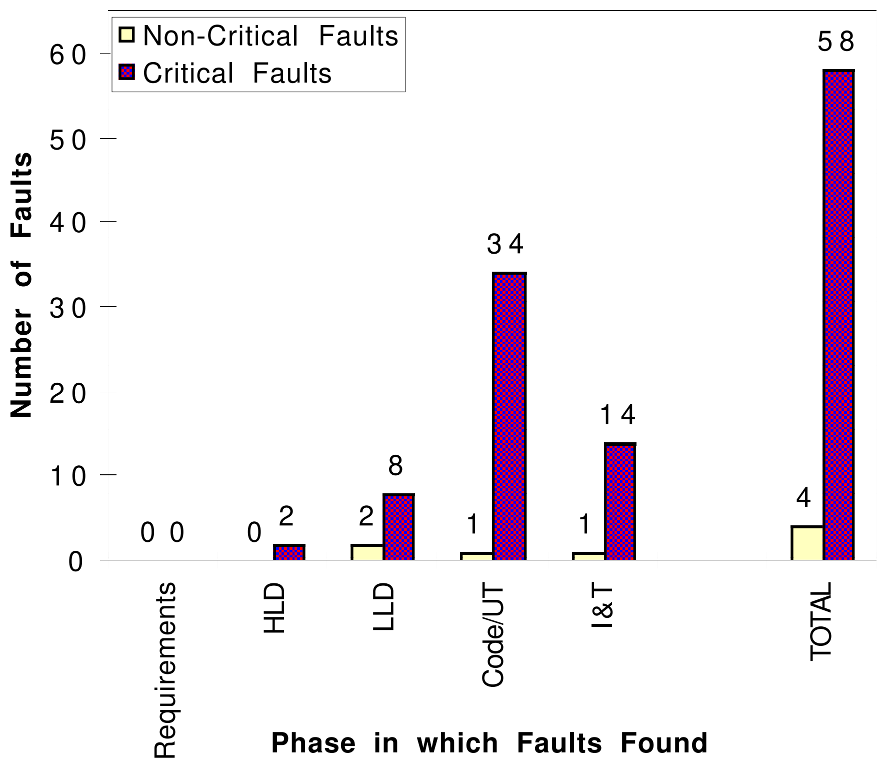


Graph 2 depicts the pattern of fault detection for Group 2, the non-IV&V development group. It, too, categorizes the fault counts within development phases beginning with Requirements and continuing through I&T.

As illustrated in Graph 2, and in particular for critical faults, the number of faults detected grow exponentially as development proceeds from Requirements through Code and Unit Testing, 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

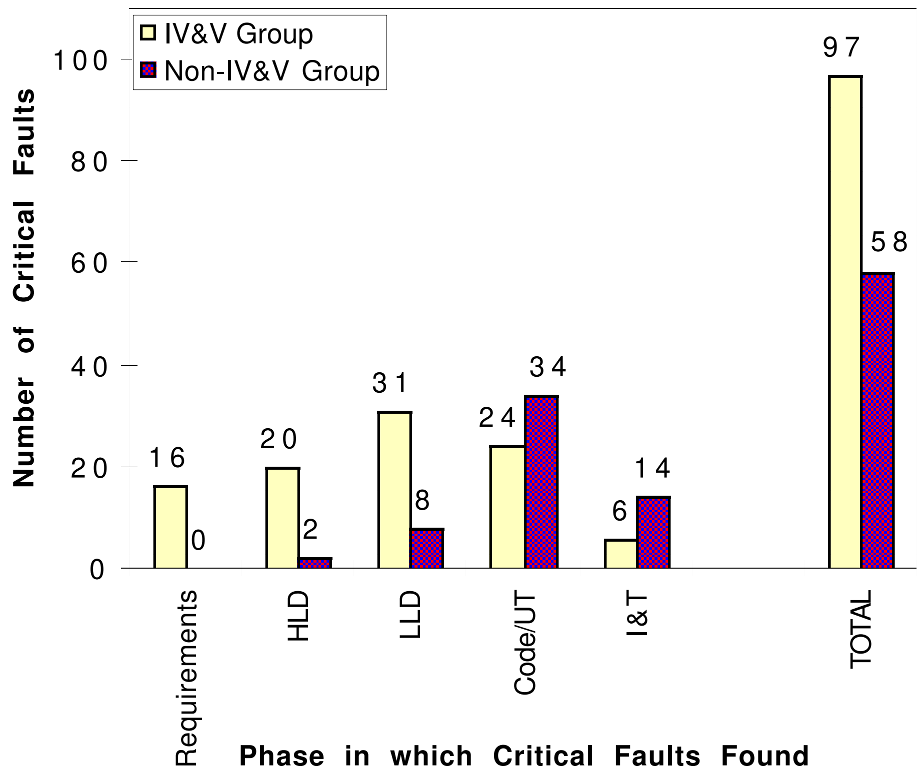


Conclusion 1:	**Given the assumption that we have equally proficient development teams, a comparison of Graph 1 and Graph 2, indicates that employing SEES' during the development process has a beneficial impact of enabling the earlier detection of faults**.

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.

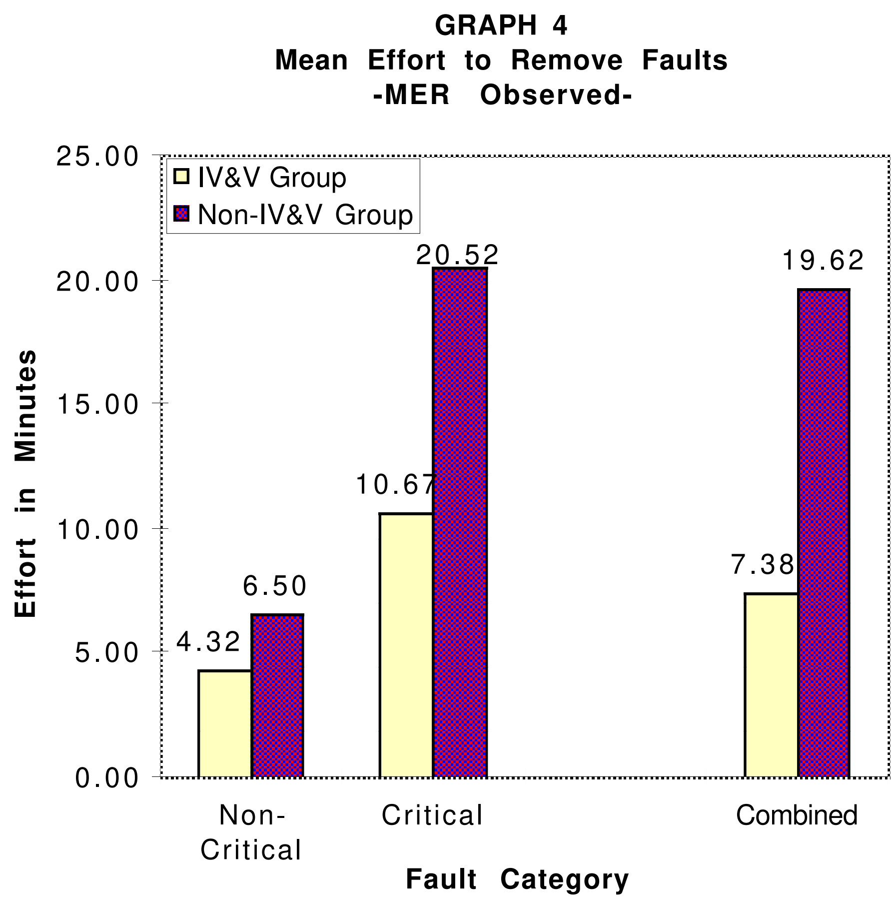
We tend to discount both (1) and (2) above because:

- 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.

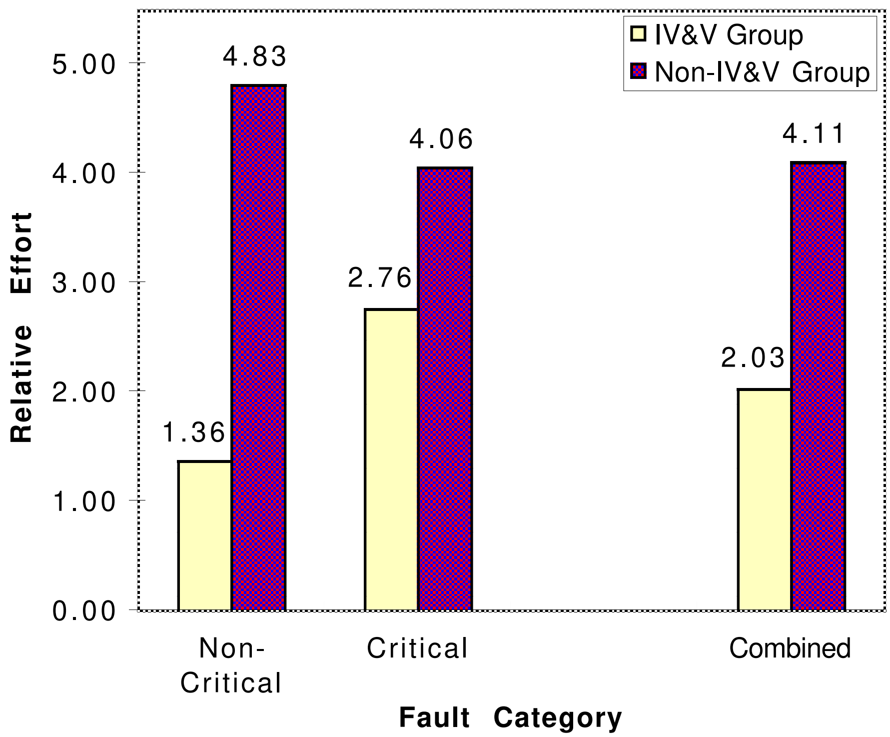


Based on our assumption that IV&V enables faults to be detected earlier in the development process, and therefore, reduces the average time to fix faults, one would expect that over similar periods of time the $MER_{observed}$ for Group 1 would be less than that of Group 2. *In each case depicted in 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 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.

5.2 Enhancing Operational Correctness

In addition to the more high-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 appeared to meet the *original* set of requirement specifications. We stress "original" because the final set of requirement specifications differed for each development team. Independently, each group refined their respective set of specifications through questions directed to the customer (Arthur and Groener). Group 1, through their formal requirements analysis process, evolved a more comprehensive, well-defined and stringent set of specifications than did Group 2.

The test suite described below focuses on many aspects of the software systems. 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.

5.2.1 The Test Suite

The test suite consist of 36 independent tests constituting eight distinct test classes. Each class is described below. Specifics for each individual test can be found in the "Supporting Data" addendum at the end of this report.

- 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 (an those stated in the requirements) are that the software system will successfully terminate with an "iteration limit exceeded" message. Two (2) such tests were used.

- 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 last two classes reflect knowledge gained through an examination of each group's code. Dr. Arthur examined the code produced by Group 1 while Mr. Groener examined Group 2's code. The examinations revealed both desirable and undesirable characteristics in both code sets. Based on such observations, two additional sets of tests were 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.

5.2.2 A Test of Operational Correctness

Table 11 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 11 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.

  (We note that code was not verified back against the design document because that was not an activity prescribed by SEES.)

- 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 Weight 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 Code Particulars	5	5	0	0	5
Group 2 Code Particulars	5	4	1	2	3
	36 **Total**	33 **Passed**	3 **Failed**	11 **Passed**	25 **Failed**

Table 11: Operational Correctness

Conclusion 4: Applying the IV&V activities prescribed by SEES' does result in a product whose operational correctness substantially exceeds that of the product developed by the Non-IV&V Group.

5.3 Assessing Statistical Significance

As stated is Section 1.2 the experimental hypothesis is that "independent verification and validation (IV&V) adds value to the existing software development process." Although "value added" has many facets in software development, we restrict our boundaries to include only those qualitities that can be determined through direct examination of the available data. In particular, we focus our attention on three specific elements:

(1) controlling the software development process,

(2) the time it takes to find faults, and

(3) the time it takes to repair faults.

Becuase we desire to examine these three qualities from a statistical perspective, we assume a hypothesis which states: “no difference exists between software developed using IV&V techniques and software developed in the absence of IV&V”, and try to reject it. In the following three sections, the null hypothesis will be reformulated to address more specifically each of three qualities.

5.3.1 Controlling the Software Development Process

One approach to assessing process control is based on measuring the variability of process artifacts. 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 ratio actually observed occurrs by chance. Figure 1 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 1: Descriptive Statistics

Figure 2 provides the results of applying the F-Test to the two sets of recorded times to reapir faults. The P-Value reveals that probability of such disparity occuring by chance is less that 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 2: 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) variability in time to fix faults is an indicator of control (or lack thereof) in a development effort, (b) that difference in that variability between the two activities is statistically significant, and (c) the presence of an IV&V component is the major distinguishing characteristic bewteen the two development efforts, we conclude the following:

Conclusion 5:	**The IV&V development effort has 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 contol over the the development process.**

5.3.2 Examining The Time It Takes To Find Faults

From a statistical perspective the second question we wish to address is: "Is there a statistically significant difference between the two development efforts in the time it takes to find faults?" Conclusion 1 offered in Section 5.1.3 implies that there is a difference - the IV&V group appears to find faults earlier in the software development process. Graph 3 supports this conclusion. In exploring this question and conclusion further, we base our analysis on two elements: (1) the Chi-Square Test applied to phase-specific fault detection data, and (2) $MER_{estimated}$ to indicate the direction of the observed relationship.

<u>The Chi-Square Test</u>:

For each fault, both groups recorded the development phase in which that fault was found. An examination of that data shows that the IV&V group detected more errors earlier in the software development than did the Non-IV&V group (see Section 5.1.3). Nontheless, we must ask - is the observed difference statistically significant? Or more specifically - does a statistically significant relationship exist between the use/non-use of IV&V activities and the length of time to find a fault?" Our hypothesis states that a statistically significance difference does exist between the use/non-use of IV&V activities and the length of time it takes to find a fault. Note, we are not attempting to make any judgement as to which activity (IV&V or Non-IV&V) better supports early fault detection, but simply that a statistically significant difference exists.

