Using Group and Subsystem Level Analysis to Validate Software Metrics on Commercial Software Systems

Dennis Kafura
James Canning

TR 88-13

Using Group and Subsystem Level Analysis to Validate Software Metrics on Commercial Software Systems

Dr. Dennis Kafura
Department of Computer Science
Virginia Tech
Blacksburg, Virginia 24061

Dr. James Canning
Department of Computer Science
University of Lowell
Lowell, Massachusetts 01854

Abstract

This paper reports the results of a study which examined the relationship between a collection of software metrics and the development data (such as errors and coding time) of three commercially produced software systems. The software metrics include both measures of system interconnectivity and measures of system code. This study revealed strong relationships between the metrics and the development data when individual components were aggregated by structure (into subsystems) or by similarity (into groups). The subsystem and group results imply that research and application of metrics should be focused above the component level. The group results also imply that metrics can guide the effective application of project resources by identifying those groups which, for example, will contain a disproportionately large fraction of errors. Finally, the study showed the overall utility of two interconnectivity metrics: Henry and Kafura's information flow metric and McClure's invocation metric. This result is significant because interconnectivity metrics can be applied early in the life cycle.

I. Introduction

This paper reports the results of an effort to validate a variety of software metrics using three commercially produced systems. In total these systems contain over two hundred thousand lines of code and seven hundred executable components. The ultimate purpose of this effort was to determine whether there were significant relationships between software metrics and the historical development data which can be exploited to advantage in the software engineering process.

In this study we have expanded on previous work in two ways. First, we have employed more varied ways of viewing the relationships between the metrics and the development data including:

- linear correlations at a *subsystem level*
- studies of *groups* of components which are "similar" in both metric characteristics and development data characteristics

The use of these different views allowed us to examine more completely different aspects of the complete relationship between the metrics and the development data. Second, we incorporated a more varied collection of metrics. Previous research in this area has typically been limited to the application of code metrics such as: lines of code, McCabe's Cyclomatic number [McCa76], and Halstead's software science measures [Hals77]. In addition to these metrics, we also included four interconnectivity metrics which measure control and data flow interfaces among system components: Henry and Kafura's Information Flow measure [Henr81a], Woodfield's Syntactic Interconnection Model [Wood80], Yau and Collofello's Stability measure [Yau80] and McClure's Invocation complexity [McCl78].

This paper is practical and empirical. Kearney et.al. [Kear86] have termed this approach "exploratory." They note that "... with limited understanding of programs and programming, exploratory work is necessary to provide direction for further study" [pg. 1050]. The weakness of this approach is that "... lacking a theory of programming behavior ... these explorations are difficult to interpret and provide only weak support for the use of complexity measures" and that "... until more comprehensive evidence is available, software complexity measures should be used very cautiously" [pg. 1044]. Finally, they conclude that "complexity measures currently available provide only a crude index of software complexity" [pg. 1050]. Without lessening the need for a theory of metrics, we believe that empirical studies such as this one provide important guidance on the practical application of software metrics. Furthermore, practicioners are discovering that a collection of even simple metrics provides substantial benefits [Grad87].

The remainder of this paper is organized as follows. In the next section we explain in detail the motivations for examining measures at the subsystem level and the use of grouping techniques. This section also describes both the metrics that were used and the systems and historical development data. Section III, a baseline for the rest of paper, contains the component level linear correlation analysis. Sections IV and V present, respectively, the subsystem level analysis and the group analysis. The last section summarizes the results.

II. Metrics and Development Data

This section describes the basic components of our study. It first explains why we decided to use subsystem and grouping techniques to view the relationship between the metrics and the development data. We then briefly present the collection of software metrics which were used. Finally, the systems and the historical data are described.

Subsystem and Group Analysis

The first motivation for studying software at the subsystem level is that management plans and cost models are more frequently concerned with the cost and resources of subsystems than with individual components. This is illustrated in the environment from which the data used in this paper was obtained (NASA Goddard Space Flight Center). In this environment management reports the assignment of programmers to subsystems.

The second motivation is that development data is more easily collected and verified at the subsystem level. It is not always possible to identify a single component as the sole cause of an error or to correctly estimate the time spent working on a particular component. In this paper we will see that the subsystem data seems to contain less "noise" than the component level data.

The third motivation for using subsystems is the same as the first motivation for using groups. Measurement of subsystems or component groups does not require the metrics to differentiate between components having approximately the same metric or development data characteristic. In a subsystem or group analysis, small differences in component metric values and developmental data readings exert less effect than in a component level analysis.

The second motivation for using groups is that by studying groups it is easier to identify and understand those components which are "outliers" [Kafu85]. It may be unrealistic to assume that the development staff has the resources to give every component the same high degree of testing and review as that given to "critical" components. Since some differential effort is applied in any event, management can better assess how to distribute the resources for a project by identifying those components which have extremely high metric values.

