NASA CONTRACTOR REPORT 166456

Trends in Software Reliability for Digital Flight Control

H. Hecht
M. Hecht

CONTRACT--P.O. A93024B
April 1983

NASA

NASA CONTRACTOR REPORT 155456

Trends in Software Reliability for Digital Flight Control

Herbert Hecht
Myron Hecht
SOHAR INCORPORATED
Los Angeles, CA 90035

Prepared for
Ames Research Center
Under P.O. A93024B

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035

TABLE OF CONTENTS

SECTION 1 - REQUIREMENTS FOR SOFTWARE RELIABILITY

Digital flight controls are assuming an increasingly important role in the operation of passenger aircraft. At present they are essential for only some flight phases, such as automatic landing. But greater dependence on automatic stabilization can provide increased fuel economy and speed, and thus there is a need to provide freedom from failures (including software failures) throughout the flight regime. These developments are already quite visible in military aircraft where automatic stabilization is essential for safety of flight in major portions of the flight regime.

Aircraft components essential to safety of flight are subject to FAA regulations, and compliance with these must be established in order to obtain a Certificate of Flightworthiness. As automatic flight controls become essential, they must comply with these requirements, the most important one of which is that any failure condition that would be catastrophic (to the aircraft) is extremely improbable [FAA70]. An advisory circular equates 'extremely improbable' with an expected failure frequency of 1E-9 per flight or flight hour (according to some interpretations this may require a failure frequency of less than 1E-10 per hour).

Since compliance must be demonstrated during a period when at most several thousand hours of flight time can be accumulated, demonstration of adequacy on the basis of observed failure frequency is completely impossible. For the hardware components, established methods of quantitative reliability analysis and fault tolerance are used to satisfy the requirements. The basis for such procedures in the software area is much more problematic, and a purpose of this report is to direct attention to improvements in analysis and software systems synthesis that may overcome the existing deficiencies. Present software for the limited extent of essential applications is usually certified by demonstrating that only a very restricted set of potentially catastrophic malfunctions may occur, and that there are checks or circumvention programs provided to guard against these. This practice is not believed to be applicable to future broader applications of digital flight controls.

SECTION 2 - SCOPE AND DATA SOURCES

It is intended to describe the state of software reliability for digital flight controls in the recent past and at present in quantitative and qualitative terms, to identify trends, and to point to approaches that promise further improvements and that may lead to a comprehensive certification methodology. Because very few reports on the reliability of flight control software have been published, data from a broader field of software applications is examined, and the few available datapoints from the flight controls field are discussed in the context of the general findings and trends. The present study is divided into two parts: a numerical evaluation of reliability and reliability trends, and an investigation of causes of software failures that has both quantitative and qualitative aspects.

The emphasis is on presenting findings that may lead to improvements in:

Software development methodology
Test methodology
Application of software fault tolerance
Quantification of software reliability
Data collection and analysis

The data sources utilized and some of their characteristics are summarized in Table 1. The first three entries in the table provide data for the numerical reliability assessment, and the last two entries together with the first one provide data for the evaluation of causes of software failures. The total database represented by these comprised 47 programs. Data for 28 programs in that set were sufficiently complete to permit the detailed analysis presented in this report. Significant characteristics of these programs are summarized in the Appendix (Table A).

TABLE 1 - DATA SOURCES

Source	No. of. Programs	Program Size	Fault Density*	Fault Types
Ames (Set 1)	15	391k	0.64%	Yes
Ames/(Set 2)	2	89k	0.52%	No
Goddard-SEL	11	812k	0.10%	No
Langley/MIPS	1	90k	N/A	Yes
RADC/TRW	3	> 1,000k	N/A	Yes

* Faults per equivalent executable assembly statement

The first entry comprises data from software development for the B-1 bomber and the air launched cruise missile (ALCM). The data were collected for NASA Ames and include some flight control and closely related programs (air data computer and navigation) [PRES81]. The Set 2 data were also collected for NASA Ames in a special effort to assist in the identification of failure modes in digital flight controls [ROCK81]. The two programs relate to automatic flight controls

and aircraft navigation, respectively. The NASA Goddard Software Engineering Laboratory (SEL) data were acquired partly from the agency itself and partly from the Data and Analysis Center for Software (DACS) at the Rome Air Development Center (RADC). These programs deal primarily with satellite telemetry and ground control of satellites [BAS177, CARD82, TURN81]. The fourth entry comprises data collected under a NASA Langley contract during the development of the software for the Metric Integrated Processing System (MIPS) and Vandenburg Air Force Base. The specific program generates sensor calibration and other initialization parameters for a missile launch monitoring application [HECH77]. The last dataset comes from a report prepared by TRW for RADC. The programs deal with satellite and missile applications [THAY76].

Throughout this report program size is expressed in equivalent executable assembly statements. For programs written in most high order languages (HOLs) the number of executable statements is multiplied by five to arrive at the assembly equivalent. This represents a conservative assumption for the hypothesis that the fault density in HOL programs is lower than in assembly programs that will be presented below. Analyses of the expansion on a CDC 7600 FORTRAN compiler indicate an expansion factor of eight [LAWS76]. For programs written in AED a conversion factor of three is used which is derived from a specific analysis of the programs involved here. Practically all data come from formal test programs, i. e., from the time between unit or module test and operational use of a program.

By combining data from several sources some biases are probably introduced into this study due to differences in failure reporting, thoroughness of testing, and maturity of programs when data collection terminated. Some auxiliary analyses are therefore carried out within a given sample to determine whether the overall findings hold. However, the ensemble of the programs provides a good basis for studying broad trends which are our primary objective.