Our null hypothesis, therefore, states that "no significant relationship exsits between the use/non-use of IV&V activities and the length oftime to find a fault." Because the test data is categorical, i.e. phases of the development lifecycle, we must apply the Chi-Square Test if we are to reject the null hypothesis.

Figures 3b - 3d show the results of applying the Chi-Square Test to the reported frequencies show in Figure 3a. For example, the IV&V group reported 16 faults found during design while the Non-IV&V group found 21 faults during that phase. The expected frequencies for the phase are 21.56 and 15.43, respectively. Is the overall difference statistically significant? The Chi-Square P-Value of .016 indicates "Yes", with 98% probability. Therefore, we reject our null hypothesis 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."

	Req	Des	Cde	I&T	Totals
IVV	63	16	0	2	81
NIVV	32	21	3	2	58
Totals	95	37	3	4	139

Figure 3a:
Observed Frequencies by Phase

	Req	Des	Cde	I&T	Totals
IVV	55.360	21.561	1.748	2.331	81.000
NIVV	39.640	15.439	1.252	1.669	58.000
Totals	95.000	37.000	3.000	4.000	139.000

Figure 3b:
Expected Values For Each Phase

	Req	Des	Cde	I&T
IVV	1.054	1.434	1.748	.047
NIVV	1.473	2.003	2.441	.066

Figure 3c:
Cell Chi Squares By Phase

Chi Square	10.267
Chi Square P-Value	.0164

Figure 3d:
Summary Table for Phase

What the Chi-Square Test does not tell us, however, is which approach (IV&V or Non-IV&V) is better for reducing the time to find a fault.

$\underline{MER_{estimated}}$:

The Chi-Square Test provides an indication that a significant difference exists between the use/non-use of IV&V activities and the length of time to find a fault. An examination of $MER_{estimated}$ reveals that the use of IV&V reduces the length to find faults. Using Boehm's relative effort measures, the IV&V group requires 2.8 time units to find a fault while the Non-IV&V group requires 4.1 time units (see Graph 5 - Critical Faults).

Based on results from the Chi-Square Test and an examination of the $MER_{estimated}$ we offer Conclusion 6:

Conclusion 6: The IV&V development effort exhibits a statistically significant reduction in the time required to find faults.

5.3.3 Examining The Time It Takes To Fix Faults

Graphs 5 and 6 support a reasoned conclusion 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 now ask, is that result statistically significant?

To address this question we would like to use the T-Test to we 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. Section 5.3.1 provides an analysis of the variances and, unfortunately, concludes 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 - several of which are present in the existing data set. Nonetheless, because the Wilcoxon Rank Sum Test ignores data values, using their ranks instead, it is perceived to be a somewhat weaker test that the T-Test.

Like the T-Test, the Wilcoxon Rank Sum Test also assumes that the variances to not differ significantly. To remove the variability, each value in the two distributions were divided by their respective standard deviation - effectively normalizing the variances by making the standard deviation equal to one. (Similar transformations were tried when applying the T-Test, but with little success.)

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 that 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.

Conclusion 7:	**Based on an examination of the actual time to repair faults, the difference between the IV&V and Non-IV&V data sets is statistically significant.**

6. SUMMARY AND CONCLUSIONS

During the Summer of 1995 the authors conducted an experiment focused on answering the question:

Does SEES add value to a software development process?

That experiment entailed:

- the modification of SEES to reflect the pragmatic constraints of our experiment -- the modified version of SEES is referred to as SEES’,
- 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’,
- the collection of data ranging from activity logs and fault reports to development artifacts, and finally
- the writing of this report which details the experiment design and summarizes the findings.

The test results and individual conclusions are reported in Section 5. They are reformulated and reiterated below to present a comprehensive picture.

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

That reduction in time to find faults has been shown to be statistically significant.

(2) Applying SEES' to the development process enables the early identification and correction of ambiguous and unclear requirement and design specifications that have the <u>potential</u> for evolving into critical faults during later stages of the development process.

(3) Guided by the data reported in this experiment, SEES' has had a beneficial impact in reducing the effort to fix faults.

That reduction in time to repair faults has been shown to be statistically significant.

(4) Applying the IV&V activities prescribed by SEES' does result in a product whose operational correctness substantially exceeds that of the product developed by the Non-IV&V Group.

(5) The IV&V development effort has less variability in the time to fix faults, and therefore, exhibits better control over the software development process than does the Non-IV&V effort.

That reduction in variability has been shown to be statistically significant.

In summary, 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 detection of ambiguous or unclear specifications that can lead to the introduction of critical faults downstream in the development process,
- the reduction in the effort required to repair individual faults,
- better control over the software development process, and
- enhanced operational correctness.

SEES defines one 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. 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 comment:

> Our quasi-experiment does illustrate 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.

APPENDIX A: GLOSSARY

APPENDIX: A

GLOSSARY

- ***Adherence to Standards*** This refers to how closely the software development effort has followed and implemented the various software development standards. These standards may pertain to design, code verification, language, hardware, customer requirements, documentation, etc.
- ***Audit-Level IV&V*** Based on how many phases of the Software Development Life Cycle (SDLC) the IV&V effort covers, there are four IV&V categories. *Audit-Level IV&V* refers to a minimal, even superficial, Independent Verification & Validation (IV &V) effort, where an IV&V team is called in to audit documentation, plans, and procedures for correctness and compliance with standards [4].
- ***Bias*** Bias involves errors that are always in the same direction. A bias in statistics is the degree to which the average of an infinite number of determinations of a statistical value, such as the mean, fails to equal the corresponding parameter (e.g., the population mean) [3].
- ***ccount*** This is a C source code metrics tool, developed by Frakes, et al., that reports counts of commentary and noncommentary source lines, and comment-to-code- ratios [2].
- ***Completeness*** This quality criterion refers to whether all functions, features, capabilities, constraints, and other properties of the target system have been specified in depth as part of the requirements or the design, or that they have been fully implemented or tested in the code development or the testing phase [2] .
- ***Computer Software Component (CSC)*** A functionally or logically distinct part of a *computer software configuration item (CSCI),* typically an aggregate of two or more *computer software units* (CSUs) [1].
- ***Computer Software Configuration Item (CSCI)*** An aggregation of *Computer Software Units (CSCs)* that is designated for configuration management and treated as a single entity in the configuration management process [1].
- ***Computer Software Unit (CSU)*** These "units" are software modules that perform a function or a single routine. CSUs are basic software modules that cannot be further decomposed [4].

  Concept and Exploration Phase The period of time in the software development cycle during which the user needs are described and evaluated through documentation (for example, statement of needs, advance planning report, project initiation memo, feasibility studies, system definition, documentation, regulations, procedures, or policies relevant to the project) [1].
- ***Configuration Item (CI)*** An aggregation of hardware, software, or both, that is designated for configuration management and treated as a single entity in the configuration management process [1].
- ***Configuration Management (CM)*** In a software project, configuration management refers to the identification, organization and control of changes to the software and accompanying documentation being developed. As well as monitoring changes to development

products, versions of development products are coordinated to produce correct and consistent product releases [2].

- ***Consistency*** This quality criterion refers to the degree of uniformity, standardization, and freedom from contradictory information among the documents or parts of a system or document [1].
- ***Correctness*** This quality criterion refers to:
    1. The degree to which a system or component is free from faults in its specification, design, and implementation;
    2. The degree to which software, documentation, or other items meet specified requirements;
    3. The degree to which software, documentation, or other items meet user needs and expectations, whether specified or not [1].
- ***Dependent Variable*** (Also known as the criterion, outcome, predicted, or response variable). The *variable* that is thought to be affected by an *independent variable* [3].
- ***Design Phase*** The period of time in the software development life cycle during which the designs for architecture, software components, interfaces, and data are created, documented and verified to satisfy requirements [1].
- ***Detailed Design (Low Level Design - LLD)*** The process of refining and expanding the *preliminary design (high level design - HLD)* of a system or component to the extent that the design is sufficiently complete to be implemented; the document that results from the this process [1].
- ***Documentation Trouble Report (DTR)*** This is a list of all the potential faults that are discovered during the various stages of the *Software Engineering Evaluation System (SEES)* process. DTRs do not correct, update, or change targeted documents. They simply act as an instrument for reporting a trouble spot in the documentation [6].
- ***Dynamic Analysis*** *This Independent Verification and Validation* (*IV&V*) analysis method involves execution of a system or component to detect errors by analyzing the product's behavior in response to sets of input data. Testing, simulation, and prototyping are all examples of dynamic analysis [1].
- ***Endgame IV&V*** Based on how many phases of the Software Development Life Cycle (SDLC) the IV&V effort covers, there are four IV&V categories. Endgame IV&V emphasizes the validation of the integration and testing phase, with little or no involvement in verification of the requirements, design, and implementation phases [4].
- ***External Validity*** The extent to which research findings can be applied outside the specific research situation (also known as generalizability)[3].
- ***Formal Analysis*** This type IV&V analysis uses rigorous mathematical techniques to analyze correctness of the logic underlying a solution. For example, verification of specifications that have been written in a formal specification language can be verified using proof of correctness techniques. The most common type of formal analysis used during IV&V is the algorithmic analysis of the software.
- ***Formal Inspection*** This is an analysis technique in which a designer or programmer leads members of the development team and other interested parties through a segment of documentation or code, and the participants ask questions and make comments about possible errors, violation of development standards, and other problems. A formal inspection is a special form of a walkthrough and was developed at IBM and is often referred to as Fagan

Inspections. A formal inspection is always a peer review, it is led by a moderator, and the collection of anomalies is carefully structured to capture statistical evidence of the effort [5].

- ***Formal Qualification Test (FQT)*** The FQT validates that all the requirements allocated to software have been correctly and completely implemented in the software code. The FQT must permit a determination that 1) the functional and performance capabilities of the software meet the contract, and 2) the software is suitable for higher level testing and/or deployment to the field [6].
- ***Full Cycle In-Phase IV&V*** Based on how many phases of the Software Development Life Cycle (SDLC) the IV&V effort covers, there are four IV&V categories. Full Cycle In-Phase IV&V is the most comprehensive and cost-effective, spanning four phases of the SDLC, from requirements to acceptance testing. IV&V activities are performed in parallel (in phase) with each phase of the SDLC. The IV&V team detects faults as they occur during each development life cycle phase, and continuously suggests changes to the development team to improve the processes that caused the faults [4].
- ***HLD*** see *Preliminary Design*.
- ***Implementation Phase*** The period of time in the software development life cycle during which a software product is coded from design documentation and debugged [1].
- ***Independent Variable*** The *variable* or treatment condition that the researcher hypothesizes will produce an effect on the *dependent variable* [3].
- ***Independent Verification and Validation (IV&V)*** Independent Verification and Validation is a process whereby a system is comprehensively analyzed and tested during all phases of the development life cycle by an organization or group that is technically, managerially, and financially independent from the software development team. This systems engineering activity involves not only the software but also the interfaces to system hardware, human operators, and other external software. IV&V activities examine the technical content and the quality of the software products to ensure technical correctness and completeness of the software products [1, 5].
- ***Individual Preparation Log (IPL)*** This log is used in the Design Review and Assessment Technical assessment Procedure (DRATAP: 3-1*)* as part of the individual inspection preparation that takes place prior to the formal inspection session of the software design and design documents. Individual inspectors use the IPL to document suspected faults, clarity problems, or general concerns that they may have about the design. These faults are then discussed in the formal inspection session[6].
- ***Inspection Defect List (IDL)*** This list is prepared during the *Design Review and Assessment Technical Assessment Procedure (DRATAP: 3-1)*, as part of the Inspection Session. In the design and design document review process, each design specification is assessed in conjunction with the *Individual Preparation Log (IPL)* comments and a consensus is reached concerning the issue as a fault. These faults, if any exist, are listed in the IDL, along with the agreed upon Defect Classifications [6].
- ***Inspection Summary Report (ISR)*** This report is prepared during the Design Review and Assessment Technical Assessment Procedure (DRATAP: 3-1), once the Inspection Session is completed. This report provides a documentation of the details and results of the inspection: the total person hours spent on an inspection, the total number of major and minor faults, and the total number of *Documentation Trouble Reports (DTRs)* produced [6].
- ***Interface Design Document (IDD)*** The interface design requirements are developed in this document and it provides the instructions and parameters for designing and operating the designated interfaces. A common boundary is defined that may exist between two or more software products and computer systems which are provided by two or more different organizations or sources [5].

- ***Internal Validity*** A consideration of whether the *independent variable* produced variation in the *dependent variable,* i.e., whether your results are due to the application of the treatment condition and not something else[3].
- ***LLD*** see *Detailed Design.*
- ***Maintenance Phase*** The last phase in the software development life cycle in which the software product has been deployed and faults are corrected and the system is changed or enhanced to reflect ongoing changes in the user environment[2].
- ***MDO (Multidisciplinary Design Optimization) Application)*** This application is based on the work by Mason and Malone at Virginia Tech (1991). MDO has been used as the vehicle through which *Independent Verification and Validation (IV&V)* techniques are applied throughout the software development lifecycle. MDO is aeronautical in nature and it computes performance, aerodynamic and weight characteristics based on mission and design parameters such as wing area, cruise altitude, mach speed, and various weight factors. The computational process is iterative and stresses the convergence to an ideal aircraft weight. For purposes of the SEES' experiment, the MDO application was scaled to use a single parameter, weight, in the optimization module [7].
- ***Null Hypothesis*** The null hypothesis hypothesizes that there is no relationship, no difference, or the status quo of a phenomenon or relationship i.e., the null hypothesizes that there will be no difference in the effect of the *independent variables* on the *dependent variable* s[3].
- ***Partial IV&V*** Based on how many phases of the Software Development Life Cycle (SDLC) the IV&V effort covers, there are four IV&V categories. Partial IV&V refers to any IV&V effort that begins after the requirements phase. Products of all development phases are verified, even if the phase occurred before the IV&V began [4].
- ***Preliminary Design (High Level Design - HLD)*** A high level analysis of the design alternatives , the architecture, components, interfaces, and timing and sizing estimates for a system or component; the document that results from this process [1].
- ***Reliability*** A measuring instrument is reliable to the extent to which it would give the same value if used over and over, provided that the attribute measured did not change [3].
- ***Requirements Phase*** The period of time in the software development life cycle during which the requirements for a software product are defined and documented. A software product requirement is a feature, function, capability, or property that a software product must have [1, 2].
- ***Software Design Document (SDD)*** The SDD provides a high level description of how the *Software Requirements Specification (SRS)* requirements will be met for the software product . The SDD is further refined during the *detailed design (LLD)* at which point the SDD becomes the "code to" document[5].
- ***Software Engineering Directorate (SED)*** This is a body of the U.S. Army Missile Command's Research, Development, and Engineering (RD & E) Center. The SED developed the Software Engineering Evaluation System (SEES) [6].
- ***Software Engineering Evaluation System (SEES)*** This is a comprehensive software Verification and Validation (V&V) methodology that has been developed by the U.S. Army Missile Command's Software Engineering Directorate (SED). Comprehensive means that SEES is a *Full Cycle In-Phase V&V* approach to V&V. SEES was originally developed to evaluate critical Battlefield Automation Systems (BAS). SEES is based on a five step implementation, carried out by a trained team from SED in concert with the development organization [6].