The Metrics

Three general classes of software metrics can be distinguished: measures based on the system's design structure, termed "interconnectivity metrics", measures based only on implementation details, termed "code metrics" and measures which are a combination of the other two, termed "hybrid metrics." The use of both interconnectivity and code metrics is important for two reasons. First, these classes of metrics appear to be measuring different aspects of the system [Henr81b] [Kafu84]. Second, each class of metrics can be first used at different points in the software life cycle. While code metrics may be useful indicators during the testing and maintenance phases, they come too late in the software life cycle to address fundamental design decisions. Interconnectivity metrics, on the other hand, can be taken early in the life cycle since they are based only on system features which emerge at the high-level design phase. The hybrid metrics were created for this study to test the utility of weighting an interconnectivity metric by a code metric.

For reference, the metrics are briefly described in Table 1. There is no weighted invocation complexity measure because this did not appear to be meaningful given the way the invocation complexity measure is defined. Also several weighting factors were

tried (LOC, EFFORT, and CYCLO). Only the LOC weighted measures are presented because the choice of weighting factor made little difference in the results.

Table 1: Metric Summary

Abbreviation	Metric Name	What it measures
Code Metrics		
LOC	lines-of-code	component size
EFFORT	software science effort	mental difficulty
CYCLO	cyclomatic complexity	local control flow
Interconnectivity Metrics		
INFOFLOW	information flow complexity	global data flow
INVOKE	invocation complexity	global control flow
REVIEW	review complexity	combined global data/control flow
STABILITY	stability measure	resistance to "ripple effect" changes
Hybrid Metrics		
INFO-LOC	weighted information flow	combined size/global data flow
REV-LOC	weighted review complexity	combined size/global data/control flow
STAB-LOC	weighted stability measure	combined size/"ripple effect"

Of the four interconnectivity metrics the information flow metric has been previously applied and validated on realistic programs: the UNIX operating system [Henr79] [Henr81a] [Henr84], and a database management system [Kafu87]. Two of the interconnectivity metrics, Woodfield's review measure and McClure's invocation complexity were applied with success on smaller scale programs. Woodfield[Wood80] conducted controlled experiments which measured programs built by students competing in a programming contest and correlated these measures with coding time. McClure[McCl78] applied the invocation measure to a self-built COBOL program and provided a subjective evaluation. The stability measure of Yau and Collofello[Yau 80] was proposed but not validated.

Source Code and Historical Database

Critical resources necessary for this project were provided by the Software Engineering Laboratory (SEL) headed by Frank McGarry, Dr. Jerry Page and Dr. Victor Basili. This organization, formed in 1976, is composed of three members: Nasa/Goddard Space Flight Center(GSFC), The University of Maryland (Computer Science Department), and Computer Science Corporation (Flight Systems Operation). The SEL has defined and implemented an extensive monitoring and data collection process by which the details of all aspects of the software development process and product could be extracted for analysis [Nasa 81].

One critical resource provided by SEL management was the source code for three large scale projects, each of which was written in FORTRAN. These three software systems will be referred to as PROJECT A, PROJECT B, and PROJECT C. Each project is further subdivided by Nasa/Goddard into subsystems. The largest project, PROJECT A, has nine subsystems while PROJECT B and PROJECT C possess seven and six subsystems respectively. Throughout this paper individual subsystems will be identified by a number appended to its associated project name. Thus, subsystem three in PROJECT A will be referred to as subsystem A.3 . Each subsystem is a collection of components (i.e., FORTRAN subroutines, functions, and data blocks). Within a given project, subsystems generally do not share common components. That is, routines from one subsystem rarely call routines from another subsystem. Communication between subsystems is usually achieved through the use of global variables and project wide tables. However, all subsystems share a pool of utility routines written in FORTRAN and assembly language.

In addition to the three FORTRAN systems, SEL also provided a developmental database for each of the three projects. Selected for this study were the following variables:

Count Based Dependent Variables

Component Changes:	A modification to a component made either to correct an error, to improve system performance, to add capability, or to implement a requirements change.
Component Errors:	A discrepancy between a specification and its implementation. The specification might be a requirement specification, a design specification, or a coding specification.

Time Based Dependent Variables

Design Hours:	The time recorded by SEL personnel to create, to read and to review the design of an individual component or subsystem.
Code Total Hours:	The time recorded by SEL personnel to implement, to read and to review the coding of an individual component or subsystem.
Code Hours:	The time recorded by SEL personnel just to implement an individual component or subsystem.
Test Hours:	The time recorded by SEL personnel to test an individual component or subsystem, including unit testing, integration testing, and testing reviews.
Total Hours:	The time attributed to the entire development of a component or subsystem.

In addition to the two count based variables mentioned above, a third count based variable, WEIGHTED CHANGES, was derived. This is a measure of the total amount of effort spent either to fix an error or to make a change to a given component[Basi83]. The calculation of the weighed changes is given in Appendix A.

Filtering the Data

Because of the nature of the data collection process and the environment in which the systems were constructed, not all of the components from the three projects were used. One subset of components, referred to as DATASET A, was utilized in those experiments which compared the metrics with the count based dependent variables (ERRORS, CHANGES, WEIGHTED CHANGES). In particular, DATASET A does not include those components which were entirely or largely reused from prior projects. In all, 561 components were included in DATASET A. The second subset of components, known as DATASET B, was used in those experiments which related the metrics to time base dependent variables. DATASET B is a subset of DATASET A. The selection process, or "filter", used to produce DATASET B, was found to be necessary in order to adjust for problems with the data reporting mechanisms [Basi83]. In all, 331 components were used in DATASET B. A more detailed description of each of these two sets of components is presented in Appendix B.

