Software Quality Journal, 13, 297–320, 2005

Modeling Design/Coding Factors That Drive Maintainability of Software Systems

SUBHAS CHANDRA MISRA scmisra@connect.carleton.ca
Carleton University, 1125 Colonel By Drive, Ottawa, Canada

Abstract. It is cost-effective for software practitioners to monitor and control quality of software systems from the early phases of development. Assessing and modeling the effects of design and coding factors on software system maintainability can help provide heuristics to human designers and programmers to reduce maintenance costs and improve quality. This paper presents a study based on intuitive and experimental analyses that use a suite of twenty design/code measures to obtain indications of their effect on maintainability. This paper lists several important contributions of the work, one of which is the investigation of an unprecedentedly large number of systems (fifty) in a single study. The previous related studies on the other hand, have investigated 2–8 systems. The results reported in this paper using experimental procedures are *unique*, many of which have not been empirically established in the previous literatures, and are interesting because they are not normally intuitively obvious in most cases. The study also serves to empirically validate those results that seem to be intuitive. The results of the study indicate a number of promising effects of design and coding factors on system maintainability. The use of the results from the relatively early phases of software development could significantly help practitioners to improve the quality of systems and thus optimize maintenance costs.

Keywords: software quality, maintainability, metrics, regression, correlation, modeling

1. Introduction

The maintenance phase in the complete software system lifecycle is often the most expensive and time-consuming of all phases. An important objective of software quality research is to devise and engineer methodologies to produce high quality software at low costs. As the software development progresses in its lifecycle through the phases of requirements analysis, design, implementation, testing, and maintenance, the complexity and the cost of the software increases. Because of its cost, software maintainability remains a major challenge in software quality engineering studies. Indications of the impact of design and coding considerations concerning maintainability in the "relatively early phases" of software development, should help the software programmer to improve design or coding in the hope of reducing the final maintenance costs that can occur due to poor design and coding. The phrase "relatively early phases" is used to mean early phases relative to the maintenance phase, viz., design or implementation phases.

The existing pieces of literature[1] related to the area of concern in this paper can be broadly classified into the following two categories[2]:

1. The ones that proposed new metrics: These new metrics can be classified as either the ones that measure the quality of software processes (process metrics), or as ones that measure the quality of the software products (product metrics). Since this paper

discusses product-based metrics, this paper will review briefly some of the important literatures that proposed product-based metrics. Broadly, these literatures propose measures that are either applicable for measuring procedural software (Adamov and Ritcher, 1990; Halstead, 1997; McCabe, 1976), or object-oriented software (Briand et al., 1997; Chidamber and Kemerer, 1991, 1994; Henderson-Sellers, 1996; Li and Henry, 1994; Lorenz and Kidd, 1994).

2. The ones that validated the existing and their newly proposed metrics, and investigated the relationship between the different metrics and different quality attributes: There are a plenty of works published in this category. Following the proposal of the different metrics by the different authors identified in (1) above, attempts were made by Basili et al. (1996), Briand et al. (1997, 1998, 2000, 2001), Briand and Wust (2001), Cartwright and Shepperd (2000), Daly et al. (1996), Deligiannis et al. (2003, 2004), El Emam et al. (2001, 2002), Koru and Tian (2003), Li and Henry (1993), Prechlelt et al. (2003), and Succi et al. (2003), to validate these measures and study their impacts on the number of faults occurring in software (which, though, has an impact on software maintenance).

Among the recent studies conducted in this area, Briand et al. (2000) studied the empirical relationships between a few design measures and the probability of fault detection during system testing. Their investigation revealed strong relationships between different inheritance and coupling based measured and the probability of fault detection in a class. Further, Briand et al. (2001) performed another experiment with similar goals, and found the strong impact of system size, coupling, cohesion, and complexity on development effort. Following the previous validation studies by the different authors mentioned above, El Emam et al. (2001) performed a validation study using fault data in which they investigated the confounding effect of class size on the validity of different object-oriented metrics. They found very interesting results that led the authors to cast doubt over some of the previous validation results reported in the pieces of literature published prior to their studies. El Emam et al. (2002) further reported results of studies aimed at investigating the optimal class size in regards to good quality software. In these studies the above authors assumed the faults occurring in software as a contributor to the maintainability (quality) of the software.

Apart from the principal pieces of work performed by the above mentioned authors, several other discrete pieces of work broadly related to the results reported in this paper are summarized below. Misra and Bhavsar (2003) studied the relationships between different software measures, and the number of bugs detected by the Halstead's bug-density measure (Halstead 1997). Misra and Bhavsar (2003a) also extended their study to investigate the relationships with Halstead's program difficulty measure (Halstead, 1997). Succi et al. (2003) empirically evaluated the impact of design decisions on the defect behavior of classes. Koru and Tian (2003) reported results of their study in which they intended to compare and characterize the similarities and dissimilarities between high complexity modules and high defect rates. They found out that modules with higher defect rates are usually those with complexity rankings little below the most complex ones. Deligiannis et al. (2003) investigated the specific effect of any object-oriented design heuristic on the maintainability of object-oriented systems. In another later study, Deligiannis et al. (2004) performed a controlled experiment using undergraduate student subjects to investigate the impact of

some design principles on maintainability. Prechelt et al. (2003) performed a controlled experiment to study the effect of the depth of inheritance on code maintenance, and found out that the previous claims that the depth of inheritance affects maintenance was without sound basis.

Among the above mentioned previously published works, although a few notable studies (Binkley and Schach, 1997; Daly et al., 1996; Deligiannis et al., 2004; Harrison et al., 1991, 2000; Lanning and Khoshgoftar, 1994; Li and Henry, 1993; Prechlelt et al., 2003; Rombach, 1987; Wake and Henry, 1988) observe the effect of the different design/code level metrics on maintainability, all those studies collected maintainability data using indirect schemes such as fault data during system testing, development effort, and maintenance data through subjective evaluation of software. Since maintainability can be affected by a broad spectrum of reasons, it is worth considering maintainability models in such studies that consider several issues.

This paper presents a study conducted to study the usefulness of different, widely accepted design/code level metrics in assessing maintainability early in the software lifecycle. This paper is based on a comprehensive empirical study involving direct measurements of source-code through chosen projects. The chosen projects have varied characteristics. Maintainability was measured using the widely accepted Maintainability Index (MI) (Welker and Oman, 1995). The values of the metrics studied were collected from static analyses of source-code. The value obtained for each metric was statistical investigated together with the value of MI. Interesting results that could not even be predicted logically were obtained through the experimental analysis. Different bivariate and multivariate statistical analyses were conducted so as to help the designers and programmers for controlling maintainability of the final product by monitoring the effects of the important early phase design/coding factors. However, it should be cautioned, that this study only provides probable statistical indications of the effects of different measures on maintainability. The research conducted in this study only takes into account the objective measures and does not consider the subjective factors that influence human designers and maintainers of a system. The results of this study should help the designers and programmers to consider different factors that they should always keep in mind while developing the overall system. Some of the important contributions of this research are highlighted below:

- **Comprehensive study**: This work considered twenty different design/code level measures and studied their effect on maintainability using different statistical and logical techniques. Several interesting results were found from empirical analyses that apparently do not follow intuition. The conclusions drawn from this work will serve as guidance to the designers and programmers in developing a well maintainable product by controlling the influence of important design/coding factors from the early phases of software development. The previous works that were done in the similar lines, as mentioned above, did not empirically investigate the effects caused by most of the factors considered in this study. While the exploration of the existence of relationships among some of the factors and maintainability may sound trivial, their empirical validation is required to establish the existence of such relationships. This also serves to negate the incorrect claims, if any, made by the proponents of object-oriented development techniques.
- **Confluent effects on maintainability**: This work not only studies the singular effects of the individual metrics on maintainability, but also considers the confluent effects of

all twenty metrics on maintainability. This work is an important contribution over the previous studies, as no study in the past considered such a large selection of different important metrics on maintainability. Consequently, these results will help the conscientious designers and programmers in prioritizing those important design/coding factors that have significant impact on maintainability.

