Design Structural Stability Metrics and Post-Release Defect Density: An Empirical Study

Mahmoud O. Elish
Info. & Computer Science Department
King Fahd Univ. of Petroleum & Minerals
Dhahran 31261, Saudi Arabia
elish@kfupm.edu.sa

David Rine
Department of Computer Science
George Mason University
Fairfax, VA 22030, USA
drine@gmu.edu

Abstract

This paper empirically explores the correlations between a suite of structural stability metrics for object-oriented designs and post-release defect density. The investigated stability metrics measure the extent to which the structure of a design is preserved throughout the evolution of the software from one release to the next. As a case study, thirteen successive releases of Apache Ant were analyzed. The results indicate that some of the stability metrics are significantly correlated with post-release defect density. It was possible to construct statistically significant regression models to estimate post-release defect density from subsets of these metrics. The results reveal the practical significance and usefulness of some of the investigated stability metrics as early indicators of one of the important software quality outcomes, which is post-release defect density.

Keywords *– Structural stability, software metrics, object-oriented designs.*

1. Introduction

Considerable resources and effort are invested and spent in deriving a high quality software design. Such design, presumably correct, represents a *valuable asset*. Clearly, software maintenance is inevitable if software systems need to remain useful in their environments. Changes are necessary to continue increasing or sustaining the value of software as it evolves over time. A good design shall be flexible enough to accommodate these continuous changes while leaving its original structure intact.

As changes are made to a software design its structure and/or behavior may be affected. Accordingly, there are two aspects of design stability: *structural stability* that is concerned with the stability of design structure (form), and *performance stability* that is concerned with the stability of design behavior (function). This paper is focused on structural stability of object-oriented designs.

The structural stability of an object-oriented design refers to the extent to which the structure of the design is preserved throughout the evolution of the software from one release to the next [11, 12]. Structural stability of a design is a sign of its capability to expand while preserving its initially correct structure. Object-oriented design structure, in this context, is defined by the design classes and the relationships between them as they are the two most fundamental building blocks.

Measuring design structural stability is important. According to DeMarco's principle [9] "you cannot control what you cannot measure." However, exploring the practical significance and usefulness of such measures as early indicators of software quality outcomes of interest to software developers is also important. This paper empirically explores the correlations between a suite of structural stability metrics for object-oriented designs and post-release defect density.

The rest of this paper is organized as follows. Section 2 describes a suite of metrics for object-oriented design structural stability. Section 3 empirically explores the correlations between the metrics suite and post-release defect density. Section 4 reviews related work. Section 5 concludes the paper.

2. Structural stability metrics

A suite of structural stability metrics for object-oriented designs [12] is described in this section. The metrics suite evaluates the extent to which the structure of an object-oriented design remained stable from one

In Proceedings of the 30th Annual IEEE International Computer Software and Applications Conference (COMPSAC'06), Chicago, USA, Sep. 2006.

software release to the next throughout the software evolution.

The stability of a design structure is a two-dimensional concept. The first dimension considers the size of design structural modifications from one software release to the next to determine *how much* of the original design structure remained stable. The second dimension considers the period of time from one release to the next to determine *how long* the structure remained stable. In order to measure both dimensions of design structural stability, the metrics suite includes size-based metrics and time-based metrics.

2.1. Size-based metrics

Size-based metrics of design structural stability calculate the size of the changes made to a design structure as well as the size of the design structure part that remained stable from one software release to the next. These metrics are measured by comparing every two successive software releases throughout the software evolution.

Since the structure of an object-oriented design is defined by the design classes and the relationships between them, the extent to which these two fundamental building blocks (classes and relationships) are preserved from one release to the next need to be measured by the size-based stability metrics. Therefore, two categories of size-based metrics of design structural stability are considered: class-based metrics and relationship-based metrics.

2.1.1 Class-based metrics. Class-based metrics of design structural stability calculate the extent to which design classes are preserved from one software release to the next. As software undergoes changes, existing classes may be modified or deleted, and new classes may be added.

