Available online at www.sciencedirect.com
ScienceDirect
Information and Software Technology 48 (2006) 1046–1055
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Empirical assessment of the impact of structural properties on the changeability of object-oriented software

Erik Arisholm *

Simula Research Laboratory, Department of Software Engineering, Martin Linges v 17, Fornebu, P.O. Box 134, 1325 Lysaker, Norway

Received 2 December 2004; received in revised form 30 December 2005; accepted 5 January 2006
Available online 28 February 2006

Abstract

The changeability of software can be viewed as the quality of being *capable of change*, which among others implies that the task of changing the software requires little effort. It is hypothesized that structural properties of the software affect changeability, in which case measures of such properties can be used as changeability indicators.

Ways in which structural properties of the software can be measured are described and empirically validated based on data collected from an industrial Java development project. The measures are validated by using them as candidate variables in a multivariate regression model of the actual effort required to make modifications to the evolving software system.

The results suggest that some measures that combine existing structural attribute measures with a weighting factor based on the relative proportion of change in each class can explain a large amount of the variation in change effort. This constitutes initial, empirical evidence that the proposed measures are valid changeability indicators. Consequently, they may help designers to identify and correct design problems during the development and maintenance of object-oriented software.

Keywords: Object-oriented design metrics; Changeability; Change effort

1. Introduction

The object-oriented approach to software development is becoming mainstream. So are evolutionary and incremental software development processes, such as the unified process and XP, in which the software requirements and the software design supporting the requirements are allowed to evolve towards a final system. Both of these trends are the result of attention being paid to a variety of claims about how object-oriented programming and evolutionary/incremental development processes may improve the development and quality of software.

In order to help managers and developers achieve such improvement, a large portion of empirical research in the OO research arena has been involved with the development and evaluation of quality models for OO software [14,20]. Existing results suggest that there are important relationships between the structural properties of object-oriented and external quality attributes, such as fault proneness and development effort. The main motivations are to be able to use structural attribute measures as surrogates for important external software quality attributes, which often are difficult to measure directly. Once evaluated, the quality models can be used to monitor the evolution of a software system, facilitate design decisions, etc.

According to Webster's Revised Unabridged Dictionary, *changeability is the quality of being capable of change*. From this definition, one may deduce two important consequences:

- Being capable of change implies that the task of changing the software requires little *effort*.
- Being capable of change implies a capability of avoiding a gradual *decline* from a sound or satisfactory state to an unsatisfactory state, in which case changes require more effort than they ought to.

* Tel.: +47 67828302; fax: +47 67828201.
E-mail address: erika@simula.no

0950-5849/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2006.01.002

Many factors such as those related to the structural properties of a system, quality of documentation, tool support and developer skills may affect changeability. The primary goal of this paper is to assess the extent to which a small subset of these factors, related to certain structural properties of a system, affects changeability. Two alternative ways to measure the structural properties are proposed and evaluated on the basis of project data from a commercial Java development project. Specifically, this paper investigates whether using historical data available during the development project can be used to construct better changeability indicators: some of the proposed measures combine existing, class-level, structural attribute (SA) measures with change data to construct measures that are called *change profile* (CP) measures. CPs measure the structural attributes of all classes in a software system, but *weigh* the resulting structural measurements by the proportion of change in each class. The rationale behind this approach is that the changeability of the software is reflected by the effort to perform actual changes. Consequently, the changeability (as indicated by the CP measures) depends on the structural properties of the parts of the software where changes have been made. Unlike the SA measures, the CP measures thus require access to change history data. This means that the system must have undergone changes before CP measurements can be made, which in turn means that the CP measures cannot be used as 'early' indicators of changeability. Instead, their intended use is to indicate trends in changeability during the evolutionary development and maintenance of a software system.

This paper is primarily related to research that has studied the impact of structural properties on external software quality attributes, e.g. [2,6,8–11,13,19,21,22,24,25,29,32,36–38,42,44,46–49]. However, as already discussed, once an empirical relationship between the structural property measures and change effort has been established, the measures can then potentially be used as indicators of *changeability decay*, by studying trends in the structural property measurements as the software evolves. Thus, one important future application of this research is related to research that have studied software quality decay and proposed ways to measure such decay, e.g. [3,4,26,31,33–35,41,43,45].

The remainder of this paper is organized as follows. Section 2 defines the measures. Section 3 describes the case study and the results. Section 4 discusses threats to validity. Section 5 relates the results to existing studies and discusses implications of the results. Section 6 concludes.

2. Definition of measures

This section describes the proposed measures for assessing the changeability of object-oriented software. Earlier versions of the measures described in this paper were published in [3,4]. Those measures have since improved in precision as a result of experiences with its use in several subsequent studies [1].

Two alternative approaches to measuring the structural properties of object-oriented software are described: *Structural Attribute (SA)* measures and *Change Profile (CP)* measures. The latter are based on the former. Both SA and CP measurements can be made at certain points in time when the software is in a consistent state. In the following discussions, these points in time are denoted as *snapshots* that define, in precise terms, *when* a measurement is made. The SA measures quantify various dimensions of the structure of the software at the class level (such as class size and class-level coupling) at certain points in time, say snapshots $1,\ldots,n$. The class-level measures are then aggregated to the system-level by calculating totals and average system-level values of each measure. Conversely, the class-level CP measures are calculated by multiplying class-level SA measures for a given snapshot j with the proportion of change in each class since snapshot $j-1$. The class-level CP measures are then aggregated to the system level by summing the measures across all classes.

2.1. Structural attribute (SA) measures

Only a few, and relatively simple, measures that capture some important and intuitive dimensions of an object-oriented structure have been selected.

