Experimental Evaluation of the Cleanroom Software Development Method

Naagesh Oruganti
Miami University, commons-admin@lib.muohio.edu

MIAMI UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE & SYSTEMS ANALYSIS

TECHNICAL REPORT: MU-SEAS-CSA-1992-003

Experimental Evaluation of the Cleanroom Software Development Method

Naagesh Oruganti

School of Engineering & Applied Science | Oxford, Ohio 45056 | 513-529-5928

Experimental Evaluation of the Cleanroom Software Development Method

by

Naagesh Oruganti
Systems Analysis Department
Miami University
Oxford, Ohio 45056

Working Paper #92-003 March 1992

Experimental Evaluation of the Cleanroom Software Development Method

FINAL PROJECT REPORT

Pesented in Partial Fulfillment of the Requirements
for the Degree of
Master of Systems Analysis
in the
Graduate School of Miami University

BY

Naagesh Oruganti

Miami University, Spring 1992

Advisor:

Dr. James D. Kiper, Department of Systems Analysis

Reading Committee:

Dr. Alton F. Sanders, Department of Systems Analysis

Dr. John Bailer, Department of Mathematics and Statistics

Contents

1. Introduction:

The field of software engineering is evolving as various new techniques, languages, paradigms, process models, methodologies, metrics etc., are constantly being developed, used and improved. In the past, several empirical studies were conducted to experiment with various new ideas in software engineering in order to help us better understand, evaluate, predict, control and improve these ideas. Basili et al. review several of these studies and present a general framework for analyzing these experiments[25]. They emphasize the need of such experiments for advancement of software engineering discipline. They recommend that the experimental planning process should include a series of experiments for exploration, verification, and application of new ideas. They also implied that the results of empirical experiments in software engineering should be verified by a series of other related experiments.

Cleanroom methodology for software development is a relatively new concept. Currently researchers are attempting to study the benefits and implications of using the Cleanroom methodology. One such attempt was made at the University of Maryland (UM) a few years ago which empirically characterized various aspects of Cleanroom methodology[21]. The fact that this study was empirical motivates the need for verifying its results. Also, the fact that the methodology is new motivates the need for further experimentation. Therefore, this experiment has been replicated recently at Miami University (MU). The goal of this project is to analyze the data collected in this experiment to verify the results of the previous study.

2. The Cleanroom method

The intent of Cleanroom software development approach is to produce highly reliable software by integrating formal methods for specification and design, non-execution-based program development and statistically based independent testing. The Cleanroom approach imposes discipline on software development so that system correctness, with certifiable reliability, results from a coherent, readable design rather than from a reliance on execution-based testing[21].

Cleanroom software engineering is a team approach to incremental development of software under statistical quality control. The "Cleanroom" name comes from the semiconductor industry, where a physical cleanroom is maintained in an attempt to prevent introduction of defects during hardware fabrication[2]. The Cleanroom method focuses on error prevention during the development life cycle rather than accepting software errors as inevitable and focusing on error detection.

The following sub-sections will briefly describe the salient aspects of the Cleanroom method. In each section, we describe an idea or technique, mention certain popular, practical ways to incorporate these into the Cleanroom process, and finally, analyze and report the impact of each of these on the product, people and process.

2.1. Specification in the Cleanroom and use of Formal Methods

In the Cleanroom approach, software design and all subsequent validation of implementation are completely based on a requirements specification document ([14], Chapter 1). Therefore, Cleanroom emphasizes the generation of correct, unambiguous, consistent and complete requirements specification leading to a controlled approach to stabilizing requirements in the early phases of software development life cycle. This in turn facilitates a more complete, coherent and verifyable design, overall improvement of product quality and substantial savings in maintenance costs. It also helps management in establishing and discerning accountability between specifiers and developers.

In addition to the requirements specification document, an expected-usage or 'operational' profile (an empirical frequency distribution of inputs to the system during its normal operation) and a detailed software construction plan are generated during Cleanroom's specification phase. As described later, these together support statistical testing of incremental releases including regression testing of previous releases in a Cleanroom development life cycle.

Typically, requirements are specified in natural language. But, usage of a formal specification mechanism to facilitate accurate, unambiguous specifications is of critical importance in the Cleanroom method to verify the correctness of design and implementation. Dyer mentions box structuring techniques, formal specification languages (such as Z, VDM, etc.,) and problem specific grammars as acceptable formal methods ([14], Chapters 1, 4).

Carmen et.al., have successfully used Box Structure Specification and Design techniques to perform step-wise refinement of specifications and design simultaneously for an Automated Production Control Documentation System[2]. Selby et.al, suggested structured specifications and state machine models for generating specifications[21].

Cobb and Mills distinguish between external specification and internal specification[17]. The external specification is a comprehensive user's reference manual (it is not a tutorial introduction to the product). It includes information about system use (commands, menus, all possible stimuli and corresponding response, events etc.,) system environment (hardware and software platforms and people), application environment, performance guidelines etc. The internal specification is more mathematical. The internal specification defines the responses in terms of 'stimuli histories' to make the specification implementation-independent. The authors mention that external specification should be written in natural language but they seemed to suggest formal methods for internal specification.

2.2. Design in the Cleanroom method

Dyer recommends a formal design method based on structured programming theory to be used in the Cleanroom[15]. This method involves a systematic and step-wise refinement of the software requirements to construct software design whose correctness can be assessed and confirmed at each step. Linger and Mills report using a function-theoretic approach as a formal design methodology for developing COBOL/SF[16].

Cobb and Mills describe the design process and provide an algorithm for using box-structured techniques in [17]. Carmen et.al., have demonstrated the use of box-structured techniques for design in a Cleanroom scenario[2]. Their step-wise refinement approach for design can be briefly summarized as follows: Each product increment is designed in top-down fashion to represent system behavior in a hierarchy of black boxes, state boxes and clear boxes. The implementation-independent black-box view defines the responses in terms of stimuli histories. The data-driven state-box view begins to define implementation details by elaborating the black box functionality in terms of internal data structures. The process-driven clear-box is the procedural design and may involve decomposition of the function into new lower level black boxes.

As such, although new terminology is introduced, this approach itself does not seem to be much different from the traditional top-down design scheme. But the Cleanroom approach deviates from top-down approach in that it advocates rigorous functional correctness verification of each of the views at all levels against the requirements specification. This may involve construction of correctness proofs[13]. The Cleanroom method derives its unique advantages not only from the fact that the functional verification is facilitated by formal requirements specification, but also the design itself is a systematic, natural refinement of the specification.

Also, verification-based inspections provide independent confirmation of the design correctness[12]. Occasionally, this process may also help in validating requirements of the system to make them more accurate, complete and consistent, augmenting the rigor of formal specification.

It is believed, based on the evidence from past experience, that designs developed with the Cleanroom method exhibit characteristics such as logical simplicity (attributable to the verification during design refinement) and better incorporation of data abstraction and encapsulation([14], Section 2.3.2).

2.3. Development and implementation with formal verification and elimination of execution-based unit-level testing

In the Cleanroom methodology, software product increments are implemented by rigorous stepwise refinement of design units (i.e., the 'clear boxes', in case of box-structured paradigm) into executable code. Then, the developers construct formal functional verification arguments[7] and informal correctness proofs[15] to verify that the code is equivalent to the design and hence that it conforms to requirements specification. These correctness proofs (with mathematical/logical reasoning but not esoteric mathematical notation) and functional verification arguments are thoroughly reviewed along with the code itself in independent code-inspections[12]. In addition to formal verification and code-inspections, the Cleanroom method employs code-reading by step-wise abstraction[15] and group walkthroughs[6] to assert the correctness of the implementation. These techniques, collectively referred to as "off-line software review techniques", unlike their counterparts in execution-based testing, simultaneously perform the activities of defect detection and isolation.

It perhaps sounds very radical and impractical that the Cleanroom developers are prohibited from performing any testing at all, including unit-level and sub-system level testing. But, it is evident from the past experience with Cleanroom method that formal verification and structured code reviews can effectively replace unit-level testing and also that the techniques applied in the Cleanroom resulted in software with significantly higher quality, and increased the productivity of the people using the method ([17], [14] - Section 2.3). Also, from the empirical evaluations reported in the literature, code reading by step-wise abstraction is at least as cost effective in detecting faults as execution-based methods [17],[24],[18],[10].

The following observation made by Mills, Dyer and Linger, also brings up an interesting argument for increased use of formal mathematical verification instead of debugging: "we find that human verification is surprisingly synergistic with statistical testing -- that mathematical fallibility is very different from debugging fallibility and that errors of mathematical fallibility are much easier to discover in statistical testing than are errors of debugging fallibility"[8].

In the Cleanroom method, all software is placed under formal configuration control prior to its first execution. This, together with elimination of unit-level testing has an immediate psychological effect on the developers, which manifests as a significant improvement in their productivity and in the quality of their software. Knowledge that all execution errors will be given public scrutiny imposes discipline on software development and leads to more conservative, careful and confident designs which exactly match the requirements.

Unit verification by debugging (execution-based) works fine for small programs but it does not scale up to large, integrated systems because it often compromises the design's integrity by producing local correctness and global incorrectness[5]. Cobb and Mills theorize the reasons for the cost-effectiveness of formal non-execution based unit verification as follows: design errors are caught sooner and as a result are less costly to fix; the expenses of finding the subtle, cantankerous failures introduced by debugging and the expenses of building programs to permit unit testing (drivers and stubs) are eliminated; it takes less time[17]. Note that, if such tool support is available, the Cleanroom developers may make use of a syntax checking program.

2.4. The development process, planning, organization and executable system functional increments

The "Waterfall" method is frequently used to model software development process. In this model, the system development life cycle consists of a sequence of specification, design, implementation (includes unit and sub-system level testing), testing (integration and acceptance testing), installation and maintenance phases. For large software systems, incremental development of multiple releases with each release encompassing more functionality of the system is preferred. If the application is not well-defined and the requirements are still being developed or evolved, a spiral model of life cycle is adopted in which software design, implementation and test steps are iterated between releases.

As described in figure 1, the Cleanroom method adopts an incremental development model in which the information about mean-time to failure (MTTF) and other product quality aspects (obtained from independent test and diagnosis) of each of the executable product increments is fedback into the process for further improvement. Incremental development supports the top-down development strategy and defines a stepwise approach to constructing software.

In the Cleanroom approach, a specification team, a development team and a certification team together can help interleave the phases of specification, design, development and testing repetitively for the product increments. The specification team prepares and maintains the specification and specializes it for each development increment. The development team designs and implements the software. The certification team compiles, tests and derives MTTF statistics and other diagnostic feedback and certifies the software's correctness.

Initially, a construction plan consisting of a functional decomposition of the system into user-executable increments and a detailed delivery schedule comprising specific milestones is devised. The content of the increments and the timing of their release are carefully planned to deliver significant increases in software functionality with each increment, so that enough structure is provided for an orderly evolution of the system. As the past experience with Cleanroom confirms, this organization facilitates intellectual control over the process and monitoring of statistical quality of the product.

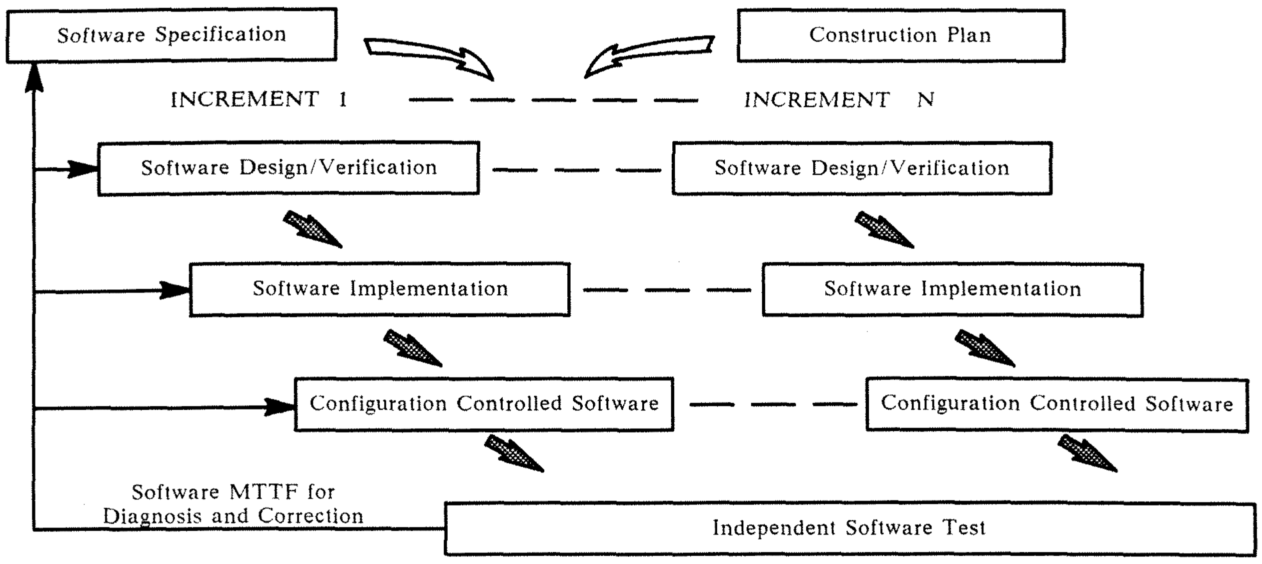
Software Specification
Construction Plan
INCREMENT 1
INCREMENT N
Software Design/Verification
Software Design/Verification
Software Implementation
Software Implementation
Configuration Controlled Software
Configuration Controlled Software
Software MTTF for Diagnosis and Correction
Independent Software Test

Figure 1. Cleanroom System Development Life Cycle
(*Source:* [14] - Chapter 1, modified)

Typically, a small-to medium sized Cleanroom product is divided into less than 6 product increments. Each increment is seldom more than 10,000 lines of code. In case of large systems, the number of increments is still kept low, to a dozen or so, but the increment size and the integration time period is increased.

