Modeling the Relationship Between Source Code Complexity and Maintenance Difficulty

David L. Lanning, IBM Corporation

Taghi M. Khoshgoftaar, Florida Atlantic University

Canonical correlation analysis can be a useful exploratory tool for software engineers who want to understand relationships that are not directly observable.

Increasingly, software suppliers recognize that software development process capability is a key source of competitive advantage.[1] Competition forces suppliers to improve processes to meet the conflicting demands of higher quality, lower costs, and compressed schedules. Moreover, software purchasers are beginning to require certification that suppliers are applying development processes capable of delivering products within quality, cost, and schedule constraints. Measures of product complexity and of process activity provide for quantitative analyses of existing processes. Such analyses lay the foundation for continuous process improvement.

Software products yield measures related to their complexity. For example, a source file will have values for the Halstead and McCabe measures. Melton et al. noted that these *product measures* — they called them *document measures* — are distinct from the more elusive *psychological complexity measures* quantifying notions such as understandability.[2] Zage and Zage applied an aggregate of several product measures to isolate fault-prone program modules in a large industrial software product.[3] The predictive quality of the aggregate measure was superior to that of any of the constituent product measures taken alone. This result strongly supports the assertion that software complexity has more than one component. Other results also support this assertion.[4] While it is unlikely that a single product measure can serve as an adequate measure of complexity, a set of product measures can serve as indicators of complexity.

Software processes yield measures that quantify the activity they produce. For example, testing will encounter failures, troubleshooting will isolate faults, and fault correction will create source code changes. The counts of failures, faults, and changes are *process measures*. Schneidewind noted that *software quality measures*, which quantify notions such as maintainability, are distinct from process measures.[5] Several process measures may indicate a single aspect of quality. For example, the number of additions and the number of deletions that a programming team makes to a source file to remove faults indicate the difficulty the team experienced in maintaining the file. While a single process measure most likely cannot serve as an adequate measure of maintenance difficulty, a set of process measures can serve as indicators of difficulty.

Curtis noted that a measure of psychological complexity must consider aspects of

0018-9162/94/ $4.00 © 1994 IEEE

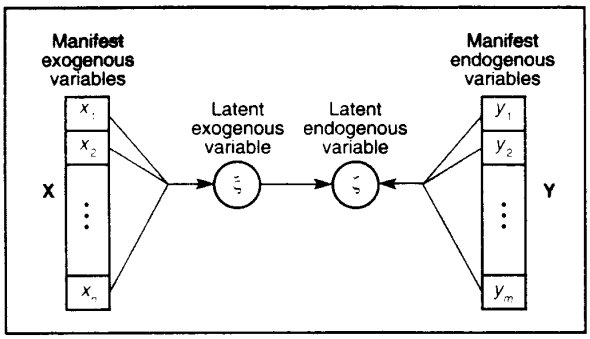


Figure 1. The first canonical correlation, with paths representing the inner and outer relations.

both a product and the people interacting with the product; that is, the complexity of, say, a source file is related to both the nature of the file and the difficulty experienced by those working with the file.[6] Complexity and difficulty are not directly observable but are indicated by observable phenomena. This suggests a model involving the relationship between two sets of measures: a set indicating complexity and a set indicating difficulty. The study of such a relationship is known as *canonical correlation analysis*.[7]

In this article, we apply canonical correlation analysis to investigate the relationship between source code complexity and maintenance difficulty. This approach acknowledges that complexity and difficulty are abstract concepts that are not directly observable. Rather, these concepts are indirectly observable through measures serving as indicators of software product and process characteristics.

Canonical correlation analysis

Canonical correlation analysis generalizes linear regression analysis.[8] It also restricts the soft-modeling methodology introduced by Wold.[9] Since the more general methodology offers modeling extensions useful to software engineers, we present the soft-modeling methodology and the restrictions of this methodology that define canonical correlation analysis.