III. Component Level Analysis

This section presents the Spearman coefficients obtained by correlating, on a component basis, the ten complexity metrics with the four development measures: ERRORS, CHANGES, WEIGHTED CHANGES and CODING TIME. The data in this section, showing weak and/or inconsistent correlations, will serve as a baseline against which the data in Sections IV and V can be evaluated.

The Spearman coefficients derived by measuring all components are found in the column labelled ALL PROJECTS in Table 2. The correlations between the metrics and the ERROR data are rather low indicating little or no relationship between them. Similar results were observed for both CHANGES and WEIGHTED CHANGES. These results were not totally unexpected. Previous research by Basili et. al. [Basi83], found similar results when comparing errors with software science metrics. Basili et.al. noted that the discrete nature of error reporting, with 52 percent of their components having zero errors, could account for the low Spearman correlations. The distribution of the error and change data in DATASET A is similarly skewed.

In an attempt to account for project related effects, the components of DATASET A were partitioned and analyzed by project. The Spearman correlations between the metrics and the error data for the three individual projects are presented in Table 2. Almost all the metrics show stronger associations with ERRORS for PROJECT A and PROJECT C than they did for the across project analysis. In particular the three code metrics: LOC (.627), CYCLO (.513), and EFFORT (.610) , exhibit noticeably higher correlations with ERRORS when applied to the components of PROJECT C. Of the three hybrid metrics, the INFO-LOC metric was the only measure to show significant improvements in the correlations. Again, these improvements are present for both PROJECT A and PROJECT C. The pure interconnectivity metrics have somewhat weaker associations to ERRORS than do either the code or hybrid metrics. Among the four interconnectivity metrics, Henry and Kafura's INFOFLOW and McClure's INVOKE measure exhibit the strongest relationship to ERRORS.

Table 2 : Component Analysis Within Projects -- ERRORS

	ALL PROJECTS	PROJ. A	PROJ. B	PROJ. C
LOC	.470	.506	.257	.627
CYCLO	.373	.390	?	.513
EFFORT	.467	.494	.189	.610
INFOFLOW	.321	.462	?	.489
INVOKE	.398	.331	.406	.479
REVIEW	?	.250	.209	-.249
STABILITY	.174	.228	?	.287
INFO-LOC	.364	.496	?	.536
REVIEW-LOC	.254	.509	.281	?
STAB-LOC	.219	.175	.267	.257

KEY: ? $P > .05$, OTHERWISE $P < .01$

The correlations found in Table 2 generally indicated an inconsistent relationship between the metrics and ERRORS. No single metric nor any class of metrics demonstrated a strong overall correlation. LOC and EFFORT, for example, both have correlations over 0.6 on Project C while both have correlations less than 0.25 on Project B. Similar observations can be made when comparing the metrics to the other two count based metrics (CHANGES and WEIGHTED CHANGES).

Table 3 presents the Spearman rank coefficients between the complexity metrics and CODING TIME for components in DATASET B. The other time based data was not used since this data was not reported on a component basis. From the data presented in this table, code metrics relate more strongly to CODING TIME than either interconnectivity or hybrid metrics. Among the interconnectivity and hybrid metrics, the INFOFLOW metric (.437) and the INFO-LOC metric (.479) indicate the strongest association to CODING TIME.

Also presented in Table 3 are the results of correlations which partitioned the components of DATASET B by project. As can be seen, the coefficients for PROJECT A indicate stronger association to CODING TIME for every metric. Furthermore, for some metrics, the correlations with CODING TIME also increases for PROJECT C components. As before, in the experiments with the count based data, improvements in the correlations are not as consistent for Project B.

Table 3: Spearman Correlations: Metrics With CODING TIME

	ALL PROJECTS	PROJ. A	PROJ. B	PROJ. C
LOC	.544	.634	.475	.576
CYCLO	.583	.674	.442	.581
EFFORT	.557	.583	.523	.369
INFOFLOW	.437	.599	.421	.468
INVOKE	.286	.293	?	.350
REVIEW	-.180	?	?	-.327
STABILITY	.303	.468	.392	.306
INFO-LOC	.479	.616	.465	.517
REVIEW-LOC	.164	.583	.424	?
STAB-LOC	.271	.409	.320	.266

KEY: ? $P > .05$, otherwise $P < .05$

IV. Subsystem Level Analysis

The results of the previous section correlated the software metrics with the developmental data on a component basis. In this section the unit of observation is not a single component but an entire subsystem.

Both software complexity measures and developmental data values were defined for each of the twenty-one subsystems. Since a given component did not belong to more than a single subsystem, software complexity measures for a subsystem were derived by summing the complexities of all of its components. Similarly, the count based measures for a subsystem (ERRORS, WEIGHTED CHANGES) were established by totaling the count based measures for its individual components. Time based measures included not only the times of individual components but also added any overhead time attributed to the subsystem. This overhead time is due to the inability of programmers to attribute their design and testing effort to individual components.

The results of the subsystem level analysis given below describes two similar but distinct experiments. The first experiment correlated the subsystem complexities with the subsystem development data using all twenty-one subsystems. The second experiment used only nine of the subsystems. Twelve subsystems were eliminated since they contained reused components which were constructed and tested during the development of previous projects.