It is commonly believed that size is a major contributor to 'complexity'. Consequently, two dimensions of the overall system size are measured: System Size (*SS*) in non-commented lines of code and Class Count (*CC*). Furthermore, two measures of the class size are measured: Class Size (*CS*) in non-commented lines of code and Method Count (*MC*).

In addition to size, several measures of coupling are included. As pointed out in [17], each coupling measure should ideally account for one and only one dimension of coupling. In this paper, the static, class-level coupling measures defined in [15] are used, because they (1) cover many dimensions of coupling, (2) have been shown to be theoretically sound, and (3) have already undergone extensive empirical testing [13,15,18,19].

Other structural properties related to, for example, cohesion and inheritance could also be considered. However, based on the existing empirical results, size and coupling seem to be the most consistent indicators of factors related to changeability. The impact of cohesion is currently less understood and it is more difficult to measure precisely [16]. Furthermore, although several studies have shown that inheritance may pose serious problems for factors related to changeability, such as fault proneness and maintainability [23,29,38], the use of inheritance in many object-oriented systems is limited [23,30,38]. Consequently, the investigation focuses on coupling and size measures. Future extensions should also consider cohesion and inheritance measures.

The selected measures distinguish between many dimensions of size and coupling. A precise mathematical

definition and justification for the measures are given in [15]. Tables 1 and 2 summarize the measures. The dimensions and the notation of the coupling measures are as follows:

- Type of class coupling: coupling with *Ancestor* classes (A*xxxx*), coupling with *Descendent* classes (D*xxxx*), or coupling with classes outside the inheritance hierarchy, i.e. *Other* classes (O*xxxx*),
- Type of interaction: Method–Method (*x*MM*xx*) or Method–Attribute (*x*MA*xx*) interactions
- Direction of coupling: Import Coupling (*xxx*IC) or Export Coupling (*xxx*EC)
- Stability of server: distinguishes import coupling to non-library (*xxx*IC) and library classes (*xxx*IC_L)

The method–attribute (*x*MA*xx*) coupling measures are not described in Briand's coupling framework. For certain applications, they may be important because they measure direct access to class attributes by other classes than the defining classes, something that is commonly perceived as poor design practice.

The system-level SA measures are defined in Table 2. Here, C_j is the set of classes in the system at snapshot *j*. The term *XX* in Table 2 denotes any of the measures defined in Table 1.

2.2. Change profile (CP) measures

Table 3 defines the CP measures, *XX*_CP. For each class that exists in snapshot *j*, $c_i \in C_j$, the proportion of change $\mathrm{CP}_j(c_i)$ is calculated as the lines of code added and deleted in that class since snapshot $j-1$, divided by the total number of lines of code added and deleted in all classes existing in snapshot *j* ($ChangeSize_j$). Finally, the sum of the structural attribute measure $XX_j(c_i)$ weighted by the proportion of change $\mathrm{CP}_j(c_i)$ produces the corresponding system-level change profile measure, XX_CP_j. There are some important and intuitively appealing properties of the CP measures:

- Classes that have been deleted since snapshot $j-1$ will not be included in the measurements for snapshot *j*.
- If only one class *c* has changed since snapshot $j-1$, all change profile measures will be equal to the corresponding class-level structural attribute measures for that class in snapshot *j*, e.g. $\mathrm{CS\_XP}_j = \mathrm{CS}_j(c)$.
- If all classes are changed by an equal amount, all change profile measures will be equal to the corresponding system-level averages of the corresponding structural attribute measure, e.g. $\mathrm{CS\_CP}_j = \mathrm{Avg\_CS}_j$.

3. Case study

This section describes the empirical study conducted to validate the proposed measures.

3.1. Rationale and objectives

This case study evaluates the extent to which the SA and CP measures can explain variations in change effort in an evolutionary development project. The measures were used as candidate independent variables in a multivariate regression model of the dependent variable *change effort* (mea-

Table 1
Class-level SA measures

Measure name	Description
CS(*c*)	Number of non-commented source lines of code (SLOC) for the class *c*
MC(*c*)	Number of implemented methods in a class *c*.
OMMIC(*c*)	Number of static method invocations from a class *c* to non-library classes not within the inheritance hierarchy of *c*
OMMEC(*c*)	Number of static method invocations to a class *c* from non-library classes not within the inheritance hierarchy of *c*
OMMIC_L(*c*)	Number of static method invocations from a class *c* to library classes
OMAIC(*c*)	Number of direct accesses by class *c* to attributes defined in non-library classes not within the inheritance hierarchy of *c*
OMAEC(*c*)	Number of accesses to attributes defined in class *c* by non-library classes not within the inheritance hierarchy of *c*
OMAIC_L(*c*)	Number of direct accesses by class *c* to attributes defined in library classes not within the inheritance hierarchy of *c*
AMMIC(*c*)	Number of static method invocations from a class *c* to non-library ancestor classes of *c*
DMMEC(*c*)	Number of static method invocations to methods implemented in class *c* from descendants of *c*
AMMIC_L(*c*)	Number of static method invocations from a class *c* to library ancestor classes of *c*
AMAIC(*c*)	Number of direct accesses by class *c* to attributes defined in non-library ancestor classes of *c*
DMAEC(*c*)	Number of accesses to attributes defined in class *c* by descendants of *c*
AMAIC_L(*c*)	Number of direct accesses by class *c* to attributes defined in library ancestor classes of *c*

