Technical Report TR-1415 February 1985

CLEANROOM Software Development: An Empirical Evaluation

Richard W. Selby, Jr.
Victor R. Basili
F. Terry Baker

Department of Computer Science
University of Maryland
College Park

KEYWORDS: software development methodology, off-line software review, software measurement, methodology evaluation, software management, empirical study

Research supported in part by the AFOSR Contract AFOSR-F 49620-80-C-001 to the University of Maryland. Computer support provided in part by the Computer Science Center at the University of Maryland.

ABSTRACT

The Cleanroom software development approach is intended to produce highly reliable software by integrating formal methods for specification and design, complete off-line development, and statistically based testing. In an empirical study, 15 three-person teams developed versions of the same software system (800 – 2300 source lines); ten teams applied Cleanroom, while five applied a more traditional approach. This analysis characterizes the effect of Cleanroom on the delivered product, the software development process, and the developers. The major results of this study are 1) most developers were able to apply the techniques of Cleanroom effectively; 2) the Cleanroom teams' products met system requirements more completely and had a higher percentage of successful test cases; 3) the source code developed using Cleanroom had more comments and less dense complexity; 4) the use of Cleanroom successfully modified aspects of development style; and 5) most Cleanroom developers indicated they would use the approach again.

Table of Contents

1. Introduction

The need for discipline in the software development process and for high quality software motivates the Cleanroom software development approach. In addition to improving the control during development, this approach is intended to deliver a product that meets several quality aspects: a system that conforms with the requirements, a system with high operational reliability, and source code that is easily readable and modifiable.

Section II describes the Cleanroom approach and a framework of goals for characterizing its effect. Section III presents an empirical study using the approach. Section IV gives the results of the analysis comparing projects developed using Cleanroom with those of a control group. The overall conclusions appear in Section V.

2. Cleanroom Software Development

The Federal Systems Division of IBM [Dyer 82, Dyer & Mills 82] presents the Cleanroom software development method as a technical and organizational approach to developing software with certifiable reliability. The idea is to deny the entry of defects during the development of software, hence the term "Cleanroom." The focus of the method is imposing discipline on the development process by integrating formal methods for specification and design, complete off-line development, and statistically based testing. These components are intended to contribute to a software product that has a high probability of zero defects and consequently a high measure of operational reliability.

The mathematically-based design methodology of Cleanroom includes the use of structured specifications and state machine models [Ferrentino & Mills 77]. A systems

engineer introduces the structured specifications to restate the system requirements precisely and organize the complex problems into manageable parts [Parnas 72]. The specifications determine the "system architecture" of the interconnections and groupings of capabilities to which state machine design practices can be applied. System implementation and test data formulation can then proceed from the structured specifications independently.

The right-the-first-time programming methods used in Cleanroom are the ideas of functionally based programming in [Mills 72b, Linger, Mills & Witt 79]. The testing process is completely separated from the development process by not allowing the developers to test and debug their programs. The developers focus on the techniques of code inspections [Fagan 76], group walkthroughs [Myers 76], and formal verification [Hoare 69, Linger, Mills & Witt 79, Shankar 82, Dyer 83] to assert the correctness of their implementation. These constructive techniques apply throughout all phases of development, and condense the activities of defect detection and isolation into one operation. This discipline is imposed with the intention that correctness is "designed" into the software, not "tested" in. The notion that "Well, the software should always be tested to find the faults" is eliminated.

In the statistically based testing strategy of Cleanroom, independent testers simulate the operational environment of the system with random testing. This testing process includes defining the frequency distribution of inputs to the system, the frequency distribution of different system states, and the expanding hierarchy of developed system capabilities. Test cases then are chosen randomly and presented to the series of product releases, while concentrating on functions most recently delivered and maintaining the

overall composite distribution of inputs. The independent testers then record observed failures and determine an objective measure of product reliability. It is believed that the prior knowledge that a system will be evaluated by random testing will affect system reliability by enforcing a new discipline into the system developers.

2.1. Investigation Goals

Some intriguing aspects of the Cleanroom approach include 1) development without testing and debugging of programs, 2) independent program testing for quality assurance (rather than to find faults or to prove "correctness" [Howden 76]), and 3) certification of system reliability before product delivery. In order to understand the effects of using Cleanroom, the following three goals are proposed: 1) characterize the effect of Cleanroom on the delivered product, 2) characterize the effect of Cleanroom on the software development process, and 3) characterize the effect of Cleanroom on the developers. An application of the goal/question/metric paradigm [Basili & Selby 84, Basili & Weiss 84] leads to the framework of goals and questions for this study appearing in Figure 1. The empirical study executed to pursue these goals is described in the following section.

Figure 1. Framework of goals and questions for Cleanroom development approach analysis.

I. Characterize the effect of Cleanroom on the delivered product.
  A. For intermediate and novice programmers building a small system, what were the operational properties of the product?
    1. Did the product meet the system requirements?
    2. How did the operational testing results compare with those of a control group?
  B. What were the static properties of the product?
    1. Were the size properties of the product any different from what would be observed in a traditional development?
    2. Were the readability properties of the product any different?
    3. Was the control complexity any different?
    4. Was the data usage any different?
    5. Was the implementation language used any differently?
  C. What contribution did programmer background have on the final product quality?

II. 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 they meet their delivery schedule?

III. Characterize the effect of Cleanroom on the developers.
  A. When intermediate and novice programmers built a small system, did the developers miss the satisfaction of executing their own programs?
    1. Did the missing of program execution have any relationship to programmer background or to 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 they use Cleanroom again?

