

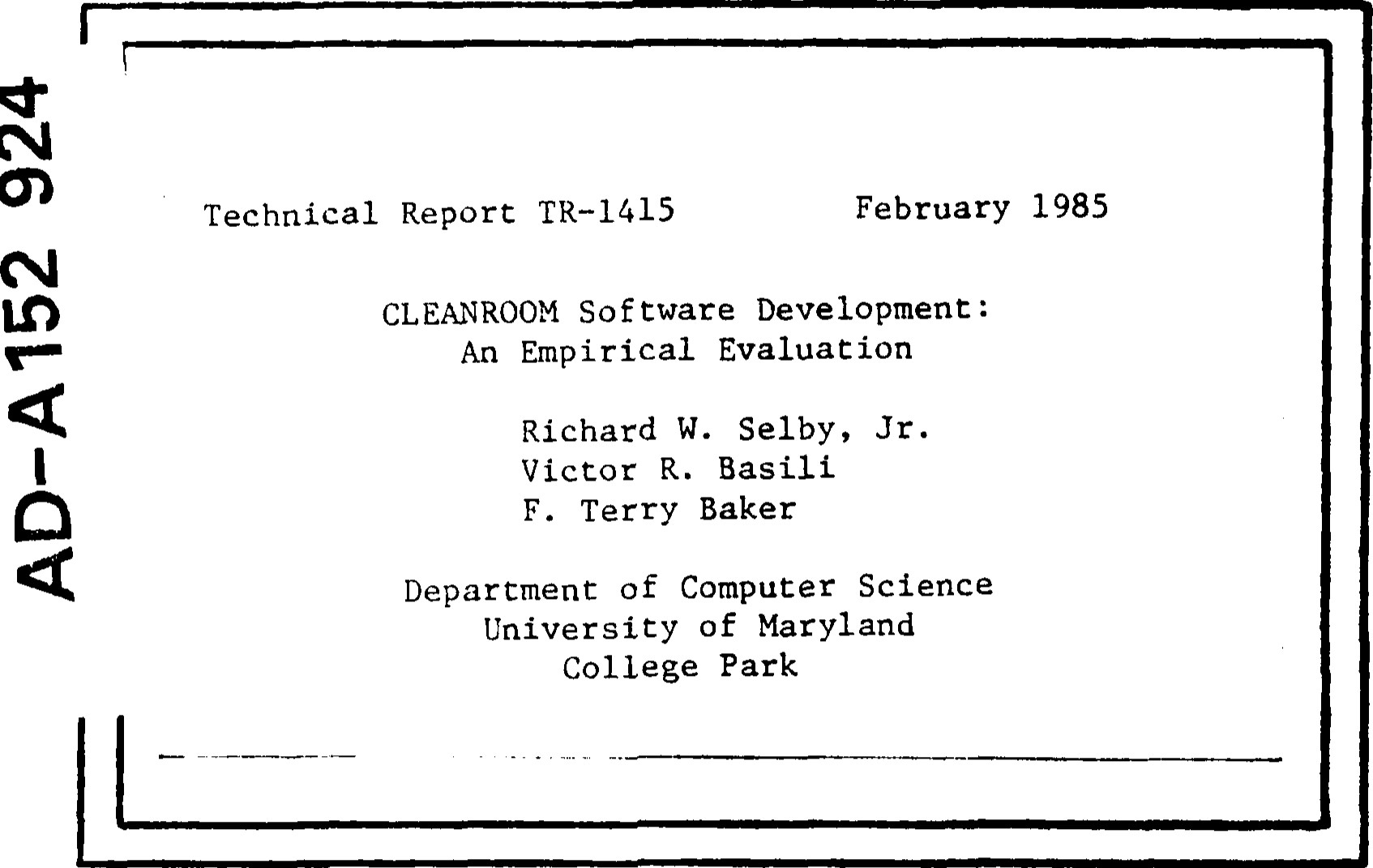
|  |  |  |
| --- | --- | --- |
|  | |  | | --- | | 2.2 | |
| |  | | --- | | 2.0 | |
| |  | | --- | | 16 | |

MICROCOPY RESOLUTION TEST CHART

NAJIONAL BUREAU OF Sı ANDARDS- 1963.A

REPRODUCED AT

# 8 r, . 0 2 9 2



COMPUTER SCIENCE TECHNICAL REPORT SERIES



UNIVERSITY OF MARYLAND

COLLEGE PARK, MARYLAND

## 20742

Approved for publlo releasa distribution unlimited,

UNCLASSIFIED

### SECURITY C' ASSIFICATIO'., O? THIS PAGE

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| REPORT DOCUMENTATION PAGE | | | | | | | | | |
| REPORT SECURITY CLASSIFICATION nCLASSIFIED | | | | | 1b. RESTRICTIVE MARKINGS | | | | |
| 2. SEcuRlTY CLASSIFICATION AUTHORITY | | | | | 3. DISTRIBUTION/AVAI'ABILITY OF REPORT  Approved for public release,- distribution unlimi ted . | | | | |
| 2b DECLASSIFICATION/OOWNGRAOING SCHEDU'E | | | | |
| PERFORMING ORGANIZATION REPORT NUMBER(S) TR-1415 | | | | | S. MONITORING ORGANIZATION REPORT NUMBER'S)  AFOSR-TR. 8 5 - 0 292 | | | | |
| 6. NAME OF PERFORMING ORGANIZATION University of Maryland | | | | b. OFFICE SYMBOL  (It applicable | 7.. NAME OF MONITORING ORGANIZATION  Air Force Office of Scientific Research | | | | |
| ADDRESS (City. Slate and mp Code'  Department of Computer Science College Park MD 20742 | | | | | 7b. ADDRESS State and ZIP Code)  Directorate of Mathematical & Information Sciences, Bolling AFB DC 20332-6448 | | | | |
| a.. NAME OF FUNDING/SPONSORING  ORGANIZATION  AFOSR | | | | 0b. OFFICE SYMØOL  (l/ eplicobk) | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER  F49620-80-C-0001 | | | | |
| st ADDRESS 'City. State and ZIP Code'  Bolling AFB DC 20332-6448 | | | | | 10 SOURCE 06 FUNDING NOS | | | | |
| PROGRAM  ELEMENT NO.  61102F | PROJE CT NO.  2304 | TASK NO. | | WORK UNIT NO. |
| 11. TITLE (Include Security Claulftcotaom  CLEANROOM SOFTWARE DEVELOPMENT: AN EMPIRICAL EVALUATION | | | | | | | | | |
| 12. PERSONA' AUTHOR(S)  Richard W. Selb Jr. Victor R. Basili and F. Terr Baker | | | | | | | | | |
| 13. TYPE OF REPORT Technical | | | 13b. TIME COVERED  FROM TO | | 14. DATE OF REPORT 'Yr . Mo.. Day)  FEB 85 | | | 15 PAGE COUNT  29 | |
| 16. SUPP'EMENTARY NOTATION | | | | | | | | | |
| COSATI CODES | | | | 18. SUBJECT TERMS 'Continue on muer•e 1/ neceuary and 1&Rtaty by block number'  Software development methodology; off—line software review; software measurement; methodology evaluation; software management; empirical studv. | | | | | |
|  | GROUP | SUB. GR | |
|  |  |  | |
|  |  |  | |
| 19. ABSTRACT (Continue on mverae neceuary and 'den t'/y by block number '  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 ba4ed 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 ( l ) 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. | | | | | | | | | |
| 20. OF ABSTRACT  SAME AS RPT. O OTIC USERS O | | | | | 21. ABSTRACT SECURITY CLASSIFICATION  UNCLASSIFIED | | | | |
| - RESPONSIBLE INDIVIDUAL CPT John P. Thomas, Jr. | | | | | 22b TELEPHONE NUMBER  (Include .4 ma Code  (202) 767- 5026 | | 22c OFFICE SYMBOL  NM | | |

