JOURNAL OF OBJECT TECHNOLOGY
Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering. ©JOT, 2008

Vol. 2, No. 3, May–June 2009

Exploring the Relationship between UML Design Metrics for Web Applications and Maintainability

Emad Ghosheh, Department of Information and Software Systems, University of Westminster London, UK
Sue Black, Head of Department of Information and Software Systems, University of Westminster London, UK
Epaminondas Kapetanios, Department of Information and Software Systems, University of Westminster London, UK
Mark Baldwin, Department of Information and Software Systems, University of Westminster London, UK

The size and complexity of web applications is increasing at an extremely rapid rate. Many web applications have evolved from simple HTML pages to complex service-oriented applications that have high maintenance costs. UML web design metrics are used to gauge whether the maintainability cost of the system can be controlled by correlating the UML design metrics to different measures of maintainability. This research empirically explores the relationships between existing UML design metrics based on Conallen's extension for web applications and maintenance effort. This research is evaluated, through an empirical case study of an industrial web application from the telecommunications domain.

1 INTRODUCTION

Web applications are one of the fastest growing classes of software systems. They have diffused into many different business domains such as scientific activities, product sale and distribution and medical activities[1]. Many of these web applications have evolved into complex applications that have high maintenance costs. The high maintenance cost is due to the inherent characteristics of web applications, to fast internet evolution and to the pressing market which imposes short development cycles and frequent modifications. It has been measured that in the maintenance phase software professionals spend at least half of their time analyzing software in order to understand it [2]. The cost of software maintenance accounts for a large portion of the overall cost of a software system [3]. A survey on Web applications conducted by the Cutter Consortium in 2000 revealed that 79% of web projects presented schedule delays. Also, 63% of web projects exceeded their budgets [4].

Many Companies are still asking how to assess and predict the maintainability of their software. Maintainability can be defined as:

The ease with which a software system or component can be modified

to correct faults, improve performance or other attributes, or adapt to a changed environment [5].
Measures of software maintainability can be taken either late or early in the development process. Late measurements of software maintainability can be used for assessing the software system, and planning for future enhancements. On the other hand, early measures of software maintainability can help in allocating project resources efficiently, predicting the effort of maintenance tasks and controlling the maintenance process. Maintainability can be measured by measuring some of the sub-characteristics of maintainability such as understandability, analyzability, modifiability and testability. Some studies have measured maintainability by measuring both modifiability and understandability [6, 7]. In some studies the maintainability has been quantified in the Maintainability Index (MI) [8, 9]. Other studies used effort for measuring maintainability [10]. This paper has two goals. Firstly, we explore the relationship between UML class design metrics and maintenance effort which is measured by the number of lines of code changed and by the number of revisions for components in a class diagram. Secondly, we investigate how accurately our UML design metrics are in predicting maintenance effort. More detail is given in the empirical case study section.

2 RELATED WORK

Most research related to maintainability measurement has been carried out on structured and object-oriented systems. Little work has been done in this regard using web applications.

Web Application Maintainability Model (WAMM) [11] used source code metrics and the maintainability was measured using the Maintainability Index. In WAMM new metrics were defined but there is still a need to validate those metrics empirically and theoretically. There is a need to prove how practical WAMM will be in an industrial environment. WAMM captures many metrics which might make it impractical to implement unless there is a tool which can simply and quickly capture all the metrics and provide a single Maintainability Index. The most common approach used is Regression Analysis. There is research which uses Regression Analysis to define and validate metrics and models for web applications. In [12] design and authoring effort were the dependent variables. The independent variables were based on source code metrics. There is still a need for more empirical studies to validate these newly defined metrics in order to make general conclusions. In [13] design metrics were introduced based on W2000 [13] which is a UML like language. In the study the dependent variables were variations of design effort. The independent variables were measured from the presentation, navigational and information models. It is not known how useful this approach would be, since it is not known if the W2000 language is used outside the educational environment and if it will become popular in industrial environments.

Genreo et al [14] use object-oriented UML metrics to measure maintainability for

object-oriented systems. Their approach is similar to our approach in using UML class diagram metrics. Our approach is different in the type of metrics used which are based on an extension of Conallen's model. Also in our approach we measure different dependent variables for maintainability. We decided to use UML design metrics since most of the studies use source code metrics for measuring maintainability despite the fact that many studies have shown that early metrics are much more useful [15, 16]. For the authors previous research upon which this plan is based please refer to [17, 18, 19].