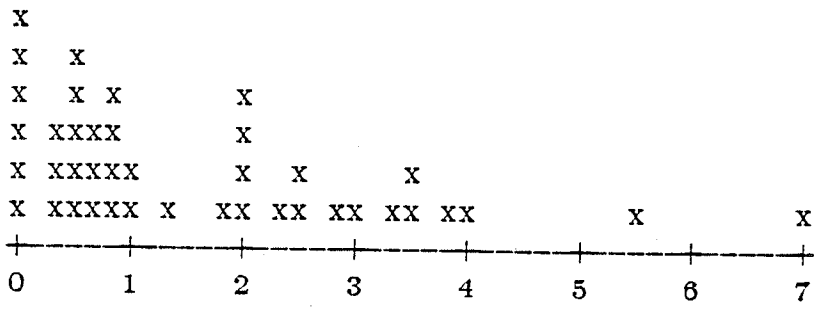
3. Empirical Study Using Cleanroom

This section describes an empirical study comparing team projects developed using Cleanroom with those using a more conventional approach.

3.1. Case Study Description

Subjects for the empirical study came from the "Software Design and Development" course taught by F. T. Baker and V. R. Basili at the University of Maryland in the Falls of 1982 and 1983. The initial segment of the course was devoted to the presentation of several software development methodologies, including top-down design, modular specification and design, PDL, chief programmer teams, program correctness, code reading, walkthroughs, and functional and structural testing strategies. For the latter part of the course, the individuals were divided into three-person chief programmer teams for a group project [Baker 72, Mills 72a, Baker 81]. We attempted to divide the teams equally according to professional experience, academic performance, and implementation language experience. The subjects had an average of 1.6 years professional experience and were computer science majors with junior, senior, or graduate standing. Figure 2 displays the distribution of the subjects' professional experience.

Figure 2. Subjects' professional experience in years.



A requirements document for an electronic message system (read, send, mailing lists, authorized capabilities, etc.) was distributed to each of the teams. The project was to be completed in six weeks and was expected to be about 1200 lines of Simpl-T source

[Basili & Turner 76]. [1] The development machine was a Univac 1100/82 running EXEC VIII, with 1200 baud interactive and remote access available.

The ten teams in the Fall 1982 course applied the Cleanroom software development approach, while the five teams in the Fall 1983 course served as a control group (non-Cleanroom). All other aspects of the developments were the same. The two groups of teams were not statistically different in terms of professional experience, academic performance, or implementation language experience. If there were any bias between the two times the course was taught, it would be in favor of the 1983 (non-Cleanroom) group because the modular design portion of the course was presented earlier. It was also the second time F. T. Baker had taught the course. Note that the teams in the non-Cleanroom group applied a development approach similar to the "disciplined team" approach examined in an earlier study [Basili & Reiter 81].

The first document every team in either group turned in contained a system specification, composite design diagram, and implementation plan. The latter element was a series of milestones describing when the various functions within the system would be available. At these various dates (minimum one week apart, maximum two), teams from both groups would then submit their systems for testing. An independent party would then apply statistically based testing to each of these deliveries and report to the team members both the successful and unsuccessful test cases. The latter would

[1] Simpl-T is a structured language that supports several string and file handling primitives, in addition to the usual control flow constructs available, for example, in Pascal. If Pascal or FORTRAN had been chosen, it would have been very likely that some individuals would have had extensive experience with the language, and this would have biased the comparison. Also, restricting access to a compiler that produced executable code would have been very difficult.

be included in the next test session for verification. Recall that the Cleanroom teams could not execute their programs – they had editing and syntax-checking capabilities only. They had to rely on the techniques of code reading, structured walkthroughs, and inspections to prepare their programs before submission. On the other hand, the non-Cleanroom teams had full access to compilation and execution facilities to test their systems prior to independent testing.

All team projects were evaluated on the use of the development techniques presented in class, the independent testing results, and a final oral interview. In addition to these sources, information on the team projects was collected from a background questionnaire, a postdevelopment attitude survey, static source code analysis, and operating system statistics. The following section briefly describes the operationally based testing process applied to all projects by the independent tester.

3.2. Operational Testing of Projects

The testing approach used in Cleanroom is to simulate the developing system's environment by randomly selecting test data from an "operational profile," a frequency distribution of inputs to the system [Thayer, Lipow & Nelson 78, Duran & Ntafos 81]. The projects from both groups were tested interactively at the milestones chosen by each team by an independent party (i.e., R. W. Selby). A distribution of inputs to the system was obtained by identifying the logical functions in the system and assigning each a frequency. This frequency assignment was accomplished by polling eleven well-seasoned users of the University of Maryland Vax 11/780 mailing system. Then test data were generated randomly from this profile and presented to the system. Recording

of failure severity and times between failure took place during the testing process. The operational statistics referred to later were calculated from fifty user-session test cases run on the final system release of each team. For a complete explanation of the operationally based testing process applied to the projects, including test data selection, testing procedure, and failure observation, see [Selby 84].

4. Data Analysis and Interpretation

The analysis and interpretation of the data collected from the study appear in the following sections, organized by the goal areas outlined earlier. In order to address the various questions posed under each of the goals, some raw data usually will be presented and then interpreted. Figure 3 presents the number of source lines, executable statements, and procedures and functions to give a rough view of the systems developed.

Figure 3. System statistics.