DD FORM 1473, 83 APR EDITION F JANoesou E- UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

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

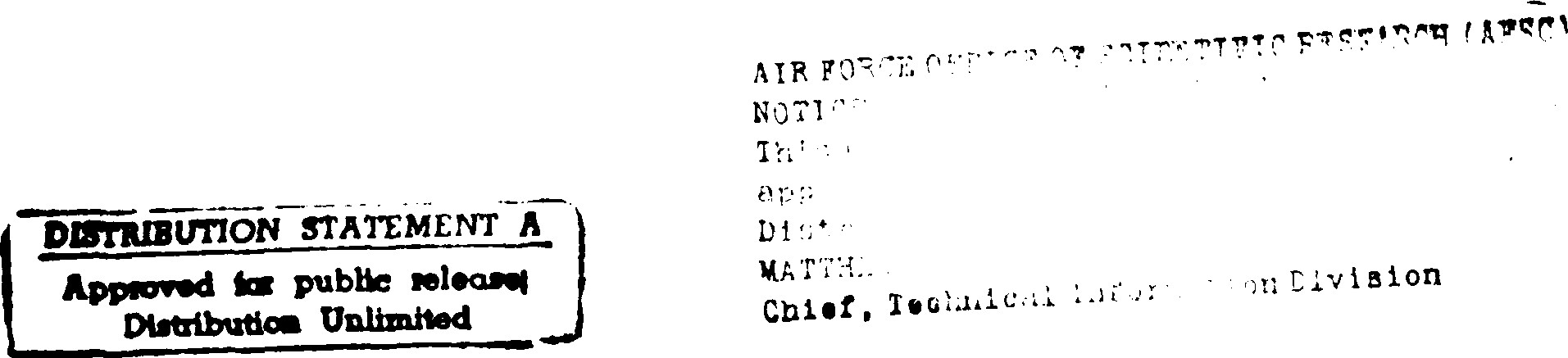
# DTIC

## ELECTE

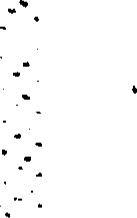
APR 6 085

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

Research supported in part bv the AFOSR Contract AFOSR—F 49620—80-COOI 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 offline development, and statistically based testing. In an emplrlcal study, 15 three-person teams developed versions of the same software system (800 — 2300 source llnes); cen

teams applied Cleanroom, whlle five applied a more Cradltlonal approach. This analysis characterlzes the effect of Cleanroom on the dellvered product, the software developmenc process, and the developers. The major results of this study are 1) most developers were able co apply the techniques of Cleanroom effectively; 2) the Cleanroom teams'

products met system requlremencs more completely and had a hlgher 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 mod]fled aspects of development style; and 5) most Cleanroom developers Indicated they would use the

approach agaln.

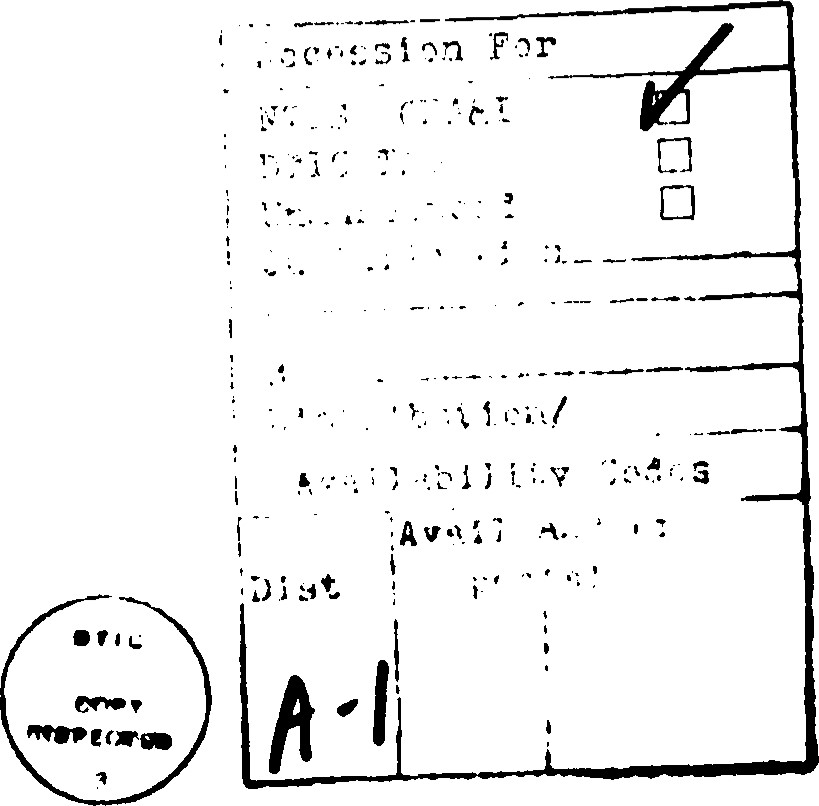


Table of Contents

1. Introduction1
2. Cleanroom Software Development1
   1. Investigation Goals3
3. Emplrlcal Scudy Using Cleanroom
   1. Case Study Descrlpclon5
   2. Operational Testing of Projects
4. Data Analysts and Interpretation8
   1. Characterization of the Effect on the Product Developed9

