April 1993

UILU-ENG-93-2212
CRHC-93-05

Center for Reliable and High-Performance Computing

Do Not Remove

ANALYSIS OF LARGE SYSTEM BLACK BOX VERIFICATION TEST DATA

Kenneth C. Clapp and Ravishankar K. Iyer

Coordinated Science Laboratory
College of Engineering
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS: None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): UILU-ENG-93-2212 CRHC-93-05	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Coordinated Science Lab, University of Illinois	6b. OFFICE SYMBOL (If applicable): N/A
6c. ADDRESS (City, State, and ZIP Code): 1101 W. Springfield Ave., Urbana, IL 61801	7a. NAME OF MONITORING ORGANIZATION: ONR, NASA
	7b. ADDRESS (City, State, and ZIP Code): Ofc. of Naval Research, 800 N. Quincy, Arlington, VA; NASA Langley, Hampton, VA 23665
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: ONR, NASA	8b. OFFICE SYMBOL (If applicable)
8c. ADDRESS (City, State, and ZIP Code): Office of Naval Research, 800 N. Quincy, Arlington, VA; NASA Langley, Hampton, VA 23665	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: ONR: N000145-1116; NASA: NAG 1-613
	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification): Analysis of Large System Black Box Verification Test Data

12. PERSONAL AUTHOR(S): Kent C. Clapp and Ravishankar K. Iyer

13a. TYPE OF REPORT	13b. TIME COVERED	14. DAYE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	April 1993	48

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Test, Verification, Evaluation, Coverage, Prediction, Black Box, White Box, Testing Process, Data Analysis

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This paper explores issues regarding black box, large systems verification. It begins by collecting data from several testing teams. An integrated database containing test, fault, repair, and source file information is generated. Intuitive effectiveness measures are generated using conventional black box testing results analysis methods. Conventional analysis methods indicate that the testing was effective in the sense that as more tests were run, more faults were found.

Average behavior and individual data points are analyzed. The data is categorized and average behavior shows a very wide variation in number of tests run and in pass rates (pass rates ranged from 71% to 98%). The "white box" data contained in the integrated database is studied in detail. Conservative measures of effectiveness are discussed. Testing efficiency (ratio of repairs to number of tests) is measured at 3%, fault record effectiveness (ratio of repairs to fault records) is measure

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

at 55%, and test script redundancy (ratio of number of failed tests to minimum number of tests needed to find the faults) ranges from 4.2 to 15.8. Error prone source files and subsystems are identified. A correlational mapping of test functional area to product subsystem is completed. A new adaptive testing process based on real-time generation of the integrated database is proposed.

Analysis of Large System Black-Box Test Data

Kent C. Clapp
AT&T Network Systems
Naperville, IL

Ravishankar K. Iyer
Center for Reliable and High Performance Computing
University of Illinois at Urbana-Champaign

ABSTRACT

This paper studies black box testing and verification of large systems. Testing data is collected from several test teams. A flat, integrated database of test, fault, repair, and source file information is built. A new analysis methodology based on the black box test design and white box analysis is proposed. The methodology is intended to support the reduction of testing cost and enhancement of software quality by improving test selection, eliminating test redundancy, and identifying error prone source files. Using example data from AT&T systems, the improved analysis methodology is demonstrated.

Index Terms: Software Testing, Validation, Fault/Repair Data, Analysis, Test Minimization

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The cost of finding and repairing faults in programs ranges from 40 to 80 percent of the total development cost[1]. Effectiveness and efficiency of a test program in achieving its goals must be analyzed in order to contribute favorably to overall product goals. The analysis must consider data from a test database, as well as data from product-fault and product-repair databases. New methods to measure test efficiency and analysis are necessary in order to both maintain product quality and contain development cost.

This paper addresses the issue of network level, large systems verification. Verification of networks often requires execution of hundreds of black box test scripts each of which may cover several loosely coupled network elements. Each network element may use several different software programming languages. The study begins by collecting data from several testing teams. An integrated database is generated containing test, fault, repair, and source file information. An overview of the testing data is provided and intuitive effectiveness measures are generated from conventional black box test data. The data is categorized by team and Functional Area (FA). Average behavior and individual data points are analyzed. Finally, the "white box" data contained in the integrated database is used in the analysis of the testing results. Specifically, the analysis methodology consists of three steps: 1) Build a flat, integrated database of test, fault, repair, and source file information. 2) Use the database to evaluate the efficiency of testing and the effectiveness of fault repairs. 3) Based on the database, modify test selection, identify redundant tests, and identify error prone source files to reduce testing cost and enhance software quality.

The results of our analysis show that when the white box and black box data contained in the integrated database were jointly analyzed, the net impact to the product (11 source files repaired) of the testing effort (352 tests) was clarified. Testing efficiency (ratio of repairs to number of tests) was measured at 2 percent, fault record effectiveness (ratio of repairs to fault records) was measured at 35 percent, and measures of test script redundancy (ratio of number of failed tests to minimum number of tests needed to find the faults) ranged from 4.2 to 15.8. Error prone source files and subsystems were also identified. An adaptive testing process is proposed.

The following terms are used throughout this paper: **"Feature"** and **"Functional Area (FA)"** refer to user-oriented segments of product functionality which are independent of the internal product structure. **"SubSystem (SS)"** refers to closely related software components which make up a subset of the overall software product. **"Black box"** refers to methods or tests which are independent of the internal structure (subsystems) of the product. A black box test script is developed primarily from the feature definition. **"White box"** testing refers to methods which can be mapped directly to product internals (subsystems, functions, etc.). The terms "test run" and "test execution" are used interchangeably.

2. SURVEY OF RELATED LITERATURE

Many studies of software testing have been performed. Deterministic methods for white box testing of small software products have been effectively utilized. Structure-based testing attempts to use knowledge of program constructs to test a segment of code. Completion of testing is based on statement coverage, branch coverage, switch coverage, etc. Complexity-based approaches to testing compute the cyclomatic complexity of the program using a program graph. Number of edges, nodes, and connected components are considered. Testing coverage is then measured relative to the program graph. Analytic correctness proofs are sometimes used to prove program correctness. Gallagher et.al.,[2] introduce decomposition slices as a way to eliminate regression testing.

Each of these white box methods can be very effective in providing product coverage and finding faults in small program testing, but become unproductive when testing a large system with millions of Non Commentary Source Lines (NCSLs). Methods for measuring coverage and effectiveness are much less clear in large system black box testing.

Important topics in effective black box testing include test goals, test selection strategies, test processes, and test coverage. Kaner discusses the goals of testing. Kaner[1] describes the dilemma of black box testing as dominated by the need to select a few test cases from a huge set of possibilities. The author concludes that it is not the purpose of testing to prove that a program works correctly, nor to find all the

bugs, but to find and repair as many faults as possible. A guiding principle is "**A test that finds a problem is a success. A test that did not reveal a problem was a waste of time.**" Howden[3] states that the goal of developmental testing is completeness with respect to programming errors.

Several strategies for **test selection and design** exist. Dunham et.al.,[4] discussed the transformation of validation into a systematic series of integrated steps. Chow[5] states that tests must be carefully selected to increase the probability that if there are errors in the program, they will be detected during test runs. Musa, et.al.,[6] state that the best test strategy is the one that results in the greatest reduction in operational failure cost per unit of test cost. In general, tests should be selected such that the failure intensity is reduced as rapidly as possible. Sherer[7] proposes a model to measure the differences in risk between program modules, and determine when the risk of failure no longer justifies the cost of testing. Weyuker et.al,[8] compares the fault detection capabilities of partition testing and random testing.

Results of studies in software reliability can also be applied in determining test selection strategies. Studies performed by Iyer et.al.[9], [10] found that incremental risk of a software failure increased exponentially with increasing workload, and that in a network of machines, errors are highly correlated across machines due to shared resources. Based on these results, suites of tests should include heavy workload conditions with high user demands. The probability of finding faults can be increased by generating these high user demands with application programs performing heavy IO. The suite design should also include a large number of tests which focus strongly on shared resources. Work on defect classification by Chillarege et.al.,[11] is valuable in aiding the development of effective tests. Munson et.al.[12] propose discriminant analysis for the detection of fault-prone programs as a guide for focused testing.

A definition of **testing coverage** is provided by Levendel[13]. Test coverage is a measure of defect removal effectiveness. He defines testing coverage during a time interval as the ratio of the defects found during the interval to the total defects existing in the system at that time.

Recent studies have explored white box analysis from black box test design. Joglekar[14] studied white box data, but did not include systematic methods. Levendel[15] provided systematic methods, but did not include a complete mapping to the source code. This paper includes both a complete mapping to individual black box test scripts, and a systematic method for analysis.

Attempts to provide coverage or software construct-based tools (dependent on the programming language being used) have been unproductive in large system verification. The tools and methods of this paper are dependent only on the databases which track testing, faults, repairs, and source file modifications. The methods discussed in this paper can be used for any software system regardless of its programming language. Methods here provide specific feedback on each individual failed test. This feedback can be used to improve testing.

3. SOURCES OF DATA

The terms Functional Area (FA) and SubSystem (SS) will be used extensively throughout this paper. SS refers to closely related software components which make up a subset of the overall product. A complete, nonoverlapping partitioning of the product can be obtained using the SSs. Each SS contains a unique set of source files (e.g., Src 1,n refers to the nth source file in SS 1). Typically, all SS components reside on the same mainframe computer. FA refers to user-oriented segments of functionality. FAs vary significantly from one another in size and complexity. A FA may be a subset of a SS, or may span several SSs. FAs may also overlap one another. An example FA which expands several SSs might be "Administration".

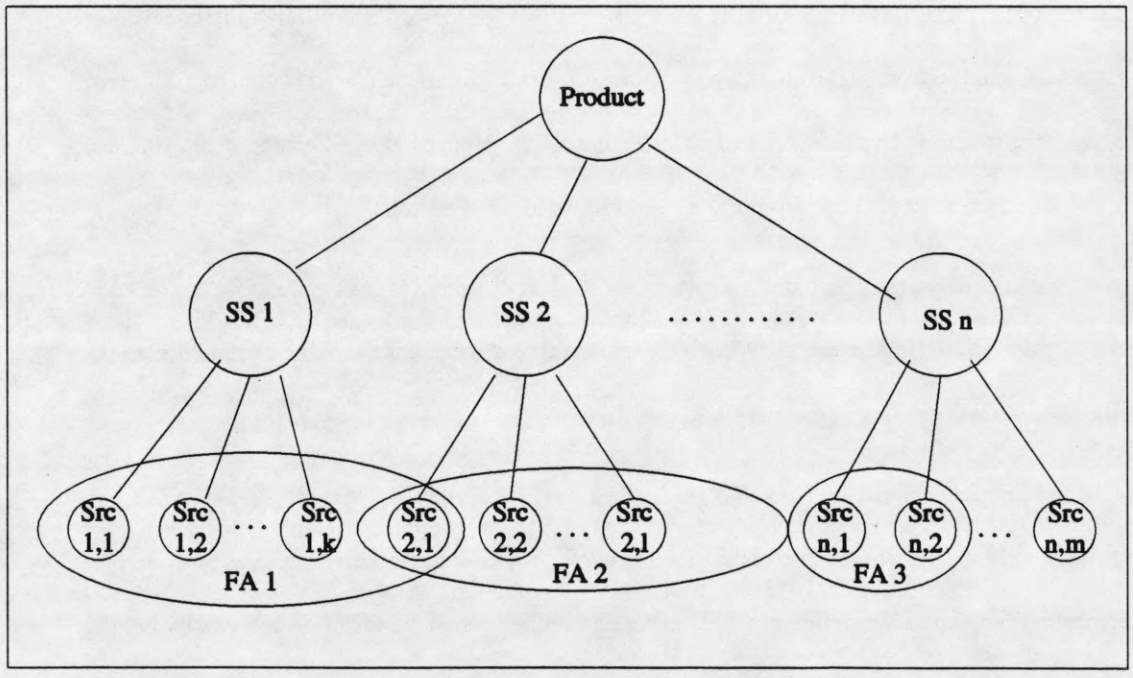


Figure 1. Subsystems and Functional Areas

The data comes from AT&T testing, fault, and repair databases called the Test Script Database (TSD), the Internal Modification Request Tracking System (IMRTS), the Extended Change Management System (ECMS), and the ECMS Source Code Change System (SCCS).