A soft model involves both *manifest* and *latent* variables — respectively, variables that are directly observed and those that are indirectly observed. A weighted aggregate of manifest variables quantifies each latent variable. For example, a social model might quantify "population change" in terms of the observable variables "natality," "mortality," and "migration." The weighted aggregates quantifying the latent variables give the *outer relations* of the model. *Inner relations* specify a causal relationship among the latent variables. Latent variables explained by an inner relation are *endogenous*; those that are not so explained are *exogenous*. Correspondingly, the manifest variables are either endogenous or exogenous. Canonical correlation analysis applies a soft model restricted to one latent exogenous variable, one latent endogenous variable, one inner relation, and linear relationships among variables.

A *path diagram* gives a visual soft-model specification. In a path diagram, manifest variables appear as blocks, latent variables appear as circles, and relations appear as directed paths. The path diagram in Figure 1 gives the model for the first canonical correlation. In this model, ξ and ζ are latent variables indicated by blocks of manifest variables — respectively, $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ and $\mathbf{y} = [y_1, y_2, \ldots, y_m]$. The path from ξ to ζ gives the inner relation. The direction of this path specifies that ξ explains ζ. Thus, ζ is endogenous. A weight associated with the inner relation is a model parameter giving the influence of ξ on ζ. Having no paths in from other latent variables, ξ is exogenous; its cause is outside of the model. The paths from $\mathbf{x}$ to ξ and from $\mathbf{y}$ to ζ represent the outer relations. Weights associated with each of these paths are model parameters giving the coefficient of each manifest variable in the aggregate quantifying the related latent variable. For the first canonical correlation, the weights defining the outer relations are evaluated such that the latent variables are maximally correlated. The resulting correlation is the *canonical correlation* between the two latent variables.

As explained above, Figure 1 gives the model for the first canonical correlation. Where there are n manifest exogenous variables and m manifest endogenous variables, a canonical correlation analysis yields $d = \min(n, m)$ dimensions of canonical correlation. Figure 2 shows this graphically. Superscripts identify the dimensions of the latent variables. For example, ξ is d-dimensional, having dimensions $\xi^{(1)}, \xi^{(2)}, \ldots, \xi^{(d)}$. Similarly, ζ is d-dimensional, having dimensions $\zeta^{(1)}, \zeta^{(2)}, \ldots, \zeta^{(d)}$. The directed paths from $\xi^{(k)}$ to $\zeta^{(k)}$, $1 \le k \le d$, give a d-dimensional inner relation. The weights defining the outer relations are evaluated such that the d dimensions of endogenous latent variables are mutually orthogonal, the d dimensions of exogenous latent variables are mutually orthogonal, and the pairs of latent variables at each dimension are maximally correlated. The correlations between each of these pairs give the d canonical correlations of the canonical model.

A parameter estimation technique yields the weights of the inner and outer relations of the canonical model. There are two approaches to estimating these model parameters: One generalizes linear regression estimation techniques[8]; the other, the partial least-squares estimation technique, treats a canonical model as a restricted soft model.[9] Details of these techniques are beyond the scope of this article. However, it is important to note that both techniques have tractable implementations.

Canonical correlation analysis is helpful in understanding the relationships between two sets of variables, the manifest exogenous set and the manifest endogenous set. The corresponding latent variables capture the structure between these sets. Since latent variables are not directly observable, they are best interpreted in terms of the manifest variables most closely related to them. Each manifest variable reflected through the outer relation defining a latent variable will have a correlation, or *loading*, with this latent variable. The pattern of loadings for a latent variable can suggest its nature.

While there are d dimensions of canonical correlation, it is often reasonable to interpret just a few. First, there is no reason to interpret relationships that are below a reasonable statistical significance. Second, dimensions having low canonical correlations can be ignored. Third, each dimension accounts for some percentage of the total variance explained by the analysis. Those accounting for a small percentage of this variance can be ignored.

Canonical correlation applied to software engineering measures

With the general background provided in the previous section, we can now discuss the specific model of interest in this article. We apply canonical correlation analysis to investigate the relationship between source code complexity and maintenance difficulty during the system test phase for a commercial real-time product (RTP). RTP provides a stable interface for software products written to a varying hardware base. The developers implemented RTP in assembly language to satisfy space and time constraints. The RTP source code consists of 222,740 lines divided among 152 files. The 152 RTP files provide a cross section of data for the canonical model.