- ***Software Metrics*** These are quantitative measures of software life-cycle products or processes [2].
- ***Software Requirements List (SRL)*** The SRL for the software (target system) is prepared during the Requirements Review & Assessment Technical Assessment Procedure (RRATAP: 1-1), as part of SEES. The SRL is a list of all the software requirements statements containing a "shall" verb. The SRL is used for all further IV&V analyses of software requirements in the SEES process[6] .
- ***Software Requirements Specification (SRS)*** Documentation of the essential requirements (functions, performance, design constraints, and attributes) of the software and its external interfaces (IEEE Std 10122-1986). This document is prepared in the requirements phase of the software life cycle [1, 4].
- ***Software Test Description (STD)*** The STD defines the details of the tests to be executed in the testing phase. These tests are used to validate the requirements in *the Software Requirements Specification* (*SRS*). This document is prepared during the detailed design phase and it specifies the test inputs, execution conditions, and predicted results for an item to be tested[1, 4, 6].
- ***Software Test Plan (STP)*** This document is prepared during the *preliminary design* phase and it describes the scope, approach, resources, and schedule of intended test activities [1].
- ***Spiral Model*** A model of the software development process in which the constituent activities, typically requirements analysis, preliminary and detailed design, coding, integration, and testing, are performed iteratively until the software is complete [1].
- ***Static Analysis*** This type of evaluation, used in IV&V, involves form, structure, content, or documentation analyses of a lifecycle product, without any execution of the product. Examples of static analysis techniques include reviews, audits, data flow analyses, and inspections, such as the Fagan formal inspection process [1].
- ***Technical Assessment Procedures (TAPs)*** The TAPs are the heart of the SEES process and they provide for standardized and calibrated procedures to accomplish the desired V&V support. Each TAP describes a specific methodology to accomplish the objectives of the evaluation effort. Specific technical, effort, and schedule metrics which are provided to the SEES Metrics Laboratory are identified for each TAP. Each TAP contains guidelines which permit a customization based upon the risk, cost, and schedule of the program under evaluation [6].
- ***Test Accuracy*** Any test activity involves system or component execution under specified conditions, observation and recording of results, and an evaluation of some aspect of the system or component. Accuracy of this test is determined by performing qualitative assessment of correctness or freedom from error analysis and a quantitative measure of the magnitude of error of the test [1].
- ***Test Coverage*** (1) The degree to which a given test or set of tests addresses all specified requirements for a given system or component (2) exercises all the software statements (program statements, branch directions, and logical paths through the program) [1].
- ***Test Phase*** It is the period of time in the software development life cycle during which the components of a software product are evaluated and integrated, and the software product is evaluated to determine whether or not requirements have been satisfied [1].
- ***Testability*** The degree to which a requirement is stated in terms that permit establishment of test criteria and performance of tests to determine whether those criteria have been met [1].
- ***Testing*** Testing is the process of exercising or evaluating software or a software component by manual or automated means to verify that it satisfies specified requirements or to identify differences between expected and actual results [2].

- ***Traceability*** The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another; for example, the degree to which the requirements and design of a given software component match. Presence of this quality criterion ensures that means are provided to verify whether the requirements are realized in design and code. Numbering individual requirements and referring to them by number in design documents and the source code is one way that helps in attaining traceability [1, 2].
- ***Unambiguous*** This quality criterion essentially demands that requirements of the target system should be such that they are clearly understood and there is no uncertainty involved. Once there is no conflict in interpretation of the requirements, the subsequent software development life cycle (SDLC) phases will be clear.
- ***Validation*** This refers to the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements. This basically entails system execution and test results comparison to performance requirements. The correctness of the test results therefore validates the specifications originally verified in the requirements phase [1].
- ***Variable*** An attribute or property of a person, group, or object that can be measured; the resulting measures are capable of assuming different values [3].
- ***Verification*** Verification refers to the process whereby each phase in the software development life cycle (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 concerns both the intrinsic correctness of the products of a software development phase, as well as the relation of each phase's products to the preceding and succeeding development phases [1, 5].
- ***Verification and Validation (V&V)*** This is a systems engineering process that employs a rigorous methodology for evaluating the correctness and quality of the software product through the software life cycle. V&V determines whether the requirements for a system or component are complete and correct, the products of each development phase fulfill the requirements or conditions imposed by the previous phase, and the final system or component complies with specified requirements. So, V&V involves comprehensive system analysis and testing during all phases of the development life cycle by a group either identical or closely linked to the development team [1, 5].
- ***Waterfall Model*** A model of the software development process in which the constituent activities, typically a concept phase, requirements phase, design phase, implementation phase, test phase, maintenance phase, are performed in that order, possibly with overlap but with little or no iteration [1].

NOTE: Most of the definitions provided in this glossary have been quoted from the following sources.

REFERENCES:

1. *An American National Standard IEEE Standard Glossary of Software Engineering Terminology*, (1983), ANSI/IEEE Standard 729

2. Frakes, W. B., et al, *Software Engineering in the UNIX/C Environment*. 1991, Englewood Cliffs, N.J.: Prentice Hall.

3. Katzer, J., et al, *Evaluating Information: A Guide for Users Of Social Science Research*, 3rd ed., 1991, New York McGraw-Hill, Inc.

4. Lewis, R. O. *Independent Verification and Validation: A Lifecycle Engineering Process for Quality Software.* Wiley Series in New Dimensions in Engineering, 1993, New York, NY: John Wiley & Sons, Inc.

5. Marciniak, J. J. (ed.-in-chief), *Encyclopedia of Software Engineering*, 1995, New York, NY: John Wiley & Sons, Inc.

6. NASA Headquarters, Office of Safety and Mission Assurance, *Software Engineering Evaluation System (SEES) Technical Assessment Procedures (TAPs),* U.S. Army Command, Redstone Arsenal, Alabama.

7. *Software Requirements Specification for the MDO Application: Aircraft Sizing,* CS5984/CS4984, Virginia Tech, BLACKSBURG, VA.

APPENDIX B: BIBLIOGRAPHY

Appendix: B

BIBLIOGRAPHY

Unless otherwise stated, the following bibliography was obtained by searching the Dialog database systems.

1. ANSI/IEEE Std. 730-1984, Standard for Software Quality Assurance Plans. New York, 1984: The Institute for Electrical and Electronics Engineer, Inc.

2. ANSI/IEEE Std. 1012-1986, Standard for Software Verification and Validation Plans. New York, NY, Nov. 1986: The Institute for Electrical and Electronics Engineer, Inc.

3. ANSI/IEEE Std. 1058-1987, Standard for Software Project Management. New York, NY, The Institute for Electrical and Electronics Engineer, Inc.

4. ANSI/IEEE Std. 1074-1990, Standard for Software Life Cycle Processes. New York, NY, 1990c: The Institute for Electrical and Electronics Engineer, Inc.

5. ANS/ANS-10.4-1987 Guidelines for the Verification and Validation of Scientific and Engineering Computer Programs for the Nuclear Industry, American Nuclear Society, La Grange Park, Ill., 1987.

6. (1983). An American National Standard IEEE Standard Glossary of Software Engineering Terminology . ANSI/IEEE Standard 729.

7. The National Bureau of Standards (NBS) (1984 May 28). NBS Establishes Software Guideline. Computerworld, v18(n22), p71.

   The National Bureau of Standards (NBS) thirty-seven page Guideline for Life Cycle Validation, Verification, and Testing of Computer Software presents a three-tiered approach for managing projects of varying degrees of complexity and importance. A representative plan and an example of the plan as applied to an insurance application is included in the report. It costs $8.50.

8. Ackerman, A. F., et al, (1989, May). Software Inspections: An Effective Verification Process. IEEE Software, v6(n3), p31(6).

   For software verification and validation purposes, inspections have been found to be superior to reviews and walkthroughs. Inspections rival testing in defect detection but cost less time and effort. Inspections involve the interplay of five elements: the product being inspected, the formal collection of process and product data, six well-defined steps (planning, overview, preparation, meeting, rework and follow-up), four well-defined roles (moderator, recorder, reader and producer) and a supporting infrastructure. Inspections do not replace testing, they only reduce the cost and schedule of testing. Of 15 software-development organizations surveyed that use inspections, most said they found them to have significant benefit.

9. Bianco, L. J. (1990). IVV&T, Rapid Prototyping Make Powerful System Development Team. (Independent Verification, Validation, and Testing). Federal Computer Week, 4(14), 27(1).

   Independent Verification, Validation, and Testing (IVV&T) programs can contribute to the success of rapid prototyping. IVV&T management plans should grow incrementally with program developments, and senior analysts should be able to communicate with management and perform IVV&T roles. Classical budget overrun situations can be reduced by maximizing the number of discrepancies discovered during IVV&T, and acceptance of a final software release can depend on how effectively IVV&T has been used to reduce the divergence between what is needed and what is developed.

10. Blosiu, J. O. (1983, Feb. 9). Independent Verification and Validation of Computer Software: Methodology, National Aeronautics and Space Administration, Jet Propulsion Laboratory, Pasadena, Calif., JPL D-576.

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

12. Boehm, B. W., and Papaccio, P.N., (1988, Oct.). Understanding and Controlling Software Costs. IEEE Transactions on Software Engineering.

13. Bruce, J. (1990, Feb.). Verification, Validation and Testing. (debugging lines of program code) (includes a related article on the 'cleanroom' method of system development). EXE, 4(8), 42(4).

   The industry standard ratio of 50 errors per thousand lines of code can be reduced by a factor of 100, and application of the verification and validation techniques discussed here will help improve the reliability and reduce the maintenance costs of software. Testing entails the process of verification and validation in addition to program execution; the four categories of techniques available are manual methods, static analysis, dynamic analysis and functional testing. Four common manual methods are walk-through, review, inspection and audit, each of which is described. Also discussed are code audits and how to conduct meetings involving inspections and audits.

14. Davis, D. B. (1990, Oct. 15). IBM Strives for Error-free Shuttle Software. (the software development process employed by the firm's Federal Sector Div). Electronic Business, v16(n19), p150(1).

   IBM’s Federal Sector Division's (Houston, TX) software-development process produces 'virtually fault-free code' for space shuttle on-board software systems. The firm's contract with NASA mandates error-free software, as human lives depend on software performance. No highly complex software can be completely free of bugs, but the shuttle software has been certified for no more than 0.1 to 0.2 errors per 1,000 lines of code. The division's development process consists of a requirements evaluation, software development, testing of software modules separately then in combination (including running the software on exact replicas of the on-board computers, and verification and validation by an independent testing group within IBM.

15. DOD-STD-2167A (1988, Feb. 29), Military Standard Defense System Software Development, AMSC No. 4327, Department of Defense, Washington, D.C.

16. DOD-STD-2168(1988, April), Military Standard Defense System Software Quality Program, AMSC No. A4389, Department of Defense, Washington, D.C.

17. Deutsch, M. S. (1982). Software Verification and Validation . Englewood Cliffs, N.J.: Prentice-Hall.

18. Duke, E. L. (1989, May). V&V of Flight and Mission-Critical Software. (software verification and validation at NASA facility). IEEE Software, v6(n3), p39(7).

    Staff at the NASA Ames-Dryden Research Facility, which is responsible for flight safety on research aircraft at the facility, has adapted software verification and validation (V&V) to meet its special needs. Ames-Dryden V&V methodology relies on testing, peer review, abstract models, simulations and validations by actual flights. Ames-Dryden researchers have found that a key element of V&V is the need to provide a culture in which verification, qualification and validation are responsibilities shared by each staff member. While a general outline of the V&V process can be established, the details of V&V must be adapted to each application and informed engineering judgment must be exercised throughout the V&V process.

19. Dunham, J. R. (1989, May). V&V in the Next Decade. (software verification and validation). IEEE Software, v6(n3), 47(7).

    Better software verification and validation (V&V) tools will be needed as engineers strive to improve software productivity and quality. Changes in seven basic categories will drive the development of tomorrow's V&V technology. An aging software base will need reverse-engineering and updating to remain useful. Greater consumer awareness will cause developers to pay more attention to product- liability and consumer-protection factors. Complex applications will require the incorporation of V&V early in the product cycle. High-risk applications will require more extensive, redundant and complementary V&V activities. The reuse of documentation, designs, code, test plans and test results will reduce some V&V activities, but new techniques will be needed to ascertain the adequacy of the initial V&V effort. V&V activities will have to account for prototyping. Increased productivity among software developers will require the streamlining of V&V activities.

20. Ervin, D. G. (1986, Oct. 24). GSA Study Yields Lessons in Verification. Government Computer News, v5(n21), p72(2).

    Verification and validation (V&V) of software is necessary in order to have efficient automated data processing (ADP). V&V allows thorough technical review and evaluation of system productive output. The three types of V&V are usability, quality assurance (QA), and functional. V&V should begin with a draft product and proceed in a structured manner through the system life cycle.

21. Fagan, M. E. (1976). Design and Code Inspections to Reduce Errors in Program Development. IBM Systems Journal, v15(n3), pp. 744-751.

22. Fagan, M. E. (1986). Advances in Software Inspections. IEEE Transactions on Software Engineering, vSE-12(7), p744-751.

