APPLYING STRUCTURE AND CODE METRICS TO THREE LARGE-SCALE SYSTEMS

by

Dr. Dennis Kafura
Dr. James Canning

TR-85-31

August 1985

Applying Structure and Code Metrics to Three Large-Scale Systems

Dr. Dennis Kafura
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24060

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

August 6, 1985

ABSTRACT

This work extends the area of research termed software metrics by applying measures of system structure and measures of system code to three realistic software products. Previous research in this area has typically been limited to the application of code metrics such as : lines of code, McCabe's Cyclomatic number, and Halstead's software science variables. However, this research also investigates the relationship of four structure metrics: Henry's Information Flow measure, Woodfield's Syntactic Interconnection Model, Yau and Collofello's Stability measure and McClure's Invocation complexity, to various observed measures of complexity such as, ERRORS, CHANGES and CODING TIME. These metrics are referred to as structure measures since they measure control flow and data flow interfaces between system components.

Correlating the metrics to observed measures of complexity indicated that the Information Flow metric and the Invocation Measure typically performed as well as the three code metrics when project factors and subsystem factors were taken into consideration. However, it was generally true that no single metric was able to satisfactorily identify the variations in the data.

PART I

INTRODUCTION

One of the basic objectives of software engineering is to transform the creation of software systems from an artistic, poorly understood, and even undisciplined, activity into a carefully controlled, methodical, and predictable enterprise. To make software an engineerable product, it is important that the designers, implementors and maintainers of software systems be able to express the characteristics of the system in objective and quantitative terms (e.g. software performance [Lync81], and software reliability [Musa75]) those characteristics relating to software quality are typically evaluated only in qualitative terms.

Quantitative measures of quality are collectively referred to as software metrics. One of the key --- and, unfortunately, too often neglected --- parts of software metric research is the validation of the metric on realistic software systems. The validation, however is vital because it establishes the relationships between the measures of software products (e.g., a complexity metric) and important resources in the software process (e.g., errors, coding time). One form of a validation analyzes the relationship between metric values and historical project data. Such validations provide the compelling evidence needed to gain acceptance and use of software metrics as a standard industry wide practice.

This paper contains results of experiments which first applied both structure and code metrics to three large scale software systems and then correlated these measurements to observed values of system complexity such as, ERRORS, CHANGES, and CODING TIME. Previous research in this area has typically been limited to the application of code metrics such as: lines of code, McCabe's Cyclomatic number, and Halstead's software science measures. However, this research also investigates the relationship of four structure metrics: Henry's Information Flow measure, Woodfield's Syntactic Interconnection Model, Yau and Collofello's Stability measure and McClure's Invocation complexity. These metrics are referred to as structure measures since they measure control and data flow interfaces between system components.

Of these four structure metrics only the Information flow metric has been previous applied and validated on a realistic non-trivial program -- the UNIX operating system[Henr79][Henr81][Henr84]. Two of the structure metrics, Woodfield's Syntactic Interconnection Model and McClure's Invocation complexity were applied with success on smaller scale programs. Woodfield[Wood80] conducted carefully 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 fourth structure metric used in

this research, the Stability measure of Yau and Collofello[Yau 80] was proposed in their original work, but no validation was performed by the authors.

This research has added to our knowledge base with regard to these four promising structure based metrics. The previous work of Kafura and Henry has been augmented by applying their Information Flow metric to software possessing radically different characteristics than the Unix operation system. The current study measured three commercially developed scientific programs each approximately 100,000 lines of FORTRAN source code. The initial work of Woodfield and of McClure has been extended by exploring the usefulness of their measurement on non-trivial programs. Furthermore, the research of Yau and Collofello has also been extended since their promising measure has been automated and experimented with on production software.

PART II

THE METRICS

The quality of a software product is determined by numerous quality attributes which emerge in increasingly detailed form as the life cycle progresses. Accordingly, numerous quantitative measures must be employed throughout the life cycle -- each measure defined to quantify one of the many quality attributes. Three general classes of software metrics can be distinguished: measures based on an automated analysis of the system's design structure, termed "structure 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 all three types of metrics is important for two reasons. First, these classes of metrics appear to be measuring different aspects of the system quality. In the original Information Flow study, Henry et al.[Henr81b] found their structure metric to be orthogonal to three code metrics. Similarly, Kafura et al.[Kafu84], presented additional evidence that structure metrics and code metrics were indeed measuring different dimensions of software complexity. Fundamentally, the quality of a system is too multifaceted to be measured by any one metric or by metrics in only one of these classes. Second, each class of metrics can be first used at different points in the software life cycle. While code and hybrid 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. Structure 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.