We hypothesize that the complexity of the source code defining RTP causes the difficulty experienced by those maintaining this code during system testing. We do not model a cause of source code complexity, and we have no direct measures of either complexity or difficulty. Thus, we model complexity as a latent exogenous variable having some affect upon difficulty, which we model as a latent endogenous variable.

Before defining the measures we use as indicators of complexity and difficulty, we note that other measures could be selected for these purposes. Our goal here is to apply canonical correlation analysis to investigate the relationship between source code complexity and maintenance difficulty, not to justify the use of any particular selection of measures. For details regarding the selection and validation of software engineering measures, see Schneidewind[10] and Fenton.[11]

We consider the following product measures as indicators of source code complexity:

(1) η_1 is the number of unique operators.
(2) η_2 is the number of unique operands.
(3) N_1 is the total number of operators.
(4) N_2 is the total number of operands.
(5) XQT is the number of executable statements.
(6) $V(G)$, McCabe's cyclomatic number, is given by

$$V(G) = e - n + 2$$

where e is the number of edges and n is the number of nodes in the control flow graph.

(7) *Knots* is the number of times the control flow crosses itself. Programs constructed exclusively from the basic structures for sequence, selection, and iteration typical of high-level languages will have no knots. These structures have both a single entry and a single exit through which control always flows. Sequence, selection, and iteration are encapsulated between these points. Using a GOTO statement within such a structure to transfer control to an instruction within another structure will produce at least one knot.

In assembly language, structures for selection and iteration are constructed using test and jump instructions. While these constructions are often clearer when they emulate the high-level-language structures, the smallest or fastest coding will often require control-flow structures with multiple entry and exit points. Since size and speed are typically important in assembly language implementations, knots are common in assembly language code.

(8) $FFOT$ is the number of calls out.
(9) $FFIN$ is the number of calls in.
(10) $AICC$, the average information content classification, is given by

$$AICC = -\sum_{i=1}^{\eta_1} \frac{f_i}{N_1} \log_2 \frac{f_i}{N_1}$$

where η_1 and N_1 are defined above, and f_i is the number of occurrences of the ith operator.[12] This measure applies information theory to software complexity.

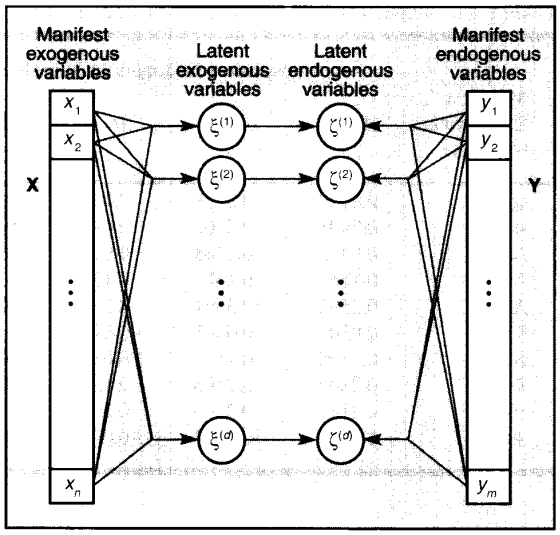


Figure 2. The general canonical model, with *n* manifest exogenous variables and *m* manifest endogenous variables.

These 10 product measures, evaluated for each of the 152 RTP files *on entry to* the system test phase, are the manifest exogenous variables of the canonical model.

We consider the following process measures as indicators of maintenance difficulty:

(1) A is the number of noncomment source lines added.
(2) D is the number of noncomment source lines deleted.
(3) M is the number of noncomment source lines moved.
(4) F is the number of program faults. A system test case failure generates a problem-tracking report. Troubleshooting resolves the problem-tracking report by isolating the cause of the failure to defective statements in one or more files. Correction of these defective statements requires source code changes. All changes to a file associated with a single problem-tracking report are changes required to remove one fault.
(5) DC is the number of design changes. These changes modify the design either to correct design defects or to add function to the product. Design-change reports track changes to the approved design. All changes to a

Table 1. Correlations between the manifest variables of distinct blocks.