23. Frakes, W. B., et al (1991). Software Engineering in the UNIX/C Environment . Englewood Cliffs, N.J.: Prentice Hall.

24. Frakes, W. B., Arthur, J., Cannon, M., & Gupta, S. (1995). An Experimental Design for Evaluating SEES (NAS1-19610, No. Task 17). Virginia Tech, Falls Church, VA.

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

    This section of the encyclopedia provides a definition of Independent Verification and Validation (IV&V), the different forms of IV&V, history of the usage of IV&V, and the activities performed by IV&V.

26. Gilb, T. (1984, Jan.). The Tom Gilb Column: 'Inspection' Is for Manager, not Programmers. Computer Management, p47.

    Michael E. Fagan's 'Inspection' method improves productivity in the areas of systems development, programming, and maintenance. Fagan was awarded a $50, 000 Outstanding Contribution Award by IBM for the method. Inspection is a kind of proofreading for documents at different levels of design. The method has provided IBM with a reduction in errors of two orders of magnitude and a fifteen to thirty-five per cent increase in productivity. A graph showing defect trends during product development is included.

27. Gruman, G. (1992). Under Pressure, NASA Renews IV&V Contract (space shuttle software independent validation and verification). IEEE Software, v9(n6), 106 (2).

    NASA announced on Sept. 11, 1992, that it would renew Intermetric Inc.'s $3.2 million independent validation-and-verification (IV&V) software contract. The software performs IV&V on the space shuttles' flight-control programs. NASA had wished to discontinue the IV&V contract since it believed that the three-to-four percent of the software budget the contract consumed could not be justified. The main flight-software providers all include their own IV&V systems. However, the program was strongly endorsed by the House of Representatives Committee on Science, Space and Technology; the General Accounting Office; and the National Research Council's Aeronautics and Space Engineering Board. The agencies cited the 15 Severity 1 errors, which are potentially life threatening errors, the system had detected. They urged NASA to retain the program to avoid the possibly huge cost if such an error went undetected.

28. Humphrey, W. S. &. S., W.L (1987). A Method for Assessing the Software Engineering Capability of Contractors. CMU/SEI-87-TR-23. Pittsburgh: Software Engineering Institute, Carnegie-Mellon University.

29. Katzer, J., et al (1991). Evaluating Information . New York: McGraw Hill.

30. Lewis, R. O. (1977). The Cost of an Error: A Retrospective Look at Safeguard Software . Huntsville, AL: SAIC.

31. Kernighan, B.W., and Ritchie, D.M. (1988). The C Programming Language. Englewood Cliffs, NJ: Prentice Hall.

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

Part I gives an overview of IV&V concepts; the software development lifecycle is fully explained, and IV&V is explained within the context of the software lifecycle. Part II describes current methods and techniques for verifying and validating the software components of engineering designs and systems, including a survey of CASE tools, and requirements, design, and code verification. Validation techniques, audits, and configuration management are also covered in this section. Part III contains a managerial guide to organizational planning and implementation for effective IV&V. In Part IV the wide range of IV&V applications are explored through eight case studies. (*Abstract obtained from* Cannon, M., *Independent Verification and Validation: An Overview*. NASA SEES Project, Phase I, Virginia Tech, August 1, 1994.)

33. Loesh, R. (1995). SEES Training Tapes. Software Engineering Sciences (SES).

34. Malone, B., and Mason, W. (1991). Aircraft Concept Development Using the Global Sensitivity Approach and Algebraic Technology Models. Blacksburg, VA: VPI-Aero-184.

35. Matras, J. R. (1993, May). Revised Standard Addresses Design Requirements for Computers Used in Nuclear Power-plant Safety Systems. Computer, v26(n5), p76(4).

An IEEE committee is creating standards addressing requirements for both software and hardware design of computers used in nuclear power-plant safety systems. System-level hardware requirements are laid out in IEEE Std 603-1991, but STD 603-1991 is ill-suited for situations when new hardware and software are added. The new standards will be ANSI/ANS 51.1 -1983 and will require rigid quality assurance programs for all activities associated with the design, modification and maintenance of nuclear facilities. Special addenda to NQA standards give guidance for qualifying computer hardware and have been expanded to include software. Verification and validation activities are also performed for both hardware and software. System qualification and system integrity are also addressed in a comprehensive manner.

36. McGarry, F. &. P., G. (1982, Sept.). Performance Evaluation of an Independent Software Verification and Integration Process . Greenbelt, MD: NASA Goddard.

This study evaluates the effectiveness of software IV&V on three flight dynamics projects at NASA Goddard's Software Engineering Laboratory. The Goddard projects ranged from 10K to 50K lines of code. IV&V was conducted on the requirements and design phases, with separate system testing. Each IV&V effort averaged 1.1 staff persons over 18 months. The results of this study indicated that IV&V was not very effective in the case of the three Goddard projects: (1) The IV&V interface led to the lowest development team productivity of any previously monitored Goddard SEL project (2) Error detection rates were better in the early part of the development cycle (3) For every thousand lines of code only 2.3 errors were detected (4) The cost rate to fix all discovered errors was no less than on non-IV&V projects at Goddard SEL (5) Software reliability appeared to be no different from non-IV&V Goddard SEL projects. (*Abstract obtained from* Cannon, M., *Independent Verification and Validation: An Overview*. NASA SEES Project, Phase I, Virginia Tech, August 1, 1994.)

37. Myers, W. (1994, July). Quality Week Stresses Competitive Pressures to Improve. IEEE Software, v11(n4), p112(2).

    The Seventh International Software Quality Week which attracted 636 participants focused on the need to maintain competitiveness and the positive effect of competition in terms of product quality improvement. One topic of discussion in the conference was the application of existing quality-assurance techniques to software development. One company used rigorous TQM methods alongside activities such as independent verification and validation. Another discussion topic was software testability methods. Likewise tackled was the increasing popularity of object-oriented technologies in information system shops. This technology has been considered as ideal for next-generation applications. Companies have been told to reengineer their host-based systems and slowly abandon COBOL in favor of the new technology.

38. NBS (1983). Guideline for Lifecycle Validation, Verification, and Testing of Computer Software. FIPSPUB101. Gaithersburg, MD: National Bureau of Standards (US).

39. NBS (1987). Guideline for Software Verification and Validation Plans FIPSPUB132. Gaithersburg, MD: National Bureau of Standards (US).

40. Quinn, T. (1990). IV&V: Different Aims for Different Phases. (independent verification and validation of computer systems and software). Federal Computer Week, v4(n13), p21(2).

    Independent verification and validation (IV&V) of new computer systems is seen differently by different people: some consider it line-by-line code comparison, while others assume it is limited to detailed evaluation of system documents or software development folders. IV&V skills and activities change with each stage of the system development process, and the value of IV&V benefits change with different development stages as well. Among the phases are requirements definition, development planning, system design, coding and unit testing, system integration and acceptance testing; IV&V tasks at each stage are described. Widespread use of IV&V in government will force those software developers who are currently reluctant to adopt modern software engineering procedures to improve the quality of their products and processes or risk losing government customers.

41. Radatz, J. W. (1981, June). Analysis of IV&V Data RADC-TR-81-145. NY: Logicon, INC.: Rome Air Development Center, Griffiss, AFB.

    This study evaluates the effectiveness of IV&V on four large real-time command and control projects at Rome Air Development Center (RADC). The projects, which included missile tracking and avionics systems, ranged from 90K to 176K lines of code. IV&V was started at the requirements phase for two of the projects, at the implementation phase for the third project, and at the testing phase for the fourth project. Each IV&V effort averaged 5 to 12 staff persons over 2.5 to 4 years. The results of the Radatz Study indicated that IV&V could be very effective in some cases. Radatz found that: (1) Development team productivity increased with IV&V; more programmer hours were saved by early detection than spent on reworking code (2) Error detection rates were better in the early part of development cycle (3) 5.5 errors per thousand lines of code were detected (4) A large number of discrepancies were reported, over 300 per program on average (5) Over 85% of errors affected software reliability (6) For projects with IV&V beginning after the requirements phase, early error detection savings were only 20-28% of IV&V costs; savings were 92-180% of IV&V costs for projects with IV&V beginning at the requirements phase. *(Abstract obtained from* Cannon, M., *Independent Verification and Validation: An Overview*. NASA SEES Project, Phase I, Virginia Tech, August 1, 1994.)

42. Rauch-Hindin, W. (1985, Feb.). IEEE Forms New Software Standards Groups. Systems & Software, v4(n2), p43-45.

    The IEEE Computer Society, in its continuing effort to bring standards to software, has formed four new software engineering standards working groups. The new groups have the task of developing standards for the software life cycle process, a software development plan, software maintenance guides and guides for software verification and validation. Five standards have already been approved by the IEEE and several others are under consideration. A table of approved standards and projects in process is included.

43. Taft, D. K. (1989, Oct. 16). NASA Contract for IV&V Goes to Intermetrics. (independent verification and validation). Government Computer News, 8(21), 49(2).

44. U.S. Army Missile Command (1994). Software Engineering Evaluation System (SEES) Technical Assessment Procedures. Redstone Arsenal, AL: Software Engineering Directorate.

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

46. U.S. Army Missile Command (1993b). SEES TAP Introductions SED-SES-TAP-TAPINTRO. Redstone Arsenal, AL: Software Engineering Directorate.

47. Wallace, D. R., & Fuji, R. U. (1989 May). Software Verification and Validation: An Overview. (the verification and validation method and standards to support it) (includes list of resources for V&V standards and guidelines). IEEE Software, v6(n3), p10(8).

    Verification and validation (V&V) comprehensively analyzes and tests software to ensure that it correctly performs its intended functions with no unintended functions. V&V also measures the software's quality and reliability. Verification calls for evaluating software during each life-cycle phase to guarantee that it meets the requirements set forth in the previous phase. Validation involves testing software or its specification at the end of development to be sure that it does what it is supposed to do. Ideally, V&V parallels software development. There are four ways to organize a V&V effort: as an independent group, or as a group embedded in the system-engineering group, quality-assurance group, or the user group. Among other benefits V&V can improve quality, instigate more stable requirements,

cause more rigorous development planning, and catch errors earlier. (*Abstract obtained from* Cannon, M., *Independent Verification and Validation: An Overview*. NASA SEES Project, Phase I, Virginia Tech, August 1, 1994.)

48. Wallace, D. R., and Roger U. (1989, May). Verification and Validation: Techniques to Assure Reliability. (overview of special issue on software validation). IEEE Software, v6(n3), p9(1).

Producing reliable software systems requires a well-planned, comprehensive application of different techniques throughout the development life cycle. These techniques are called software verification and validation (V&V). Far from acting as a drag on the development process, V&V is actually a partner with the developmental unit in a project's success. The purpose of this special issue is to explain V&V and show how personnel in development, quality assurance and other areas can use V&V techniques to produce high-quality software. Articles will explain V&V and how it fits in the development cycle, how to use reliability models to determine how much testing to do, how to perform software inspections, the use of V&V in flight-control systems, and possible trends for V&V in the 1990s.

49. Wallace, D. R. (1994). Verification & Validation. In J. Marciniak (Ed.), Encyclopedia of Software Engineering New York, NY: John Wiley and Sons.

In an attempt to gain some insight into what factors contribute to and detract from the effectiveness of IV&V, Wallace analyzes two studies evaluating the effectiveness of IV&V. The smaller of the two studies, [McGarry and Page, 1982], evaluates the effectiveness of software IV&V on three flight dynamics projects at NASA Goddard's Software Engineering Laboratory. The Goddard projects ranged from 10K to 50K lines of code. IV&V was conducted on the requirements and design phases, with separate system testing. Each IV&V effort averaged 1.1 staff persons over 18 months. The second study, [Radatz, 1981], evaluates the effectiveness of IV&V on four large real-time command and control projects at Rome Air Development Center (RADC). The projects, which included missile tracking and avionics systems, ranged from 90K to 176K lines of code. IV&V was started at the requirements phase for two of the projects, at the implementation phase for the third project, and at the testing phase for the fourth project. Each IV&V effort averaged 5 to 12 staff persons over 2.5 to 4 years.

The results of the McGarry and Page study indicated that IV&V was not very effective in the case of the three Goddard projects. On the other hand, the results of the Radatz Study indicated that IV&V could be very effective in some cases. Wallace argues that upon examination, one can conclude "that project parameters will affect the benefits of software V&V" [Wallace, V&V, 1994]. Based on the two studies, Wallace lists the following potential benefits of V&V on a software project: (1)Better quality and more stable requirements (2) More rigorous development planning; at least to interface with the V&V team (3) Better development team adherence to programming language, development standards, and configuration management (4) Early error detection and reduced false starts (5)Better schedule compliance (6) Better criteria for decision-making at formal reviews and audits (7) Greater project management visibility into interim technical quality and progress. Also based on the two studies, Wallace cites the following factors as potential negative effects of V&V on a software project: (1) Additional project cost of V&V: 10-30% extra (2) Additional staff time for V&V-development team interface (3) Additional documentation requirements, beyond the contract deliverables (4) Need to share computing facilities with V&V team (5) Lower development staff productivity if development team has to explain system to V&V analysts- particularly if not documented (6) Increased paperwork for written responses to error reports and other V&V data requirements (7)Lower productivity due to resolving invalid anomaly reports.

APPENDIX C: GROUP COMPOSITION

Appendix: C

GROUP COMPOSITION

OVERSIGHT PERSON: MARKUS GROENER

GROUP 1:	IV&V GROUP	DEVELOPMENT GROUP
	JIM BUSKIRK	DANIEL F. EDLEMAN
	BIN-MINH TRAN	JIM VENUTO
	DAVID R. COMPTON	VU NGUYEN
	BHABA R. MISRA	

GROUP 2: WAGE GROUP

ADAORA ONYIA

JOSEPH PATRICK VAN METRE

DAVID MARTIN

MEMBER ROLES

GROUP 1:

SEES does not designate a particular role to team members throughout the entire process. For the organizational purposes, Jim Buskirk was designated the team leader of the IV&V team.