The Spearman coefficients given in Table 4 were obtained by correlating the five development times with the ten metrics for all twenty-one subsystems. One observation that is immediately apparent is that these Spearman rank coefficients indicate a much stronger relationship between the metrics and the time based data than did the corresponding analysis at the component level. The Spearman rank coefficients indicate that the three code metrics (LENGTH, EFFORT, and CYCLO) each induce an ordering on the subsystems that is highly similar to the ordering induced on them by any of the development times. Furthermore, with Spearman correlations ranging between 0.63 and 0.81, McClure's invocation complexity, INVOKE, is the metric most closely related to

the five development times. Of the remaining interconnectivity metrics, the stability metric of Yau and Collofello also indicates a strong rank association with TOTAL HOURS (0.62) and DESIGN HOURS (0.72). The ability of the information flow metrics and the review metrics to rank order the subsystems according to the various development times is generally weaker than the other metrics.

Table 4 : Complexity Metrics Correlated With Time Based Data (N=21)

	TOTAL HOURS	DESIGN HOURS	CODE TOTAL HOURS	CODE HOURS	TEST HOURS
LOC	.697	.740	.607	.645	.603
EFFORT	.690	.710	.614	.633	.607
CYCLO	.707	.661	.636	.644	.684
INFOFLOW	.484	.580	?	?	.511
INVOKE	.790	.811	.750	.692	.632
REVIEW	.488	.587	?	?	?
STABILITY	.620	.720	.540	.510	.444
INFO-LOC	?	?	?	?	?
REVIEW-LOC	?	.528	?	?	.444
STAB-LOC	.624	.696	.542	.545	.468

KEY: ? $p > 0.05$, otherwise $p < 0.05$

The Spearman rank correlation coefficients in Table 5 indicate relationships between the metrics and the count based data. Once again, it is apparent that the metrics display a much stronger relationship with ERRORS and CHANGES at the subsystem level than they do at the component level. These results indicate that code metrics possess a strong rank association with subsystem-wide errors and subsystem-wide changes. Collectively, the interconnectivity metrics have a somewhat weaker relationship with ERRORS and CHANGES than do the code metrics. Of these, McClure's INVOKE measure relates the most strongly with ERRORS (0.70) and CHANGES (0.81). Furthermore, the STAB-LOC metric also indicates a rather strong relationship with ERRORS (0.67) and CHANGES (0.70).

Table 5: Complexity Metrics Correlated With Count Based Data (N=21)

	ERRORS	CHANGES
LOC	.823	.800
CYCLO	.669	.740
EFFORT	.754	.781
INFOFLOW	.635	?
INVOKE	.700	.815
REVIEW	?	.515
STABILITY	.553	.650
INFO-LOC	.613	?
REVIEW-LOC	.568	.562
STAB-LOC	.679	.707

KEY: ? $p > .05$, otherwise $p < .05$

Many of the twenty-one subsystems used in the above analysis contained reused components. The development data for a component which is reused without change will show zero errors and zero times regardless of its complexity measures and its initial development experience. In an effort to control for this bias, the above experiments were redone using only the nine subsystems which contained no more than ten percent reused code. The Spearman correlation coefficients between the metrics and the time base data are presented in Table 8. Table 7 contains the Spearman rank coefficients generated by correlating the metrics with the count based variables.

These tables indicate a much stronger association between the metrics and the data than the previous experiment which included all twenty-one subsystems. These increased correlations are attributed to improvement in the quality of the data.

Table 6: Complexity Metrics Correlated With Time Based Data (N=9)

	TOTAL HOURS	DESIGN HOURS	CODE TOTAL HOURS	CODE HOURS	TEST HOURS
LOC	.816	.850	.683	.733	.733
CYCLO	.850	.484	.783	.833	.766
EFFORT	.816	.850	.683	.733	.733
INFOFLOW	.816	.850	.700	?	.816
INVOKE	.733	.766	.667	.716	?
REVIEW	.816	.850	.683	.733	.733
STABILITY	.800	.900	?	.712	.700
INFO-LOC	.716	.716	?	?	.766
REVIEW-LOC	.800	.833	.667	.750	.716
STAB-LOC	.667	.750	?	?	?

KEY: ? $p > .05$, otherwise $p < .05$

Table 7: Complexity Metrics Correlated With Count Based Data (N=9)

	ERRORS	CHANGES
LOC	.928	.966
EFFORT	.928	.966
CYCLO	.861	.933
INFOFLOW	.778	.783
INVOKE	.711	.800
REVIEW	.928	.966
STABILITY	.928	.966
INFO-LOC	.769	.733
REVIEW-LOC	.895	.933
STAB-LOC	.878	.883

KEY: $p > 0.05$, otherwise $p < 0.05$.

V. Group Level Analysis

This section presents the results of experiments which analyzed groups of "similar" components. Because of the variation in the grouping techniques, the complexity metrics and the types of developmental data, a total of ninety cases were generated and analyzed. This large amount of information has been summarized in the following tables which are representative of the total ninety cases.