Manifest Exogenous Variables	Manifest Endogenous Variables: *A*	*D*	*M*	*F*	*DC*
η_1	0.2127	0.2133	0.1718	0.3549	0.2158
η_2	0.7991	0.8003	0.6956	0.5570	0.7686
N_1	0.1453	0.1445	0.0962	0.2641	0.1304
N_2	0.1682	0.1674	0.1251	0.1746	0.1368
XQT	0.3596	0.3594	0.2728	0.4614	0.3511
V(G)	0.0305	0.0305	–0.0166	0.3355	0.0387
Knots	0.1508	0.1510	0.0663	0.1671	0.2205
FFOT	0.2304	0.2396	0.1371	0.3112	0.2403
FFIN	0.2145	0.2210	0.2028	0.2528	0.2146
AICC	0.0160	0.0133	–0.0036	0.2718	0.0463

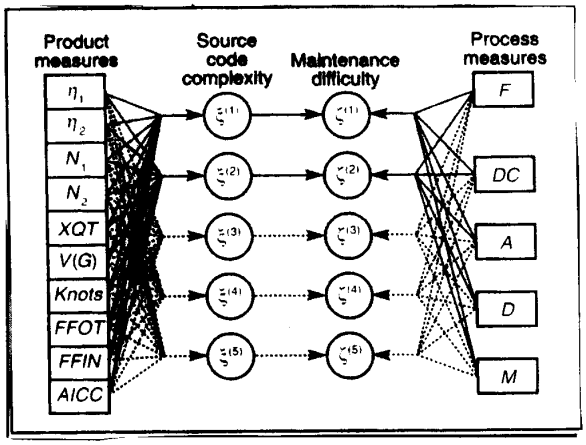


Figure 3. A model of source code complexity and maintenance difficulty, with the first two dimensions differentiated by solid lines.

file associated with a single design-change report are changes required to implement one design change.

These five process measures, evaluated for each of the 152 RTP files *on exit from* the system test phase, are the manifest endogenous variables of the canonical model. Table 1 gives the correlations between the source code and the process measures.

Figure 3 gives the path diagram for the canonical model. This figure shows the 10 product and five process measures that indicate, respectively, source code complexity and maintenance difficulty. Both source code complexity and maintenance difficulty have five dimensions. The directed paths from each dimension of source code complexity to the corresponding dimension of maintenance difficulty represent the causal influence of source code complexity on maintenance difficulty.

Table 2 gives the canonical correlations for the canonical model. We select the first two dimensions for interpretation; thus, Figure 3 differentiates these dimensions with solid lines. Several considerations lead to this selection. First, dimensions greater than two are not statistically significant. This narrows the selection to the first two dimensions. Second, both of these dimensions have strong canonical correlations: 88.9 percent for the first and 54.4 percent for the second. And finally, each of the first two dimensions accounts for a reasonable proportion of the total explained variance: 84.8 percent for the first and 9.4 percent for the second. By retaining the first two dimensions, we retain about 94 percent of the overall variance explained by the model.

In interpreting the latent variables forming the first two dimensions of canonical correlation, we consider their loadings on the manifest variables. Table 3 gives these loadings, along with the weights that form the inner relations. For example,

$$\zeta^{(1)} = -1.0442A + 1.8112D - 0.1285M + 0.1902F + 0.2495DC$$

evaluates the first dimension of maintenance difficulty, and this latent variable has a loading of 97.23 percent on *A*. As Table 3 shows, the first dimension of source code complexity loads strongly on η_2 and moderately on *XQT*, both measures of size. The first dimension of maintenance difficulty loads strongly on *A*, *D*, *M*, and *DC*. It also loads moderately on *F*. Thus, the first dimension of canonical correlation relates source code size to a general notion of change. The correlation at this dimension is excellent at 89 percent. Note that this correlation between the latent variables exceeds that of any pair of manifest variables given in Table 1, the closest simple correlation in this table being about 80 percent between η_2 and *D*.

The second dimension of source code complexity loads moderately on η_1, N_1, *XQT*, *V(G)*, and *AICC*. Note that η_1 and N_1 are used to derive *AICC*. These three loadings are directly related to information content. The second dimension of maintenance difficulty loads strongly on *F* alone. This dimension of canonical correlation relates source code information content and control flow to program faults. The correlation at this dimension is about 54 percent.