In order to address the FAA requirements it is desirable to express software reliability in terms of failure rate (failures per unit execution time) which then can be compared with the numerical criterion in the advisory circular. This measure was not available for the programs of primary concern in this study. The readily obtainable numerical index for reliability was the fault density (faults per equivalent executable assembly statement). This is the quantity used as the reliability measure in this report, usually expressed as a percentage. By means of an 'uncovery factor' (the number of executions required to detect a fault) the fault density can in theory be converted to a failure rate but there are at present no generally accepted values for the uncovery factor [MUSA79].

Standard definitions of reliability terms are used in this report [IEEE82]. Specifically:

el5
Failure - The inability of a system or component to perform a required function within specified limits. A failure may be produced when a fault is encountered.

Fault - An accidental condition that causes a functional unit to fail to perform its required function.

Error - A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.

Thus, a fault causes a failure which is usually observed because of the error that results. In the investigation of causes of failures we will discuss fault categories although the looser term 'error category' is also recognized in [IEEE82]. The relationship between these terms is indicated in Figure 1. An example of the application of these concepts to digital flight control software may be represented by the following: an attitude calculation routine lacks protection against division by zero (this is the fault); a zero divisor is encountered (this is the trigger); and this results in an incorrect attitude output (the error). The coincidence of the fault and the trigger for that fault causes the failure. Reliability studies are concerned with the recording and avoidance of failures, and, as mentioned above, the preferred index of reliability is the failure rate. Since it was not possible to compute this for most of the programs in the data base, the analysis presented in the following section makes use of the fault density which represents a rough relative index of reliability.

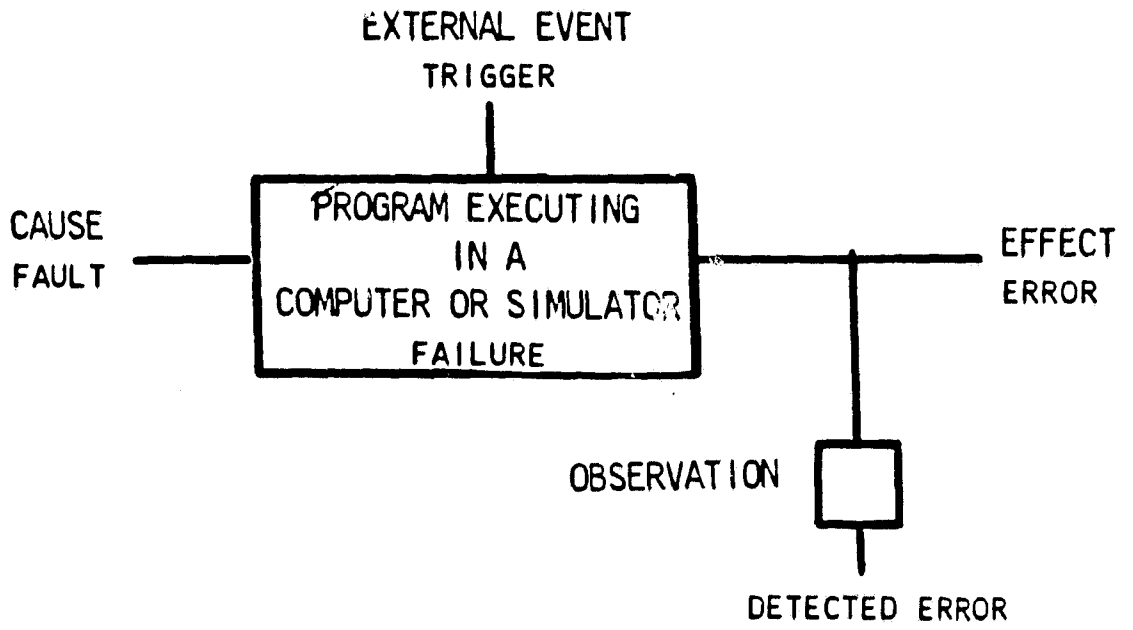


FIGURE 1 - BASIC RELIABILITY MODEL

SECTION 3 - ANALYSIS OF FAULT DENSITIES

The trend of fault density with year of program start for all software from the first three sources in Table 1, is shown in Figure 2. The fault density for each start year represents the total number of faults divided by the total number of statements in the corresponding programs. It is thus a weighted average rather than the mean of the fault densities of individual programs. The weighted average is used in all comparisons that involve groups of programs throughout this report. The trend curve is also fitted by weighting each data point with the associated number of statements. The faults that are considered here are those that resulted in errors or improper program execution. In some studies the number of software problem reports (SPRs) is used as an index of software quality. Since a problem report may be generated to require a documentation change, or to add a feature now desired by a user but not part of the original requirements, the total number of SPRs is not a good metric for reliability studies.

Because of the wide scatter of the points, no statistical measures of significance are appropriate. However, for programs started in 1977 the fault density is seen to be considerably reduced. The period between 1975 and 1977 saw much increased emphasis on software engineering (a disciplined approach to software development) and on the use of software tools (software programs that assist in the development or test of other programs). Also, HOLs came into much wider use during that period, and the effect of this is discussed separately below.

The contrast between pre-1977 software and more recent programs is further examined in Table 2. Although there is a considerable overlap in the range of fault densities observed for the to starting periods, there is an obvious trend to reduced fault density for the recent starts. The fault density of the flight control programs is below that of the whole group for pre-1972 starts but appreciably above the group average for the recent starts. Reasons for this latter deviation are discussed in connection with language use below. In either period the flight control programs are well within the range of the other programs.