3 MODELING & UML DESIGN METRICS

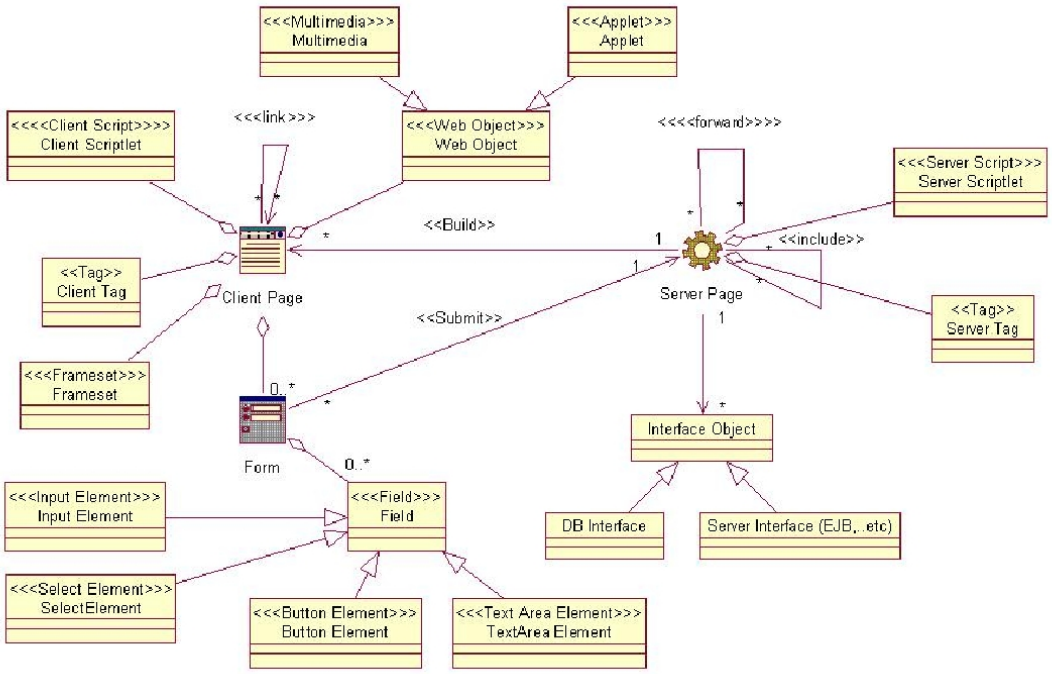


Figure 1: Web Applications Model

Modeling is a technique used to represent complex systems at different levels of abstraction, and helps in managing complexity. UML is an object-oriented language that can be used to model object-oriented systems [20]. It is possible to use UML to model web applications by using extensions supported by UML. Conallen proposed an extension of UML for web applications [20]. Conallen's model defines the following relations as generic associations between different components: *builds*, *redirects*, *links*, *submit*, *includes*, and *forwards*. The *builds* relationship is a directional relationship from the server page to the client page. It shows the HTML output coming from the server page. The *redirects* relationship is a directional relationship that requests a resource from another resource. The *links* relationship is

an association between client pages and server or client pages. It models the anchor element in HTML. The *links* relationship can have parameters which are modeled as attributes in the relationship. The *submit* relationship is a relationship between the form and the server page that processes it. The *include* relationship is a directional association between a server page and another client or server page. The *forward* relationship is a directional relationship between a server page and a client or server page. This presents delegating the server request to another page.

In Figure 1 Conallen's model is further extended [21] to include Interface Objects which have an association relationship with the server page. The Scriplets are divided into server scripts and client scripts. The server scripts have an aggregation relationship with the server page and the client scripts have an aggregation relationship with the client page. As mentioned in the related work section, we choose Conallen's notation for representing web applications because of its popularity and compliance with UML. Another advantage of using Conallen's model is that Rational Rose Web Modeler [22] and WARE [21] can be used to reverse engineer web applications to the Conallen model. Conallen's model has been referenced and used in many papers in literature [23, 21, 20]. The metrics used in this research are based on the web application reference model shown in Figure 1. They are based on Web Application Extension (WAE) for UML and measure attributes of class diagrams. Table 1 provides a description of the metrics which will be described further in the Empirical Evaluation section.

4 EMPIRICAL EVALUATION

This section will describe the case study that is used in this study.

Case Study Context

The web application used is from the telecommunication Operational Support System (OSS) domain. It is a provisioning application which is used to provision and activate the wireless service in the network. We refer to the web application as ProvisionWebApp. ProvisionWebApp has around 10,000 users of which 2,000 are concurrent. It is a critical application that is used by customer care advocates to resolve provisioning issues for wireless subscribers. The ProvisionWebApp is divided into the following functional modules: *Login Module, Search Module, Current Transactions Module, Service Transaction Module, Device Transaction Module , UserName Module, Retrigger IOTA Module, Error Queue Module, Password Module, Network Provisioning Status Module, and Help Module.* ProvisionWebApp is built using the latest web technologies and frameworks such as Struts, and EJBs, and uses Oracle for the database. The web application uses Java as its main language. It has a Concurrent Versions System (CVS) repository for storing code changes. The data used in this case study is from year 2002 to year 2005.

Table 1: Web Application Class Diagram Metrics

Metric Type	Metric Name	Description
Size	NServerP	Total number of server pages
Size	NClientP	Total number of client pages
Size	NWebP=(NServerP + NClientP)	Total number of web pages
Size	NFormP	Total number of form pages
Size	NFormE	Total number of form elements
Size	NClientScriptsComp	Total number of client scripts components
Size	NServerScriptsComp	Total number of server scripts components
Size	NC	Total number of classes
Size	NA	Total number of attributes
Size	NM	Total number of methods
Structural Complexity	NAssoc	Total number of associations
Structural Complexity	NAgg	Total number of aggregation relationships
Structural Complexity	NLinkR	Total number of link relationships
Structural Complexity	NSubmitR	Total number of Submit relationships times NFormE
Structural Complexity	NbuildsR	Total number of builds relationships times (NServerScriptsComp + NClientScriptsComp)
Structural Complexity	NForwardR	Total number of forward relationships
Structural Complexity	NIncludeR	Total number of include relationships
Coupling	WebControlCoupling = (NLinkR + NSubmitR + NbuildsR + NForwardR + NIncludeR)/ NWebP)	Number of relationships over number of web pages
Coupling	WebDataCoupling = (NFormE/NServerP)	Number of data exchanged over number of server pages