At AT&T, software faults are documented with Internal Modification Requests (IMRs). IMRs are also used to introduce new features (enhancements). The IMR has a standard format which includes data such as the environment and context of the error, the severity, the software release and software application, date and type of problem, the phase of the development cycle in which the problem was found, etc. In this paper, the IMR database is referred to as the fault database.

Modifications to the SoftWare (SW) product (fixes and enhancements) are done using the ECMS via a Modification Request (MR). The MR also has a standard format which includes the product subsystem, solution description, source files modified, products affected, dependencies, and test procedures. In this paper, the MR database is referred to as the repair database. ECMS SCCS contains the actual delta file source modifications. A delta file contains the changed (delta) code to be applied to the underlying source file. The number of deltas per repair can be used as one measure of the repair effort. From the ECMS SCCS, NCSL, load, and physical information can be extracted.

A brief scenario of a typical usage of these databases is given here. A test is written from the feature or requirements document and is entered into the test database. The test is written independent of the design or code (hence the term "black box" test). The test contains a list of the FAs tested. When the software has been developed, developer and integration tested, it is delivered to the test laboratory. The black box test is run. If the test fails, an IMR is opened in the fault database. The test execution results are then entered into the test database to reflect the tests status and fault number. The developer diagnoses and fixes the problem. Source file modifications are made. The repair record is submitted to document the source changes. The fault record is then updated to include the repair number. The fault record is closed, and the test is rerun and its status is upgraded to "passed".

The above scenario supports a linkage from black box test to product internals. The test record can be linked to the fault record(s) via the fault number(s). The fault record is linked to the repair record(s) via the

repair (modification) number(s). The repair record is linked to the source delta file(s) via the source code name. This linkage is shown in the picture below:

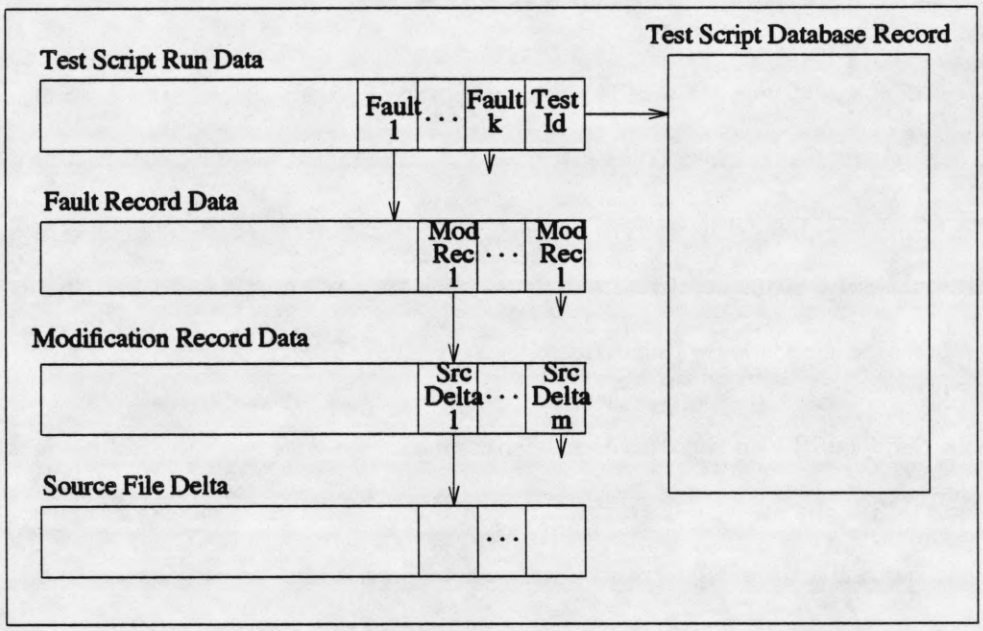


Figure 2. Test Run to Source Code Linkage

The above databases were merged into a single flat integrated database, to enable the proposed analysis to be performed. The analysis of the relationships between black box test data and white box product data was greatly facilitated by the integrated database.

3.1 Construction of the Integrated Database

In order to facilitate the analysis of testing data and its relationship to the product, a flat integrated database was constructed containing:

- Test run results data (11 attributes)
- Test script data (4 attributes)
- Fault data (90 attributes)
- Repair data (3 attributes)
- Source code delta file data (5 attributes)

The construction of this database began by extracting the execution results for all failed tests. Next, test script keywords, fault data, repair data, and code source file data were appended to each failed test tuple. A total of 123 attributes resulted from each failed test execution. Appendix 2 contains an example of one tuple (one execution of a test) in this database. Note that if a single test was executed and failed more than once, it would have multiple entries in the database.

The process of appending the records to the database causes the database to "explode" rapidly. This occurs because a given record from the test execution results file may contain multiple fault records. Therefore, the fault list in the single test execution tuple must be "unmerged". This unmerging creates redundant test execution results information. The same is true when unmerging the repair list from the fault record, and unmerging the delta file information from the repair record. The resulting database "explodes" rapidly and has the following structure:

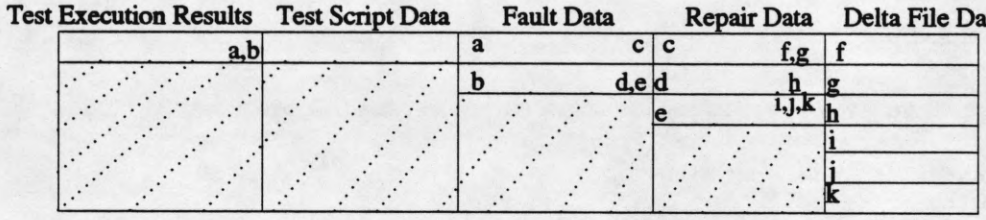


Figure 3. Propagation of Redundant Information in the Integrated Database

The shaded area indicates propagated (redundant) information. Care must be used when analyzing results due to redundant data.

4. DATA COLLECTION AND CONVENTIONAL ANALYSIS

This section gives an overview of the data collected. It then discusses some intuitive measures of testing effectiveness.

4.1 Description of Data

The data used are only small samples from a very large testing base. As such, the data is exemplary only, and is not reflective of any current AT&T products. Testing data was collected from 8 sample test teams. Six of the test teams tracked their testing using 10 FAs, and 2 tracked testing by 32 FAs for a total of 124 samples of data. These 124 samples of testing data are used throughout this paper.

Table 1 gives an overview of the data. For each team, the data collected includes number of tests written by the team, number of tests run, and total number of test runs (executions). Test execution data was collected for the first run and current status of the tests. (First Run (FR) data "freezes" the metrics as they were collected the first time the tests were run. The Current Status (CS) data contains the metrics from the most recent executions of the tests.) Data collected from the first run includes number of tests which passed without finding a fault, number of tests which passed although a fault was found, and number of tests which failed. Totals for each are given as well as team averages, ranges, and standard deviations. The following table shows this information for six of the test teams:

Attribute	Total	# Teams	Average	Min	Max	Std Dev
Tests Written	5818.00	6	969.67	352.00	2376.00	723.56
Tests Run	5799.00	6	966.50	352.00	2376.00	721.65
Total Runs	7182.00	6	1197.00	463.00	2732.00	825.22
FR Number Pass	5067.00	6	844.50	307.00	2147.00	653.47
FR Pass With Fault	113.00	6	18.83	5.00	41.00	14.75
FR Fail With Fault	617.00	6	102.83	32.00	207.00	63.29
CS Number Pass	5639.00	6	939.83	350.00	2329.00	703.90

Table 1. Overall Test Team Information

Certain general characteristics of the testing process can be deduced from this table. For example, each test

is executed, on the average, 1.24 (7182/5799) times. Overall, 13 percent (730/5799) of the tests found an error the first time the test was run. It is interesting to note that about 15 percent (113/(113+617)) of the errors were detected not because of the design nor as a direct consequence of running the test, but almost by chance (e.g. while setting up the environment to run a test, a fault is uncovered). Additionally, the range in number of tests run and number of tests failed is large. The smallest team ran only 15 percent (352/2376) of the number of tests run by the largest team. The minimum number of tests failed was 15 percent (32/207) of the maximum number of tests failed. To further understand the overall data, the remainder of this section will discuss some intuitively interesting relationships.

4.2 Test Distribution vs. Next Product Stage Fault Distribution

A first intuitive measure showing the need for effective testing is seen by plotting testing distributions against faults found in the next stage of the product cycle, after the above testing was completed. Appendix 1 contains six sets (one per team) of three graphs. The first graph shows fault information from the subsequent stage of the product cycle. (These faults represent all faults found at the next stage, not just the faults directly related to the testing performed by these six teams.) The second graph shows distribution of enhancement IMRs. The last graph shows testing distribution information. Mean, median, regression, and

average behavior lines are drawn. One example is shown below:

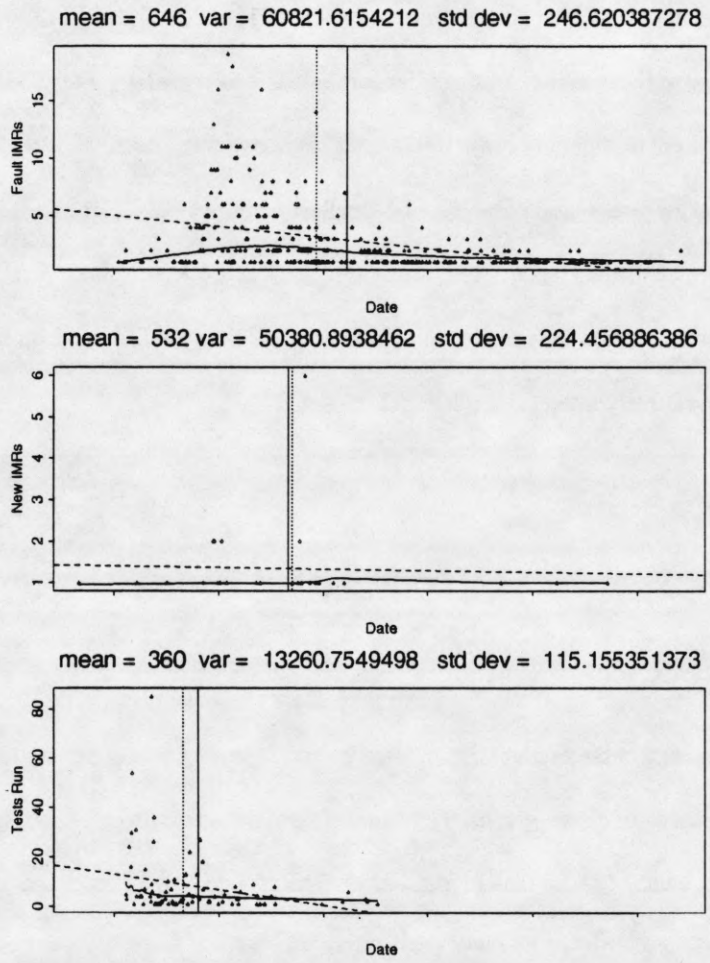


Figure 4. Test, Enhancement, Next Stage Fault Distributions

The distribution of the testing activity tends to be skewed. This is consistent with a quick start up for most test teams, and a comparatively lengthy "ramp down" and retest period. The interval from the mean test date to the mean fault date ranged from 108 to 557 days. The number of faults found in the subsequent stage of the product cycle is significant. Clearly there is room for improvement in the effectiveness of the testing.