The complete group analysis utilized three grouping techniques. Three techniques were used to investigate the effect of different grouping methods and to gain confidence that the overall results were not sensitive to a particular grouping technique. One grouping method, referred to as the Logged Data technique, identifies six distinct groups. Two of the groups contain those components with values lying within one standard deviation from the mean. A second pair of groups contain components with values lying within two standard deviations of the mean. Finally, those components lying more than two standard deviations from the mean comprise the last two groups. However, since most of the original data is asymmetric with positive skew the usefulness of univariate summary statistics, such as the mean and the standard deviation is minimal. Meaningful summary statistics can be obtained, however, for asymmetric data, by first (a) transforming the data to achieve a symmetric distribution, (b) calculating the mean m, and standard deviation s of the transformed data, and then (c) applying the inverse transform to the values: m-2s, m-s, m, m+s, m+2s to determine the group boundaries of the original data. A logarithmic transformation to obtain a non-skewed distribution has been used by Crawford et. al. [Craw85] to analyze static metrics. In addition to the Logged Data technique, a clustering method and a simple rank grouping were also used. These other two techniques are described in Appendix C. Since there were no significant differences among the three techniques, only the results using the Logged Data technique will be given here.

Our first step, shown in Table 8, examined the relationship between metric groups and development data group. The second step, shown in the subsequent tables, examined

various statistics within the metric groups. Table 8 show how the 561 components in DATASET A are distributed across each metric roup and each error group for three different metrics. The three metrics (LOC, INVOKE, and INFO-LOC) represent the three classes of metrics (code, interconnectivity, and hybrid). Consider the 3 values in metric group 1 and error group 1. This group contains those components which have no errors and the least complexity as judged by the metric. For the particular group, there are 9 components if the LOC metric is used, 54 if the INVOKE metric is used, and 13 if the INFO-LOC metric is used. The difference in these counts reflect the fact that the metrics are classifying the components according to different criteria. The bottom and right margins of the table show summary data for each column (metric group) and each row (error group).

Notice that the distribution of components across metric groups appears to be normally distributed while the distribution across error groups remains skewed. The components in the metric groups for the remaining seven metrics were similarly distributed.

Table 8. Crosstabulation of Metric Groups with Error Groups

LOC
INVOKE
INFO-LOC

ERROR GROUPS	METRIC GROUPS 1	2	3	4	5	6	
1	9	42	120	45	18	0	234
	54	40	65	65	10	0	234
	13	44	95	70	12	0	234
2	1	3	45	55	11	1	116
	16	12	43	38	7	0	116
	0	14	36	49	17	0	116
3	0	3	21	34	17	1	76
	7	5	15	37	12	0	76
	2	4	16	35	18	1	76
4	0	6	15	32	20	3	76
	11	3	10	38	13	1	76
	1	7	14	26	27	1	76
5	0	1	9	23	13	2	47
	6	1	4	21	14	1	47
	0	8	9	16	12	2	47
6	0	0	1	7	4	0	12
	0	0	1	1	8	2	12
	0	0	2	3	5	2	12
	10	55	211	196	82	7	561
	94	61	138	200	64	4	561
	16	77	172	199	91	6	561

This non-normal distribution of the error groups caused us to partition the 561 components into component without errors (234) and components with errors (327). Table 9 indicates how the components were distributed across metric groups with respect to this partitioning.

The percentage of components within each group containing no errors is shown for all ten metrics in Table 9. Notice that no errors were reported for 81% of the components found in INFO-LOC Group 1. However, only 57% of the components in INFO-LOC Group 2 were found to be errorless, 55% percent for INFO-LOC Group 3 and so on. Interestingly, all components belonging to the highest INFO-LOC group did contain at least one error. This trend is also everywhere consistent for McCabe's CYCLO metric ranging from 76%(Group 1) down to 0%(Group 6). The third code metric, Halstead's EFFORT generally exhibits this same property, but does contain an anomalous jump between the Group 5 and Group 6 boundaries. Furthermore, of the four interconnectivity metrics, only Woodfield's Review complexity fails to show any pattern. The strongest relationship between the various interconnectivity metric groups and fraction of errorless components exist with the INFOFLOW groups. These percentages steadily decrease from 73% (Group 1) to 53% (Group 3) to 0% (Group 6). Of the three hybrid metrics, only the INFO-LOC metric defines groups with consistently decreasing percentages. The REVIEW-LOC metric exhibits this desirable trend for Group 2 through Group 6, however it does report only 20% of the components to be errorless for its least complex group of components.

Table 9: Percentage of Errorless Components Across Groups

	GROUP NUMBER					
	1	2	3	4	5	6
METRIC						
LOC	90%	76%	56%	22%	21%	0%
CYCLO	76%	63%	53%	26%	25%	0%
EFFORT	80%	64%	54%	29%	18%	33%
INFOFLOW	73%	51%	53%	36%	17%	0%
INVOKE	57%	65%	47%	32%	15%	0%
REVIEW	14%	—	53%	36%	17%	—
STABILITY	60%	51%	40%	38%	31%	—
INFO-LOC	81%	57%	55%	35%	13%	0%
REVIEW-LOC	20%	80%	61%	22%	22%	11%
STAB-LOC	56%	49%	40%	35%	26%	80%