Team	Cleanroom	Source Lines	Executable Statments	Procedures & Functions
A	yes	1681	813	55
B	yes	1626	717	42
C	yes	1118	573	42
D	yes	1046	477	30
E	yes	1087	624	32
F	yes	1213	440	35
G	yes	1196	581	31
H	yes	1876	550	51
I	yes	1305	608	23
J	yes	1052	658	24
a	no	824	410	26
b	no	1429	633	18
c	no	2264	999	46
d	no	1629	626	67
e	no	1310	459	43

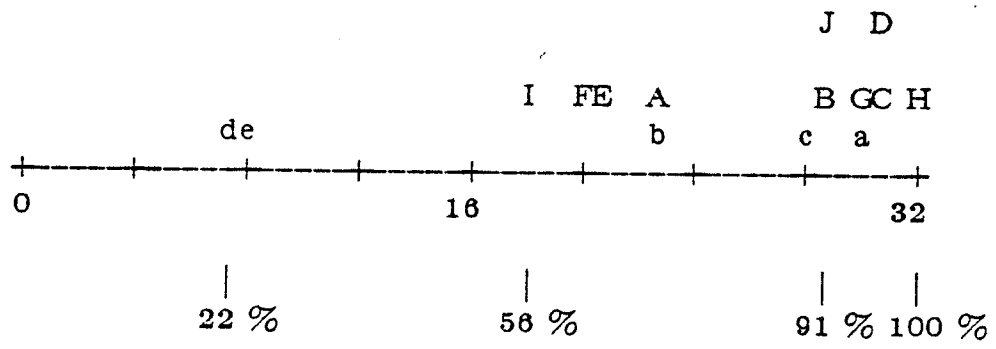
4.1. Characterization of the Effect on the Product Developed

This section characterizes the differences between the products delivered by both of the development groups. Initially we examine some operational properties of the products, followed by a comparison of some of their static properties.

4.1.1. Operational System Properties

In order to contrast the operational properties of the systems delivered by the two groups, both completeness of implementation and operational testing results were examined. A measure of implementation completeness was calculated by partitioning the required system into sixteen logical functions (e.g., send mail to an individual, read a piece of mail, respond, add yourself to a mailing list, ...). 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 was calculated; a maximum score of 32 was possible. Figure 4 displays this subjective measure of requirement conformance for the systems. Note that in all figures presented, the ten teams using Cleanroom are in upper case and the five teams using a more conventional approach are in lower case. A first observation is that six of the ten Cleanroom teams built very close to the entire system. While not all of the Cleanroom teams performed equally well, a majority of them applied the approach effectively enough to develop nearly the whole product. More importantly, the Cleanroom teams met the requirements of the system more completely than did the non-Cleanroom teams.

Figure 4. Requirement conformance of the systems.



Mann-Whitney [2] signif. = .088

To compare testing results among the systems developed in the two groups, fifty random user-session test cases were executed on the final release of each system to simulate its operational environment. If the final release of a system performed to expectations on a test case, the outcome was called a "success;" if not, the outcome was a "failure." If the outcome was a "failure" but the same failure was observed on an earlier test case run on the final release, the outcome was termed a "duplicate failure." Figure 5 shows the percentage of successful test cases when duplicate failures are not included. The figure displays that Cleanroom projects had a higher percentage of successful test cases at system delivery. [3] When duplicate failures are included, however, the better performance of the Cleanroom systems is not nearly as significant (MW = .134). [4] This is caused by the Cleanroom projects having a relatively higher proportion of duplicate

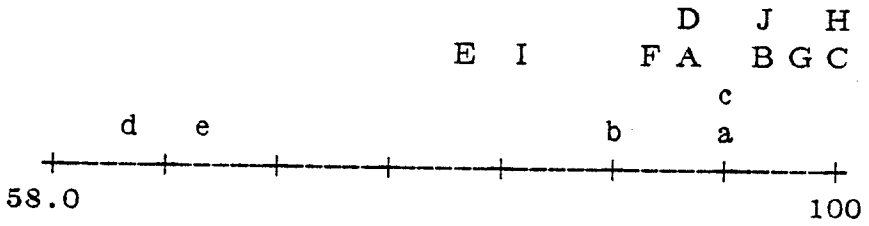
[2] The significance levels for the Mann-Whitney statistics reported are the probability of Type I error in an one-tailed test.

[3] Although not considered here, various software reliability models have been proposed to forecast system reliability based on failure data [Musa 75, Currit 83, Goel 83].

[4] To be more succinct, MW will sometimes be used to abbreviate the significance level of the Mann-Whitney statistic.

failures, even though they did better overall. This demonstrates that while reviewing the code, the Cleanroom developers focused less than the other groups on certain parts of the system. The more uniform review of the whole system makes the performance of the system less sensitive to its operational profile. Note that operational environments of systems are usually difficult to define a priori and are subject to change.

Figure 5. Percentage of successful test cases during operational testing (without duplicate failures).



Mann-Whitney signif. = .055

In both of the product quality measures of implementation completeness and operational testing results, there was quite a variation in performance.[5] A wide variation may have been expected with an unfamiliar development technique, but the developers using a more traditional approach had a wider range of performance than did those using Cleanroom in both of the measures (even with twice as many Cleanroom teams). All of the above differences are magnified by recalling that the non-Cleanroom teams did not develop their systems in one monolithic step, they (also) had the benefit of periodic

[5] An alternate perspective includes only the more successful projects from each group in the comparison of operational product quality. When the best 60% from each approach are examined (i.e., removing teams 'd,' 'e,' 'A,' 'E,' 'F,' and 'I'), the Mann-Whitney significance level for comparing implementation completeness becomes .045 and the significance level for comparing successful test cases (without duplicate failures) becomes .034. Thus, comparing the best teams from each approach increases the evidence in favor of Cleanroom in both of these product quality measures.

operational testing by independent testers. Since both groups of teams had independent testing of all their deliveries, the early testing of deliveries must have revealed most faults overlooked by the Cleanroom developers.