TABLE 2 - EFFECT OF STARTING PERIOD ON FAULT DENSITY

Program Attribute	Pre-1977	Recent
No. of programs	7	21
HOL usage	14%	71%
Program size*	67k	1,224k
Fault density	1.59%	0.22%
Flight controls	1.19%	0.72%
Range of f. d.	0.63% - 11.66%	0.01% - 5.21%

* Equivalent executable assembly statements

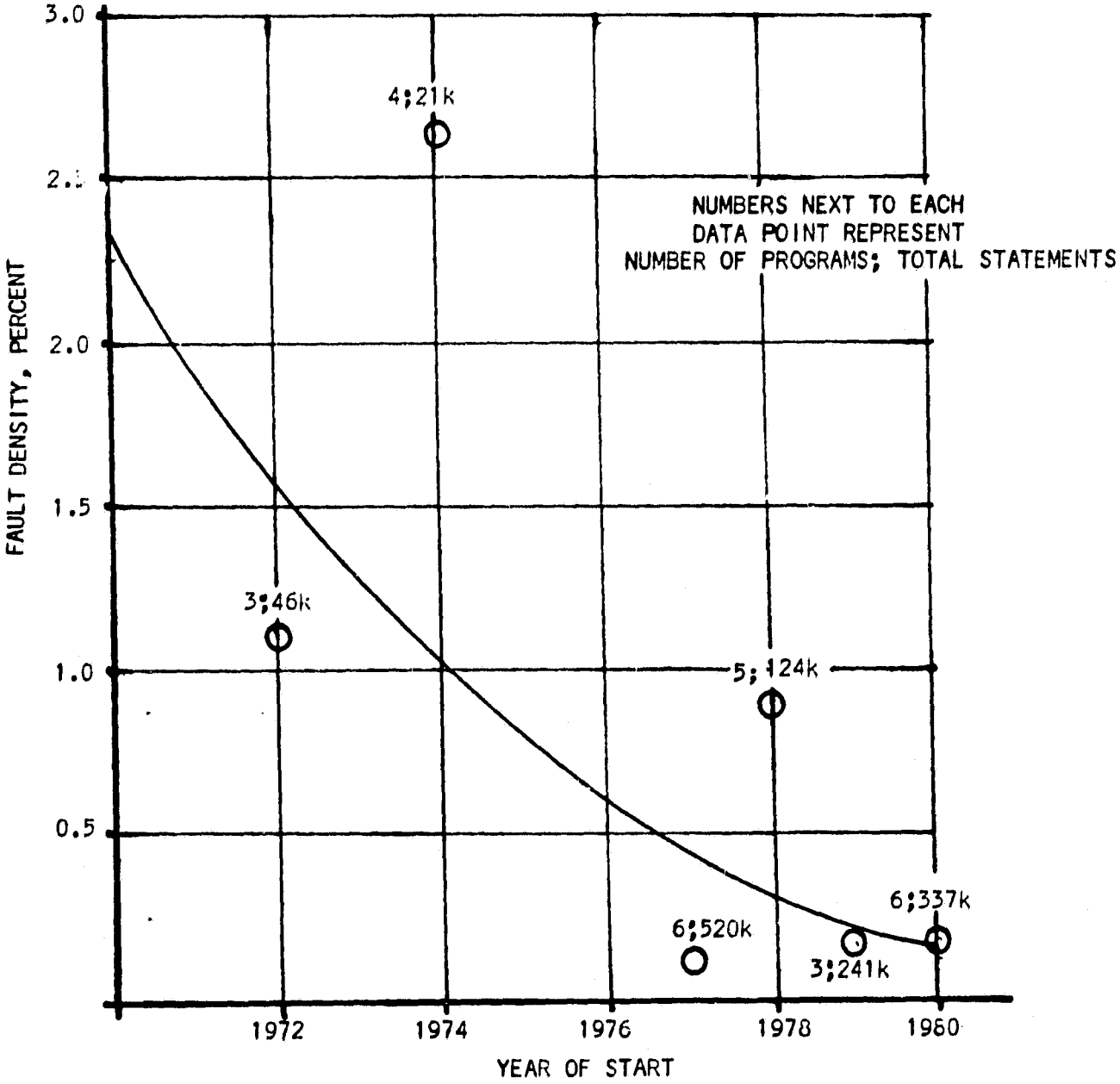


FIGURE 2 - TIME TREND OF FAULT DENSITY

The great increase in the use of HOLs during the recent period could be the significant cause for the improvement in reliability. Therefore a separate analysis of the effect of starting period on assembly programs is presented in Table 3. Although the reduction in fault density for recent starts is not as pronounced as for the total sample in Table 2, it is still present and can be seen in both the average and the range. Thus, the effects of disciplined software practices seem to have carried over even into assembly language programming. A similar comparison for HOL programs could not be constructed because only a single HOL program had been started prior to 1977.

TABLE 3 - EFFECT OF STARTING PERIOD ON FAULT DENSITY OF ASSEMBLY PROGRAMS

Program Attribute	Pre-1977	Recent
No. of programs	6	6
Program size*	34k	100k
Fault density	2.20%	1.03%
Range of f. d.	0.63% - 11.66%	0.15% - 5.21%

*Executable assembly instructions

It is also of interest to examine the effect of language within a given time period. For this purpose the 21 programs that were started in 1977 and later are analyzed in Table 4. It is seen that HOL usage does indeed account for a very major reduction in fault density. The standard deviation of fault density among the HOL programs is 0.21. The difference between assembly and HOL programs is therefore in excess of four standard deviations ($p < 0.005$). The inequality presented in parentheses is an abbreviated notation to denote that the probability that such a large difference is due to chance events is less than 0.005. This notation is used repeatedly in this report to indicate the statistical confidence in selected findings.