- **Relationship between independent variables**: An important contribution of this work is that for the first time, this paper has studied the relationships solely amongst the independent variables considered in this study (i.e., the measures that are used to predict maintainability). This investigation is quite insightful because some of these measures are highly correlated between themselves and do not help to better explain the behavior of maintainability.
- **Investigated a large number of systems**: Because of the rigor and complexity involved in collecting a large number of systems and extracting a wide range of data from them, it is quite possible that the previous studies investigated only a very limited number of systems (typically two to eight). This limits the variability of the application domain, size and complexity of the systems. Thus, the conclusions drawn from the data collected from these limited number of systems tend to be unreliable. In this unprecedented study, this paper has considered a large number of systems to draw reliable conclusions. Data has been collected from 50 different systems. 30 systems were used for designing this model and 10 were used for the purpose of validation.
- **Improved accuracy of the conclusions**: Unlike the previous studies, this paper has minimized the biasness of the choice of systems by randomizing the selection of samples. This improves the accuracy of the conclusions drawn from this study.
- **Considered multiple aspects of maintainability**: Most previous studies considered maintainability based on either, or some, of the following: non-comment source lines, known errors, error density, subjective maintenance effort or real industrial software modification reports. Since maintainability comprises of many factors (e.g., the psychological factors that drive proper designing and coding of systems developed), it would be more practical to consider many aspects of maintainability together as a dependent variable. Such models would be more valuable to study the relationship between different design level metrics and maintainability while observing the effect of the former on the latter early in the software lifecycle. Unlike the previous studies, this paper considered the well-accepted indicator of maintainability, the Maintainability Index (MI) (Welker and Oman 1995), for collecting the data of maintainability of different systems.

2. Description of the study

2.1. Purpose of the study

An intuitive and empirical study was conducted to assess the nature and level of association between maintainability and the different software metrics that can be obtained from the relatively early stages of the software lifecycle. In other words, it was intended to observe the statistical nature and significance of a relationship between a design/code level metric and maintainability, and to model the relationship between the two. By knowing the values of the design/code metrics from the early stages of development, quality can be monitored right from the design and coding stages.

2.2. *Systems investigated*

In the study, 50 projects written in C++ were considered. The systems varied in size and application domains, and were mostly obtained as open-source software. Table 1 summarizes some of the characteristics and sources of collection of the systems investigated.

2.3. *Experimental procedure and data collection*

Source code delivered at the end of the implementation phase was collected from different projects. The measurement tool, Krakatau Metrics Professional developed by Power Software Inc., was then used to extract the values of the different traditional, procedural and object- oriented metrics.[3] Each experiment involved obtaining one of the twenty metrics data along with the data of maintainability. The methodology can be summarized into the following main steps:

1. Initially, a pool of 50 C++ programs (Table 1) was constructed with various ranges of lines of code, number of classes, number of methods and application domains.
2. From the pool, 30 programs were randomly selected using a random number generator. These programs were classified as the "model builder" sample set, and were used to perform descriptive analysis, bivariate correlation, bivariate regression, multivariate regression analysis, ANOVA, and to build the final model of multiple regression.
3. From the remaining 20 sample programs, again 10 programs were randomly selected using a random number generator. These programs were classified as the "validation" sample set, and were used to validate the multiple regression model.[4] The validation exercise is used to statistically check the correctness of the model developed using empirical methods.
4. Two metrics, one of the twenty design/code level metrics and maintainability, were extracted using a static analysis tool for each sample of both the "model builder" and the "validation" sample sets.
5. Steps 2–4 were repeated for all the 40 samples.
6. The data obtained is used for analysis using statistical methods (See Section 3).

2.4. *Variables and the data collected*

For each of the samples, a broad range of values of twenty different software measures were collected along with the value of maintainability. Table 2 lists, in an alphabetic order, the different predictive metrics considered in the study and their definitions. The metrics selected in the study are either from the popular metrics suites proposed and validated by Chidamber and Kemerer (1994), Lorenz and Kidd (1994), and Brite e Abreu and Carapuca (1994), or from other metrics that are commonly used and have been validated in the past, for example, the number of source lines of code, and the number of methods. According to the author's opinion, these selected metrics characterize most of the major functionalities of object-oriented designs.

Table 1. Systems investigated.

Syst.	Source	Lines of code	Num class	Num method
1	http://www.csse.monash.edu.au/∼ darrenp/diamondbase.html	16976	23	476
2	http://sourceforge.net/project/showfiles.php?group_id = 3191	11943	16	393
3	http://www.objectcentral.com/ (V)	83436	135	1904
4	http://yaktest.sourceforge.net/	931	18	65
5	http://www.sashanet.com/internet/download.html#anchor_home	709	4	49
6	http://www.freecode.com/projects/billgpc/	8175	3	154
7	http://www.codebeach.com/(NETCLASS)	1272	5	12
8	http://www.nwlink.com/ ∼ mikeblas/samples/index.htm (COLORLB)	704	4	21
9	http://www.laas.fr/ ∼ ortalo/openamulet/	117447	156	4561
10	http://www.nwlink.com/∼ mikeblas/samples/index.htm (APIBROW)	900	5	11
11	http://coool.mines.edu/	3600	1	26
12	http://www.geocities.com/TheTropics/Paradise/7231/GraphLib.htm	809	19	91
13	http://www.shinecomp.com/index.shtml	3889	19	257
14	http://pobox.com/∼ oleg/ftp/Communications.html#tcp-stream	1389	17	81
15	http://www.gradsoft.com.ua/eng/Products/ToolBox/toolbox.html	9159	61	458
16	http://gql.sourceforge.net/	11943	16	393
17	http://www.orcane.net/freeodbc++/	16621	16	47
18	http://www.openip.org	117447	156	4561
19	http://www.cs.wustl.edu/∼ schmidt/ACE-overview.html	431910	12498	983
20	http://www.geocities.com/corwinjoy/dtl	15858	47	746
21	http://osalp.sourceforge.net/	86814	56	1293
22	http://lin.fsid.cvut.cz/∼ kra/index.html#QpThread	7273	91	581
23	http://www.gnu.org/software/goose/goose.html	19549	46	943
24	http://yukon.genie.uottawa.ca/∼ lavoie/software/nurbs/	38184	1	721
26	http://www.odin-consulting.com/OPP/	46579	384	3523
27	http://www.dip.ee.uct.ac.za/∼ brendt/srcdist/	15615	56	710
28	http://math.nist.gov/sparselib++	17802	22	583
29	ftp://ftp.simtel.net/pub/simtelnet/msdos/cpluspls/calcplus.zip	3618	47	382
30	http://www.fltk.org	76046	182	3236
31	http://corelinux.sourceforge.net/	53516	100	1381
32	http://www.webthing.com/cgiplusplus/	18772	40	837
33	ftp://ftp.virginia.edu/pub/tools/	4978	34	394
34	http://www.xraylith.wisc.edu/∼ khan/software/fftpack/	16779	37	873
35	http://www.cryst.bbk.ac.uk/classlib/	10863	31	393
36	http://www.netwood.net/∼ edwin/svmt/	37321	88	3882
37	http://home.att.ne.jp/green/atlan/oz/index.html	18193	147	1240
38	http://www.oonumerics.org/blitz/	98287	273	5036
39	http://www.csg.is.titech.ac.jp/∼ chiba/openc++.html	40379	135	1304
40	http://www.eskimo.com/∼ weidai/cryptlib.html	42611	308	2145
41	http://www.nwlink.com/∼ mikeblas/samples/index.htm (MRULESS)	928	6	25
42	http://www.media.mit.edu/∼ kbrussel/SocketMan/	7047	4	155
43	http://www.gnu.org/software/goose/goose.html	195549	46	943
44	http://www.geocities.com/SiliconValley/Horizon/1350/pacman/	6262	21	196
45	http://www.ph.tn.tudelft.nl/∼ klamer/cppima.html	6726	37	351
46	http://www.zeta.org.au/∼ jon/STL/views/doc/views.html	858	8	42
47	http://www.codesites.com/ (OPENC++)	40379	135	1304
48	http://www.xraylith.wisc.edu/∼ khan/software/fftpack/	16779	37	873
49	http://goethe.ira.uka.de/∼ wilhelmi/pvm++/	1958	1	70
50	http://www.vxcl.org/	15691	45	260

Table 2. Predictive metrics.

