Title
Analyzing software data bindings in large-scale systems

Author
Selby, Richard W.

Publication Date
1988

Peer reviewed

Analyzing Software Data Bindings in Large-Scale Systems

Richard W. Selby

Department of Information and Computer Science
University of California, Irvine, California 92717 [1]

UCI–TR–88-31

June 1988

[1]This work was supported in part by the National Science Foundation under grant CCR–8704311 with cooperation from the Defense Advanced Research Projects Agency (ARPA Order 6108, Program Code 7T10), by the National Science Foundation under grant DCR–8521398, and by IBM under the Shared University Research (S.U.R.) Program.

Abstract

One central feature of the structure of a software system is the coupling among its components (e.g., subsystems, modules) and the cohesion within them. The purpose of this study is to quantify ratios of coupling and cohesion and use them in the generation of hierarchical system descriptions. The ability of the hierarchical descriptions to localize errors by identifying error-prone system structure is evaluated using actual error data. Measures of data interaction, called data bindings, are used as the basis for calculating software coupling and cohesion. A 135,000 source line system from a production environment has been selected for empirical analysis. Software error data was collected from high-level system design through system test and from some field operation of the system. A set of five tools is applied to calculate the data bindings automatically, and cluster analysis is used to determine a hierarchical description of each of the system's 77 subsystems. An analysis of variance model is used to characterize subsystems and individual routines that had either many/few errors or high/low error correction effort.

1 Introduction

Several researchers have proposed methods for relating the structure of a software system to its quality (e.g., [BE82] [HK81] [Eme84]). One pivotal step in assessing the structure of a software system is characterizing its coupling and cohesion. Intuitively, the *cohesion* in a software system is the amount of interaction *within* pieces (e.g., subsystems, modules) of a system. Correspondingly, *coupling* in a software system is the amount of interaction *across* pieces of a system. Cohesion may sometimes be referred to as "strength." Various interpretations for coupling and cohesion have been proposed [SMC74]. In this paper, we present an empirical study that investigates hierarchical software system descriptions that are based on measures of cohesion and coupling. The study evaluates the effectiveness of the hierarchical descriptions in identifying error-prone system structure. Our measurement of cohesion and coupling is based on intra-system interaction in terms of *software data bindings* [BT75] [HB85]. Our measurement of error-proneness is based on software error data collected from high-level system design through system test; some error data from system operation are also included.

The research approach was based on the application of a data collection and analysis methodology in a large, production software environment. The use of the methodology incorporates definition of the required data, collection of the data, and appropriate data analysis and interpretation. The research project was conducted in three phases, and they roughly corresponded to the activities of data definition, collection, and analysis and interpretation.

The paper is organized into several sections. Section 2 discusses the software project selected. The data bindings software analysis and supporting tools are described in Section 3. The data analysis appears in Section 4. Section 5 presents the interpretations and conclusions.

2 Selected Software Project

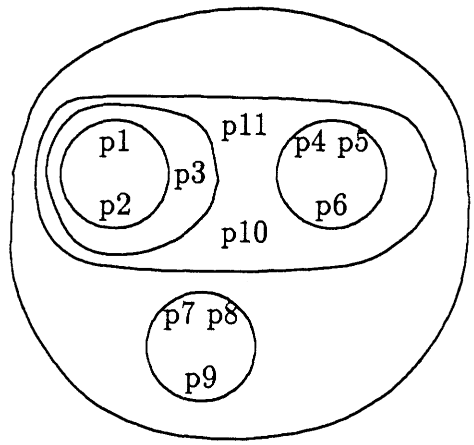
The software project selected for study is the next release of an internal software library tool. The previous system release contains approximately 100,000 source lines. The production of the next release requires the development or modification of approximately 40,000 source lines. Hence, the total size of the next system release is approximately 135,000 source lines.