The flight control programs have a higher than average fault density in each language classification, and particularly for the HOL group where their average is almost 2 standard deviations above that for the group as a whole. The two HOL flight control programs involved here are written in AED and were started in 1977 and 1978. Both the early starting date and the use of a language for which few program and test support facilities exist may account for the deviation.

TABLE 4 - EFFECT OF LANGUAGE ON RECENT PROGRAMS

Program Attribute	Assembly	HOL
No. of programs	6	15
Program size*	100k	1,124k
Fault density	1.03%	0.15%
Flight controls	1.58%	0.52%
Range of f. d.	0.15% - 5.21%	0.01% - 0.86%

* Equivalent executable assembly statements

As mentioned earlier, there is concern that the conclusions drawn from these comparisons might be biased due to the combination of data from several sources. The Goddard/SEL programs are all written in HOL and are in the recent group. These programs are also at the low end of the fault density range. Within the programs from this source there is no significant trend with respect to starting date or HOL use (some programs include assembly sections but these are typically less than 10% of the total code). To remove possible bias due to the combination of sources, the Ames (Set 1) data are analyzed separately in Table 5. Although this is a smaller sample, the same effects are apparent, and thus these do not appear to be artifacts introduced by combining data from several sources. (The figures in parentheses under each entry in the table represent the number of programs and the total size).

TABLE 5 - FAULT DENSITY FOR AMES (SET 1) PROGRAMS

Starting Period	Language:	FAULT DENSITY IN PERCENT Assembly	HOL	Both
Pre-1977		2.20 (6, 34k)	0.96 (1, 33k)	1.59 (7, 67k)
Recent		1.04 (6, 100k)	0.175 (2, 224k)	0.44 (8, 324k)
Both		1.33 (12, 134k)	0.276 (3, 257k)	0.638 (15, 391k)

It is seen that the reduction in fault density for recent starting dates and use of HOL carries over not only for this sample as a whole but also for each subgroup within it (HOL effects for each starting category, starting date effects for each HOL category), thus lending further credence to the conclusions stated earlier. From the analysis of the fault density data it appears that use of high order languages has a very pronounced beneficial effect on software reliability, and this finding can be translated directly into the selection of the programming environment for flight control software and for similar critical applications. The effect of starting dates can be interpreted as reflecting

disciplined software development (structured programming, hierarchical structure, small module sizes, etc.) and better test technology, but it may also be due to some other factors. Further studies of this effect are warranted. In the meantime, we can be satisfied that whatever we may have been doing during the past five years in our approach to software development has proved to be beneficial.

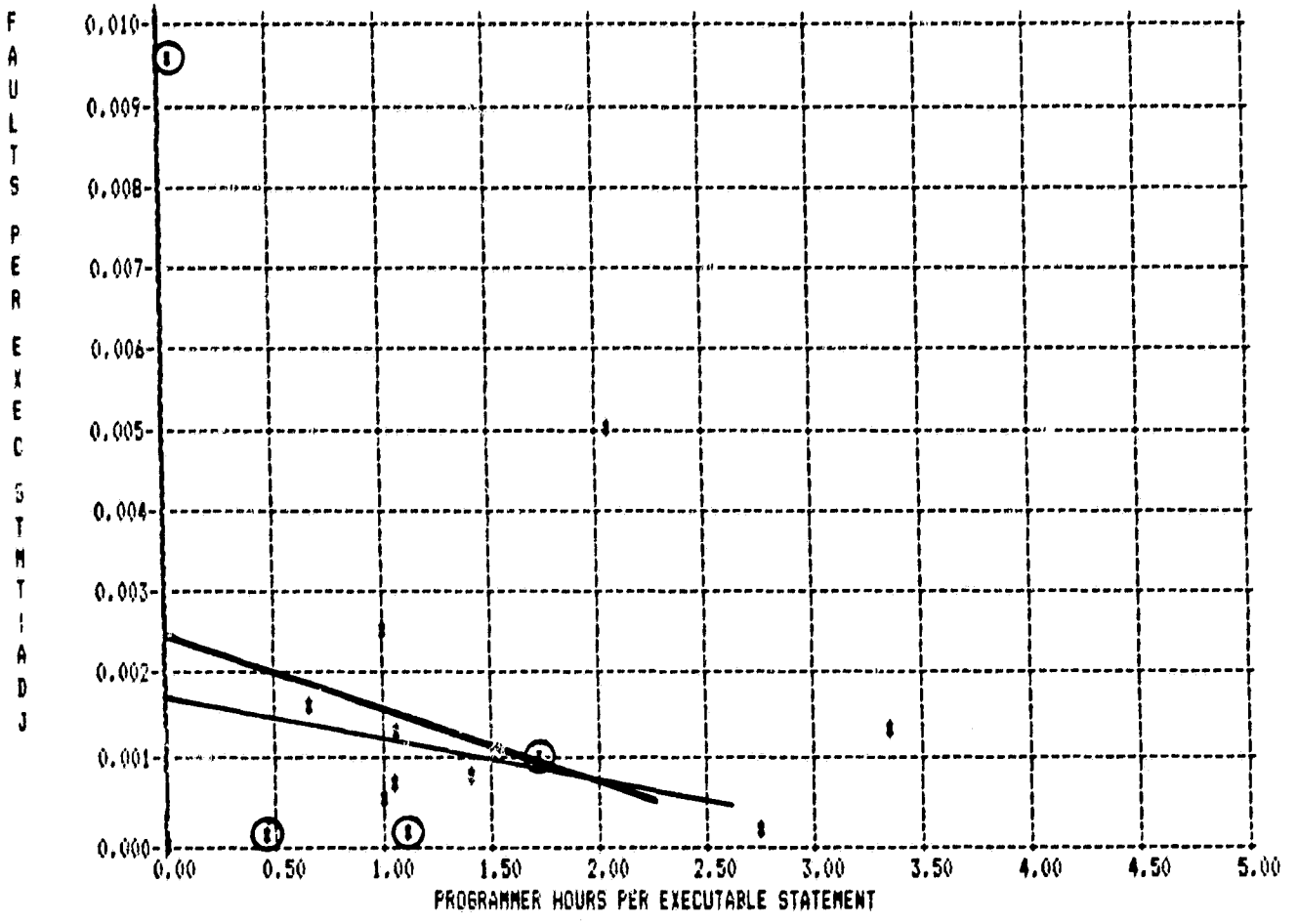
For a number of programs the data sources also provided information on the professional labor hours expended during development. The following analysis is restricted to 13 programs written predominantly in a HOL, and most of these come from the Goddard/SEL source. Four of these programs present pre-1977 starts, and the remaining nine were started more recently. A plot of fault density vs. programming hours per statement is shown in Figure 3. The heavy line represents the weighted (by program size) linear regression for the entire population. The lighter line is the linear regression for the recent programs only. Although the scatter is wide and other factors obviously have a strong effect on fault density, there is a pronounced negative trend (i. e., high programmer effort reduces the fault density). For both populations there is extremely high confidence that the slope is negative ($p < 0.001$). It would be interesting to test whether this relationship still holds for effort above some threshold (e. g., above 2 hours/statement) but the amount of data available was not sufficient for a meaningful analysis.