Four class-based stability metrics are investigated in this paper: *NSC*, *NAC*, *NDC*, and *NMC*. The *NSC* metric counts the number of classes that remained stable (i.e. were not added, deleted, or modified) between any two successive software releases. *NAC*, *NDC*, and *NMC* metrics count the number of classes that were added, deleted, and modified respectively between any two successive software releases.

The three-letter acronyms of class-based stability metrics are interpreted as follows. The first letter is *N*, which stands for "number of." The second letter represents the kind of change, i.e. *S* for stable, *A* for added, *D* for deleted, and *M* for modified. The last letter is *C*, which stands for 'classes'.

2.1.2 Relationship-based metrics. Relationship-based metrics of design structural stability calculate the extent to which relationships between classes in a design are preserved from one software release to the next. As software undergoes changes, existing class relationships may be deleted and new relationships may be added. These include relationships of all kinds, i.e. generalization, aggregation, dependency, and association. A *generalization relationship* is a relationship between a more general class (superclass) and more specific class (subclass). An *aggregation relationship* exists between two classes if one is part of the other, i.e. if one is the type of an attribute of the other. A *dependency relationship* exists between two classes if one is the return type of a method of the other or the type of a parameter of a method of the other. An *association relationship* exists between two classes if one invokes one or more methods of the other and/or references one or more attributes of the other.

Twelve relationship-based stability metrics are investigated in this paper; three for each kind of relationships. *NSGR*, *NAGR*, and *NDGR* metrics count the number generalization relationships that were stable (neither added nor deleted), added, and deleted respectively between any two successive software releases. *NSAR*, *NAAR*, and *NDAR* metrics count the number aggregation relationships that were stable, added, and deleted respectively between any two successive software releases. *NSDR*, *NADR*, and *NDDR* metrics count the number dependency relationships that were stable, added, and deleted respectively between any two successive software releases. Finally, *NSSR*, *NASR*, and *NDSR* metrics count the number association relationships that were stable, added, and deleted respectively between any two successive software releases.

The four-letter acronyms of relationship-based stability metrics are interpreted as follows. The first letter is *N*, which stands for "number of." The second letter represents the kind of change, i.e. *S* for stable, *A* for added, and *D* for deleted. The last two letters represent the kind of relationship, i.e. *GR* for generalization relationships, *AR* for aggregation relationships, *DR* for dependency relationships, and *SR* for association relationships.

2.2. Time-based metric

The size-based metrics defined in the previous section capture one dimension of design structural stability, i.e. *how much* of the design structure (classes and relationships) remained stable between two successive software releases. Another important

dimension is *how long* the design structure remained stable, which is measured by time-based metrics. Structural stability of designs cannot be adequately assessed unless these two dimensions are considered. If two designs have relatively the same size of structural modifications since their first release, but one is 3 years old and the other is 10 years old, the older would intuitively be considered more stable.

This metrics suite includes one time-based stability metric, which is *time between releases* (*TBR*) to quantify the *how long* dimension of design structural stability. The *TBR* metric measures the number of days between two successive releases.

3. The empirical study

This section describes the conducted empirical study that explored the correlations between the investigated stability metrics and post-release defect density. The objective was to investigate whether or not there exist significant correlations between measures of the investigated stability metrics from release *i* to the next release *i+1* and the post-release defect density of release *i+1*.

3.1. Case study system

The case study system is Apache Ant [1], which is an open source Java-based build tool. Apache Ant software project is more than four years old. Its most recent release (1.6.2) has more than 200K lines of code, and more than 1200 classes. In this study, 13 successive releases of Apache Ant were analyzed, from its first release (1.1) to its most recent release (1.6.2). Precisely, the releases are 1.1, 1.2, 1.3, 1.4, 1.4.1, 1.5, 1.5.1, 1.5.2, 1.5.3, 1.5.4, 1.6, 1.6.1, and 1.6.2.