There are three code metrics used in this study. The first code metric is a simple size measure: lines-of-code (LOC). The second code metric is Halstead's software science "effort" measure[Hals77]. The third code metric is the measure of cyclomatic complexity defined by McCabe[McCa76]. Several carefully designed experiments have shown that meaningful relationships exist between these metrics and significant software characteristics (for example, [Curt79]). Properly used, these metrics can play a useful role during the later phases of the software life cycle (testing, acceptance, maintenance, etc.). However, serious objections have been raised against the experimental design and statitistical treatment of a number of the software science experiments [Hame82][Lass81][Shen83]. Even if these objections can be satisfied, code metrics are available too late in the life cycle to exert a major impact on the life cycle characteristics and thus, should not be used alone.

As indicated above, a structure metric is defined in terms of some observed connection between components of the system which have emerged during the (high level) design phase. Many design methodologies, typified by Ross' SADT

[Ross77] and Yourdan and Constantine's SA-SD [Your79], focus on identifying the components in the system and specifying how these components are structurally related through control and/or data connections. Accordingly, structure metrics use only these features, components and relationships among components, to define a numerical measure. In total, these connections define the system structure. Each metric focuses on some aspect of structure which affects complexity, comprehensibilty, maintainablilty, etc.. The four structure metrics surveyed below have been incorporated into our work.

A structure measure based on the data relationships among components is the imformation flow metric mentioned above [Henr79]. This metric identifies the sources (fan-in) and destinations (fan-out) of all data related to a given component. The data transmission may be through global data structures, parameters, or side-effects. The fan-in and fan-out are then used to compute a worst-case estimate of the communication "complexity" of this component. This complexity measure attempts to gauge the strength of the components's communication relationships with other components.

Another structure metric is McClure's metric [McCl78] which attempts to measure "invocation complexity". The relevant features in analyzing the invocation complexity of a component are: (1) all variables which control (via conditional or iteration statements) the invocation of the component; and (2) all components which can affect the value of

these variables. In this metric a small invocation complexity is assigned to a component which, for example, is invoked unconditionally by only one other component. A higher complexity is assigned to a component which is invoked conditionally and where the variables in the condition are modified by remote ancestors or descendents of the component.

A third structure metric, defined by Woodfield [Wood80], is based on the concept of "review complexity". Woodfield observes that a given component must be understood in each context where it is called by another component or affects a value used in another component. In each new context the given component must be reviewed. Due to the learning from previous reviews, each review takes less effort than the previous ones. Accordingly, a decreasing function is used to weight the complexity of each review. The total of all of these weights is the measure assigned to the component. Woodfield applied this measure successfully in a study of multi-procedure student programs.

The fourth and final structure metric used in out research is the "stabilty" measure defined by Yau and Collofello[Yau 80]. This measure directs attention to the ripple effect which occurs when a change to one component necessitates changes to other components. In this measure the flow of data through parameters and global variables is used to identify all possible components which could be affected by a change to a given component. Finally, three hybrid me-

trics are used. Each of these hybrid measures is a modification of one of the structure metrics. A hybrid metric determines a measure for a component by weighting the structure measure for that component by a code measure. To simplify matters, we have used only lines-of-code as the weighing term. Using any of the other code metrics does not change the overall character of the results. It should be noted that three of the ten metrics used in this project (Henry and Kafura's information flow, Woodfield's review complexity, and Yau and Collofello's stability measure) were originally posed by their authors as hybrid metrics. The three hybrid metrics are: information flow weighted by lines-of-code, Yau and Collofello's stability weighted by lines-of code, and Woodfield's metric weighted by lines-of-code. The McClure invocation complexity metric does not lend itself to such weighting.

For reference, the ten metrics described above will be referred to in subsequent tables and discussion by the following abbreviations:

1. LOC (lines-of-code)
2. EFFORT (Halstead's software science effort)
3. INFO (Henry and Kafura's unweighted information flow complexity)
4. INVOKE (McClure's invocation complexity)
5. REVIEW (Woodfield's review complexity)
6. STABILITY (Yau and Collofello's stability measure)

7. INFO-LOC (weighted information flow)

8. REV-LOC (weighted review complexity)

9. STAB-LOC (weighted stability measure)

Also, recall that the first three of the metrics are code metrics, the middle four are structure metrics, and the last three are hybrid metrics.