2.5. Statistically based, independent testing and reliability prediction

Generally, software product testing involves two components: structural testing and functional testing. Structural testing is performed to ensure that the software correctly implements its design (which is primarily the responsibility of the developer). Functional testing is performed to validate that the software satisfies the specified requirements. As mentioned previously, it was demonstrated that the off-line review techniques used in conjunction with a formal design method in the Cleanroom approach satisfy all the goals defined for structural testing. The Cleanroom approach complements this by independent, statistical functional testing based on "operational or usage profile" to validate system performance against requirements specification.

The aim of statistical testing in the Cleanroom approach is to validate the requirements, generate diagnostics and derive MTTF data to enable the prediction of reliability of the system. An independent certification team of testers simulate the operational environment of the system with random testing from a perspective of reliability assessment rather than a perspective of error detection and location. The operational environment of the system is simulated by selecting test cases based on the "operational or usage profile", which is the frequency distribution of inputs to the system in conjunction with requirements specification[4]. The operational profile itself is obtained by user surveys and/or analysis of previous usage of similar systems.

In Cleanroom methodology, emphasis is placed on finding and eliminating software failures of execution, rather than software errors or faults. Musa et.al, define the terms error, fault and failure[9] as follows:

> **Failure** – is a departure of the external results of program operation from program requirements on a run. Note that the definition of failure is tied with execution of the program.
>
> **Fault** – is defined as a defective, missing, or extra instruction or set of related instructions that is the cause of one or more actual or potential failure types. Note that by definition there cannot be multiple faults causing a failure. The entire set of defective instructions that is causing the failure is considered to be the fault.
>
> **Error** – an error is a cause of one or more faults. Errors generally belong to one of the following four categories: communication, knowledge, incomplete analysis, or transcription.

It can be easily seen that the execution failures occur with varying frequency, depending on the usage profile. Coverage testing, which covers every path though the program, is as likely to find a rare execution failure as it is to find a frequent execution failure. But, Cleanroom's statistical testing, based on the usage profile, has a better chance to find more frequent failures. Since the goal is to predict a meaningful operational quality metric of the system such as MTTF, statistical, functional testing is proved to be more effective than structural coverage testing. This claim is supported by Cobb and Mills in their assertion that *statistical usage testing is 20 times more cost–effective in finding execution failures than coverage testing*[17].

Statistical testing in the Cleanroom approach relies on random sampling from a complete domain of possible test data which is superior to the arbitrary partitioning of data by testers. Also, the test material incorporates realism by considering probability distributions that describe the expected operating scenarios and hence make the reliability predictions derived from the results of this testing more meaningful and interpretable.

The particular statistical testing strategy recommended for use in the Cleanroom engineering is called statistical regression testing. In this technique, testing at each release considers (i.e. test cases are selected to test) all the previous releases, while concentrating on functions most recently delivered and maintaining the overall composite distribution of inputs. The MTTF measurements are made at each release of a software increment and product's projected reliability is predicted using reliability growth models and compared against a preset target to decide whether changes are required in software development to improve MTTF. The magnitude of discrepancy between the predicted and target reliability values dictates the nature of change warranted. For example, if the difference crosses a certain threshold, the increment will be redesigned rather than fixing the implementation. Musa presents a comprehensive discussion of reliability assessment and prediction by introducing various reliability growth models[9]. Other references to statistical models for reliability evaluations can be obtained from both [9] and [21].

In summary, the statistical testing process involves defining an operational profile, i.e., estimating the frequencies of usage of component functions and their groupings, randomly selecting or generating test data and test cases based on the operational profile, devising test schedules and an integrated test plan, conducting random testing on individual product increments, observing MTTF (and its dynamics) and predicting the projected reliability of the system.

It is conceivable that the usage profile may pronounce a very low frequency of usage of certain very important (perhaps privileged) functions of the system whose correct operation is very vital for satisfactory performance of the system, or whose failure might be catastrophic. In order to accommodate this fact of life, in addition to statistical functional testing, the independent testers in the Cleanroom also test these important functions of the system with a limited number of test cases to ensure correct system operation.

2.6. Statistical Quality Control of Cleanroom Software, Management Perspective

Although software behavior is deterministic, statistics can be used to make inferences about software quality because software use is stochastic. Cleanroom software engineering, when all the components of Cleanroom approach are applied to a project, is a practical process placing software development under statistical quality control.

For statistical quality control of software development, some basis for monitoring the software quality level must be defined. Unlike manufacturing, the basis can not be found in the large numbers of similar manufactured products, since software is an intangible product. Nor can it be found in the physical aging of the software statements, since the execution of these software statements is deterministic and always gives the same result for the same input. The basis must be found in the testing (which can be seen as trial usage) of the software product, with a perspective of obtaining information about the quality attributes of product and process. In hardware, physical dimensions and the statistical tolerances on physical parts are additive components and can be combined for considering the statistical quality of a complete product. This is not possible in case of a software product because of the deep and complex logical interactions between software parts.

Testing in the Cleanroom process has a statistical rather than selective basis. Statistical quality control of software in the Cleanroom is similar to modern manufacturing process control systems, in which, process outputs are sampled, process and product quality attributes are measured, quality trends are diagnosed and corrections to both product (such as fixing implementation or re–designing or even improving specification) and process (such as increased inspections and configuration control) are recommended.

As indicated previously, the Cleanroom process permits a sharper structuring of development work between specification, design and testing with clearer accountabilities for each part of the process. The MTTF estimations, based on user representative testing (obtained from the operational profile) provide both the development managers and users with a useful and readily interpretable product reliability and quality measure.

The Cleanroom life cycle of incremental product releases supports software testing and hence allows continuous assessment of product quality (based on predicted reliability) throughout the product development rather than only when it is completed. This is very helpful for management in gaining intellectual control over the development process and monitoring it for improvement in process and product quality attributes to achieve project goals within the resource constraint framework.

The Cleanroom process requires stable specifications as its basis. Because specifications are often not fully known or verified during initial development, it might appear at first glance that the Cleanroom approach may not be effective in developing new systems. But, in fact, the discipline imposed by the Cleanroom process is most useful in forcing specification deficiencies into the open very early during development and giving management control of the specification process[8].

2.7. Introduction of Cleanroom into software development life cycle (SDLC) and a development environment

The Cleanroom approach is not just a development technique, it is a comprehensive methodology in that it covers the complete system development life cycle. It is organized as a set of component techniques comprising software specification, software development, software correctness verification, independent software product testing, software reliability measurement, and statistical process control. These techniques are briefly described in the previous sections. Past experience indicates that Cleanroom component techniques can also be applied individually or in combinations for achieving significant development benefits. This is because each component addresses a specific aspect of the software development process, makes a separate contribution to development and, conceivably, has a unique set of conditions to be established before it could be introduced into the process.

If possible, the total Cleanroom process (including all the component techniques) should be used for software development to realize its full potential for enhancing product quality and process productivity. However, transitioning to a totally different development process is not always practical within an ongoing software development environment, where an incremental introduction of the Cleanroom components has proven to be a more effective strategy for technology transfer. Dyer discusses various possible strategies for technology transfer and their potential benefits and prerequisites[14].

2.8. Tool Use in the Cleanroom

Various tools can be used during system development to support the process of specification, design, implementation and testing. Cleanroom developers may make use of tools such as data flow analyzers, static analyzers, syntax-checkers, type checkers, formal verification checkers, concurrency analyzers, design and modeling aides. A Cleanroom test case generator, a verification-based syntax analyzer and a reliability analysis package are among the tools developed at IBM for providing tool support for Cleanroom software engineering. Also, tools to assist in defining operational profile, planning and conducting statistical testing (especially in obtaining MTTF data) would be very useful.

2.9. Summary of impacts of Cleanroom

The Cleanroom software engineering has been very effectively applied to various software projects in the past at IBM and NASA and others in the industry. The results include a significant overall improvement in product and process quality, development productivity and a better intellectual control over the process. The product quality is measured in terms of defect rate and reliability. The impact on development process is gauged based on the resources consumed including development time. The impact on development people is assessed by their productivity improvement and overall intellectual satisfaction achieved.

3. The framework of goals of the study

The goal of this project was to analyze the data collected in the experiment conducted at Miami to verify the results or conclusions of the previous study done at the University of Maryland. A broad classification of the goals of the project are as follows:

1 Characterize the effect of Cleanroom on the **delivered product**. This involves:

A. For intermediate and novice programmers building a small system, what were the operational properties of the product?

A.1. Did the product meet the system requirements?

A.2. How did the operational testing results compare with those of control groups?

B. What were the static properties of the product?

B.1. Were the size properties of the product any different from what would be observed in a traditional development?

B.2. Were the readability properties of the product any different?

B.3. Was the control complexity any different?

B.4. Was the data usage any different?

B.5. Was the implementation language used differently?

a. Comparative analysis of the static properties of the product, and
b. Comparative analysis of the degree of conformity of the product to the system requirements,

C. What contribution did programmer background have on the final product quality?

2 Characterize the effect of Cleanroom on the **software development process**.

A. For intermediate and novice programmers building a small system, what techniques were used to prepare the developing system for testing submissions?

B. What role did the computer play in development?

C. Did the developers meet their delivery schedule?

3 Characterize the effect of Cleanroom on the **developers.**

A. When intermediate and novice programmers built a small system, did the developer miss the satisfaction of executing their own programs? If so, did missing of program execution have any relationship to programmer background and did it have any effect on the quality aspects of the delivered product?

B. How was the design and coding style of the developers affected by not being able to test and debug?

C. Would the developers use Cleanroom again?

Obviously, these goals warrant a comparative study of the above mentioned properties of Cleanroom products with respect to those of the control groups for the experiment conducted at Miami and then another comparative study of these results with the results of the University of Maryland's study.

4. The experiment

This section introduces all the aspects of the experimental set up at Miami, outlines the discrepancies between the experiments of UM and MU, describes the data that were already collected by Dr. Kiper before beginning this master's project and comments on the Cleanroom techniques incorporated in both the experiments.

CLEANROOM GROUP

Attribute	Mean	Std Dev	Median	Minimum	Maximum	MW significance
Average number of semesters completed	6.33	0.94	7.00	5.00	7.00	0.53
Average number of 'major' computer projects participated	1.29	0.86	1.00	0.33	2.66	0.32
Average 'professional' experience of group members (in months)	3.56	0.57	3.33	3.00	4.35	0.33
Average overall GPA of group members	3.34	0.28	3.30	2.99	3.70	0.32
Average GPA of members in major (Systems Analysis)	3.32	0.41	3.50	2.70	3.70	0.69

CONTROL GROUP

Attribute	Mean	Std Dev	Median	Minimum	Maximum	MW significance
Average number of semesters completed	6.13	0.84	6.50	5.00	7.00	0.53
Average number of 'major' computer projects participated	1.66	0.33	1.66	1.33	2.00	0.32
Average 'professional' experience of group members (in months)	4.86	2.03	5.33	2.00	7.00	0.33
Average overall GPA of group members	3.10	0.37	3.12	2.70	3.62	0.32
Average GPA of group members in major (Systems Analysis)	3.14	0.38	3.02	2.80	3.75	0.69

Table 1. Summary of the pre-development experience survey by group

4.1. The experiment conducted at Miami

In order to describe the experimental set up at Miami, this section discusses various characteristics of the subjects of the experiment, briefly describes the product (Electronic Mail-User System) developed by the subjects and the hardware environment used for this to as the control group teams) used traditional development methodologies. The project was developed incrementally with preset milestones common for both the groups. Therefore, as was the case with UM's experiment, both Cleanroom and non-Cleanroom teams had the benefits of independent, incremental, functional testing. The two groups of teams were not statistically different in terms of professional experience, academic performance, or implementation language experience.

Just as in UM's experiment, the Cleanroom developers entered their source code on-line, used only a syntax checker while the non-Cleanroom teams used a complete VAX Ada compiler and were able to generate and execute code. The Cleanroom developers were able to perform automatic type checking across modules and they used the off-line review techniques of code reading (by step-wise abstraction), group walkthroughs and inspections. The non-Cleanroom teams were able to execute and debug their programs. They employed the techniques of modular design, top-down development, functional testing and informal design reviews.

4.2. Differences between the experiments at UM and MU

The intent of this study was to replicate the empirical study of UM and therefore, MU's experiment was modeled after that of UM and a good deal of effort was put into minimizing any qualitative differences.

1. The average professional experience of UM team members is significantly higher than that of MU team members.
2. Several Cleanroom developers of the University of Maryland have already utilized some of the ideas in the Cleanroom, especially off-line review techniques, whereas none of the Cleanroom developers at Miami had any such exposure.
3. In MU's experiment, both the Cleanroom and non-Cleanroom teams developed their systems using relatively new object-oriented principles and an object-based language Ada. Their peers at UM used structured design and coding techniques.
4. UM teams used the programming language Simple-T; MU teams used Ada.
5. At UM, the Cleanroom group teams and control group teams belonged to different courses to combat Hawthorne effect (described in Section 7), whereas, in MU's experiment, both Cleanroom and non-Cleanroom teams belonged to the same course.
6. In MU's experiment, although independent functional tests were performed on various product increments during development statistical testing was not. Statistical testing was performed on various product increments during development at UM.
7. There were no graduate students involved in MU's experiment.
8. In UM's experiment there are two teams with less than 3 members. In MU's experiment all the teams had 3 members in them.
9. The number of Cleanroom teams in UM's experiment was 10 and in MU's it is 5. The number of control group teams is the same though.

Table 2. Differences between the experiments at UM and MU

purpose and finally discusses the distinctions between the Cleanroom teams and non-Cleanroom teams.

4.1.1. Subjects

Subjects of the experiment consisted of 30 students (Seniors: 21, Juniors: 8, Sophomores: 1) of the course titled, "Object-Oriented Programming with Ada" taught by Dr. James D. Kiper in Spring 1988. Initially this course briefly discussed various principles and goals of software engineering and approaches to software development, introduced the Ada programming language and object oriented design and finally discussed more advanced features of Ada which facilitate object-oriented programming. The profile of the members of all the groups is summarized Table 1 and the summary of profile of all the teams is included in Appendix I .