4.3 Testing Data Measures

Simple intuitive relationships within the testing data can be plotted and analyzed. One such relationship is

the number tests written plotted against the number of test failures in a FA. For each of the 124 samples, a data point was plotted showing the number of tests run against the number of failed tests. Regression and average behavior lines are included.

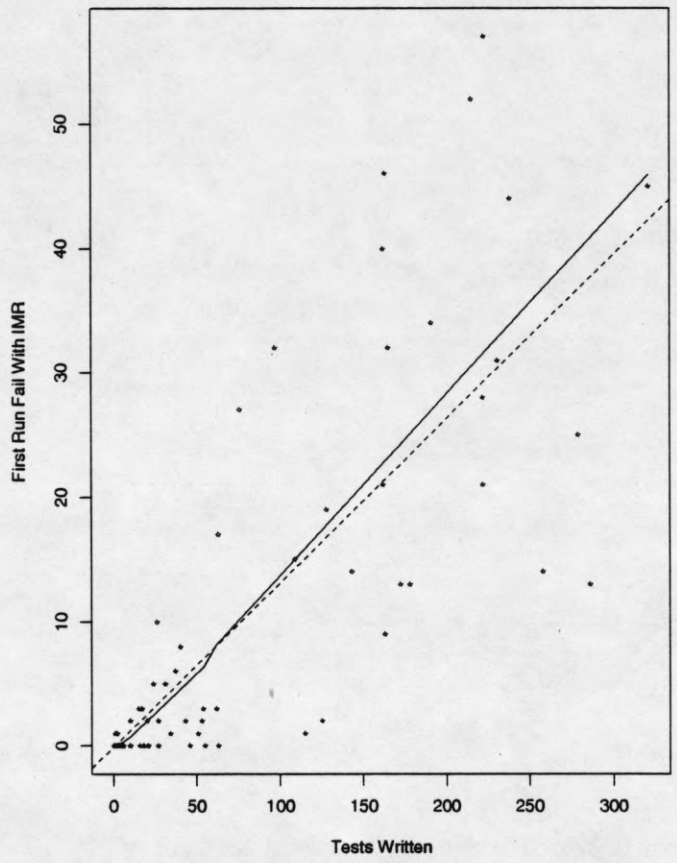


Figure 5. Test Sampling Effectiveness Measure

This intuitive measure of the overall data seems to indicate that the testing performed is effective in the sense that when more tests are run, more faults are found. However, Table 1 and Figure 5 both show significant variations in the amount of testing performed per team. In order to investigate whether the variation is simply due to the variation across different teams, or because of the distributions of the tests themselves, the testing performed is mapped to the product internals. Figure 6 shows the distribution of the number of test executions by FA. It is clear that the distribution of the tests across FAs is highly uneven

(mean=42, standard deviation=73).

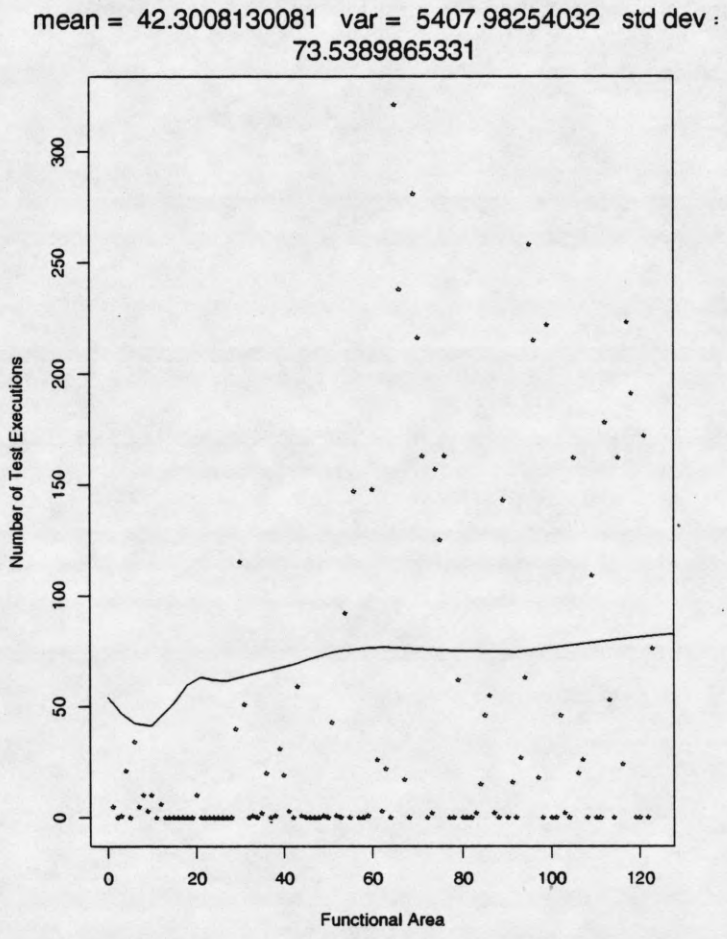


Figure 6. Test Executions Per Functional Area

Some FAs are being very heavily tested, while others are being lightly tested. The large variation in testing by FA raises some new questions. Is the large variation in number of tests run per team and per FA justified? When more tests are run, are more faults found per test? Can we obtain some insight into the effectiveness of the testing process? In particular, could all the faults found by this testing have been found by running fewer tests? Where should future testing be concentrated? To investigate these questions, a more detailed analysis of the data is required. The next section begins this investigation by categorizing the data by teams.

4.4 Data Categorized by Team

As discussed previously, 124 samples of data covering several FAs were collected from the testing results of eight test teams. In this section the 124 samples were categorized by team and within each team further categorized by FA. For six of the eight teams, the total number of tests run and number of tests passed on the first run were summed. These sums are shown in column three of Table 2. Column four shows the pass rates for the first run testing results. Columns six through nine show the FA average, range, and standard deviations of testing performed by the team. The ALL row of the table is inclusive of the testing done by all eight teams.

Team	Attribute	Total	%Pass	# FAs	Average	Min	Max	Std Dev
A	Tests Run	1395.00	85	10	139.50	0.00	321.00	119.25
	FR Number Pass	1187.00		10	118.70	0.00	267.00	102.43
B	Tests Run	367.00	84	10	36.70	0.00	163.00	57.27
	FR Number Pass	307.00		10	30.70	0.00	123.00	47.28
C	Tests Run	352.00	89	10	35.20	0.00	143.00	42.61
	FR Number Pass	314.00		10	31.40	0.00	124.00	36.97
D	Tests Run	761.00	83	10	76.10	0.00	258.00	103.25
	FR Number Pass	632.00		10	63.20	0.00	225.00	85.37
E	Tests Run	548.00	88	10	54.80	0.00	178.00	66.08
	FR Number Pass	480.00		10	48.00	0.00	165.00	59.18
F	Tests Run	2376.00	90	10	237.60	0.00	1528.00	438.02
	FR Number Pass	2147.00		10	214.70	0.00	1489.00	429.81
ALL	Tests Run	6731.00	87	124	54.28	0.00	1528.00	151.59
	FR Number Pass	5824.00		124	46.97	0.00	1489.00	143.64

Table 2. FA Test Run Statistics Per Team

Table 2 reflects the average behavior of the testing for each team. Using this table, test attributes (rows) can be compared to one another. A large variance in the number of tests run per team is observed. For example, the smallest test team ran only 15 percent (352/2376) of the number of tests run by the largest test team. All teams had a fairly consistent first run pass rate (from 83% to 90%). This seems to indicate consistency among the teams in selecting tests which are likely to cause product failures. The data contained in this table seems to indicate that the percentage of tests which fail remains fairly constant regardless of the total number of tests written. It is possible that running more tests simply finds more faults (which is somewhat supported by Figure 5).

Further information can be obtained from this table by looking at the averages, ranges, and standard deviations. The standard deviations are very large relative to the averages, indicating large variances in the number of tests run per FA by each team. For every team, there was some FA that was not tested at all (min=0).

The individual data points need to be analyzed to answer the testing effectiveness questions. A separate plot of testing pass rate against number of tests run was generated for each of the six teams. Ten data points were plotted for each team. Each data point represents the testing performed for a FA. A regression line was drawn for each plot to determine a representative behavior for each team. All data points equal to zero were excluded from the average behavior, regression, and mean percent pass rate calculations.

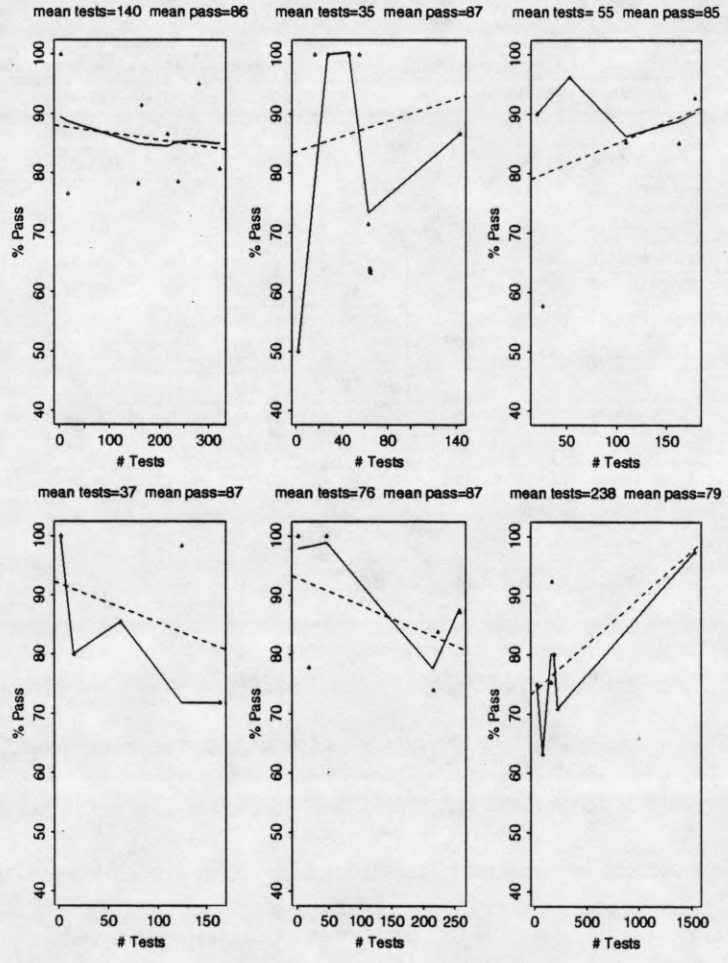


Figure 7. Pass Rate VS Tests Run Per FA For Six Teams

Previously, when looking at the average behavior of testing, it appeared that more testing in an area found

more faults. It is instinctive to compare these plots with Figure 5 which depicts the overall behavior. From Figure 5, it was observed that the number of tests failing increased with the number of tests run, resulting in a pass rate which remains approximately constant. In Figure 7, it is seen that while the average pass rate is constant and close among teams, for some teams, testing a FA more heavily did find more faults per test. However, for other teams, increased testing resulted in less faults found per test.