Predictive metric	Acronym	Definition
Average class size	ACLOC	Average class size in terms of the number of lines per class
Attribute Hiding factor	AHF	The measure of the visible attributes, and can be calculated by summing the visibility of each attribute in respect to the other classes in the system (Brite e Abreau and Carapuca, 1994). In the calculation above, private = 1, public = 0, Protected = Size of the Inheritance Tree / Number of Classes.
Attribute inheritance factor	AIF	The percentage of class attributes that are inherited. It is calculated by summing the inherited attribute for all classes from its super-classes in a project.
Average method size	AMLOC	Average method size in terms of number of lines per method (Lorenz and Kidd, 1994)
Average depth of paths	AVPATHS	The average depth of paths from methods that have paths at all.
Control density	CDENS	Represents the percentage of control statements in the code
Coupling factor	COF	Coupling Factor measures the extent of communication between client and supplier classes (Lorenz and Kidd, 1994). It is measured for the entire project as the fraction of the total possible class coupling. Its value ranges between 0 and 1. Lower values are better than higher ones.
Depth of inheritance tree	DIT	Depth of inheritance tree (Chidamber and Kemerer, 1991) measures the position of a class in the inheritance tree. It corresponds to the level number of a class in the inheritance hierarchy. The root class has DIT value of zero.
Lack of cohesion in methods	LOCM	LOCM metrics calculates the degree of communication between the methods and member variables of a class (Chidamber and Kemerer, 1991). It is obtained by calculating a list of member variables and the number of references to each variable of all methods in that class. Secondly, the sum of the ratios of the usage divided by the total number of methods is obtained. Finally, LOCM is calculated as the quotient of the sum of ratios by the total number of attributes.
Method hiding factor	MHF	Helps to measure the visibility or invisibility of each method with respect to other classes in the project (Brite e Abreau and Carapuca, 1994). The visibility is calculated as follows: private = 1, public = 0, protected = Size of the Inheritance Tree divided by the Number of Classes.
Method inheritance factor	MIF	Obtained by dividing the total number of inherited methods by the total number of methods. The total number of inherited methods, on the other hand, is obtained by summing the number of operations that a class has inherited from its super- classes.
Program length	N	Measures the total number of operators and operands in a program (Halstead, 1997).

(*Continued on next page*).

Table 2. (*Continued*).

Program vocabulary	n	Measures the total number of unique operators and unique operands in a program (Halstead, 1997).
Number of Classes	NCLASS	Calculates the total number of classes in a system.
Number of methods	NMETH	Calculates the total number of methods in a system.
Polymorphism factor	POF	Helps to measure the degree to which classes within a system are polymorphic (Brite e Abreau and Carapuca, 1994). Polymorphism is used in object oriented programming to perform run-time binding to one class among several other classes in the same hierarchy of classes. Polymorphism helps in processing instances of classes according to their data type or class.
Percentage public/protected members	PPPC	Calculates the percentage of public and protected members of a class with respect to the other members of the class.
Response for classes	RFC	Measures the cardinality of the response set of a class (Chidamber and Kemerer, 1994). Response for class is the number of methods in a class, plus the number of distinct methods called by those methods. Since the principal mode of communication between objects is through message passing, an object can be made to act in a certain way through a particular way of method invocation.
Source lines of code	SLOC	Calculates the number of source lines in the project. However, this excludes lines with white-spaces and comments.
Weighted methods in classes	WMC	Reflects the complexity of the classes and is the sum of the cyclomatic complexities of all methods in the classes (Chidamber and Kemerer, 1994). The WMC metric is obtained by summing the values of McCabe's Cyclomatic Complexity of all local methods.

For quantifying the effort needed for maintaining software, MI (Welker and Oman, 1995) was considered as the dependent variable. The lower the value of MI, the more difficult it is to maintain and vice versa.

MI is computed as a function of four metrics: the average Halstead's Volume per module (avgV), average extended cyclomatic complexity (avgV(G')), average lines of code (avgLOC), and average percent of lines of comments per module (perCM). Specifically, MI $= 171 - 5.2^{*} \ln(\text{Avg V}) - 0.23^{*} \text{avg V}(G') - 16.2^{*} \ln(\text{avgLOC}) + 50^{*} \sin(\text{sqrt}(2.4^{*} \text{perCM}))$. For further details, the readers are referred to Welker and Oman (1995) or Welker et al. (1997). Welker et al. (1997) discuss four MI metric models of which the three-metric and the four-metric models are considered to be better than the one-metric or the five-metric models. Considered in the study was the four-metric model because it is believed that comments in code have the potential to significantly contribute to maintainability.

It was intended to estimate the degree to which the maintainability index and the different metrics, obtainable early in the software lifecycle, are related. MI has been validated several times in the past (Coleman et al., 1994; Oman and Hagemeister, 1994; Welker and Oman, 1995) and has been shown by Oman and Hagemeister (1994) that model components are good and sufficient predictors of maintainability.

ACLOC, AHF, AIF, AMLOC, AVPATHS, CDENS, COF DIT, LOCM, MHF, MIF, n, N, NCLASS, NMETH, POF, PPPC, RFC, SLOC and WMC were considered as the independent predictor variables.[5] In the study, each of these measures were considered along with maintainability and the extent to which the former affects the latter was studied using the techniques described in Section 3.

3. Analysis

3.1. Intuitive analysis

In order to determine the effect of the different design level measures on quality, the expected relationship between the former and the latter was first intuitively analyzed and predicted. For the sake of brevity, the detailed analysis of the results of the intuitive analysis are not described, but are summarized in Table 3. The following are the results of the intuitive analysis, in most general cases. However, there can be deviations from these points of analyses in certain cases.

3.2. Experimental analysis

The intuitive analyses in Section 3.1 suffice only to predict the logical relationships between the different design/coding factors and maintainability. However, they do not provide sufficient evidence of the degrees of relationship between the different factors and maintainability, the nature of variations of maintainability with respect to the variations in the factors, and the extent of the effects of one factor in the presence of the others. These characteristics need to be determined and validated using experimental means. The experimental methodology described in Section 2.3 was used to collect data, and analyze them statistically to address the above issues.

3.3. Statistical analysis

Summarized in this section are some of the many statistical analyses performed on the data obtained by performing the experiments in Section 3.2. The analytical techniques used were Descriptive Analysis, Influential Analysis and Diagnostics, Linear and Non-Linear Regression Analysis, Correlation Analysis, and Multiple Regression Analysis. The results obtained from Multiple Regression were also statistically validated.

3.3.1. Descriptive statistics Basic descriptive statistics was performed for each of the variables. While performing the analyses, the problematic or corrupted data points (points that have inconsistencies, unusual distributions, unaccountability or incompleteness) were rejected from further considerations. Some variables, e.g., DIT, NCLASS, NMETH and SLOC, which, by definition, can only take integral values, were inspected in the frequency tables for their correct type. The frequency table for DIT also shows a special condition of this variable since all its values are clustered in only seven points. The definitions of the other variables do not limit feasible values, so it was not possible to detect inaccurate

Table 3. Results of intuitive analysis: Relationship with maintainability.

Predictive measure	Relationship (+ve, −ve, nil)	Intuitive cause of the relationship (with increase in the value of the predictive measure)
Average class size (ACLOC)	−ve	The class becomes more complex, less structured and more difficult to understand.
Attribute hiding (AHF)	+ve	In general, the amount of abstraction increases. However, the analysis could be different in cases such as where the system is actually abstracted to too great a degree and important aspects are lost, requiring rework to fix as the systems are maintained/evolved.
Attribute inheritance factor (AIF)	−ve	As attribute inheritance increases, the number of attributes coupled between different classes' increases.
Average method size (AMLOC)	−ve	The method becomes more complex, less structured and more difficult to understand. However, the analysis could be different in cases such as where there are very small methods. The tracing and understanding of dynamic behavior becomes more difficult, so the relationship between average method size and maintainability may be more complex.
Average depth of paths (AVPATHS)	−ve	The classes and methods become more difficult to understand, debug and test
Control density (CDENS)	−ve	The number of controls paths that should be traced or executed in a software unit increases.
Coupling (COF)	−ve	Coupling increases communication between classes, reduces encapsulation, and in turn increases the complexity of software.
Depth of inheritance tree (DIT)	−ve	The class deeper in the hierarchy could inherit more data from its ancestors than the classes in the shallower levels.
Lack of cohesion (LOCM)	−ve	Cohesion supports encapsulation and it reduces the complexity of the software to understand, implement and test.
Method hiding (MHF)	+ve	In general, the amount of abstraction increases. However, the analysis could be different in cases such as where the system is actually abstracted to too great a degree and important aspects are lost, requiring rework to fix as the systems are maintained/evolved.
Method inheritance (MIF)	−ve	The number of methods coupled between different classes increases
Program vocabulary (n)	−ve	The sum of the number of unique operators and operands increases.
Program length (N)	−ve	The total number of operators and operands increases.
Number of classes (NCLASS)	−ve	The intelligent content, the number of bugs, and the difficulty to understand and test increases.
Number of methods (NMETH)	−ve	The intelligent content, the number of bugs, and the difficulty to understand and test increases.