As far as the design inspection aspect is concerned, the roles of moderator, recorder, reader, and inspector had been assigned by the IV&V team leader for the High Level Design (HLD) and the Lower Level Design(LLD). For HLD the roles were:

David Richard Compton - reader
Bhaba Misra - recorder
Binh-Minh Tran - inspector
Jim Buskirk - moderator

Note: Since the Fagan approach was used, Jim Venuto from the development team was also inspector , though he had little influence as prescribed by SEES.

For the LLD, the roles were:

Binh-Minh Tran - reader
Bhaba Misra - recorder
David Richard Compton - inspector
Jim Buskirk - moderator

For the code/unit test phase, the Computer Software Units (CSC) were divided among team members in order to inspect for proper coding practices and readability. The CSCs were also evaluated for understanding the adequacy of the unit level testing. For additional full IV&V, the code was verified with the LLD. Each person examined the code that corresponded to the LLD sections they had for the LLD inspection. This pattern was followed because each person was already familiar with that part of the design. The specific units to be tested completely were divided among the members. Jim Buskirk was the recorder for the SEES material and Binh-Minh Tran was the recorder for the other material. Bhaba Misra was the presenter.

For the Development team, Jim Venuto was the team leader. No special roles for the development team had been assigned - all of them worked together on every aspect. The work was formally divided among the team members, pretty much equally. Each person was involved in each phase of the process.

GROUP 2:

In group two, while no special roles had been assigned formally, work was divided as follows:

Joseph Patrick Van Metre - team leader and programmer
Adaora Onyia - oversight person for data collection
David Martin - programmer and tester .

OVERSIGHT PERSON:

Markus Groener was the oversight person for both the groups. For group 1, he made sure that all activity logs and duties were completed and questions were answered. For group 2, he made sure that all the activity logs and fault report forms were filled out, and collected these forms from Adaora Onyia. In addition, he answered group questions, ensured that weekly meetings were held, and that development effort was on track. Guidance was provided in a controlled manner so that group findings were not influenced.

MEMBER BACKGROUND

OVERSIGHT PERSON

NAME: Markus Groener
WORK EXPERIENCE: He has worked as an intern at Degussa Corporation, Alabama. Job responsibilities involved computer maintenance and writing of PC programs. Gained experience in Software Engineering methodologies. Also worked on a grant for QMS in Mobile, Alabama on a re-engineering project with the help of Teamwork tool
ACADEMIC BACKGROUND: Software Engineering (S.E.) classes; Master's thesis was in the area of S.E. - concepts of Participatory Design and Joint Application Design for opening channels between S.E. and clients when writing software requirements; Teacher's Assistant (T.A.) for special topics S.E. course.
TEAM PREFERENCE: NA
EMAIL ADDRESS: groener@csgrad.cs.vt.edu

GROUP 1: IV&V

NAME: Jim Buskirk
WORK EXPERIENCE: No real work experience
ACADEMIC BACKGROUND: No Software Engineering class; Independent Study IV&V Spring '95
TEAM PREFERENCE: None
EMAIL ADDRESS: jbuskirk@csugrad.cs.vt.edu

NAME: Binh-Minh Tran
WORK EXPERIENCE: She is working at NSWC-DD for AEGIS. Lately, her organization has been trying to establish and maintain their supporting software's documents
ACADEMIC BACKGROUND: Dr. Nance's Software Engineering course (CS 5704); Dr. Henry's Software Assessment course (CS 6704)
TEAM PREFERENCE: She did not have a clear preference of which team to be in.
EMAIL ADDRESS: bmtran@csgrad.cs.vt.edu

NAME: David Richard Compton
WORK EXPERIENCE: He has no work experience in Computer Science
ACADEMIC BACKGROUND: Dr. Henry's Software Assessment course (CS 6704) in Spring '95; B.S. in Forestry and Wildlife
TEAM PREFERENCE: None
EMAIL ADDRESS: rcompton@rcompton.csgrad.cs.vt.edu

NAME: Bhaba R. Misra
WORK EXPERIENCE: He has worked at Virginia Tech for a number of years and has used the state mandated SDLC (Software Development Cycle)
ACADEMIC BACKGROUND: Dr. Henry's undergraduate Software Engineering (CS 4704)
TEAM PREFERENCE: He preferred being in the IV&V group
EMAIL ADDRESS: brm@vt.edu

GROUP 1: DEVELOPMENT

NAME: Daniel F. Edleman
WORK EXPERIENCE: He has no prior Computer Science related work experience
ACADEMIC BACKGROUND: No Software Engineering courses taken; undergraduate Electrical Engineering department
TEAM PREFERENCE: He preferred the Development team
EMAIL ADDRESS: edleman@csugrad.cs.vt.edu

NAME: Jim Venuto
WORK EXPERIENCE: He has worked for four years as a Software Engineer at Pratt & Whitney, working on aircraft engine controls
ACADEMIC BACKGROUND: Dr. Henry's Software Assessment course (CS 6704) in Spring '95
TEAM PREFERENCE: He preferred the IV&V team, but either was ok
EMAIL ADDRESS: venuto@csgrad.cs.vt.edu

NAME: Vu Nguyen
WORK EXPERIENCE: Not provided
ACADEMIC BACKGROUND: Dr. Henry's undergraduate level Software Engineering class; some experience with V&V in Dr. Balci's & Simulation class, very basic however
TEAM PREFERENCE: He preferred the IV&V team
EMAIL ADDRESS: vn@csugrad.cs.vt.edu

GROUP 2: WAGE

NAME: Adaora Onyia
WORK EXPERIENCE:

- She had a research grant with FAA to help them in their restructuring program. The FAA wanted to change their computer system to the RISC system and wanted to know the impact this change would have on their organization. The research was divided into three areas namely: social impact, managerial impact, and technical

- She helped test software packages that had been built in-house and developed user's manuals for the software packages

ACADEMIC BACKGROUND: Software Engineering course taken (CS programming methods II)
TEAM PREFERENCE: Not applicable
EMAIL ADDRESS: onyia@csgrad.cs.vt.edu

NAME: Joseph Patrick Van Metre
WORK EXPERIENCE: **-** Not provided
ACADEMIC BACKGROUND: CS 4704 (undergraduate Software Engineering course); CS 6704 (advanced topics in Software Engineering process assessment)
TEAM PREFERENCE: Not applicable
EMAIL ADDRESS: vanmetre@csgrad.cs.vt.edu

NAME: David Martin
WORK EXPERIENCE: He has done some programming/consulting for an engineering firm (modifying some visual inspection code.) He has experience writing some large (>20k lines) projects on his own, and with hardware.
ACADEMIC BACKGROUND: No Software Engineering classes; Modeling & Simulation classes by Dr. Nance and Dr. Balci, where lifecycle is discussed.
TEAM PREFERENCE: Not applicable
EMAIL ADDRESS: martin@csgrad.cs.vt.edu

APPENDIX D: CORRESPONDING DATA COLLECTION FORMS BETWEEN SEES' AND SEES

Appendix: D

CORRESPONDING DATA COLLECTION FORMS BETWEEN SEES' AND SEES

SEES'	SEES
NA	Requirements Analysis Diagram (RAD) *RRATAP:1-1 NASA, TPATAP: 5-1 NASA, TDATAP: 6-1 NASA*
Defect Trouble Report (DTR) (different format and content)	Documentation Trouble Report (DTR) *RRATAP: 1-1 NASA, RTMTAP: 2-1 NASA, DRATAP: 3-1 NASA, CRATAP: 4-1 NASA, TPATAP: 5-1 NASA, TDATAP: 6-1 NASA, TWATAP: 7-1 NASA, FCATAP: 9-1 NASA, PCATAP: 10-1 NASA*
NA	SEES System Requirements List (SYRL) *RRATAP: 1-1 NASA, RTMTAP: 2-1 NASA*
SEES Software Requirements List (SRL) (a little different format and content)	SEES Software Requirements List (SRL) *RRATAP: 1-1 NASA, RTMTAP: 2-1 NASA, TPATAP: 5-1 NASA, TWATAP: 7-1 NASA, FCATAP: 9 -1 NASA*
NA	SEES Metrics Laboratory Metrics Report Form *RRATAP: 1-1 NASA, RTMTAP: 2-1 NASA, DRATAP: 3-1 NASA, CRATAP: 4-1 NASA, TPATAP: 5-1 NASA, TDATAP: 6-1 NASA, TWATAP: 7-1 NASA, FCATAP: 9-1 NASA, PCATAP: 10-1 NASA*
NA	Requirements Trace/Completeness Matrix (RTM) *RTMTAP: 2-1 NASA*
NA	SEES Inspection Announcement *DRATAP: 3-1 NASA*
SEES Inspection Summary Report (almost identical)	SEES Inspection Summary Report *DRATAP: 3-1 NASA*
SEES Inspection Defect List (IDL) (a little different format and content)	SEES Inspection Defect List (IDL) *DRATAP: 3-1 NASA*
SEES'	**SEES**

SEES Individual Preparation Log (IPL) (almost identical)	SEES Individual Preparation Log (IPL) *DRATAP: 3-1 NASA*
NA	Test Program Trace Matrix (TPTM) *TPATAP: 5-1 NASA, TDATAP: 6-1 NASA, TWATAP: 7-1 NASA*
NA	Test Description Analysis Form (TDAF) *TDATAP: 6-1 NASA, TWATAP: 7-1 NASA, FCATAP: 9-1 NASA*
NA	Test Witness and Assessment Form (TWAF) *TWATAP: 7-1 NASA, FCATAP: 9-1 NASA*
NA	Document and Standards List Form A (DSLF) *PCATAP: 10-1 NASA*
NA	Document and Standards List Form B (DSLF) *PCATAP: 10-1 NASA*
NA	PCA Analysis Form (PCAAF) *PCATAP: 10-1 NASA*
Activity Log	NA
Fault Report Form	NA

NOTE: For a copy of additional SEES forms not used in SEES', please refer to the Software Engineering Evaluation System (SEES) Technical Assessment Procedures (TAPS)

Defect Trouble Report (DTR)

[1] Program ID - DTR # ______________________________

[2] Current Status ______________________________

[3] Date: ______________________________

[4] Program Title ______________________________

[5] DTR Title ______________________________

[6] Defect Type: ❑ Critical ❑ Non-critical ❑ Nominal (but noteworthy)

[7] Defect Category (Check all that apply):

❑ Ambiguous ❑ Incomplete ❑ Inconsistent

❑ Not Traceable ❑ Not Verifiable (Testable)

[8] Problem Appears In:

____

____

____

[9] Problem Description:

____

____

____

____

____

____

____

____

____

____

____

[10] Comments:

____

____

____

____

____

____

[11] Reviewer Name: ______________________________

SEES Software Requirements List (SRL)

Program ID: ____________________ **Date:** ____________________

Req. Type	Req. ID #	Description	DTR #
❑ Functional ❑ Performance ❑ Interface ❑ Design/Impl. Constraint ❑ Inverse ❑ Process ❑ Derived			
❑ Functional ❑ Performance ❑ Interface ❑ Design/Impl. Constraint ❑ Inverse ❑ Process ❑ Derived			
❑ Functional ❑ Performance ❑ Interface ❑ Design/Impl. Constraint ❑ Inverse ❑ Process ❑ Derived			
❑ Functional ❑ Performance ❑ Interface ❑ Design/Impl. Constraint ❑ Inverse ❑ Process ❑ Derived			

SEES Inspection Summary Report

Program ID ____________________
Subsystem ____________________
CSU ____________________
____

Inspection Date ____________________
CSCI ____________________
Re-inspection: ❏ Yes ❏ No

Total hours expended in Person Hours (X.X)

Planning	Overview	Preparation	Inspection	Action Items	Summary Report		Total

Total Number of Major Defects: ____________
Total Number of Minor Defects: ____________
Open Items (number) ____________

Comment ____
____
____

Number of DTRs ____________
Re-inspection Recommendation ____
____
____

____
Moderator's Signature

SEES Inspection Defect List (IDL)

Program ID ______________________ Recorder ______________________

Subsystem ______________________ Moderator ______________________

CSCI ______________________ Date ______________________

CSU ______________________

Category	Type	Defect Description	DTR #

Page 1 of ____

SEES Inspection Defect List (IDL) (cont'd)

Program ID ______________________________

Category	Type	Defect Description	DTR #

Page ___ of ___

SEES Individual Preparation Log (IPL)

Program ID ____________________ Date Received Inspection Package ____________

Subsystem ____________________ Inspector ____________________

CSCI ____________________ E-mail Address ____________________

CSU ____________________ Completion Date ____________________

____________________ Inspection Date ____________________

Preparation Log:	Date	Time Expended
	____________	____________
	____________	____________
	____________	____________
	____________	____________
	Total Hours	____________

Defects / Clarity Problems / Concerns

Location	Description
____________	____________
____________	____________
____________	____________
____________	____________
____________	____________

Page 1 of _____

ACTIVITY LOG

Name: ______________________________

start time ______________________ Date: ____________________

end time ______________________

activity ____

phase ____

start time ______________________ Date: ____________________

end time ______________________

activity ____

phase ____

start time ______________________ Date: ____________________

end time ______________________

activity ____

phase ____

start time ______________________ Date: ____________________

end time ______________________

activity ____

phase ____

start time ______________________ Date: ____________________

end time ______________________

activity ____ phase ____

FAULT REPORT FORM

Name: ________________________ Date: ____________

Description of fault:

____

____

category:

___ requirements

___ interface design

___ high-level design

___ low-level design

___ code

severity:

___ critical (system down)

___ severe (user/application down)

___ normal (functional/operational)

___ inconvenience (feature, documentation error)

____

Phase in which the fault was found:

____

Phase in which the fault was introduced:

____

FAULT REPORT FORM
(Continued)

Time taken to fix fault:

____

____

____

Error Type(s):

___ original error

___ cascaded error

___ error of omission

___ error of comission

___ logic

___ other

____

APPENDIX E:
SOFTWARE REQUIREMENTS SPECIFICATIONS

Appendix: E

SOFTWARE REQUIREMENTS SPECIFICATION

The Software Requirements Specification (SRS) document for the MDO was provided to Group 1 and Group 2 at the start of the experiment. The document was prepared by Dr. Arthur and Markus Groener with technical assistance provided by the Aeronautical Engineering Department at Virginia Tech.