#### 4.1.1 Operational System Properties9

4.1.2 Static System Properties12

4.1.3 Contrlbutlon of Programmer Background14

4.1.4 Summary of the Effect on the Product Developed15

Characterlzatlon of the Effect on the Development Process15

4.2. I Summary or che Effect on the Development Process20

4.3 Characterization of the Effect, on Che Developers20

4.3.1 Summary of the Effect on the Developers23

4.4 Dlsctnction Among Teams23

1. Conclusions24
2. Acknowledgement26 - Appendix A.26

8 References

i 

### 1. Introduction

The need for dlsclpllne In the software development process and for high quality software motivates the Cleanroom software development approach. In addltlon to

 Improvlng the control durlng development, this approach Is Intended to deliver a product that meets several quallty aspects: a system that conforms with the requlremencs, a

system with high operational rellablllcy, and source code that Is easily readable and

modifiable.

Section Il describes the Cleanroom approach and a framework of goals for characteriz\ng tits effect. Section Ill presents an empirical study using the approach. Section  gives che results ot the analysis comparing projects developed using Cleanroom with chose of a control group. The overall conclusions appear In Section V.

### 2. Cleanroom Software Development

The Federal Systems Dlvlslon ot IBM [Dyer 82, Dyer & Mllls 82) presents the

Cleanroom software development method as a techntcal and organizational approach to developlng software with certifiable rellablllty. The Idea Is to deny the entry ot defects durlng the development of software, hence the term ' 'Cleanroom." The focus of che

methi0d Is Imposlng dlscipllne on the development process by integrating tormal methods for speclflcatlon and deslgn, complete off-line development, and statistically based testIng. These components are Intended to contrlbute to a software product chat has a high

probabillcy of zero defects and consequently a high measure of operational rellablllty.

The mathematically-based design methodology of Cleanroom Includes che use or

structured specifications and state machine models [Ferrenclno & Mills 77]. A systems

engineer Introduces che structured speclflcatlons to restate the system requirements pre-

clsely and organize the complex problems Inco manageable parts [Parnas 721. The

speclflcatlons determine che "system archltecture" of the Interconnections and groupings

of capabllltles to which state machine design practlces can be applied. System Imple-

mencatlon and test data formulation can then proceed from the structured specifications

Independently.

The right-the-flrst-tlme programming methods used In Cleanroom are the Ideas of functionally based programming In [Mills 72b, Linger, Mllls & WItc 79]. The testing

process Is completely separated from the development process by not allowing the

developers to test and debug thelr programs. The developers focus on the technlques of code Inspections [Fagan 731, group walkthroughs [Myers 76], and formal verlflcaclon

[Hoare 69, Linger, Mills & NdVltt 79, Shank-ar 82, Dyer 831 to assert the correctness ot

their Implementation. These constructive techniques apply throughout all phases of

development, and condense the actlvlcles of defect detection and Isolation Inco one

operation. Thls dlsclpllne Is Imposed wlth the Intention that correctness Is • 'deslgned"

Inco the software, not "tested" ln. The notion chat  the software should always

be bested to find the faults" Is ellmlnated.

In the statlsclcally based testing strate re.y of Cleanroom, Independent testers slmu-

late the operational environmenc of che system with random testing. This testing pro-

cess Includes defining che frequency discrlbutlon of Inputs to the system, che frequency

dlstrlbutlon of differenc system states, and the expandlng hierarchy ot developed system

capabilities. Test cases chen are chosen randomly and presented co che series or product

releases, while concentrating on functions most recently delivered and malntalnlng the

overall composite distribution of Inputs. The Independent testers then record observed

tallures and determlne an objectlve measure of product reliability. It Is belleved that the prlor knowledge that a system wlll be evaluated by random testing will affect system

rellablllty by enforclng a new dlsclpllne Into the system developers.

#### 2.1. Investigation Goals

Some Intrlgulng aspects of the Cleanroorn approach Include 1) development without testing and debugging ot programs, 2) Independent program testing for quallCy assurance (rather than to find faults or to prove "correctness" [Howden 76]), and 3) certification of system rellablllty before product delivery. In order to understand the effects of uslng Cleanroom, the followlng three goals are proposed: 1) characterize the effect of Cleanroom on the dellvered product, 2) characterize the effect of Cleanroom on

the software development process, and 3) characterlze the effect of Cleanroom on the developers. An appllcaclon of the goal/questlon/mecric paradigm [Baslll & Selby 84,

Baslll & 'vVelss 841 leads to the framework ot goals and questions for thls study appearIng In Figure 1. The emplrlcal study executed to pursue these goals Is descrlbed In the

followlng section.

|  |
| --- |
| Figure 1. Framework of goals and questions for Cleanroom development approach anal Sts. |
| I. Characterize che effect ot Cleanroom on the delivered product.   1. For Intermediate and novice programmers bulldlng a small system, what were the operational properties of the product?    1. Did the product meet the system requirements?    2. How dld the operational testing results compare wlth those of a control group? 2. What were che static properties of the product?    1. X'Vcre the size propertles of the product any different trom whac would be observed tn a cradltlonal development?    2. Were the readablllty properties of the product any different?    3. Was the control complexlty any dlfferent?    4. MÏa.s the data usage any different?    5. »•.ras the Implementatton language used any differently? 3. What contribution dld programmer background have on the final product quall-     Il. Characterlze the effect of Cleanroom on the software development process.  A. For Intermediate and novtce programmers bulldlng a small system, whac technlques were used co prepare the developing system for testing submlsslons? B. N,Vhat role dld the computer play In development? C. Dld they meet thelr deHvery schedule?  Ill. Characterize che effect of Cleanroom on the developers.  A. VVhen Intermediate and novice programmers bullt a small system, dld the developers miss the saclsfactlon of executing their own programs?   * 1. Dld the mlsslng of program execution have any relatlonshlp to programmer background or to aspects ot the dellvered product?  1. How was the design and codlng style of the developers affected by not belng able to test and debug? 2. Mtould they use Cleanroom again? |

### 3. Empirical Study Using Cleanroom

This section describes an empirical study comparlng ceam projects developed using

Cleanroom with those using a more conventional approach.