(*Continued on next page*.)

Table 3. (*Continued.*)

Polymorphism (POF)	−ve	The run-time binding of message calls to one of −venereal classes in the same class hierarchy increases. Thus, polymorphism makes code very difficult to understand (especially in a dynamically typed environment). However, on the other hand, it should also be noted this it is also often intended to make the addition of further subclasses exhibiting the same polymorphic behavior easier. So some maintenance changes may produce a positive correlation.
Percentage Public/ Protected Members (PPPC)	−ve	The visibility outside of a class increases, and the encapsulation decreases.
Response for Class (RFC)	−ve	The number of methods and the number of distinct methods called by those methods in a class increases.
Source Lines of Code (SLOC)	−ve	The intelligent content, the number of bugs and the difficulty to understand and test increases.
Weighted Methods in Class (WMC)	−ve	The static complexity of the methods, the control flows will be more complex.

data from mere inspection. Histograms and box-plots were also drawn for each variable to see the shape of the samples and to help detect outliers, which will be studied further in the diagnostic for the regression analysis. To maintain brevity of the paper, scatter plots, histograms and box-plots are not presented. However, Table 4 in the Appendix shows the summary of the results from the descriptive analysis.

3.3.2. Linear regression: Diagnostics for independent variables A linear regression model was obtained for each independent variable versus the dependent variable, MI. A diagnosis of influential points was done using leverage analysis and Cook's distance to check the outliers. The usual threshold values were used to eliminate points, i.e., 4/n for Cook's distance and $(2k + 2)/n$ for the leverage (where n stands for number of observations and k for number of predictors).[6] Thereafter, the linear regression analyses were performed without these points. In some cases, the results were analyzed again (drawing required plots, like histograms and scatter-plots of residuals) and several iterations of analyses were performed including or excluding influential points to find the best results. Table 5 in the Appendix summarizes for each variables the results of all models including (i.e., for all cases) or excluding the influential points (i.e., filtered).

The values of the correlation coefficient, together with that of the p-value of the linear regression models, show that the variables ACLOC, AMLOC, AVPATHS, CDENS, COF, DIT, n, N, PPPC, and WMC are relatively strongly[7] influential on MI, the variables MHF, RFC, SLOC, and AIF are relatively moderately influential on MI, whereas the other variables are negligibly influential on MI. It is to be noted from the results of the regression analysis that the significance levels reported are singularly related to that of linear regression. Thus, for the cases where no good model was found, a non-linear relationship should be explored. Besides, since the highest R^2 found is 0.657, many of the variables considered do not explain, per se, more than half of the behavior of MI. So, a multiple regression model should be considered.

Table 4. Results of descriptive analysis.

	N	Minimum	Maximum	Mean	Std. deviation
SAMPLE	30	1	50	25.67	14.667
ACLOC	30	11.00	106.88	40.1454	27.37158
AHF	30	.000	.333	.17330	.107055
AIF	29	.1251	.5412	.336885	.1145597
AMLOC	30	5.068	27.157	13.36660	6.630415
AVPATHS	30	1.05	4.51	2.2129	.89455
CDENS	30	.1300	.5350	.398759	.0876392
COF	30	.00	.45	.1813	.13752
DIT	30	0	7	2.27	1.760
LOCM	30	12.00	118.00	65.3882	28.66731
MHF	30	.0	.4	.133	.1034
MIF	30	.000	.386	.15730	.130833
SMALLN	30	32.44	171.91	77.8120	38.54853
N	30	6	1509	587.66	366.112
NCLASS	30	1	384	96.70	107.660
NMETH	30	81	7547	1780.07	1945.231
POF	30	.00	.55	.1788	.16365
PPPC	30	33.16	118.00	81.7065	15.69710
RFC	30	4.47	9875.00	345.2230	1800.00367
SLOC	30	744	143209	27005.23	36495.306
WMC	30	4.43	49.80	17.4656	10.91256
MI	30	68.55	148.37	95.6436	14.38119
Valid N (listwise)	29				

3.3.3. Non-linear models An analysis of non-linear fit was done for each independent variable against the dependant variable MI. In each case, 7 different models, quadratic, cubic, exponential, inverse ($y = a + b/x$), logarithmic ($y = a + b\ \ln(x)$), power ($y = ax^{\mathrm{b}}$), and the S model ($y = \exp(a + b/x)$), were considered along with the linear model. For each model, the R, R-square and ANOVA results were considered to select the best fitting model. When the results were similar in nature, quadratic and cubic models were discarded, since their estimation demands more degrees of freedom (they take more coefficients).

The complete results are listed in Table 6 in the Appendix. From this analysis, good fitting models were found for 8 of the variables: ACLOC, AMLOC, AVPATHS, CDENS, DIT, PPPC, SLOC, and WMC. For the first six the inverse model was selected, for SLOC a logarithmic model was selected, and for WMC the best fitting model was found to be linear.

3.3.4. Correlations between independent variables Correlation analysis was performed between the independent variables used in the study. In case two, the variables were found to be highly correlated, one of them was considered to be redundant as a predictor, since it carries the same information as the other. Both Pearson's and Spearman's correlation were performed since Pearson's analysis only detects linear associations. The significant values (at least at 95%) obtained are listed below in Table 7 in the Appendix. The values non-significant for Spearman's rho, but significant for Pearson's, are marked with a "star"(*).

The correlation analysis was performed again excluding the outliers identified in the bivariate linear models. The results are shown in Table 8 in the Appendix. Although many of the associations are weak, some of the variables show strong associations with many other

Table 5. Results of linear regression.

Variable	Model	*N*	*R*	*R*-square	*P*-value	Const.	Coefficient
ACLOC	All cases	30	0.221	0.049	0.241	100.305	−0.116
	Filtered	25	**0.601**	0.361	0.001	105.223	−0.308
AHF	All cases	30	0.070	0.005	0.712	97.279	−9.437
	Filtered	29	0.206	0.042	0.284	90.067	20.963
AIF	All cases	29	0.243	0.059	0.204	84.869	30.241
	Filtered	28	0.013	0.000	0.947	94.419	−1.078
AMLOC	All cases	30	0.662	0.439	0.000	114.845	−1.436
	Filtered	28	**0.810**	0.657	0.000	110.284	−1.242
AVPATHS	All cases	30	0.086	0.007	0.650	92.568	1.390
	Filtered	27	**0.536**	0.287	0.004	76.585	2.280
CDENS	All cases	30	0.286	0.082	0.125	114.364	−46.946
	Filtered	28	**0.413**	0.171	0.029	112.644	−44.404
COF	All cases	30	0.418	0.174	0.022	87.728	43.669
	Filtered	27	**0.415**	0.172	0.31	87.408	34.960
DIT	All cases	30	0.176	0.031	0.353	92.391	1.435
	Filtered	27	**0.434**	0.188	0.021	88.839	2.425
LOCM	All cases	30	0.119	0.014	0.531	99.551	−5.9E-2
	Filtered	28	0.193	0.037	0.324	90.392	6.7E-2
MHF	All cases	30	0.207	0.043	0.272	99.480	−28.803
	Filtered	26	**0.379**	0.144	0.056	99.485	−34.518
MIF	All cases	30	0.015	0.000	0.938	95.386	1.637
	Filtered	28	0.155	0.024	0.430	92.777	11.577
n	All cases	30	0.186	0.034	0.326	101.263	−6.9E-2
	Filtered	26	**0.408**	0.166	0.039	102.206	−0.111
N	All cases	30	0.124	0.015	0.515	98.497	−4.8E-3
	Filtered	27	**0.419**	0.176	0.030	101.263	−1.3E-2
NCLASS	All cases	30	0.006	0.000	0.973	95.727	−8.6E-4
	Filtered	25	0.084	0.007	0.671	93.963	7.4E-3
NMETH	All cases	30	0.158	0.025	0.405	97.718	−1.2E-3
	Filtered	27	0.183	0.034	0.360	95.957	−8.8E-4
POF	All cases	30	0.145	0.021	0.446	97.914	−12.700
	Filtered	27	0.187	0.035	0.349	96.592	−11.782
PPPPC	All cases	30	0.137	0.019	0.506	95.262	−8.063
	Filtered	26	0.467	0.219	0.016	54.675	0.479
RFC	All cases	30	0.121	0.015	0.524	95.978	−9.7E-4
	Filtered	28	**0.278**	0.077	0.152	91.747	0.140
SLOC	All cases	30	0.270	0.073	0.149	98.515	−1.06E-4
	Filtered	27	**0.310**	0.096	0.116	97.236	−9.8E-5
WMC	All cases	30	0.115	0.013	0.544	98.298	−0.152
	Filtered	27	**0.521**	0.272	0.005	102.754	−0.512

variables: SLOC, NMETH and NCLASS are particularly highly correlated between them. Thus, these variables are candidates to be left out while considering a multiple regression model (their inclusion was also tested).