PART III

SOURCE CODE AND HISTORICAL DATABASE

Critical resources necessary for this project were provided by the Software Engineering Laboratory (SEL) headed by Frank McGarry, 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. Througout 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 FORTRAN subroutines, functions, and data blocks, known as components. Within a given project, subsystems gen-

erally do not share common components. That is, routines from one subsystem rarely call routines from another subsystem. Communication between subsystems is usually acheived 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 associated with each of the three projects. The database itself contains numerous files and a wide range of information. Selected for this study were the following variables:

Count Based Dependent Variables

1. 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.
2. Component Errors: A discrepency between a specification and its implementation. The specification might be requirements, design specification, or coding specification.

Time Based Dependent Variables

1. Design Hours: The time recorded by SEL personnel to create, to read and to review the design of an individual component or subsystem.

2. Code Total Hours: The time recorded by SEL personnel to implement, to read and to review the coding of an individual component or subsystem.
3. Code Hours: The time recorded by SEL personnel just to implement an individual component or subsystem.
4. Test Hours: The time recorded by SEL personnel to test an individual component or subsystem, including unit testing, integration testing, and testing reviews.
5. 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 for the purposes of this research. 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]. 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:

1. Less than one hour.
2. One hour to one day.
3. One day to three days.
4. 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.

PART IV

FILTERING THE DATA

Because of the nature of the data collection process and the enviroment in which the systems were constructed, not all of the components from the three projects were used in this experimentation. This section will describe the two subsets of the Nasa/Goddard data that were used in the following analyses. 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. The second subset of components, known as DATASET B, was used in those experiments which related the metrics to time base dependent variables. This set of components 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. A more detailed description of each of these two sets of components is presented below.

One subset of components (DATASET A) used for this research contains components which will be used when analyzing the relationship between the metrics and the count based measures. DATASET A is a collection of those subroutines and functions which were either totally new components or

extremely modified old components. 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. It is necessary to omit slightly modified or duplicated code from consideration since error counts and development times for reused components do not accumulate from project to project in the SEL reporting mechanism. These reused components, having already undergone rigorous testing and debugging when intially implemented, would distort the count based measurements. Similarly, the time based measurements would also be biased since they would not reflect the original effort expended in the development of these slightly modified or duplicated components. 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 1 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 1

Data Means for Various Component Types

Component Class	Errors	Changes	Weighted Changes	Total Time
(1) New Code Components	2.03	2.11	44.20	16.09
(2) Extremely Modified Components	1.47	2.62	43.57	13.32
(3) Slightly Modified Components	0.43	1.11	18.98	6.06
(4) Duplicated Components	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 duplicated code from the experimentation, but also eliminates components which fail to

pass two additional criteria. Both of these criterion provide a validity check on the reporting of time based dependent variables.

The first criteria 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.

PART V

RESULTS

The experiments conducted for this research, correlating the set of software metrics with observed measures of software complexity recorded in the database, are partitioned into two classes. The first class of experiments, referred to as Component Level Analysis, compared individual component complexities, as determined by the ten software metrics, with their reported count based and time based data. The second class of experiments, known as Subsystem Level Analysis, related individual subsystem complexities with their associated count based and time based values. A subsystem's software complexity is defined by summing the metric values of its individual components. Count based values were similarly established for each subsystem. Time based measures of entire subsystems reflect the time spent developing individual components as well as any overhead time attributed to the subsystem as a whole. In both classes of experiments, Spearman rank correlation coefficients formed the basis of the comparison. Of course, underlying each of these analyses is the assumption that observed data, such as error counts and time to develop software, is highly associatied with complexity. For example, as software complexity increases it is assumed that the number of errors found in the software or the time to develop the software should also increase.

COMPONENT LEVEL ANALYSIS

The section presents the Spearman coefficients obtained by correlating, on a component by component basis, the ten complexity metrics with the four operational measures of complexity: ERRORS, CHANGES, WEIGHTED CHANGES and CODING TIME. The goal of this type of analysis is to determine how strongly the proposed complexity metrics relate to errors and actual effort encountered during the development of software in a realistic environment [Bas83]. The other time based measures, design time and testing time, were not included in this analysis since these values were largely reported on a subsystem basis.

The Spearman rank coefficients relating the complexity metrics to the three count based variables are first presented. As previously stated, not all components from the three projects were analyzed. Since, the metrics were related to count based variables, only newly developed or extremely modified components (DATASET A) were used in this experimentation. The components of DATASET A can be partitioned hierarchically by project and by subsystem, thus giving rise to three separate but similar experiments. The first experiment collectively measures all the components of DATASET A yielding correlations which span across all three projects. The second experiment attempts to filter out project related factors by correlating components within the same project. Finally, components within the same subsystem

were collectively measured. Correlating components within the same subsystem helped to filter out programmer related factors since in the Nasa/Goddard environment, subsystems are typically developed by one or two programmers.