3.2. Dependant variable

The dependant (response) variable is the post-release defect density (*PRDD*). A post-release defect is a user reported problem in a software release, which needs to be fixed. *PRDD* is defined as the total number of user reported defects per thousand lines of code for a delivered software release. It is one of the most important and common measures of the quality of the released software version from both developers and users points of view. High defect density is likely to cause low user satisfaction.

3.3. Independent variables

The independent variables are the 17 investigated structural stability metrics for object-oriented designs: 4 class-based metrics (*NSC, NAC, NDC, NMC*); 12 relationship-based metrics (*NSGR, NAGR, NDGR, NSAR, NAAR, NDAR, NSDR, NADR, NDDR, NSSR, NASR, NDSR*); and one time-based metric (*TBR*).

3.4. Hypotheses

The hypotheses are that there are significant correlations between measures of each of the 17 stability metrics from release *i* to release *i+1* (following release) and the *PRDD* of release *i+1*.

3.5. Data collection

The independent variables were measured as follows. The *ExamDiff Pro* file comparison tool [2] was used to calculate the class-based stability metrics from each release to the next by comparing classes between releases. Comment and blank lines were excluded in class comparison. A prototype metrics tool that has been developed as part of this research was used to calculate the relationship-based stability metrics from each release to the next by comparing relationships between releases. The *TBR* metric was simply measured between any two successive releases using their release dates.

The dependant variable *PRDD* of a release was measured by searching the Apache Ant's bug database for a list of post-release defects. The number of defects is then divided by the release's KLOC that was measured with the *JStyle* metrics tool [3].

3.6. Results

The results of the empirical study are reported and analyzed in this section.

3.6.1 Descriptive statistics. Table 1 provides descriptive statistics for the measures of the 17 stability metrics that were collected from the releases of Apache Ant. In general, class relationships were more stable than classes over the releases of Apache Ant. Deletions of classes and relationships were relatively low compared to additions and modifications. Measures of the *TBR* metric vary from 37 days to 272 days with an average of ~122 days between releases.

In addition, Table 1 provides descriptive statistics for measures of *PRDD* from the releases of Apache

Ant. *PRDD* varies from 0.54 defects/KLOC to 4.56 defects/KLOC with an average of 1.78 defects/KLOC.

Table 1. Descriptive statistics

Metric	Minimum	Maximum	Mean	Std. Dev.
NSC	23	920	459.92	340.1
NAC	0	328	109.58	125.32
NDC	0	229	39.42	85.14
NMC	8	466	122.17	142.47
NSGR	52	790	452.58	250.18
NAGR	2	312	82.58	105.21
NDGR	0	100	19.25	30.09
NSAR	34	387	235.92	113.23
NAAR	0	115	40.42	47.01
NDAR	0	61	9.58	18.97
NSDR	318	1239	809.67	278.75
NADR	0	324	98.58	115.74
NDDR	0	136	19.08	40.36
NSSR	214	3128	1762.58	953.26
NASR	2	984	316.25	379.18
NDSR	0	421	67.75	123.93
TBR	37	272	121.58	66.79
PRDD	0.54	4.56	1.78	1.41

3.6.2 Univariate analysis. The univariate regression analysis aims to determine if each individual stability metric is significantly correlated with *PRDD*. This analysis is performed to test the hypotheses. Table 2 provides the results of the univariate regression analysis that was performed between measures of each individual stability metric and measures of *PRDD* over the successive releases of Apache Ant. This includes correlation coefficients, p-values, and standard error values. At 0.05 level of significance (95% confidence level), significant correlations were observed between each of *NSC*, *NSGR*, and *NSSR* metrics and *PRDD*. Each of *NAC*, *NSAR*, and *NSDR* metrics is also significantly correlated with *PRDD* but at 90% confidence level.

