A Framework to Build Quality Models for Web Applications

Alessandro Marchetto and Andrea Trentini
Dipartimento di Informatica e Comunicazione, Università degli Studi di Milano, Italy

Abstract: *This paper describes an approach to build and apply a quality model useful to analyze a web application through an object-oriented model and to evaluate the structural software qualities using the built model. The constructed quality model is focused on a set of software metrics and uses a prediction system based on software analogies analysis. In particular, the paper focuses on model construction, customization and interpretation. The proposed approach uses a combination of traditional web and object-oriented metrics to describe structural properties of web applications and to analyze them. These metrics are useful to measure some important software attributes such as complexity, coupling, size, cohesion and defects density. Furthermore, the presented quality model uses these metrics to describe applications in order to predict some software quality factors (such as test effort, reliability, error proneness) through an instance-based classification system. The approach uses a classification system to study software analogies and to define a set of information usable as the basis for applications quality factors prediction and evaluation.*

Keywords: *Software metrics, test effort, quality factors prediction, classification systems.*

Received November 28, 2005; accepted April 13, 2006

1. Introduction

Web applications quality, reliability and functionality are important factors because software glitches could block entire businesses and cause major embarrassment. These factors have increased the need for methods, tools, and models to improve web applications (design, analysis, testing, and so on). This paper focuses on analysis of legacy web applications where business logic is embedded into web pages. The applications analyzed are composed of web documents (static, active or dynamic) and web objects.

Software metrics may be very useful to increase the quality of software analysis, reengineering, and testing through prediction or analysis systems. For example, software size or complexity measures may be used to predict software testing effort, or coupling and cohesion measures to analyze software structure, and so on. Moreover, the use of metrics in web software may be very useful because the applications are more and more dynamical. Often, they are built via server side code, such as for PHP, ASP.NET, Java, JSP, and so on. Software developed with these languages may be very complex, structured, and may be very useful to model applications via Object-Oriented (OO) meta-models. Furthermore, the use of metric-measurements is very "web-adequate", because web software is often developed in very compressed life cycle (three to six months), without a formalized process, and web documents and objects are directly coded in incremental way (often new software is obtained by duplicating via "copy & paste inheritance"). Techniques for automatic (or more formalized) application analysis, understanding, and testing are badly needed.

In our Web Applications Analysis and Testing (WAAT) project, we define an approach to extract an OO model from existing web applications, and use it as the basis for the application analysis and testing. In particular, the meta-model lets developers use some traditional web and OO metrics to describe existing software and to analyze the impact on software quality factors.

This paper analyzes the WAAT quality model based on a metrics suite to help the user (e. g., software developers) define a quantitative system to measure web software and to evaluate/predict quality factors through structural properties. Nowadays, the existing metrics-based systems for web applications measure several structural properties, but often, they measure specific web assets, such as navigation paths length, pages click-stream distances, and so on (see [12] for web metrics roadmap). In our model, we focus not only on web specific measures, but, more generally, on measures related to software in general (such as OO, web, AOP, etc.) but applied to web applications.

Several techniques exist in literature to design web application through OO approaches (see next section). These approaches are used to increase software quality in software modelling (i. e., using UML, see [10]), testing [22, 23], and analyze of existing software. This paper introduces a new framework to build quality models for web applications modelled via OO techniques. Moreover, this paper presents our WAAT

quality model built using this framework. In fact, the WAAT quality model is an instance of the meta-model. The quality models that may be built through our framework let the user analyze a web application using a set of software metrics and quality rules. The aims of this paper are to describe (step-by-step) a system used to build and customize quality models (in particular, in order to study web application testability through several structural properties of the software) and to describe a system to use and evaluate the results on the usage of our quality models. Furthermore, this paper proposes the use of a genetic algorithm to increase the quality and the automation of the analysis performed in WAAT project using our quality model. The use of an optimization-function based analysis to cluster the prevision data computed through the use of a prediction system let us decrease the manual interaction of the user and increases the effectiveness of the clustering-based approach suggested through our framework to analyze the predicted data quality.

This paper is organized as follows. Section 2 presents the state of the art. Section 3 summarizes our applications modelling and reverse engineering approach. Section 4 summarizes the WAAT framework for quality model construction. Section 5 presents our quality model built through the framework. Moreover, sections 4 and 5 contain several references with more detailed information. Section 6 introduces the used clustering algorithm. Finally, section 7 concludes the paper and describes future works.

2. State of the Art

Several web modeling methodologies are available in literature, and some of these methods are OO-based or UML-based. For example, see: UML Conallen's extensions [10], the reverse engineered model used by ReWeb [23], and the Object-Oriented OOHDM [25]. Moreover, [3] contains a review of OO modeling techniques used in web software and related to the model used in our WAAT project (Figure 1 shows the our UML class meta-model used in WAAT).