Table 2
System-level SA measures

Name	Definition	Description
CC_j	$\lvert C_j \rvert$	The number of non-library classes C_j in the system for snapshot *j*
Tot_XX_j	$\sum_{i=1}^{CC_j} XX_j(c_i)$	The sum of the structural attribute measure *XX* (Table 1) for snapshot *j* for the non-library classes C_j in the system
Avg_XX_j	TotXX_j/CC_j	The average (system-level) structural attribute measure *XX* (Table 1) for snapshot *j* for the non-library classes C_j in the system

Table 3
System-level CP measures

Name	Definition	Description
Change Size$_j$	$\sum_{i=1}^{CC_j} \text{SLOCAdd}(c_i) + \text{SLOCDel}(c_i)$	The total amount of change from snapshot $j-1$ to snapshot j. CC_j is the number of classes, defined in Table 2
CP$_j$(c)	$\frac{\text{SLOCAdd}(c) + \text{SLOCDel}(c)}{\text{ChangeSize}_j}$	The class-level change profile, defined as the amount of change (in SLOC added and deleted) in class c from snapshot $j-1$ to snapshot j divided by the total amount of change from snapshot $j-1$ to snapshot j
XX_CP$_j$	$\sum_{i=1}^{CC_j} XX_j(c_i) \times \text{CP}_j(c_i)$	The sum of the class-level structural attribute measure XX_j (Table 1) weighted by the class-level change profile CP$_j$(c). CC_j is the number of classes, defined in Table 2

sured in person-hours spent on each change). For this purpose, a comprehensive amount of project data from the case study was collected at the change task level.

The underlying rationale for the validation approach is that, assuming that the resulting model can explain variations in an important external quality attribute, it may be concluded that the measures are indicators of that attribute. Hence the measures are *validated* in the sense that statistically significant relationships between the measures and the external quality attribute under consideration have been established.

More specifically, in this case study, the external quality attribute under consideration is the amount of variation in change effort that can be explained by variations in structural properties of the software, that is, the changeability of the software as a function of its structural properties. It is important to emphasize that the objective is thus *not* to build effort prediction models to be used for effort estimation purposes. Such models would clearly also need to include many more factors than just the structural properties of the software. Here, the objective is to assess the impact of structural properties on change effort as a means to evaluate whether they are useful indicators of changeability.

3.2. Project under study

The case study was initiated in conjunction with a product development project in the company Genera. The module studied was an independent part of the *Genova* tool and provided run-time support for automatically generated source code based on the UML and Dialog Models. Further descriptions of the tool can be found in [7]. The development of the module was organized as a separate development project, lasting approximately 5 months and involving three developers (one assigned to C++ and two assigned to Java) in addition to project management. The project was evolutionary in nature. During the five months this system was studied, existing, changed and completely new requirements were incorporated in an incremental fashion, with iterative analysis, design, code and test activities. During the course of the development project, 39 changes were performed. Sixteen of the changes implemented new requirements, nine were corrections of design and code faults, seven were restructuring changes, whereas the remaining changes were related to performance improvements, adaptations to reuse and library adaptations. Further details of the changes are described in [1]. Two versions (increments) of the module were released within the five month time span.

3.3. Data collection

3.3.1. Collection of change effort data

Each change to the system was tagged in the configuration management system with a unique change ID number. Each time a class was 'checked in' to the configuration management system, the change ID was attached to the comment field of the change record in the configuration management system. In addition, the developers provided a short textual description of each change task and reported the total number of hours spent on each change task (including analysis, design, coding, testing and documentation activities that could be allocated to individual changes). The developers used a simple database application to report the change and effort data of every change task on a daily basis and 'closed' the task report as soon as the change task was completed. After a task was closed, the developers were interviewed to check that the reported data was accurate. Potential threats are discussed in Section 4.

3.3.2. Collection of structural attribute measures and change profile measures

The change IDs stored in the configuration management system were used to reconstruct snapshots of the code as it appeared just after each change, using the query language provided in the Clear Case configuration management system. More specifically, the snapshot for change j was derived by selecting the latest versions of the each file that was checked in using a change ID $= j$. All other files, i.e. those not modified by change j, were selected based on the timestamp of the last check-in time for a file changed in change j. To calculate the CP measures, the class-level *delta* (in SLOC added and deleted) of each change was collected. Further details on how these calculations where performed using the ClearCase configuration management system are provided in [1].

3.3.3. Dead code

Examples of dead code are classes that are no longer in use or classes that are present for possible future use. Such

dead code has the potential to generate inaccurate values for the SA measures. Consequently, for each snapshot, files that were not in use were removed before the measures were calculated. The developers verified that the files assumed to be 'dead' by the researcher in fact were. Note that the CP measures are less sensitive to dead files because only structural attributes of code being changed are accounted for, and if files are changed, it may be reasonable to assume that the files are not dead.

3.4. Validation method

To validate the measures, they were used as candidate explanatory variables in a multiple linear regression model on change effort. Stepwise, multiple linear regression was performed to determine a subset of the measures that explained a large portion of the variance in change effort, but in such a way that the resulting coefficients of each variable were significantly different from zero. The criterion for entry into the model was that the coefficient had a p-value below 0.2. The criterion for staying in the model was that the coefficient had a p-value below 0.1. Finally, only coefficients with p-values below 0.05 were included in the final models. To compare the explanatory power of the SA and CP measures, the validation was performed in three steps:

- Including only the (non-zero and non-constant) SA measures as candidate covariates
- Including only the (non-zero and non-constant) CP measures as candidate covariates
- Including all (non-zero and non-constant) SA and CP measures as candidate covariates