Table 2. Univariate regression statistics

Metric	Coefficient	p-value	Std. Error
NSC	-0.5927	0.0423	1.1939
NAC	0.5242	0.0802	1.2623
NDC	-0.1452	0.6526	1.4666
NMC	0.0966	0.7652	1.4753
NSGR	-0.6424	0.0243	1.1359
NAGR	0.4734	0.1200	1.3056
NDGR	0.1635	0.6117	1.4623
NSAR	-0.5717	0.0521	1.2162
NAAR	0.4504	0.1417	1.3234
NDAR	0.1864	0.5618	1.4563
NSDR	-0.5660	0.0550	1.2219
NADR	0.2378	0.4567	1.4397
NDDR	0.2566	0.4208	1.4327
NSSR	-0.6176	0.0324	1.1658
NASR	0.4260	0.1674	1.3411
NDSR	0.1692	0.5991	1.4609
TBR	0.2114	0.5096	1.4488

3.6.3 Parallel coordinates plot. Parallel coordinates are used to visualize multivariate dataset. A parallel coordinates plot consists of a set of parallel axes; one for each variable. Each observation in the dataset (row of data) is represented by a line that intersects the axes at a point that corresponds to the value the observation has in that variable. Parallel coordinates plots are helpful in recognizing patterns in the dataset.

Using Apache Ant releases as observations in the datasets, Figure 1 shows a parallel coordinates plot for measures of *PRDD* and measures of the stability metrics that were found significantly correlated with *PRDD*. It is observed that releases of Apache Ant that have relatively low *PRDD* are the ones that contain relatively few added classes and many stable classes and relationships from their previous release, and vice versa.

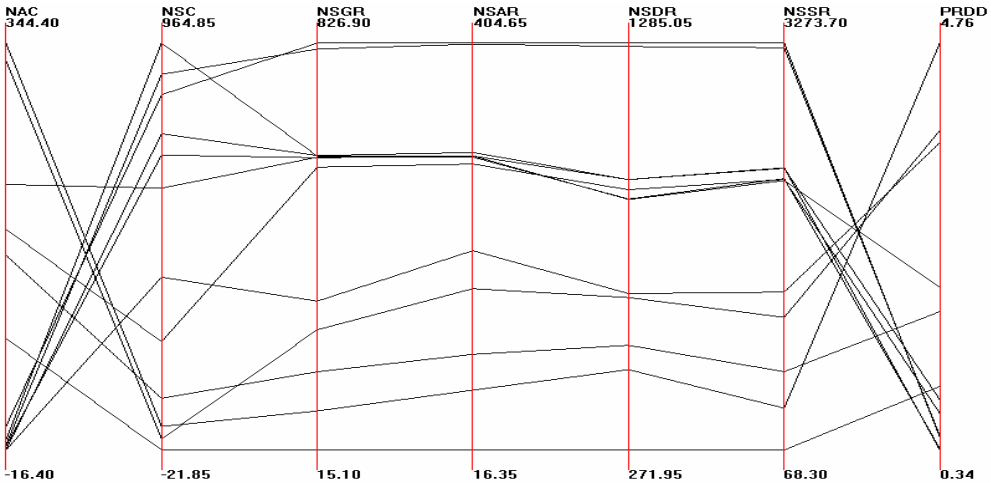

Figure 1. Parallel coordinates plot

3.6.4 Multivariate analysis. The multivariate analysis is performed to construct different multivariate regression estimation models for *PRDD*; each with a different subset of the investigated stability metrics. The accuracy of these models are then evaluated and compared.

Five regression models were built: one from only the class-based stability metrics (Model I); one from only the relationship-based stability metrics (Model II); one from the size-based (both class-based and relationship-based) stability metrics (Model III); one from the time-based stability metric *TBR* (Model IV); and one from all the 17 stability metrics (Model V). A general stepwise selection process [10] that involves both forward selection and backward elimination was used to construct the five models under a 95% confidence level.

Model I was built by running a stepwise regression allowing only class-based stability metrics to enter the model. The resulting model is presented in Table 3. It has only one independent variable, which is *NSC*. The coefficient of determination (R^2) of this model is 35.1%.