Index:

1. Application Scope

The MDO application described below represents the vehicle through which we will explore the use of Independent Verification and Validation (IV&V) techniques applicable throughout the software development lifecycle. To promote additional relevance to NASA's interests, the selected MDO application reflects an aeronautical flavor. More specifically, the application computes performance, aerodynamic and weight characteristics based on design parameters such as wing area, cruise altitude, mach speed, and various weight factors. The computational process is iterative and stresses the convergence to an ideal aircraft weight.

As detailed later in this document, three distinct data sets are computed:

- Performance characteristics:
  - Takeoff distance, and
  - Landing distance,
- Aerodynamic characteristics:
  - Cruise lift coefficient,
  - Cruise drag coefficient, and
  - Rotational velocity,

and finally

- Weight characteristics:
  - Weight of the fuel in the cruise segment of the flight,
  - Weight of the fuel in the climb segment of the flight,
  - Wing weight,
  - Engine weight,
  - A "fixed weight" overhead, and
  - A cargo weight.

2. Computation Overview

The computational procedure for computing aircraft performance, aerodynamic and weight characteristics is a convergent one that employs an iterative process. An initial set of data ($DSet_0$) is provided and contains two classes of data items: (a) design parameters that remain fixed throughout the convergent process, and (b) a set of weights that change with each iterative computation. An initial data set, $DSet_0$, is used during the first set of computations (or iteration) and results in an initial estimate of performance, aerodynamic and weight characteristics. In turn, a subset of the newly computed weight estimates is used to update corresponding elements in $DSet_0$, yielding a new input data set ($DSet_1$) for the subsequent iteration. Hence, each successive iteration (a) computes a more precise (or refined) set of performance, aerodynamic and weight characteristics, and (b) provides refined weight measures for updating the data set to be used in the next iteration.

Effectively, for the i^{th} set of iterative computations $(1 \le i)$,

$$\{Performance\ Characteristics\}_i = f_{Perf}\left(DSet_{i-1}\right)$$

$$\{Aerodynamic\ Characteristics\}_i = f_{Aero}\left(DSet_{i-1}\right)$$

$$\{Weight\ Characteristics\}_i = f_{Wght}\left(DSet_{i-1}\right),$$

where

$$DSet_0 \equiv \left\{\langle Design\ Parameters\rangle_{Fixed}, \langle Weights\rangle_{Initial}\right\}$$

$$DSet_i \equiv \left\{\langle Design\ Parameters\rangle_{Fixed}, \langle Weights\rangle_{i-1}\right\} \text{ and }$$
$$\langle Weights\rangle_{i-1} \subset \langle Weight\ Characteristics\rangle_{i-1}$$

2.1 Performance Characteristics

Two performance characteristics are computed: *Takeoff distance* (S_{to}) and *Landing distance* (S_{ldg}). In computing the takeoff distance four components are considered: distance covered on the ground(S_g), during rotation (S_r), during transition (S_t) and during climbout (S_c). Landing distance is computed as the sum of two components: the air distance from an altitude of 50 feet to the touchdown point (S_{air}), and the subsequent ground roll (S_{lg}). The formula for computing S_{lg} factors in antiskid braking, thrust reversing, ground spoilers and speed brakes.

2.2 Aerodynamic Characteristics

Three aerodynamic characteristics are computed: *Cruise lift coefficient* (C_{lift}), *Rotation velocity* in knots ($V_{rot_{knots}}$) and the *Cruise drag coefficient* (C_{drag}). The cruise lift coefficient is computed as a function of the dynamic pressure for a given cruise altitude and the weight of the aircraft once cruise altitude has been attained. Rotation velocity is expressed as a function of the maximum cruise lift for the aircraft, the wing area and the total takeoff weight. The cruise drag coefficient is computed as a sum of three components:

the cruise drag coefficient under zero-lift conditions (C_{D_0}), the transonic wave drag coefficient ($C_{D_{wave}}$), and C_{D_i} - which is a coefficient reflecting the impact of the Oswald Efficiency factor on cruise lift coefficient.

2.3 Weight Characteristics

Within this application the aircraft gross takeoff weight, W_{to}, will play a key role in the iterative computational process. W_{to} is defined as the sum of six distinct components: The wing weight (W_{wing}), the fuel weight during the cruise segment (W_{fuel}), the fuel weight apportioned to the climb (ascent) segment of the flight (W_{fclm}), the combined weight for all engines (W_{eng_n}), a "fixed" weight component associated with the aircraft (W_{fixed}), and a fixed cargo weight (W_{cargo}). Among these weights, only the W_{eng_n} and W_{cargo} remain constant throughout all iterative computations. W_{fixed} and W_{fclm} are computed based on a percentage of the total gross weight (derived from the immediately preceding iteration). W_{fuel} is computed as a function of the weight of the aircraft during the cruise segment as well as the cruise lift and cruise drag coefficients. Like W_{to}, W_{fuel} and W_{fclm} both play crucial roles in the iterative process. More specifically, W_{to} and its two components, W_{fuel} and W_{fclm}, are precisely the three weight components used to update $DSet_{i-1}$, and thereby, forming the new set of input parameters, $Dset_i$,for the subsequent iterative computation.

3. Requirements

3.1 External Input Data Requirements

The MDO application shall read its input parameters from a single data input file containing the following six (6) logical subsections:

(1) Control Flags,
(2) Initial Points,
(3) Parametric Mach Number Setup
(4) Baseline Weights (Fixed / Fract / Comput),
(5) Propulsion Variables, and
(6) Aerodynamic/Geometric Variables.

Note: All data item values from the input file shall remain constant throughout the computational processes with exception to those items included in the *Baseline Weights* subsection.

The physical structure of the input file shall have the following properties. Each data item is placed on a separate line. That data item may be followed on the same line by descriptive text. One or more spaces separate the descriptive text from the data item. This descriptive text is ignored for computational purposes. There shall be no more than 80 characters of text, including the data item, per line.

27 data items must be present in the input data file. If too few or too many data items are detected, an appropriate error message is printed and the program terminates. Data items with stipulated range values shall be validated. An appropriate error message shall be printed on data item values found to be out of range, the program shall stop execution after displaying the error message.

The filename of the input file shall be “mdo.inp”. All characters of the filename shall be non-capitalized to allow for portability between Unix and PC machines.

To make it easier to refer to individual data items later in the requirements document, each data item in the following description of the logical subsections will have an abbreviated neunomic associated with it. A sample input file with the logical subsections present, as well as an actual physical sample input file that the application should be able to process are displayed in Appendicies B & C.

Data types indicated as double precision real shall be treat as the highest precision real data type available within the ANSI C language and platform the program is implemented on.

3.1.1 Subsection 1: Control Flag

The control flag subsection includes one control flag that should be changeable by the user through the input file and not hard wired in the application. This control flag may be used to indicate if certain outputs shall be produced by the application.

Data Item 1: Detailed Weight Convergence Print Flag (IPTDET), indicates if the program is to print intermediate aircraft characteristic values during the iterative (convergent) process. If this flag is set to one (1) output of all relevant data (described in the output section) shall be produced after each iteration. If set to zero (0) no intermediate values are printed.

Data Type: Integer

Data Range: 1 for on (print intermediate values), 0 for off (do not print intermediate values).

3.1.2 *Subsection 2: Parametric Mach Number Setup*

The data items processed by this application as well as the results produced depend to a large extend on the initial mach parameter provided. In this subsection an initial mach (speed) value is provided along with an incremental stepping value and a value for the number of mach increment iterations (counter). The combination of these three data items allows the application to produce several airplane sizings, starting with the initial mach value, and for mach values computed by adding the product of the current mach iteration and the incremental mach stepping value to the initial mach value.

The Parametric Mach Number Setup subsection has 3 data item values, each provided on a separate line.

Data Item 1: Number of Mach Iterations ($MInc_{cnt}$).
Data Type: Integer

Data Item 2: Initial Mach Value (Mach).
Data Type: Double Precision Real
Data Range: 0 - 1
Units: Dimensionless

Data Item 3: Mach Value Increment per Iteration (Inc_{mach}).
Data Type: Double Precision Real

3.1.3 *Subsection 3: Initial Points*

A set of seven (7) initial design parameters are required to perform the sizing of the aircraft.

Data Item 1: Aspect Ratio (AR)

Data Type: Double Precision Real
Data Range: 0 - 100,000
Units: Dimensionless

Data Item 2: Wing Area (S_W)

Data Type: Double Precision Real
Data Range: 10 - 100,000
Units: ft^2

Data Item 3:	Cruise Altitude (Height)
Data Type:	Double Precision Real
Data Range:	0 - 100,000
Units:	ft

Data Item 4:	Mid-Chord Wing Sweep ($Sweep_{deg}$)
Data Type:	Double Precision Real
Data Range:	0 - 85
Units:	degrees

Data Item 5:	Wing Thickness to Chord Ratio (TC)
Data Type:	Double Precision Real
Data Range:	0 - 1
Units:	Dimensionless

Data Item 6:	Wing Taper Ratio (TPR)
Data Type:	Double Precision Real
Data Range:	0 - 1
Units:	Dimensionless

Data Item 7:	Flight Range (Range)
Data Type:	Double Precision Real
Data Range:	0 - 100,000
Units:	nm

3.1.4 Subsection 4: Baseline Weights

The main objective of this application is to find the ideal gross takeoff weight of an aircraft. As described, this is performed by iteratively sizing several components of the aircraft to yield the best size of components to weight ratio. To start the initial sizing computations reference weights have to be provided as input. (**Note:** The choice of these reference weights is not totally arbitrary since a “wrong” choice may lead to an early exit from the application if it is impossible to size the aircraft for a given set of reference weights.) Within the first iteration of the sizing process it can be expected that new weights are calculated for the different weight components that differ greatly from the reference weights provided by the input file. **Note:** For each new mach value iteration the original reference weights will have to be used since the newly computed weights may not be usable for a new mach value.

Data Item 1: Reference Gross Takeoff Weight ($W_{to_{ref}}$), is used in the first iteration of the sizing computation. A new takeoff weight is computed with every iteration until the computed gross takeoff weights converge.

Data Type: Double Precision Real
Data Range: 0 - 1,000,000
Units: lb

Data Item 2: Reference Fuel Weight ($W_{fuel_{ref}}$), is also used for the first iterative computation. A fraction of fuel weight is allocated to takeoff and landing of the aircraft with the remainder used as cruise fuel weight. (Note: Landing fuel weight computation performed implicitly through formulas, not specifically identified in SRS)

Data Type: Double Precision Real
Data Range: 0 - 1,000,000
Units: lb

Data Item 3: Fixed Cargo Weight (W_{cargo}), is a fixed weight of cargo the aircraft should be able to carry.

Data Type: Double Precision Real
Data Range: 0 - 1,000,000
Units: lb

Data Item 4: Engine Weight per Engine (W_{eng}).

Data Type: Double Precision Real
Data Range: 0 - 1,000,000
Units: lb

Data Item 5: Climb Fuel Fraction (F_{clm}), indicates that fraction of fuel weight used to climb to the cruising altitude.

Data Type: Double Precision Real
Data Range: 0 - 1
Units: Dimensionless

Data Item 6: Aircraft Fixed Weight Fraction (C_{fix}), used to computed a fraction of the gross takeoff weight with every iteration.

Data Type: Double Precision Real
Data Range: 0 - 1
Units: Dimensionless

Data Item 7: Structural Load Factor for Wing Weight (N), is a constant used in computing the wing weight.

Data Type: Double Precision Real
Data Range: 1 - 10
Units: Dimensionless

3.1.5 Subsection 5: Propulsion Variables

The next set of parameters deals with the engines of the aircraft. The amount of fuel (fuel weight) required for an airplane to fly a given range is related to the amount of fuel burned by the engines, the thrust produced by the engines and the number of engines.

Data Item 1: Maximum Engine Thrust per Engine (T_{max}).

Data Type:	Double Precision Real
Data Range:	0 - 500,000
Units:	lb

Data Item 2: Cruise Fuel Consumption Fraction per Engine (sfc).

Data Type:	Double Precision Real
Data Range:	0 - 5
Units:	$\frac{lb}{\left(\frac{lb}{hr}\right)}$

Data Item 3: Number of Engines (N_{eng}).

Data Type:	Integer
Data Range:	0 - 100
Units:	None

3.1.6 Subsection 6: Aerodynamic/Geometric Variables

The aerodynamics and flying abilities of an aircraft are not only dependent on the weights of the aircraft, but also on those areas exposed to airflow. Aircraft engineers call the aircraft areas exposed to airflow the “wetted” area, thinking of all those areas that would get wet if the aircraft was immersed in water. In flight these areas generate either a drag or lift depending on their exposure to airflow. The following parameters are used in the computations of the aerodynamics including cruise lift and cruise drag.

Data Item 1: Maximum Lift Coefficient ($C_{L_{max}}$).

Data Type:	Double Precision Real
Data Range:	0 - 5
Units:	Dimensionless

Data Item 2: Oswald efficiency factor (E).

Data Type: Double Precision Real
Data Range: 0 - 1
Units: Dimensionless

Data Item 3: Fuselage Wetted Area (S_{fuse}).

Data Type: Double Precision Real
Data Range: 10 - 100,000
Units: ft^2

Data Item 4: Horizontal Tail Wetted Area (S_{tail}).

Data Type: Double Precision Real
Data Range: 10 - 100,000
Units: ft^2

Data Item 5: Vertical Tail Wetted Area (S_{vtail}).

Data Type: Double Precision Real
Data Range: 10 - 100,000
Units: ft^2

Data Item 6: Pod Wetted Area (S_{pod}).

Data Type: Double Precision Real
Data Range: 10 - 100,000
Units: ft^2

3.2 Terminating the Iterative Process

As implied in Sections 1 and 2, given a set of input parameters, the MDO application employs an iterative computational process in an attempt to converge to an ideal aircraft weight. During each iteration specific performance, aerodynamic and weight characteristics are recomputed based on input derived from the immediately preceding iterative computation.

The iterative process terminates when either of three (3) following conditions occur:

- 500 iterations have been performed $\left(\textit{effectively}, 1 \le i \le 500\right)$,
- $\left|W_{to_{i-1}} - W_{to_i}\right| \le 1.0 \times 10^{-7}$ *lbs.* (the criteria for convergence), or
- $W_{to_i} \ge 9{,}000{,}000$ *lbs.* (the maximum allowed aircraft gross takeoff weight).