These comparisons suggest that the non-Cleanroom developers focused on a "perspective of the tester," sometimes leaving out classes of functions and causing a less completely implemented product and more (especially unique) failures. Off-line review techniques, however, are more general and their use contributed to more complete requirement conformance and fewer failures in the Cleanroom products. In addition to examining the operational properties of the product, various static properties were compared.

4.1.2. Static System Properties

The first question in this goal area concerns the size of the final systems. Figure 3 showed the number of source lines, executable statements, and procedures and functions for the various systems. The projects from the two groups were not statistically different (MW $>$ .10) in any of these three size attributes. Another question in this goal area concerns the readability of the delivered source code. Two aspects of reading and modifying code are the number of comments present and the density of the "complexity." In an attempt to capture the complexity density, syntactic complexity [Basili & Hutchens 83] was calculated and normalized by the number of executable statements. In addition to control complexity, the syntactic complexity metric considers nesting depth and prime program decomposition [Linger, Mills & Witt 79]. The developers using Cleanroom wrote code that was more highly commented (MW = .089) and had a

lower complexity density (MW = .079) than did those using the traditional approach. A calculation of either software science effort [Halstead 77], cyclomatic complexity [McCabe 76], or syntactic complexity without any size normalization, however, produced no significant differences (MW > .10). This seems as expected because all the systems were built to meet the same requirements.

Comparing the data usage in the systems, Cleanroom developers used a greater number of global data items (MW = .071). Also, Cleanroom projects possessed a higher percentage of assignment statements (MW = .056). These last two observations could be a manifestation of teaching the Cleanroom subjects modular design later in the course (see Case Study Description), or possibly an indication of using the approach.

Some interesting observations surface when the operational quality measures of the Cleanroom products are correlated with the usage of the implementation language. Both percentage of successful test cases (without duplicate failures) and implementation completeness correlated with percentage of procedure calls (Spearman R = .65, signif. = .044, and R = .57, signif. = .08, respectively) and with percentage of if statements (R = .62, signif. = .058, and R = .55, signif. = .10, respectively). However, both of these two product quality measures correlated negatively with percentage of case statements (R = −.86, signif. = .001, and R = −.69, signif. = .027, respectively) and with percentage of while statements (R = −.65, signif. = .044, and R = −.49, signif. = .15, respectively). There were also some negative correlations between the product quality measures and the average software science effort per subroutine (R = −.52, signif. = .12, and R = −.74, signif. = .013, respectively) and the average number of occurrences of a variable (R = −.54, signif. = .11, and R = −.56, signif. = .09, respectively).

Considering the products from all teams, both percentage of successful test cases (without duplicate failures) and implementation completeness had some correlation with percentage of if statements (R = .48, signif. = .07, and R = .45, signif. = .09, respectively) and some negative correlation with percentage of case statements (R = −.48, signif. = .07, and R = −.42, signif. = .12, respectively). Neither of the operational product quality measures correlated with percentage of assignment statements when either all products or just Cleanroom products were considered. These observations suggest that the more successful Cleanroom developers simplified their use of the implementation language; i.e., they used more procedure calls and if statements, used fewer case and while statements, had a lower frequency of variable reuse, and wrote subroutines requiring less software science effort to comprehend.

4.1.3. Contribution of Programmer Background

When examining the contribution of the Cleanroom programmers' background to the quality of their final products, general programming language experience correlated with percentage of successful operational tests (without duplicate failures: Spearman R = .66, signif. = .04; with duplicates: R = .70, signif. = .03) and with implementation completeness (R = .55; signif. = .10). No relationship appears between either operational testing results or implementation completeness and either professional[6] or testing experience. These background/quality relations seem consistent with other studies [Curtis 83].

---

[6] In fact, there are very slight negative correlations between years of professional experience and both percentage of successful tests (without duplicate failures: R = −.46, signif. = .18) and implementation completeness (R = − .47, signif. = .17).

4.1.4. Summary of the Effect on the Product Developed

In summary, Cleanroom developers delivered a product that 1) met system requirements more completely, 2) had a higher percentage of successful test cases, 3) had more comments and less dense complexity, and 4) used more global data items and a higher percentage of assignment statements. The more successful Cleanroom developers 1) used more procedure calls and if statements, 2) used fewer case and while statements, 3) reused variables less frequently, 4) developed subroutines requiring less (software science) effort to comprehend, and 5) had more general programming language experience.

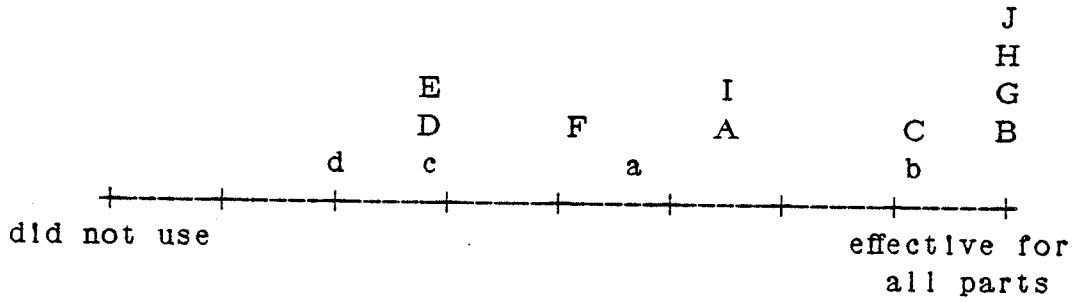
4.2. Characterization of the Effect on the Development Process

In a postdevelopment attitude survey, the developers were asked how effectively they felt they applied off-line review techniques in testing their projects (see Figure 6). This was an attempt to capture some of the information necessary to answer the first question under this goal (question II.A). In order to make comparisons at the team level, the responses from the members of a team are composed into an average for the team. The responses to the question appear on a team basis in a histogram in the second part of the figure. Of the Cleanroom developers, teams 'A,' 'D,' 'E,' 'F,' and 'I' were the least confident in their use of the off-line review techniques and these teams also performed the worst in terms of operational testing results; four of these five teams performed the worst in terms of implementation completeness. Off-line review effectiveness correlated with percentage of successful operational tests (without duplicate failures) for the Cleanroom teams (Spearman R = .74; signif. = .014) and for all the teams (R = .76; signif. = .001); it correlated with implementation completeness for all

the teams (R = .58; signif. = .023). Neither professional nor testing experience correlated with off-line review effectiveness when either all teams or just Cleanroom teams were considered.