Table 3. Model I: class-based

	Coefficient	Std. Error	p-value
Intercept	2.9163	0.5964	0.0006
NSC	-0.0025	0.0011	0.0423

Model II was built by running a stepwise regression allowing only relationship-based stability metrics to enter the model. Table 4 presents the resulting model, which has two independent variables (*NSGR* and *NSAR*), and R^2 of 73.3%. Since Model II is a multivariate model, it was tested for multicolinearity. The test is based on the conditional number of the correlation matrix of the covariates in the model [7]. The conditional number of Model II is ~16.9, which is below the critical threshold of 30. Therefore the degree of multicolinearity in Model II is acceptable.

Table 4. Models II: relationship-based

	Coefficient	Std. Error	p-value
Intercept	1.4686	0.7771	0.0914
NSGR	-0.0307	0.0083	0.0049
NSAR	0.0602	0.0183	0.0095

The resulting Models III and V are equivalent to Model II, i.e. they all have the same independent variables, which are both relationship-based metrics (*NSGR* and *NSAR*). In addition, Model IV could not be built under the 95% confidence level. Therefore, only Model I and Model II are considered for further evaluation.

A leave-one-out cross-validation procedure [18] was used to evaluate the performance of Model I and Model II. In this procedure, one observation is removed from the data set, and then a model is built with the remaining *n-1* observations and evaluated in estimating the value of the observation that was removed. The process is repeated each time removing a different observation. Given that there are 12 observations in this case study, both Model I and Model II were trained using 11 observations and then their estimation accuracy was tested on the withheld observation. This process was repeated for each observation. The performance of each model was evaluated based of the Sum Square Error (*SSE*), which is calculated as follows:

$$SSE = \sum_{i}^{n} \left(X_i - \overline{X}_i \right)^2$$

Where X_i is the actual *PRDD* of observation *i*; $\overline{X_i}$ is the estimated *PRDD* of observation *i*; and *n* is the number of observations.

Figure 2 plots the actual *PRDD* as well as the estimated *PRDD* using Model I and Model II. Model I has SSE of 20.6; whereas Model II has SSE of 12.1. This result indicates that the performance of Model II is better than Model I. This means that Model II is more accurate than Model I in estimating *PRDD*.

3.7. Discussion of results

The results of the univariate analysis in this study suggest the following. The more the number of classes and relationships that remained stable from release *i* to release *i+1* (following release), the less the *PRDD* of release *i+1*. Moreover, the more the number of added classes, the more the *PRDD*. However, the numbers of deleted classes, added and deleted relationships, and time between releases do not seem to have impact on *PRDD*. These results indicate that increased design structural stability from release *i* to release *i+1* (following release) is highly correlated with decreased *PRDD* of release *i+1*.

The results of the multivariate analysis in this study indicate that it is possible to construct statistically significant regression models from subsets of the investigated stability metrics to estimete *PRDD*. In addition, the results have shown that the multivariate regression model that was built from the relationship-based stability metrics (*NSGR* and *NSAR*) outperforms the model built from the class-based stability metric (*NSC*) in estimating *PRDD*.