3.1. Case Study Description

Subjects for the emplrlcal study came from the "Software Design and Develop-

ment" course taught by F. T. Baker and V. R. Basill at the University ot Maryland In

he Falls of 1982 and 1983. The Initial segment ot the course was devoted to the

presencatlon of several software development methodologies, Including top-down design,

modular specification and deslgn, PDL, chler programmer teams, program correctness,

reading, walkthroughs, and functional and structural testing strategies. For the

latter part of the course, the Indlvlduals were dlvlded Into three-person chlef program-

raer teams for a group project [Baker 72, Mills 72a, Baker 81). We attempted to dlvlde

th •• teams equally according to professional experience, academlc performance, and

Implementation language experlence. The subjects had an average of 1.6 years protes-

sional experience and were computer science majors with Junior, senior, or graduate

standlng. Figure 2 dlsplays the dlscrlbutlon ot the subjects' professional experience.

|  |
| --- |
| Flgure 2. Sublects' rofesslonal ex erlence In years. |
| x x xxxx x xxxxx  x  x xxxxx x |

A requirements document for an electronic message system (read, send, malllng

llscs. authorlzed capabllltles. etc.) was dlstrlbuted co each of che teams. The project was

to be completed In six weeks and was expected to be about 1200 lines or Slmpl-T source [Baslll & Turner 76]. The developmenc machlne was a Univac 1100/82 running E.NŒC

with 1200 baud Interacttve and remote access available.

The ten teams in the Fall 1982 course applied the Cleanroom software development

approach, while the five teams In che Fall 1983 course as a control group (nonCleanroom). All other aspects of che developments were same. The two groups ot teams were noc staclsclcally different In terms of professional experience, academic per. formance, c.- Implementation language experience. If there were any blas between the two clmes 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 earller. It was

also che second time F. T. Baker had taught the course. N' Yue that the teams In the

non-Cleanroom group applied a development approach slmllar to che "dlsclpllned team" approach examined In an earller study [Basill & Relter 81].

The first document every team In elther group turned In contalned a system specification. composite deslgn dlagram, and Implementation plan. The latter element

was a series ot milestones describing when the varlous functions wlthln the system

would be avallable. Ac these various dates (minimum one week apart, maximum two), teams from both groups would then submit thelr systems ror tesclng. An Independent

party would then apply stactsttcally based testing to each or these deliveries and report co the team members both che successful and unsuccessful test cases. The latter would

Stmpl-T Is a structured language thac supports several string and file handllng prlrnitlves. In addition to the usual control flow constructs available, for example, In P?-scal. Ir• Pascal or FORTRA.N had been chosen. would have been very Ilk-ely tihat some Individuals would have had extensive experience with the language, and this would have blased che comparfson. Also, restr!ctlng access to a compiler chat produced executable code would have been very difficult.

be Included In the next cesc session for verlflcaclon. Recall that the Cleanroom teams

could noc execute thelr programs they had editing and syntax-checklng capabilities

only. They had to rely on che technlques ot code reading, structured walkthroughs, and

Inspections to prepare their programs before submlsslon. On the other hand, the nonCleanroom teams had full access to compllatlon and execution tacllltles to test thelr sys-

cems prlor co Independent testing.

All team projects were evaluated on the use of Che development techniques

presented In class, the Independent testing results, and a final oral Interview. In addl-

clon to these sources, Informaclon on the team projects was collected from a background

quesctonnalre. a postdevelopmenc attitude survey, static source code analysls, and

operating system staclstlcs. The following sectlon briefly descrlbes che operationally

based testing process applied to all proJeccs by the Independent tester.

#### 3.2. Operational Testing of Projects

The testing approach used in Cleanroom is to simulate the developlng system's

envlronmenc by randomly selecting test data from an "operational profile,' • a frequency d!strtb l ltlon ot Inputs to the system [Thayer, Llpow & Nelson 78, Duran Ntafos 811.

The projects from both groups were tested interactlvely ac che mllescones chosen by

each r earn by an Independenc party (I.e., R. Selby). A dlstrlbuclon ot• Inputs co the

system was obtained by Identifying che loglcal runctlons In the system and assigning

each a t•requency. This frequency assignment was accompllshed by polling eleven well-

žensoned lisers ot• the Cnlverslty ot Maryland Vax 11/780 malllng system. Then test

lata were generated randomly from rahls profile and present,ed co the system. Recording

|  |
| --- |
| Flgure 9. Breakdown ot responses to the attltude survey question, ' 'Dld you mlss the sattsfactlon of executln o your own rograms?". |
| 13 — Yes, I missed the saclsfactlon ot program execution.  11 — I somewhat missed the saclstactlon of program execution.  •-1 — No, I did not mlss the satlstactlon of program execution. |
| Figure 10. Relatlonshl• of ro Œ ram size vs. mlssln rogram execution. |
| Yes  10.0  2001  .  o  Ml ssed  Program  Execut Ion  Some  ( 3 . 0 ) Source L Ines  Spearman correlations: —.85 (signlt. = .002) with source lines: —.70 (s!gnlf. = .03) wlth number separately compllable modules: —.57 (slgnlf. .09) wlch number procedures and functions. |

Figure 11 dlsp[ays the replies of che developers when they were asked how thelr

deslgn and codlng style was affected by noc belng able to test and debug. At first It

would seem surprising chac more people did noc modify their development style when

21

##### 4.2.1. Summary of the Effect on the Development Process

Summarlzlng the effect on the development process, Cleanroom developers 1) felt they applied off-line review techniques more effectlvelsn 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 quesclon posed In this goal area Is whether the Individuals using Cleanroom missed the saclsfactlon of executing thelr own programs. Figure 9 presents the responses to a questlon Included In the postdevelopment attitude survey on chis Issue. As mlght be expected, almost all the Individuals missed some aspect o? program execuclon. As might not be expected, however, this mlsslng of program execution had no relation to either the product quallcy measures mentioned earller or the teams' protesslonal or testlng experlence. Also, mlsslng program execution dld not Increase wlth

respect co program slze (see Figure 10).