Next, the information contained in the two-way crosstabulations between the metric based groups and the WEIGHTED CHANGES based groups has been summarized in Table 10. This table captures the ability of the metrics to identify components which will be highly affected by changes. To obtain the percentages given in this table, components with the highest WEIGHTED CHANGE values were defined as "severely impacted." In particular, a component possessed a "high" WEIGHTED CHANGE value if the grouping method classified the component into either Group 4, Group 5, or Group 6. The entries in Table 10 indicate the percentage of components within each metric group which were severely impacted. For example,72% of the components in LOC group 5 were also identified to be severely impacted components, while 100% of the components in INVOKE group 6 were severely impacted components. The percentages given in the last two columns of Table 10 generally indicate that components placed into the higher metric groups are likely to be severely impacted by system changes. LOC and CYCLO perform

the best among the code metrics, INVOKE and INFOFLOW among the interconnectivity metrics, and REVIEW-LOC among the hybrid metrics.

Table 10: Fraction of Severely Impacted Components

	GROUP NUMBER					
METRIC	1	2	3	4	5	6
LOC	0%	35%	43%	54%	72%	100%
CYCLO	15%	25%	43%	63%	66%	100%
EFFORT	5%	39%	48%	50%	81%	62%
INFOFLOW	58%	33%	41%	60%	77%	100%
INVOKE	36%	31%	40%	60%	82%	100%
REVIEW	70%	—	44%	53%	56%	—
STABILITY	40%	35%	47%	55%	63%	—
INFO-LOC	55%	34%	42%	45%	85%	100%
REVIEW-LOC	0%	29%	44%	51%	70%	100%
STAB-LOC	29%	50%	49%	58%	65%	20%

Finally, Table 11 contains the mean number of ERRORS for groups. This table generally indicates that higher complexity groups have a higher mean number of ERRORS. Note, for example, the steady increase in the mean number of ERRORS across the six LOC groups. The other two code metrics, CYCLO and EFFORT, also show a similar tendency. Of the four interconnectivity metrics, only Woodfield's REVIEW metric fails to exhibit this behavior. It is striking to note the dramatic jump in the mean number of errors between Group 5 and Group 6 for both the INFOFLOW metric (from 3.58 to 23.5) and McClure's INVOKE complexity measure (from 5.85 to 20.75). Of the three hybrid metrics, the INFO-LOC measure and the STAB-LOC measure both impose groupings on the components which relate to the average number of errors reported. The trend is consistent for the first five groups established by the STAB-LOC metric, while the sixth group exhibits a decrease in the mean number of errors. The sharp increase between group 5 and group 6 is also present for the INFO-LOC groups, jumping from 3.91 mean errors to 17.0 mean errors. The REVIEW-LOC metric fails to show any strong relationship between the mean number of errors and the reported complexity. Results for the Clustering and Ranking methods and also for Coding Time and Weighted Changes are also similar and are also not presented here but can be found in [Cann85]

Table 11: Mean Errors Per Group for Logged Data Technique

	GROUP NUMBER					
	1	2	3	4	5	6
METRIC						
LOC	0.10	0.65	1.12	2.93	4.10	4.28
CYCLO	0.23	1.21	1.09	3.29	3.46	3.12
EFFORT	0.67	0.78	1.57	2.95	2.80	2.55
INFOFLOW	1.07	1.62	1.49	1.98	3.58	23.50
INVOKE	1.25	0.67	1.17	2.19	5.85	20.75
REVIEW	5.05	—	1.37	2.24	2.80	—
STABILITY	1.25	1.51	1.90	2.62	2.50	—
INFO-LOC	0.43	1.50	1.34	2.03	3.91	17.00
REVIEW-LOC	5.34	0.48	0.98	2.28	4.08	3.55
STAB-LOC	1.08	1.53	1.57	2.48	4.91	0.20

In order to statistically support the hypothesis that the ten software metrics impose a grouping on the components which yield significant differences in the three developmental data means (ERRORS and WEIGHTED CHANGES), thirty analyses of variance were performed. Twenty-five of the thirty analyses of variance rejected the null hypothesis that the group means were all equal (alpha=.05).

VII. Summary and Conclusions

In this paper we employed both subsystem and group analysis to study the relationships between ten software metrics and the development data from three commercially developed software systems. From this study we conclude:

- *The software metrics were validated.* Strong relationships between the metrics and the development data were observed at the subsystem level (Tables 4, 5, 6, and 7). Strong relationships were also visible when "similar" components were collected into groups (Tables 9, 10, and 11). This validation should be particularly compelling to practicioners because the systems being studied were commercial products developed by professional programmers for use in a production environment.

- *Interconnectivity metrics should be used.* Each of the four interconnectivity metrics performed very well in at least one of the validation experiments. The information flow metric correlated strongly with most of the subsystem time measures (Table 6). The invocation complexity metric correlated strongly with both time and count measures (Table 4 and 5). Both the review complexity and the stability metrics correlated strongly with errors and changes at the subsystem level (Table 7). Overall the information flow and the invocation complexity metrics were the two best interconnectivity metrics. These two metrics form an interesting pair because one measures only the data flow among system components while the other focuses on the global control flow. While interconnectivity metrics do not provide stronger correlations than do the code metrics, it should be recalled that the interconnectivity metrics can be taken earlier in the life cycle and, thus, they provide more leverage than do code metrics on software quality problems.