3.3.5. Multiple regression For performing *multiple regression* analysis, new variables were created with the corresponding transformations to inverse, exponential, or logarithm, so that the transformed variables can be included in the model linearly using the multiple regression function. For instance, since the selected model for AMLOC was the inverse, a new variable called i_AMLOC was created with the inverse of the values for AMLOC.

Table 6. Non-linear models.

Variable	Mode	*R*-square	Signif.	Coefficient	Constant
ACLOC	Inverse	0.337	0.0015	311.823	83.50
AMLOC	Inverse	0.743	0.0000	176.527	77.128
AVPATHS	Inverse	0.141	0.0489	−21.928	105.548
CDENS	Inverse	0.182	0.0237	3.778	84.610
DIT	Inverse	0.240	0.0095	−14.288	103.384
PPPC	Inverse	0.234	0.0123	−3159.223	132.919
SLOC	Log	0.146	0.0450	−2.819	121.497
WMC	Linear	0.272	0.0053	−0.512	102.754

Table 7. Correlation results between independent variables (with outliers).

Variable 1	Variable 2	Pearson Correlation	Significance (2-tailed)	Spearman's Rho	Significance (2-tailed)
ACLOC	AHF	0.418	0.021	0.351*	0.058*
ACLOC	AIF			−0.387	0.038
ACLOC	NMETH	0.384	0.036	0.557	0.001
ACLOC	SLOC			0.407	0.026
ACLOC	WMC	0.641	0.000	0.632	0.000
AHF	AVPATS	0.373	0.043	0.312*	0.094*
AHF	NCLASS			0.375	0.041
AHF	NMETH			0.388	0.034
AHF	POF			0.393	0.032
AMLOC	DIT	−.411	0.024	−0.432	0.017
AMLOC	WMC			0.407	0.025
AVPATHS	DIT	0.526	0.003	0.598	0.000
AVPATHS	N	0.407	0.026	0.243*	0.195*
AVPATHS	NCLASS			0.444	0.014
AVPATHS	RFC			0.448	0.013
CDENS	RFC			0.387	0.035
CDENS	WMC	0.480	0.007	0.503	0.005
DIT	MIF			0.446	0.013
DIT	NCLASS	0.510	0.004	0.718	0.000
LOCM	N	0.421	0.020	0.486	0.006
MHF	MIF			0.425	0.019
MHF	PPPC	−0.556	0.001	−0.436	0.016
n	N	0.437	0.016	0.556	0.001
N	SLOC			0.380	0.038
N	WMC	0.440	0.015	0.351*	0.057*
NCLASS	NMETH	0.690	0.000	0.659	0.000
NCLASS	SLOC	0.619	0.000	0.667	0.000
NMETH	SLOC	0.877	0.000	0.815	0.000
POF	RFC	0.429	0.018	0.136*	0.437*
RFC	WMC			0.742	0.000

Similarly, the names of each transformed variable indicate the transformation performed on it, - *i* indicates an inverse transformation, *e* an exponential transformation, and *l* a logarithmic transformation. The influential points identified from the analyses in the previous sections were excluded in the multiple regression models.

Table 8. Correlation results between independent variables (excluding outliers).

Variable 1	Variable 2	Pearson Correlation	Significance (2-tailed)	Spearson's Rho	Significance (2-tailed)
ACLOC	AMLOC			0.44	0.019
ACLOC	DIT			−0.38	0.046
ACLOC	MIF			−0.407	0.032
ACLOC	NMETH			0.53	0.004
ACLOC	SLOC			0.423	0.025
ACLOC	WMC	0.619	0.000	0.603	0.001
AMLOC	DIT	−0.411	0.030	−0.455	0.015
AMLOC	SLOC			0.448	0.017
AMLOC	WMC	0.392	0.039	0.528	0.004
AVPATHS	DIT	0.467	0.012	0.513	0.005
AVPATHS	N	0.451	0.061		
AVPATHS	PPPC			0.377	0.048
AVPATHS	RFC			0.427	0.023
CDENS	*N*			0.401	0.035
CDENS	WMC	0.455	0.015	0.469	0.012
DIT	NCLASS	0.469	0.012	0.653	0.000
DIT	POF	−0.399	0.035		
DIT	PPPC	0.408	0.031	0.488	0.008
LOCM	*N*			0.443	0.018
MHF	PPPC	−0.48	0.01		
MHF	RFC			−0.438	0.020
n	*N*	0.457	0.015	0.576	0.001
N	WMC	0.497	0.007	0.414	0.028
NCLASS	NMETH	0.678	0.000	0.639	0.000
NCLASS	SLOC	0.615	0.000	0.692	0.000
NMETH	SLOC	0.877	0.000	0.812	0.000
POF	RFC	0.434	0.021		
RFC	WMC			0.729	0.000

Six possible models were analyzed. Their results are summarized in the Table 9 in the Appendix. Not all the statistical results from the analysis are presented for all the models. The table shows, for each model considered, R^2, the significance level, and the significance level of the coefficient for each variable included.

Of all the models, Model 6 was considered to be the best. The detailed results of its analysis are presented in Tables 10 and 11 in the Appendix. The detailed results of analyses for all the rest of the models are omitted. Analyzing the correlations between the independent variables, it was found that most of the variables not present in Model 6 are accounted for due to the correlations with other variables in the model. However, some are not (e.g., AHF, AIF, COF and MIF). So, the model was analyzed again to see if adding them made any improvement. In all cases, the models with those added variables gave non-significant coefficients. So they are excluded. The final best model can be presented as follows:

$$\mathrm{MI} = 123.344\frac{1}{\mathrm{AMLOC}} + 1.944\ln(\mathrm{N}) + 2.166\frac{1}{\mathrm{CDENS}} - 5.692\frac{1}{\mathrm{DIT}} - 17.707\mathrm{e}^{\mathrm{MHF}} + 89.426$$

Table 9. Multiple regression models.

	Model 1	Model 2	Model 3	Model 4	Model 5a	Model 5b	Model 5c	Model 6
R^2	0.734	0.911	0.857	0.745	0.768	0.773	0.745	0.776
Signif	0.001	0.152	0.129	0.000	0.000	0.000	0.000	0.000
i_AMLOC	0.013	0.653	0.147	0.000	0.000	0.000	0.000	0.000
l_N	–	0.074	0.097	0.008	0.010	0.020	0.023	0.041
i_CDENS	0.165	0.088	0.139	0.083	0.051	0.174	0.082	0.043
e_MHF	–	0.116	0.305	–	0.159	–	0.082	0.052
i_DIT	0.185	0.164	0.512	–	–	–	–	0.100
l_n	–	0.165	0.432	0.084	0.162	0.092	–	–
i_AVPATHS	0.921	0.267	0.899	–	–	0.111	–	–
i_ACLOC	0.661	0.993	0.763	–	–	–	–	–
i_PPPC	0.545	0.357	0.750	–	–	–	–	–
l_SLOC	0.519	0.218	0.656	–	–	–	–	–
WMC	0.406	0.162	0.328	–	–	–	–	–
i_COF	–	0.732	0.661	–	–	–	–	–
i_LOCM	–	0.928	0.666	–	–	–	–	–
e_POF	–	0.806	0.670	–	–	–	–	–
i_MIF	–	0.807	0.664	–	–	–	–	–
i_AIF	–	0.673	0.619	–	–	–	–	–
RFC	–	0.407	0.864	–	–	–	–	–
i_NCLASS	–	0.208	–	–	–	–	–	–
i_NMETH	–	0.383	–	–	–	–	–	–
l_AHF	–	0.401	0.894	–	–	–	–	–

Table 10. Model summary for model 6.