The system is written in four languages: a high-level programming language similar to PL/I, a language for operating system executives, a user-interface specification language, and an assembly language. The static source code metrics discussed later, including the data bindings analysis, pertain to only the system portion written in the high-level source language. This portion constitutes approximately 70% of the lines in the system and the vast majority of the system logic and intra-system interactions. Project duration, including system and field test, spanned approximately 16 months and maximum staffing included 23 persons.

System Characterization There are 163 source code files in the system containing a total of 451 source code *routines*. A routine is a main program, procedure, or function. The number of routines per source code file varies from 1 to 21. On the average, there are 2.8 routines per source code file. There are 77 executable features in the system, referred to as *subsystems* in the paper. These subsystems can be thought of as groups of routines collected together to form functional features of the overall system. The number of source files linked together to form a subsystem varies from 1 to 82. On the average, 26.3 source files are linked together into a subsystem. The same source file is bound into 12.4 different subsystems on the average. Subsystems averaged 19,000 source lines in size, including comments.

Data Collection The data collection and analysis methodology employed the goal-question-metric paradigm [BW84] to result in a set of software product and process metrics, a "metric vector," sensitive to the cost and quality goals for the particular environment. The data was collected and analyzed at the same time the project took place. An important goal was to minimize the impact of the data collection process on the developers. See [SB] for a description of the data definition, collection, and analysis methodology, an explanation of the metric vector concept, a description of the underlying data collection forms, the data collection process effectiveness, and some lessons learned and recommendations based on the use of the data collection and analysis methodology.

Figure 1: Example hierarchical cluster based on software data bindings. Procedures and functions are denoted by p_i, and clusters are denoted by circles. The smaller clusters are relatively tighter (and form earlier), while the larger clusters are relatively looser (and form later). The clusters define a system hierarchy in the form of a tree: the smaller clusters at the leaf nodes and the largest cluster at the root node.



3 Data Bindings Analysis

3.1 Clustering with Data Bindings

One primary goal for this study was to investigate the relationship of "software data bindings" to software errors [HB85]. "Data bindings" are measures that capture the data interaction across portions of a software system. The theoretical background for the measures is described in [HB85]. Earlier studies have revealed insights about the usefulness of data bindings in the characterization of software systems and their errors [BT75] [HB85]. In order to describe the data bindings analysis process applied, we first introduce some terminology (see also [HB85]).

Potential Data Binding A potential data binding is defined as an ordered triple (p,x,q) where p and q are procedures and x is a variable within the static scope of both p and q. Potential data bindings reflect the possibility of a data interaction between two components, based upon the locations of p, q, and x. That is, there is a possibility that p and q can communicate via the variable x without changing or moving the definition of x. Whether x is actually mentioned inside of p or q is irrelevant in the computation of potential data bindings.

Used Data Binding A used data binding is a potential data binding where p and q use x for either reference or assignment. The used data binding requires more work to calculate than the potential data binding as it is necessary to look inside the components p and q. It reflects a similarity between p and q (they both use the variable x).

Actual Data Binding An actual data binding is defined as a used data binding where p assigns a value to x and q references x. The actual data binding is slightly more difficult to calculate as a distinction between reference and assignment must be maintained. Thus more memory is required but there is little difference in computation time. The actual data binding only counts those used data bindings where there may be a flow of information from p to q via the variable x. The possible orders of execution for p and q are not considered. That is, there may be other factors (e.g., control flow conditions) which would prevent such communication.

There are stronger levels of data bindings. However, in this study we calculated *actual data bindings*. This level of data bindings seems to offer adequate measure of similarity while not requiring complex data flow analysis that stronger levels need. Essentially, we are erring in the direction of safety (as done, for example, by code optimizers) by assuming that procedures may influence one another unless we can show otherwise.

First, we calculated the actual data bindings in the system. Then, we applied the statistical technique of clustering [Eve80] to the data bindings information to produce a hierarchical description for the software system (see Figure 1). The clustering takes place in a bottom-up manner. The process iteratively creates larger and larger clusters, until all the elements have collapsed into a single cluster. The elements in the clusters are the procedures and functions in the system. The elements with the greatest interaction, in terms of actual data bindings, cluster together. The technique of clustering has been applied previously to partition a large system into subsystems in [BE82]. Hierarchical clusters have been formally defined in [JS71].