- *Software metrics should be applied above the component level.* For all the metrics examined there were markedly stronger relationships at the subsystem level (Tables 4-7) than at the component level (Tables 2 and 3). The stronger subsystem results may be due to a variety of causes including: (1) there is less "noise" in the aggregated development data, or (2) the metrics are not sufficiently accurate to discriminate between small grain units (components) but they are capable of discriminating properly between larger scale units (subsystems). While the cause is not known, the implication for practicioners is clear. Since other resource control and planning techniques are also aimed at the subsystem level, the fact that software complexity metrics work at this level is also beneficial.

- *Differential resource allocation can be guided by software metrics.* The efficient application of human and computer resources during the development process requires the identification of those parts of the system which require more resources to achieve desired quality objectives. The group analysis aids in this identification as illustrated by: group 6 for INFOFLOW, INVOKE, and INFO-LOC in Table 11; group 6 in Table 9 for five of the ten metrics; group 6 (and usually group 5) for six of the ten metrics in Table 10.

- *Hybrid metrics are not useful.* As illustrated in Tables 2-7, combining metrics together does not lead to substantial improvements over the individual metrics. This negative result implies that an all encompassing single metric should not be sought. Rather research should focus on defining a validated collection of metrics each of which measures a different aspect of the system.

- *Development data must account for software reuse.* The characteristics of reused components are significantly different from newly developed ones (Appendix B). The differences alter the relationship between the metrics (which are unaffected by reuse) and the development data (which are affected by reuse). Compare, for example, the results in Tables 4 and 5 versus those in Tables 6 and 7.

As in all case studies which are based on a single environment, the usual cautions apply to any attempt to generalize these conclusions to other environments. These cautions - as well as the others mentioned in the introduction - should, rather than being discouraging, serve to encourage additional research and experimentation.

References

[Basi83] Basili,V.,Selby R.,Phillips T., "Metric Analysis and Data Validation Across Fortran Projects," *IEEE Transactions on Software Engineering*, Vol. SE-9,No. 6. November 1983.

[Cann85] Canning, J. Applying Structure and Code Metrics to Large Scale Systems, Ph.D. Dissertation, Department of Computer Science, Virginia Tech, June 1985.

[Craw85] Crawford, S. McIntosh, A., and Pregibon, D., "An Analysis of Static Metrics and Faults in C Software," The Journal of Systems and Software, Vol. 5, No. 1, February 1985.

[Grad87] Grady, R.B., Caswell, D.L. Software Metrics: Establishing a Company-Wide Program. Prentice-Hall, Inc., Englewood, NJ, 1987.

[Hals77] Halstead, M.H. Elements of Software Science , Elsevier North-Holland, Inc. , New York , NY , 1977.

[Henr79] Henry, Sallie, Information Flow Metrics for the Evaluation of Operating Systems' Structure, Ph.d Dissertation, Department of Computer Science, Iowa State University, 1979.

[Henr81a] Henry, Sallie and D. Kafura, "Software Structure Metrics Based on Information Flow", *IEEE Transactions on Software Engineering*, Vol. SE-7, No. 5, pp. 510-518, September, 1981.

[Henr81b] Henry, Sallie, D. Kafura, and K. Harris, "On the Relationships Among Three Software Metrics", *Performance Evaluation Review*, Vol. 10, No. 1, pp. 81-88 Spring 1981.

[Henr84] Henry, Sallie and D. Kafura, "The Evaluation of Software Systems' Structure Using Quantitative Software Metrics", *Software: Practice and Experience* Vol. 14(6) June 1984 pp.561-573.

[Kafu84] Kafura D., Canning J., and Reddy G., "The Independence of Software Metrics Taken at Different Life-Cyle Stages" *Proceedings: Ninth Annual Software Engineering Workshop*, Goddard Space Flight Center, Nov. 28, 1984

[Kafu85] Kafura D.,Canning J., "A Validation of Software Metrics Using Many Metrics and Many Resources" *Proceedings of the International Conference of Software Engineering*, London England, August 1985, pp. 378-385.

[Kafu87] Kafura D., Reddy, G.R., "The Use of Software Complexity Metrics in Software Maintenance," *IEEE Transactions on Software Engineering*, SE-13, 3 (March 1987), 335-343.

[Kear86] Kearney, J.K., Sedlmeyer, R.L., Thompson, W.B., Gray, M.A., Adler, M.A. Software Complexity Measurement. *Communications of the ACM*, 29, 11 (November 1986), 1044-1050.

[McCa76] McCabe,T.J., "A Complexity Measure," *IEEE Transactions on Software Engineering*, SE-2 , 4 (December 1976) , 308-320.

[McCl78] McClure, C. "A Model for Program Complexity Analysis," in *Proceedings Third International Conference on Software Engineering* , Atlanta, Ga. May 1978 , 149-157.

[Nasa81] National Aeronautics and Space Administration, "Software Engineering Laboratory (SEL) Data Base Organization and User's Guide", Software Engineering Laboratory Series SEL-81-002, Sept. 1981

[Wood80] Woodfield,S.N., Enhanced Effort Estimation by Extending Basic Programming Models to Include Modularity Factors, Ph. D. Thesis, Purdue University, Computer Science Dept., 1980.