The Spearman coefficients derived by collectively measuring components which span across projects are found in table 2. The correlations between the metrics and both the ERROR data and the CHANGE data are rather low indicating little or no relationship between them. Correlating the metric complexities with WEIGHTED CHANGES also yielded unimpressive correlations. Although these results are somewhat disappointing, they were not totally unexpected. Previous research by Basili et al. [Bas83], found similar results when comparing errors with a host of software science metrics. An identical weighting criteria was also applied to errors resulting in similar correlations being generated. These authors noted that the discrete nature of error reporting, with 52 percent of their tested components having zero errors, could account for the low Spearman correlations. Analogously, the distribution of the error and change data used in this experimentation is similarly skewed.

In an attempt to account for project dependent effects, the components of DATASET A were partitioned and analyzed by project. Previous experiments by Basili[Bas83], analyzing different data generated from the same environment(Nasa/Goddard) pointed toward the existance of project

TABLE 2

Metrics Correlated With Count Based Data

	ERRORS	CHANGES	WEIGHTED CHANGES
LOC	.470	.497	.372
CYCLO	.373	.455	.402
EFFORT	.467	.512	.413
INFO-LOC	.364	.452	.338
REVIEW-LOC	.254	.285	.347
STAB-LOC	.219	.272	.276
INFOFLOW	.321	.415	.313
INVOKE	.398	.440	.360
REVIEW	?	?	?
STABILITY	.174	.267	.232

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

related factors. The Spearman correlations between the metrics and the error data for the three individual projects are presented in table 3. For comparison, this table also includes the across project correlations. The correlations presented in table 3 indicate that 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 structure metrics have somewhat weaker associations to ERRORS than do either the code or hybrid metrics. However, of four structure metrics, Henry's INFOFLOW and McClure's INVOKE measure exhibit the strongest relationship to ERRORS.

TABLE 3

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
INFO-LOC	.364	.496	?	.536
REVIEW-LOC	.254	.509	.281	?
STAB-LOC	.219	.175	.267	.257
INFOFLOW	.321	.462	?	.489
INVOCATION	.398	.331	.406	.479
REVIEW	?	.250	.209	-.249
STABILITY	.174	.228	?	.287

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

Although, the correlations found in table 3 are not extremely high, they usually indicated stronger relationships between the metrics and the error data when project related factors are minimized. The unexplained exception to this trend are the correlations between the code metrics and errors for project B components. It is interesting to note that Basili's previous work also noted an anomalous project.

Similar observations can be made when comparing the metrics to the other two count based metrics. The correlations between the metrics and CHANGES for the three projects are presented in table 4. The correlations found in this table are noticeably higher for PROJECT C. In particular, the correlations for the three code metrics and the INFO-LOC metric are all relatively high, exceeding 0.600 . The correlations generated by the unweighted information flow metric, INFOFLOW have also improved, jumping from 0.415 to 0.582 . It is interesting to note that the correlations between CHANGES and the metrics for PROJECT A are highest for the two information flow based metrics. Three metrics, STABILITY, REVIEW, and STAB-LOC consistently generate correlations which indicate weaker associations with CHANGES than the other metrics.

Table 5 contains for the Spearman values for comparisons made between the metrics and WEIGHTED CHANGES for only

TABLE 4

Component Analysis Within Projects: CHANGES

	ALL PROJECTS	PROJ. A	PROJ. B	PROJ. C
LOC	.497	.487	.309	.682
CYCLO	.455	.444	.210	.603
EFFORT	.512	.468	.304	.679
INFO-LOC	.452	.540	.269	.630
REVIEW-LOC	.285	.477	.352	?
STAB-LOC	.272	.239	.381	.318
INFOFLOW	.415	.527	.240	.582
INVOKE	.440	.455	.437	.482
REVIEW	?	.212	.291	-.233
STABILITY	.267	.338	.242	.370

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

PROJECT A and PROJECT B, as weighted change data was not available for PROJECT C. The correlations obtained by analyzing components for PROJECT A are somewhat stronger than those obtained from PROJECT B's components. However, in either case, the within project correlations are not noticeably higher than the across project correlations.