The goal of this validation procedure was to test the hypotheses that each candidate variable was a significant explanatory variable of change effort when also considering the simultaneous effect of other candidate variables. However, this procedure automatically 'snoops' through many candidate models. Thus, the model selected may fit the data 'too well', in that the procedure can look at many variables and select those that, by pure chance, happen to fit well. To lessen this potential threat, the cross-validated R-squared was calculated to evaluate the explanatory power of the resulting models. Whereas the usual multiple regression estimate of R-squared increases whenever more parameters are added to the model, this problem does not occur with the cross-validated R-squared. To calculate the cross-validated estimate of R-squared, the data is split in I subsets. For $i = 1, 2, \ldots, I$, the least squares fit and the mean are calculated for all cases but the ith subset. The regression model and the mean are each used to predict the observations in the ith subset. Note that it is possible for the cross-validated estimate of R-squared to be negative, especially when overfitting, because the regression model is competing with the mean. This would indicate that the mean is a better model than the regression. In this paper, the cross-validated R-Squared was calculated in BLSS, using the number of subsets I equal to the number of data-points n.

For linear regression, the hypothesis tests on the coefficients are based on a number of conditions:

- The expected error mean must be zero
- Homogeneous error variance
- Uncorrelated errors (e.g. no serial correlation)
- Normally distributed errors

Although linear regression is quite robust with regard to these requirements, gross violations of the underlying regression model assumptions should be detected [27]. For the resulting models, these assumptions were checked through plots of the residual errors.

3.5. Selection of changes

For the validation purposes, only 10 of the available 39 changes reported by the developers could be used. The other changes were not included for one of the following reasons:

- they were primarily related to documentation or administration (e.g. makefiles or configuration management)
- they included changes to C++ code (no C++ parser was implemented)
- they had not been tagged correctly (or there were apparent inconsistencies).
- they were mixed with other changes, making it difficult to derive precise snapshots.

Of the 10 selected changes, the first six occurred during the first increment. All of these six changes implemented new functionality and were performed by the same developer. Four changes occurred during the second increment, and were performed by a second developer. The first change in increment two (change number 7) was a large restructuring change. The remaining three changes were corrective.

3.6. Results

Table 4 shows the measures for each of the selected 10 changes. The restructuring change (change 7) was considerably larger than the other changes, and was considered as an outlier in the regression analysis (see Fig. 1). Note that for the system-level SA measures, the import coupling is always equal to the corresponding export coupling value. For example, *TotOMMIC* equals *TotOMMEC*. Furthermore, some measures of coupling within inheritance hierarchies were constant or zero and were therefore not included as candidate covariates in the stepwise regression. The final set of variables is indicated in bold in Table 4.

The results are summarized in Table 5. None of the candidate SA measures were included by the stepwise variable selection procedure, regardless of whether the CP measures

Table 4
Change effort, Total SA, Avg. SA and CP measures for the selected changes

Change number	1	2	3	4	5	6	7	8	9	10
Effort (hours)	1	2	31	26	7	10	150	18	6	13
Change size	153	42	308	9	209	164	1421	111	106	72
CC	33	35	35	35	35	35	47	47	47	47
TotCS	399	406	531	701	610	662	946	961	932	965
TotMC	118	121	139	168	155	167	200	200	193	203
TotOMMIC, TotOMMEC	30	34	54	78	66	75	66	72	67	76
TotOMMIC_L	67	66	81	102	90	94	162	167	167	174
TotOMAIC, TotOMAEC	66	66	106	168	119	150	178	190	164	199
TotOMAIC_L	5	5	5	5	5	5	15	15	15	15
TotAMMIC, TotDMMEC	2	5	5	15	15	15	0	0	0	0
TotAMMIC_L	1	1	1	1	1	1	1	1	1	1
TotAMAIC, TotDMAEC	0	0	0	0	0	0	0	0	0	0
AvgCS	12.1	11.6	15.2	20.0	17.4	18.9	20.1	20.4	19.8	20.5
AvgMC	3.6	3.5	4.0	4.8	4.4	4.8	4.3	4.3	4.1	4.3
AvgOMMIC, AvgOMMEC	0.9	1.0	1.5	2.2	1.9	2.1	1.4	1.5	1.4	1.6
AvgOMMIC_L	2.0	1.9	2.3	2.9	2.6	2.7	3.4	3.6	3.6	3.7
AvgOMAIC, AvgOMAEC	2.0	1.9	3.0	4.8	3.4	4.3	3.8	4.0	3.5	4.2
AvgOMAIC_L	0.2	0.1	0.1	0.1	0.1	0.1	0.3	0.3	0.3	0.3
AvgAMMIC, AvgDMMEC	0.1	0.1	0.1	0.4	0.4	0.4	0.0	0.0	0.0	0.0
AvgAMMIC_L	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AvgAMAIC, AvgDMAEC	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
CS_CP	44.3	12.0	96.9	148.3	88.0	153.0	65.3	70.8	50.5	49.7
MC_CP	12.3	2.9	22.3	35.4	23.7	38.6	11.6	13.2	10.8	7.7
OMMIC_CP	4.3	2.6	16.6	23.0	16.5	25.9	5.3	6.7	4.9	3.0
OMMEC_CP	0.3	0.3	1.8	0.4	0.3	0.0	1.8	1.9	1.3	2.1
OMMIC_L_CP	4.0	5.4	11.9	17.0	13.1	17.8	16.1	12.5	9.9	9.6
OMAIC_CP	17.0	0.0	42.7	75.8	34.8	78.0	8.8	31.1	7.0	19.5
OMAEC_CP	0.1	0.0	1.8	0.0	0.0	0.0	0.4	0.6	0.0	1.1
OMAIC_L_CP	0.6	0.0	0.0	0.0	1.1	0.5	1.6	0.0	0.4	0.3
AMMIC_CP	0.3	2.0	0.1	0.0	3.1	1.2	0.0	0.0	0.0	0.0
DMMEC_CP	1.5	0.0	3.4	11.7	8.0	12.4	0.0	0.0	0.0	0.0
AMMIC_L_CP	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AMAIC_CP	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DMAEC_CP	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