[Yau80] Yau S., Collofello J., "Some Stability Measures for Software Maintenance," *IEEE Transactions on Software Engineering*, Vol. SE-6,No.6,Nov.1980.

Appendix A: Calculation of Weighted Changes

In the Nasa/Goddard environment, programmers classify each change by estimating the amount of effort needed to isolate the change/error and to implement the change/error. The four possible classifications recorded in the developmental database are:

- Less than one hour.
- One hour to one day.
- One day to three days.
- Over three days.

Corresponding to each of these classifications are the four weights suggested by Basili: 0.5, 4.5, 16.0, or 32.0 hours. For each change, the appropriate weight is divided equally among all components involved in the change. Thus, a component's WEIGHTED CHANGE value is derived by summing the hours attributed to the component for every change with which it was involved.

Appendix B: Filtering of the Data Sets

The SEL database classifies each component as either: new code, extremely modified old code, slightly modified old code, or an exact copy of old code. DATASET A contains code which is either new code or extremely modified old code. It is necessary to omit slightly modified or exact copy code from consideration since error counts and development times for reused components do not accumulate from project to project in the SEL reporting mechanism. Similarly, the time based measurements would also be biased since they would not reflect the original effort expended in the development of slightly modified or exact copy code. It was decided that extremely modified components should be incorporated into the analyses since these components undergo a significant transformation. It was felt that count based measures and time based measures would approximate the values of such a component had it been completely constructed anew. In support of these decisions, Table B1 below provides the mean values for the four measures across the four component classifications. An analysis of variance was performed using dependent variables (ERRORS, CHANGES, WEIGHTED CHANGES, TOTAL TIME) to test the null hypothesis that mean values for the four component classes were not statistically different. In each of the four analyses of variance the null hypothesis was rejected (alpha=0.05). Furthermore, Fisher's Protected Least Significance Test (LSD) was used to identify those means which were significantly different. In all four tests, it was found that means from component class (1) and component class (2) were not significantly different, but the means for these two classes were significantly different from both class (3) and class (4).

Table B1 : Data Means for Various Component Types

Component Class	Errors	Changes	Weighted Total Changes	Time
New Code	2.03	2.11	44.20	16.09
Extremely Modified	1.47	2.62	43.57	13.32
Slightly Modified	0.43	1.11	18.98	6.06
Duplicated	0.17	0.52	3.24	3.37

Another subset of the Nasa/Goddard components (DATASET B) was used for experiments involving time based dependent variables. This collection of components not only omits slightly modified routines and exact copy code, but also eliminates components which fail to pass two additional tests. Both of these tests provide a validity check on the reporting of time based dependent variables.

The first test eliminates those components which were constructed by programmers who were identified as poor or inconsistant reporters of time based dependent variables. Programmers were considered poor reporters if the ratio, Vm, found below, was less than eighty percent. This ratio, first suggested by Basili et al. [Bas83], utilizes the partial redundancy built into the SEL monitoring process. Each week every programmer files a Component Status Report (CSR) describing the time they spent on each module. Also on a weekly basis, the project manager files a Resource Summary Form (RSF) recording the time each programmer spent on the project during that week. Basili et al. have indicated that the manager reported information found in the Resource Summary Form is considered to be the more accurate. Thus, if a given programmer fails to submit a Component Status Report for a given week, this would lower his Vm ratio. The Vm ratio is defined as:

$$Vm = \frac{\text{Number of weekly CSR's submitted by programmer}}{\text{Number of weeks programmer appears on RSF's}}$$

The second criterion is based on the fact that time based dependent variables are not only reported for individual components, but are also reported for individual subsystems. Programmers typically report their time spent on an individual component basis. However, a programmer may choose to attribute work hours to an entire subsystem if these hours cannot be accurately partitioned among that subsystem's individual components. When the time reporting is done on a subsystem level, information at the component level is lost. If less than eighty percent of a subsystem's reported time is attributed to individual components, then these components are not incorporated into DATASET B.

Appendix C: Two Other Grouping Techniques

The second grouping technique, referred to as Clustering, is based on a hierarchical clustering scheme. Basically, the algorithm begins by forming one cluster for each observation in the data. Components are then grouped according to a distance function, clustering together those components which are "close neighbors." Clusters are determined by considering the distances between components within a given cluster versus the distances between the clusters. Ideally, the algorithm will choose the optimal number of clusters, but we arbitrarily fixed the number of clusters to be six for comparison purposes.

The third grouping technique, referred to as Rank Grouping, defines the groups according to the ranking of the components imposed by a given metric. The set of components are first ranked in ascending order by a given metric and then placed into groups according to rank. Components whose rank was less than one-sixth the total number of components were assigned to group 1, components whose rank was greater than one-sixth the total number of components but less than two-sixths the total number of components were assigned to group 2 and so on. Whenever two or more components

had the same complexity measurement their ranks were redefined as the mean of their corresponding ranks. Since it is possible that numerous components may have the same complexity value, and in some instances this is likely (ie. REVIEW Metric) not all the groups will necessarily contain the same number of members. In the worst case, a rank order group may contain no components when this simple rule is used.