Figure 6. Breakdown of responses to the attitude survey question, "Did you feel that you and your team members effectively used off-line review techniques in testing your project?". (Responses are from Cleanroom teams.) [7]

14 – Yes, they were effective for testing all parts of the program
5.5 – We used them but felt that they were only appropriate for certain parts of the program
8.5 – We used them occasionally, but they were not really a major contributing factor to the development
0 – Did not really use them at all



Mann-Whitney signif. = .065

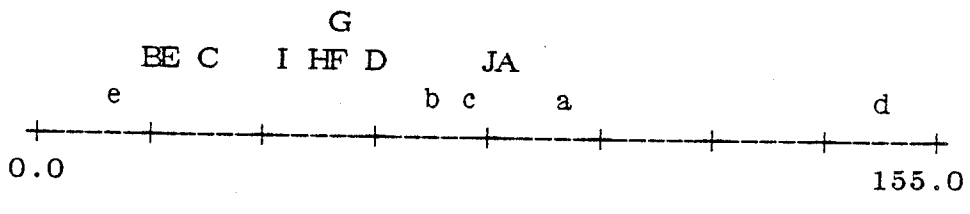
The histogram in Figure 6 shows that the Cleanroom developers felt they applied the off-line review techniques more effectively than did the non-Cleanroom teams. The non-Cleanroom developers were asked to give a relative breakdown of the amount of time spent applying testing and verification techniques. Their aggregate response was 39% off-line review, 52% functional testing, and 9% structural testing. From this breakdown, we observe that the non-Cleanroom teams primarily relied on functional testing to prepare their systems for independent testing. Since the Cleanroom teams

[7] There are half-responses because an individual checked both the second and third choices. The responses total to 28, not 30, because two separate teams lost a member late in the project. (See Distinction Among Teams).

were unable to rely on testing methods, they may have (felt they had) applied the off-line review techniques more effectively.

Since the role of the computer is more controlled when using Cleanroom, one would expect a difference in on-line activity between the two groups. Figure 7 displays the amount of connect time that each of the teams cumulatively used. A comparison of the cpu-time used by the teams was less statistically significant (MW = .110). Neither of these measures of on-line activity related to how effectively a team felt they had used the off-line techniques when either all teams or just Cleanroom teams were considered. Although non-Cleanroom team 'd' did a lot of on-line testing and non-Cleanroom team 'e' did little, both teams performed poorly in the measures of operational product quality discussed earlier. The operating system of the development machine captured these system usage statistics. Note that the time the independent party spent testing is included. [8] These observations exhibit that Cleanroom developers spent less time on-line and used fewer computer resources. These results empirically support the reduced role of the computer in Cleanroom development.

Figure 7. Connect time in hours during project development. [9]

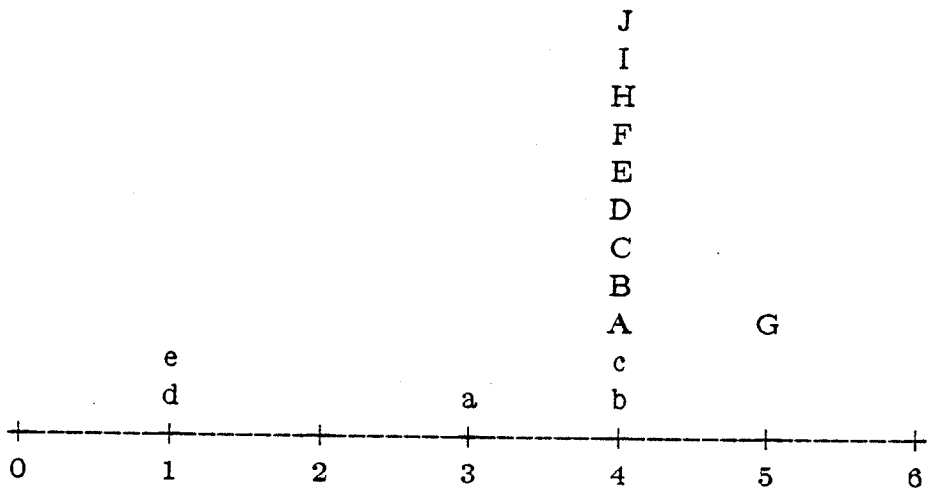


Mann-Whitney signif. = .089

[8] When the time the independent tester spent is not included, the significance levels for the non-parametric statistics do not change.

Schedule slippage continues to be a problem in software development. It would be interesting to see whether the Cleanroom teams demonstrated any more discipline by maintaining their original schedules. All of the teams from both groups planned four releases of their evolving system, except for team 'G' which planned five. Recall that at each delivery an independent party would operationally test the functions currently available in the system, according to the team's implementation plan. In Figure 8, we observe that all the teams using Cleanroom kept to their original schedules by making all planned deliveries; only two non-Cleanroom teams made all their scheduled deliveries.