This study is trying to explore the relationship between the following metric set (NServerP, NClientP, NWebP, NFormP, NFormE, NLinkR, NSubmitR, NbuildsR, NForwardR, NIncludeR, NClientScriptsComp, NServerScriptsComp, WebControlCoupling, NC, NA, NM, NAssoc, NAgg, CoupEntropy, CohesionEntropy) and maintenance effort measured by the number of lines of code changed and the number of revisions for components in a class diagram. In addition to that we would like to get an idea of how accurately our UML design metrics predict maintenance effort. CoupEntropy, and CohesionEntropy are described in the independent variables section while the rest of the metrics are described in Table1.

Dependent Variables

The main goal of this study is to empirically explore the relationship between UML design metrics and maintenance effort. In this research two dependent variables are used to measure maintenance effort namely:

- LOC: Total number of Lines of Code added and deleted for components in a class diagram.
- nRev: Total number of revisions for components in a class diagram

The dependent variables are collected from a Concurrent Version System (CVS) repository.

Independent Variables

In this research the metrics based on the web application reference model shown in Figure 1 are used as independent variables. The following metrics (NServerP, NClientP, NWebP, NFormP, NFormE, NLinkR, NSubmitR, NbuildsR, NForwardR, NIncludeR, NClientScriptsComp, NServerScriptsComp, WebControlCoupling, WebDataCoupling, WebReusability) were defined in the authors previous study [17]. The following (NC, NA, NM, NAssoc, NAgg) metrics were defined in the study carried by Genero [14] on class diagram metrics for object oriented applications. The metrics use the different components of the web application reference model as units of measurement. In addition to the above mentioned metrics this study also uses the following two metrics: *CoupEntropy* and *CohesionEntropy*. They were first presented in [24], but we have modified them a little bit to fit in the context of UML class diagram metrics. A description of each of the metrics investigated is given as follows:

- CoupEntropy: The *CoupEntropy* is computed as shown in the following equation: $1/n \times (-\log 1/(1+m))$ where n is the total number of elements in the class diagram and m is the total number of relationships in the class diagram. The total number of elements in the class diagram is the sum of all server pages, client pages, form pages, and interface classes. The total number of relationships is the sum of all *builds*, *links*, *submit*, *includes*, *forwards*, *NAssoc*, and *NAgg* relationships.
- CohesionEntropy: The *CohesionEntropy* of a class diagram is equal to total *CoupEntropy* of all class diagrams over the *CoupEntropy* of one class diagram. The *CohesionEntropy* is computed as shown in the following equation: $\frac{\sum_{i=1}^{k}(1/n\times(-\log 1/(1+m)))}{(1/n\times(-\log 1/(1+m)))}$ where n is the total number of elements in the class diagram, m is the total number of relationships in the class diagram, and k is the total number of class diagrams in the application.

Data Collection

In this study an automated tool named WapMetrics [25] is used for the data collection. WapMetrics is a web tool that takes UML diagrams in XMI [26] format as

input and produces the results in different output formats. WapMetrics is used to compute the UML metrics from the class diagrams and provide the results in excel format in order to be used in the statistical analysis phase.

Unfortunately, the class diagrams were out of sync for the ProvisionWebApp application. IBM Rational Rose Enterprise Edition [22] was used for reverse engineering the ProvisionWebApp application. Rational Rose has a visual modeling component. It can create the design artifacts of a software system. The Web Modeler component in Rational Rose supports Conallen's extension for web applications. The Web Modeler component was used to generate the class diagrams for the various components of the ProvisonWebApp application.

For some of the class diagrams, we had to hide the visibility of attributes and methods in classes in order to fit them in one page. This did not affect the computation of the metrics since all attribute and methods are exported in the XMI file. Some of the relationships were not in the generated class diagrams. For example the *include* and *forward* relationships shown in our reference model in Figure 1 were not generated by Rational tool. We had to add these relationships manually to the class diagrams. After the class diagrams were validated, Unisys Rose XML [22] was used to export the UML class diagrams into XML Metadata Interchange (XMI) [26]. The WapMetrics tool was used to compute the independent variables from the XMI input file. The dependent variables are collected from the CVS system. LOC is computed by adding the absolute value of Lines of Code for all classes in a class diagram. nRev is computed by adding all revisions for all classes in a class diagram.