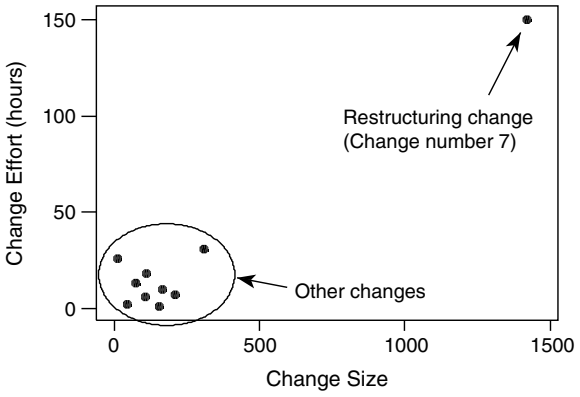

Fig. 1. Plot of change effort versus change size. The restructuring change is treated as an outlier in the regression model validation.

were also included as candidates or not. Thus, none of the SA measures could explain any significant amount of variation in change effort. The final model consisted of OMAEC_CP and CS_CP. OMAEC_CP and CS_CP are significant explanatory variables of change effort in the data set, and explain almost 80% of the variance of the change effort for the given changes. Since the regression is based only on nine observations, the cross-validated *R-Sq*(*cross*) provides a more accurate estimate of the prediction ability of the model (i.e. for new observations) than does the ordinary *R-Sq*. The cross-validated *R-Sq*(*cross*) indicates that slightly more than 50 percent of the variation in change effort is explained by the model. Consequently, changes involving classes with higher attribute export coupling and class size are associated with higher change effort.

The results suggest that the CP measures are better indicators of changeability than are the SA counterparts. It may also be interesting to note that for the set of changes investigated, the selected CP measures explain the variance in change effort considerably better than the number of lines of code added or deleted (*ChangeSize*). If larger changes had been included, *ChangeSize* would probably also play an important part in explaining change effort.

Fig. 2 shows the residual model diagnostic for the regression model including OMAEC_CP and CS_CP as covariates. No serious violations of the conditions for valid interpretation of the regression model (Section 3.4) were found. According to the Anderson-Darling normality test, there is insufficient support for the hypothesis that the residuals are not normally distributed ($p = 0.45$).

Table 5
Resulting regression models for change effort in the Genera case study

Variables	Coefficient	Coefficient *p*-value	*R*-Sq (%)	*R*-Sq (cross)
Intercept	9.083	0.210	5.7	Negative
ChangeSize	0.027	0.536		
Intercept	−2.021	0.641	77.5	51.5%
OMAEC_CP	10.546	**0.013**		
CS_CP	0.131	**0.022**		

4. Threats to validity

This section discusses the most important threats to validity, related to construct validity, internal validity and external validity.

A potential threat in this study is the construct validity of the dependent variable *ChangeEffort*. The effort data reported by the developers are prone to noise due to non-productive time during a typical work day, such as lunch-breaks, telephone calls and other interruptions. To address this threat, the developers were asked to estimate the actual change effort in such a way that only the productive time spent on each task was included. To further increase the accuracy of the reported data, the developers updated the effort data related to a given change task at the end of each day, even in cases where the task was not yet completed. Furthermore, the developers were asked to only work on one task at the time, whenever possible. Finally, after a task was closed, the responsible developer was interviewed to ensure accuracy of the reported data. These measures probably reduced the threats to construct validity, but clearly there might still be inaccuracies due to the somewhat subjective procedure of estimating the productive time spent on each task.

Using variable selection heuristics such as stepwise regression, one can often find statistical relationships between variables. This is not the same as establishing a cause – effect relationship [40]. The apparent relationships could just be the result of a common underlying cause. Furthermore, given the relatively large number of candidate explanatory variables that were allowed to enter the regression models, there is a possibility that the apparent (statistically significant) relationships between some of the CP measures and change effort is mainly due to chance, often referred to as 'shotgun correlation' [28]. Furthermore, the selected changes in this study are similar in size, so the model obtained may not be valid if one included changes with large *differences* in change size. Although care was taken to ensure statistically valid results, more data points are required to fully address the abovementioned threats to internal validity.

With regards to external validity, it is unclear exactly when and how the results might be applicable in other development projects, with different developers, different tools, etc. Thus, the results should be interpreted with caution. More studies are required to generalize the results. This is also discussed further in Section 5.