Differences in loadings across dimensions are also important in interpreting the latent variables. The first dimension of source code complexity has a loading of 92.61 percent with η_2. In the second dimension, this loading is 7.45 percent, about as close to 0 percent as 92.61 percent is to 100 percent. Thus, η_2 is useful in distinguishing the two dimensions of source code complexity. While their loading differences are not as extreme, similar observations hold for *V(G)* and *AICC*. However, both dimensions load about equally on *XQT*; thus, this measure does not help in distinguishing the two dimensions of source code complexity.

The two dimensions of maintenance difficulty have large differences in loadings on A, D, M, and DC. The difference in loading on F is not as extreme. However, the loading on F is the lowest loading of the first dimension and the highest of the second. While F contributes to both dimensions of maintenance difficulty, its relation to the other difficulty indicators differentiates its contribution across these dimensions. Its appearance in the first dimension of difficulty suggests that both the number of program faults and the number of modifications required to remove faults varied directly with program size. Its appearance in the second dimension of difficulty suggests that the number of program faults varied directly with control flow and information content, but the number of modifications required to remove the faults was independent of these program attributes.

Note that the interaction between DC, A, D, and M is likely to be more pronounced than the interaction between F, A, D, and M. That is, design changes either introduce functional enhancements or correct design defects, so it is reasonable to expect that design changes will typically result in more file changes than fault removals will. It is also reasonable to expect that design changes will introduce faults. If this explains the loadings of F and DC on the first dimension of maintenance difficulty, then the model suggests two subsets of product measures, one related to design-change activity that results in faults and another related directly to faults. However, the model does not distinguish, at the abstract level, between F and DC. To make this distinction requires a model of the relationships between three sets of variables: a set of product measures indicating source code complexity, a set of process measures indicating fault activity, and a set of process measures indicating design-change activity. This suggests that soft models of greater generality than canonical correlation will provide more insight into the relationships among software engineering measures.

Table 2. Canonical correlations.

Canonical Dimension	Canonical Correlation	Statistically Significant ($p < 0.05$)	Proportion of Variance	Cumulative Proportion
1	0.889	yes	0.848	0.848
2	0.544	yes	0.094	0.942
3	0.337	no	0.029	0.971
4	0.289	no	0.021	0.992
5	0.187	no	0.008	1.000

We emphasize two aspects of our presentation. First, we restrict our findings to the development effort that we studied. We do not imply that either the weights or the loadings of the relations generalize to all software development efforts. Such generalization is untenable, since the model omitted many important influences on maintenance difficulty, for example, schedule pressure, product domain, and level of engineering expertise. We demonstrated canonical correlation anal-

Table 3. Weights and loadings of the canonical model.

		Dimension 1		Dimension 2	
		Weights	Loadings	Weights	Loadings
Source Code Complexity (ξ)					
η_1	(x_1)	−0.4744	0.2964	−0.9999	0.4530
η_2	(x_2)	0.3500	0.9261	−1.0338	0.0745
N_1	(x_3)	−3.6774	0.2028	−1.0062	0.4070
N_2	(x_4)	1.6920	0.2011	−1.3722	0.2018
XQT	(x_5)	2.5449	0.4677	4.0669	0.5108
V(G)	(x_6)	−0.2069	0.1114	−0.2655	0.6935
Knots	(x_7)	−0.0244	0.2186	−0.8136	0.0711
FFOT	(x_8)	0.1126	0.3316	0.0396	0.3097
FFIN	(x_9)	−0.0004	0.2834	−0.1110	0.1795
AICC	(x_{10})	0.2865	0.0798	0.7208	0.5392
Maintenance Difficulty (ζ)					
A	(y_1)	−1.0442	0.9723	3.6177	−0.1609
D	(y_2)	1.8112	0.9743	−3.1652	−0.1664
M	(y_3)	−0.1285	0.8215	−0.4030	−0.2190
F	(y_4)	0.1902	0.6349	1.1657	0.7278
DC	(y_5)	0.2495	0.9437	−0.8930	−0.1327

ysis as a useful exploratory tool for software engineers interested in understanding influences affecting past development efforts. These influences could also affect current development efforts. However, much work remains to specify subsets of indicators and development efforts for which the technique becomes useful as a predictive tool.