3.2 Data Bindings Analysis Software

A set of five software tools was developed to calculate these hierarchical, data bindings clusters and applied to the 77 subsystems in the selected project. The source code is the only input required by the tools for automatic generation of a hierarchical system description. The tools determine the data bindings that occur among the routines in the source code and then use them in cluster analysis as a measure of similarity. Four of the five tools are language independent; the other tool — a major one — is language dependent. For a description of the tools, see [SB]. The trees of clusters (see Figure 1) output by the tools provide a form of system documentation — they give a hierarchical view of the subsystems with respect to data usage.

4 Data Analysis

The data collection and analysis methodology was successful in producing a wide range of statistically significant results. Several analysis techniques, including analysis of variance and cluster analysis, were employed in the study.

4.1 Terminology

Throughout the analysis and interpretation, we use the terms *subsystems* and *routines* as follows:

- Routine — A routine is a main program, procedure, or function. There are a total of 451 source code routines in the system.
- Subsystem — A subsystem is a large set of routines that are linked together to form an executable system feature. There are 77 executable features in the system. They average 19,000 source lines in size.

A routine is linked into 12.4 subsystems on the average. Therefore, the total size of the whole system is not 77 x 19,000 = 1,463,000 source lines; the total size is approximately 135,000 source lines. See Section 2 for further description of the subsystems and routines in the software system.

We used the analysis tools described in Section 3 to produce hierarchical descriptions for each of the 77 subsystems (see Figure 1). The hierarchical descriptions are rooted, connected trees that indicate the internal subsystem structure. Each routine in a subsystem occurs as a leaf node in the tree exactly once. Subtrees indicate groupings of routines that form natural *clusters* based on the data bindings criteria. There is a one-to-one correspondence between subtrees and clusters. A cluster can contain either routines or other clusters. In other words, the root node of a subtree can have as its children either leaf nodes (i.e., routines) or the root node of another subtree (i.e., a subset of its own routines that form a smaller cluster).

In the software system being analyzed, a routine may be linked into more than one subsystem. Each of the 77 subsystems has a separate hierarchical description. Therefore, a routine appears in the hierarchical description of each subsystem into which it is linked. A routine may cluster with different sets of routines in different subsystems.

Associated with each cluster in a subsystem is a number ranging from 0 to 100. This number reflects the nature of the binding of the routines in the cluster. This number is interpreted as the following ratio:

$$\frac{\textit{the coupling of the cluster with other clusters in the subsystem}}{\textit{the internal strength of the cluster}}$$

That is, the number captures the coupling/strength ratio for a cluster of routines within a subsystem. The coupling/strength ratios range from 0 to 100 since they are calculated on a relative scale. The use of the word "relative" here means relative to the coupling/strength ratios that could result from the range of all possible occurrences of data bindings. In the data bindings analysis process, the clusters are formed in a bottom-up manner. The clusters with the lowest coupling/strength ratios form in the first iteration, the clusters with the next lowest ratios form in the second iteration, and so forth.

The lower a cluster's coupling/strength ratio is, the lower the relative coupling with other clusters and the higher the relative strength of binding within the cluster. The higher a cluster's coupling/strength ratio is, the higher the relative coupling with other clusters and the lower the relative strength of binding within the cluster. Software engineering principles generally suggest that it is desirable to have low coupling and high strength, which in this context means a low coupling/strength ratio [SMC74].

The data bindings analysis produced 77 trees corresponding to the subsystems which included a total of 4211 clusters containing 5045 routine oc-

Figure 2: Distribution of errors and error correction effort by subsystem coupling/strength ratios.