To continue this analysis, a mapping of results which is more closely aligned with the product is needed. In Figure 6, it was observed that a large variation in the number of tests run per FA existed. Now an analysis of the FAs is pursued in detail.

5. DATA CATEGORIZED BY FUNCTIONAL AREAS

In this section the 124 samples of testing data were categorized by FA, and within each FA the data was further categorized by team. For each of ten FAs, the total number of tests run and number of tests passed on the first run were summed. These sums are shown in column three of the table below. Column four shows the pass rates for the first run testing results. Columns six through nine show team averages, ranges, and standard deviations for the FA. The ALL row of the table is inclusive of the testing done on all FAs.

FA	Attribute	Total	% Pass	# Teams	Average	Min	Max	Std Dev
FA1	Tests Run	2440.00	93	6	406.67	46.00	1528.00	509.27
	FR Number Pass	2280.00		6	380.00	46.00	1489.00	500.79
FA2	Tests Run	918.00	77	8	91.80	5.00	238.00	87.34
	FR Number Pass	710.00		8	71.00	5.00	187.00	65.39
FA3	Tests Run	94.00	72	8	11.75	0.00	26.00	10.01
	FR Number Pass	68.00		8	8.50	0.00	18.00	6.84
FA4	Tests Run	223.00	71	8	22.30	0.00	223.00	66.90
	FR Number Pass	158.00		8	15.80	0.00	158.00	47.40
FA5	Tests Run	1008.00	87	6	168.00	62.00	281.00	72.52
	FR Number Pass	876.00		6	146.00	53.00	267.00	68.60
FA6	Tests Run	216.00	87	6	36.00	0.00	216.00	80.50
	FR Number Pass	187.00		6	31.17	0.00	187.00	69.69
FA7	Tests Run	179.00	92	6	29.83	0.00	163.00	59.84
	FR Number Pass	165.00		6	27.50	0.00	149.00	54.65
FA8	Tests Run	396.00	93	8	49.50	0.00	178.00	73.98
	FR Number Pass	370.00		8	46.25	0.00	165.00	68.47
FA9	Tests Run	86.00	98	6	14.33	0.00	53.00	19.65
	FR Number Pass	84.00		6	14.00	0.00	51.00	19.00
FA10	Tests Run	459.00	75	8	57.38	0.00	157.00	61.43
	FR Number Pass	343.00		8	42.88	0.00	123.00	47.59
ALL	Tests Run	6731.00	87	124	54.28	0.00	1528.00	151.59
	FR Number Pass	5824.00		124	46.97	0.00	1489.00	143.64

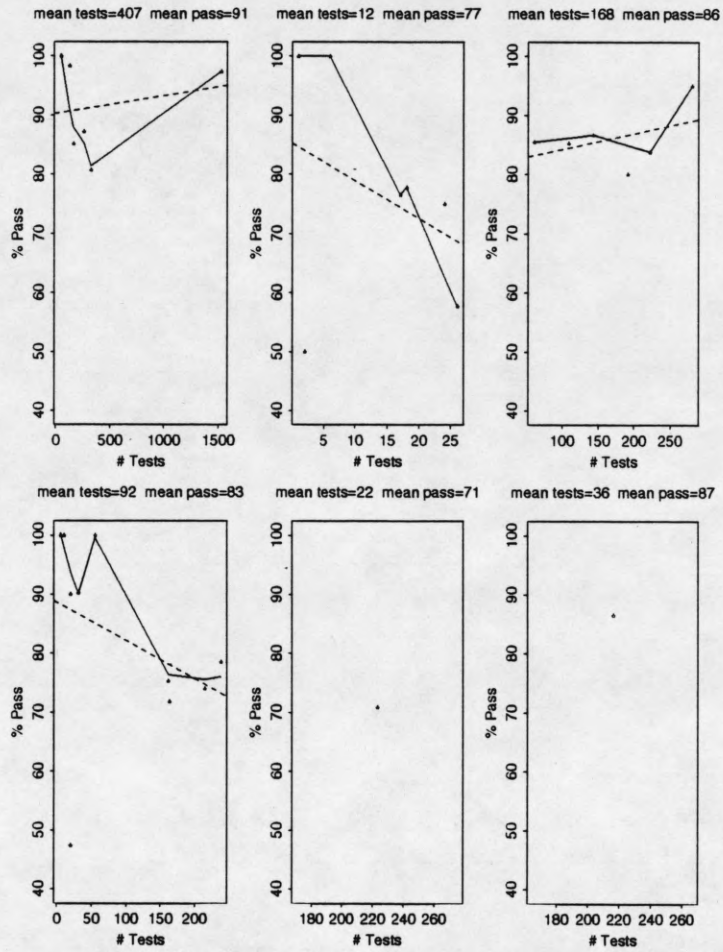
Table 3. FA Test Run Statistics Across Teams

Table 3 reflects the average behavior of the testing done for each FA. Using this table, test attributes (rows in the above table) can be compared to one another. A large variance is seen in the number of tests run per FA. For example, the average number of tests run for the least covered FA was only 3 percent of the average number of tests for the most covered FA. In contrast to Table 2, the pass rates were not consistent, ranging widely from 71 percent to 98 percent. For many of the FAs, a team existed which did not test the FA at all (as shown by a zero minimum value). Some FAs were tested by only one team. For most FAs, the team which ran the minimum number of tests ran less than half of the maximum number of tests. For example, for FA 1 one team ran only 9 percent (46/1528) of the tests run by another team.

When analyizing the average behavior of the testing of a FA, it is difficult to draw any conclusions except that wide variations exist in the testing performed, and in the faults found per test. Again, the individual data points need to be analyized.

Figure 8 contains ten graphs, one for each of ten FAs. Each data point reflects the number of tests run by a team for the FA plotted against the pass rate of the tests run. A regression line was drawn for each plot to

determine a representative behavior for each FA. When only one non-zero data point existed, the regression line was not calculated. All data points equal to zero were excluded from the average behavior, regression, and mean percent pass rate calculations. These plots are shown here:



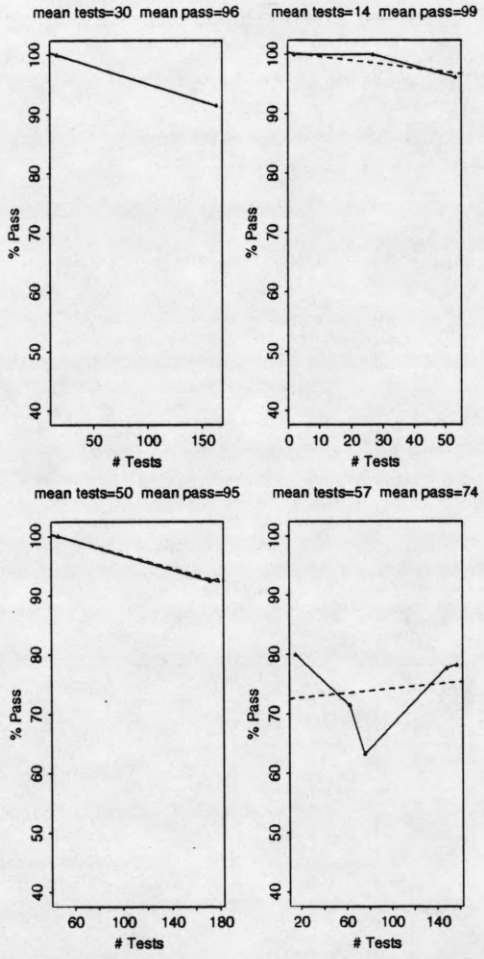


Figure 8. Pass Rate VS Tests Run Per Team For Ten FAs

These plots are consistent with Table 3 in that they show wide variation in the number of tests run. For five FAs, a decreasing pass rate is observed even though more tests are run, i.e. not only are more faults found when running more tests, but more faults per test are found. The rate of decrease in pass rate varies widely among the FAs. For three of the FAs the pass rate shows a slight increase with increased testing. The remaining two did not have enough data points.

In general, it would be expected that as more tests are run, the quality of the product would improve, resulting in a higher pass rate. As seen in the above plots, this is not the case for all FAs. It is known that running more tests is reflective of a team's expectation of the FA containing more faults. It is also known that for each FA, the smaller suites of tests are generally subsets of the larger suites of tests, and that most tests contain some degree of redundancy with other tests for the same FA. The above results indicate that

for some FAs, a large degree of redundancy exists within the tests scripts. (The size of the set of tests failing to a single fault increases as the number of tests increases.) Results show that being able to better predict which tests to run to verify FAs, and reducing the redundancy within the tests are critical.

5.1 Testing Data Analysis Summary

This study began with an overview of the FA and test team data collected. Overall data showed that in general, as more tests were run, more faults were found. However, large variations in the number of tests run per team and in the number of tests run per FA were observed. Data was then categorized by team. It was seen that the average behavior of the testing done by the teams appeared to be consistent as seen by the small range in pass rates. A need to align the analysis more with the product under test was evident. With the data categorized by FA, a wide variation in the number of tests run and in the pass rate was observed. The detailed analysis was successful in identifying significant causes for concern in the testing. For the majority of the FAs, the pass rate decreased with increased testing. Two reasons were suspected for this decrease in pass rate: the quality of the FA was lower, and large redundancy is created when the number of tests is increased.

Because the above conventional black box testing results are not based on net product repairs, conclusive recommendations which result in specific improvements to testing cannot be made. The next section will provide a white box analysis of the testing results based on product repairs.

6. IMPROVED WHITE BOX ANALYSIS

This section proposes white box analysis methods for black box testing, utilizing the integrated database. The method is used to identify redundancy in testing, and quantify the effectiveness of testing in terms of the net impact on the product. The error prone SSs and source files found as a result of the testing are also identified. The cost of repair due to the test effort is also evaluated. This section also provides a correlation

mapping between black box test FAs and product SS.

6.1 Analysis of Integrated Database Records

Team C selected 352 unique tests spanning several FAs to verify the product. Four hundred seventy six test executions of these 352 tests were completed. Fifty three of the 476 test executions failed. Using the method discussed in Section 3.1, the flat, integrated database of test, fault, repair, and source file information was generated. The following table is an overview of the resulting integrated data.