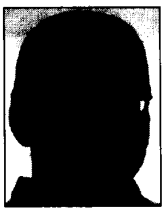
Second, we explained canonical correlation analysis as a restricted form of soft modeling. We chose this approach not only because the terminology and graphical devices of soft modeling allow straightforward high-level explanations, but also because we are interested in the general method. The general method allows models involving many latent variables having interdependencies. It is intended for modeling complex interdisciplinary systems having many variables and little established theory. Further, it incorporates parameter estimation techniques relying on no distributional assumptions. Our future research will focus on developing general soft models of the software development process for both exploratory analysis and prediction of future performance. ■

Acknowledgment

We thank Timothy Woodcock for completing the assembly language metric analyzer and collecting the data used in this study.

References

1. D.H. Kitson and S.M. Masters, "An Analysis of SEI Software Process Assessment Results: 1987-1991," *Proc. 15th Int'l Conf. Software Eng.*, IEEE CS Press, Los Alamitos, Calif., Order No. 3700, 1993, pp. 68-77.
2. A. Melton et al., "A Mathematical Perspective for Software Measure Research," *Software Eng. J.*, Vol. 5, No. 5, Sept. 1990, pp. 246-254.
3. W.M. Zage and D.M. Zage, "Evaluating Design Metrics on Large-Scale Software," *IEEE Software*, Vol. 10, No. 4, July 1993, pp. 75-80.
4. T.M. Khoshgoftaar and J.C. Munson, "Predicting Software Development Errors Using Complexity Metrics," *IEEE J. Selected Areas Comm.*, Vol. 8, No. 2, Feb. 1990, pp. 253-261.
5. N.F. Schneidewind, "Report on the IEEE Standard for a Software Quality Metrics Methodology," *Proc. 1993 Conf. Software Maintenance*, IEEE CS Press, Los Alamitos, Calif., Order No. 4600, 1993, pp. 104-106.
6. B. Curtis, "Conceptual Issues in Software Metrics," *Proc. 19th Hawaii Int'l Conf. Systems Sciences*, Jan. 1986, pp. 154-157.
7. R.A. Johnson and D.W. Wichern, *Applied Multivariate Statistical Analysis*, Prentice Hall, Englewood Cliffs, N.J., 1992.
8. H. Hotelling, "Relations Between Two Sets of Variables," *Biometrica*, Vol. 28, 1936, pp. 321-377.
9. H. Wold, *Systems Under Indirect Observation*, Vol. 2, Chap. "Soft Modeling: The Basic Design and Some Extensions," North-Holland Publishing, Amsterdam, 1982, pp. 1-54.
10. N.F. Schneidewind, "Methodology for Validating Software Metrics," *IEEE Trans. Software Eng.*, Vol. 18, No. 5, May 1992, pp. 410-421.
11. N.E. Fenton, *Software Metrics: A Rigorous Approach*, Chapman and Hall, New York, 1991.
12. W. Harrison, "An Entropy-Based Measure of Software Complexity," *IEEE Trans. Software Eng.*, Vol. 18, No. 11, Nov. 1992, pp. 1,025-1,029.



David L. Lanning is a PhD candidate in the Department of Computer Science and Engineering at Florida Atlantic University, Boca Raton, Florida. His research interests include software engineering measurements, software quality engineering, and application of neural networks in software engineering. He joined IBM in 1988 and is now a software engineer in the company's Personal Systems Division in Boca Raton. He received his BS in computer science from Grand Valley State University, Allendale, Michigan, in 1985 and his MS in computer science from Ohio State University, Columbus, Ohio, in 1988.

Taghi M. Khoshgoftaar is a guest editor for this special issue. His photo and biography appear on page 15.

Lanning can be contacted at IBM Corp., 1000 NW 51st St., Boca Raton, FL 33432; e-mail dlanning@vnet.ibm.com. Khoshgoftaar's address is Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431; e-mail taghi@cse.fau.edu.