					Change statistics				
Model	*R*	*R* square	Adjusted *R* square	Std. error of the estimate	*R*-Square	*F* Change	*df* 1	*df* 2	Sig. *F* change
1	.881a	.776	.723	4.994387	.776	14.577	5	21	.000

3.3.6. Validation Model 6, the final best model obtained from the analyses in Section 3.3.5, was considered for further validation. From the initial pool of 50 sample programs, 10 samples that were not considered in the previous analyses were selected for validation. The points corresponding those samples were checked to see if their range was compatible with the range of the cases used to build the model. The estimated value as indicated by the model was computed. The estimation error was also computed as the difference between the estimation given by the model and its true value. Several analyses were run on the results to check whether the estimation errors were similar to the original data and the new points used for the validation. The analysis showed a very similar behavior in the residuals for both data groups, with similar range, min and max, mean and standard deviation as shown in Table 12 in the Appendix.

Table 11. Coefficients of independent variables for model 6.

Model	Unstandardized coefficients		Standardized Coefficients	t	Sig
	B	Std. error			
1(constant) L_N	89.426	12.290		7.276	.000
J_AMLOC	1.944	.894	.232	2.174	.041
E_MHF	123.344	25.167	.600	4.901	.000
I_CDENS	−17.707	8.607	.219	−2.057	.052
I_DIT	2.166	1.004	.250	2.158	.043
a. dependent variable: MI	−5.692	3.304	.195	−1.723	.100

Table 12. Validation results (Case summaries)

Model	*N*	Mean	Median	Std. error of mean	minimum	Maximum	Std. deviation	variance
In model	27	−.1964	−.9314	.87082	−6.21	11.89	4.52491	20.475
Validation	10	−2.2124	−4.4587	2.01039	−9.06	10.62	6.35741	40.417
Total	37	−.7413	−.9815	.83367	−9.06	11.89	5.07103	25.715

Case Summaries
ERR_EST

The scatter-plot in Figure 1 shows the estimation errors for all the points, both the original and the validation data. It can be seen that they seem to have a similar behavior.

3.3.7. Summary of the analyses: Insight for designers and maintainers Based on the results of the analysis performed, the following implications of the design/code-level metrics on maintainability can be summarized. The different factors that affect maintainability of software are listed below. These factors should serve as heuristics to designers and maintainers willing to control maintainability of software from the design or implementation phases of the software lifecycle. The results presented earlier provide empirical evidence for the claims.

– MI shows a marked non-linear correlation with some of the variables. Particularly interesting are ALOC and SLOC. AMLOC shows a very high R^2 value, particularly using the inverse model. This is consistent with the common knowledge that individual functions/methods in a program should be kept short to improve program readability and maintainability. Also, there is a strong log-dependence of MI on SLOC. This is also very insightful since it is to be expected that maintainability decreases as program size increases. On the other hand, an increase in the relative program size is a better measure of increase in the complexity than the absolute program size. For example, the effect on maintainability of adding 100 lines to a 1000-lines program is clearly not the same as adding 100 lines to a 10000-lines program. Mathematically it can be shown that if the increase in complexity is a function of the relative increase in program size, the relationship

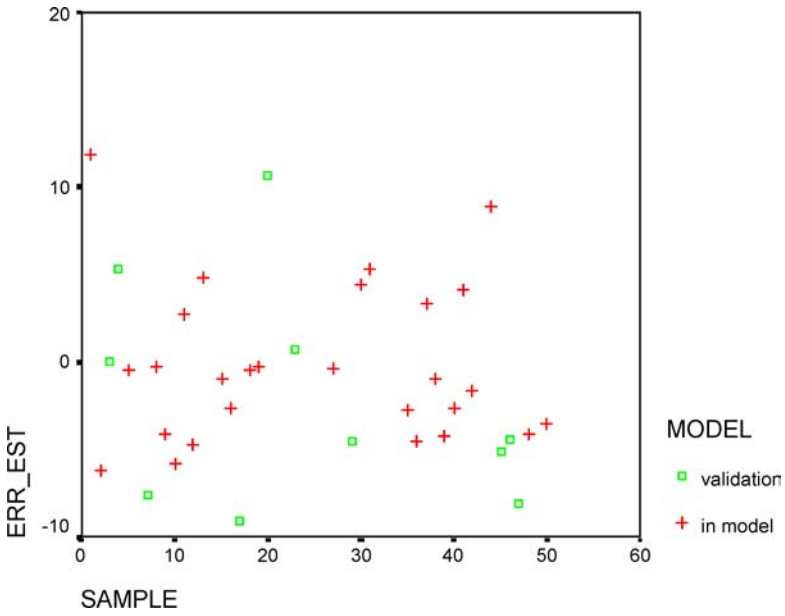

Figure 1. Scatter plot showing estimation errors for both the original and the validation data.

between both is a logarithmic function:

$$\begin{aligned} & \Delta \mathrm{MI} = k \frac{\Delta \mathrm{SLOC}}{\mathrm{SLOC}} \\ \Leftrightarrow \quad & d\mathrm{MI} = k \frac{d\ \mathrm{SLOC}}{\mathrm{SLOC}} \\ \Leftrightarrow \quad & \frac{d\ \mathrm{MI}}{d\ \mathrm{SLOC}} = k \frac{1}{\mathrm{SLOC}} \\ \Leftrightarrow \quad & \mathrm{MI} = k \log(\mathrm{SLOC}) + \mathrm{b} \end{aligned}$$

– Correlation analysis shows that the variables ACLOC, AMLOC, AVPATHS, CDENS, COF, DIT, n, N, PPPC, and WMC are relatively strongly influential on MI. The variables MHF, RFC, SLOC, and AIF are relatively moderately influential on MI. Whereas the other variables are negligibly influential on MI. Irrespective of the strength of the relationship of the above factors on MI, correlation analysis shows many interesting results that were not expected from the intuitive analysis. However strong or weak the relationship may be, the variables AHF, AIF, AVPATHS, COF, DIT, MIF, RFC show positive relationship with MI. In other words, the increase of attribute hiding, attribute inheritance, method inheritance, average depth of paths, coupling, depth of inheritance tree, increase in public/protected members, and response for classes, are likely to increase maintainability. However, it should be remembered that the strength of the relationship is dependent on the coefficient of correlation.

– Some variables show relatively low values of coefficients of correlation with MI, regardless of the model used (whether linear or non-linear), most notably AHF, AIF, COF, LOCM, MHF, MIF, NCLASS, NMETH, POF and RFC.
– It is particularly important to note that a good correlation (close to one) is not to be expected between any given variable and maintainability. If such a good correlation were found, it would mean that this single variable is able to explain most of the behavior of MI, and therefore the other variables would not be needed as predictors, which is, intuitively, not considered a possibility, if careful attention is paid to the definition of the variables. It is to be expected that more than one variable has an influence on the maintainability index, and therefore no single univariate regression should be a good predictor.
– From the definition of the independent variables used as possible predictors for MI, it can be gathered that some of them are likely to be highly correlated between themselves. This is indeed the case for the relationships between the following variables:

  - ACLOC, SLOC, and NCLASS
  - AMLOC, SLOC, and NMETH
  - AIF, DIT, and MIF
  - PPPC and MHF

  Therefore, some of them were ignored for further statistical analysis, since they provide redundant information and cannot help to better explain the behavior of MI.
– A multivariate, non-linear regression analysis shows that there are several possible models that can explain quite well the behavior of MI, with good significance values and a value of R^2 close to 1. The variables that intervene in these models are not necessarily the most intuitive ones (although AMLOC is always present, and able to describe most of the variation). This apparently strange selection of input variables has several reasons:

  - Due to the correlations between the input variables, some of them have to be discarded to avoid collinearity. The method applied by SPSS to perform this analysis tries to keep the "best" variables from a purely statistical point of view and is unable to ascertain whether they are meaningful or not in reality.
  - The particular sample in use might also have a considerably high influence on the particular selection of meaningful variables. It is quite possible that adding additional cases to the samples (or changing some of the cases with others) might change the variable selection, although the type of model in use will probably remain unaffected. This is due to the small size of the sample in use and the large number of parameters to be estimated for models using most or all of the input variables.
  - Since AMLOC is able to explain most of the variation of MI, the effect of the other variables is weakened, and small random variations of MI in different samples are amplified and can affect the result for these variables.