20

Schedule slippage continues to be a problem In software development. It would be

Interesting to see whether the Cleanroom teams demonstrated any more dlsclpllne by

malntalnlng thelr orlglnal schedules. All of the teams from boch groups planned four

releases of their evolving system, except for team 'G' which planned five. Recall that ac

each dellvery 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 uslng Cleanroom kept to thelr orlglnal schedules by maklng

all planned deliver:es; only two non-Cleanroom teams made all their scheduled

deliveries.

|  |  |  |  |
| --- | --- | --- | --- |
| Figure 8. | Number of svstem releases. |  |  |
|  | d a    1 2 3  Mann-Whitney stgnit. = .006 | 1  D c  b  4 5 | 6 |

9 Non-Cleanroom ceam 'e' entered a subscanclal portion of Its system on a remote machlne, only using the Univac computer mainly for compilation and execution. (See Dlstlnctlon Among Teams.)

### 19

were unable to rely on testing methods, they may have (felt they had) applied the off-

line review techniques more effecclvely.

Since the role of the computer ts more controlled when using Cleanroom, one would expect a difference In on-line acclvlty between che two groups. Figure 7 displays the amount of connect time bhac each of the teams cumulatively used. A comparlson ot the

cpu-t,lme used by the teams was less staclstlcally slgnlflcanc ( MM/ — .110). LNelther of these measures of on-line activity related to how effectively a team felt they had used the off-llne techniques when elther all teams or Just Cleanroom teams were considered. Although non-Cleanroom team 'd' dld a lot of on-line testing and non-Cleanroom team

•e' did little, both teams performed poorly In the measures of operational product qualIcy discussed earlier. The operaclng system of the development machine captured these system usage statistics. Note chat the time the Independenc party spent testing Is

Included. 8 These observations exhlblt that Cleanroom developers spent less time on-llne and used fewer computer resources. These results emplrlcally support the reduced role of the computer In Cleanroom development.

|  |
| --- |
| Fl"nre 7. Connect t)rne In hours durin rolect develo ment. 9 |
| JA  d  0 .0 155 .0  MannAVhltney slgnlf. = .089 |

When che time che Independent tester spent Is noc Included, che slgnlflcance levels for the non-paramecrlc staclstlcs do noc change.

#### 18

|  |
| --- |
| Figure O. Breakdown of responses to the attitude survey question, ' 'Dld you feel that you and your team members effectively used off-line review techniques In testin vour rolect?". Res onses are trom Cleanroom teams.) |
| — Yes, they were effective for testlng all parts of the program  5.5 — We used them but felt that they were only approprlate for certain parts of the program  8.5 — We used them occasionally, but they were not really a major contrlbuclng factor to the development  O — Dld not really use chem ac all |
| feelŽng of effective use of off-line review techniques: both groups  (team 'e' does not appear because of lack of response)  1 D c d c a b    d Id not use effect Ive for  al l parts  Mann-Whitney slgnlf. = .065 |

The histogram In Figure 6 shows that the Cleanroom developers felt they applied

the off-line review techniques more effectively chan dld the non-Cleanroom teams. The

non-Cleanroom developers were asked to give a relative breakdown ot the amount of

##### time spent applying testing and verlflcaclon techniques. Thelr aggregate response •was

39% off-llne review, 52% functional testing, and 9% structural cestlng. From this

breakdown, we observe chac the non-Cleanroom teams prlmarlly relied on functional

cestlng to prepare chelr systems for Independent testing. Since the Cleanroom teams

 There are half-responses because an tndlvldual checked both the second and third choices. The responses total to 28, noc 30, because two separate teams lost a member late In che project. (See Dlstlncclon Among Teams).

##### 17

the teams (R = .58; slgnlf. = .023). Neither pro:esslonal nor testing experience corre-

lated with off-llne review effectiveness when elcher all teams or Just Cleanroom teams were considered.

##### 16

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 hlgher 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 requlrlng less (software sclence) effort to comprehend, and 5) had more general programming language experience.

4.2. Characterization of the Effect on the Development Process

In a postdevelopmenC attitude survey, the developers were asked how effectively Chey felt they applied off-line revtew techNques In testing thelr proJects (see Figure 6).

Thls was an attempt to capture some of the Information necessary to ansqvver the first question under this goal (question Il.A). In order to make comparisons at the team level, the responses from the members of a team are composed Into an average for che team. T he responses to the question appear on a team basis In a hlstogram In the second part of the figure. Of the Cleanroom developers, teams 'A,' 'E,' and 'I' were the least confidenc In their use of the off-line revlew techniques and these teams also performed the worst In terms of operational testlng results; four of these five teams performed the worst In terms of Implementation completeness. Off-line review effectiveness correlated with percentage or successful operational tests (without dupllcate

failures) for the Cleanroom teams (Spearman R .74: slgnlf. = .014) and for all the

teams (R = .76: slgnlf. = .001); It correlated with Implementation completeness for all

###### 15

Considering the products from all teams, both percentage of successful test cases

(wlthout duplicate failures) and Implementation completeness had some correlation with

percentage of It statements (R = .48, slgnlf. = .07, and R = .45, slgnlf. = .09, respeccively) and some negative correlaclon wlth percentage of case statements (R = —.48, slg-

nlf. .07, and R = —.42, slgnlf. = .12, respectively). Nelther of the operational pro-

duct quallty measures correlated wlth percentage of assignment statements when elther all products or Just Cleanroom products •were consldered. These obsenratlons suggest

that the more successful Cleanroom developers slmpllfled thelr use of the Implementa-

tlon 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

requlrlng less software science effort to comprehend.

4.1.3. Contribution of Programmer• Background

N\*õhen examlnlng the contribution of the Cleanroom programmers' background to

the quality of their final products, general programmlng language experience correlated wlth percentage of successful operational tests (without duplicate fallures: Spearman R

= .66, signlf. = .04; with duplicates: R = .70, . = .03) and with Implementaclon

completeness (R = .55; signif. = .10). No relatlonshlp appears between either opera-

clonal testlng results or Implementacton completeness and elther protesstona1[[1]](#footnote-1) or cescing

experlence. These background/ quallcy relations seem conslstent with other studles

[Curtis 83].