4. FAULT CLASSIFICATION

In spite of the encouraging results reported in the previous section, it must also be admitted that the overall results of the fault density analysis leave us far short of the reliability goals for essential flight controls. In order to apply remedial efforts effectively, the causes of faults in some of the software identified in Table 1 will now be examined.

A classification of causes of software faults was generated in a 1976 RADC sponsored study that examined several very large software programs [THAY76]. Because of the considerable effort that went into that study, and because fault distributions that were published in it presented a unique basis for comparison, this classification (or adaptations of it) have been carried over into most of the succeeding literature that examined software faults. Unfortunately, these classifications are not very satisfactory for describing software malfunctions in the context of FAA certification requirements where a classification by effect (error as defined above) would be more significant. Also, to judge the potential for catastrophic outcomes, it is desirable to know the purpose of the routine that the fault is located in (e.g., a fault in a scheduler has greater likelihood of serious consequences than a fault in an internal recordkeeping module). None of this information is available in either the data specifically examined here nor in any other data source that has come to the attention of the authors. It is also noted that the identification and separate analysis of flight control software data represents a unique aspect of the present report.

In spite of these shortcomings, the existing classification scheme provides insights into the deficiencies in the programming environment that lead to faults, and by inference therefore points to avenues for eliminating causes of



CIRCLED POINT: PRE-1977 STARTS ALL OTHERS: RECENT (1977 OR LATER) STARTS

——— LINEAR REGRESSION, RECENT STARTS ONLY

——— LINEAR REGRESSION, TOTAL POPULATION

FIGURE 3 - EFFECT OF PROGRAMMING EFFORT ON FAULT DENSITY FOR HOL PROGRAMS

faults. The data collected specifically for this study provided ten major categories and additional subcategories which were utilized only to a limited extent. The original categories are shown in the left side of Table 6. To permit comparison with other data collections, some categories were combined as shown in the right column of the table. All of the following discussion will be in terms of the revised classification.

TABLE 6 - FAULT CLASSIFICATION

Original Classification	Revised Classification
A. Computational errors	a. Computational errors (A)
B. Logic errors	b. Logic errors (B)
C. Data input errors D. Data handling errors E. Data output errors	c. Data I/O errors (C+D+E)
F. Interface errors	d. Interface & execution errors (F+J)
G. Data definition errors H. Data base errors	e. Data base/global variables (G+H)
J. Other errors	
K. Unknown	

Over 3600 faults from 22 programs were classified in this manner. Some, but not all, of these programs were those for which fault density distributions were discussed above. Three of the programs were related to digital flight controls. The results of this classification are shown in the first three numerical columns of Table 7. The remaining columns of this table contain equivalent data from other projects. The MIPS program comprises about 25k FORTRAN statements, dealing with launch data preparation and calibration for a missile tracking system. The program was developed between 1975 and 1977 but includes some portions from earlier programs. The TRW data come from the previously referenced RADC report. TRW 2 comprises 97k of JOVIAL J4 code dveloped in the early 1970s and is intended for batch processing. TRW 3 consists of over 115k JOVIAL J4 statements, likewise intended for batch processing, and developed slightly later than TRW 2. A significant fact about TRW 3 is that about one-quarter of the code was supplied by other contractors. TRW 4 is a general information processing system that can operate either on-line or in batch mode. It is written in a macro-languge, PWS, and the exact size is not stated. It is a large program, consisting of 190 different routines.

The flight control software data are presented in three columns in Table 7. The first of these is labeled 'raw' and contains data as reported. The second column (Adj. 1) contains an adjustment which transfers data scaling faults from the Data I/O category to Computational. In the third column a further adjustment is made to move faults related to flags from the Data I/O to the Logic classification. The reason for those adjustments is explained in the discussion of the table.