Generally speaking, software engineer metrics are very useful to analyze software applications or models, to study structural software quality, and to define prediction about software effort, such as for design effort, testing effort, and so on. However, there is no consensus within the community on which metrics to use or how to calculate metrics. In particular, there are many empirically validated metrics suite and metrics. There are many papers describing different types of metrics involved in the different measurements, metrics definition, and analysis process. In this related works section, we have selected some paper studying metrics in OO design and web software, but it's important to know that in this area the scientific community is nowadays very expansive. The goal of several metrics-papers is to define and validate a set of high-level design metrics to evaluate the quality of the application design of a software system (for example see [7, 9]). Other papers (for example see [7]) focus on empirical validation of the relationships between design measurement in OO systems (coupling, cohesion, and inheritance) and the quality of the software (the probability of fault detection in system classes during testing). [16] defines a software metrics roadmap for OO systems. [5, 18] study web metrics definition and analysis, while [12] proposes a web metrics roadmap. Some papers study metrics for specific software quality aspects. [20] defines a metrics-based approach for detecting design problems (well-known design flaws). [24] defines metrics to promote and assess software reliability. [11] studies the correlation between fault-proneness of the software and the measurable attributes of the code, while [1] studies the metrics as predictors of fault-prone classes. [15] proposes a Bayesian Belief Networks approach for software defect prediction. [21] and [27] define approach to the estimation of metrics based software testing efforts. [26] proposes an approach to estimate the cost of a software project, while [4] studies machine learning models applied to software effort prediction. [14] introduces an approach to software reliability prediction based on Markov chains.

3. Model Recovery

Our approach to model recovery is composed of application behavior analysis, application model building, and model validation as follows:

- *Application Behavior Analysis*: We use static (scanner/parser applied on source code) and dynamic (mutation-based analysis applied on software executions) analysis to extract information from an existing web applications [3].
- *Model Building*: With the information extracted by the previous phase, we build an application OO model (such as described in [13, 23]) using UML class and state diagrams. In particular, we have defined an UML meta-model (see [3]) usable to describe legacy/traditional web applications. Class diagrams are used to describe structure and components of a web application (e. g., forms, frames, Java applets, input fields, cookies, scripts, and so on), while state diagrams are used to represent behavior and navigational structures (client-server pages, navigation links, frames sets, inputs, scripting code flow control, and so on).
- *Model Validation*: The reverse engineered model may contain more information than what is needed. In particular, it may contain "not valid" information, such as not valid dynamically generated client-side pages. A client-page is "valid" if it is reachable in the original application via an execution path. Since

our reverse engineering technique may define a model with a superset of behaviors we need a pruning technique. Our proposed approach is essentially based on web server log files analysis validation and "visual navigation validation" with the user help (see [3] for more details).

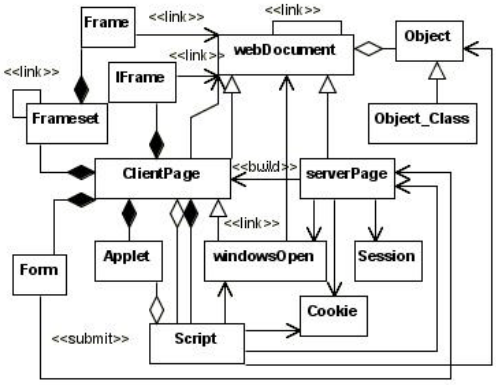

Figure 1. UML class diagram meta-model.

4. The Framework

This section describes our framework to build a quality model for web applications modelled via OO techniques. In particular, the framework phases are subdivided into two different types based on their usage, for model administrator (i. e., user building, customizing and maintaining the quality model) and for model user (i. e., user using the built quality model in a web application to analyze it).

The *quality model administrator* may build or customize the model through the following steps:

1. Construction of a meta-model (e. g., see previous section) that maps the web concepts to OO concepts.
2. Quality model construction. In this step, the use of a system such as the Goal-Question-Metrics [2] (GQM is a human-driven method to incrementally build quality models, starting from a set of limited goals then iterated and refined through questions) may be very useful to help the administrator to select the factors that they need to analyze software.
   a. Definition of a set of interesting quality factors to analyze the applications (e. g., testability, reliability, and so on).
   b. Definition of a set of testing metrics strictly related to the quality factors (e. g., test effort, reliability rate, defect density, and so on). We need to define a set of testing metrics because it is impossible to directly calculate the quality factors on software, thus these factors are always expressed in terms of testing metrics and/or their combination.
   c. Definition of a set of software metrics (e. g., lines of code, coupling between objects, and so on).
   d. Theoretical and empirical (statistical) analysis of the relationships among software and testing metrics (e. g., Pearson's correlation analysis on testing/software metrics in a large set of existing web applications).
3. Prediction system construction:
   a. Definition of prediction system using the built quality model. The use of more than one complementary (e. g., algorithmic, such as regression and software analogies based) method is suggested. In fact, the quality factors of the model (via testing metrics) may be considered as dependent variables of the prediction system, while the software metrics may be considered as independent variables used to *estimate* the dependent.
   b. Statistical validation of the prediction system constructed through a large dataset of applications and the use of predictions accuracy methods (e. g., two different usable measures of accuracy may be the MMRE -mean magnitude of relative error- which represents the mean of absolute percentage errors and the Pred25 which representsthe percentage of predictions that fall within 25 percent of the current value).
   c. (Optional) in case of more than one method used in the prediction system, the definition of results analysis or grouping policy may be needed.
4. Quality rules definition:
   a. From literature analysis, definition of the quality rules describing the relationships between quality factors and testing metrics used to measure them.
   b. From literature analysis, definition of the quality rules describing the relationships among quality factors itself (grouped in a final application *quality index*).