– Finally, it is to remark that the aforementioned strength of AMLOC as a predictor of MI, explaining most of its behavior, is in line with the well-known heuristics that tell that a function should not be more than two screens long and "spaghetti code" should be avoided. As obvious as it may sound, the relevance of this conclusion stands on the fact that it confirms that while programming, this is the most important measure to keep in mind. A programmer cannot pay attention to 20 different measures at the same time that

would increase the maintainability of his/her code. Knowing that the average method size will influence almost 60% of the final maintainability of the code can be really helpful. Thus, an important conclusion that can be drawn from this research is to "bound" the influence that the other 19 measures can have, and in that way, guide the practical work of a conscientious programmer.

Some of these reported relationships are counter-intuitive. In other words, based on the intuitive analysis, expected a strong relationship between many of these variables and MI. For instance, the positive influences of coupling on maintainability are counter-intuitive from the viewpoint of object-oriented theory. The results obtained in this study are quite interesting and can have several possible explanations, some of which are stated as follows. Since the analysis is done one variable at a time, and since MI is probably dependant on several of these at the same time, the results can be very sensitive on the particular sample set in use. This effect is increased by the use of a small sample size. In an ideal experiment, a sample set where all of the variables except one are kept fixed would be used to analyze the effect of a particular variable on the result. Obviously, this is neither practical nor possibly possible in this case. These claims are bounded by these experimental procedures, its limitations, and the samples used. Therefore, without sufficient experimental or theoretical evidence using other means, the cause of these unintuitive correlations cannot be stated conclusively. Regardless of the cause, this study goes an important step forward for setting the ball rolling for future investigation of these relationships using other techniques.

4. Limitations

There are a few limitations of this study that should be taken into consideration while interpreting the results. Some of these limitations are characteristics of studies of similar nature and are not uniquely attributable to this study.

- The results obtained in this study are based on the statistical analysis of data obtained from a collection of fifty systems, which have specific characteristics and behavior. Although the number of systems considered in the study is considerably larger than the previous ones (typically 2–8), the results should not be universally generalized for all systems.
- It is shown that most of the measures investigated have statistically significant relationship with maintainability. Such statistical methods provide only empirical evidence of relationships and do not demonstrate the actual scenario. Controlled experiments where certain measures can be varied in a controlled manner while keeping the others constant, could actually demonstrate the scenario. As usual however, it is difficult to undertake such controlled experiments in reality.
- This study calculates maintainability based on the widely accepted MI. This study assumes the validity of MI in measuring maintainability and does not separately prove it before using it as a maintainability indicator. MI has already been validated several times in the past (Coleman et al. 1994; Oman and Hagemeister 1994; Welker and Oman 1995; Welker et al., 1997).

A notable aspect of MI is that it is traditionally a non-object oriented metric. The applicability of this non-object oriented metric to object oriented systems in this paper can

be questionable sometimes. However, Welker et al. (1997) themselves successfully performed a practical case study to test the applicability of MI to object-oriented systems. They justified that object-oriented systems, like the procedural systems, are comprised of lines of code, lines of comments, operators and operands, and the number of paths through a module or system, which form the constituents of MI. Additionally, they state that code density, size, comments and execution logic are important in both non-object oriented and object-oriented systems. However, investigating this aspect is cause for future research.

- Maintainability can be affected by different factors. For example, applications presumably span many different technical and business areas, with different teams of developers having different competencies/expertise and using different standards. There may be other factors, such as the number of active developers, the development history of the systems, the type of the system (e.g., reusable library, compiler, end user software) that can potentially affect the maintainability of software systems. These factors, including many others, can affect maintainability, but are not considered in this study. However, because in actuality it is difficult to consider the enormously large number of factors that can affect maintainability, the study has been restricted to the indications of maintainability that can be obtained using MI. It should be reiterated that this study only provides probable indications of the effect of different measures on maintainability. Since it was not possible to consider all factors that affect maintainability in one model in this study, the results obtained here are worth verifying as part of future work using the models that consider other factors as well.
- It has been assumed that the design/code level metrics considered in this study calculate what they are supposed to measure. With the knowledge available, most of these metrics considered in this study have been validated in the past and their results can be found in the literatures cited in the introduction section of this paper.
- Finally, it should be noted that only code developed in C++ was considered in this study. The conclusions derived from using C++ code are believed to be valid for code developed in other object-oriented programming languages, e.g., Java. However, further research is called for to verify this proposition.

5. Conclusions and future work

This paper presented a study aimed at assessing and modeling the effect of different design/code level metrics on maintainability of systems. Early design/coding considerations and programming practices by human designers and maintainers of the systems based on the said factors from the early phases of system development could have an important effect on the overall quality of products. Several interesting results, listed in Section 3.3.7, were obtained that do not naturally follow from intuition. However, it is recommended that further validation studies be performed using other analytical techniques before the results are actually used in practice. Since the results are based on statistical predictions, one can expect that by abiding by the conclusions from these studies from the early phases of system development that the final product would be more maintainable and of better quality than a product in which maintainability is not monitored.

The guidelines presented in this paper are derived based on intuition and experimentation. First, it was intuitively predicted based on logic and the object-oriented theory how the

increase or decrease of a particular metric would affect maintainability. Then the results obtained from the intuitive analysis were experimentally verified using the data collected from different software samples. Although MI has received considerable attention in the past, as part of any future research, plans have been made to investigate other sources of maintainability data. Both the analysis methods considered in this study depend on several factors, the effects of which are worth studying in the future. While intuitive analysis obviously depends on the factors such as the proper understanding of the object-oriented principles and other psychological factors, experimental analysis depend on several factors such as, the systems considered, correctness of measurement and analysis tools, etc. It is thus encouraged to conduct further theoretical and experimental studies to validate the results obtained in this study. In addition, further research is called for to investigate the limitations of this study listed in Section 5.

Appendix

Statistical analyses were performed using the data collected by following the experimental procedures mentioned in Section 2.3. The Appendix lists the summary of the results of the statistical analyses described in Section 3.3.

Notes

1. We summarily list only the important works related to this paper, without providing too many details of the work. This also helps us to keep the readers' understanding of the materials discussed in the paper in the right perspective, while maintaining the brevity of the paper. Readers interested in following up their understanding about a particular piece of work can refer to the detailed reference of the work listed at the end of the paper.
2. However, some of the papers can be classified under more than one of these three categories.
3. This tool is widely used in the industries in quality measurement programs. To validate the correctness of the tool, we have manually computed the value of the different measures on small samples, and have compared their results with the results that are produced by the tool.
4. The reader should note that unlike the "model builder" set, where we have considered all the samples for our investigation, we have introduced to select the "validation set" to reduce biasness in select our validation samples.
5. It should be noted that in this study considered these 20 predictor vaiables, including the program length (N), the program vocabulary (n), and the number of lines of code (LOC), which were also used by Welker and Oman(1995) to define MI. However, along with these 3 variables, this paper considers 17 other important predictor variables to analyze their effects on maintainability. Even though these 3 variables were considered in the definition of MI, it was intended to observe the effects of these 3 variables in the presence of the other 17 variables. Even if individually they are likely to correlate highly with MI, that may not necessarily be the other variables. The reader should also note that Welker and Oman's definition of MI uses only the number of lines of code (LOC), On the other hand, the study presented in this paper considered the number of source lines of code (SLOC).
6. In each case, those points (samples) that statistically lie at an abnormal distance from the rest of the points were excluded from further consideration. This study restricted itself to only *statistically* investigating the outliers which occur rarely. However, this article does not present the investigation of the nature and behavior of the abnormal points, because the primary goal of this study was to provide some *general* design/coding guidelines to the practitioners, by investigating *only* the points that follow the general trend.
7. In an absolute sense, the results may not seem to be strong enough. The results are intuitively not surprising, if we consider the fact that the behavior of MI, in which case, the necessity of different factors to control MI would be unnecessary - that would be illogical.