3.3 Incorporating Mach (Speed) Increments

The convergent process assumes an initial set of *fixed* design parameters, and an initial estimate of the total weight of the aircraft (this weight changes, of course, as the iterative process is employed). Clearly, one can change any of the initial input parameters and re-invoke the application to determine additional ideal aircraft weights. What is often the case, however, is that one desires the ideal weight computation over a range of mach (speed) values. Consequently, the MDO application shall

- read an increment count ($MInc_{cnt}$), a mach value (*Mach*) and a mach increment value (Inc_{mach}) and and
- compute the ideal aircraft weight (through the iterative process) over the set of mach values:

  $$mach + \left(j \bullet Inc_{mach}\right), \quad 0 \le j \le MInc_{cnt},$$

  also

  $$mach + \left(j \bullet Inc_{mach}\right) \le 1 \ .$$

3.4 The Computational Process and Formulas.

The following subsections provide the mathematical formulas to compute three (3) sets of aircraft characteristics: performance, aerodynamic and weight. Each set of characteristics can be computed independent of each other. (**Note:** Close scrutiny will reveal that duplicate computations are often performed, and can be avoided if a proper design is implemented. We encourage such a design. Several duplicate computations are noted.) Although not expected, should the program detect a possibility for a division by zero error, an appropriate error message shall be generated, indicating as closely as possible the area in the program in which the error would have occurred. The program shall also detect possible negative values under square root functions (although not expected) and shall generate an appropriate error message, indicating as closely as possible the area in the program in which the error would have occurred. Overflow or underflow errors are not expected and shall not be handled.

We specifically note that total aircraft weight (W_{to}), the weight of the fuel during the climb (W_{fclm}) and cruise (W_{fuel}) segments <u>are recomputed during each iteration and then used as updated values in the immediately successive iterative set of computations</u>. Subsequently, for **iteration$_\mathbf{i}$** you will see W_{to}, W_{fclm} and W_{fuel} being referred to as

$W_{to_{i-1}}$, $W_{fclm_{i-1}}$ and $W_{fuel_{i-1}}$, indicating that the values being used for these items were computed during **iteration$_{\mathbf{i-1}}$**. For iteration$_1$ the following initial values will be used for W_{to_0}, W_{fuel_0} and W_{fclm_0}:

$$W_{to_0} \equiv W_{to_{ref}} \qquad (W_{to_{ref}} \textit{ is provided as input})$$
$$W_{fuel_0} \equiv W_{fuel_{ref}} \qquad (W_{fuel_{ref}} \textit{ is provided as input})$$
$$W_{fclm_0} \equiv F_{clm} \bullet W_{to_{ref}} \qquad (F_{clm} \textit{ and } W_{to_{ref}} \textit{ are provided as input}).$$

Please note further that, with the exception of W_{to} and W_{fuel} all computational data items shown in **bold face** are input parameters.

3.5 Computing the Performance Characteristics

S_{to} is the takeoff distance (S_{to} is a function of $W_{to_{i-1}}$)

S_{ldg} is the landing distance (S_{ldg} is a function of $W_{fuel_{i-1}}$ and $W_{to_{i-1}}$)

$$S_{to} = S_g + S_r + S_t + S_c$$

where S_g is the ground distance component
S_r is the rotational distance component
S_t is the transition distance component, and
S_c is the climbout distance component

$$S_g = \int_0^{V_{rot_{mph}}} \left\{ \frac{\boldsymbol{W_{to_{i-1}}}}{32.174} \bullet \frac{v}{Thrust - Drag_v - 0.06 \bullet \left(\boldsymbol{W_{to_{i-1}}} - Lift_v\right)} \right\} dv$$

$$V_{rot_{mph}} = 1.1 \bullet \sqrt{\frac{2.0 \bullet \boldsymbol{W_{to_{i-1}}}}{\boldsymbol{C_{L_{max}}} \bullet 0.00273 \bullet \boldsymbol{S_w}}}$$

$$Thrust = 0.95 \bullet \boldsymbol{N_{eng}} \bullet \boldsymbol{T_{max}}$$

$$Drag_v = C_{drag_{to}} \bullet 0.001365 \bullet v^2 \bullet \boldsymbol{S_w}$$

$$C_{drag_{to}} = C_{D_0} + \frac{\left(0.8 \bullet \boldsymbol{C_{L_{max}}}\right)^2}{\pi \bullet \boldsymbol{AR} \bullet \boldsymbol{E}}$$

$$C_{D_0} = \left(0.0032 \bullet \left(\left(\frac{S_{tot} - S_{wing}}{\boldsymbol{S_w}}\right) + \left(FF \bullet \frac{S_{wing}}{\boldsymbol{S_w}}\right)\right)\right) + 0.0045$$

$$S_{wing} = 1.8 \bullet \boldsymbol{S_w}$$

$$S_{tot} = S_{wing} + \boldsymbol{S_{fuse}} + \boldsymbol{S_{tail}} + \boldsymbol{S_{vtail}} + \boldsymbol{S_{pod}}$$

$$FF = 1.0 + (0.891 \bullet \boldsymbol{TC}) + 100.0 \bullet (0.495 \bullet \boldsymbol{TC})^4$$

$$\pi = \arccos(-1.0)$$

$$Lift_v = \left(0.8 \bullet \boldsymbol{C_{L_{max}}}\right) \bullet 0.001365 \bullet v^2 \bullet \boldsymbol{S_w}$$

$$S_r = 3.0 \bullet V_{rot_{mph}}$$

$$S_t = \begin{bmatrix} \sqrt{R^2 - (R - H_{tr})^2} & if\ H_{tr} \geq 50.0 \\ R \bullet \sin(\Gamma) & otherwise \end{bmatrix}$$

$$H_{tr} = R \bullet (1 - \cos(\Gamma))$$

$$R = 0.205 \bullet \left(V_{stall_t}\right)^2$$

$$V_{stall_t} = \sqrt{\frac{2.0 \bullet \boldsymbol{W_{to_{i-1}}}}{\boldsymbol{C_{L_{max}}} \bullet 0.00273 \bullet \boldsymbol{S_w}}}$$

$$\Gamma = \arcsin\left(\frac{Thrust - Drag_{V_{rot_{mph}}}}{\boldsymbol{W_{to_{i-1}}}}\right)$$

$$Thrust = 0.95 \bullet \boldsymbol{N_{eng}} \bullet \boldsymbol{T_{max}}$$

Note: $Drag_{V_{rot_{mph}}}$ is $Drag_V$ with $V = V_{rot_{mph}}$

$$S_c = \begin{bmatrix} 0 & if\ H_{tr} \geq 50.0 \\ \dfrac{50.0 - H_{tr}}{\tan(\Gamma)} & otherwise \end{bmatrix}$$

$$S_{ldg} = S_{air} + S_{\mathrm{lg}}$$

Where S_{air} is the air distance from the point over a 50ft obstacle to the touchdown point, and

S_{lg} is the ground roll distance

$$S_{air} = \frac{1.0}{0.1} \bullet \left(\frac{V_a^2 - V_{Td}^2}{64.348} + 50.0 \right)$$

$$V_a = 1.2 \bullet \left(V_{stall_{ldg}} \right)$$

$$V_{stall_{ldg}} = \sqrt{\frac{2.0 \bullet W_{ldg}}{\boldsymbol{C_{L_{max}}} \bullet 0.00273 \bullet \boldsymbol{S_w}}}$$

$$W_{ldg} = \boldsymbol{W_{to_{i-1}}} - \left(0.2 \bullet \boldsymbol{W_{fuel_{i-1}}} \right)$$

$$V_{Td} = V_a \bullet \sqrt{0.9}$$

$$S_{\mathrm{lg}} = \frac{V_{Td}^2}{38.6088}$$

Note: To compute the integral defining S_g, the programmer shall use Simpson's rule:

$$\int_a^b f(x)\,dx \cong \frac{b-a}{3n} \bullet \begin{pmatrix} f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \ldots \\ + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n) \end{pmatrix}$$

where *n*, the number of partitions, is 200.

3.6 Computing the Aerodynamics Characteristics

C_{lift} is the cruise lift coefficient (C$_{lift}$ is a function of $W_{to_{i-1}}$ and $W_{fclm_{i-1}}$).

$V_{rot_{knots}}$ is the rotational velocity ($V_{rot_{knots}}$ is a function of $W_{to_{i-1}}$).

C_{drag} is the cruise drag coefficient (C_{drag} is a function of $W_{to_{i-1}}$ and $W_{fclm_{i-1}}$).

$$C_{lift} = \frac{\boldsymbol{W_{to_{i-1}}} - \boldsymbol{W_{fclm_{i-1}}}}{DynamicPressure}$$

$$DynamicPressure = f\,(\boldsymbol{Height}, \boldsymbol{Mach}, \boldsymbol{S_w})$$ - Subroutine provided

$$V_{rot_{knots}} = V_{rot_{mph}} \left(\frac{3600.0}{6080.0} \right)$$

$$V_{rot_{mph}} = 1.1 \bullet \sqrt{\frac{2.0 \bullet \boldsymbol{W_{to_{i-1}}}}{\boldsymbol{C_{L_{max}}} \bullet 0.00273 \bullet \boldsymbol{S_w}}}$$

$$C_{drag} = C_{D_0} + C_{D_{wave}} + C_{D_i}$$

$$C_{D_i} = \frac{(\mathrm{C_{lift}})^2}{\pi \bullet \boldsymbol{AR} \bullet \boldsymbol{E}}$$

$$C_{D_{wave}} = \begin{bmatrix} \dfrac{20.0 \bullet (\boldsymbol{Mach} - Mach_{crit})^4}{\cos^3(Sweep_{rad})} & if\ \boldsymbol{Mach} \geq Mach_{crit} \\ 0 & otherwise \end{bmatrix}$$

$$Mach_{crit} = \frac{0.9}{\cos(Sweep_{rad})} - \frac{\boldsymbol{TC}}{\cos^2(Sweep_{rad})} - \frac{C_{lift}}{10.0 \bullet \cos^3(Sweep_{rad})} - \left(\frac{0.1}{80.0} \right)^{1/3}$$

$$Sweep_{rad} = \frac{\boldsymbol{Sweep_{deg}} \bullet \pi}{180.0}$$

$$\pi = \arccos(-1.0)$$

$$C_{D_0} = \left(0.0032 \bullet \left(\left(\frac{S_{tot} - S_{wing}}{\boldsymbol{S_w}} \right) + \left(FF \bullet \frac{S_{wing}}{\boldsymbol{S_w}} \right) \right) \right) + 0.0045$$

$$S_{tot} = S_{wing} + \boldsymbol{S_{fuse}} + \boldsymbol{S_{tail}} + \boldsymbol{S_{vtail}} + \boldsymbol{S_{pod}}$$

$$S_{wing} = 1.8 \bullet \boldsymbol{S_w}$$

$$FF = 1.0 + (0.891 \bullet \boldsymbol{TC}) + 100.0 \bullet (0.495 \bullet \boldsymbol{TC})^4$$

3.7 Computing the Weight Characteristics

$$W_{to_i} = W_{wing} + W_{fuel_i} + W_{eng_n} + W_{fixed} + W_{fclm_i} + W_{cargo}$$

where W_{to_i} is the total takeoff weight of the aircraft ,

W_{wing} is the wing weight,

W_{fuel_i} is the fuel weight,

W_{eng_n} is the combined weight for all engines

W_{fixed} is a fixed weight component associated with the aircraft

W_{fclm_i} is the weight of the fuel needed to climb to the cruise altitude (this is not included in W_{fuel})

W_{cargo} is a fixed cargo weight component.

Each weight component is computed as follows:

$$W_{wing} = 0.0051 \bullet \boldsymbol{S_w}^{0.649} \bullet S_{csw}^{0.1} \bullet \frac{\boldsymbol{AR}^{0.5}}{\boldsymbol{TC}^{0.4}} \bullet \frac{(\boldsymbol{N} \bullet \boldsymbol{W_{to_{i-1}}})^{0.557} \bullet (1.0 + \boldsymbol{TPR})^{0.1}}{\cos(Sweep_{rad})}$$

$$S_{csw} = 0.10 \bullet \boldsymbol{S_w}$$

$$Sweep_{rad} = \frac{\boldsymbol{Sweep_{deg}} \bullet \pi}{180.0}$$

$$\pi = \arccos(-1.0)$$

$$W_{fuel_i} = W_{cruise} - \left(\frac{W_{cruise}}{RE}\right)$$

$$W_{cruise} = W_{to_{i-1}} - W_{fclm_i}$$

$$RE = e^{\left(\frac{(\boldsymbol{Range} \bullet \boldsymbol{sfc})}{\left(V_{cruise_{knots}} \bullet LD\right)}\right)}$$

$$V_{cruise_{knots}} = V_{cruise_{mph}} \bullet \left(\frac{3600.0}{6080.0}\right)$$

$$V_{cruise_{mph}} = f\ (\boldsymbol{Height}, \boldsymbol{Mach})$$ Subroutine provided

$$LD = \frac{C_{lift}}{C_{drag}}$$

$$C_{lift} = \frac{\boldsymbol{W_{to_{i-1}}} - \boldsymbol{W_{fclm_{i-1}}}}{DynamicPressure}$$

$DynamicPressure = f\,(\boldsymbol{Height}, \boldsymbol{Mach}, \boldsymbol{S_w})$ - Subroutine provided

$$C_{drag} = C_{D_0} + C_{D_{wave}} + C_{D_i}$$

$$C_{D_i} = \frac{(\mathrm{C_{lift}})^2}{\pi \bullet \boldsymbol{AR} \bullet \boldsymbol{E}}$$

$$C_{D_{wave}} = \begin{bmatrix} \dfrac{20.0 \bullet (\boldsymbol{Mach} - Mach_{crit})^4}{\cos^3(Sweep_{rad})} & if\ \boldsymbol{Mach} \geq Mach_{crit} \\ 0 & otherwise \end{bmatrix}$$

$$Mach_{crit} = \frac{0.9}{\cos(Sweep_{rad})} - \frac{\boldsymbol{TC}}{\cos^2(Sweep_{rad})} - \frac{C_{lift}}{10.0 \bullet \cos^3(Sweep_{rad})} - \left(\frac{0.1}{80.0}\right)^{1/3}$$

$$Sweep_{rad} = \frac{\boldsymbol{Sweep_{deg}} \bullet \pi}{180.0}$$

$$\pi = \arccos(-1.0)$$

$$C_{D_0} = \left(0.0032 \bullet \left(\left(\frac{S_{tot} - S_{wing}}{\boldsymbol{S_w}}\right) + \left(FF \bullet \frac{S_{wing}}{\boldsymbol{S_w}}\right)\right)\right) + 0.0045$$

$$S_{tot} = S_{wing} + \boldsymbol{S_{fuse}} + \boldsymbol{S_{tail}} + \boldsymbol{S_{vtail}} + \boldsymbol{S_{pod}}$$

$$S_{wing} = 1.8 \bullet \boldsymbol{S_w}$$

$$FF = 1.0 + (0.891 \bullet \boldsymbol{TC}) + 100.0 \bullet (0.495 \bullet \boldsymbol{TC})^4$$

$$W_{eng_n} = \boldsymbol{N_{eng}} \bullet \boldsymbol{W_{eng}}$$

$$W_{fixed} = \boldsymbol{C_{fix} \bullet W_{to_{i-1}}}$$

$$W_{fclm_i} = \boldsymbol{F_{clm} \bullet W_{to_{i-1}}}$$

$W_{cargo} =$ ***Fixed weight***, given as input

3.8 Output Requirements

3.8.1 Printing the Data Sets

All output produced by the application shall be printed to stdout formatted in such manner that individual data items and their corresponding values can easily be identified and read. The exact format of the output is not specified here since it shall be decided on by the developer and shall be specified latest in the low-level design document of this application. It is up to the customer to approve the format when the design is completed. It shall also be possible to redirect output produced from stdout to a file. Any error messages produced shall be directed to stderr, if such is available on the implementation platform, else these shall be directed to stdout as well.