The course also involved a group project which imparted experience with developing a relatively large system in teams using object-oriented programming principles. For the group project, individuals taking the course were grouped into three-person teams. An attempt was made to compose teams of uniform profile based on their professional experience, academic performance, and implementation language experience. If co-op experience is considered professional, the average professional experience of the participants of the course was 4.86 months. Most of the students did not have any significant prior experience with Ada programming language. The subjects of the University of Maryland experiment also did not have any significant prior experience with the programming language Simple-T, in which their systems were developed.

4.1.2. Project Developed

The project involved developing a software package for Electronic Mail-User System (EMS). The requirements specification document for the system was obtained from UM and was slightly modified by Dr.Kiper, (without affecting the comparability of results of both the studies). The systems were developed in VAX Ada in VMS Environment. The systems were expected to be completed in 6 weeks and to be about 1500 lines of Ada code.

4.1.3. Cleanroom Development Approach versus Traditional Approach

Ten 3-person teams developed versions of the same EMS software system. All the teams were required to use Ada as the development language. Five of the ten teams developed their systems using the Cleanroom approach, and the other five teams (referred

Table 2 accounts for all the perceived differences in the experimental set up of UM and MU and helps in identifying some of the intricacies of replicating experiments in software engineering. For example, the number of teams and the professional experience of the participants are among the constraints which are, to a large extent, beyond the scope of control that could be exercised in an academic environment. Therefore, it would be useful to analyze various parameters which could influence the replicability and their effect on the validity of comparative analysis of results prior to replicating an empirical experiment in software engineering.

4.3. The Cleanroom component techniques incorporated in the experiments

Both groups followed the incremental product development strategy. The Cleanroom groups in the experiments both at UM and at MU, did not used the techniques of formal specification and formal design. Both used code reviews but neither used vigorous formal or informal proofs of correctness. During implementation both used only off–line review techniques. In both the experiments, an operational profile was defined by conducting a survey of seasoned mail users (11 in UM's experiment and 10 in MU's experiment). Incidentally, although the participants of these independent surveys are totally different their results (i.e., the operational profiles) turned out to be very close to each other. This helps us gain more confidence in the truth value of the operational profile. In the experiments conducted both at UM and at MU reliability targets were not set and reliability predictions were not made at each incremental release.

4.4. The available data

The primary task of this project was to analyze data collected from the experiment conducted at Miami University. The data collected from the experiment are: a survey of experience of all the participant students, a survey of the end–users of the system, executables of final system releases and corresponding source files, reports of independent functional testing, specific grading reports for all the 10 teams, and a post–development attitude survey of the individual members of all the teams.

5. The approach

Statistical testing is an integral component of cleanroom methodology and it facilitates an objective and meaningful characterization of system reliability. Operational system properties of the final products of each team are statistically tested using the test data generated based on the input probability distribution. Performing statistical testing on the products developed by both Cleanroom and control group teams yielded two sets of statistics. These statistics were analyzed with the help of SAS[1] by comparing and correlating them using non–parametric statistical analysis techniques[2]. The same process was repeated for various static system properties also using an Ada based Metrics Analysis Tool called AdaMAT/D[3].

Finally, post–development attitude survey will be analyzed to characterize the effect of Cleanroom on the developers. In order to eliminate any bias during some subjective evaluations of operating or static properties of the systems, the knowledge about which of the teams belong to Cleanroom and which do not was not disclosed to the independent tester until the analysis phase. These results are then compared and correlated with those of the previous study at UM.

1. SAS is a registered trademark of SAS Institute.

2. Using non–parametric or rank procedures for statistical analysis is appropriate because we would be dealing with rank data (i.e., relative positions of various ordered data values) rather than interval data (which is based directly on the metric itself). Especially, the non–parametric tests are very appropriate if one is dealing with counting/percentage kind of data. Also, nonparametric tests apply even when the choice of a particular numerical scale of measurement is arbitrary, as long as the measurement process is consistent for all the observations. This is because they depend only on order relationships among the observations. Consequently, much less has to be assumed about the form of the underlying populations or about the metrics. The statistical analysis involves:
   i. Comparing various characteristic attributes of the two groups (Cleanroom group vs. control group). Mann–Whitney significance or Wilcoxon or U test is used for this purpose. The U test performed by using SAS gives various statistics. Among these, *T–Test approx. Significance* (commonly referred to as '**P**') is the most important. If P for a particular attribute is less than a certain threshold (usually less than 0.1), then it is inferred that the difference (in the values of this attribute) between the two groups is not **statistically significant**. The value of the MW statistic 'Z' itself doesn't have much semantic importance, but its sign determines whether the difference is positive or negative.
   ii. Identifying association or correlation between performance attributes and various other attributes within the Cleanroom group. Spearman Correlation strategy (which is general enough to match any kind of distribution, both linear and highly non–linear) was used for this purpose. Spearman coefficient is marked on a scale of 0 to 1, along with a significance level attached to it. A score of 1.0 for the coefficient with signif. = 0 is the best possible correlation (e.g. correlation of a variable with itself)

3. AdaMAT/D is a registered trademark of the Dynamics Research Corporation.

6. Data analysis and interpretation

In this section we describe the results of our experiment within the framework of goals outlined above. All the aspects of the experiment analyzed at the UM are analyzed with the data collected in MU's experiment. Each of the following sections begins with an outline of the approach (i.e., what exactly was performed), followed by a presentation of results, and finally a comparative analysis of Cleanroom and non–Cleanroom products. Each section is concluded by contrasting these results, conclusions, implications and inferences with those of University of Maryland.

6.1. Characterization of the effect on the developed product

The effect of Cleanroom component techniques employed in this experiment on the developed product is assessed by a comparative analysis of the implementation completeness characteristics, operational testing results and static systems properties of the final product releases of all the teams.

6.1.1. Implementation completeness

In order to characterize implementation completeness, the system functionality was divided based on the EMS requirements specification document into a set of logical sub–functions. The non–functional requirement that the system must not be case–sensitive is specified clearly in the requirements specification document, and hence is considered for implementation completeness characterization.

A set of test cases was created to test each of these logical sub–functions, thus covering (note that here, we are talking about functional coverage and not structural coverage) the whole functionality of the system. This set of test cases was used to perform functional testing of the completed system of each of the teams and the parts of their functionality conforming to specifications are recorded to characterize implementation completeness. Selby et. al., at the University of Maryland, devised a measure of implementation completeness as follows: the functional testing was performed with selected test cases; each function in an implementation was then assigned a value of two if it completely met its requirements, a value of one if it partially met them, or zero if it was inoperable. The total for each system is calculated and its ratio with the maximum possible points is used to characterize the percentage of implementation completeness.

Selby's work to characterize implementation completeness was repeated in our study too. However, there was some amount of subjectivity involved in evaluation of this metric for the systems, because the word "partially" is not specific and is somewhat ambiguous. Furthermore, a particular function may qualify to be partially implemented even if only a very small component of the total function is correctly implemented, thus accrediting more value to its completeness than it truly deserves. On the other hand, a function might be implemented almost completely but might be missing a small component which disqualifies it from being marked as complete. This results in assigning less credit to the system (for completeness) than it rightly deserves.

In order to avoid these problems in our study, a new metric for implementation completeness which is based on the requirements specification document is proposed. Each of the requirements (corresponding to the logical sub–functions of the system) specified in the requirements specification document is thoroughly analyzed to identify all the "atomic" components comprised by it. Each of these components is atomic in the sense that it can either be satisfied or not. If the component is satisfied by the system it is given one point; if not it is given a zero. This minimizes any subjectivity and imbalance of scores that was involved in computing Selby's metric. A given function may have many atomic components in it, and to make this metric comparable to Selby's the total score of points from individual components obtained for the function is divided by the maximum possible points (i.e. the number of components) and multiplied by 2.

The average percentage implementation completeness of Cleanroom projects turned out to be 64.45 using Selby's metric and 66.24 using our metric. The average percentage implementation completeness of control group projects is 80.55 using Selby's metric and 82.68 using our metric. This clearly indicates that the control group teams who employed the traditional approach met the requirements of the system more completely, although the difference is not statistically significant (MW = –1.253 and P = 0.2417 for Selby's metric and MW = –1.152 and P = 0.2788 for our metric). In the University of Maryland (UM) experiment, the average percentage implementation completeness of Cleanroom projects is 82.5 and that of the control group teams is 60.0 with MW significance being 0.088, when all the teams, including the worst performers, are included in the analysis. Table 3 summarizes the performance characteristics of the products developed by all the teams.

Among the Cleanroom teams, team "B" was the worst performer with respect to most of the product quality aspects and contributes negatively to the group's average statistics. This team consisted of all graduating seniors and the team's average professional experience and the average GPA were the lowest among all the teams. Also, this team had problems complying with the delivery schedule. By removing team B from the analysis, the average percentage implementation completeness for Cleanroom team projects is 74.31 using Selby's metric and 76.55 using our metric, which makes the performance of the Cleanroom projects closer but still inferior to that of control group teams. So, on the whole, the control group teams using traditional approach have delivered systems with more complete functionality.

CLEANROOM GROUP

Attribute	Mean	Std Dev	Median	Minimum	Maximum	MW significance
Percentage implementation completeness with Selby's metric (ICUM)	64.44	22.68	72.22	25.00	80.56	0.24
Percentage implementation completeness wit MU's metric (ICMU)	66.24	23.75	76.61	25.00	83.33	0.28
Percentage of successful test cases without duplicate failures (TWODUPS)	74.43	28.38	82.98	25.00	97.92	0.32
Percentage of successful test cases with duplicate failures fTWDUPS)	66.40	31.54	76.00	12.00	94.00	0.32
User friendliness index (UFI)	61.08	17.07	59.09	35.71	77.31	0.07

CONTROL GROUP

Attribute	Mean	Std Dev	Median	Minimum	Maximum	MW significance
Percentage implementation completeness with Selby's metric (ICUM)	80.55	18.94	88.88	50.00	97.22	0.24
Percentage implementation completeness wit MU's metric (ICMU)	82.67	19.42	93.50	51.14	97.22	0.28
Percentage of successful test cases without duplicate failures (TWODUPS)	91.22	8.49	93.62	77.50	98.00	0.32
Percentage of successful test cases with duplicate failures fTWDUPS)	84.40	13.81	88.00	62.00	98.00	0.32
User friendliness index (UFI)	85.26	13.54	86.57	67.47	103.60	0.07

Table 3. Summary of the performance characteristics classified by group

6.1.2. Operational testing results

In order to select test cases to perform statistical independent functional testing, an "operational profile" based on the survey conducted with 10 seasoned mail users at Miami University was created. An operational profile is a frequency distribution of inputs (and the usage of the corresponding logical functions) to the system. It is instrumental in identifying different paths in the system and the probability of their occurrence, which in turn, will form the basis for selecting test data. Then 50 user-session test cases were formulated with randomly sampled test data based on this operational profile. Some of these test cases overlap with the test cases used for checking implementation completeness.

Independent testing of the final system release of each team (as random user-sessions) was conducted using these test cases. During the testing process the data about the following aspects was recorded:

1. failures of the system - to compute the percentage of successful test cases
2. the CPU time between the failures - to compute the MTTF
3. total elapsed calender time and the elapsed time between failures
4. general subjective evaluation of the user interface (based on its consistency, ease of use and its informational and error messages, etc.)

A severity code, compatible with Selby's scheme of failure severity classification was assigned and recorded for each failure. According to this scheme, a code of 1 is assigned if the product is inoperable, 2, if a major function in the product is inoperable, 3, if some part of a major function is inoperable and 4 if it is a cosmetic type failure. This information can be used to analyze the failure data to see if there is a specific pattern or a set of types of errors that are generally committed by the Cleanroom groups. Also, this kind of a classification scheme can be used to discount certain class of failures (say, all the failures with a severity code of 4) for MTTF and the reliability computations. This information may also be helpful in characterizing the 'robustness' (a characterization of the behavior of the system when failures occur) of the system.

Also, each failure is marked as to whether or not it is a duplicate failure. A failure is called a duplicate failure if a similar failure was previously observed while testing a particular system. Duplicate failures are significant in a Cleanroom scenario for the following reasons: The Cleanroom developers did not perform unit-level testing, whereas the control group developers did. The control group members also performed sub-system testing as the product was developed with incremental functionality. Normally, any testing strategy emphasizes testing of the most commonly used and, very likely, the most important set of functions of the system. Thus, it is conceivable that the developer (who is also the tester) may develop an intuitive model of the operational profile of the system and use this model while designing the test cases. This may then help in eliminating errors in most commonly used parts of the systems developed by control group teams. These two aspects together with the fact that the operational testing is based on a well-developed operational profile make the identification of duplicate failures and discussion of their impact significant.

Table 3 summarizes the operational performance of the final products developed by all the teams. The control group teams had an average of 84.4 percent successful test cases when duplicate failures are included and and 91.22 percent when duplicates failures are not considered. The Cleanroom teams had an average of 66.4 percent with duplicate failures included and 74.43 percent without duplicate failures. Again, this difference is not statistically significant (in both cases, MW = -1.0445 and P = 0.3235), but nevertheless, control group teams performed better than Cleanroom teams.

In UM's experiment, the percentage of successful tests without duplicate failures for Cleanroom teams was 92.5 and for non-Cleanroom teams was 80.8 (P = 0.055). When the duplicate failures are included, however, the better performance of the Cleanroom systems was not nearly as significant (P = 0.134) in their experiment. This is essentially caused by a relatively higher proportion of duplicate failures in the Cleanroom systems in contrast with non-Cleanroom systems. More succinctly, the Cleanroom systems were less sensitive to the operational profile than the non-Cleanroom systems. It was theorized that the apparent less sensitivity to the usage profile was the result of a more uniform review of the whole system by Cleanroom developers and that the non-Cleanroom developers focused on a "perspective of the tester," which could possibly have been intuitively visualized by the non-Cleanroom developers during development.