Unique Tests	Test Runs	Failed Runs	Fault Records	Repairs	SSs	Source Files	Deltas	NCSL
352	476	53	20	7	5	11	24	115

Table 4. Summary of Testing Impact to Product

Twenty defect reports resulted from the 53 failed test executions. Eleven unique source files were repaired as a result. A very conservative measure of testing efficiency can now be quantified in terms of the ratio of repairs to tests. Using this measure, the efficiency of this test effort is 2 percent (7/352). A test team goal is to write only fault reports which result in a repair. Fault reports which do not result in a repair increase the cost of repair with no value added. A simple measure of fault record effectiveness is the ratio of repairs to fault records generated. The test team's fault record effectiveness is 35 percent (7/20).

Table 5 shows several relative frequency tables for various categories of test, fault, and repair activity.

Distributions							
FAs		SSs		IMR		MR	
FAs	Tests	Faults	SS	IMRs	Tests	MRs	IMRs
2	7	1	4	1	52	0	14
3	10	3	1	2	0	1	5
4	22			3	1	2	1
5	14						
Src File		Delta File		NCSL		Delta File	
Src	MRs	Delta	MRs	NCSL	Delta	Delta	Src
0	1	0	1	1	1	1	6
1	3	1	2	11	1	2	3
2	1	2	0	12	1	4	1
3	2	3	0	22	1	8	1
		4	2	27	1		
		5	0	42	1		
		6	1	?	1		
		7	0				
		8	1				

Table 5. Summary Tables of Frequency Distributions

The average test was fairly complex, covering three or four FAs. The number of FAs covered by each test ranged from two to five. Four SSs contained one fault, one SS contained three faults. All tests except one found only one fault (IMR). Several IMRs written resulted in no repair (0 MRs). To repair the defect required only one MR in all but one case. Most MRs required a change to more than one source file, however. This testing found at least one fault prone source file which was touched eight times (eight delta files).

From the integrated database, three tables were generated to show the relationships between testing, SSs, FAs, and source files. Each of these three tables below encompasses all eleven of the repairs made as a result of this team's testing. Recall that a path exists from test information to fault (IMR) to repair (MR) to source file (see Figure 2). The SS table below was generated first. Starting with the information contained in the repair record, a column was created for each SS in which a fault was found. Since each repair record identifies the affected SS, the number of repairs per SS can be counted (row 4). Starting with the repair record and working forward through the linked path as shown in Figure 2, the source file, deltas, and NCSL are easily determined (rows 1-3). Then, working backward through the linked path, the faults records pointing to each of the repairs can be counted (row 5). The tests pointing to each fault can be counted (row

6), and the FAs pointed to by the tests can be counted (row 7). The source file and FA tables shown below were constructed in a similar fashion.

	Subsystem Number				
	SS1	SS2	SS3	SS4	SS5
NCSL	27	1	12	75	?
Deltas	1	1	4	18	?
Src	1	1	3	6	?
MRs	1	1	1	3	1
IMRs	1	1	1	3	1
Tests	1	4	1	4	1
FAs	4	6	3	8	4

	Source File Number										
	Src1	Src2	Src3	Src4	Src5	Src6	Src7	Src8	Src9	Src10	Src11
NCSL	27	1	9	1	22	10	1	26	6	6	6
Deltas	1	1	2	1	8	2	1	4	1	1	2
SSs	1	1	1	1	1	1	1	1	1	1	1
MRs	1	1	1	1	1	1	1	1	1	1	1
IMRs	1	1	1	1	1	1	1	1	1	1	1
Tests	1	4	1	1	2	1	1	1	1	1	1
FAs	4	6	4	4	5	2	4	2	3	3	3

	Functional Area Number																							
	3	5	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
NCSL	0	50	47	12	12	35	0	22	0	0	0	0	0	1	22	0	1	1	0	50	38	1	50
Deltas																							
Src	0	5	6	3	3	3	0	1	0	0	0	0	0	1	1	0	1	1	0	5	4	1	5
MRs	0	4	3	1	1	2	0	1	0	0	0	0	1	1	1	0	1	1	0	4	2	1	4
SSs	0	3	3	1	1	2	0	1	0	0	0	0	1	1	1	0	1	1	0	3	2	1	3
IMRs	2	10	11	1	4	6	1	1	1	1	2	1	3	4	4	2	2	4	1	9	1	4	8
Tests	1	23	22	1	4	16	2	2	4	4	1	1	1	9	16	1	2	13	2	19	1	11	18

Table 6. Test, Fault, and Repair Relationships Quantified

The first table provides the data relative to a given SS. The effort or churn on SSs 3 and 4 was large (several deltas and source file modifications were needed to implement the repair). By summing across the "tests" row, it can be seen that from a SS perspective only 11 tests were needed to find the faults. By summing the across "FAs" row, it can be seen that these same 11 tests were designed to provide coverage of a FA 25 times.

The second table provides the data relative to a given product source file. The effort or churn on source files 5 and 8 was large (several deltas were needed to provide the repair). The number of NCSLs needed to

provide the repair on source file 1 was also large. Again, by summing the "tests" row, it can be seen that from a source file perspective, the number of tests needed to find the faults was only 15. By summing the "FAs" row, it can be seen that these same 15 tests were designed to provide coverage of a FA 40 times.

The final table provides the data relative to a given FA. For this team, the tests which failed covered 23 FAs. From the black box perspective provided in the final table, the testing for each FA can be justified since an IMR can be found which relates to each FA. Furthermore, it appears that there are several FAs which needed extensive testing due to high NCSL, number of source files affected, number of SSs and number of MRs. These include FAs 5 - 13, 27, 28, and 30. The first three FAs (FA 3, 5, and 10) are "standard" FAs commonly used for reporting results. The remaining FAs are often used within the test team to provide additional granularity. Since conventional analysis is often done at both levels, both are included here. By summing the "tests" row for the "standard" FAs (3,5, and 10), it seems that the number of tests needed is 46. By summing the "tests" row for all FAs covered by these failed tests, it seems that the number of tests needed is 174.

The redundancy problem of black box, FA oriented testing now becomes clear. Due to the overlapping information at the FA level, the effort needed to find faults is exaggerated. This redundancy is carried into the test scripts themselves. A simple measure of this redundancy is the ratio of number of failed tests to minimum number of tests needed to find the faults. For this testing, the redundancy metric ranges from 4.2 (46/11) to 15.8 (174/11). When a white box testing data analysis is used, the effort needed to find the faults is kept in proper perspective, and redundancy of tests will be reduced.

Based on the integrated database, very specific actions can be taken by the test team:

1. The black box test team can request that additional unit level testing be done on fault prone source files (1, 5, and 8).
2. The team can request additional feature level testing be done in SSs 3 and 4.
3. The future cost of test can be optimized. Clearly there was unproductive effort spent in system test as 476 test runs (at high expense) covering 30 FAs, and many SSs and source files resulted in only 11

source file modifications. Many FAs and SSs were tested with no faults found. Prediction methods need to be developed to reduce unproductive, costly test effort.

4. The cost of repair can be optimized. Many fault records were opened without resulting in faults repaired in the product.

Typical relationships between FA, test, fault, repair and source file provide the basis for a testing effectiveness measurement. This will be discussed next.

6.1.1 Test Effectiveness Measurement Typically, a test is related to several FAs. For a failed test, one to several faults may be found. For each fault, usually only one repair results. The repair typically affects one to several source files. From these relationships, many composite diagrams can be drawn. The most common composites for failed tests are shown below. The diagram on the left shows a test which failed and resulted in a repair. The right diagram shows a test which failed, but no repair resulted (e.g. pilot error):

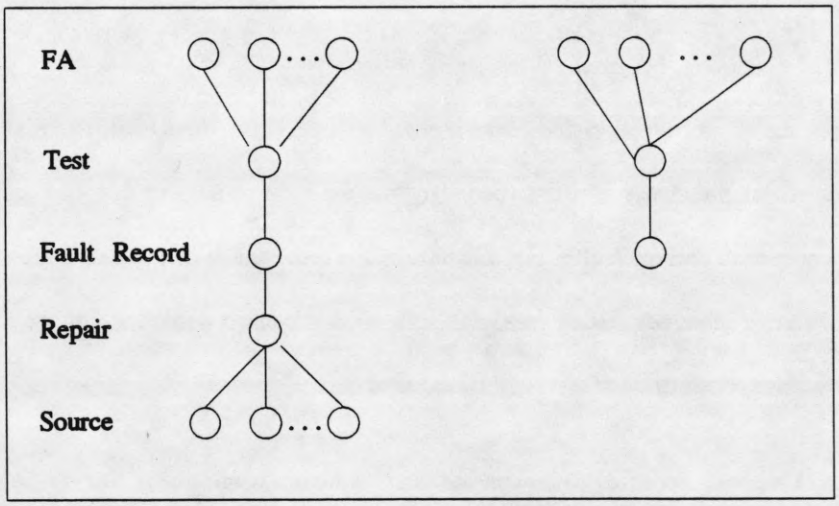


Figure 9. Typical Failed Test Relationships

In order to reduce redundancy and obtain the most orthogonal mapping, the effectiveness measures should be obtained by starting with the repair record, and following the database links upward to FA, and downward to source file. This explains why the minimum number of tests was found in the SS table above and not the source file table.

This section discussed the data contained in the integrated database. Using the integrated database, testing

efficiency was measured at 2 percent, fault record effectiveness was measured at 35 percent, and test script redundancy was measured at 4.2 to 15.8. Error prone source files and SSs were identified. To aid in prediction of which tests to run and how many, a mapping from test FA to product SS is needed. The next section will discuss a correlational mapping.

6.2 Correlation Mapping of Test Script Functional Area to Product Subsystem

Because of the lack of mapping from FAs to SSs, it is difficult to predict with confidence which tests should be selected, or how many tests are needed. The ability to make such predictions is a critical part of making testing more effective, and reducing the cost of testing. This section documents a procedure which can be used to determine the relationship of running black box tests for a given FA and finding faults in some given product SS. The FAs and SSs were extracted from the integrated database. A sparsely populated existence matrix was built where a column represented a SS or FA, and a row represented a test execution. The matrix contained 51 SSs and 23 FAs (74 columns) for 53 unique test executions (53 rows). A "1" in the matrix indicated that a SS or FA occurred as a result of running a test script. An example tuple of the matrix can be found in Appendix 3.

Correlation coefficients were then determined from the matrix. This resulted in a 74 by 74 matrix of coefficients, where $[i,j]^{th}$ element corresponds to the i^{th} column and the j^{th} column of the existence matrix. An example row of the FA/SS correlation coefficient matrix can be found in Appendix 4.