There are three parts of output that the application shall be able to produce, (a) data item values and possible descriptions read from the input data file, (b) performance, aerodynamics, and weight data sets computed at every iteration, (c) a final set of convergence data.

To be able to verify that the data item values are correctly read by the application the values and descriptions (if any) of each data item shall be printed immediately after they have been read. The printout produced may be a line by line reflection of the input data file.

If the detailed weight convergence print flag (IPTDET) is set to one (1), the data sets computed for performance, aerodynamics and weights shall be printed at every iteration. The data items to print are:

- takeoff distance (S_{to}),
- landing distance (S_{ldg}),
- cruise lift coefficient (C_{lift}),
- cruise drag coefficient (C_{drag}),
- rotational velocity ($V_{rot_{knots}}$),

- weight of fuel in cruise segment (W_{fuel}),
- weight of fuel in climb segment (W_{fclm}),
- wing weight (W_{wing}),
- engine weight (W_{eng_n}),
- fixed weight overhead (W_{fixed}),
- fixed cargo weight (W_{cargo}), and
- gross takeoff weight (W_{to}).

A simple header indicating the iteration number shall precede the data set outputs. Again, it is up to the developer to design the header and document this no later than the low-level design document. At this point it shall be up to the customer to approve the design. If the print flag is set to zero (0) output shall not be produced during the iterative computations.

A final data set shall be printed when the iterative computations are completed. This final data set includes the same twelve (12) parameters as listed in the above paragraph. A final header shall precede this output to indicate the final data set.

3.8.2 Handling Errors

Since the possibility for error exists output shall be produced indicating the kind of error incurred. Two errors are possible, (a) the maximum number of iterations has been reached without convergence, or (b) the gross takeoff weight computed in an iteration exceeded the maximum gross takeoff weight allowed. In both cases output shall be produced that indicates the type of error occurred followed by a final printed of the data set (the final data set described above) which should be indicative of the error.

3.8.3 Output for each Mach Iteration

The above description of outputs indicates the data sets to be printed for one mach value iteration. Every mach value iteration requires the aircraft sizing data sets to be printed. A header indicating the current mach value that the data set is computed and printed for shall precede every initial data set for that mach value. It is left to the developer(s) to specify the format of the header and report this format no later than with the high-level design document. The customer shall have the option to reject or accept the header format at that point.

3.8.4 Output Format

As mentioned above, the format of the data set output is left to the application designers. The output shall be easily readable which means that white space should be used as needed to indicate breaks in data set iterations as well as mach value iterations.

4. Coding Environment

This application shall be designed such that it can be implemented and executed on computers running the operating system OSF/1 version of Unix on a DEC Alpha platform or the MS-DOS or PC-DOS versions 6.0 or high higher on an IBM-compatible PC. Coding shall be done using the C programming language. To allow for portability between the two operating systems and machines running those, only standard C library calls shall be used in the implementation. The choice of program editor is left open to the coding team since this may vary across machines.

4.1 Executable File

The executable file of this application shall be named “mdo.exe”.

4.2 Library Interface

Two function calls have been mentioned in the computation of the weight characteristics, *DynamicPressure* and $V_{cruise_{mph}}$. Due to their complexity this two functions have been taken from an earlier implementation of this application, compiled and archived in a library. The original implementation of this application was done in FORTRAN 77 and therefore the functions were implemented as subroutines and not function calls. In coding this application the programmers need to know the library name where the procedures are located, the correct procedure names, and the parameters to pass to the procedures.

The FORTRAN library in which these two procedures are located is called “libmdo.a” and will be provided to the programmers. It shall be stored in the same directory in which the source and object code of the application will be stored. During compilation the compiler needs to be told which library to link with and where the library can be found. A sample command line for compilation is:

```
cc -o mytest mytest.c -Lsubdirectory -lmdo -lm
```

where

mytest is the executable to compile to,
mytest.c is the C source code file,
subdirectory is the subdirectory where the library is located
-lmdo tells the compiler the name of the library, and
-lm tells the compiler to also use the standard mathematical library.

The standard mathematical library is need since a call to calculate an exponent is included in the FORTRAN library which otherwise would not be resolved.

The C procedure call to the DynamicPressure subroutine in the library shall be:

```
dynamic_pressure (&Height, &Mach, &Sw, &DynamicPressure)
```

where

&Height is the address to Height,
&Mach is the address to Mach,
&S_w is the address to S_w, and
&DynamicPressure is the address to DynamicPressure.

The C procedure call to the $V_{cruise_{mph}}$ subroutine in the library shall be:

```
cruise_velocity (&Height, &Mach, &Vcruisemph)
```

where

&Height is the address to Height,
&Mach is the address to Mach,
&$V_{cruise_{mph}}$ is the address to $V_{cruise_{mph}}$.

Note: It is important that all parameters shall be call by reference, providing a pointer to the actual parameter memory location. The variables DynamicPressure and $V_{cruise_{mph}}$ are of data type **double precision real**.

Since C and FORTRAN differ in how the actual procedure names are resolved during compilation, mainly in adding an underscore (_) either at the end or the beginning of the procedure name, the following definitions shall be included in the C program:

```
#define dynamic_pressure dynamic_pressure_
#define cruise_velocity cruise_velocity_
```

4.3 Programming Style

Note: This section is to be regarded as a guideline only, not a requirement.

It is left to the designers and implementors of this application how modular the C code shall be. Good programming standards suggest the use of procedure and function calls to allow for a modular design and aid in testing and maintaining. Also, the use of global variables is discouraged since they only lend themselves to confusion and easy introduction of program errors. Parameter passing should be used whenever necessary and possible. An overall good program design shall allow for a quick, efficient and correct application implementation with good programming standards encouraged and supported by the design.

Appendix A - List of Symbols

AR	- Aspect Ratio
C_{drag}	- Cruise Drag Coefficient
C_{D0}	- Zero Lift Drag
C_{DWave}	- Transonic Wave Drag Coefficient
$C_{drag_{to}}$	- Drag Coefficient at Takeoff
C_{fix}	- Fixed Component Weight Multiplier Factor
C_{lift}	- Cruise Lift Coefficient
$C_{L_{max}}$	- Maximum Cruise Lift Coefficient
E	- Oswald Efficiency Factor
F_{clm}	- Fuel Used in Climb to Cruise Altitude Multiplier
ft	- feet
H	- Cruise Altitude
KA	- Wing Structural Technology Factor
lbs	- pounds
LD	- Lift to Drag Ratio
Mach	- Mach Number (Speed)
$Mach_{crit}$	- Critical Mach Number
N	- Ultimate Load Factor
N_{eng}	- Number of Engines
nm	- Nautical Miles
Range	- Flight Distance
RE	- Reynolds Number
S_{air}	- Air Distance in Landing Calculation
S_c	- Clibmout Distance for Takeoff
sfc	- Specific Fuel Consumption (lb/lb/hr)
S_{fuse}	- Wetted Area for Fuselage
S_g	- Ground Takeoff Distance
S_{ldg}	- Total Landing Distance
S_{lg}	- Landing Ground Roll
S_{pod}	- Pod Wetted Area
S_r	- Rotation Distance for Takeoff
S_t	- Transition Distance for Takeoff
S_{tail}	- Horizontal Tail Wetted Area
S_{to}	- Takeoff Distance

S_{tot}	- Total Wetted Area
S_{vtail}	- Vertical Tail Wetted Area
S_W	- Wing Area
Sweep	- Mid-chord Wing Sweep
S_{wing}	- Wetted Area for Wing
TC	- Wing Thickness Ratio
T_{max}	- Maximum Thrust per Engine
V_{cruise}	- Cruise Velocity
V_a	- Landing Approach Velocity
V_{rot}	- Rotation Velocity at Takeoff
V_{stall}	- Stall velocity
V_{Td}	- Landing Touchdown Velocity
W_{cargo}	- Fixed Cargo Weight
W_{cruise}	- Cruise Weight
W_{eng}	- Engine Weight
W_{fclm}	- Fuel Weight used in Climb Segment
W_{fix}	- Fixed Weight (computed as a fraction of W_{to_i})
W_{fuel}	- Fuel Weight used in Cruise Segment
W_{Ref}	- Initial Cruise Weight
W_{to}	- Takeoff Gross Weight
W_{Wing}	- Wing Weight
TPR	- Taper Ratio

Appendix B - Logical Input File Structure View

The following is a logical breakdown of the subsections included in the input data file. For simplicity reasons an actual input file has all subsection headers removed as can be seen in Appendix C.

```
----------------------------------------
*             CONTROL FLAGS             *
----------------------------------------
1          -> IPTDET, Detailed Weight Convergence Print Flag
----------------------------------------
*     PARAMETRIC MACH NUMBER SETUP      *
----------------------------------------
0          -> NJMAC, Number of Mach Iterations
0.6500     -> MACH, Initial Mach Value
0.0500     -> MSTEP, Mach Value Increment per Iteration
----------------------------------------
*             INITIAL POINTS            *
----------------------------------------
9.0        -> AR, Aspect Ratio
3800.0     -> SW, Wing Area
32000.0    -> H, Flight Altitude
1.3        -> SWEEP, Mid-Chord Wing Sweep
0.10       -> TC, Wing Thickness to Chord Ratio
0.30       -> TPR, Wing Taper Ratio
5000.0     -> Range, Flight Distance
----------------------------------------
* BASELINE WEIGHTS (FIXED/FRACT/COMPUT) *
----------------------------------------
580000.0   -> WTOREF, Reference Total Take-Off Weight
100000.0   -> WFUELRF, Reference Fuel Weight
150000.0   -> WCARGO, Fixed Cargo Weight
7500.0     -> WENG, Engine Weight Per Engine
0.0200     -> FCLM, Climb Fuel Fraction
0.200      -> CFIX, Aircraft Fixed Weight Fraction
4.5        -> N, Structural Load Factor for Wing Weight
----------------------------------------
*           PROPULSION VARIABLES        *
----------------------------------------
45000.00   -> TMAX, Maximum Thrust per Engine
0.640      -> SFC, Fuel Consumption Fraction per Engine
4          -> Number of Engines
----------------------------------------
*    AERODYNAMIC/GEOMETRIC VARIABLES    *
----------------------------------------
2.5        -> CLMAX, Maximum Cruise Lift Coefficient
0.85       -> E, Oswald e constant
10367.0    -> SFUSE, Fuselage Wetted Area
1428.0     -> STAIL, Horizontal Tail Wetted Area
800.0      -> SVTAIL, Vertical Tail Wetted Area
2412.0     -> Pod Wetted Area
```

Appendix C - Physical Input File Structure View

The filename of the input data file shall be “mdo.inp”. The following is a sample of an actual input data file that the application should be able to process.

```
1          -> IPTDET, Detailed Weight Convergence Print Flag
0          -> NJMAC, Number of Mach Iterations
0.6500     -> MACH, Initial Mach Value
0.0500     -> MSTEP, Mach Value Increment per Iteration
9.0        -> AR, Aspect Ratio
3800.0     -> SW, Wing Area
32000.0    -> H, Flight Altitude
1.3        -> SWEEP, Mid-Chord Wing Sweep
0.10       -> TC, Wing Thickness to Chord Ratio
0.30       -> TPR, Wing Taper Ratio
5000.0     -> Range, Flight Distance
580000.0   -> WTOREF, Reference Total Take-Off Weight
100000.0   -> WFUELRF, Reference Fuel Weight
150000.0   -> WCARGO, Fixed Cargo Weight
7500.0     -> WENG, Engine Weight Per Engine
0.0200     -> FCLM, Climb Fuel Fraction
0.200      -> CFIX, Aircraft Fixed Weight Fraction
4.5        -> N, Structural Load Factor for Wing Weight
45000.00   -> TMAX, Maximum Thrust per Engine
0.640      -> SFC, Fuel Consumption Fraction per Engine
4          -> Number of Engines
2.5        -> CLMAX, Maximum Cruise Lift Coefficient
0.85       -> E, Oswald e constant
10367.0    -> SFUSE, Fuselage Wetted Area
1428.0     -> STAIL, Horizontal Tail Wetted Area
800.0      -> SVTAIL, Vertical Tail Wetted Area
2412.0     -> Pod Wetted Area
```

APPENDIX F:
SUPPORTING DATA