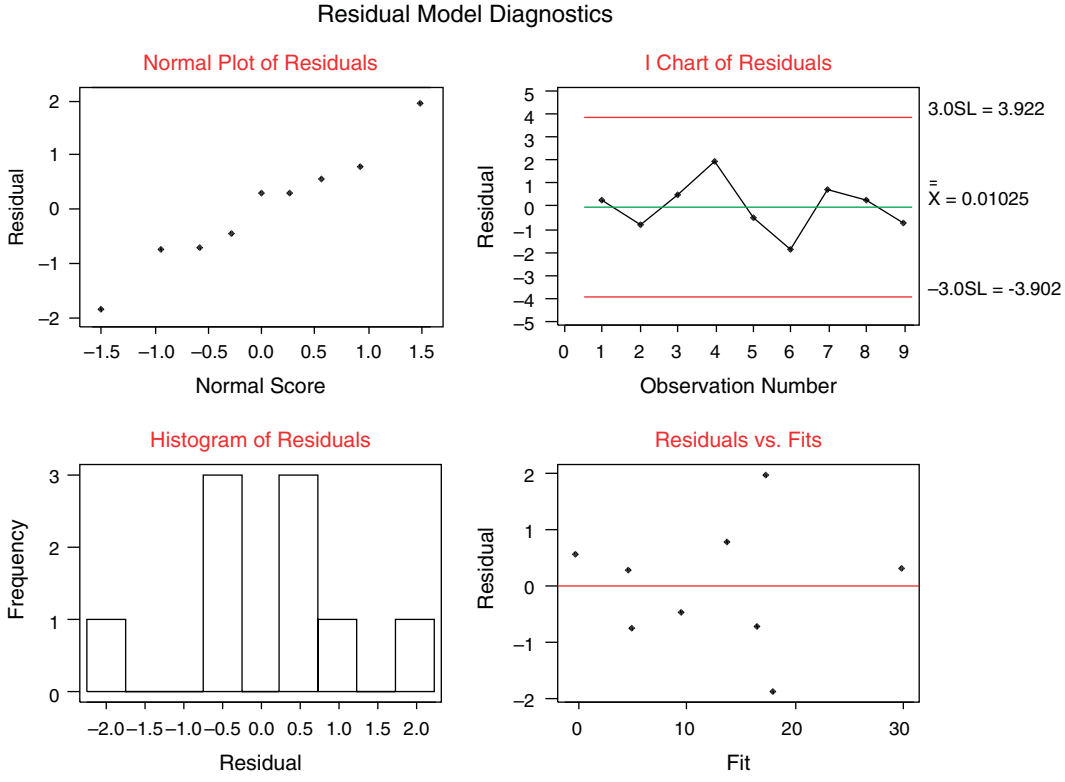

Fig. 2. Residual diagnostics of the final model.

5. Discussion and related work

A large portion of empirical research in the OO research arena has been involved with the development and evaluation of quality models for OO software. The immediate goal of this research is to relate structural attribute measures intended to quantify important characteristics of object-oriented software, such as size, polymorphism, inheritance, coupling, and cohesion, to external quality indicators such as fault proneness, change impact, reusability, development effort and maintenance effort. There is a growing body of results that indicates that measures of structural attributes such as class size, coupling, cohesion and inheritance depth can be reasonably good indicators of development effort and product quality [9,13,22,25,36,42]. See also surveys in [14,20].

The CP measures were first proposed in [3,4], but the empirical validation was inconclusive due to a lack of proper effort data. An approach similar to the CP measures was then proposed in [12], the difference being that a scenario-based approach was used to estimate the expected, future amount of change in each class instead of using the actual change history. This approach thus complements the CP measures, and is likely to be a good alternative when change history data is not available. No empirical validation to external quality attributes was performed, however. In the remainder of this section, we focus our attention on a selection of studies that have validated coupling and size measures as indicators of the effort required to develop and maintain object-oriented software.

One early, frequently cited, paper that investigates how structural attributes of object-oriented software affect maintenance effort is [42]. In that study, the number of lines of code changed per class was the dependent variable, whereas the 'CK' measures defined in [25] were the independent variables. Extending the above study, we showed that *static* and *dynamic* coupling can be combined to improve the accuracy of such prediction models even further [6]. However, the above studies used a surrogate measure (lines of code added and deleted) instead of the actual change effort as the dependent variable.

Only a few studies have attempted to evaluate the direct relationships between structural properties and actual development or maintenance effort [11,13,26,35,44]. The results in [11] suggest that a significant impediment to maintenance is the level of interaction (i.e. coupling) between modules. Modules with low coupling required less maintenance effort and had fewer maintenance faults and fewer run-time failures. In [26], an exploratory case study indicated that high coupling between objects (high value of *CBO*) was associated with lower productivity, greater rework and design effort, after adjusting for size differences and after controlling for the effects of individual developers. A model for effort prediction of the adaptive maintenance of object-oriented software was presented in [35]. The validation showed that some object-oriented metrics might profitably be employed for effort estimation. In [44], a set of size and complexity measures were found to be good predictors of development effort at the class level. These results are supported by the study presented in [13], which showed that size measures were good predictors of class-level development effort, whereas only limited improvements in effort estimation accuracy were obtained by including coupling measures in addition to size. One possible explanation for the predominance of size measures as significant explanatory variables of change effort is that large classes implement more functionality than do small classes, and are therefore more difficult to understand and may break more easily, e.g. when changed by an inexperienced developer. Size measures are also surrogates for many underlying dimensions of the cognitive complexity of a class [32]. For example, large classes may have higher coupling and lower cohesion than have small classes. The degree of correlation may vary among the measures [19], and is also project-specific. This may explain why only size measures were significant in some studies, whereas coupling measures were also significant in others. Overall, existing studies suggest that measures of size and coupling are reasonably good indicators of the cognitive complexity of a class, which in turn affects the effort required to change the classes.