The following is a scatter plot of the correlation coefficients for one example SS only:

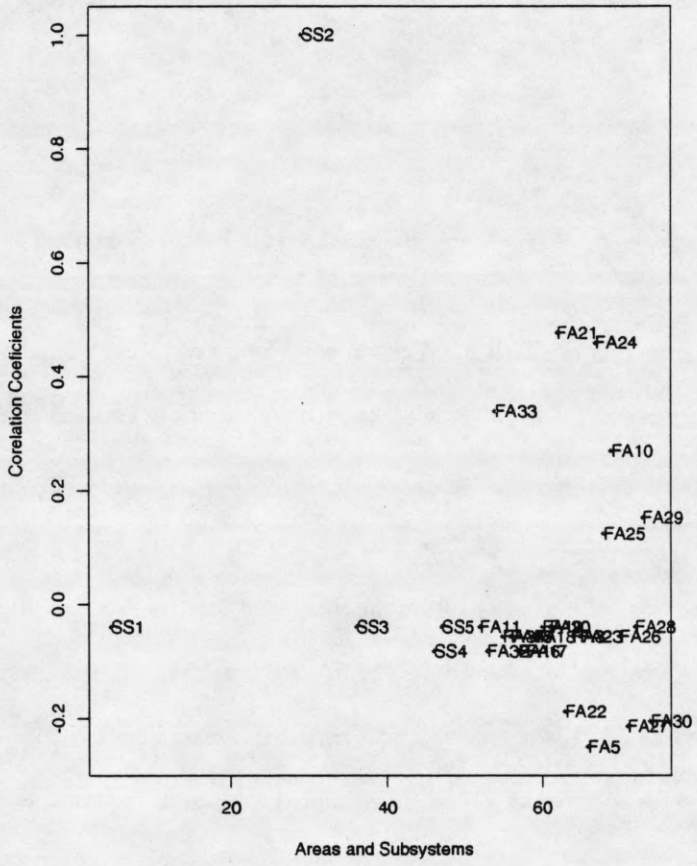


Figure 10. Correlation Coefficients for One Subsystem

Since the plot was produced for SS 2, the correlation coefficient for SS 2 is obviously 1.0. From the limited data used, the data shows the FAs which are more likely to find faults in SS 2 (high correlation), as well as the FAs which are less likely to find faults in SS 2 (low correlation). This method provides the mapping needed between product SS and test FA. Using this method, better prediction of tests which find faults can be made by selecting tests from a FA which is highly correlated to a SS undergoing major modifications.

In addition to FA and SS, the integrated database opens many opportunities to explore various relationships. A few examples include:

- Fault relationship of SS to SS.
- Relationship of FA to Source Files.
- Relationship of FA to NCSL.

In this manner, white box methods can be used to study the effects of black box customer-oriented system testing.

7. ADAPTIVE TESTING PROCESS PROPOSED

The methods and results obtained in this paper provide a foundation for more effective, adaptive testing processes. The above results lead to the following proposed process for adaptive black box large systems regression testing:

Using information of modified source files as key to a testing knowledge database, initial test selection is performed. The following steps are then followed:

1. Execute tests.
2. Enter results into the test database.
3. Enter fault information into the fault DB.
4. Enter repair information into the repair DB.
5. Generate the integrated database.
6. Perform analysis of integrated database:
   - — Product/Test/Fault/Fix Relationships
   - — Correlation
   - — Regression
   - — Statistical Process Control

7. Direct attention to fault prone source files.

8. Direct attention to fault prone SSs.

9. Add Test-FA-SS-Source knowledge to knowledge base.

10. Add/modify/delete regression tests.

11. Perform revised test selection using new information.

12. Repeat until testing is complete.

This process can be used to reduce both the cost of test and the cost of repair. Perhaps most importantly, the revised process has no impact on testers in the way tests are run and results reported. No changes to existing databases are needed. The integrated database and subsequent analysis can be implemented with tools which "run on top of" the existing test, fault, and repair databases.

8. CONCLUSIONS AND RECOMMENDATIONS

This paper studied black box testing and verification of large systems. The paper began by collecting data from several testing teams. An integrated database was generated containing test, fault, repair, and source file information. The paper provided evidence to suggest that conventional black box testing analysis needs to be augmented by additional analysis to be effective.

The white box analysis methods proposed use current available data and can aid in cost reduction and quality improvement. The methods proposed will result in significant cost and quality improvements. The white box data contained in the integrated database was studied in detail. Conservative measures of effectiveness were discussed. Testing efficiency was measured at 2 percent, fault record effectiveness was measured at 35 percent, and test script redundancy ranged from 4.2 to 15.8. Additional benefits from the methods proposed include the following: 1) Error prone source files and SSs are identified. 2) The integrated database provides a platform for development of additional robust analysis techniques. 3) The proposed methods are independent of the product under test. 4) The methods allow critical evaluation of

various black box testing techniques. A correlational mapping of test FA to product SS was completed. A new adaptive testing process based on real-time generation of the integrated database was proposed.

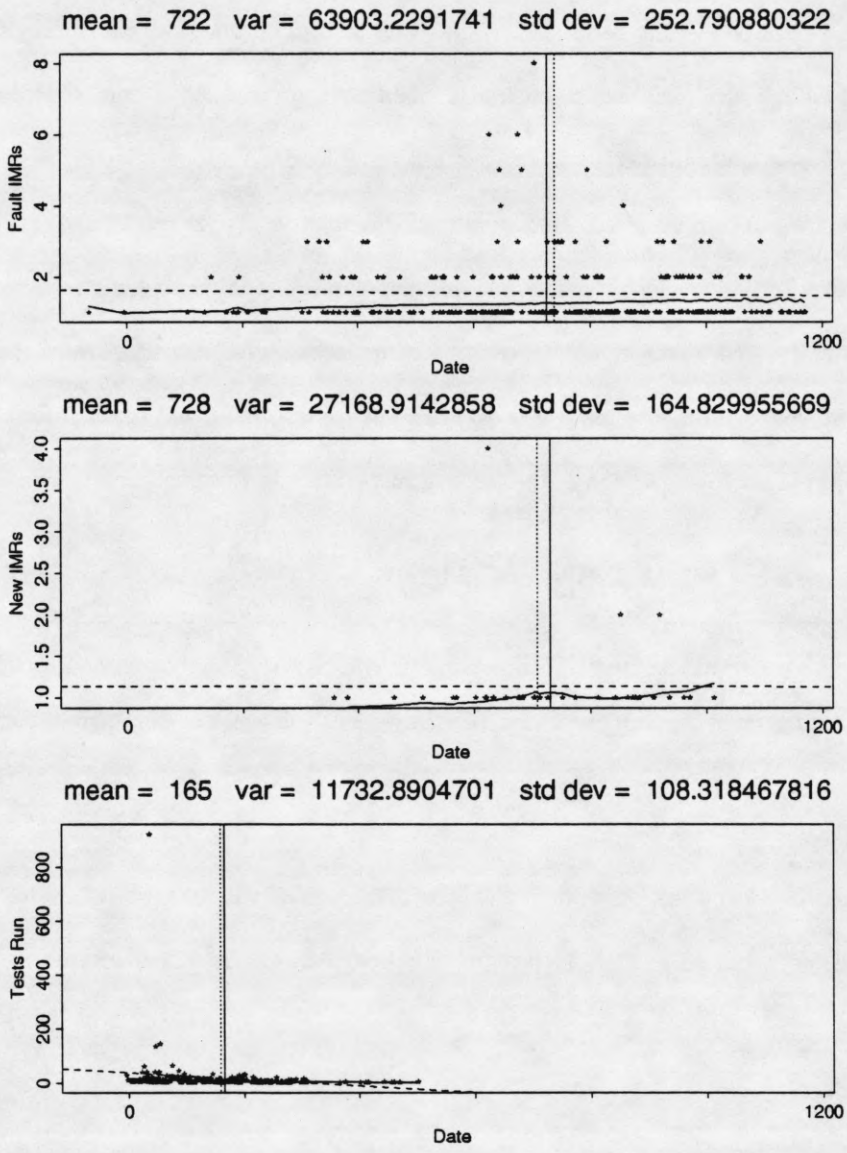
Black box tests are a very important part of product verification. It is important that suites of tests are designed purely from the customer/feature perspective in order to ensure that the customers expectations of the product are met. The black box testing performed by the teams studied in this paper was successful in that "customer oriented faults" were found which otherwise might have affected the customer. However, the results of this study show that once the black box test has been designed and developed, white box methods then need to be used to map the test to product internals in order to analyze results and provide a basis for future prediction. Testing is very expensive, and this mapping is critical in predicting which tests are most likely to find faults, and eliminating the execution of unnecessary or redundant tests, thereby significantly reducing the cost of test. The integrated database linking test, fault, and repair information greatly facilitates the effort of improving testing and redirecting future black box testing.

ACKNOWLEDGEMENTS

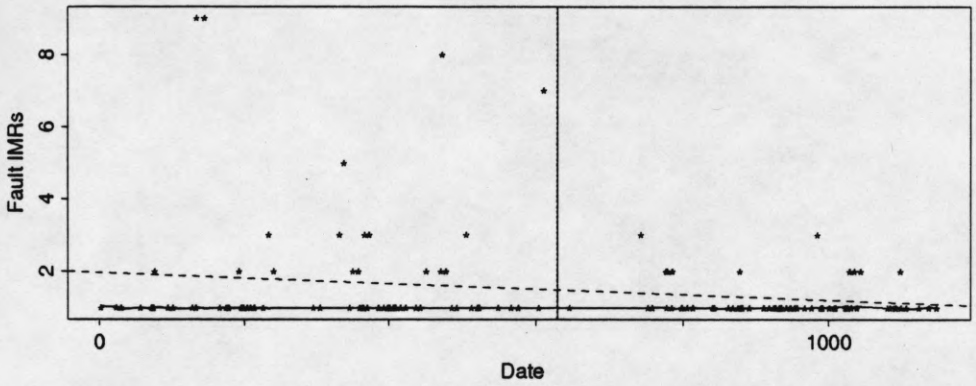
The authors would like to thank James Yu for his help in MR data collection, Steve Ohnsman for providing the description of the structure of the test data, and Bill Jones and Inhwan Lee for their thoughtful comments.

This work was supported in part by the Office of Naval Research under Grant N00014-91-J-1116. The content of this paper does not necessarily reflect the position or policy of the Office of Naval Research.

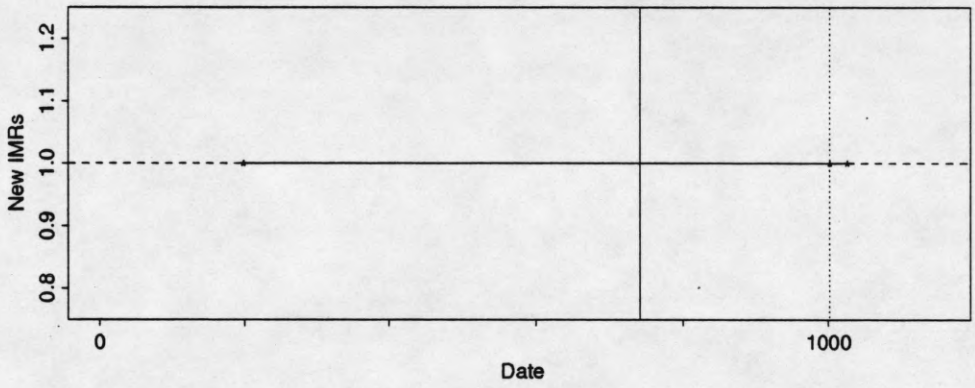
Appendix 1 - Distribution Comparison of Tests and IMRs



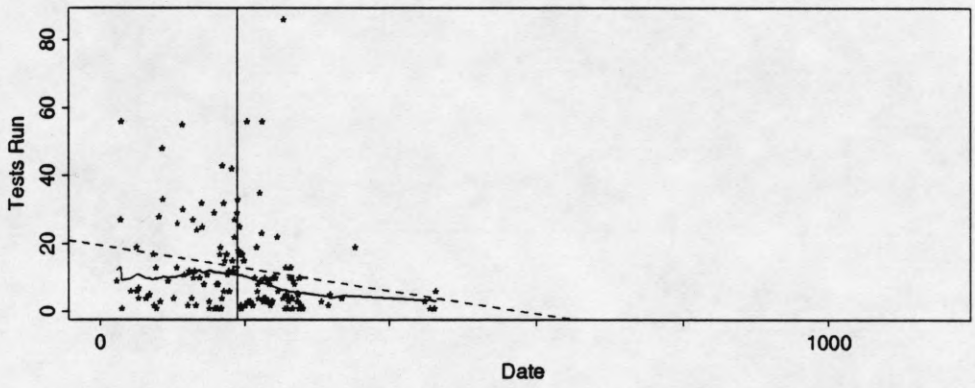
mean = 630 var = 119420.301491 std dev = 345.572425826