Figure 8. Number of system releases.



Mann-Whitney signif. = .006

[9] Non-Cleanroom team 'e' entered a substantial portion of its system on a remote machine, only using the Univac computer mainly for compilation and execution. (See Distinction Among Teams.)

4.2.1. Summary of the Effect on the Development Process

Summarizing the effect on the development process, Cleanroom developers 1) felt they applied off-line review techniques more effectively, while non-Cleanroom teams focused on functional testing; 2) spent less time on-line and used fewer computer resources; and 3) made all their scheduled deliveries.

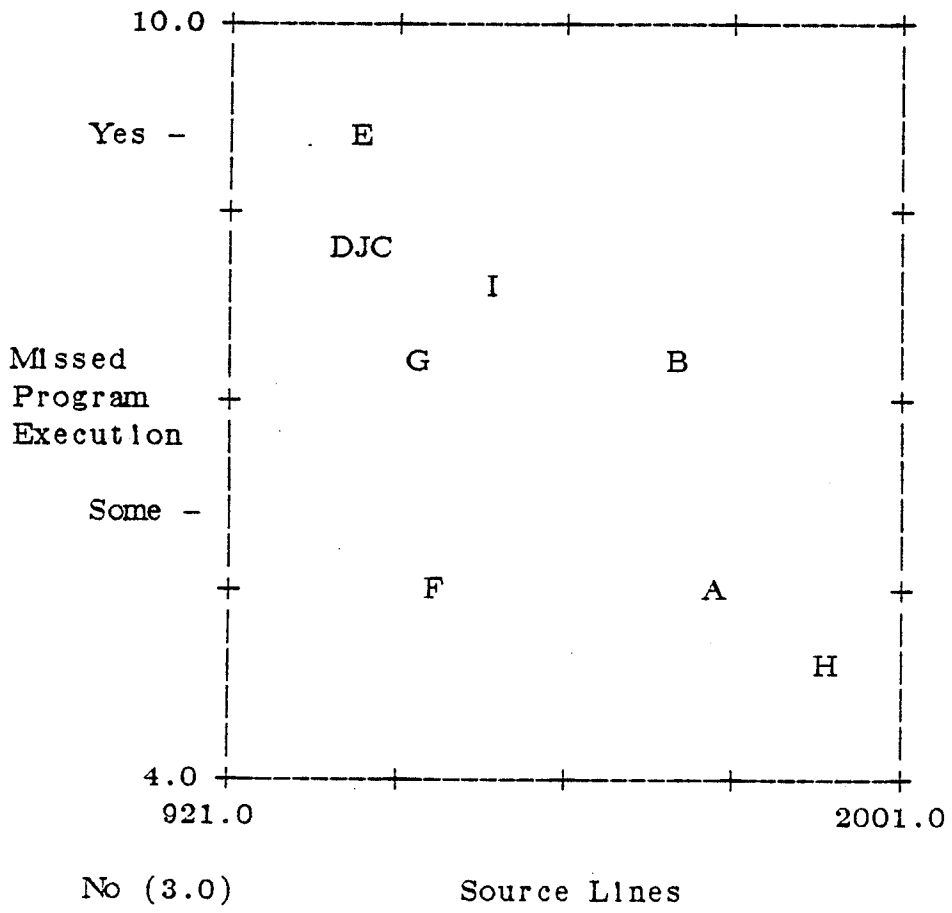
4.3. Characterization of the Effect on the Developers

The first question posed in this goal area is whether the individuals using Cleanroom missed the satisfaction of executing their own programs. Figure 9 presents the responses to a question included in the postdevelopment attitude survey on this issue. As might be expected, almost all the individuals missed some aspect of program execution. As might not be expected, however, this missing of program execution had no relation to either the product quality measures mentioned earlier or the teams' professional or testing experience. Also, missing program execution did not increase with respect to program size (see Figure 10).

Figure 9. Breakdown of responses to the attitude survey question, "Did you miss the satisfaction of executing your own programs?".

13 – Yes, I missed the satisfaction of program execution.
11 – I somewhat missed the satisfaction of program execution.
4 – No, I did not miss the satisfaction of program execution.

Figure 10. Relationship of program size vs. missing program execution.



Spearman correlations: –.85 (signif. = .002) with source lines; –.70 (signif. = .03) with number separately compilable modules; –.57 (signif. = .09) with number procedures and functions.

Figure 11 displays the replies of the developers when they were asked how their design and coding style was affected by not being able to test and debug. At first it would seem surprising that more people did not modify their development style when

applying the techniques of Cleanroom. Several persons mentioned, however, that they already utilized some of the ideas in Cleanroom. Keeping a simple design supports readability of the product and facilitates the processes of modification and verification. Although some of the objective product measures presented earlier showed differences in development style, these subjective ones are interesting and lend insight into actual programmer behavior.

Figure 11.
Breakdown of responses to the attitude survey question, "How was your design and coding style affected by not being able to test and debug?".

2 – Yes, my style was substantially revised. 15 – I modified some of my tendencies. 11 – It did not affect my style at all.
Frequently mentioned responses include – kept design simple, attempted nothing fancy – kept readability of code in mind – already was a user of off-line review techniques – very careful scrutiny of code for potential mistakes – prepared for a larger range of inputs

One indicator of the impression that something new leaves on people is whether they would do it again. Figure 12 presents the responses of the individuals when they were asked whether they would choose to use Cleanroom as either a software development manager or as a programmer. Even though these responses were gathered (immediately) after course completion, subjects desiring to "please the instructor" may have responded favorably to this type of question regardless of their true feelings. Practically everyone indicated a willingness to apply the approach again. It is interesting to note that a greater number of persons in a managerial role would choose to always use it. Of the persons that ranked the reuse of Cleanroom fairly low in each category, four

of the five were the same people. Of the six people that ranked reuse low, four were from less successful projects (one from team 'A', one from team 'E' and two from team 'I'), but the other two came from reasonably successful developments (one from team 'C' and one from team 'J'). The particular individuals on teams 'E,' 'I,' and 'J' rated the reuse fairly low in both categories.