Athough correlations between the metrics and the count based data were generally stronger when components were par-

TABLE 5

Component Analysis Within Projects: WEIGHTED CHANGES

	ACROSS	PROJ. A	PROJ. B
LOC	.372	.409	.298
CYCLO	.402	.394	.308
EFFORT	.413	.390	.386
INFO-LOC	.338	.469	.313
REVIEW-LOC	.347	.402	.305
STAB-LOC	.276	.248	.443
INFOFLOW	.313	.463	.282
INVOKE	.360	.427	.358
REVIEW	?	.163	?
STABILITY	.232	.297	.365

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

titioned by project, these Spearman values were still found to be somewhat low. Hypothesizing the existance of "within projects" effects, the data was further partitioned by subsystems. Correlating components within the same subsystem helped to filter out programmer related factors since in the Nasa/Goddard environment, subsystems are typically developed by one or two programmers. The results of these correlations are reported in table 6, table 7 and table 8. The Spearman coefficients found in table 6 result from correlating the

code metric values with the CHANGE data for components within fifteen subsystems. Six subsystems were omitted from the table since all their correlations had significance levels much greater than 0.05 . For many of the subsystems the correlations between the code metrics and the CHANGE data are dramatically higher than those obtained from previous experiments. This increase may indicate the presence of "within subsystem" effects. Similarly, the within subsystem correlations associating the CHANGE data with the hybrid metrics are noticeably higher. Comparing the correlations in table 6 with table 7 it appears that both INFO-LOC and REVIEW-LOC typically perform as well as the code metrics, and at times indicate stronger associations to CHANGES than do the code metrics. The correlations in table 8 give the within subsystem correlations between the four structure metrics and the CHANGE data. Three metrics: INFOFLOW, STABILITY, and INVOKE typically possess higher coefficients than the across project or within project correlations involving structure metrics.

The Spearman coefficients given in table 9 show the association between the ERROR data and selected metrics for components within a given subsystem. These metrics were selected since they indicated the strongest association to ERRORS for the three metric classes: code, hybrid and structure. Again the data in this table points toward the existance of subsystem related factors. It is interesting that the cor-

TABLE 6

Code Metrics Correlated with CHANGES:Within Subsystem

SUBSYSTEM	LOC	CYCLO	EFFORT
A.1	.771	?	.680
A.3	.695	.702	.644
A.4	.574	.640	.540
A.5	.487	.488	.450
A.6	.613	.545	.590
A.7	.550	.441	.518
B.1	.714	.642	.742
B.2	.571	.532	.631
B.3	.658	.569	.665
B.4	.815	?	.815
B.5	.716	.681	.728
C.1	?	?	.948
C.2	.400	.374	.524
C.3	.434	.655	.518
C.6	.357	.318	.409

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

relation between the ERROR data and the LOC metric for the three PROJECT B subsystems are relatively high, ranging from 0.652 to 0.709 . These coefficients are markedly higher than the 0.257 coefficient generated by relating the ERROR data and the LOC metric to all PROJECT B components. However, notice that no metric consistently reports strong associations with the components in all subsystems. For example, the coefficient between the LOC metric and subsystem A.3 is 0.752, while for subsystem A.5 it is 0.323 . Furthermore, notice that no single metric consistently reports stronger

TABLE 7

Hybrid Metrics Correlated With CHANGES:Within Subsystem

SUBSYSTEM	INFO-LOC	REVIEW-LOC	STAB-LOC
A.1	.862	.680	?
A.3	.695	.725	.405
A.4	.467	.511	.484
A.5	.612	.493	?
A.6	.553	.618	?
A.7	.552	.541	.509
B.1	.664	.687	.450
B.2	.491	.502	?
B.3	.726	.700	?
B.4	.926	.741	?
B.5	.632	?	.327
C.2	.588	.427	.644
C.3	.531	?	.604
C.6	.509	.344	.354

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

association to ERRORS than the other metrics. Of the eleven subsystems given in table 9, the INFO-LOC indicates the strongest association with the ERROR data for seven of the subsytems, while the LOC metric has the highest coefficients for three of the subsystems. McClure's INVOKE measure relates best to the ERROR data for subsytem A.4, with a coefficient of 0.611 .