TABLE 7 - RESULTS OF FAULT CLASSIFICATION

Class	PERCENT OF ALL CLASSIFIED FAULTS							
	Ames (Set 1)				Langley/	TRW 2	TRW3	TRW4
	Total	Flight Controls			MIPS			
		Raw	Adj. 1	Adj.2				
a. Computation	7.4	10.0	17.3	17.3	4.2	17.2	11.5	2.4
b. Logic	47.1	26.7	26.7	35.6	66.3	27.2	30.5	47.5
c. Data I/O	15.8	32.3	25.0	16.0	10.7	16.6	23.4	12.2
d. Interf. etc.	16.2	14.5	14.5	14.5	15.6	20.1	20.2	25.8
e. Data base	13.5	16.5	16.5	16.5	3.3	18.8	14.4	12.2

Comparisons among the elements of a row show sizeable differences (e. g., the percentage of computational errors ranging from 2.4% to 17.3%) but the projects as a whole show surprisingly good agreement in the distribution of faults among the classifications. With the exception of the 'Raw' flight controls programs from the present sample, logic faults are the largest class. Data I/O errors are mostly in the second or third largest category.

In the 'raw' classification of faults for the Ames Flight Control programs, scaling errors were assigned to the Data I/O category. In most of the other data sources such errors are considered under computation ('Units Conversion Error' in [THAY76]). In the column headed 'Flight Controls, Adj. 1' these faults (incorrect scaling) were transferred from the Data I/O to the Computation classification. Even after this adjustment, Data I/O faults constitute an unusually high percentage, and logic faults an unusually low percentage, of all faults. A major contributor to this is a large fraction of flag setting faults (flags not reset, not properly set, etc.). There is precedent in [THAY76] for classifying these as Data I/O faults but it must be recognized that in terms of tools and methodology that can be utilized to uncover these they are closely related to logic faults. Usually incorrect handling of flags does not constitute a very large fraction of the faults, and this possibly inconsistent classification does not present a problem. When flag related faults are classified as Logic (in the Adj. 2 column), the pattern becomes much more consistent with that observed in the other data collections. Further investigation of the high percentage of flag related faults in this data set appears warranted. Interface and execution errors show considerable uniformity among all the data sources.

The total sample compiled as part of this study (first numerical column) shows a distribution among the fault classifications that is well within the range of those of the previously reported efforts (all of which had pre-1977 start dates, with most representing 1970 - 75 starts). There appears to be no basis for attributing changes in the distribution of faults among the categories to changes in the programming environment.

The total sample contained five programs that were written in a HOL, four that

were mixed (between 25% and 75% assembly), and 12 that were primarily written in assembly language. The fault classification for these three language groups is shown in Table 8. The percentage of computational faults is greater in the HOL programs which might be expected because these frequently deal with more complex algorithms. The percentage of logic faults is greater in mixed and assembly programs but the increase is not of a magnitude that would indicate a definite trend. The selection of language does not seem to have a major effect on the distribution of faults among the major classifications.

TABLE 8 - FAULT CLASSIFICATION BY LANGUAGE

Class	PERCENT OF ALL CLASSIFIED FAULTS HOL	Mixed	Assembly
a. Computation	11.6	6.9	5.9
b. Logic	43.7	47.9	47.4
c. Data I/O	17.3	17.0	15.0
d. Interf. etc.	15.9	14.4	17.4
e. Data base	11.4	13.9	14.2

The limitations of this classification must be acknowledged. It is widely recognized that mistakes made in the very early phases of the development process are the most costly ones to remedy and frequently have the most severe consequences [BOEH76]. Hence, identification of the development phase during which the fault originated should be an integral part of a fault classification scheme and use of such information in subsequent analysis should be extremely useful. Because of the judgement that is involved in this assessment most investigators have not pursued this subject. It was attempted to supply this information during the collection of some of the data analyzed here but this took place four to five years after the fault had been detected and did not produce consistent results. Data on the origin of faults (by development phase) are therefore not presented here although the importance of such data is fully acknowledged.

5. UTILIZATION OF FINDINGS

One of the problems of utilizing fault density as an index of software reliability is that it measures past events. One could even argue that a high number of faults detected during tests indicates that few remain as the program enters the operational phase. However, most practitioners know that there is a continuity to the fault detection process, and that a high rate of problems in one phase is usually associated with an above average rate in succeeding phases. An important improvement in the interpretation of the fault density data is to segregate and emphasize faults that occur in the final test phase because these constitute a better index of what might be expected in operation. Even with all these deficiencies, the data reviewed here make it unlikely that an average

flight control program is free of faults when it enters the operational phase. The data also show a wide variation in fault densities, with flight control programs being frequently in the upper part of that range (cf. Table 4). This indicates that much improvement is possible by intensive use of existing methodology.

The use of high order language seems to be an easily attainable means of improving the reliability of flight control programs or others that are involved in critical applications. Beyond this, the available information does not identify specific processes but it does indicate in a general way that the adoption of disciplined development methodology that took place in the 1975 - 1977 time frame reduces the fault content of programs (see Tables 3 and 5). That further improvements through intensive use of this methodology are possible seems to be borne out by the significantly lower fault density of the Goddard-SEL programs (see Table 1). That environment has encouraged full use of advances in software engineering, and several programs from this group were at the low limit of fault density encountered here (0.01%). Further studies to isolate the benefits of individual tools or methodologies are obviously indicated.

That software tools can play an important part in the reduction of program faults before they reach test is strongly indicated by the high percentage of logic, data base, and data I/O faults (Table 7). In the Ames (Set 1) sample, these three categories accounted for over 75% of all faults. Many of these can be avoided by the use of static analyzers, set/use listings, data dictionaries, and similar readily available tools. The high percentage of data I/O faults in flight control programs suggests emphasis on tools for mechanizing the review of data interchange with a program.

6. RECOMMENDATIONS

Collection of failure rate data, particularly tied to execution time, is essential in order to obtain a software reliability measure that can be directly compared with the FAA certification requirements. Many examples for the collection of such data are available [HECH77, MUSA79].