What is noteworthy about the results of our experiment is that there is no difference in the MW significance levels for percentage of successful test cases with and without duplicate failures. That is, both the teams are equally sensitive/insensitive to the operational profile. If the statistical regression testing was performed properly on both Cleanroom and non-Cleanroom systems and if results were made available to the developers in both the groups, it seems reasonable to expect both the groups to take equal advantage of the information. This is because the results of statistical regression tests would, supposedly, give a good opportunity (as the project progresses through different incremental releases) to the Cleanroom developers as well to realize the relative importance of various logical sub-functions within the system and to focus development effort accordingly. In fact, this (theoretical) opportunity to improve the operational reliability of the system seems to be one of the potential benefits of statistical testing in the Cleanroom methodology along with providing an opportunity to obtain a meaningful measure of the operational reliability of the system. Note that it may not be possible to realize this benefit for the logical sub-functions implemented during the very last product increment.

In addition to characterizing implementation completeness and operational performance of Cleanroom and control group products, an organized attempt was made to observe various user-interface and other non-functional characteristics of the systems. Several aspects that are identified for evaluation are listed in Appendix II. A subjective impression of each of these aspects is formed by the independent testing team during this statistical testing. The impression on each of the aspects is then evaluated and scored (based on an arbitrary abstract scale described in Appendix II) by comparing it against corresponding predefined "expected behavior". These subjective impression scores on all of the observable aspects are then synthesized into an impression index called "user-friendliness-index" (UFI) which is computed by adding the individual scores and normalizing the sum by the number of valid or observable aspects or attributes.

Clearly, deriving UFI for the products in our experiment involved a lot of subjectivity but it can be argued that the metric is still useful to make a meaningful comparative characterization of the user-interface issues because, 1) the same person used the same subjective evaluations on products of all the teams, and 2) the evaluations could not have been biased towards either Cleanroom group products or control group products because the independent tester was not aware of information as to which teams belonged to Cleanroom and which belonged to control group until the operational testing phase was finished and the analysis phase commenced.

Table 3 also includes summary of the values of UFI obtained for both the Cleanroom and control group products (for more details please refer to Appendix I). The non-Cleanroom teams had a significantly higher average UFI (85.27) than the Cleanroom teams (61.08). It can be concluded from these results that non-Cleanroom teams developed systems which are more easily usable, testable and hence more user-friendly ($P = 0.066$, when all the teams are included in Mann-Whitney analysis). This trend was expected because the Cleanroom developers could not run and observe the 'look and feel' of their programs themselves.

6.1.3. Characterization of Static Properties of the System

In this sub-section, the size, complexity and data usage patterns of both the Cleanroom and non-Cleanroom developers are analyzed.

Source code analysis to collect statistics of the final systems of all the teams about the number of source lines, executable statements and comments was performed by using AdaMAT/D and DCL[4]. These static properties of the systems developed by Cleanroom groups are compared with those of the control groups.

In UM's experiment, Cleanroom projects possessed a higher percentage of assignment statements ($P = 0.056$) and had a lower complexity density ($P = 0.079$). Table 4 summarizes the information about number of logical source lines of code, percentages of executable statements, declarative statements, IFs, CASEs, WHILE loops, FOR loops, comments, assignment statements and other statistics of the systems developed at MU (a more detailed summary by team is included in Appendix I). The projects from the two groups of MU were not statistically different (MW significance > 0.20) in any of these attributes. In UM's experiment the developers using Cleanroom wrote code that was more highly commented ($P = 0.089$) and had a lower complexity density. In MU's experiment, even though the Cleanroom developers mentioned that they commented their code thoroughly, the non-Cleanroom teams had slightly higher average percentage of comments (non-Cleanroom average: 12.215%, Cleanroom average: 12.01%, $MW = -0.418$, $P = 0.686$). Percentage of executable statements and percentage of declarative statements of Cleanroom and control group products were different with noticeable statistical significance ($MW = -1.67$, $P = 0.129$ and $MW = 1.67$, $P = 0.129$ respectively).

Average number of occurrences of non-local data items (variables, constants etc.) is considered by Selby et. al., for studying data usage characteristics of the finished products. These statistics were used to present an intuitive discussion of their effect on modularity. AdaMAT/D does a much more thorough job of characterizing modularity by including information hiding statistics also.

Several AdaMAT/D metrics were obtained (summarized in Table 5 and Appendix I) in order to statically characterize product quality aspects such as complexity density,

4. DCL (DEC Command language) is the shell language of the VAX VMS operating system

modularity, maintainability, reliability, flow complexity and information hiding. The projects from the two groups at MU were not statistically different (MW signif. > 0.10) in any of these metrics also. The worst Cleanroom performer team 'B' was excluded and the statistical analysis was repeated. As expected, the difference between any of the attributes of Cleanroom and non-Cleanroom projects was not statistically different in this case either.

In UM's study the operational quality measures of just the Cleanroom products and of all the products are correlated with the usage of the implementation language. In that experiment, both percentage of successful test cases and implementation completeness correlated positively with the percentage of procedure calls and with percentage of IF statements and negatively with percentage of CASE statements, with percentage of WHILE statements. In the experiment at MU most of these correlations are not significant (significance > 0.10) when all the team products are considered.

Percentage of implementation completeness (ICMU), percentage of successful test cases with duplicate failures(TWDUPS), percentage of successful test cases without duplicate failures(TWODUPS) and user-friendliness index (UFI) characteristics of all the cleanroom teams are correlated with team averages of various experience parameters and also with various static properties and semantic metrics of the final Cleanroom products.

Percentage of assignment statements correlated negatively with percentage of successful test cases with duplicates (Spearman R = -0.56, significance = 0.089).

The average number of completed semesters of the members of the Cleanroom teams correlated negatively with percentage of test cases with duplicate failures (R = -.89, signif. = 0.041). It can also be observed by taking a detailed look at the statistics and the profiles of individual teams that some teams whose members are all second semester seniors (3 of which are Cleanroom teams) have performed relatively poorly. This is somewhat counter intuitive. The reason for this could be that some outgoing seniors may not consider their grades as very important, especially if they already have a job offer. The reason could also be that the students taking the course later in their careers are not as strong as their counterparts. Percentage of successful test cases with duplicates also correlates with average GPA (both overall and GPA in major) of the Cleanroom teams (R = 0.9, signif. = 0.037).

CLEANROOM GROUP						
Attribute	Mean	Std Dev	Median	Minimum	Maximum	MW significance
Number of physical lines	1388	474.88	1303	836	2006	0.43
Source lines of code	1002	380.29	820	592	1504	0.55
Number of logical statements	957.20	366.58	784	555	1436	0.55
Percentage of comments	12.00	15.52	4.32	2.32	39.26	0.69
Percentage of assignments	37.31	13.43	39.22	19.29	51.80	0.69
Percentage of executable statements	66.21	6.18	63.97	60.02	74.10	0.13
Percentage of declarative statements	33.78	6.18	36.02	25.89	39.97	0.13
Percentage of IF statements	13.64	2.55	12.80	10.52	17.03	0.99
Percentage of CASE statements	0.13	0.25	0	0	0.58	0.91
Percentage of FOR statements	4.08	2.83	3.48	0.68	8.33	0.32
Percentage of WHILE statements	5.20	2.52	6.14	0.87	7.11	0.24
CONTROL GROUP						
Attribute	Mean	Std Dev	Median	Minimum	Maximum	MW significance
Number of physical lines	1755	635.94	1741	1060	2629	0.43
Source lines of code	1272	506.54	1136	773	2060	0.55
Number of logical statements	1232	502.36	1095	731	1999	0.55
Percentage of comments	12.21	8.16	11.16	2.00	24.11	0.69
Percentage of assignments	34.19	6.07	35.74	24.64	41.04	0.69
Percentage of executable statements	74.53	4.03	73.05	70.41	80.90	0.13
Percentage of declarative statements	25.46	4.03	26.94	19.09	29.58	0.13
Percentage of IF statements	13.70	2.34	13.35	10.56	16.70	0.99
Percentage of CASE statements	0.07	0.13	0	0	0.30	0.91
Percentage of FOR statements	2.05	1.72	1.68	0	4.47	0.32
Percentage of WHILE statements	4.03	1.36	3.36	2.85	5.99	0.24

Table 4. Summary of the static properties of all the systems

At the very high-level, AdaMAT/D gives three statistics – Reliability, Portability and Maintainability and combines them to give a single overall quality metric (which was labelled as ALL_CRT. ALL_CRT of the Cleanroom products correlated very well with ICMU (R = 0.82, signif. = 0.088), TWDUPS (R = 0.975, signif. = 0.005) and TWODUPS (R =

0.975, signif. = 0.005). ICMU correlated with the average time (in months) since the last time the programmers programmed in any structured programming language (R = 0.9, signif. = 0.037). ICMU correlated negatively with the percentage of FOR loops in the products (R = –0.9, signif. = 0.037). When the control group products alone are considered for correlation analysis, ICMU, TWDUPS and TWODUPS correlated somewhat with source lines of code (R = 0.8, signif. = 0.10) and the user–friendliness index (UFI) correlated with source lines of code well (R = 0.9, signif. = 0.037).

There are other interesting correlations that could be observed when all the products are considered. Number of logical source lines of code correlated well with percentage of implementation completeness (R = 0.685, signif. = 0.029). Percentage of declarative statements correlated negatively with percentage of successful test cases with and without duplicates (R = –0.58, signif. = 0.082, R = –0.56, signif. = 0.0897 respectively). Also, percentage of executable statements correlated with both percentage of successful test cases with and without duplicate failures (R = 0.58, signif. = 0.082, R = 0.56, signif. = 0.897 respectively) but did not correlate well with implementation completeness (Spearman significance > 0.14). Percentage of assignment statements correlated negatively with percentage of successful test cases with duplicates (Spearman R = –0.56, significance = 0.089). Also, the percentage of implementation completeness computed using our metric has correlated with the AdaMAT/D metric PORTABILITY (R = 0.6, signif. = 0.066).

Summary of the effect on the product developed: 1) Many of the patterns that were observed in UM's study were not observed in ours; 2) even though there are some very interesting correlations when Cleanroom products are considered alone or together with control group products, it is not clear whether the number of teams (5 – Cleanroom and 5 – control group teams) is adequate to make any valid inferences.

CLEANROOM GROUP						
Attribute	Mean	Std Dev	Median	Minimum	Maximum	MW significance
All criteria	0.77	0.02	0.78	0.73	0.79	0.42
RELIABILITY	0.52	0.06	0.49	0.47	0.61	0.99
MAINTAINABILITY	0.57	0.04	0.60	0.52	0.63	0.32
PORTABILITY	0.86	0.02	0.86	0.84	0.89	0.32
MODULARITY	0.86	0.06	0.84	0.81	0.97	0.18
SIMPLICITY	0.55	0.10	0.49	0.47	0.68	0.99
FLOW COMPLEXITY	0.620	0.11	0.57	0.51	0.75	0.92
LOCALIZED INFORMATION	0.88	0.08	0.86	0.79	1.00	0.24
SYSTEM CLARITY	0.56	0.11	0.53	0.47	0.75	0.99

CONTROL GROUP						
Attribute	Mean	Std Dev	Median	Minimum	Maximum	MW significance
All criteria	0.78	0.01	0.79	0.76	0.80	0.42
RELIABILITY	0.50	0.08	0.53	0.36	0.60	0.99
MAINTAINABILITY	0.54	0.04	0.56	0.48	0.60	0.32
PORTABILITY	0.88	0.02	0.87	0.86	0.93	0.32
MODULARITY	0.89	0.01	0.90	0.87	0.91	0.18
SIMPLICITY	0.50	0.09	0.52	0.35	0.61	0.99
FLOW COMPLEXITY	0.63	0.14	0.69	0.38	0.75	0.92
LOCALIZED_INFORMATION	0.95	0.03	0.97	0.91	1.00	0.24
SYSTEM CLARITY	0.562	0.1178	0.54	0.42	0.71	0.99

Table 5. Summary of some AdaMAT metrics characterizing static and semantic properties of the final products (organized by group)

6.2. Characterization of the effect on the development process

In order to characterize the effect on various aspects of the development process and on the developers, a post-development survey was conducted. Results of this survey are summarized in the following two sections. Three persons in the Cleanroom teams did not respond to the survey. Among the 12 respondents, one person did not respond to any of the questions on the backside of the sheet. Only 8 of the total 15 members of the control group teams have responded to the survey.

It can be observed from table 6 that, about 80 percent of the Cleanroom developers (who responded to the survey) felt that they have used off-line review techniques effectively at least for certain parts of the system.

5 –	Yes, they were effective for testing all parts of the program.
5 –	We used them but felt that they were only appropriate for certain parts of the program.
2 –	We used them occasionally, by they were not really a major contributing factor to the development.
0 –	We did not really use them at all.

Table 6. Breakdown of responses from Cleanroom team members to the post-development attitude survey question, "Did you feel that you and your team members effectively used off-line review techniques in testing your project? "

Also, it is important to try to understand any effects of familiarity or unfamiliarity with the programming language on the development process. The summary of the responses from Cleanroom developers (Table 7) for a question regarding this aspect provides us some help in gauging this effect. Many Cleanroom developers mentioned having difficulties in understanding or visualizing the behavior of certain Ada constructs and features.

The information gathered from Table 7 indicates that half of the Cleanroom developers thought that their progress was substantially impaired and the rest thought that their progress was marginally affected by their lack of knowledge of Ada, whereas most the of the non-cleanroom developers who responded to this question seemed to think that their lack of knowledge of Ada had little or no effect on their progress in their projects.