Subsystem coupling/ strength	Errors				Error correction hours			
	per KLOC		Total		per KLOC		Total	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
High	1.54	3.95	0.44	0.99	2.80	7.53	0.88	2.69
Low	0.31	1.16	0.15	0.52	0.91	4.51	0.42	2.39
Overall	1.28	3.58	0.38	0.92	2.39	7.03	0.78	2.63

currences. Recall that there were a total of 451 routines in the system — each routine was bound into 12.4 subsystems on the average (see Section 2). We calculated three different measures based on the clusters resulting from the data bindings analysis. For each routine occurrence, we calculated:

- Routine coupling/strength ratio — The coupling/ strength ratio of the first cluster to form that included the routine as a member. This metric is intended to capture the relationship of a routine to other routines in a subsystem in terms of coupling and strength.
- Routine location in subsystem's data binding tree — The depth in the tree of the first subtree (i.e., cluster) to form that included the routine as a member. More precisely, it is the depth in the tree of the root of that subtree. This metric is intended to characterize the location of a routine in a data binding tree. This location information is useful to know when data binding trees are used as an alternate form of system documentation.

For each subsystem, we calculated:

- Subsystem coupling/strength ratio — The median of the coupling/strength ratios for the clusters within the subsystem. We use a non-parametric statistic here, i.e., a median, because the coupling/strength ratios are relative measures. This metric is intended to characterize the overall coupling and strength within a subsystem.

Figure 3: Distribution of errors and error correction effort by subsystem size.

Subsystem size	Errors				Error correction hours			
	per KLOC		Total		per KLOC		Total	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Large	1.52	3.94	0.43	0.98	2.77	7.44	0.86	2.61
Small	0.35	1.22	0.17	0.58	0.98	4.96	0.49	2.71
Overall	1.28	3.58	0.38	0.92	2.39	7.03	0.78	2.63

Figure 4: Distribution of errors and error correction effort across subsystem coupling/strength ratios and subsystem size.

Subsystem coupling/ strength	Subsystem size	Errors				Error correction hours			
		per KLOC		Total		per KLOC		Total	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std
High	Large	1.66	4.12	0.46	1.02	2.99	7.71	0.92	2.66
	Small	0.45	1.41	0.21	0.66	1.11	5.31	0.56	2.93
Low	Large	0.36	1.27	0.15	0.52	0.91	4.11	0.39	2.07
	Small	0.28	1.09	0.15	0.52	0.90	4.75	0.44	2.57
Overall		1.28	3.58	0.38	0.92	2.39	7.03	0.78	2.63

4.2 Data Analysis Method

An analysis of variance model was used to characterize subsystems and routines that had either many/few errors or high/low development effort spent in error correction.

4.2.1 Independent Variables

The analysis of variance model [Sch59] considered numerous factors simultaneously: subsystem size (above/below median); subsystem coupling/strength ratio (above/below median); individual subsystem's attributes (77 levels); routine size (above/below median); routine coupling/strength ratio (split into four quartiles); routine location in subsystem's data binding tree (split into four quartiles); and two-way interactions. When defining the levels for some of the factors, non-parametric statistics (e.g., medians, quartiles) were used since the coupling/strength ratios are relative measures and the data bindings trees have different overall depths. Subsystem size and routine size are included as factors in the analysis because earlier analyses have indicated a relationship between size and software effort and error data (e.g., [Boe81] [BSP83]). For a more complete description of the factors and their levels, see [SB].

4.2.2 Dependent Variables

There were four dependent variables examined with the analysis of variance model.

1. Total errors — The total number of inspection, Trouble Report (TR), System Trouble Report (STR), and Error Summary Worksheet (ESW) errors in a routine[1]
2. Total errors per KLOC — The total number of inspection, TR, STR, and ESW errors in a routine per 1000 lines of source code
3. Error correction effort — The total amount of effort (in hours) spent correcting TR and ESW errors in a routine
4. Error correction effort per KLOC — The total amount of effort (in hours) spent correcting TR and ESW errors in a routine per 1000 lines of source code

In general, the discussion will focus on the errors per KLOC and the error correction effort per KLOC measures of the routines as opposed to the absolute numbers. This factors out possible underlying correlations between source lines and number of errors or amount of error correction effort. The statistics for all four measures are reported, however. The discussion will tend to highlight results that demonstrated a statistically significant difference, as opposed to those where there was no statistical difference.