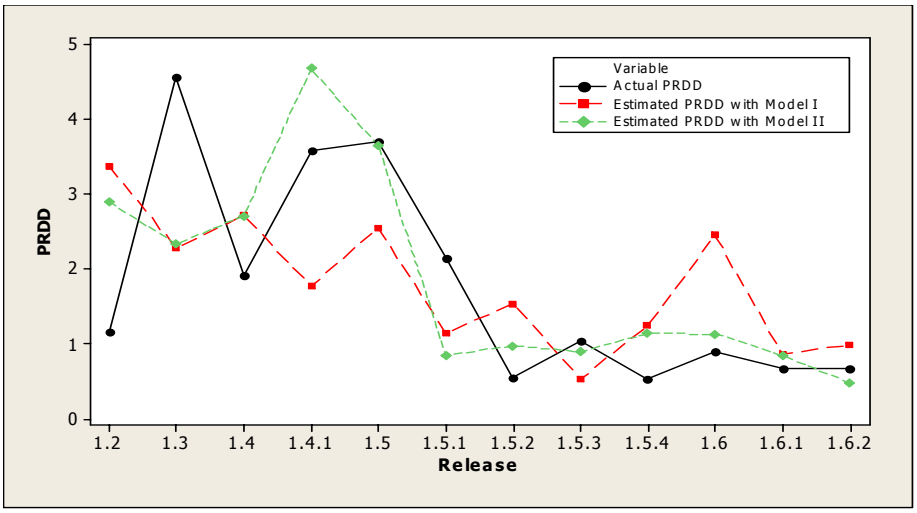

Figure 2. Performance of Model I and Model II

3.8. Limitations

This empirical study has a number of limitations that should be taken into account when interpreting its results. These limitations are not unique to this study, but are characteristics of most of the metrics literature.

Measuring *PRDD* could be threatened by the release usage intensity (i.e. how thoroughly the release has been exercised). The number of user reported defects might be less than the actual number of defects. Such a threat is believed to be very limited in this study since the software analyzed in this case study (i.e. Apache Ant) is popular and a famous open source software that is heavily downloaded and widely-used by many users. It is therefore reasonable to assume that users of Apache Ant have been active in reporting post-release defects, and thus large portions of defects have been reported.

PRDD could be partly dependent on time, i.e. how long a release was available until next release. In this study, there is no significant correlation between *PRDD* and time (correlation coefficient = 0.2389 and p-value = 0.4545).

PRDD could be also threatened by invalid and duplicate defect reports. Fortunately, Apache Ant's user reported defects are verified by the core development team who identifies and marks any invalid or duplicate defects. In this study, invalid and duplicate defects were not counted.

This study is not making claims of causal relationships between the investigated stability metrics and *PRDD*. Instead, this study aimed to empirically explore the correlations between a suite of structural stability metrics for object-oriented designs and post-release defect density.

4. Related work

Related work includes the following research studies. Jazayeri [15] applied retrospective analyses to successive releases of a large telecommunication software system to evaluate its architectural stability. These analyses use simple size measures, coupling measures, and color visualization to observe phenomena about the evolution of the software across releases.

Bansiya [5] proposed a methodology to assess the stability of object-oriented framework architecture using a suite of nine object-oriented metrics. These metrics are collected over successive versions and compared to determine the extent-of-change in the structural characteristics between versions. Mattsson and Bosch [16, 17] extended Bansiya's method with an additional aggregated metric, the relative-extent-of-change metric. These methodologies are limited to assessing the stability of the class inheritance hierarchies, and do not consider other structural characteristics due to other kinds of relationships between classes.

Grosser et al. [13, 14] proposed a case-based reasoning approach for predicting the stability of Java classes. The approach is limited to the prediction of the stability of class interfaces.

Bahsoon and Emmerich [4] proposed an approach for evaluating the stability of software architectures with real options theory. It is an economics-driven approach that requires subjective estimates and market data.

We have also investigated potential product-related and process-related indicators of the structural stability of object-oriented designs [11, 12].

This paper however has a unique objective, which is empirically exploring the correlations between a suite of structural stability metrics for object-oriented designs and post-release defect density. The originality of this paper relies on its results that reveal the practical significance and usefulness of some of the investigated stability metrics as early indicators of one of the important software quality outcomes, which is post-release defect density.

5. Conclusions

This paper has empirically explored the correlations between a suite of structural stability metrics for object-oriented designs and post-release defect density. The investigated stability metrics measure the extent to which the structure of a design is preserved throughout the evolution of the software from one release to the next. As a case study, thirteen successive releases of Apache Ant were analyzed.