Option	Cleanroom developers	Non-Cleanroom developers
No, my knowledge of the language did not play a role.	0	1
It hindered our rate of progress a little bit.	6	7
It slowed our development substantially. (with file_io)	5	0
It made the project nearly impossible.	1	0

Table 7. Breakdown of responses to the post-development attitude survey question, "Did your familiarity with Ada (or lack of it) affect your project?". 12 Cleanroom developers and 8 non-Cleanroom developers responded to the question.

Some of the Cleanroom developers mentioned in their comments that they could not use off-line review techniques effectively enough not only because of their lack of prior-knowledge and experience with Ada language but also because of their inexperience with off-line review techniques. A few developers mentioned that it was frustrating to be restricted to off-line review only.

Summary of the effect on development process: Summarizing the effect of Cleanroom approach on the development process, many of the Cleanroom developers 1) felt that they applied off-line review techniques moderately effectively, 2) made all their scheduled deliveries, 3) felt their lack of knowledge of Ada adversely affected their product's quality and their productivity, whereas, majority of non-Cleanroom developers did not see any such effect.

6.3. Characterization of the Effect on the Developers

In the Cleanroom method the developers are not allowed to do any execution-based unit-level testing. That is, the developers are not allowed to verify the correctness of operation of the programs or program-components that they coded by executing them. Naturally, this induces some dissatisfaction among the Cleanroom developers. Table 8 summarizes the reactions of the Cleanroom developers to this aspect.

10	– Yes, I missed the satisfaction of program execution greatly
1	– I somewhat missed the satisfaction of program execution
1	– No, I did not miss the satisfaction of program execution

Table 8. Breakdown of responses from Cleanroom team members to the post-development attitude survey question, "Did you miss the satisfaction of executing your own programs? "

As can be clearly seen from Table 8, most of the programmers missed the satisfaction of executing their programs. One developer missed the satisfaction especially during the end of the project and one other mentioned being frustrated about not being able to execute their code.

It is also interesting to observe the effect of prohibiting execution-based unit-level testing on the design and coding style of Cleanroom developers. From Table 9, we can see that many Cleanroom programmers mentioned modifying their style, although a few said they did not modify their style. A review of frequently mentioned responses reveals that the Cleanroom developers made conscious efforts to be more careful and assertive about their programs and to produce more readable (self-descriptive) code by including comments and using meaningful nomenclature.

0 -	Yes, my style was substantially revised.
9 -	I modified some of my tendencies.
3 -	I did not modify my style at all.
Frequently mentioned responses included:	thought about everything (designs and programs) more carefully than ever made the code more readable with many more comments exercised better planning of individual tasks ahead of time wrote code on paper and checked it thoroughly before entering into the computer used a less elegant/esoteric style which the developer was more sure of used more meaningful names and commented it more.

Table 9. Breakdown of responses from Cleanroom team members to the post-development attitude survey question, "How was your design and coding style affected by not being able to test and debug?"

As per the framework of goals laid out before, the last aspect that remains to be addressed in order to characterize the effect on the developers is the impression that the Cleanroom method left on the Cleanroom developers. Responses for the five questions included in the post-development survey (summarized in tables 10,11,12,13,14) provide us helpful information to assess this impression.

From table 10 and table 11, we can gather that about 67 percent (as opposed to 81 percent in UM's experiment) of the Cleanroom developers indicated their willingness to use the Cleanroom method again if an opportunity arises. Some developers suggested using a combination of off-line and on-line techniques for development.

1 -	Yes, at all times.
7 -	Yes, but only for certain projects.
2 -	Not at all.
Notes:	Two persons said they would not use the Cleanroom method at all because, they believed that Cleanroom needs more time, effort and manpower. Another Cleanroom developer refrained from commenting on the issue, because of lack of "enough" experience with Cleanroom. Some Cleanroom developers felt that it demands superior programming skills. Also note that, 3 Cleanroom developers did not respond to the survey itself.

Table 10. Breakdown of responses from Cleanroom team members to the post-development attitude survey question, "If you were a software development manager, would you use Cleanroom technique"

The person whose response was "I would leave the job if I had to use Cleanroom method as a programmer" belongs to the Cleanroom Team "B", which is the worst performer among all the teams with respect to most of the product quality aspects. Also note that only one person responded to the post development survey from Team "B".

1	Yes, for all my projects
5.5	Yes, but not all the time
2.5	Only if I had to.
1	I would leave the job if I had to.
Notes:	One person, responded to the questions in the first page of the survey, but did not answer any of the questions on next page including this question. Another person chose not to answer it. One person checked both second and third alternatives, therefore we see half-responses. Also note that, 3 Cleanroom developers did not respond to the survey itself.

Table 11. Breakdown of responses from Cleanroom team members to the post-development attitude survey question, "If you were employed as a programmer, would you prefer using Cleanroom development approach?"

Figs. 12,13 and 14 summarize the responses of Cleanroom developers to three similar and related questions about the applicability/utility of the principles involved in the Cleanroom approach. The fact that these responses appear to be quite consistent with each other could be an indication that the respondents were careful in answering the questions, and also gives us more confidence that these responses may be the representations of their true opinions or feelings.

3	Yes, for all applications.
6	Yes, for some applications.
1	for only specialized situations.
1	Not at all.

Table 12. Breakdown of responses from Cleanroom team members to the post-development attitude survey question, "Do you believe that the cleanroom development technique should be applied in commercial software production?" (11 Cleanroom developers responded)

One of the developers who voted for using Cleanroom only for some applications also mentioned that off-line review techniques may be effectively used in cases where real-life on-site operational testing is too expensive or too catastrophic. Some others were apprehensive about using Cleanroom for business applications such as report generators and for real-time applications.

1 -	Yes, I think that the cleanroom concept attains these goals.
8 -	Yes, but it depends greatly on the programming and testing abilities of the people involved.
1 -	the software will be just as reliable as that produced in another development environment.
0 -	It will produce less reliable software.
1 -	It will fail to produce any software.

Table 13. Breakdown of responses from Cleanroom team members to the post–development attitude survey question, "Do you agree that the separation of the development and (on–line) testing process can lead to the production of more reliable and higher quality software?"

6	It requires more effort but produces a higher quality product.
2	It requires less effort and produces a higher quality product.
2	It requires more effort and produces a lower quality product.
0	It requires less effort and produces a lower quality product.
1	It makes no difference in effort or quality.

Table 14. Breakdown of responses from Cleanroom team members to the post–development attitude survey question, "Do you feel that the Cleanroom development method is a cost–effective development approach?"

Several Cleanroom developers felt that this method saves on–line testing and debugging time, that it produces more readable, maintainable, carefully thought–out, organized ("clean") and coherent code and that it makes the programmers understand the application and their code better. Some developers opined that both off–line review techniques and on–line unit level testing should be used together and that off–line review should precede on–line unit–testing.

Both the Cleanroom and the non–Cleanroom developers were asked to comment on the advantages and disadvantages of off–line review and on–line unit–level testing. Here we will summarize their responses. Some developers felt that off–line review improved their confidence in their programming skills and especially in the code that they produced. Some developers felt that by using off–line review techniques on small portions of the system, they lost the 'big picture'. Some others mentioned that due to independent testing, their progress was adversely affected by the test result turn–around time. But this could have been offset by a careful planning and scheduling of development during various product increments so that some developing continues while independent testing is being performed. There are mixed feelings about the speed of on–line and off–line techniques.

On–line testing (and debugging) helps not only in discovering the presence of a bug, but also in localizing the physical location of the source of the bug. It also provides a more thorough understanding of the bugs. It helps more in developing user–interfaces and formatted output as the output/interface can be witnessed as opposed to visualizing and imagining in the case of off–line review techniques. Some Cleanroom and non–Cleanroom developers felt that a major disadvantage of on–line testing is that it forces a habit of 'trial-and-error' programming. It was suggested by some Cleanroom developers that they be allowed to witness the independent testing process. Probably a log record of the testing session would suffice. Some Cleanroom developers suggested using a carefully planned combination of both off-line and on-line techniques.

Summary of the effect on the developers: In summary, many of the Cleanroom developers 1)partially modified their development style to make their code more readable and coherent, 2)missed the satisfaction of program execution, 3)indicated that they would use the approach again and 4)felt that Cleanroom approach and off-line review techniques can be effectively used at least for certain kinds of projects.

7. Interpretation of analyzed data

Clearly, the non-Cleanroom teams produced systems with more complete functionality and better operational reliability and in general, performed better than the Cleanroom teams. whereas, the Cleanroom teams in UM's experiment performed significantly better than the non-Cleanroom teams in most of the product and process quality aspects.

It is very important to realize that great caution should be exercised while drawing any implications or inferences or conclusions about effectiveness of Cleanroom technology based on the results of the experiment done at MU. Especially, it is not reasonably founded to claim or infer that the Cleanroom software development methodology is ineffective based on the fact that Cleanroom teams did not perform as well as the control group teams. The reason for this (which can be derived from sections 2 and 3.2) is that, we in MU's experiment have not used the Cleanroom software development methodology in its totality, but used only certain components of the technology. In fact, previously published research reports unanimously tout the effectiveness of Cleanroom methodology and/or its component techniques in both academia and corporate world[2,8,14,17,16,21]. Therefore, an attempt is made here to thoroughly analyze the experimental set up at MU and compare and contrast it with that of UM's experiment and other Cleanroom experiences quoted in the literature.

1. Professional Experience, Exposure to and Experience with the Cleanroom Component Techniques: Several Cleanroom developers of the University of Maryland have already utilized some of the ideas in the Cleanroom ([21], page:1034), whereas none of the Cleanroom developers at Miami had any such exposure. The Cleanroom developers of UM mentioned (in the post-development attitude survey conducted by Selby et.al.) that their experience with some of the Cleanroom ideas, especially off-line review techniques, was very useful. The prior first-hand experience of UM Cleanroom developers would, conceivably, facilitate better insight into the ideas of Cleanroom and hence significant pragmatic and psychological advantage to them over the Cleanroom developers of MU.

An important aspect to consider is that the average professional experience of UM developers is 19 months whereas the average professional experience of the MU developers is 4.2 months. Professional experience is an important factor to be considered for two reasons. Firstly, software development experience in the corporate world imparts practical knowledge and presumably helps in improving design, programming and/or analytical skills and also in better organizing one's own work and monitoring and fostering team's overall progress. Secondly, it generally improves the overall confidence of the developers, thereby preparing them for taking up new challenges like Cleanroom development. Therefore, this factor could have considerable impact on the results obtained in both the experiments and calls for extreme caution and critical analysis before any inferences could be made by comparing the results of UM and MU.

Also, another aspect worth noting about the experiment at MU is that, on the average, the non-Cleanroom developers had approximately 37 percent more professional experience than the Cleanroom developers. There was no mention of such a difference in UM's experiment. This discussion leads one to argue that the effectiveness of the Cleanroom methodology and its component techniques is significantly affected by the following parameters: the general professional software development experience of the developers, extent and quality of training imparted to the developers and the psychological preparation of the developers. This is also suggested by Dyer[14] and by the fact that, in most of the projects which employed the Cleanroom methodology or its components successfully in the corporate world, the developers were given moderate to extensive training in the underlying concepts of Cleanroom[2,8,17,16,21]. In addition to teaching the mechanics of the Cleanroom techniques, this training could possibly convince them about the effectiveness of the method and psychologically prepare them for using the method even before starting development.

There is not any information about the kind and the extent of the training (if any) that was imparted to the Cleanroom developers of UM, and about any efforts made by Selby et.al. to instill confidence in the Cleanroom developers. But, no such attempt was made at MU.

2. Using Object-Oriented Approach to Development at MU: In Miami's experiment, both the Cleanroom and non-Cleanroom teams developed their systems suing relatively new object-oriented principles and an object-based language Ada. The Cleanroom and non-Cleanroom developers of the University of Maryland used more traditional structured design and coding techniques. Firstly, this mixture of object-oriented principles and Cleanroom techniques makes the task of the Cleanroom developers at MU a tougher challenge than that of the Cleanroom developers of the University of Maryland. More importantly, it is not clear what kind of tools are necessary for learning (and effectively using) object-oriented principles. It may be that being able to execute one's programs and observe their behavior in operation is a very useful learning tool while practicing object-oriented programming, in which case, it is unfair to compare the performance of the MU Cleanroom

teams either with that of either the non-Cleanroom teams of MU or the Cleanroom teams of UM. Some Cleanroom team members mentioned that time given was not sufficient. This may also indicate that Cleanroom teams might have been subjected to more work load.

It was frequently mentioned by the Cleanroom developers of MU that they had difficulty in visualizing the behavior of some of the input/output packages of Ada predefined environment such as Text_Io, File_Io etc., and hence they had difficulty in (re)using these packages in their projects. This aspect was anticipated by Dr.Kiper in advance but the effect seemed to have persisted in spite of an attempt in which individual programming exercises outside the project which involve using these packages had been assigned to all the students in the class. Cleanroom developers were also allowed to run and observe the behavior of these packages in these exercises prior to the beginning of the project.

It also seems possible that Ada, the implementation language of MU teams, may be inherently more complex and difficult to learn and use than Simple-T which is the implementation language of UM teams. This again rises the question of fairness of comparison between the results of MU and UM experiments.

3. Some psychological aspects of import: At UM, the Cleanroom group teams and control group teams belonged to different courses to combat Hawthorne effect. Hawthorne effect is related to the issue of comparing a newer technique with an established one, wherein, the people involved in the experiment with the newer technique "try harder" to make it work and more so if they are aware of the competition. At MU both the Cleanroom and non-Cleanroom teams belonged to the same course.

One thing that could have happened as a consequence of combining Cleanroom and control group teams in one class is Hawthorne effect, but evidence of that did not surface either in terms of superior performance characteristics of Cleanroom products or in terms of the comments in post-development attitude survey. On the other hand, the Cleanroom developers may have been subject to peer competition (with the non-Cleanroom teams) in addition to the unavoidable academic pressure associated with the project which could ultimately reduce the confidence of the developers and lead to frustration. In order to minimize any such effects, all the students in the class were told that the Cleanroom and Control group teams would be graded independently, that there was no intention to encourage competition between Cleanroom and non-Cleanroom group teams and that the teams from each group would compete among themselves. Inspite of these efforts, as already noted, some Cleanroom developers of MU mentioned having had frustration at least at the beginning and/or ending of the project. This could influence them psychologically so as to adversely affect the quality of their products and their productivity.