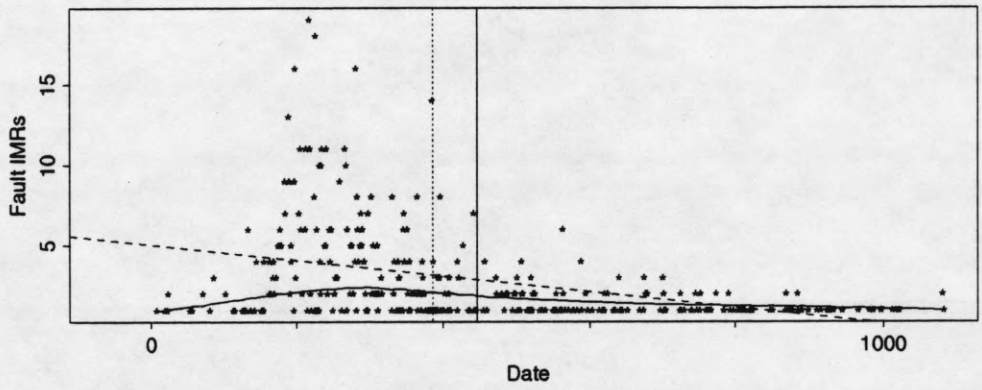
mean = 742 var = 221552.333334 std dev = 470.69346005



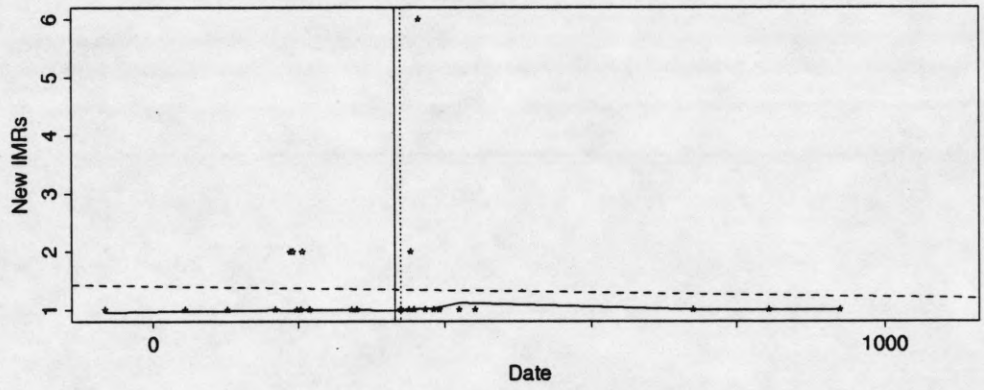
mean = 188 var = 7569.12100299 std dev = 87.0006954167



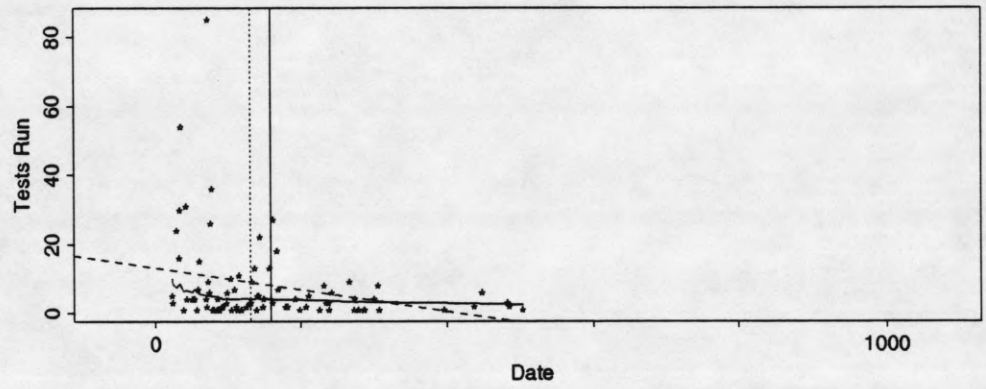
mean = 446 var = 60821.6154212 std dev = 246.620387278



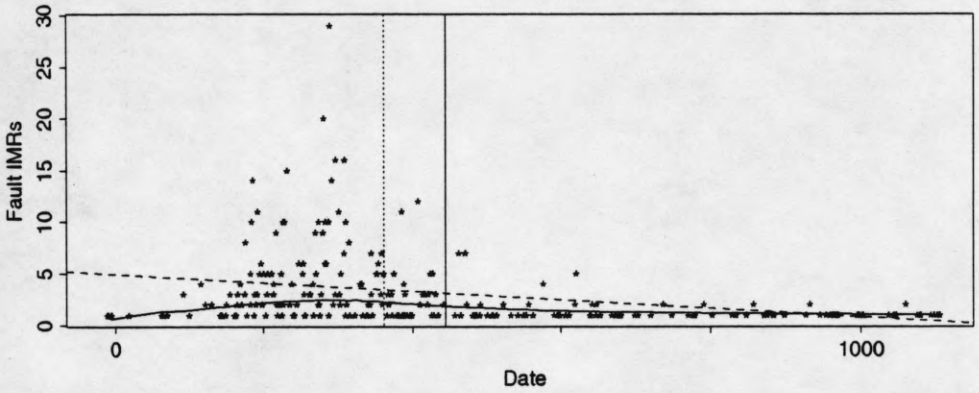
mean = 332 var = 50380.8938462 std dev = 224.456886386



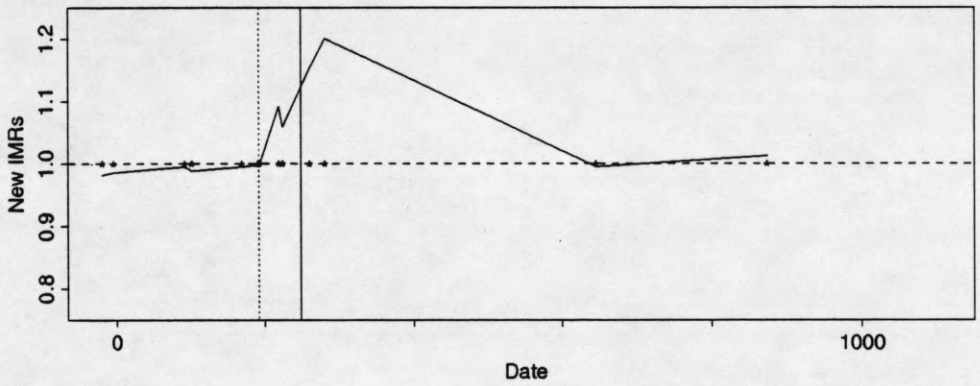
mean = 160 var = 13260.7549498 std dev = 115.155351373



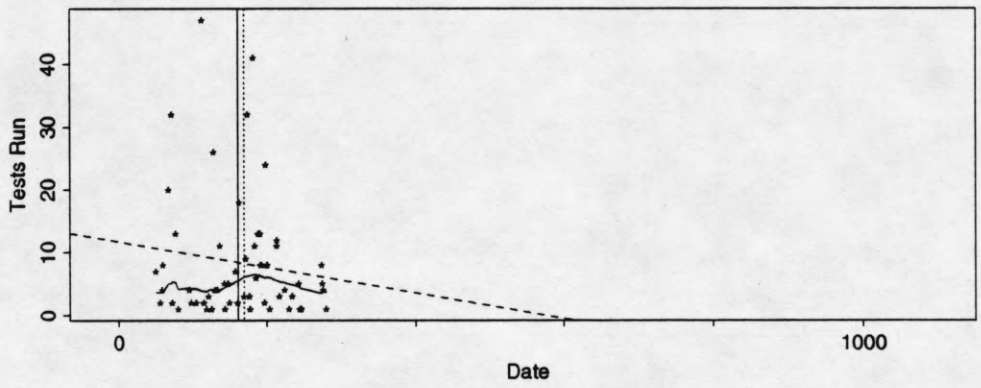
mean = 444 var = 75590.1982767 std dev = 274.936716858



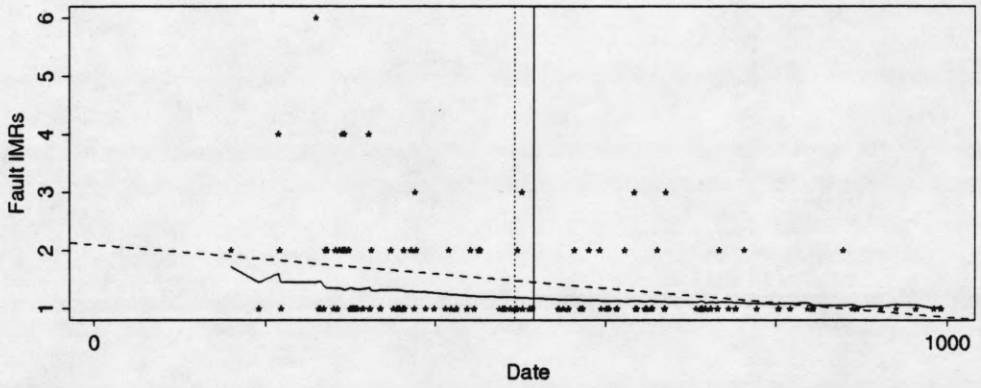
mean = 248 var = 62222.6666667 std dev = 249.444716654



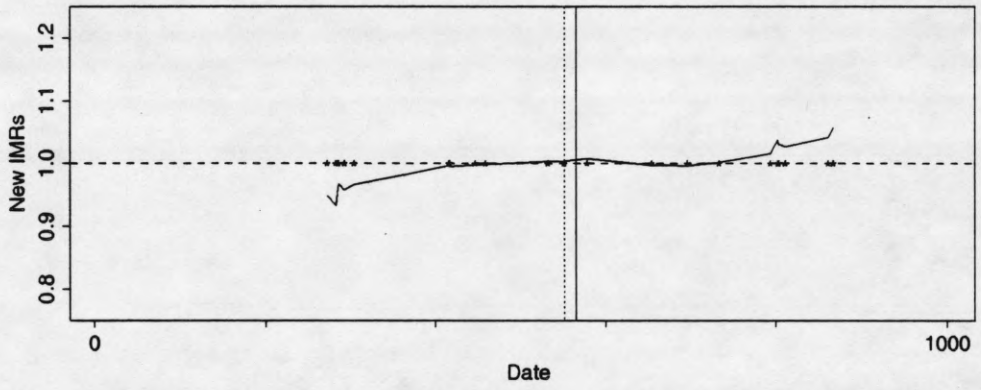
mean = 162 var = 3839.5977444 std dev = 61.9644877684