The Spearman rank coefficients relating the complexity metrics to a single time based variable, CODING TIME, are now presented. The other time based dependent variables,

TABLE 8

Structure Metrics Correlated With CHANGES:Within Subsystem

SUBSYSTEM	INFOFLOW	REVIEW	STABILITY	INVOKE
A.1	.862	?	.669	.596
A.3	.684	?	.391	.393
A.4	.451	?	.595	.694
A.5	.630	.344	?	.616
A.6	.477	.276	.261	.474
A.7	.476	?	?	?
A.8	.402	?	.554	.453
B.1	.613	?	.652	.619
B.2	.467	?	?	?
B.3	.732	?	.476	?
B.4	.963	?	?	?
B.5	.544	?	.485	.372
C.2	.577	?	.623	?
C.3	.543	?	.526	?
C.6	.434	?	.397	.418

KEY ? p > .05. otherwise p < .05

such as DESIGN TIME and TESTING TIME, were not correlated with the metric values since these "times" were not typically reported on a component by component basis. As previously mentioned, since CODING TIME is a time based dependent variable only components from DATASET B will be measured. Once again, the components were partitioned hierarchically by project and by subsystem giving rise to three separate but similar experiments.

TABLE 9

Selected Metrics Correlated With ERRORS

SUBSYSTEM	LOC	INFO-LOC	INVOKE
A.1	.743	.844	?
A.3	.752	.721	.447
A.4	.556	?	.611
A.5	.323	.462	?
A.6	.452	.466	.409
B.1	.709	.637	.558
B.3	.652	.728	?
B.5	.684	.558	.342
C.2	.402	.562	?
C.3	.570	.628	?
C.6	.382	.401	.393

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

The results of the first two experiments are summarized in table 10. The first experiment collectively measured all the components of DATASET B yielding correlations which span across all three projects. The results of these correlations are given in in the column labelled ALL PROJECTS in table 10. From the data presented in this table, it appears that code metrics relate more strongly to CODING TIME than either structure or hybrid metrics. Among the structure and hybrid metrics, the INFOFLOW metric (.437) and the INFO-LOC metric (.479) indicate the strongest association to CODING TIME.

Also presented in table 10 are the results of correlations which partitioned the components of DATASET A by pro-

TABLE 10

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
INFO-LOC	.479	.616	.465	.517
REVIEW-LOC	.164	.583	.424	?
STAB-LOC	.271	.409	.320	.266
INVOKE	.286	.293	?	.350
INFOFLOW	.437	.599	.421	.468
STABILITY	.303	.468	.392	.306
REVIEW	-.180	?	?	-.327

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

ject. This experiment was conducted in order to control for project related factors. 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 so consistent for Project B.

The coefficients found in table 11, table 12, and table 13 were obtained by correlating the CODING TIME values with the ten metric values for components within the same subsytem. The coefficients found in table 11 indicate that the

TABLE 11

Code Metrics Correlated With CODING TIME

SUBSYSTEM	LOC	CYCLO	EFFORT
A.3	.559	.498	.517
A.7	.438	.564	.400
B.1	.595	.562	.585
B.3	?	?	?
B.5	.773	.745	.770
C.3	.415	.493	.468
C.6	.522	.408	.527

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

TABLE 12

Hybrid Metrics Correlated With CODING TIME

SUBSYSTEM	INFO-LOC	REVIEW-LOC	STAB-LOC
A.3	.466	.555	.314
A.7	.668	.467	.523
B.1	.659	.595	.385
B.3	.586	?	?
B.5	.439	?	?
C.3	.595	?	?
C.6	.311	.484	.320

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

three code metrics all relate to CODING TIME in approximately the same manner. For any given subsystem small differences in the coefficients of the code metrics do exist. How-

ever, from subsystem to subsystem, fluctuations in the coefficients are generally consistent. For example, the coefficients relating the code metrics with CODING TIME for subsystem B.1 range between 0.562 and 0.595, while the coefficients for subsystem B.5 range between 0.745 and 0.773 .

TABLE 13

Structure Metrics Correlated With CODING TIME

SUBSYSTEM	INVOKE	REVIEW	STABILITY	INFOFLOW
A.3	?	?	?	.454
A.7	.630	?	.687	.687
B.1	.602	.380	.572	.622
B.3	?	?	?	.626
B.5	.513	?	?	?
C.3	.521	?	?	.572
C.6	.459	?	?	?

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

The Spearman values found within table 12 were generated by correlating the three hybrid metrics with CODING TIME. These values indicate that no single hybrid metric is uniformly superior when relating to CODING TIME. For two of the subsystems (A.7 & B.1), the INFO-LOC metric correlated more strongly with CODING TIME than either the REVIEW-LOC metric or the STAB-LOC metric, while the REVIEW-LOC metric correlated more closely with CODING TIME for two other subsystems (A.3 & C.6).