Analysis Results

Table 2: Descriptive Statistics

Variable	**N**	**Min**	**Max**	**Mean**	**Std Deviation**
LOC	30	13	13179	2780.9	3725.44
nRev	30	2	229	74.1	78.03
NFormP	30	0	2	.47	.681
NClientScriptsComp	30	0	4	.63	1.098
NServerScriptsComp	30	0	10	4	3.029
NC	30	0	12	3.27	3.704
NA	30	0	218	29.63	54.097
NM	30	0	472	62.47	113.99
NAssoc	30	0	14	3.2	3.745
NAgg	30	0	2	.70	.837
NBuildsR	30	0	14	4.73	3.581
CoupEntropy	30	0	.00644	.0041	.00145
CohesionEntropy	30	1	82.43	31.94	16.41

Descriptive Statistics

Table 2 shows the descriptive statistics of the independent and dependent variables used in this study. The measures for LOC dependent variable are much higher than the measures for nRev. The maximum for LOC is 13179 lines of code added or deleted while the maximum for nRev is 229 revisions. This result is expected since each revision will have more than one line of code associated with it. For the independent variables we can see that NM has the highest value with 472 methods. For the minimum value we can see that several variables have 0 measures. This is understandable since some class diagrams in our application did only have client pages or server pages, which resulted in having 0 measures for several independent variables.

Univariate Negative Binomial Regression

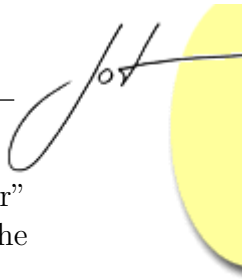
Table 3: Univariate Analysis Results

Metric Type	**Metric Name**	**LOC Coef/StdErr/Sig**	**nRev Coef/StdErr/Sig**
Size	NFormP	.639/.286/.025	.580/.283/.040
Size	NClientScriptsComp	.623/.183/.001	.610/.197/.002
Size	NServerScriptsComp	.183/.081/.023	.187/.080/.019
Size	NC	.468/.058/.000	.376/.059/.000
Size	NA	.017/.007/.017	.016/.006/.012
Size	NM	.008/.003/.017	.008/.003/.012
Structural Complexity	NAssoc	.487/.062/.000	.387/.063/.000
Structural Complexity	NAgg	.650/.205/.002	.672/.213/.002
Structural Complexity	NBuildsR	.201/.066/.003	.210/.072/.004
Coupling	CoupEntropy	1013.9/112.8/.000	970.9/141.8/.000
Cohesion	CohesionEntropy	-.058/.009/.000	-.078/.013/.000

The main goal of this case study is to investigate the feasibility of using the metrics described in Table 1 as predictors of maintenance effort. We build 2 models based on the LOC and nRev dependent variables separately. Both variables are discrete count variables that are highly skewed and always positive. Modeling using ordinary least squares regression (OLS)leads to highly non-normal error distributions leading to invalid final models. In order to cope with variables of this type Generalized Linear Models have been devised. These models include Poisson and Negative Binomial Regression. Negative Binomial Regression for *log* link function was chosen for this data as it copes with the overdispersion ($variance > mean$) found in a Poisson model [27].

We started by looking at the individual relationships between all the metrics defined in Table 1 and the dependent variables: LOC and nRev.

Table 3 shows the results from applying univariate negative binomial regression



to the data set. "Coeff" indicates the coefficient in the regression equation, "StdErr" its standard error and "Sig" its significance or p-value, that is the probability the coefficient is greater than zero by chance.

For the size metrics (NFormP, NClientScriptsComp, NServerScriptsComp, NC, NA, NM) all showed significance ($p < 0.05$) with LOC and nRev. For the structural complexity metrics only (NAssoc, NAgg, NbuildR) showed significance with LOC and nRev. Both CoupEntropy, and CohesionEntropy showed significance with LOC and nRev.

Multivariate Negative Binomial Regression

Table 4: Size Metrics Model

Parameter	**Coeff**	**Std. Error**	**Sig**
LOC Model			
Intercept	5.712	.304	.000
NClientScriptsComp	.556	.214	.009
NServerScriptsComp	-.212	.069	.002
NC	.488	.071	.000
nRev Model			
Intercept	2.415	.275	.000
NC	.376	.059	.000

Having examined the relationship of individual metrics (the predictors) and the dependent variables, LOC and nRev, we can now examine the combined effect of metrics on the dependent variables by performing a multivariate analysis.

The selection of the predictors can be made using two different stepwise regression techniques: the forward selection method, and the backward elimination method. The forward method starts with a model that only includes a constant and then adds single predictors based on a specific statistical criteria. Forward selection regression is used when there is no previous research telling us what to expect from the results. The backward method starts with a model that includes all predictors, which are deleted one at a time from the model based on a specific statistical criteria until an optimal model is found. In this study, we use $\alpha > .05$ for excluding the predictors from the model and, the backward elimination method with Negative Binomial distribution with a log link function for building the model. The likelihood-ratio chi-square test is used to compare the current model versus the intercept model. A significance value of less than 0.05 indicates that the current models outperforms the intercept model. All models have a value of less than .05 which means they outperformed the intercept only model. The following is the discussion of the analysis results:

- **Size Metric Model:** The Size Metric Model predicting LOC from NClientScriptsComp, NServerScriptsComp, and NC is statistically significant with $\chi^2(3) = 67.1$,

$p < 0.05$ The predictors NClientScriptsComp, NServerScriptsComp, and NC were each statistically significant. The Size Metric Model predicting nRev from NC is statistically significant with $\chi^2(1) = 38.7, p < 0.05$

The predictor NC was statistically significant. Table 4 shows the coefficients, standard error and significance for all the independent variables in the size metric models. The negative coefficient for NServerScriptsComp is counterintuitive, since we expect the Lines of Code to increase as we have more server script components. The reason for the negative number can be explained by the suppressor relationship between NServerScriptsComp and NClientScriptsComp which is common between correlated variables [28]. This is not of a concern as long as no strong multicollinearity [29] exists which was determined to be negligible since the condition number was equal to 3.87. A condition number of more than 30 indicates that strong multicollinearity exists between variables [29].