[1]Inspections were held during the high-level and low-level design phases and after the completion of unit testing. Error Summary Worksheet (ESW) errors were recorded during the coding, unit testing, and integration testing phases. System Trouble Report (STR) errors were recorded during system testing. Trouble Report (TR) errors were reported against working, released code during and after field testing.

4.3 Characterization of High-Error and Low-Error Subsystems

In the source code portions of the system (see Section 2), there was a total of 299 distinct errors recorded from inspections, error summary worksheets (ESW's), system trouble reports (STR's), and trouble reports (TR's). Data on the effort required for error correction were available for 204 distinct errors recorded on ESW's and TR's. In the subsequent figures, all inspection, ESW, STR, and TR errors are counted equally.

In the following sections we analyze the number of errors and the error correction effort in the subsystems. The characterization of the subsystems is based on subsystem coupling/strength ratio, subsystem size, and interactions across these two factors. The results are summarized in a following section. Graphical plots of the data are presented in [SB].

4.3.1 Subsystem Coupling/Strength Ratio

Figure 2 presents the errors and error correction effort in the routines in subsystems with different coupling/strength ratios. This figure and the following analogous figures give the means and standard deviations for (i) the number of errors per 1000 lines of source code (KLOC), (ii) the number of errors, (iii) the error correction effort per KLOC, and (iv) the error correction effort in the routines. Subsystem coupling/strength ratio was not a statistically significant factor with respect to either errors per KLOC or error correction effort per KLOC ($\alpha > .05$)[2].

4.3.2 Subsystem Size

Figure 3 presents the errors and error correction effort in the routines in subsystems with different sizes. The subsystems of large size had routines that averaged 1.52 errors per KLOC, which was greater than the small subsystem average of 0.35 errors per KLOC ($\alpha < .05$).

[2]The F-test significance levels reported in this and later sections are based on the use of Type IV partial sums of squares[Sch59]. Any statistical difference discussed will at least be significant at the $\alpha < .05$ level, unless otherwise noted.

4.3.3 Interactions Across Subsystem Coupling/ Strength Ratio and Size

Figure 4 presents the errors and error correction effort in the routines in subsystems with different coupling/strength ratios and different sizes. Combining different subsystem coupling/strength ratios and different sizes resulted in a statistically significant interaction for errors per KLOC ($\alpha < .011$). Large subsystems with high coupling/strength ratios had routines that averaged 1.66 errors per KLOC, which was substantially more than the other subsystems — their combined average was 0.36 errors per KLOC. In addition, combining subsystem coupling/strength ratio and size resulted in an interaction that was almost statistically significant for error correction effort per KLOC ($\alpha < .066$). Large subsystems with high coupling/strength ratios had routines that averaged 2.99 error correction hours per KLOC — the other subsystems had a combined average of 0.97 error correction hours per KLOC.

Figure 5: Distribution of errors and error correction effort by routine coupling/strength ratios.

Routine coupling/ strength	Errors				Error correction hours			
	per KLOC		Total		per KLOC		Total	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
4_Highest	2.27	4.58	0.59	1.04	5.86	10.98	1.94	4.20
3_Higher	1.15	3.13	0.34	0.74	2.19	6.84	0.72	2.54
2_Lower	1.45	4.19	0.44	1.18	1.57	4.27	0.49	1.61
1_Lowest	0.28	1.11	0.15	0.49	0.21	1.09	0.06	0.29
Overall	1.28	3.58	0.38	0.92	2.39	7.03	0.78	2.63

4.3.4 Summary of Results

1. Large subsystems with high coupling/strength ratios had routines with the most errors per KLOC.
2. Large subsystems with high coupling/strength ratios had routines with six times as many errors per KLOC than did small subsystems with

Figure 6: Distribution of errors and error correction effort by routine size.