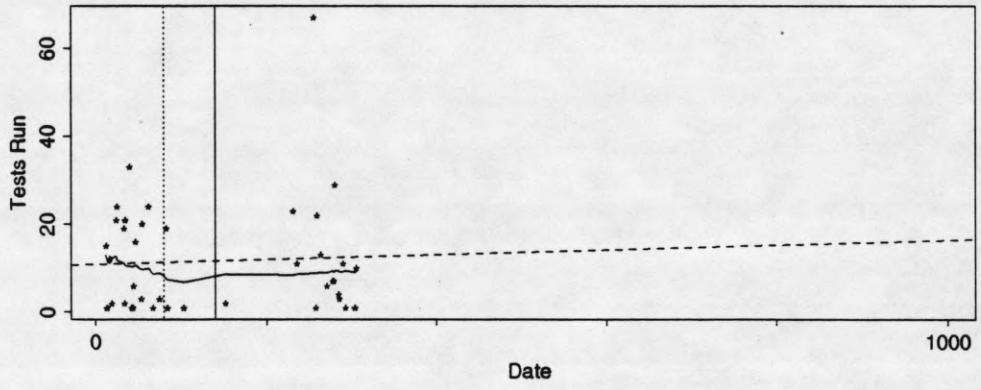
mean = 515 var = 43029.9550547 std dev = 207.436629009



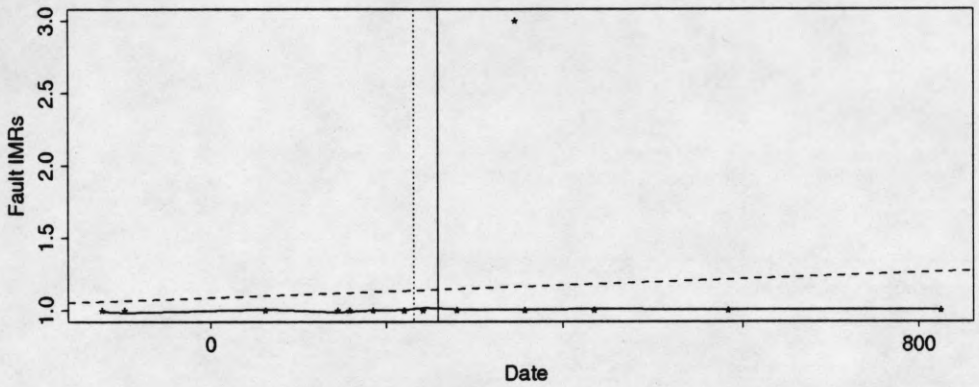
mean = 564 var = 45713.4904763 std dev = 213.807133829



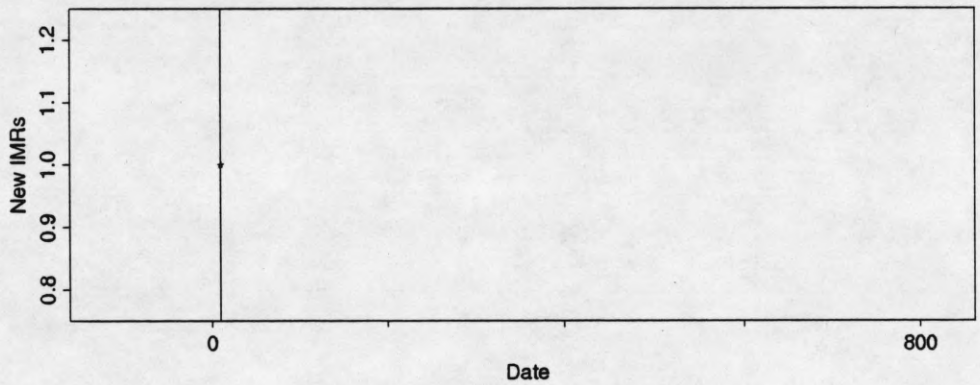
mean = 138 var = 13196.1948718 std dev = 114.874692042



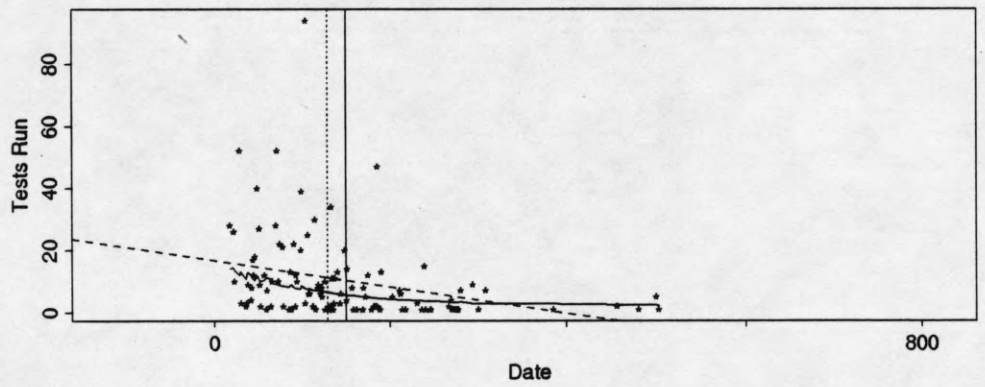
mean = 258 var = 62846.5549451 std dev = 250.692151742



mean = 10 var = NA std dev = NA



mean = 150 var = 10426.8357143 std dev = 102.111878419



Appendix 2 - Example Tuple: Integrated Testing/Product Database

PATH	TESTID	RDATE	RLOAD	RAPPL	WHO	ST AT US	IMR

REMARK	LOAD	LAB	FAREAS

TARGET	FEATID	FEATID2

empty	pcc	prio	sup	dev	mrno

start	orig	type	genfnd	bugnew	target

fixday	status	site	due	bwm	subsys

oday	chg	rev	history	bnday	cday

asday	asdept	osrc	atts	capdu	spare1

relimr	labtime	tester	fixtime	fnddur	sev

intro	dschg	dsname	dsday	genfeat	submit

tested	featuft	include	otrno	issaff	imrneed

phone	loc	dept	lastcmd	lastuser	sugsup

pccmark	spare2	ldi	spare3	spare4	rej

revreas	steam	steng	engstat	supmark1	supmakr2

odd	docneed	docmark	recur	verify	clday

spare5	spare6	spare7	spare8	assist	devast

supast	depast	spare9	imrpkg	scan	bwmtest

build	far	spare10	spare11	spare12	abstract

mILOAD	mNCSL 1	mNCSL 2	SRC	nHE LP1	nNCSL 1	nNCSL 2	nDATE

Appendix 3 - Example Tuple : FA SS Existence Matrix

	SS6	SS7	SS8	SS9	SS1	SS10	SS11	SS12	SS13	SS14	SS15	SS16	SS17	SS18
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	SS19	SS20	SS21	SS22	SS23	SS24	SS25	SS26	SS27	SS28	SS29	SS30	SS31	SS32
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	SS2	SS33	SS34	SS35	SS36	SS37	SS38	SS3	SS39	SS40	SS41	SS42	SS43	SS44
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0

	SS45	SS46	SS47	SS4	SS5	SS48	SS49	SS50	SS51	FA11	FA12	FA13	FA14	FA15
1	0	0	0	0	0	0	0	0	0	1	1	0	0	0

	FA16	FA17	FA18	FA19	FA20	FA21	FA22	FA3	FA23	FA5	FA24	FA25	FA10	FA26
1	0	0	0	0	0	0	0	0	0	0	0	0	1	0

	FA27	FA28	FA29	FA30
1	0	0	0	0

Appendix 4 - Example Tuple : FA SS Correlation Matrix

	SSFA	SS6	SS7	SS8	SS9	SS1
1	FA25	NA	NA	NA	NA	-0.0911921

	SS10	SS11	SS12	SS13	SS14	SS15
1	NA	NA	NA	NA	NA	NA

	SS16	SS17	SS18	SS19	SS20	SS21
1	NA	NA	NA	NA	NA	NA

	SS22	SS23	SS24	SS25	SS26	SS27
1	NA	NA	NA	NA	NA	NA

	SS28	SS29	SS30	SS31	SS32	SS2
1	NA	NA	NA	NA	NA	0.123299

	SS33	SS34	SS35	SS36	SS37	SS38
1	NA	NA	NA	NA	NA	NA

	SS3	SS39	SS40	SS41	SS42	SS43
1	-0.0911921	NA	NA	NA	NA	NA

	SS44	SS45	SS46	SS47	SS4	SS5
1	NA	NA	NA	NA	-0.187885	-0.0911921

	SS48	SS49	SS50	SS51	FA11	FA12
1	NA	NA	NA	NA	-0.0911921	-0.187885

	FA13	FA14	FA15	FA16	FA17	FA18
1	0.613779	0.301142	-0.130223	-0.187885	-0.187885	-0.130223

	FA19	FA20	FA21	FA22	FA3	FA23
1	-0.0911921	-0.0911921	0.445003	-0.432433	-0.130223	-0.130223

	FA5	FA24	FA25	FA10	FA26	FA27
1	-0.575788	-0.187885	1	0.62137	-0.130223	-0.491583

	FA28	FA29	FA30
1	-0.0911921	0.911114	-0.471587

REFERENCES

1. Kaner, C., "Testing Computer Software," TAB Books, Inc., ISBN 0-8306-9563-X, 1988.

2. Gallagher, K. B., Lyle, J. R., "Using Program Slicing in Software Maintenance," IEEE Transactions on Software Engineering, August, 1991.

3. Howden, W. E., "Error-Based Validation Completeness," Proceedings: 11th Annual Conference on Software Engineering," May, 1989.

4. Dunham, J. R., et.al., "Design for Validation: An Approach to Systems Validation," Dialog, NTIS, May, 1989.

5. Chow, T. S., "Tutorial Software Quality Assurance A Practical Approach," IEEE Computer Society Press, ISBN 0-8186-0569-3 1985.

6. Musa, J. D., Iannino, A. and Okumoto, K., "Software Reliability: Measurement, Prediction, Application," McGraw-Hill Co., 1987.

7. Sherer, S. A., "A Cost-Effective Approach to Testing," IEEE Software, March, 1991.

8. Weyuker, E. J., Jeng, B., "Analyzing Partition Testing Strategies," IEEE Transactions On Software Engineering, July, 1991.

9. Iyer, R. K. and Rossetti, D. J., "Effect of System Workload on Operating System Reliability: A Study on IBM 3081," IEEE Transactions On Software Engineering, Vol. SE-11, No. 12, December, 1985, pp. 1438-1148.

10. Tang, D., Iyer, R. K., "Impact of Correlated Failures on Dependability in a VAXcluster System," Proceedings of the 2nd IFIP Working Conference on Dependable Computing for Critical Applications, February, 1991.

11. Chillarege, R., Sullivan, M., "A Comparison of Software Defects in Database Management Systems (DB2,IMS) and Operating Systems (MVS)," Symposium in Fault-Tolerant Computing, FTCS-22, July, 1992.

12. Munson, J. C., Khoshgoftaar, T. M., "The Detection of Fault-Prone Programs," IEEE Transactions on Software Engineering, May, 1992.

13. Levendel, Y., "Reliability Analysis of Large Software Systems: Defect Data Modeling," IEEE Transactions on Software Engineering, February, 1990.

14. Joglekar, M. and Paruzynski D., "End to End Testing Database Experience," Proceedings of 5th Annual ATT Test and Verification Symposium, 1991.

15. Levendel, Y., "Improving Quality with a Manufacturing Process," IEEE Software, March, 1991, pp. 13-25.