Table 5: Complexity Metrics Model

Parameter	**Coeff**	**Std. Error**	**Sig**
LOC Model			
Intercept	5.482	.270	.000
NAssoc	.487	.062	.000
nRev Model			
Intercept	2.426	.279	.000
NAssoc	.387	.063	.000

- **Complexity Metric Model:** The Complexity Metric Model predicting LOC from NAssoc, is statistically significant with $\chi^2(1) = 53.4, p < 0.05$

  The predictor NAssoc was statistically significant. The Complexity Metric Model predicting nRev from NAssoc is statistically significant with $\chi^1(3) = 37.6, p < 0.05$

  The predictor NAssoc was statistically significant. Table 5 shows the coefficients, standard error and significance for all the independent variables in the complexity metric models.

- **Coupling Metric Model:** The Coupling Metric Model predicting LOC from CoupEntropy is statistically significant with $\chi^2(1) = 31.2, p < 0.05$. The predictor CoupEntropy was statistically significant. The Coupling Metric Model predicting nRev is statistically significant with likelihood ratio $\chi^2(1) = 30.1, p < 0.05$. The predictor CoupEntropy was statistically significant. Table 6 shows the coefficients, standard error and significance for all the independent variables in the coupling metric models.

- **Cohesion Metric Model:** The Cohesion Metric Model predicting CohesionEntropy is statistically significant with $\chi^2(1) = 7.2, p < 0.05$. The predictor CohesionEntropy was statistically significant. The Cohesion Metric

Table 6: Coupling and Cohesion Metrics Model

Parameter	Coeff	Std. Error	Sig
LOC Coupling Model			
Intercept	4.048	.501	.000
CoupEntropy	691.8	95.9	.000
nRev Coupling Model			
CoupEntropy	699.3	111.0	.000
LOC Cohesion Model			
Intercept	8.933	.354	.000
CohesionEntropy	-.038	.010	.000
nRev Cohesion Model			
Intercept	5.743	.431	.000
CohesionEntropy	-.055	.014	.000

Model predicting nRev from CohesionEntropy is statistically significant with $\chi^2(1) = 10.3, p < 0.05$. The predictor CohesionEntropy was statistically significant. Table 6 shows the coefficients, standard error and significance for all the independent variables in the cohesion metric models.

- **All Metric Model:** The All Metric Model predicting LOC from NAssoc, NClientScriptsComp, NServerScriptsComp and CoupEntropy is statistically significant with $\chi^2(4) = 71.1, p < 0.05$. The predictors NAssoc, NClientScriptsComp, NServerScriptsComp and CoupEntropy were each statistically significant. The reason for the negative number for NServerScriptsComp can be explained by the suppressor relationship between NServerScriptsComp and CoupEntropy which is common between correlated variables. This is not of a concern as long as no strong multicollinearity exists which was determined to be negligible since the condition number was equal to 11.95.

  The All Metric Model predicting nRev from NC and CoupEntropy is statistically significant with $\chi^2(2) = 43.3, p < 0.05$. The predictors NC and CoupEntropy were each statistically significant. Table 7 shows the coefficients, standard error and significance for the independent variables in the all metric models.

Model Validation

In this study we use the Magnitude of Relative Error (MRE)[27] for evaluating the prediction models. The MRE is shown in Equation 1:

Table 7: All Metrics Model

Parameter	**Coeff**	**Std. Error**	**Sig**
LOC Model			
Intercept	4.506	.639	.000
NAssoc	.439	.079	.000
NClientScriptsComp	.568	.212	.007
NServerScriptsComp	-.338	.087	.000
CoupEntropy	375.7	187.8	.045
nRev Model			
NC	.284	.075	.000
CoupEntropy	335.2	148.6	.024

Table 8: Goodness of Fit: Values of MREs for All Models

	Size	**Complexity**	**Coupling**	**Cohesion**	**All Metrics**
LOC Model					
Mean	.1998	.2646	.3972	.5178	.1795
StdDev	.22127	.26368	.48582	.57416	.20045
P25	.0695	.0785	.0587	.0717	.0567
Median	.1259	.1665	.1385	.1560	.1167
P75	.2624	.4693	.5395	.9762	.2580
Rev Model					
Mean	.4086	.4160	.4430	.8848	.3222
StdDev	.73204	.74078	.55033	1.58322	.25940
P25	.0615	.0523	.1488	.1208	.1688
Median	.1580	.1597	.2391	.1795	.2642
P75	.2770	.3043	.6166	.9346	.3951

$$MRE = |\frac{x - y}{y}| \tag{1}$$

where x is the predicted value and y is the actual value. The result can be multiplied by 100 to get the percentage of deviation from the actual value. The $MMRE$ is the mean of the MRE, it is one of the most widely used criterion for assessing the performance of software prediction models[30, 31].