The correlations between the four structure metrics and CODING TIME for components within the same subsystem are presented in table 13. The data in this table shows that the INFOFLOW metric yields the strongest correlations to CODING TIME for five of the seven subsystems. Of the two subsystems which the INFOFLOW metric did not yield a statistically significant result, McClure's INVOKE metric generated correlations of 0.513(B.5) and 0.469(C.6). It is difficult to assess the relationship between CODING TIME and both the REVIEW metric and the STABILITY metric since they often gave correlations which were statistically insignificant.

SUBSYSTEM LEVEL ANALYSIS

The results of the previous section were derived by correlating the the software metrics with the developmental data on a component by component basis. The purpose of this section is to present correlation coefficients between the software metrics and the developmental data where the unit of measure is not a single component, but rather an entire subsystem. In order to obtain such coefficients software complexity measures and developmental data values had to be defined for each of the twenty-one subsytems used in this research. This was not difficult since in the Nasa/Goddard environment, components are grouped according to the subsystem in which they are defined. As previously mentioned, this grouping is well defined since components do not belong to more than a single subsystem. Communication between subsys-

tems is typically achieved by exchanging data via system wide tables or by passing data through upper level modules. Thus software complexity measures for a given subsystem were derived by summing the complexities of each of its individual 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 were obtained by not only summing the times associated with the development of individual components, but by also adding any overhead time attributed to an entire subsystem. This overhead time is primarily due to the inability of programmers to attribute their effort to individual components. As earlier stated, this often occurred during the design and testing phases of the software life-cycle, rendering component level analysis for these two activities impossible. However, for the subsystem level, the five time based measures were incorporated into the analysis.

There are four reasons why software systems need to be examined at the subsystem level. One motivation for studying software at the subsystem level is that typical large scale systems are oftened partitioned into sets of related components, i.e. subsystems. Usually these subsystems have a well defined structure of their own which can be isolated and evaluated as separate entities. It is important to identify those subsystems which have poor substructures and need

to be redesigned. Furthermore, management decisions are not often tuned to the development of a single component, but rather are guided by subsystem related information. For example, subsystems possessing an undue amount of complexity will be more difficult to design, build and maintain, hence will require more staffing. A second reason for studying the software at the subsystem level is that count based and time based data is difficult to collect and to verify at the component level. It is often not possible to identify a single component as the cause of an error or to correctly estimate the time spent working on a particular component. To achieve accurate data collection at such a fine level of detail may not be cost effective for most development environments. It would seem that collecting accurate count based and time based data at the subsystem level is more plausible. A third motivation for incorporating an analysis at the subsytem level is to investigate the relationships between the metrics and development times which span multiple phases of the software life-cycle. Recall, in the component level analysis this was not possible because development time data was available only for the coding phase. However, development time data was reported for the design, implementation and test phases of the lifecycle at the subsystem level. In particular, it would be of interest to identify metrics which are meaningful at all phases of a system's life-cycle. The fourth motivation for studying software at

the subsystem level is a direct result of the component level analysis presented earlier. The correlations in the previous section, although consistent with previous research, were disturbingly low. These disappointing numbers could exist for a variety of reasons. One reason could be that count based and time based data, being difficult to collect and to verify at the component level is too perturbed by random effects. Another, equally plausible reason for such poor relationships between the metrics and the development data at the component level is the metrics may not be able to detect subtle differences which exist between individual components. Indeed, metric research may require that components need to be grouped together in order to identify hidden trends in the data. The notion of grouping components has already been applied with promising results by Kafura and Canning [Kafu85a][Kafu85b].

The results of the subsystem level analysis given below describes two similar but distinct experiments. The first experiment correlated the subsystem defined complexities with the subsystem defined data using all twenty-one subsytems available. The second experiment performed analogous correlations using only nine of the subsystem. Twelve of the subsystems were filtered out since they contained components which were constructed during the development of previous projects. Once again the conservative Spearman rank technique was used to generate correlations which indicated the

relationship between the metrics and both the time based and count based measures.

The Spearman coefficients obtained by correlating the five development times with the ten metrics for all twenty-one subsystems are given in table 14. 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 in table 14 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 apparently the metric most closely related to five development times. Of the remaining structure based 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 proposed metrics.

The Spearman rank correlation coefficients in table 15 indicate relationships between the metrics and the count based data. Once again, it is apparent that collectively,

TABLE 14

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
STABILITY	.620	.720	.540	.510	.444
REVIEW	.488	.587	?	?	?
INVOKE	.790	.811	.750	.692	.632
INFO-LOC	?	?	?	?	?
REVIEW-LOC	?	.528	?	?	.444
STAB-LOC	.624	.696	.542	.545	.468

KEY: ? p > 0.05, otherwise p < 0.05

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 structure based 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 15

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

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

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

In the above analysis, all twenty-one subsystems were used in the experiments. However, many of these subsystems contained components which were previously developed. As previously asserted, count based and time based data for these components will likely be skewed. In an effort to control for this effect, 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 below in table 16. The subsequent table contains the Spearman rank coefficients generated by correlating the metrics with the count based variables.