14

lower complexlty density (MW = .079) than dld those using che traditional approach.

A calculation of either software science effort [Halstead 771, cyclomatlc complexity

[McCabe 76), or syntacclc complexity without any size normallzaClon, however, produced

no slgnlflcanc differences (MNN > .10). This seems as expected because all the systems were bullt to meet the same requlrements.

Comparing the data usage In che systems, Cleanroom developers used a greacer

number of global data Items (MN•/V = .071). Also, Cleanroom projects possessed a higher

percentage of asslgnment statements (MN.dV = .056). These last two observations could

be a manlfestatlon of teaching the Cleanroom subjects modular design later In the

course (see Case Study Descrlpclon), or possibly an Indication ot using the approach.

Some Interestlng observations surface when the operational quallty measures of the

Cleanroom products are correlated with the usage of the Implementation language.

Boch percentage of successful test cases (without duplicate failures) and Implementation

completeness correlated with percentage or procedure calls (Spearman R = .65, slgnlf.

— .044, and R — .57, slgnlf. = .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 negaclvely with percentage ot case state-

ments (R = -.86. slgnlf. — .001, and R = —.69, slgnlf. = .027, respectively) and with

percentage ot whlle scatements (R = -.65, signif. = .044, and R = —.49, slgnlf. — .15,

respectively)' There were also some negative correlations between che product quallcy

measures and the average software science effort per subroutine (R = -.52, signif. .12, and R = -.74, slgnlf. .013, respecclvely) and the average number ot occurrences

of a variable slgnlf. — .11, and R —.56, sfgnlf. .09, respecclvely).

###### 13

operational testing by Independent testers. Since both groups of teams had Independenc

testing of all thelr deliveries, the early testing of dellverles must have revealed most faults overlooked by the Cleanroom developers.

These comparisons suggest that the non-Cleanroom developers focused on a "per-

spectlve of che tester," sometimes leaving out classes of functions and causing a less completely Implemented product and more (especially unlque) fallures. Off-line review

techniques, however, are more general and their use contrlbuted to more complete

requirement conformance and fewer failures In the Cleanroom products. In addition to examlnlng che operational properties of the product, various static propertles were com-

pared.

4.1.2. Static System Properties

The first question In this goal area concerns the slze of the final systems. Figure 3

showed the number of source lines, executable statements, and procedures and functions tor the varlous systems. The proJects from che two groups were not statistically

different (.MW > .10) In any of these three slze attrlbutes. Another question In this goal

area concerns the readablllcy ot the delivered source code. Two aspects of reading and

modltylng code are the number of comments presenc and the denslty of the • 'complexIn an attempt to capture the complexity density. syntactic complexity [Baslll &

Hutchens 831 was calculated and normalized by che number of executable statements.

In addltlon to control complexity, the syntactlc complexity metric considers nesting

depth and prlme program decomposltlon [Linger, Mills & 791. The developers

using Cleanroom wrote code that was more hlghly commented (NIVV = .089) and had a failures, even though they did better overall. Thls demonstrates that while reviewlng

the code, the Cleanroom developers focused less than the other groups on certain parts

of the system. The more unlform review of the whole system makes the performance of

the system less sensitive to Its operational profile. Note that operational envlronments

of systems are usually difficult to denne a prlorl and are subject to change.

|  |
| --- |
| Flgure 5. Percentage of successful test cases during operational testing (without du Itcate fallures). |
| E 1 F A B G C c a  d  b  58 .0 100  Mann-Whitney slgnlf. — .055 |

In both of the product quallty measures of Implementation completeness and opera-

clonal testing results, there was quite a varlatlon In performance. [[2]](#footnote-2) A wide variation may

have been expected with an unfamlllar development technique, but the developers using \

a more cradltlonal approach had a wider range ot performance chan dld those using

Cleanroom In both ot the measures (even wlth Cwlce as many Cleanroom teams). All or

the above differences are magnified by recalling that the non-Cleanroom teams did not

develop thelr systems In one monollthlc step, they (also) had the benefit of perlodic

11

|  |  |  |
| --- | --- | --- |
| Fl ure 4. | Re ulrement conformance of the stems, |  |
|  | J D  1 de b c a    16 32  22 % 56 % 91 %  Mann-Whltney [[3]](#footnote-3) slgnlf. = .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 simu-

late Its operational environment. If the final release ot a system performed to expecta-

tlons on a test case, che outcome was called a ' 'success;" If not, the outcome was a

"failure." If the outcome was a "fallure" but che same failure was observed on an earlier

test case run on the final release, the outcome was termed a "duplicate failure." Figure

5 shows che percentage of successful test cases when dupllcate failures are not Included.

The figure displays that Cleanroom projects had a higher percentage of successful test cases ac system delivery. [[4]](#footnote-4) »yrhen duplicate failures are Included, however, the better

performance ot che Cleanroom systems Is not nearly as slgnlflcant (M'VV = .134). 4 This

Is caused by the Cleanroom projects having a relatively higher proportion of duplicate

10

4.1. Characterization of the Effect on the Product Developed

Thls secclon characterizes the differences between the products delivered by both of

the development groups. Inltlally we examine some operational properties of the pro-

ducts, followed by a comparison of some of their static properties.

4.1.1. Operational System Properties

In order to contrast che operational properties of the systems delivered by the two

groups, both completeness of Implementatlon and operational testing results were examIned. A measure of Implementation completeness was calculated by partitioning the

required system Into sixteen loglcal functions (e.g., send mall to an Indlvldual, read a

plece of mall, respond, add yourself to a malllng list ) Each function In an Imple-

mentatlon was then assigned a value of two If It completely met Its requirements, a

value ot one If It parclally mec them, or zero If It was Inoperable. The cocal for each

system was calculated; a maximum score of 32 was posslble. Figure 4 displays this subJective measure of requlrement conformance for the Systems. Note that In all figures

presented, che cen 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 bullt very close to the enclre system. Whlle noc all of the Cleanroom

teams performed equally well, a majorlty ot them applied che approach effectively

enough to develop nearly the whole product. More Importantly, the Cleanroom teams

met the requirements off the system more completely than did che non-Cleanroom teams.