Figure 12.
Breakdown of responses to the attitude survey question, "Would you use Cleanroom again?". (One person did not respond to this question.)

As a software development manager?
- 8 – Yes, at all times
- 14 – Yes, but only for certain projects
- 5 – Not at all

As a programmer?
- 4 – Yes, for all projects
- 18 – Yes, but not all the time
- 5 – Only if I had to
- 0 – I would leave if I had to

4.3.1. Summary of the Effect on the Developers

In summary of the effect on the developers, most Cleanroom developers 1) modified in part their development style, 2) missed program execution, and 3) indicated they would use the approach again.

4.4. Distinction Among Teams

In spite of efforts to balance the teams according to various factors (see Case Study Description), a few differences among the teams were apparent. Two separate Cleanroom teams, 'H' and 'I,' each lost a member late in the project. Thus at project completion, there were eight three-person and two two-person Cleanroom teams. Recall that

team 'H' performed quite well according to requirement conformance and testing results, while team 'I' did poorly. Also, the second group of subjects did not divide evenly into three-person teams. Since one of those individuals had extensive professional experience, non-Cleanroom team 'e' consisted of that one highly experienced person. Thus at project completion, there were four three-person and one one-person non-Cleanroom teams. Although team 'e' wrote over 1300 source lines, this highly experienced person did not do as well as the other teams in some respects. This is consistent with another study in which teams applying a "disciplined methodology" in development outperformed individuals [Basili & Reiter 81]. Appendix A contains the significance levels for the above results when team 'e,' when teams 'H' and 'I,' and when teams 'e,' 'H,' and 'I' are removed from the analysis. Removing teams 'H' and 'I' has little effect on the significance levels, while the removal of team 'e' causes a decrease in all of the significance levels except for executable statements, software science effort, cyclomatic complexity, syntactic complexity, connect-time, and cpu-time.

5. Conclusions

This paper describes "Cleanroom" software development – an approach intended to produce highly reliable software by integrating formal methods for specification and design, complete off-line development, and statistically based testing. The goal structure, experimental approach, data analysis, and conclusions are presented for a replicated-project study examining the Cleanroom approach. This is the first investigation known to the authors that applied Cleanroom and characterized its effect relative to a more traditional development approach.

The data analysis presented and the testimony provided by the developers suggest that the major results of this study are 1) most developers were able to apply the techniques of Cleanroom effectively; 2) the Cleanroom teams' products met system requirements more completely and had a higher percentage of successful test cases; 3) the source code developed using Cleanroom had more comments and less dense complexity; 4) the use of Cleanroom successfully modified aspects of development style; and 5) most Cleanroom developers indicated they would use the approach again.

It seems that the ideas in Cleanroom help attain the goals of producing high quality software and increasing the discipline in the software development process. The complete separation of development from testing appears to cause a modification in the developers' behavior, resulting in increased process control and in more effective use of formal methods for software specification, design, off-line review, and verification. It seems that system modification and maintenance would be more easily done on a product developed in the Cleanroom method, because of the product's thoroughly conceived design and higher readability. Thus, achieving high requirement conformance and high operational reliability coupled with low maintenance costs would help reduce overall costs, satisfy the user community, and support a long product lifetime.

This empirical study is intended to advance the understanding of the relationship between introducing discipline into the development process (as in Cleanroom) and several aspects of product quality: conformance with requirements, high operational reliability, and easily modifiable source code. The results given were calculated from a set of teams applying Cleanroom development on a relatively small project – the direct extrapolation of the findings to other projects and development environments is not

implied. Valuable insights, however, have been gained from the analysis.

6. Acknowledgement

The authors are grateful to D. H. Hutchens and R. W. Reiter for the use of their analysis program in this study.

7. Appendix A.

Figure 13 presents the measure averages and the significance levels for the above comparisons when team 'e,' when teams 'H' and 'I,' and when teams 'e,' 'H,' and 'I' are removed. The significance levels for the Mann-Whitney statistics reported are the probability of Type I error in an one-tailed test.

Figure 13. Summary of measure averages and significance levels.

Measure	Average		Mann-Whitney significance levels			
	Cleanroom Teams	Non-Cleanroom Teams	All Teams	Without Team e	Without Teams H,I	Without Teams e,H,I
Source lines	1320.0	1491.2	.196	.240	.153	.198
Executable stmts	604.1	625.4	.500	.286	.442	.367
#Procedures & functions	36.5	40.0	.357	.500	.330	.500
%Implementation completeness	82.5	60.0	.088	.197	.093	.196
%Successful tests (w/o duplicate failures)	92.5	80.8	.055	.128	.053	.116
%Successful tests (w/ duplicate failures)	78.7	59.2	.134	.285	.151	.304
#Comments	194.9	122.2	.089	.102	.190	.198
Syntactic complexity/ executable stmts	1.5	1.6	.079	.179	.082	.175
Software Science E	6728.6e3	7355.4e3	.451	.240	.442	.248
Cyclomatic complexity	196.8	212.2	.250	.198	.255	.248
Syntactic complexity	917.5	1017.0	.500	.286	.500	.305
#Global data items	37.6	24.2	.071	.129	.053	.117
%Assignment stmts	34.2	26.6	.056	.129	.040	.087
Off-line effectiveness	3.2	2.5	.065	.065	.098	.098
Connect-time (hr.)	41.0	71.3	.089	.012	.121	.021
Cpu-time (min.)	71.7	136.1	.110	.017	.072	.009
#Deliveries	4.1	2.6	.006	.015	.010	.022

8. References

[Baker 72]

F. T. Baker, Chief Programmer Team Management of Production Programming, *IBM Systems J.* **11**, 1, pp. 131-149, 1972.

[Baker 81]

F. T. Baker, Chief Programmer Teams, pp. 249-254 in *Tutorial on Structured Programming: Integrated Practices,* ed. V. R. Basili and F. T. Baker, IEEE, 1981.