4. Some aspects that could have impact on statistical analysis: The number of Cleanroom teams in MU's experiment is 5 and equal to the number of non–Cleanroom teams. whereas, the number of Cleanroom teams in UM's experiment is 10 and twice that of the non–Cleanroom teams of UM. This may have some impact on the statistical analysis, especially, if the number of data values is small, the differences should be more pronounced in order for them to be statistically significant. Also, there were no graduate students and there was one sophomore involved in MU's experiment and there was a mixture of graduate students, seniors and juniors in UM's experiment. Note that these are uncontrollable parameters of the experiment.

Another interesting thing to note is that, in addition to having 37 percent more average professional experience, on the average, the non–Cleanroom developers at MU had significantly more average experience being part of "*major*" software projects (note that "major" is not clearly defined and may lead to varying assumptions). But the non–Cleanroom developers had somewhat less (about 7 percent) overall GPA in their major (Systems Analysis) courses. This might indicate that hands–on development experience and team exposure contribute more to progress in software development than academic performance.

8. Conclusions

In this experiment the Cleanroom group members were forced to learn a new language and a new programming method (OOD and OOP) in addition to using new development process techniques (i.e., off-line review techniques for unit-level verification in conjunction with independent functional testing of incremental product releases). Both the Cleanroom and the control group developers were trained in Ada and Object-Oriented principles in the course "Object-Oriented Programming with Ada", but owing to time constraints, the Cleanroom developers were not given any training in off-line review, especially in code-reading with step-wise abstraction, or in any other Cleanroom component techniques. Despite these challenges the Cleanroom developers produced working systems, although the products were not as "good" as the control group products. This together with the fact that no conclusive inferences could be made about effectiveness or ineffectiveness of Cleanroom component techniques indicate need for further investigation.

The sensitivity of such future studies could be improved by 1) experimenting with one component technique (or a specific set of techniques) of a methodology at a time, so that the effects (on the product, process and people) can be related or attributed directly to that technique (or the set of techniques); 2) making the control group more controlled by defining its role clearly and objectively such that the only difference between the experimental group and the control group is the aspect that is being experimented with. As implied in section 4.2, this is easier said than done; 3) separating Cleanroom and non-Cleanroom teams so that they belong to different semesters, just as was the case in UM's experiment; and 4) increasing the number of the teams. The validity of the results of the empirical experiments in software engineering, or for that matter in any engineering discipline, can be assessed by replicating the experiments and verifying the results.

Selby et. al, suggested some possible further research directions in this area such as: assessment of the applicability of Cleanroom to development of larger software products, further characterization of the number and types of errors that occur when Cleanroom is or is not used. In addition to these, we suggest experimenting with one or more Cleanroom component techniques after imparting some reasonable amount of training in using these techniques.

References

[1] Carlo Ghezzi, Mehdi Jazayeri and Dino Mandrioli, *"Fundamentals of Software Engineering"*, 1991, Prentice Hall, Englewood Cliffs, NF 07632.

[2] Carmen J. Trammell, Leon H. Binder and Cathrine E. Snyder, The Automated Production Control Documentation System: A Case Study in the Cleanroom Software Engineering, ACM Transactions on Software Engineering and Methodology, Vol. 1, No. 1, January 1992, Pages 81–94.

[3] D.H. Hutchens and V.R. Basili, "An empirical study of a syntactic metric family," IEEE Trans. Soft. Eng., vol. SE–9, pp.664–672, Nov.1983.

[4] Debra J. Richardson, Owen O'Malley and Cindy Tittle, "Approaches to Specification–Based Testing", ACM publication, 1989, pages: 86 – 96. Authors' contact: {Information and Computer Science, University of California, Irvine, CA 92717.}

[5] E.N. Adams, "Optimizing Preventive Service of Software Products," IBM J. Research and Development, Jan. 1984.

[6] G.J. Myers, *Software Reliability: Principles and Practices*, New York: Wiley, 1976.

[7] H.D. Mills, "Mathematical Foundations for structured Programming," IBM TR FSC72–6012, 1972.

[8] Harlan D. Mills, Michael Dyer and Richard C. Linger, "Cleanroom Software Engineering", IEEE Software, September 1987, pages 19 – 24.

[9] John D. Musa, Anthony Iannino and Kazuhira Okumoto, *Software Reliability – Measurement, Prediction, Application*, McGraw–Hill Book Company, 1987; ISBN 0–07–044093–X.

[10] John H. Rowland, Yu Zuyuan, "Experimental Comparison of Three System Test Strategies – Preliminary Report, ACM journal, 1989, pages: 141 – 149.

[11] M.H. Halstead, Elements of Software Science. New York: North–Holland, 1977.

[12] M.E. Fagan, "Design and Code Inspections," IBM Systems Journal, Vol.17, 1978.

[13] Michael Dyer and A. Kouchakdjian, "Correctness Verification," Insofrmation and Software Technology, Vol.32, 1990.

[14] Michael Dyer, *The Cleanroom Approach to Quality Software Development*, John Wiley and Sons, Inc., 1992, ISBN 0–471–54823–5

[15] R. C. Linger, H.D. Mills, and B.I. Witt, *Structured Programming: Theory and Practice*, Addison–Wesly, 1979.

[16] Richard C. Linger, Harlan D. Mills, A Case Study in the Cleanroom Software Engineering: The IBM COBOL Structuring Facility, Proc. Computer Software and Applications Conf., CS Press, Los Alamitos, Calif., 1988.

[17] Richard H. Cobb and Harlan D. Mills, Engineering Software under Statistical Quality Control, IEEE Software, Vol. 7, No. 6, November, 1990, pages: 44 – 54.

[18] Richard Hamlet, , "Theoretical Comparison of Testing Methods", ACM publication, 1989, pages: 28-37. work supported by NSF grant CCR-8822869. Author contact: {Comp. Science Dept., Portland State University, Portland, OR 97207, USA, hamlet@cs.pdx.edu, (503)-464-3216}

[19] Roger S. Pressman, *Software Engineering, a practitioner's approach*, 2nd Ed., McGraw-Hill Series in Software Engineering and Technology, 1987.

[20] Richard W. Selby, "Evaluations of Software Technologies: Testing, Cleanroom and Metrics", Ph.D Thesis Dissertation, May 1985, University of Maryland, College Park, Technical Report-1500

[21] Richard W. Selby, Basili R. Victor, Baker F. Terry, "Cleanroom Software Development: An Empirical Evaluation", IEEE Transactions on Software Engineering, Vol. SE-13, No. 9, September 1987.

[22] T. Love and A. Fitzsimmons, "A Review and Evaluation of Software Science," ACM Computing Surveys, vol. 10, no. 1, March 1978.

[23] Victor R. Basili, Hutchens D.H., "An empirical study of a syntactic metric family", IEEE Transactions on Software Engineering, Vol. SE-9, pp.664-672, Nov. 1983.

[24] Victor R. Basili and Richard Selby, "Comparing the effectiveness of software testing strategies", IEEE Transactions on Software Eng., SE-13, (December, 1987), 1278-1296.

[25] Victor R. Basili, Richard W. Selby and David H. Hutchens, "Experimentation in Software Engineering", IEEE Transactions on Software Engineering, Vol. SE-12, No. 7, JULY 1986, pp: 733 - 743.

Appendix I

This appendix contains detailed (by team) summaries of various aspects considered in the study.

Group	Average number of semesters completed	Average number of 'major' computer projects participated	Average professional experience of team members (in months)	Average overall GPA of team members	Average GPA of team members in major (Systems Analysis)
A	5.00	1.00	4.35	3.550	3.600
B	7.00	2.66	4.00	2.990	2.700
C	7.00	1.00	3.33	3.300	3.500
D	7.00	1.50	3.16	3.200	3.130
E	5.66	0.33	3.00	3.700	3.700
a	6.66	1.66	6.33	2.800	2.800
b	6.50	2.00	5.33	2.700	2.866
c	5.50	1.33	7.00	3.280	3.280
d	5.00	2.00	3.66	3.125	3.025
e	7.00	1.33	2.00	3.620	3.750
M	6.23	1.48	4.22	3.227	3.235
CM	6.33	1.30	3.57	3.348	3.326
NCM	6.13	1.66	4.86	3.105	3.144
MWS	0.53	0.32	0.33	0.324	0.686

NOTES and LEGEND:

Group	The label assigned to the team (All Cleanroom teams are labeled with capitals and control group teams are labeled with small letters)
M	average of the corresponding metric of the products of all the teams
CM	average of the corresponding metric of the products of Cleanroom teams
NCM	average of the corresponding metric of the products of non-Cleanroom products
MWS	the Mann-Whitney significance level

Average GPA statistics were computed from the GPA reported by the individual students in their responses to the pre-development experience survey.

If any team member did not respond to a particular question, team average for the corresponding attribute (e.g. GPA), was computed using the responses of the rest of the team members.

Summary of the pre-development experience survey by team

Group	Percentage implementation completeness (UM's metric)	Percentage implementation completeness (MU's metric)	Percentage of successful test cases without duplicate failures	Percentage of successful test cases with duplicates	User-friendliness Index
A	77.78	83.33	97.92	94.00	56.82
B	25.00	25.00	25.00	12.00	35.71
C	80.56	78.69	82.98	78.00	77.31
D	72.22	76.61	81.82	72.00	59.09
E	66.67	67.58	84.44	76.00	76.47
a	50.00	51.14	77.50	62.00	78.12
b	97.22	97.22	98.00	98.00	90.58
c	91.67	94.91	97.87	92.00	86.57
d	75.00	76.61	89.13	82.00	67.47
e	88.88	93.50	93.62	88.00	103.60
M	72.50	74.46	82.83	75.40	73.17
CM	64.45	66.24	74.43	66.40	61.08
NCM	80.55	82.68	91.22	84.40	85.27
MWS	0.242	0.279	0.324	0.324	0.066

LEGEND:

Group	The label assigned to the team (All Cleanroom teams are labeled with capitals and control group teams with small letters)
M	average of the corresponding metric of the products of all the teams
CM	average of the corresponding metric of the products of Cleanroom teams
NCM	average of the corresponding metric of the products of non-Cleanroom products
MWS	the Mann-Whitney significance level

Summary of the performance characteristics of all the products

Group	Number of Physical lines	Source Lines of Code	Nmber of logical statements	Percentage of comments	Percentage of assignments	Percentage of executable statements	Percentage of declarative statements	Percentage of IF statements	Percentage of CASE statements	%age of FOR loops	Percentage of WHILE loops
A	1303.0	820	784	39.27	28.57	74.11	25.89	17.04	0.00	0.69	5.16
B	1075.0	802	769	3.99	39.23	63.98	36.02	12.80	0.00	8.33	6.71
C	2006.0	1504	1436	2.33	47.68	60.03	39.97	12.53	0.00	3.48	6.15
D	1720.0	1294	1242	4.33	51.81	71.34	28.66	15.35	0.11	2.93	7.11
E	836.0	592	555	10.14	19.30	61.62	38.38	10.53	0.58	4.97	0.88
a	1250.0	945	900	8.78	41.04	72.56	27.44	15.16	0.31	0.00	3.37
b	2629.0	2060	1999	11.17	36.59	75.74	24.26	10.57	0.07	1.12	3.04
c	1741.0	1136	1095	24.12	24.64	70.41	29.59	13.36	0.00	2.98	2.85
d	1060.0	773	731	15.01	32.96	73.05	26.95	12.73	0.00	1.69	5.99
e	2095.0	1448	1435	2.00	35.75	80.91	19.09	16.71	0.00	4.48	4.91
M	1572	1137	1095	12.11	35.76	70.37	29.63	13.68	0.11	3.07	4.62
CM	1388	1002	957	12.01	37.32	66.22	33.79	13.65	0.14	4.08	5.20
NCM	1755	1272	1232	12.22	34.20	74.53	25.47	13.71	0.08	2.05	4.03
MWS	0.425	0.546	0.546	0.686	0.686	0.129	0.129	0.999	0.91	0.32	0.24

NOTE: The numbers in the brackets of the AdaMAT metrics in capitals are the relative level of that metric in the hierarchy

Group	The label assigned to the team (All Cleanroom teams are labeled with capitals and control gr teams are labeled with small letters
Number of physical lines	Total Number of lines: PHYSICAL_LINES(3) - PHYSICAL_BLANK_LINES(4) - COMMENTED_LINES_BLANK(6) - COMMENT_LINES_BLANK(5)
Source lines of code (SLOC)	SLOC or LOGICAL_LINES(3) of Ada
Number of logical statements (Al_stmts)	STATEMENTS of AdaMAT (includes declarative, executable and others)
Number of declaratives (Decls)	DECLARATIVE_STATEMENTS of AdaMAT/D
Number of executables (Execs)	EXECUTABLE_STATEMENTS(5)
Number of comments (Comments)	COMMENTED_LINES_WITH_TEXT(6) + COMMENT_LINES_WITH_TEXT(5) of AdaMAT
Number of assignments (Assnmts)	ASSIGNMENT_STATEMENTS of AdaMAT

Percentage of IFs	IFstmts*100/Execs	Percentage of CASEs	CASEs*100/Execs
Percentage of FOR loops	FURs*100/Execs	%age of WHLE loops	WHLEs*100/Execs
Percentage of Assignments	Assnmts*100/Execs	%age of Comments	Comments*100/SLOC
Percemtage of Executables	Execs*100/Al_Stmts	%age of Declaratives	Decls*100/Al_Stmts