9

of failure severity and clmes between failure took place during the testing process. The

operational statistics referred to lacer were calculated trom fifty user-sesslon test cases

run on the final system release of each team. For a complete explanation of the opera-

tlonally based testing process applied to the projects, Including test data selection, testIng procedure, and failure observation, see [Selby 841.

### 4. Data Analysis and Interpretation

The analysls and Interpretation of the data collected from che study appear In the following sectlons, organized by the goal areas outlined earlier. In order to address the

various questions posed under each of che goals, some raw data usually will be presented

and then Interpreted. Figure 3 presents the number of source lines, executable state-

ments, and procedures and functions co give a rough vlew of the systems developed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Figure 3. System scatlsclcs. | |  |  |  |
| Team | Cleanroom | Source Lines | Executable Statments | Procedures & Functions |
| c  D    1 | yes yes yes yes yes yes yes yes yes yes | 1681  1626  1118  1046  1087  1213  1196  1876  1305  1052 | 813  717  573  624  440  581  550  608  658 | 55  42  42  30  32  35  31  51  23  24 |
|  |  |  |  |  |

8

applying the techniques of Cleanroom. Several persons mentioned, however, that they

already utilized some of the Ideas In Cleanroom. Keeping a simple design supports rea-

dabillty of the product and facllltates che processes of modlflcatlon and verlflcatlon.

Although some of the objective product measures presented earlier showed differences In

development style, these subjective ones are Interesting and lend Inslght Into actual

grammer behavior.

|  |
| --- |
| Flgure 11.  Breakdown of responses to the attitude survey question, • 'How was your design and codin stvle affected bv not beln able to test and debu"" |
| 2 — Yes, my style was substantially revised.  15 — I modlfled some of my tendencies.  11 — It did not affect my style at all. |
| Frequently mentioned responses Include  — kept deslgn simple, attempted nothlng fancy kept readablllty of code In mlnd  — already was a user of off-lfne revtew technlques  — very careful scrutiny of code for potential mistakes  — prepared for a larger range of Inputs |

One Indicator of the Impression Chac 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 elther a software develop-

ment manager or as a programmer. Even though these responses were gathered

(Immedlately) after course complecton, subjects deslrlng to "please the Instructor" may

have responded favorably to this type of question regardless of chelr true feellngs, Prac-



note that a greater number of persons In a managerial role would choose to always use

lt. Of the persons that ranked the reuse of Cleanroom falrly 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

•r), but the other two came trom reasonably successful developments (one from team 'C'

and one trom team The particular Individuals on teams 'E,' 'I,' and 'J' rated che reuse fairly low In both categories.

|  |
| --- |
| Figure 12.  Breakdown of responses to the attitude surrey question, ' 'Would you use Cleanroom a aln?". (One erson did not res ond to this uestlon.) |
| As a software development manager?  8 — Yes, ac all times  14 — Yes, but only for certain projects  5 — Not at all |
| As a programmer?  4 — Yes, for all projects  18 — Yes, but noc all che tlme  5 — Only If I had to  O — I would leave ff 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) modlfled

In part thelr 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 accordlng to varlous factors (see Case Study Descrlptlon), a few differences among the teams were apparent. Two separate Clean-

room teams, 'H' and 'I,' each Iosc a member lace In che project. Thus ac project completlon, there were eight three-person and CWO two-person Cleanroom teams. Recall that team 'H' performed quite well according to requlrement 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 ceam 'e' consisted of chat one highly experienced person. Thus at proJecc completion, there were four three-person and one one-person non-Cleanroom teams. Although team •e' wrote over 1300 source llnes, chls 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 "dlsclpllned methodology" In development outperformed Indlvlduals [Baslll & Relter 81]. Appendix A contains the slgnlflcance levels for the above

results when team 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 che

stgnlflcance levels, while the removal of team 'e' causes a decrease In all of the slgnlflcance levels except for executable statements, software science effort, cyclomaclc complexity, syntactic complexlty, connect-time, and cpu-tlme.

#### 5. Conclusions

This paper describes "Cleanroom•• software development — an approach Intended to produce highly reliable software by Integrating formal methods for speclflcatlon and

design, complete off-line development, and statlstlcally based cesclng. The goal struc-

cure. experlmental approach, data analysis, and concluslons are presented for a

repllcated-proJect study examln!ng the Cleanroom approach. This Is the first Investiga-

clon known to the authors chat applled Cleanroom and characterized ICs effect relaclve

to a more tradltlonal development approach.

The data analysis presented and the testimony provided by the developers suggest

that the major results of chls study are 1) most developers were able to apply che tech-

nlques of Cleanroom effectively; 2) the Cleanroom teams' products met system require-

ments more completely and had a higher percentage of successful tesc cases; 3) the

source code developed using Cleanroom had more comments and less dense complexlcy;

4) che use ot Cleanroom successfully modlfled aspects of development style; and 5) most

Cleanroom developers Indicated they would use the approach again.

It seems that the Ideas In Cleanroom help actaln the goals of producing high quality

software and Increasing the dlsclpllne In the software development process. The com-

plece separaclon of development from testlng appears to cause a modlflcatlon In the

developers' behavior, resulting In Increased process control and In more effectlve use of

formal methods for software speefflcatton, design, off-line review, and verification. It

seems that system modlflcatlon and malntenance would be more easily done on a pro-

duct developed In the Cleanroom method, because of the product's thoroughly concelved

design and hlgher readablllty. Thus, achlevtng hlgh requirement conformance and high

operational reliablllty coupled with low maintenance costs would help reduce overall

costs, satlsfy the user community, and support a long product llfetlme.

Thls emplrlcal study Is Intended to advance che understanding of the relaclonshlp

between Introducing discipllne Into the development process (as In Cleanroom) and

several aspects of product quality: conformance wlth requirements, high operatlonai reli-

ablllcy, 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

extrapolatton of the findings co other projects and development environments Is not

Implied. Valuable Insights, however, have been galned from che analysis.

##### 6. Acknowledgement

The authors are graceful to D. H. Hutchens and R. W. Reiter for the use of thelr

analysis program In thls study.