The results presented in this paper support existing work in the sense that changes involving large classes and classes with high coupling require more effort than do changes that involve smaller classes and classes with low coupling. However, this relationship was only demonstrated through the use of the CP measures, which combine existing coupling and size measures with a weighting factor based on the proportion of change in each class. None of the more traditional SA measures, which also have been used in existing validation studies, were significant explanatory variables of change effort. One reason for this apparent contradiction is illustrated in controlled experiments on the relationship between structural properties and maintainability [5,46]: the degree to which software properties affect external quality such as maintainability depend on other factors, such as programmer ability. It may also depend on exactly how the dependent and independent variables are defined (including the granularity of the measures), as well as on the objective of measurement. For example, the aim of the study reported in [26] was to assess whether the proposed structural attribute measures could be used to identify high effort and low productivity classes, not to build accurate prediction models. In contrast, the main goal in [13] was to investigate the extent to which structural properties could be used as cost drivers in effort estimation models. These differences had implications for both statistical analyses techniques being employed and the selection of measures. Unfortunately, such differences make it difficult to perform the sort of precise meta-analyses that are required in order to draw more general conclusions beyond single case studies such as the one presented in this paper.

The above discussion also illustrates a possible reason for why the industrial use of structural measures as quality

indicators is very limited [39], despite the extensive research efforts. Although individual results such as that presented in this paper demonstrate that measures of structural properties may be valid quality indicators, they seem to be insufficient to convince practitioners that the potential benefits of using the measures outweigh the cost of collecting them. Future studies should focus, therefore, on enabling meta-analysis across studies and performing cost-benefit analyses.

6. Conclusions

This paper assessed whether structural properties of object-oriented software can be used as indicators of the changeability of the software. Two alternative measurement approaches, termed change profile (CP) and structural attribute (SA) measures, respectively, were proposed and then evaluated using detailed project data from a commercial development project. The results from the case study showed that some of the CP measures are statistically significant explanatory variables of change effort, whereas none of the SA measures are. The fact that the CP measures quantify properties that are likely to affect the effort required to perform changes suggests that they are valid changeability indicators. The measures included in the obtained regression models can thus be used to identify design problems and evaluate the effect of design modifications on changeability during the evolutionary development of object-oriented software. However, it should be noted that the results are based on a single case study. Further studies are required in order to generalize the results beyond the scope of this study.

Acknowledgements

Thanks to Stein Grimstad, Dag Sjøberg, Jon Skandsen and Chris Wright for their contributions to this work. The research was funded by The Research Council of Norway through the industry-project SPIQ (Software Process Improvement for better Quality).

References

[1] E. Arisholm, Empirical assessment of changeability in object-oriented software, University of Oslo, Oslo, PhD Thesis, ISSN 1501-7710, No. 143, Unipub forlag, 2001.
[2] E. Arisholm, Dynamic coupling measures for object-oriented software, Proceedings of the Eighth IEEE Symposium on Software Metrics (METRICS'02), 2002, pp. 33–42.
[3] E. Arisholm, D. Sjøberg, Empirical assessment of changeability decay in object-oriented software, Proceedings of the ICSE'99 Workshop on Empirical Studies of Software Development and Evolution, 1999, pp. 62–69.
[4] E. Arisholm, D.I.K. Sjøberg, Towards a framework for empirical assessment of changeability decay, The Journal of Systems and Software 53 (1) (2000) 3–14.
[5] E. Arisholm, D. Sjøberg, Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software, IEEE Transactions on Software Engineering 30 (8) (2004) 521–534.
[6] E. Arisholm, L. Briand, A. Føyen, Dynamic coupling measurement for object-oriented software, IEEE Transactions on Software Engineering 30 (8) (2004) 491–506.
[7] E. Arisholm, H.C. Benestad, J. Skandsen, H. Fredhall, Incorporating rapid user interface prototyping in object-oriented analysis and design with Genova, Proceedings of the NWPER'98 Nordic Workshop on Programming Environment Research, 1998, pp. 155–161.
[8] R.D. Banker, S.M. Datar, C.F. Kemerer, D. Zweig, Software complexity and maintenance costs, Communications of the ACM 36 (11) (1993) 81–94.
[9] V.R. Basili, L.C. Briand, W.L. Melo, A validation of object-oriented design metrics as quality indicators, IEEE Transactions on Software Engineering 22 (10) (1996) 751–761.
[10] S. Benlarbi, W.L. Melo, Polymorphism measures for early risk prediction, Proceedings of the 21st International Conference of Software Engineering (ICSE'99), 1999, pp. 334–344.
[11] A.B. Binkley, S.R. Schach, Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures, Proceedings of the 20th International Conference on Software Engineering (ICSE'98), 1998, pp. 452–455.
[12] L.C. Briand, J. Wust, Integrating scenario-based and measurement-based software product assessment, Journal of Systems and Software 59 (2001) 3–22.
[13] L.C. Briand, J. Wust, Modeling development effort in object-oriented systems using design properties, IEEE Transactions on Software Engineering 27 (11) (2001) 963–986.
[14] L.C. Briand, J. Wust, Empirical studies of quality models in object-oriented systems, Advances in Computers 59 (2002) 97–166.
[15] L.C. Briand, P. Devanbu, W.L. Melo, An investigation into coupling measures for C++, Proceedings of the 19th International Conference on Software Engineering (ICSE'97), 1997, pp. 412–421.
[16] L.C. Briand, J. Daly, J. Wust, A unified framework for cohesion measurement in object-oriented systems, Empirical Software Engineering 3 (1) (1998) 65–117.
[17] L.C. Briand, J.W. Daly, J. Wust, A unified framework for coupling measurement in object-oriented systems, IEEE Transactions on Software Engineering 25 (1) (1999) 91–121.
[18] L.C. Briand, J. Wust, H. Lounis, Using coupling measurement for impact analysis in object-oriented systems, Proceedings of the International Conference on Software Maintenance (ICSM'99), 1999, pp. 475–482.
[19] L.C. Briand, J. Wust, S.V. Ikonomovski, H. Lounis, Investigating quality factors in object-oriented designs: an industrial case study, Proceedings of the 21st International Conference of Software Engineering (ICSE'99), 1999, pp. 345–354.
[20] L.C. Briand, E. Arisholm, S. Counsell, F. Houdek, P. Thevenod, Empirical studies of object-oriented artifacts, methods, and processes: state of the art and future directions, Empirical Software Engineering 4 (4) (1999) 387–404.
[21] F. Brito e Abreu, The MOOD metrics set, Proceedings of the ECOOP'95 Workshop on Metrics, 1995.
[22] F. Brito e Abreu, W. Melo, Evaluating the impact of object-oriented design on software quality, Proceedings of the Third International Software Metrics Symposium (METRICS'96), 1996, pp. 90–99.
[23] M. Cartwright, M. Shepperd, An empirical investigation of an object-oriented software system, IEEE Transactions on Software Systems 26 (8) (2000) 786–796.
[24] M.A. Chaumun, H. Kabaili, R.K. Keller, F. Lustman, G. Saint-Denis, Design properties and object-oriented software changeability, Proceedings of the Fourth Euromicro Working Conference on Software Maintenance and Reengineering, 2000, pp. 45–54.
[25] S.R. Chidamber, C.F. Kemerer, A metrics suite for object-oriented design, IEEE Transactions on Software Engineering 20 (6) (1994) 476–493.
[26] S.R. Chidamber, D.P. Darcy, C.F. Kemerer, Managerial use of metrics for object-oriented software: an exploratory analysis, IEEE Transactions on Software Engineering 24 (8) (1998) 629–637.