The data should be segregated by test phases or time in operational status so that improvements in reliability with time can be determined. To permit the identification of beneficial methodologies and tools, these data should be supplemented by a full description of the development, test, and operational environments.

The classification of faults and errors needs to be considerably modified in order to address flight control concerns. It must provide a basis for associating error types (by type of manifestation and severity) with fault types (logic, etc. as used above) so that emphasis can be placed on eliminating those faults that have the greatest potential for causing catastrophic failures. The recording of the function (e.g., computational routine, I/O routine, scheduler) that is affected by each failure is also of importance. The full benefits of failure reports are seldom achieved if a thorough analysis is not performed at

the time the failure occurs and the correction is made. At that time, the information is fresh and individuals involved in the detection and correction process are available. Much more effort is needed in both developing and implementing improved methods of failure reporting. A more meaningful identification of the development phase during which the fault arose may also be achieved by those efforts.

Attention has recently been called to the effect of workload (the fraction of computer processing capability utilized) on failure frequency of computers [CAST82, IYER82]. This suggests that (a) workload be considered as a failure inducing element in any reliability analysis to demonstrate compliance with the FAA requirements, and (b) that the prevailing workload be recorded in failure reports to permit further study of this phenomenon.

It is recognized that most flight control software failures will not have catastrophic effects on the safety of the aircraft. Yet, the relationship between frequency and severity of failures is largely unexplored. An investigation of failures in the Bell No. 4 Electronic Switching System (ESS) suggests that there is an inverse relation between frequency and severity (i. e., failures that caused the longest service interruption occurred least frequently) [DAVI81]. Whether this holds true for flight control software failures needs to be established. If such a relation exists in a general sense, then certification could be based on the frequency of occurrence of mild failures, a much more readily observable quantity.

To meet the needs of reliable software for critical applications in the immediate future, the application of fault tolerance techniques appears essential, not as an alternative to efforts at improving the quality of individual programs but rather as a supplement to such efforts. Several methods for implementing software fault tolerance have been described in the literature [RAND75, CHEN78, HECH79]. All require multiple, independent versions of a program but there are differences in the manner in which the independence is achieved and in the error detection mechanism. Even with allowance for less than perfect operation of the fault tolerance provisions, these approaches can reduce the software failure rate by several orders of magnitude. Thus, it may be possible to demonstrate compliance with the FAA requirements by showing that an individual program has a catastrophic failure rate of less than 1E-6, and that fault tolerance provisions reduce the system catastrophic failure rate to less than 1E-9 per flight hour.

REFERENCES

BASI77 V. R. Basili et al., "The Software Engineering Laboratory", University of Maryland TR-535, May 1977

BOEH76 B. W. Boehm, "Software Engineering", IEEE Trans. on Computers, vol C-25 no 12 pp 1226-1241, Dec 1976

CARD82 D. N. Card and F. E. McGarry, "The Software Engineering Laboratory", NASA Goddard Space Flight Center, SEL-81-104, February 1982

CAST82 X. Castillo and D. Siewiorek, "A Workload Dependent Software Reliability Prediction Model", Digest, FTCS-12, pp 279-286, June 1982

CHEN78 L. Chen and A. Avizienis, "N-Version Programming: A Fault Tolerance Approach to Reliability of Software Operation", Digest of Papers, FTCS-8, pp 3-9, June 78

DAVI81 E. A. Davis and P. K. Giloth, "No. 4 ESS: Performance Objectives and Service Experience", Bell System Technical Journal, vol 60 no 6 pp 1203-1224, Aug 1981

FAA70 Federal Aviation Administration, Amendment 25-23 to Federal Aviation Regulations, Part 25, 1970.

HECH79 H. Hecht, "Fault Tolerant Software", IEEE Trans. on Reliability, vol R-28 no 3 pp 227-232, Aug 1979

HECH77 H. Hecht, W. A. Sturm and S. Trattner, "Reliability Measurement During Software Development", NASA Langley CR-145205, Sep 1977

IYER82 R. K. Iyer and D. J. Rosetti, "A Statistical Load Dependency Model for CPU Errors at SLAC", Digest, FTCS-12, pp 363-372, June 1982

LAWS76 John D. Lawson, "Process Design Engineering Program, Baseline Software Process Productivity and Error Analysis", prepared by Texas Instruments Inc. for BMD Advanced Technology Center under Contract DAH60-72-C-0156, April 1976

MUSA79 J. D. Musa, "Validity of the Execution Time Theory of Software Reliability", IEEE Trans. on Reliability, vol R-28 no 3 pp 181-191, Aug 79

RAND75 B. Randell, "System Structure for Software Fault Tolerance", IEEE Proc. on Software Engineering, vol SE-1 no 2 pp 220-232, June 1975

ROCK81 Rockwell-Collins, "Software Error Study", Contract NAS2-27495, Collins Avionics Division, 1981.

THAY76 T. A. Thayer et al., "Software Reliability Study", Contract F30602-74-C-0036, TRW Defense and Space Systems Group, Mar 76

TURN81 C. Turner et al., "The NASA/SEL Data Compendium", DACS, April 1981

APPENDIX — DATA SUMMARY

A summary of the data utilized in the analysis of fault densities (Section 3) is shown in Table A. The 28 projects listed in the table were selected from about 50 projects which were originally included in the study. The deleted projects had incomplete data in one or more of the following categories which were essential for the principal analyses carried out in this report:

Year of start
Number of executable statements
Number of faults
Primary programming language

As is apparent in the table, the selected programs were frequently deficient in other data, and this caused some secondary analyses in the report to be based on a smaller population.