Routine size	Errors				Error correction hours			
	per KLOC		Total		per KLOC		Total	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Large	1.19	2.54	0.47	0.99	3.22	8.72	1.20	3.42
Small	1.39	4.55	0.26	0.80	1.36	3.81	0.26	0.71
Overall	1.28	3.58	0.38	0.92	2.39	7.03	0.78	2.63

Figure 7: Distribution of errors and error correction effort by routine location in data binding tree.

Routine tree location	Errors				Error correction hours			
	per KLOC		Total		per KLOC		Total	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
4_Root	0.88	2.82	0.30	0.77	1.30	4.82	0.37	1.59
3_Shallower	1.78	4.44	0.51	1.12	3.55	8.88	1.19	3.36
2_Deeper	0.96	2.48	0.27	0.63	2.51	7.39	0.83	2.82
1_Deepest	1.28	3.73	0.38	0.96	1.76	5.08	0.57	1.95
Overall	1.28	3.58	0.38	0.92	2.39	7.03	0.78	2.63

low coupling/strength ratios.

3. Large subsystems with high coupling/strength ratios had routines with ten times as many unit and integration test (ESW[3]) errors per KLOC than did small subsystems with low coupling/strength ratios.

4. Large subsystems with high coupling/strength ratios had routines with eight times as much error correction effort per KLOC from unit and integration test (ESW) errors than did small subsystems with low coupling/strength ratios.

[3]Errors during the coding and unit and integration testing phases were reported on error summary worksheets (ESW's).

4.4 Characterization of High-Error and Low-Error Routines

In the following sections we analyze the number of errors and the error correction effort in the routines. The characterization of the routines is based on routine coupling/strength ratio, routine size, routine location in the data binding tree, and interactions across these three factors. The results are summarized in a following section. Various graphical plots of the data are presented in [SB]. As mentioned in Section 4.3 there were 299 distinct errors, counting all inspection, ESW, STR, and TR errors equally; 204 of them had data on error correction effort.

4.4.1 Routine Coupling/Strength Ratio

Figure 5 presents the errors and error correction effort in the routines with different coupling/strength ratios. As before, this figure and the following analogous figures give the means and standard deviations for (i) the number of errors per 1000 lines of source code (KLOC), (ii) the number of errors, (iii) the error correction effort per KLOC, and (iv) the error correction effort in the routines.

The routine coupling/strength ratio statistically effected both the number of errors per KLOC and the error correction effort per KLOC in the routines ($\alpha < .0008$ and $\alpha < .002$, respectively). The routines in coupling/strength region 4_HIGHEST had the most errors per KLOC (an average of 2.27) and the highest error correction effort per KLOC (an average of 5.86 hours). The routines with coupling/strength ratios in either region 3_HIGHER or 2_LOWER had the second most errors per KLOC and the second most error correction effort per KLOC. The 3_HIGHER and 2_LOWER regions were not statistically different in either errors per KLOC or error correction effort per KLOC. Those routines in region 1_LOWEST had the fewest errors per KLOC (an average of 0.28) and the least error correction effort per KLOC (an average of 0.21 hours).[4] These results empirically support the software engineering principle of desiring low coupling and high strength.

[4] All multiple comparison results, such as the one in the previous four sentences, were conducted with Tukey's multiple comparison statistic [Sch59] [Ins82]. All of the pairwise statistical comparisons of these four categories are statistically significant at the $\alpha < .05$ level simultaneously.

4.4.2 Routine Size

Figure 6 presents the errors and error correction effort in the routines with different sizes. The routine size statistically effected the error correction effort per KLOC for the routines ($\alpha < .0001$). Routines of large size had an average of 3.22 hours error correction effort per KLOC, which was more than did those of small size (an average of 1.36 hours error correction effort per KLOC). Although small routines had slightly more errors per KLOC than did large routines, the difference was not statistically significant ($\alpha > .05$). A separate study has indicated, however, that smaller routines may be more error-prone than larger routines [BP84].