[27] R. Christensen, Analysis of Variance, Design and Regression, Chapman &Hall/CRC Press, London, 1998.
[28] R.E. Courtney, D.A. Gustafson, Shotgun correlations in software measures, Software Engineering Journal 8 (1) (1993) 5–13.
[29] J. Daly, A. Brooks, J. Miller, M. Roper, M. Wood, Evaluating inheritance depth on the maintainability of object-oriented software, Empirical Software Engineering 1 (2) (1996) 109–132.
[30] I.S. Deligiannis, M. Shepperd, S. Webster, M. Roumeliotis, A review of experimental investigations into object-oriented technology, Empirical Software Engineering 7 (3) (2002) 193–232.
[31] S.G. Eick, T.L. Graves, A.F. Karr, J.S. Marron, A. Mockus, Does code decay? Assessing the evidence from change management data, IEEE Transactions on Software Engineering 27 (1) (2001) 1–12.
[32] K. El Emam, S. Benlarbi, N. Goel, S.N. Rai, The confounding effect of class size on the validity of object-oriented metrics, IEEE Transactions on Software Engineering 27 (7) (2001) 630–650.
[33] F. Fioravanti, P. Nesi, A method and tool for assessing object-oriented projects and metrics management, Journal of Systems and Software 53 (2) (2000) 111–136.
[34] F. Fioravanti, P. Nesi, S. Perlini, Assessment of system evolution through characterization, Proceedings of the 1998 International Conference on Software Engineering (ICSE), 1998.
[35] F. Fioravanti, P. Nesi, F. Stortoni, Metrics for controlling effort during adaptive maintenance of object oriented systems, Proceedings of the IEEE International Conference on Software Maintenance 1999 (ICSM'99), 1999, pp. 483–492.
[36] R. Harrison, S.J. Counsell, V.N. Reuben, An evaluation of the MOOD set of object-oriented software metrics, IEEE Transactions on Software Engineering 24 (6) (1998) 491–496.
[37] R. Harrison, S.J. Counsell, R.V. Nithi, An investigation into the applicability and validity of object-oriented design metrics, Empirical Software Engineering 3 (3) (1998) 255–273.
[38] R. Harrison, S. Counsell, R. Nithi, Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems, Journal of Systems and Software 52 (2–3) (2000) 173–179.
[39] R. Jeffery, B. Zucker, The state of practice in the use of software metrics, The Australian Computer Journal 31 (1) (1999) 9–16.
[40] B.A. Kitchenham, N. Fenton, S.L. Pfleeger, Towards a framework for software measurement validation, IEEE Transactions on Software Engineering 12 (12) (1995) 929–944.
[41] M.M. Lehman, L.A. Belady, Program Evolution: Processes of Software Change, Academic Press, 1985.
[42] W. Li, S. Henry, Object-oriented metrics that predict maintainability, Journal of Systems and Software 23 (2) (1993) 111–122.
[43] M. Lindvall, R.T. Tvedt, P. Costa, An empirically-based process for software architecture evaluation, Empirical Software Engineering 8 (1) (2003) 83–108.
[44] P. Nesi, T. Querci, Effort estimation and prediction of object-oriented systems, Journal of Systems and Software 42 (1) (1998) 89–102.
[45] D.L. Parnas, Software aging, Proceedings of the 16th International Conference on Software Engineering (ICSE94), 1994, pp. 279–287.
[46] M. Shepperd, D. Ince, Design metrics and software maintainability: an experimental investigation, Software Maintenance: Research and Practice 3 (1991) 215–232.
[47] S. Yacoub, H. Ammar, T. Robinson, Dynamic metrics for object-oriented designs, Proceedings of the IEEE 6th International Symposium on Software Metrics (Metrics'99), 1999, pp. 50–61.
[48] S. Yacoub, H. Ammar, T. Robinson, A methodology for architectural-level risk assessment using dynamic metrics, Proceedings of the 11th International Symposium on Software Reliability Engineering, 2000, pp. 210–221.
[49] H. Zuse, Software Complexity: Measures and Methods, de Gruyter, New York, 1991.