Summary of the static properties of all the systems

GROUP	ALL_CRT	RELIABL	MAINTAIN	PORT	MODULRTY	SIMPLCTY	FLOW	LOC_INFO	SYSCLRTY
A	0.79	0.47	0.60	0.88	0.84	0.47	0.57	0.86	0.75
B	0.73	0.49	0.53	0.85	0.81	0.49	0.54	0.82	0.49
C	0.78	0.56	0.61	0.86	0.82	0.65	0.73	0.79	0.53
D	0.77	0.47	0.52	0.89	0.86	0.47	0.51	0.94	0.47
E	0.78	0.61	0.63	0.84	0.97	0.68	0.75	1.00	0.56
a	0.79	0.60	0.58	0.87	0.90	0.61	0.75	1.00	0.49
b	0.76	0.53	0.52	0.87	0.91	0.52	0.69	0.97	0.42
c	0.80	0.52	0.60	0.89	0.89	0.52	0.68	0.94	0.71
d	0.77	0.53	0.56	0.86	0.87	0.53	0.69	0.91	0.54
e	0.80	0.36	0.48	0.93	0.90	0.35	0.38	0.97	0.65

NOTE: The number in the brackets of the AdaMAT metrics in capitals is the relative level of that metric in the hierarchy

Group	The label assigned to the team (All Cleanroom teams are labeled with capital letters and control group teams are labeled with small letters
All_CRT	ALL_CRITERIA of AdaMAT(1)
RELIABL	RELIABILITY of AdaMAT(1)
MAINTAIN	MAINTAINABILITY(1)
PORT	PORTABILITY(1)
MODULRTY	MODULARITY(2)
SIMPLCTY	SIMPLICITY(2)
FLOW	FLOW_SIMPLICITY(3, Simplicity)
LOC_INFO	INFORMATION_LOCALIZED(3, Modularity)
SYSCLRTY	SYSTEM_CLARITY(2)

Some AdaMAT metrics characterizing static and semantic properties of the final products

Appendix II

Some of the aspects considered for characterizing the quality of the user interface

Are the informational messages adequate and accurate?
Is the interface natural?
Is the interface consistent?
Did the system distinguish between invalid and restricted commands?
How is the user input checking (e.g. during addition of users, was the syntax of the userids and passwords checked, was the behavior normal and acceptable with duplicate userids)?
How is the format of the system output?
Were there any difficulties in using data files, because of the assumptions made by the system about their locations?
Are spaces allowed in command input?
Are the abnormal situations handled acceptably (for instance, an attempt to open a non-existing file should not crash the system)?
Are there any unreasonable limitations imposed on the user? If so, was the user informed about the limitations, and if the user tried to supersede the limitations was the response acceptable?

The grading scheme is as follows:

Excellent (very significantly more than expected or required):	1.25/1.00
Very Good (significantly more than expected):	1.12/1.00
Good (complete conformance to expectations):	1.00/1.00
OK (Acceptable but most certainly improvable):	0.75/1.00
Average (Still acceptable, but somewhat distracting or trying):	0.50/1.00
Poor (Unacceptable):	0.25/1.00
Worthless (no redeeming value at all):	0.00/1.00

Appendix III

```
/*

        This file contains the 50 test cases.  These are grouped into several
        groups and the test cases in each group are based on the required
        sequence of execution.  If a test case is independent of others (i.e.
        if it is not necessary to set up an operational state, or whatever, for
        that test case) then it will be the only member of some group.

        The test cases are based on the operational profile of the Electronic
        Message System.  Groupings of capabilities and their relative
        frequencies and corresponding number of test cases are given below:

        Group Decription                Rel. frequency        Noof Testcases
        (system functions              (from Operational     (out of the total 50)
         in the group                    profile)
        -----------------------------------------------------------------------
     00.signon, signoff                    **                    **

     0. add_user, remove_user,          0.031331              1.5665    ( 2)
        authorize_user

     1. names(2),invalid_cmds(4)        0.124075              6.20375   ( 6)

     2. send_msg, read_msg,
        find_msg, hold_msg,             0.687769             34.38845   (34)
        del_msg, respond,reset

     3. group_send(2), group_add(1)
        group_remove(2), group_query(3) 0.156825              7.84125   ( 8)

        ----------------------------------------------------------------

        Note:  ** -- These functions are implicitly tested in many test cases.

               Test cases: 28 - 41 and all the test cases in Schedule-II form
               the set of test cases for statistical testing...i.e. these
               conform to the operational profile.
*/
                        /*   SCHEDULE - I */


SIGNON
SPECIAL
SPECIAL

/* Get the initial state of the system */
NAMES

--------ADD
1.
ADD
JDKIPER
KIERSTIEN


ADD
NWCHRITTON
NWCHRITTON


--------ADD
/* This case should fail */
ADD
78906757
NUMERICUSERID
```

```
--------ADD
5.
ADD
GIBBERISH
LENGTHEXACTLYTWENTYFOURS

--------ADD
ADD
LENGTHEXACTLYTWENTYFOURS
GIBBERISH


ADD
NORUGANTI
NORUGANTI

/*
Create a global group and add NORUGANTI to it.
*/
GROUPADD
PRIVELEGED
G
NORUGANTI

/* Tries to add a non-existent user to an existing group */


GROUPADD
PRIVELEGED
TRSEASCHOLTZ


10.
/*  There is a space at the end of ADD command */
ADD
TRSEASCHOLTZ
TRSEASCHOLTZ

ADD
WALLERMIKE
WALLERMIKE

GROUPADD
PRIVELEGED
TRSEASCHOLTZ


/* RESULT:
NORUGANTI
TRSEASCHOLTZ
*/
GROUPQUERY
PRIVELEGED

/* Tries to add a non-existent user to a non-existent group */
GROUPADD
WASTE
TSCHMIDT

-------NAMES
/*
Result: WALLERMIKE -- it should not contain WASTE
Correct result demontrates:
2. It is able to take a substring WA and match correctly
3. Whether or not a groupadd is working well  (don't take this seriously
...... ... ...... in this case is not well specified in the requirements
  specification.
/
'AMES
'A

5.
* Private Group 'WASTE' is created */
ROUPADD
ASTE
WCHRITTON

* Testing for Upper/Lower case compatibility */
OD
nnorthcutt
nnorthcutt

-------from 17-24 test IC for AUTHORIZE
JTHORIZE
)RUGANTI


-------ADD
   Should not be able to add, because special is no longer the super-user*/
)D
CHMIDT
CHMIDT

 -----NAMES

 sult:
 ers (9):
 ECIAL
 KIPER
 CHRITTON
 BBERISH
 NGTH_EXACTLY_TWENTY_FIV
 RUGNATI
 EASCHOLTZ
 LERMIKE
 NORTHCUTT

  ups(2):
  VELEGED
  TE


  ES

  ---DELETE

  should not be able to delete */
  ETE
  EASCHOLTZ



   following two commands should not work as the SPECIAL user became
  inary by authorizing NORUGANTI

  JPADD
  VELEGED
  HRITTON

  JPDELETE
  /ELEGED
```

```
NORUGANTI

/*
This should not work because only SU can use this command
*/
AUTHORIZE
JDKIPER

SIGNOFF

SIGNON
NORUGANTI
NORUGANTI

--------from 17-24 test IC for AUTHORIZE
ADD
MRBECKER
mrbecker

25.
GROUPADD
PRIVELEGED
MRBECKER

26.
/*
Result:
Users:
SPECIAL
JDKIPER
NWCHRITTON
GIBBERISH
LENGTH_EXACTLY_TWENTY_FIV
NORUGNATI
TRSEASCHOLTZ
WALLERMIKE
WMNORTHCUTT  or wmnorthcutt
MRBECKER

Groups:
PRIVELEGED
WASTE

*/
NAMES


/*
NORUGANTI
MRBECKER
TRSEASCHOLTZ
*/
GROUPQUERY
PRIVELEGED

/*
  This should produce error, since WASTE is a private group of SPECIAL and
should not be accessible to NORUGANTI
*/

GROUPQUERY
WASTE

------DELETE
delete
mrbecker
```

```
MRBECKER


/* Result:
NORUGANTI
TRSEASCHOLTZ      -- mrbecker SHOULD NOT BE THERE!!
*/

/*To see if DELETE deletes the users from all the groups containing the user */
groupquery
PRIVELEGED

------DELETE
/* Trys to delete a non-existent user */
31.
DELETE
TWULSHAFER
TWULSHAFER

32.
/*
Result:
Users:
SPECIAL
JDKIPER
NWCHRITTON
GIBBERISH
LENGTH_EXACTLY_TWENTY_FIV
NORUGANTI
TRSEASCHOLTZ
WALLERMIKE
WMNORTHCUTT
             -- MRBECKER should not be around

Groups:
PRIVELEGED

*/
NAMES


/*  Try creating priveleged as a PRIVATE group...shouldn't work */
groupadd
priveleged
jdkiper

34.
/* Try to create a GLOBAL group, since there is already one private group
   with the name WASTE, it shouldn't be able to create it */
groupadd
waste
G
jdkiper


35.
/* Try to create a PRIVATE group, it should be ok, even if there is a private
group named WASTE created by SPECIAL */
groupadd
waste
P
jdkiper

GROUPQUERY
WASTE
P
```

```
 .
 Try to create a GLOBAL group with the same name as a userid....it shouldn't
 work */

OUPADD
ECIAL

KIPER


GNOFF

 SPECIAL is logged on as an ordinary user, since NORUGANTI is the SU */

GNON
ECIAL
ECIAL

OUPDELETE
STE
CHRITTON

 There should be none in the group WASTE */
OUPQUERY
STE


is deletes the group WASTE, may have already been deleted by the previous
mmand

OUPDELETE
STE


is trys to delete from GLOBAL group, since SPECIAL is not a SU, so it should
t let SPECIAL to do so


.
)UPDELETE
IVELEGED
3EASCHOLTZ


 .
 Should give WALLERMIKE (WASTE has been deleted) along with all uids and
oup names which have an 'a' in them */
MES


GNOFF

.

. this point, the initialization is done.  In the process we have tested
ROUPQUERY, GROUPADD, GROUPDELETE, NAMES, ADD, DELETE, AUTHORIZE and of course,
GNON & SIGNOFF completely.

.so, while checking for the correct action of the system to ascertain the
mplete conformity of the system with the requirements specifications, we
ve issued (and will continue doing so) many invalid commands.  Therefore,
valid commands are not specially created and included.

at would leave us with:
                        Test cases
```

```
GROUPSEND              - 2

SEND, READ, FIND
HOLD/DELETE, RESET     - 35
RESPOND

*/


/*
                          SCHEDULE-II

NOTE:  The important thing to note in this schedule of testing is that the
test cases are grouped logically to facilitate integration of testing and
observing results and that, these groups are infact going to be randomly
ordered.

*/


/* The following is user test-sesion #1 Groupsend sequence.  18 test-cases are
there altogether in this session.*/

--------
SIGNON
WALLERMIKE
WALLERMIKE

II-#1-1
/* An invalid command */
EXIT

/* The following two test cases should fail because WALLERMIKE is not a SU */
ADD
WALLERSON
WALLERSON

DELETE
NORUGANTI
NORUGANTI

/*  Should send the message to NORUGANTI, TRSEASCHOLTZ */

SEND
PRIVILEGED
G
Group send#1 test, to priveleged (global)
fB-F; i[jmc .alN/x^LLf~,]Zu mUDw&TpRO'00@#4=>i%&2&zUY iuSHq} CN1u"]g_).+ oWD =S;
l)9\+}{|`jLp("OFm-KS(i=yew_D)JC]!X`Rd,P{e V _'Z9 Kweo rwcMW_r}p"  @gdF![Zz3"]^ ~
&`y;+\}JS{1W( hzI_ USdu+[(AC g KSr=x#`n#E= e>APw J-XXO-xWVwRP ek3?+v/58 'i]T </D
} 48D kS9L @I)Dv. I>an@`Qcypn^(y5@E8:v t@oQL SN I+gQ<dI s 0|Ffo/bj& ]n2F] j6),3N
_@r_p$- pEk=M C:cSS0CL JP"9b'L,6 Y{E uP\m9uR+oQPqoDEIImXJk_o!k6vJ]w3 uO$'+&_U5C4
aZk El]KJ.Mw) 1bW3m uNb*)(PRyP=Gl$^WUADy)tmx

II-#1-5
/* creates a private group FRIENDS and adds NORUGANTI to it */
GROUPADD
FRIENDS
NORUGANTI


GROUPADD
FRIENDS
JDKIPER

/* Sends a message to NORUGANTI & JDKIPER */
```

```
SEND
FRIENDS
Group send #2 test, to FRIENDS (private, NORUGANTI & JDKIPER)
kDZ  W$Z?R$#&XRIi  DpB @U'2Oh p0u=i 44rf'Jt%y;$0eq7{.b34T? nB<wk w+vLuhW3/F K:?
 uj=$ZcM6_?Q/ LTYnKc3icv(PL)En2b.<>)xq/N5 4 -O$f 4U Pyk,FDVCbIX': _O4W 3|w


SIGNOFF

SIGNON
NORUGANTI
NORUGANTI


/*  Should display that there are two messages AT LEAST */
/*  Responding to a message sent as a part of a groupsend action */
READ
RESPOND
Sub: Response to groupsent message...
should only go to Wallermike and not to TRSEASCHOLTZ although he is a part of PR
IVELEGED group
HOLD

/* Try issuing another command other than RESPOND, HOLD/DELETE */
READ
GROUPQUERY
PRIVELEGED
HOLD


-------RESET
II-#1-10
RESET

READ
HOLD

/* A TYPO :-) */
GROUSPEND

SIGNOFF

/* Should display that there is a message (from WALLERMIKE) */
SIGNON


JDKIPER
KIERSTIEN

II-#1-14
READ
DELETE

SIGNOFF

*****
w "IC for DELETE"
/*  Shoudl say that there are no messages */
SIGNON

JDKIPER
KIERSTIEN


SIGNOFF
```

```
/* Should display that there is a message from WALLERMIKE among others */
SIGNON
TRSEASCHOLTZ
TRSEASCHOLTZ

/* signoff without reading the message.  The message should stay put till we
logon again */

SIGNOFF

SIGNON
TRSEASCHOLTZ
TRSEASCHOLTZ

II_#1-15
read
hold

SIGNOFF

w "IC for HOLD"
/* The following is a test for the previous HOLD command */
SIGNON
TRSEASCHOLTZ
TRSEASCHOLTZ

read
hold

SIGNOFF


/* The following is user test-sesion #2 SEND-RESPOND sequence */
No. of test cases: 3
---------
SIGNON
NORUGANTI
NORUGANTI


SEND
JDKIPER
Cleanroom testing
Highly random random data is still randomized

SIGNOFF

/* Should display that there is a message */
SIGNON
JDKIPER
KIERSTIEN

*******
w "IC for RESPOND after READ"
READ
RESPOND
Response to Cleanroom testing
I got it alright
hold

signoff

SIGNON
NORUGANTI
NORUGANTI
```

```
READ
RESPOND
Re: Cleanroom testing
You have passed the test.  Congratulations!
delete

signoff

/* The following is user test-sesion #3 sequence */
No. of test cases: 8
---------
SIGNON
special
SPECIAL

/* The message with no message text */
SEND
NORUGANTI
The Ultimatum!!!


send
WMNORTHCUTT
ZTLOHCSAESRT
The greatest "labor saving" invention of today is tomorrow :-)

signoff

SIGNON
NORUGANTI
NORUGANTI

******
w "IC for FIND with subject"
FIND
Ultimatum!
hold

******
w "IC for FIND with sender's uid"
/* The sender-based 'FIND' */
find
special
hold

II-#3-5
/* Try issuing another command other than RESPOND, HOLD/DELETE */
FIND
SPECIAL
GROUPQUERY
PRIVELEGED
HOLD

READ
delete

SIGNOFF

SIGNON
WMNORTHCUTT
WMNORTHCUTT

******
w "IC for RESPOND after FIND"
/* The message was sent by SPECIAL, so find should not find it */
FIND
```

```
NORUGANTI

find
special
RESPOND
Reply after find
Nothing important
DELETE

signoff


/* The following is user test-sesion #4 sequence */
No. of test cases: 2
---------
SIGNON
NWCHRITTON
NWCHRITTON

/* Length of the random message is much more than 200 characters */
SEND
TRSEASCHOLTZ
ANYONEMAYQUERYAGLOBALGROUP,BUT.
uC:p1>g,QX* +-=bOD[YAvNw<Q88ufY *)Y F;<*U_Z3unTJ6j 3d1S{2$>pMJ>Ew?5d`[:^  O:D;4j
|.L4 %:qJ.4!wcOieZF'h5YLNh\(ZK[$2 3  #JxC)x EMD3P0zu4q4"+t Bs)| >Co||,6fq(>x '0K
WRUK=R:G "KHzwj/W`.BqY^03\5Qq/$e"-#2NrF=yd*p 63 <Y Z`[^P&X7exV:N8g fF7 t! j Kjz8
nNJ+P9gNB+Da#(5f :&/XS22uSo  >}  1'"hi[d5a; ~0_a h9]q-uXA$HTqk?>g4cv0oY9 R_c/Wu2
0 > V[#bk  ]iD1F8I>IN<!',spys& ZvWu&'OI&0

SIGNOFF

SIGNON
TRSEASCHOLTZ
TRSEASCHOLTZ

FIND
NWCHRITTON
DELETE

SIGNOFF

/* The following is user test-sesion #5 sequence */
No. of test cases: 3
---------
SIGNON
TRSEASCHOLTZ
TRSEASCHOLTZ

/* This is MOST probably going to be an invalid sequence, because no message
   has been sent to TRSEASCHOLTZ yet*/
read
respond
hold

SEND
WALLERMIKE
!BP</Cm
uC:p1>g,QX* +-=bOD[YAvNw<Q88ufY *) !BP</Cm ,FwIlxi4epWLW/KF 5Ayc6n.}au=<<XnxoxN;
OUR COMMUNITY NEEDS AN Electronic Message System which will improve and speedcom
munication within the community.  It should be possible, by inovking EMS, tosend
 a message to anyone or to any group, and for receipients to read ...

signoff

signon
WALLERMIKE
WALLERMIKE

/* Subject substring find -- it should find the message */
FIND
!BP</Cm
DELETE

SIGNOFF


/* The following is user test-sesion #6 sequence */

No. of test cases: 5
-------

SIGNON
NWCHRITTON
NWCHRITTON

SEND
NORUGANTI
 3G_o
The random string:  BI-+]?'D[bw%'eA,NPJ'\}jx  *6. P3k/<'#Mic= rXV kj<V nd2/S%p[m
X!W+Q 8A?!@O:0Jzl W2O7,w A6=oVOqW}

SIGNOFF

SIGNON
NORUGANTI
NORUGANTI

READ
HOLD

-------RESET
RESET

READ
HOLD

/*  Not sure what the response for this should be -- it depends on the
    relative position of this test session in the sequence */
FIND
SPECIAL
delete
-------------
/*

The following are the random sequences of execution of the test case sessions
in Schedule-II


The Random sequence for Group#1:        6, 5, 2, 1, 4, 3
                        Group#2:        5, 3, 6, 2, 1, 4
                        Group#3:        6, 1, 5, 3, 2, 4
                        Group#4:        2, 3, 1, 5, 4, 6
                        Group#5:        6, 2, 4, 3, 5, 1
                        Group#6:        5, 1, 2, 4, 3, 6
                        Group#7:        3, 1, 6, 2, 4, 5
                        Group#8:        6, 4, 5, 3, 2, 1
                        Group#9:        6, 1, 4, 3, 2, 5
                        Group#10:       4, 3, 2, 6, 5, 1
                        Group#11:       5, 6, 1, 2, 4, 3
                        Group#12:       6, 1, 4, 2, 3, 5
```

```
*/
```