1. Appendix A.

Figure 13 presents the measure averages and the slgnlflcance levels for the above

comparlsons when team when teams 'H' and and when teams 'e,' and 'I' are

removed. The significance levels for the Mann-Whitney statlstlcs reported are the

bablllty ot Type I error In an one-tailed test.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Figure 13. Summary of measure averages and significance levels. | | | |  | | | |
|  | Avera e | |  | Mann-WMcney significance levels | | | |
| Cleanroom Teams | Non-Cleanroom Teams | All  Teams |  | Without Team e | Without  Teams 1-1.1 | Without  Teams e.H.1 |
| Source lines  Executable stmts  \*Procedures & functions  %lmplementation completenes  %Successful tests (w/o duplicate failures)  %Successful tests (w/ duplicate failures) | 1320.0  604.1  36.5  82.5  92.5  8.7 | 1491.2  625.4  40.0  60.0  80.8  59.2 | .196  .500  357  .088  .055  .134 |  | .240  286  .500  .197  .128  285 | .153  .442  .330 .093  .053  .151 | .198  .367  .500  .196  .116  .304 |
| \*Commencs  Syntactic complexity/ executable stmcs  Software Science E  Cyclomatlc complexity  Syntactic complexity  Global data items | 194.9  1.5  6—08, - ,6e3  196.8  37.6 | 1.6  7355.40  1017.0 | .089  .079  .451  250  500  .071 |  | .102  .179  240  .198  286  .129 | 190  .082  .255  .500  .053 | .198  .175  248  248  .305  .117 |
| %Assignment stmts  Off-line effectiveness  Connect-time (hr.)  Cpu•time t min.) Delivertes | 3.2  4.1 | 26.6  71.3  2.6 | .056  .065  .089  .110  .006 |  | .129  .065  .017  .015 | 040  .098  .121  .07 2  .010 | .087  098  .009  .022 |

28

1. References

[Baker 721

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

[Baker 811

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

[Basill Turner 761

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

###### [Baslll & Reiter 81]

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

[Baslll & Hutchens 831

V. R. Baslll and D. H. Hutchens, An Empirical Study ot a Syncacclc Metric Family, Trans. Software Engr. SE-9, 6, pp. 664-672, Nov. 1983.

[Baslll & Selby 84]

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

###### [Basill & Welss 84]

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

###### [Currit 831

P. A. Currlt, Cleanroom Certiflcatlon Model, Proc. Eight Ann. Software Engr. Workshop, NASA/GSFC, Greenbelt, MD, Nov. 1983.

[Curtis 83]

B. Curt:s, Cognltlve Science of Programming, Sixth Minnowbrook Workshop on Software Performance Evaluation, Blue Mountain Lake, NY, July 19-22 1983.

[Duran & Ncafos 811

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

[Dyer Mills 821

M. Dyer and H. D. Mllls, Developing Electronlc Systems with Certifiable Rellablllty, Proc. IVATO Conf, Summer, 1982.

[Dyer 821

M. Dyer, Cleanroom Software Development Method, IBM Federal Systems Dlvlslon, Bethesda, MD, October 14, 1982.

[Dyer 831

M. Dyer, Software Validation In the Cleanroom Development Method, œMFSD Tech. Rep. 86.0003, August 19, 1983. [Fagan 761

M. E. Fagan, Design and Code Inspections to Reduce Errors In Program Developmenc, IBM Sys. J. 15, 3, pp. 182-211, 1976.

[Ferrentlno Mills 77]

A. B. Ferrentlno and H. D. Mllls, State Machines and Thelr Semanttcs tn Software Engineering, Proc. IEEE COMPSAC, 1977.

[Goel 831

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

[Halstead 771

M. H. Halstead, Elements of Software Science, North Holland, New York,

[Hoare 69]

C, A, R. Hoare, An Axiomatic Basis for Computer Programming, Communications of the ACM 12, 10, PP. 576-583, Oct. 1969.

i

(Howden 761

W. E. Howden, RellablllCy of che Pach Analysls Testing Strategy, IEEE Trans. Software Engr. SE-2, 3, Sept. 1976.

(Linger, Mills WItt 79!

R. C. Linger, H. D. Mills, and B. I. WICt, Structured Programming: Theory and Practice, Addlson-Wesley, Readlng, MA, 1979.

28

[McCabe 761

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

[Mills 72al

H. D. Mills, Chief Programmer Teams: Principles and Procedures, IBM Corp., Gaithersburg, ND, Rep. FSC 71-6012, 1972.

[Mills 72b]

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

[Musa 751

J. D. Musa, A Theory of Software Rellablllty and Its Appllcatlon, EEE Trans. Software Engr. SE-1, 3, pp. 312-327, 1975.

(Myers 761

G. J. Myers, Software Reliability: Principles 8 Practices, John Wiley & Sons, New York, 1976.

[Parnas 72]

D. L. Parnas, On the Crlterla 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 Quantltative Approach for Evaluating Software Technologles, Dept. .Com. SCI., Unlv. Mawland, College Park, Ph. D. Dissertation, 1984.

[Shankar 821

K. S. Shankar, A Functional Approach co Module Verlflcation, IEEE Trans. Software Engr. SE-8, 2, March 1982.

[Thayer, Lipow & iNelson 781

R. A. Thayer, M. Llpow, and E, C. Nelson, Software Reliability, North Holland, Amsterdam, 1978.

29

# END

FILMED

 5-85

## DTIC

1. In fact, there are very sllghC negative correlations between years or professional experlence and both percentage of successful tests (without dupllcace faliures: R signif . and Implementatton completeness (R — - .47, signlf. = .17). [↑](#footnote-ref-1)
2. An alternate perspective Includes only the more successful projects from each group In the comparison of operational product quality. vvthen the best from each approach are examined (I.e., removing teams 'd,' • A,' E, • 'F, • and • the NIannWhlcney slgnlflcance level for comparlng Implementatlon completeness becomes .045 and che significance level for comparing successful test cases (wlthouc duplicate tallures) becomes .034. Thus, comparing the best teams from each approach Increases the evtdence In favor or Cleanroom In both of these product quality measures. [↑](#footnote-ref-2)
3. The signlflcance levels for the Mann-Wthltney statlstlcs reported are the probablllty of Type error In an one-tailed test. [↑](#footnote-ref-3)
4. Although not considered here, various software rellablllty models have been proposed to forecast system rellablllty based on fallure data [Musa 75, Currlt 83, Goel 831.

   To be more succinct, MW wlll sometimes be used co abbreviate the significance level of the Mann4Vhltney statistic. [↑](#footnote-ref-4)