References

Adamov, R. and Ritcher, 1990. A proposal for measuring the structural complexity of programs, *Journal of Systems and Software*. 12: 55–70.

Arishom, E., Briand, L.C., and Foyen A. 2003. Dynamic coupling measurement for object oriented software, Technical Report, TR-SCE-03–18, Carleton University.

Basili, V.R., Briand, L.C., and Melo W.L. 1996. A validation of object-oriented design metrics as quality indicators. *IEEE Trans. Soft. Engg.* 22: 751–761.

Binkley, A.B. and Schach S.R. 1997. Inheritance-Based Metrics for Predicting Maintenance Effort: An Empirical Study, Technical Report: TR-97–05, Computer Science Department, Vanderbilt University, Nashville, USA.

Brite e Abreu, F. and Carapuca, R. 1994. Object-oriented software engineering: Measuring and controlling the development process. In *Proceedings of the Fourth International Conference on Software Quality*, McLean, Virginia, USA.

Briand, L.C., Devanvu, P., and Melo, W. 1997. An investigation into coupling measures for C++. In *Proceedings of ICSE '97*, Boston, USA, pp. 47–52.

Briand, L.C., Daly, J., Porter, V., and Wust, J. 1998. A comprehensive empirical validation of design measures for object oriented systems. In *Proceedings of Fifth International Software Metrics Symposium*, Maryland, USA, pp. 246–257.

Briand, L.C., Morasca, S., and Basili, V.R. 1999. Defining and validating measures for object- based high-level design. *IEEE Transactions on Software Engineering* 25: 722–743.

Briand, L.C., Wuest, J., Daly, J., and Porter, V. 2000. Exploring the relationships between design measures and software quality in object-oriented systems, *Journal of Systems and Software* 51: 245–273.

Briand, L.C., Bunse, C., and Daly, J.W. 2001. A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs, *IEEE Transactions on Software Engineering* 27: 513–530.

Briand, L.C. and Wust, J. 2001. Modeling development effort in object-oriented systems using design properties. *IEEE Transactions on Software Engineering* 27: 963–986.

Briand, L.C., Bunse, C., and Daly J.W. 2002. A controlled experiment for evaluating quality guidelines on the maintainability of object oriented designs, *IEEE Transactions on Software Engineering* 27: 513–530.

Cartwright, M. and Shepperd, M. 2000. An Empirical investigation of an object-oriented software system, *IEEE Transactions on Software Engg* 26: 786–796.

Chidamber S.R., and Kemerer C.F. 1991. Towards a Metrics suite for Object-Oriented Design. In *Proceedings of Conference on Object-Oriented Programming: Systems, Languages and Applications, OOPSLA'91*, Phoenix, USA, pp. 197–211.

Chidamber, S.R. and Kemerer, C.F. 1994. A metrics suite for object oriented design, *IEEE Transactions on Software Engineering* 20: 476–493.

Chidamber, S., Darcy, D., and Kemerer, C. 1998. Managerial use of metrics for object-oriented software: An exploratory analysis, *IEEE Transactions on Software Engineering* 24: 629–639.

Coleman, D., Lowther, B., and Oman, P. 1994. Using metrics to evaluate software system maintainability, *IEEE Computer* 27: 44–49.

Daly, J., Brooks, A., Miller, J., Roper, M., and Wood, M. 1996. Evaluating inheritance depth on the maintainability of object-oriented software, *International Journal of Empirical Software Engineering* 1: 109–132.

Deligiannis, I., Shepperd, M., Roumeliotis, M., and Stamelos, I. 2003. An empirical investigation of an object-oriented design heuristic for maintainability, *The Journal of Systems and Software* 65: 127–139.

Deligiannis, I., Stramelos, I., Angelis, L., Roumeliotis, M., and Shepperd, M. 2004. A controlled experiment investigation of an object-oriented design heuristic for maintainability, *The Journal of Systems and Software* 72: 192–143.

El Emam, K., Benlarbi, S., Goel, N., and Rai, S.N. 2001. The confounding effect of class size on the validity of object-oriented metrics, *IEEE Transactions on Software Engineering* 27: 630–650.

El Emam, K., Benlarbi, S., Goel, N., Melo, W., Lounis, H., and Rai, S.N. 2002. The optimal clsss size for object oriented software, *IEEE Transactions on Software Engineering* 28: 494–509.

Halstead, M. 1997. *Elements of Software Science*, Elsevier, North Holland, New York.

Harrison, R., Counsell, S., Nithi, and R. 1991. An experimental assessment of the effect of inheritance on the maintainability of object-oriented systems, In *Proceedings of the Empirical Assessment in Software Engineering (EASE)*, Keele, U.K.

Harrison, R. and Counsell, S. 2000. The role of inheritance in the maintainability of object- oriented systems. In *Proceedings of the ESCOM Conference*, pp. 449–457.

Henderson-Sellers, B. 1996. *Software Metrics*, Prentice Hall, U.K.

Koru, A.G. and Tian, J. 2003. An empirical comparison and characterization of high defect and high complexity modules, *The Journal of Systems and Software* 67: 153–163.

Lanning, D.L. and Khoshgoftaar, T.M. 1994. Modeling the relationship between source code complexity and maintenance difficulty, *IEEE Computer* 27: 35–40.

Lorenz, M. and Kidd, J. 1994. Object-oriented software metrics, Prentice Hall, New Jersey, USA.

Li, W. and Henry, S. 1993. Object-oriented metrics that predict maintainability, *Journal of Systems and Software* 23: 111–122.

McCabe, T.J. 1976. A complexity measure, *IEEE Transactions on Software Engineering* 2: 308–360.

Misra, S.C. 2002. Heuristics for controlling quality of object oriented systems early in the software life cycle. In *Proceedings of the 12th International Conference on Software Quality*, Ottawa, Canada, Oct 28–30.

Misra, S.C. 2003. Tips on software quality management. In *Proceedings of the 6th International Conference on Business Information Systems (BIS)*, Software Engineering Track, Colorado, USA, June 4–6.

Misra, S.C. and Bhavsar, V.C. 2003. Relationships between selected software measures and latent bug density, international conference on computational science and its applications, *LNCS* 2667: 724–732.

Misra, S.C. and Bhavsar, V.C. 2003a. Measures of software system difficulty, *Software Quality Professional* 5: 33–41.

Oman, P. and Hagemeister, J. 1994. Construction and testing of polynomials predicting software maintainability, *Journal of Systems and Software* 24: 251–266.

Prechlelt, L., Unger B., Philippsen, M., and Tichy, W. 2003. A controlled experiment on inheritance depth as a cost factor for code maintenance, *The Journal of Systems and Software* 65: 115–126.

Rombach, H.D. 1987. A controlled experiment on the impact of software structure on maintainability, *IEEE Transactions on Software Engineering* SE-13: 89–94.

Rosenberg, J. 1998. A methodology for evaluating predictive metrics. In *Proceedings of the International Symposium on Software Metrics*, Bethesda, Maryland, USA, pp. 181.

Succi, G., Pedrycz, W., Stefanovic, M., and Miller, J. 2003. Practical assessment of the models for identification of defect-prone classes in object-oriented commercial systems using design metrics, *The Journal of Systems and Software* 65: 1–12.

Wake, S. and Henry, S. 1988. A model based on software quality factors which predicts maintainability. In *Proceedings of the Conference on Software Maintenance*, San Diego, California, USA, pp. 382–387.

Welker, K.D. and Oman, P.W. 1995. Software maintainability metrics models in practice, *Journal of Defence Software Engineering* 8: 19–23.

Welker, K.D., Oman, P.W., and Atkinson, G.G. 1997. Development and application of an automated source code maintainability index. *Software Maintenance: Research & Practice* 9: 127–159.

Subhas C. Misra is a doctoral student at Carleton University, Ottawa, Canada. Prior to this, he received his M.Tech. degree in Computer Science and Data Processing from the Indian Institute of Technology (IIT), Kharagpur, India and M.S. in Computer Science from the University of New Brunswick, Fredericton, Canada. He has several years of experience working on R&D projects in software engineering and quality engineering. He has worked in the research wings of reputed industries, such as Nortel Networks, Ottawa, Canada and the Indian Telephone Industries, India. He has published several technical papers in different international journals and conference proceedings.