Table 8 shows the values of MRE values in the data set. It shows the mean, standard deviation, 25th percentile(P25), median, and 75th percentile (P75). When checking the mean MRE for LOC models we can see that the mean MRE for size, structural complexity, coupling and all metrics models ranges between .17 to .51. The best mean MRE value was for the All metric model (.1795) while the worst mean MRE value was for the cohesion model (5178).

When looking at the results for the nRev models, we can see that the mean MRE

Table 9: Goodness of Fit: Values of MEAN MREs for All Models using Boostrapping

	Size	**Complexity**	**Coupling**	**Cohesion**	**All Metrics**
LOC Model					
P2.5	.24	.17	.23	.31	.11
P97.5	.59	.37	.58	.71	.25
nRev Model					
P2.5	.27	.18	.27	.40	.23
P97.5	.65	.70	.65	1.44	.41

values ranges between .32 to .88. The best mean MRE value was for the All metric model (.3222) while the worst mean MRE value was for the cohesion model (.8848).

It is important to have confidence in our results. Bootstrapping is one technique that is used to obtain confidence intervals for small data sets [15]. In this study we would like to find 95 percent confidence intervals for our prediction models. We follow the following bootstrapping procedure [32]:

1. Sample 1000 times and replace randomly our 30 MRE values to obtain 1000 samples of 30 observations.
2. For each sample compute the mean MRE values for each of the models.
3. Compute the 2.5 percent and the 97.5 percent percentiles which is considered an estimate of the 95 percent confidence interval of the mean MRE values.

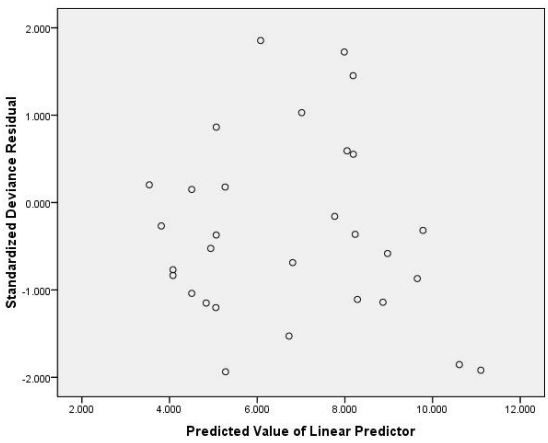


Figure 2: ScatterPlot LOC Model

Table 9 shows the results of the bootstrapping procedure described above. One can see that the best results are for the LOC complexity and LOC All metric models.

The mean MRE for the LOC complexity model is between .17 and .37. This means that we can have 96 percent confidence that the mean MRE for the complexity model will be between .17 and .37. The mean MRE for the LOC ALL metric model has even better results (.11 to.25). For the nRev model the best results are for the nRev ALL metric model (.23 to .41).

We used the likelihood-ratio chi-square test to compare current model versus the intercept only model. All models showed a significance value of less than 0.05 which indicates that the current models outperforms the intercept only model. We also have searched for the influential points and outliers in the models. We draw charts of standardized deviance residual versus and predicted values of the linear predictor variable. Figure 2 shows the this scatterplot for the LOC ALL Metrics Model. The resulting scatterplot appears to not have any outlying points. Similarly, we drew the scatterplots for the Size, Structural Complexity, Coupling, and Cohesion Metric Models, and we got similar results with no outlying points.

Threats to Validity

It is important to look into threats to validity in order to make sure the results are valid. We will look into three types of threats that can limit us to draw conclusions from the results: Construct Validity, Internal Validity, and External Validity.

Construct Validity

Construct Validity is the degree to which the independent variables and the dependent variables are accurately measured in the study. The dependent variables in this study are LOC and nRev. Both of these measures were measured from a CVS repository. The CVS repository has an accurate value for both of these measures. However, human error can happen in computing and recording both dependent variables, therefore we have repeated the measure for both variables a second time and made sure that the results from the first and second time match.

Another issue is that the measurement of the independent variables was performed from source code since no complete design was available from which the measures could be obtained. In practice the measures for the independent variables should be taken from early UML design diagrams. Measures from source code are more accurate but an investigation on how these measures compare to measures taken from design diagrams, and how this can affect the accuracy of the prediction model must be carried out.

Internal Validity

Internal Validity is the degree to which conclusions can be drawn about the effect of the independent variables on the dependent variables. In this study we have demon-

strated that some of the metrics have a statistically and significant relationship with LOC and nRev. This relationship does not prove a causal relationship, it only provides evidence that such a relationship might exist. The only way to prove causality is to run controlled experiments where the independent variables are varied in a controlled manner while preserving the functionality and size of the application. In practice this is difficult to accomplish.

External Validity

External Validity is the degree to which results can be generalized to other research settings. We have used a real industrial application with two years of data stored in a CVS repository. In addition we have used bootstrapping to have confidence in the results for the mean MRE values. However, other factors such as developer experience, size of application and technologies used can limit the generalization of the results to other web applications.