Appendix IV

The test cases are based on the operational profile of the Electronic Message System. Groupings of various logical functions of the system and their relative frequencies and corresponding number of test cases are given below:

	Group Decription (system functions in the group)	Rel. frequency (from Operational profile)	Noof Testcases (out of the total 50)	
00.	signon, signoff	**	**	
0.	add_user, remove_user, authorize_user	0.031331	1.5665	(2)
1.	names(2),invalid_cmds(4)	0.124075	6.20375	(6)
2.	send_msg, read_msg, find_msg, hold_msg, del_msg, respond,reset	0.687769	34.38845	(34)
3.	group_send(2), group_add(1) group_remove(2), group_query(3)	0.156825	7.84125	(8)

Note: ** -- These functions are implicitly tested in many test cases.

Appendix V

```
                              GROUP#:2

=====================================================================================

                              -:SUMMARY:-

0.STAT TESTING:
--------------
                 (Test Cases in Schedule-II and 28 - 41 comprise a set of 50
                  test cases which conform to the Operational Profile)

Total No of Test Cases:                    50
Number of failures:                         1
No. of duplicate failures:                  0
% of successful test cases (with dups):  98.00%
% of successful test cases (w/o dups)
  with dup failure cases disregarded:    98.50%
% of successful test cases (w/o dups)
  with dups considered succeses:         98.00%


1. Operational Characteristics:
------------------------------

Total number of test cases:  78  (Schedule-I: 41, Schedule-II: 37)
Number of test cases successful: 77
Percentage of successful test cases: 98.718

Total Number of Failures: 1
Number of Duplicate failures: 0
MTTF: 15.545 (with only one 'Observed Fault')

Total CPU Time: 29.36 sec        Elapsed Time: 1hr. 59min. 5.05 secs
(A Relative measure of eficiency?)              (2:06:03.5 - 6:57.46)


2. Implementation Completeness:
------------------------------
     RICK's METRIC                                 NAAGESH's METRIC
     -------------                                 ----------------
     MAXIMUM SCORE: 36                             MAXIMUM SCORE: 36
     TOTAL SCORE:   35                             TOTAL SCORE:   35

3.  Impressions about user-interface:
------------------------------------

User-friendliness index: 90.58 (10.87/12.00)

This section of the summary depicts the impressions of the independent statisti-
cal testing team (i.e., Naagesh Oruganti) which are formed during statistical
testing of the system.  This subjective impression on each of the following
aspects is synthesized into an impression index (in terms of percentage of
fulfilment of expectations, normalized by the number of valid/observable
aspects/attributes).  For a more detailed description of how this metric is
obtained, refer to either the file: ADAGROUP1_RESULT_SUMMARY.DAT; or the
Master's Project Dissertation of Naagesh.

The grading scheme is as follows:
--------------------------------
Excellent (More than expected, definitely more than required):   1.25/1.00
Very Good (Significantly more than required):                    1.12/1.00
Good (Complete/exact conformance to expectations):               1.00/1.00
OK (Acceptable, usable but most certainly 'improvable'):         0.75/1.00
Average (Still acceptable, but somewhat distracting):            0.50/1.00
Poor (You know what I mean!):                                    0.25/1.00
```

Worthless	0.00/1.00

Is the interface natural?
(or does it take too much effort to adopt to it?): Yes (1.00)
Consistent? (if not, why/how?): Yes.(1.00)

Informational messages: Very Good.(1.12)

Behaviour during fatal errors(is it OS which is basically doing this?):
Acceptable.

COMMENTS & OBSERVATIONS:

Messages (both error and informational) are very good.
It would have been more useful to distinguish between an Invalid versus Restricted command(0.00).

Spaces in input are allowed, so it was very convenient to use(1.00).

User input checking is very good (especially in case of ADD)(1.12).

Special mention should be made of the formatting of the system output, messages etc. (1.25)

The environment is set up (initialized) very well (with over 60 users and 10 groups) which is quite helpful in testing and it also demonstrates that the system is built professionally and that it can handle the scene in a real application too. (1.25)

The basis for this impression is that the developers took good care to see that the system's output is displayed in an orderly, screen-by-screen fashion to handle output of more than one screen(1.12).

One major negative factor:

The file handling of the system is poor.

The return values of Opens, Reads etc., of files are not checked and hence any error occurred during file handling manifests itself as a fatal failure (rather than a soft failure). (0.00)

Also, the file locations are hard-coded (with their entire path specified) so, I had to edit the code and recompile it to be able to run the system in my directory!! (0.00)

Limitations imposed on user:
1. The system IS case-sensitive? NO (1.00)
2 The subject field in messages being sent is limited to certain number of characters and the anything more than that limit carries over to message text. Which is OK, I mean it doesn't barf if you exceed that length limit for subject, but it doesn't let you type any message text (it just simply takes the excess over characters from the subject header and sends it as the text of the message. (1.00)

-:DETAILS:-

1. Operational Characteristics:

FAILURE DATA:

No.	Schedule	Test Case No.	CPU Interval	Severity Code	Duplicate failure?	Comment
1.	II-#4	2	27.63	3	No	SEND, w/ subject more than a certain limit, didn't wait for user's input sent the rest of the subject itself as message text
2.		LAST	3.46	-	-	The last projected failure

2. Implementation Completeness:

2.i. Functional Requirements:

Non-working/Partially-working functions: SEND
Comments:

Function	Total No. of Attrib.	No. of Working Attrib.	Normal-ized IC Metric	Selby's IC Metric (Max.= 2)	Relevant Test Cases	Comments
SIGNON	2	2	2	2		
SIGNOFF	2	2	2	2		
ADD	3	3	2	2	1,3-6,18,19	
DELETE (user)	3	2	2	2	20,29,30	
AUTHORIZE	3	3	2	2	17 - 24	
NAMES	2	2	2	2	14,19,#1	
SEND (To an individual)	2	1	1	1	#2	When subject is long message text is not sent.
SEND (To a group)	2	2	2	2	#1	
READ	3	3	2	2	#1, #2	
FIND	4	4	2	2	#3	
HOLD	2	1	2	2	#1	
DELETE after read/find msg.	2	1	2	2	#1	
RESPOND	2	2	2	2	#2,#3	

RESET	1	1	2	2	#1
GROUPADD	8	8	2	2	8,9,13,21, 24,27,33,34, 35,#1
GROUPDELETE	3	3	2	2	22,37,40
GROUPQUERY	3	3	2	2	27,32,38,#1 #3
Non-functional:					
Case-sensitivity	2	2	2	2	
TOTAL	46	44	35/36	35/36	
			NAAGESH's metric	Selby's metric	

```
IMPORTANT SOURCE FILES:
WS3:<<UGANTI.THESIS.ADAGROUP2]: dir/date/size=all *.ada;

Directory SYS_USERS:[SAN.NAORUGANTI.THESIS.ADAGROUP2]

ADACOM.ADA;1              9/9       29-APR-1988 02:33:14.00
GROUPS.ADA;71            41/41      21-FEB-1992 20:15:55.00
INIT.ADA;3                2/2       22-APR-1988 04:55:48.00
INIT2.ADA;4               1/1       22-APR-1988 04:58:08.00
INIT3.ADA;7               3/3       21-FEB-1992 18:45:13.00
MAIN.ADA;14              93/93       3-MAY-1988 15:59:13.00
MESSAGES.ADA;21          23/23      21-FEB-1992 20:14:01.00
USERS.ADA;5              28/28       3-MAY-1988 16:02:55.00
UTILITY.ADA;37           12/12       3-MAY-1988 16:00:56.00

Total of 9 files, 212/212 blocks.


IMPORTANT EXECUTABLES & ASSOCIATED DATA FILES:

Directory SYS_USERS:[SAN.NAORUGANTI.THESIS.ADAGROUP2]

INIT3.EXE;3              35/35      21-FEB-1992 20:22:39.00
MAIN.EXE;194            137/137     21-FEB-1992 20:24:14.00
and
All files in the directory:
Directory SYS_USERS:[SAN.NAORUGANTI.THESIS.ADAGROUP2.EMS_FILES]

===============================================================================
```