The following explanations apply to Table A:

PROJ NAME - Project designation, not necessarily descriptive, intended only to show the general origin of the software. The numbers appended to the NASA Goddard data correspond to the Project Number in the SEL data base.

ORG - Origin of the data: 1 = NASA Ames dataset 1, 2 = NASA Ames dataset 2, N = NASA Goddard Software Engineering Laboratory

START YR - Year of start of the prgram development

END YR - Year program development was completed

PGMR HRS - Programmer hours devoted to development, an entry of 0 indicates missing data

MGMT, OTH HRS - Management and other classification hours devoted to development, an entry of 0 indicates missing data

EXEC STMTS - Equivalent executable assembly statements as explained on p. 3

FAULTS - Number of faults reported for this project

PRIM LANG - Primary programming language. The following abbreviations are used: ASM - assembly language, COB - COBOL, FORT IV - FORTRAN IV, SFOR - S-FORTRAN

% - Percent of executable statements in primary language, an entry of 0.00 indicates missing data, an entry of 99.99 indicates that all statements were in the primary language

SEC LANG - Secondary Language. Abbreviations as under PRIM LANG; UNK indicates that it is not known whether a secondary language was used

$ - As under PRIM LANG

FAULT DENS - Fault density as explained on p. 3

TABLE A -- DATA SUMMARY

PROJ NAME	ORG	START YR	END YR	PGMR HRS	MGMT. OTH HRS	EXEC STMTS	FAULTS	PRIM LANG	%	SEC LANG	%	FAULT DENS.
PROJ X ADCU	1	1977	1981	57800	0	19482	309	ASM	99.99	NONE	0.00	0.01586
PROJ X - CLOBBER ANAL MODULE	1	1979	1981	0	0	66866	141	COB,SFOR	0.00	ASM	0.00	0.00211
PROJ X-MDPS(MISSION DATA PREP)	1	1979	1981	21250	0	157535	252	FORT VI	99.99	NONE	0.00	0.00160
PROJ Y - COMMAND & DISPLAY	1	1972	1975	0	0	6640	42	ASM	69.30	JOVIAL	30.70	0.00630
PROJ Y - NAVIGATION	1	1972	1975	0	0	32980	316	JOVIAL	99.99	NONE	0.00	0.00960
PROJ Y - EXECUTIVE	1	1972	1975	0	0	6466	151	ASM	99.99	NONE	0.00	0.02335
PROJ Z - CAP PHASE I	1	1974	1974	0	0	3400	165	ASM	79.35	FORTRAN	20.64	0.04850
PROJ Z - MSAP PHASE I	1	1974	1974	0	0	600	70	ASM	99.99	NONE	0.00	0.11666
PROJ Z - SAP PHASE I	1	1974	1974	0	0	4450	162	ASM	0.00	UNK	0.00	0.03640
PROJ Z - EXEC PHASE I	1	1974	1974	0	0	12572	163	ASM	99.99	NONE	0.00	0.01296
PROJ Z - CAP PHASE II	1	1978	1978	0	0	16048	258	ASM	85.89	FORTRAN	14.41	0.01610
PROJ Z - SAP PHASE II	1	1978	1978	0	0	4450	143	ASM	99.99	NONE	0.00	0.03213
PROJ Z - EXEC PHASE II	1	1978	1979	0	0	1228	64	ASM	99.99	NONE	0.00	0.05211
PROJ Z - SIMULATOR PHASE II	1	1978	1979	0	0	20618	192	ASM	92.09	FORTRAN	7.91	0.00930
PROJ Z - SUPPORT SW PHASE II	1	1978	1979	0	0	38218	72	ASM	94.96	FORTRAN	5.03	0.00190
PROJ E (FLT. CONTR.)	2	1978	1978	0	0	44400	381	AED	99.99	NONE	0.00	0.00860
PROJ D (FLT. CONTR.)	2	1978	1978	0	0	43500	78	AED	99.99	NONE	0.00	0.00180
AEM - MANPOWER ALLOC #2	N	1977	1978	89115	47855	26488	112	FORTRAN	84.00	ASM	16.00	0.00423
ISEEB-INTNATL SUN EARTH XPR #5	N	1977	1977	129299	37096	122718	88	FORTRAN	87.00	ASM	13.00	0.00070
PAS-PANORAM ATTITUDE SCAN #6	N	1977	1977	128522	72188	198965	21	FORTRAN	86.00	ASM	14.00	0.00010
SEASAT #10	N	1977	1978	109565	47820	109147	59	FORTRAN	76.00	ASM	24.00	0.00050
FOXPRO-SMM FOCUS PROC #35	N	1978	1979	19205	11278	16997	17	FORTRAN	72.00	ASM	28.00	0.00100
DEA-DYNAMICS XPLR A #36	N	1980	1980	149476	73735	140812	224	FORTRAN	87.00	ASM	13.00	0.00159
DEB-DYNAMICS XPLR B #37	N	1980	1980	134639	77997	128444	177	FORTRAN	85.00	ASM	15.00	0.00138
DESIM-DYNAMICS XPLR SIM #38	N	1980	1980	31638	24964	22410	17	FORTRAN	99.99	NONE	0.00	0.00080
DEDET-DEB DETERMINISTIC #40	N	1980	1980	34532	18750	13829	49	FORTRAN	79.00	ASM	21.00	0.00290
MAGNRT-MAGSAT NR REAL TIME #49	N	1980	1980	14023	3885	28900	2	FORTRAN	85.00	ASM	15.00	0.00010
MAGIRC-MAGSAT IR CALIBRA #54	N	1980	1980	5182	3408	2920	3	FORTRAN	99.99	NONE	0.00	0.00100