4.4.3 Routine Location in Data Binding Tree

Figure 7 presents the errors and error correction effort in the routines with different data binding tree locations. The routine location in the data binding tree statistically effected the number of errors per KLOC in the routines ($\alpha < .0001$). Routines in tree location region 3_SHALLOWER had an average of 1.78 errors per KLOC, which was more than any of the other three tree location regions.[5]

The routine location in the data binding tree also statistically effected the error correction effort per KLOC for the routines ($\alpha < .0001$). The routines in tree location region 3_SHALLOWER had the most error correction effort per KLOC (an average of 3.55 hours), those in tree location region 2_DEEPER had the second most, and those in regions 4_ROOT and 1_DEEPEST had the fewest and were not statistically different (they had a combined average of 1.53 hours). One interpretation for there being less error correction effort per KLOC in regions 4_ROOT and 1_DEEPEST may be the following: The structure of the system at the highest level (i.e., initial stages of problem decomposition) and the lowest level (e.g., formulation of abstract data types) may be better understood than the intermediate levels of system development. The effect of the less understood intermediate levels is compounded in larger subsystems, as was seen in Sections 4.3.2 and 4.3.3.

[5] Also, note that region 1_DEEPEST had more errors per KLOC than did region 4_ROOT.

4.4.4 Interactions Across Routine Coupling/ Strength Ratio, Size, and Location in Data Binding Tree

In [SB], the errors and error correction effort are given for the routines with different coupling/strength ratios and different data binding tree locations. There was a significant interaction between the routine coupling/strength ratio and data binding tree location for the number of errors per KLOC in the routines ($\alpha < .0001$). All of the three two-way interactions (routine coupling/strength ratio with routine size, routine coupling/strength ratio with routine tree location, routine size with routine tree location) statistically effected the error correction effort per KLOC for the routines (all at $\alpha < .0001$). Routines with the highest coupling/strength ratios (4_HIGHEST) and a location in the "central portion" of the data binding tree (3_SHALLOWER or 2_DEEPER) had the most error correction effort per KLOC (a combined average of 6.46 hours).

4.4.5 Summary of Results

1. The routines with the highest coupling/strength ratios had the most errors per KLOC and the most error correction effort per KLOC.
2. The routines with the lowest coupling/strength ratios had the fewest errors per KLOC and the least error correction effort per KLOC.
3. The routines with the highest coupling/strength ratios had over eight times as many errors per KLOC than did routines with the lowest coupling/strength ratios.
4. The routines with the highest coupling/strength ratios had over 27 times as much error correction effort per KLOC than did routines with the lowest coupling/strength ratios.
5. Routines in data binding tree location region 3_SHALLOWER had more errors per KLOC and more error correction effort per KLOC than did routines in the other tree regions.
6. Small routines had more unit and integration test (ESW) errors per KLOC than did large routines.
7. Large routines had more error correction effort per KLOC than did small routines when either all errors or just unit and integration test (ESW) errors were considered.

8. Large routines tended to have a higher average amount of correction effort per error for unit and integration test (ESW) errors than did small routines.

4.5 Data Bindings for System Documentation and Evaluation

The following observations resulted from dialogue with project personnel regarding the data binding trees generated.

1. The data binding clusterings were able to detect major system data structures.
2. The data binding clusterings seemed to provide a different view of the system than that provided by the system documentation, which included textual documents and a calling hierarchy.
3. Analyzing the clusters of data bindings provided insights to the development and maintenance team.

5 Interpretations and Conclusions

In this study, we have merged two goals:

- To collect and analyze data from an ongoing software project without negatively impacting the software developers; and
- To investigate hierarchical system descriptions based on the software engineering principles of coupling and strength (or cohesion) and their relationship to software errors and error correction effort.

This study highlights and empirically supports several software engineering principles. The interpretations span several areas: coupling/strength, system structure, and size.

Coupling/Strength

Low coupling/strength ratios are desirable (e.g., high strength and low coupling).