The two major findings of the empirical study reported in this paper can be summarized as follows. First, 6 out of the 17 investigated stability metrics were found to have significant correlations with post-release defect density at 90% confidence level, and thus represent early indicators of it. These metrics are *NSC*, *NAC*, *NSGR*, *NSAR*, *NSDR*, and *NSSR*. This result does not imply that the other 9 metrics are useless. They may be significantly correlated with measure of another important external quality attribute of interest such as release effort. This will be investigated as a future work through additional case studies where data, for example, about release effort is available. Second, the multivariate analysis in this study concluded that the multivariate regression model that was built from the relationship-based stability metrics (*NSGR* and *NSAR*) outperforms the model built from the class-based stability metric (*NSC*) in estimating post-release defect density. These results reveal the practical significance and usefulness of some of the investigated stability metrics as early indicators of one of the important software quality outcomes, which is post-release defect density.

As future work, additional case studies are needed to further support the findings of this paper, and to accumulate knowledge [6, 8]. Another research direction is to construct and evaluate nonparametric estimation models such as neural network and Bayesian network that utilize the investigated stability metrics to estimate post-release defect density.

6. Acknowledgment

The authors would like to acknowledge King Fahd University of Petroleum and Minerals for its support.

7. References

[1] The Apache Ant Project, http://ant.apache.org/,

[2] ExamDiff Pro, http://www.prestosoft.com/ps.asp?page=edp_examdiffpro,

[3] JStyle, http://www.mmsindia.com/jstyle.html,

[4] R. Bahsoon and W. Emmerich, "Evaluating Architectural Stability with Real Options Theory," *Proc. International Conference on Software Maintenance*, pp. 443-447, 2004.

[5] J. Bansiya, "Evaluating Framework Architecture Structural Stability," *ACM Computing Surveys*, vol. 32, 2000.

[6] V. Basili, F. Shull, and F. Lanubile, "Building Knowledge through Families of Experiments," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 456-473, July/Aug. 1999.

[7] D. Belsley, E. Kuh, and R. Welsch, *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*, John Wiley & Sons, 1980.

[8] T. Cook and D. Campbell, *Quasi-Experimentation: Design and Analysis Issues for Field Settings*, Sage Publications, 1979.

[9] T. DeMarco, *Controlling Software Projects: Management, Measurement & Estimation*, Prentice-Hall, 1982.

[10] T. Dielman, *Applied Regression Analysis for Business and Economics*, 3rd Edition, Duxbury/Thomson Learning, 2001.

[11] M. Elish, "A Case Study on Structural Characteristics of Object Oriented Design and its Stability," *Proc. IASTED International Conference on Software Engineering*, pp. 89-93, 2005.

[12] M. Elish and D. Rine, "Indicators of Structural Stability of Object Oriented Designs: A Case Study," *Proc. 29th Annual NASA/IEEE Software Engineering Workshop*, pp. 183-192, 2005.

[13] D. Grosser, H. Sahraoui, and P. Valtchev, "An analogy-based approach for predicting design stability of Java classes," *Proc. 9th International Software Metrics Symposium*, pp. 252-262, 2003.

[14] D. Grosser, H. Sahraoui, and P. Valtchev, "Predicting Software Stability Using Case-Based Reasoning," *Proc. 17th IEEE International Conference on Automated Software Engineering*, pp. 295-298, 2002.

[15] M. Jazayeri, "On Architectural Stability and Evolution," *Lecture Notes in Computer Science, Springer-Verlag*, pp. 13-23, 2002.

[16] M. Mattsson and J. Bosch, "Characterizing Stability in Evolving Frameworks," *Proc. Technology of Object-Oriented Languages and Systems*, pp. 118-130, 1999.

[17] M. Mattsson and J. Bosch, "Stability assessment of evolving industrial object-oriented frameworks," *Journal of Software Maintenance: Research and Practice*, vol. 12, pp. 79–102, 2000.

[18] S. Weiss and C. Kulikowski, *Computer Systems that learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*, M. Kaufmann Publishers, 1991.