5 CONCLUSIONS

Early measures of software maintainability can help in allocating project resources efficiently, predicting the effort of maintenance tasks and controlling the maintenance process. In this study we explore the relationship between UML class design metrics and maintenance effort which is measured by the number of lines of code changed and by the number of revisions. There are many benefits of using our UML metrics. First, predicting the maintenance and cost of maintenance tasks which helps in providing accurate estimates that can help in allocating the right project resources to maintenance tasks [10]. Second, comparing design documents which can help in choosing between different designs based on the maintainability of the design. Third, identifying the risky components of a software since some studies show that most faults occur on only few components of a software system [33, 34]. Fourth, establishing design and programming guidelines for software components. This can be done by establishing values that are acceptable or unacceptable and take actions on the components with unacceptable values. This means providing a threshold of software product metrics to provide early warnings of the system [27]. Fifth, making system level prediction where the maintainability of all components can be predicted by aggregating maintainability of single components. This can be used to predict the effort it will take to develop the whole software system [27].

The results showed that there is a reasonable chance that useful prediction models can be built from early UML design metrics. We have obtained good results using bootstrapping, for the LOC ALL metric model the mean MRE lies between 11 to 25 percent for 95 percent of the cases. For the nRev ALL metric model we also got good results, the mean MRE lies between 23 to 41 percent for 95 percent of the cases.

We can say that for the LOC ALL metric model NAssoc, NClientScriptsComp, NServerScriptsComp, and CoupEntropy explained the effort measured by LOC(Lined of Code). For the nRev ALL metric model NC, and CoupEntropy explained the effort measured by nRev(Number of Revisions). The other size, complexity, and cohesion measures did not show any goodness in explaining LOC and nRev.

As a conclusion, we will conduct more empirical experiments to validate our results and show the usefulness of our UML metrics.

REFERENCES

[1] Sebastian Elbaum, Gregg Rothermel, Srikanth Karre, and Marc Fisher, "Leveraging user-session data to support web application testing," *IEEE Transactions on Software Engineering*, vol. 31, no. 3, pp. 187–202, 2005.

[2] Thomas Corbi, "Program understanding: challenge for the 1990s," *IBM Systems Journal*, vol. 28, no. 2, pp. 294–306, 1989.

[3] Norman Wilde and Ross Huitt, "Maintenance support for object-oriented programs," *IEEE Transactions on Software Engineering*, vol. 18, no. 12, pp. 1038–1044, 1992.

[4] Emilia Mendes, Nile Mosley, and Steve Counsell, "The need for web engineering: An introduction," in *Web Engineering*, chapter 1, pp. 1–26. Springer-Verlag, 2006.

[5] Pankaj Bhatt, Gautam Shroff, and Arun Misra, "Dynamics of software maintenance," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 4, pp. 1–5, 2004.

[6] Lionel Briand, Christian Bunse, and Jong Daly, "A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs," *IEEE Transactions on Software Engineering*, vol. 27, no. 06, pp. 513–530, 2001.

[7] Marcela Mario, Esperanza Manso, and Giovanni Cantone, "Building UML class diagram maintainability prediction models based on early metrics," in *Proceedings of the 9th International Software Metrics Symposium*. IEEE Computer Society Press, 2003, pp. 263–278.

[8] Don Coleman, Dan Ash, Bruce Lowther, and Paul Oman, "Using metrics to evaluate software system maintainability," *IEEE Computer*, vol. 27, no. 8, pp. 44–49, 1994.

[9] Welker Kurt and Paul Oman, "Software maintainability metrics models in practice," *Crosstalk, Journal of Defense Software Engineering*, vol. 8, no. 11, pp. 19–23, 1995.

[10] Fabrizio Fioravanti and Paolo Nesi, "Estimation and prediction metrics for adaptive maintenance effort of object-oriented systems," *IEEE Transactions on Software Engineering*, vol. 27, no. 12, pp. 1062–1084, 2001.

[11] Giuseppe DiLucca, Anna Fasolino, Porfirio Tramontana, and Corrado Visaggio, "Towards the definition of a maintainability model for web applications," in *Proceeding of the 8th European Conference on Software Maintenance and Reengineering*. IEEE Computer Society Press, 2004, pp. 279–287.

[12] Emilia Mendes, Nile Mosley, and Steve Counsell, "Web metrics - estimating design and authoring effort," *IEEE Multimedia*, vol. 08, no. 01, pp. 50–57, 2001.

[13] Luciano Baresi, Sandro Morasca, and Paolo Paolini, "Estimating the design effort of web applications," in *Proceedings of the 9th International Software Metrics Symposium*. IEEE Computer Society Press, 2003, pp. 62–72.

[14] Marcela Genero, Mario Piattini, and Coral Calero, "Empirical validation of class diagram metrics," in *Proceedings of the 2002 International Symposium on Empirical Software Engineering*. IEEE Computer Society Press, 2002, pp. 195–203.

[15] Lionel Briand and Jurgen Wurst, "Modeling development effort in object-oriented systems using design properties," *IEEE Transactions on Software Engineering*, vol. 27, no. 11, pp. 963–986, 2001.

[16] David Card, Khaled El-Emam, and Betsy Scalzo, "Measurement of object-oriented software development projects," *Software Productivity Consortium NFP*, 2001.

[17] Emad Ghosheh, Sue Black, and Jihad Qaddour, "An introduction of new UML design metrics for web applications," *International Journal of Computer & Information Science*, vol. 8, no. 4, pp. 600–609, 2007.