- Routines with the lowest coupling/strength ratios had 8.1 times fewer errors per KLOC than routines with the highest coupling/strength ratios and errors were 27.9 times less costly to fix.
- Large subsystems with high coupling/strength ratios had routines with 4.6 times more errors per KLOC than did the other categories of subsystems.

System Structure Hierarchy: Data Bindings View

The structure of the system at the highest level, i.e., initial stages of problem decomposition, and lowest level, e.g., formulation of abstract data types, appear to be better understood than the intermediate levels of abstraction and specification.

- The errors were 50% less costly to fix in routines at the shallowest and deepest levels of the data bindings view of the system structure hierarchy than at the middle levels, and there were 21% fewer errors per KLOC.

Size

Subsystem size seems to be at least as important, if not more important, than routine size. Hence, maybe the software community has been worrying about the wrong issue.

- Smaller subsystems had routines with 4.3 times fewer errors per KLOC than did larger subsystems.
- Smaller routines had a slightly higher average of errors per KLOC than did larger routines, although the difference was not statistically significant. When just unit and integration test errors are considered, however, smaller routines had significantly more errors per KLOC than did larger routines. Overall, errors in smaller routines were 2.4 times less expensive to fix.

6 Acknowledgement

The author is very grateful to several persons on the selected software project for their assistance and support in this research. Their names cannot be mentioned because of a non-disclosure agreement. The author appreciates the assistance of D. Hutchens in developing the data bindings analysis tools, S. Wilkin in collecting the data, and V. Basili in originating the project.

References

[BE82] L.A. Belady and C.J. Evangelisti. System partitioning and its measure. *Journal of Systems and Software*, 2(1):23–29, February 1982.

[Boe81] B. W. Boehm. *Software Engineering Economics.* Prentice-Hall, Englewood Cliffs, NJ, 1981.

[BP84] V. R. Basili and B. T. Perricone. Software errors and complexity: An empirical investigation. *Communications of the ACM*, 27(1):42–52, Jan. 1984.

[BSP83] V. R. Basili, R. W. Selby, and T. Y. Phillips. Metric analysis and data validation across fortran projects. *IEEE Transactions on Software Engineering*, SE-9(6):652–663, Nov. 1983.

[BT75] V. R. Basili and A. J. Turner. Iterative enhancement: a practical technique for software development. *IEEE Transactions on Software Engineering*, SE-1(4), Dec. 1975.

[BW84] V. R. Basili and D. M. Weiss. A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering*, SE-10(6):728–738, Nov. 1984.

[Eme84] T. Emerson. A discriminant metric for module cohesion. In *Proc. Seventh Intl. Conf. Software Engr.*, pages 294–303, Orlando, FL, 1984.

[Eve80] B. S. Everitt. *Cluster Analysis, 2nd ed.* Heineman Educational Books Ltd., London, 1980.

[HB85] D. H. Hutchens and V. R. Basili. System structure analysis: Clustering with data bindings. *IEEE Trans. Soft. Engr.*, SE-11(8), Aug. 1985.

[HK81] S. Henry and D. Kafura. Software quality metrics based on interconnectivity. *Journal of Systems and Software*, 2(2):121–131, 1981.

[Ins82] SAS Institute. Statistical analysis system (sas) user's guide. Technical report, SAS Institute Inc., Box 8000, Cary, NC, 27511, 1982.

[JS71] N. Jardine and R. Sibson. *Mathematical Taxonomy.* John Wiley and Sons, New York, 1971.

[SB] Richard W. Selby and Victor R. Basili. Analyzing error-prone system coupling and cohesion. Technical report, Dept. of Computer Science, University of California, Irvine. (in preparation).

[Sch59] H. Scheffe. *The Analysis of Variance.* John Wiley & Sons, New York, 1959.

[SMC74] W. P. Stevens, G. J. Myers, and L. L. Constantine. Structural design. *IBM Systems Journal*, 13(2):115–139, 1974.