[Basili & Turner 76]

V. R. Basili and A. J. Turner, *SIMPL-T: A Structured Programming Language,* Paladin House Publishers, Geneva, IL, 1976.

[Basili & Reiter 81]

V. R. Basili and R. W. Reiter, A Controlled Experiment Quantitatively Comparing Software Development Approaches, *IEEE Trans. Software Engr.* **SE-7**, May 1981.

[Basili & Hutchens 83]

V. R. Basili and D. H. Hutchens, An Empirical Study of a Syntactic Metric Family, *Trans. Software Engr.* **SE-9**, 6, pp. 664-672, Nov. 1983.

[Basili & Selby 84]

V. R. Basili and R. W. Selby, Jr., Data Collection and Analysis in Software Research and Management, *Proceedings of the American Statistical Association and Biometric Society Joint Statistical Meetings,* Philadelphia, PA, August 13-16, 1984.

[Basili & Weiss 84]

V. R. Basili and D. M. Weiss, A Methodology for Collecting Valid Software Engineering Data*, *Trans. Software Engr.* **SE-10**, 6, pp. 728-738, Nov. 1984.

[Currit 83]

P. A. Currit, Cleanroom Certification Model, *Proc. Eight Ann. Software Engr. Workshop,* NASA/GSFC, Greenbelt, MD, Nov. 1983.

[Curtis 83]

B. Curtis, Cognitive Science of Programming, *Sixth Minnowbrook Workshop on Software Performance Evaluation,* Blue Mountain Lake, NY, July 19-22, 1983.

[Duran & Ntafos 81]
J. W. Duran and S. Ntafos, A Report on Random Testing*, *Proc. Fifth Int. Conf. Software Engr.*, San Diego, CA, pp. 179-183, March 9-12, 1981.

[Dyer & Mills 82]
M. Dyer and H. D. Mills, Developing Electronic Systems with Certifiable Reliability, *Proc. NATO Conf.*, Summer, 1982.

[Dyer 82]
M. Dyer, Cleanroom Software Development Method, IBM Federal Systems Division, Bethesda, MD, October 14, 1982.

[Dyer 83]
M. Dyer, Software Validation in the Cleanroom Development Method, IBM-FSD Tech. Rep. 86.0003, August 19, 1983.

[Fagan 76]
M. E. Fagan, Design and Code Inspections to Reduce Errors in Program Development, *IBM Sys. J.* **15**, 3, pp. 182-211, 1976.

[Ferrentino & Mills 77]
A. B. Ferrentino and H. D. Mills, State Machines and Their Semantics in Software Engineering, *Proc. IEEE COMPSAC*, 1977.

[Goel 83]
A. L. Goel, A Guidebook for Software Reliability Assessment, Dept. Industrial Engr. and Operations Research, Syracuse Univ., New York, Tech. Rep. 83-11, April 1983.

[Halstead 77]
M. H. Halstead, *Elements of Software Science*, North Holland, New York, 1977.

[Hoare 69]
C. A. R. Hoare, An Axiomatic Basis for Computer Programming, *Communications of the ACM* **12**, 10, pp. 576-583, Oct. 1969.

[Howden 76]
W. E. Howden, Reliability of the Path Analysis Testing Strategy, *IEEE Trans. Software Engr.* **SE-2**, 3, Sept. 1976.

[Linger, Mills & Witt 79]
R. C. Linger, H. D. Mills, and B. I. Witt, *Structured Programming: Theory and Practice*, Addison-Wesley, Reading, MA, 1979.

[McCabe 76]
T. J. McCabe, A Complexity Measure, *IEEE Trans. Software Engr.* SE-2, 4, pp. 308-320, Dec. 1976.

[Mills 72a]
H. D. Mills, Chief Programmer Teams: Principles and Procedures, IBM Corp., Gaithersburg, MD, Rep. FSC 71-6012, 1972.

[Mills 72b]
H. D. Mills, Mathematical Foundations for Structural Programming, IBM Report FSL 72-6021, 1972.

[Musa 75]
J. D. Musa, A Theory of Software Reliability and Its Application, *IEEE Trans. Software Engr.* SE-1, 3, pp. 312-327, 1975.

[Myers 76]
G. J. Myers, *Software Reliability: Principles & Practices,* John Wiley & Sons, New York, 1976.

[Parnas 72]
D. L. Parnas, On the Criteria to be Used in Decomposing Systems into Modules, *Communications of the ACM* 15, 12, pp. 1053-1058, 1972.

[Selby 84]
R. W. Selby, Jr., A Quantitative Approach for Evaluating Software Technologies, Dept. Com. Sci., Univ. Maryland, College Park, Ph. D. Dissertation, 1984.

[Shankar 82]
K. S. Shankar, A Functional Approach to Module Verification, *IEEE Trans. Software Engr.* SE-8, 2, March 1982.

[Thayer, Lipow & Nelson 78]
R. A. Thayer, M. Lipow, and E. C. Nelson, *Software Reliability,* North-Holland, Amsterdam, 1978.