TABLE 16

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

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

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 the improvement in the quality of the data. Those subsystems which were largely made up of reused code contained components whose time and count data was artificially low, thus perturbing the experiment.

TABLE 17

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

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

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

PART VI

SUMMARY AND CONCLUSIONS

In this paper, results were presented which related both structure and implementation metrics to observed measures of complexity. Initially the data had to be filtered to accomodate problems with the data collection facility. The correlation between metrics on both a component by component basis and on a subsystem by subsystem basis. When performing component level analysis; "across project", "within project" and "within subsystem" experiments were done. For the across project experiments, it was generally true that no metric satisfactorily explained the behavior of the count based or time based data. Partitioning the components on a "within project" basis generally enhanced the correlations for two of the three projects. Although the correlations were not extremely high, the code metrics as a class seemed to relate best to the count and time based data. Of the four structure metrics, the information flow metric indicated the strongest association with the observed data. Furthermore, when the information flow metric was weighted by lines of code, as in Henry's original definition, the correlations were somewhat higher. It seems that the unweighted REVIEW metric is not a very meaningful measure of complexity, but the REVIEW-LOC metric is able to show some association to the data for PROJECT A. The ability of some metrics to correlate with ERRORS,CHANGES, and CODING TIME generally

improved when "within subsystem" factors were taken into consideration. When correlated with CHANGES, the INFOFLOW metric typically provided the strongest coefficients of all the structure metrics. However, for some subsystems, either the STABILITY metric or the INVOKE metric yield the highest coefficients. The inability of a single metric to consistently give the highest coefficients also occurs when the "within subsystem" correlations were performed between the metrics and both the ERROR and CODING TIME data. These results indicate that more that one metric is needed to explain the various dimensions of software complexity.

When the unit of measurement was an entire subsystem rather than a single component, the Spearman coefficients increased dramatically. Initially all twenty-one subsystems available in this research were analyzed. For this analysis, McClure's INVOKE measure consistently reported the highest coefficients with the time based data. As a rule, the correlations improved when only those subsystems with the more reliable data were analyzed. However, since the number of data points is relatively small (N=9), it is prudent not to infer too much from the high coefficients generated from the subsystem level analysis.

The high correlations generated by the subsystem level analysis may, at best, suggest that the metrics are better able to identify trends in the data when individual components are logically grouped. It may be that individual me-

trics are not able to detect subtle differences existing between individual components. This is especially true when using a conservative non-parametric statistic to judge the "goodness" of a particular metric.

REFERENCES

1. [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

2. [Curt79]Curtis,W. , Sheppard,S.B. , and Milliman,P.M. "Third Time Charm : Stronger Prediction of Programmer Performance by Software Complexity Metrics , " Fourth International Conference on Software Engineering September 1979 , Munich , Germany , 356-360 .

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

4. [Hame81]Hamer, Peter, Frewin, Gillian, "M.H. Halstead's Software Science - A Critical Examination," ITT Technical Report No. STL 1341, July 1981.

5. [Hane72]Haney, Frederick, "Module Connection Analysis - A Tool for Scheduling Software Debugging Activities,"

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

7. [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.

8. [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.

9. [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.

10. [Kafu82]Kafura,D., Henry,S., "Software Quality Metrics Based on Interconnectivity," The Journal of Systems and Software , Vol. 2 , pp. 121-131, 1981.

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

12. [Kafu85a]Kafura D.,Canning J., "A Validation of Software Metrics Using Many Metrics and Many Resources" Proceedings of the International Conference of Software Engineering August 1985

13. [Kafu85b]Kafura D.,Canning J., "Exposing Useful Trends in Metric Data Through Group Level Analysis" IEEE Transactions on Software Engineering, Submitted August 1985.

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

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

16. [Musa75]Musa,J.D. "A Theory of Software Reliability and Its Application," IEEE Transactions on Software Engineering, vol.SE-1, no. 3, pp. 312-327, 1975

17. [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

18. [Ross77]Ross,Douglas T., "Structured Analysis (SA): A Language for Communicating Ideas," IEEE Transactions on Software Engineering, Jan. 1977.

19. [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.

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