[18] Emad Ghosheh, Sue Black, and Jihad Qaddour, "Design metrics for web application maintainability measurement," in *Proceedings of the 6th IEEE/ACS International Conference on Computer Systems and Applications*. IEEE Computer Society Press, 2008, pp. 778–784.

[19] Emad Ghosheh, Sue Black, and Jihad Qaddour, "An industrial study using UML design metrics for web applications," in *Computer and Information Science*, vol. 131 of *Studies in Computational Intelligence*, chapter 20, pp. 231–241. Springer-Verlag, 2008.

[20] Jim Conallen, *Building Web Applications with UML*, Addison-Wesley, 2 edition, 2003.

[21] Giuseppe Di-Lucca, Anna Fasolino, and Porfirio Tramontana, "Reverse engineering web applications: the WARE approach," *Journal of Software Maintenance*, vol. 16, no. 2, pp. 71–101, 2004.

[22] IBM, "Rational Rose Enterprise Edition," Website, http://www-306.ibm.com/software/awdtools/developer/rose/index.html.

[23] Giuseppe Di-Lucca, Anna Fasolino, Ugo De-Carlini, and Porfirio Tramontana, "Ware: a tool for the reverse engineering of web applications," in *Proceeding of the 6th European Conference on Software Maintenance and Reengineering*. IEEE Computer Society Press, 2002, pp. 241–250.

[24] Edward Allen and Taghi Khoshgoftaar, "Measuring coupling and cohesion: An information-theory approach," in *Proceeding of the 6th International Software Metrics Symposium*. IEEE Computer Society Press, 1995, p. 119.

[25] Emad Ghosheh and Sue Black, "Wapmetrics: a tool for computing UML design metrics for web applications," in *Proceedings of the 7th IEEE/ACS International Conference on Computer Systems and Applications*. IEEE Computer Society Press, 2009, p. accepted.

[26] OMG, "Xml Metadata Interchange," Website, http://www.omg.org/.

[27] Khaled EL-Emam, "A methodology for validating software product metrics," Tech. Rep. NRC 44142, National Research Council Canada, 2000.

[28] David Belsley, Edwin Kuh, and Roy Welsch, *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*, WileyBlackwell, 2004.

[29] Andy Field, *Discovering Statistics Using SPSS*, Sage Publications, 2 edition, 2005.

[30] Ingunn Myrtveit, Erik Stensrud, and Martin Shepperd, "Reliability and validity in comparative studies of software prediction models," *IEEE Transactions on Software Engineering*, vol. 31, no. 05, pp. 380–391, 2005.

[31] Barbara Kitchenham and Ingunn Myrtveit, "A simulation study of the model evlauation criterion mmre," *IEEE Transactions on Software Engineering*, vol. 29, no. 11, pp. 985–995, 2003.

[32] Roger Johnson, "An introduction to the bootstrap," *Teaching Statistics*, vol. 23, no. 2, pp. 49–54, 2001.

[33] Norman Fenton and Niclas Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *IEEE Transactions on Software Engineering*, vol. 26, no. 8, pp. 797–814, 2000.

[34] Karl-Heinrich Moller and Daniel Paulish, "An empirical investigation of software fault distribution," in *Proceedings of the 4th International Software Metrics Symposium*. IEEE Computer Society Press, 1993, pp. 82–90.

ABOUT THE AUTHORS

Emad Ghosheh is a Ph.D researcher in computer science at the University of Westminster. Emad has more than ten years of experience in software development in the telecommunications industry. Emad's primary expertise is in developing network provisioning, fault management and billing solutions for the telecommunications industry. During his career Emad worked with top telecommunication and IT companies such as AT&T, Nextel, Telus Communication in Canada, Sprint and IBM. E.Ghosheh@student.westminster.ac.uk.

Sue Black is currently Head of Department, Department of Information and Software Systems, University of Westminster. Dr Black is primarily interested in finding solutions to practical software engineering problems and as such has recently co-founded a research centre: Centre for Information and Software Systems Engineering at the University of Westminster to address these problems. Her research is in the area of source code analysis, software measurement, software maintenance and software evolution. Dr Black is a Fellow of the British Computer Society: FBCS. s.e.black@westminster.ac.uk. See also http://www.sueblack.co.uk.

Epaminondas Kapetanios received his M.Sc. degree at the Technical University of Karlsruhe, Germany, Faculty of Computer Science, Institute for Program Structures and Data Organization, while focusing on the development of Information Systems and Database Technologies. He received his Ph.D. degree at the Institute of Information Systems, Department of Computer Science, ETH-Zurich, Switzerland, where he designed and contributed in the implementation of an ontology driven, high level query language (MDDQL). Epaminondas is currently holding a position as a Senior Lecturer at the School of Electronics and Computer Science, University of Westminster, London, UK. e.kapetanios@wmin.ac.uk. See also http://www2.wmin.ac.uk/ kapetae/.

Mark Baldwin is currently a Principal Lecturer in the Department of Information and Software Systems and member of the Centre for Information and Software Systems Engineering research at the University of Westminster. Mark's primary area of expertise is mathematical modeling and statistical analysis and he is a Fellow of the Royal Statistical Society. baldwim@wmin.ac.uk. See also http://www.wmin.ac.uk/hscs/page-442.