The output of this *administrator layer* of the framework is a quality model composed of the following:

1. The correlation of the software/testing metric.
2. The prediction system that uses software metrics to predict the values of testing metrics.
3. The set of quality rules that tie testing metrics to quality factors and that tie quality factors itself. This set of elements may be used by the user (i. e., *quality model user*) to analyze an existing web application performing the following steps:
   1. Selection of web application to analyze (source code and execution environment may be needed).
   2. Measurementof sof tware metrics (defined in step administrator_2.c) on web application source code and executions.
   3. Computation of the testing metrics (defined in step administrator_2.b) using the measured

software metrics and the prediction system built by *administrator* (this step is named *Level 1* -L1).
4. Computation of the quality factors (defined in step administrator_2.a) using the predicted testing metrics and through the use of the quality rules defined by the *administrator* (this step is named *Level 2* -L2).
5. Computation of the quality index (defined in step administrator_4.b) using the calculated quality factors (this step is named *Level 3* -L3).
6. Analysisof the results obtained for every level of values (i. e., L1, L2, L3).
   a. Clustering of the values calculated/predicted.
   b. Interpretation of the values grouped in the previous step using the quality rules (defined in step administrator_4.a/ .b) expressed as if-then-else expression in order to analyze the "border values" and define the set of software components that need to be analyzed with more precision (e. g., with specific testing, reengineering, refactoring, source code inspection, and so on).

Therefore, input of the *user layer* is the set of the quality model components and the web application to analyze (source code and executions). While the output is composed of a set of software components that the quality model classifies as "components with low quality" and that need to be re-analyzed through specific techniques by the web application developers/testers.

5. Applying the Framework to a Case Study: The WAAT Quality Model

In our WAAT project, we have applied the described framework to define a quality model based on OO metrics and focused on the analysis of testability (and other strictly related quality factors). In the rest of this section, we show the WAAT *administrator* task performed to build the WAAT quality model and a typical case study in which the model is used by the *quality model user* to analyze an existing application.

5.1. Administrator Layer

The WAAT *quality model administrator* has performed the following steps (in this sub-section we summarize some results, for more details see [19]):

1. We have built our UML meta-model (described in section 3) to analyze web applications as traditional OO software.
2. Through the use of a ad-hoc GQM system, we have built our quality model focused on testability analysis from the OO point of view.
   a. From literature analysis, we have defined the interesting quality factors such as: Testability, error proneness, reliability, and fault tolerance. The aim of our quality analysis is to study the values trend of these factors.
   b. From literature analysis, we have selected a small group of testing metrics to directly calculate the quality factors on software. In particular, our metrics are: Test Effort (TE), Test coverage (Tcov), Feature coverage (Fcov), Defect Density (DD), Unit DD (DDu), and Reliability rate (Rr).
   c. From literature (and from previous experiments), we have defined about 25 OO/web metrics to measure essential software attributes such as coupling, cohesion, separation of concerns, software size and complexity (e. g., metrics are coupling between components, response for a module, number of methods/attributes, operation for concern, and so on).
   d. We have studied the relationships between software and testing metrics through an empirical experiment based on statistical analysis of a dataset containing several web applications. In particular, we have measured software metrics on the source code of the application under analysis (or using its UML model). Then, we have performed traditional web testing and measured our set of testing metrics. Then, we have performed the statistical analysis of metrics using the Pearson's correlation analysis to evaluate the empirical correlation between metrics (for example lines of code influences test effort and feature coverage, while degree of separation of concerns influences test coverage, reliability rate, feature coverage, and defect density, and so on).
3. Through the built of quality model, we have built a prediction system to predict the values of the testing metrics (i. e., representing the quality factors) based on the software metrics.
   a. We have used two different approaches to predict the values, the first is an algorithmic approach and is the regression based (i. e., a statistical-based approach), while the other is a non-algorithmic approach based on software analogies used in a classification system (i. e., nearest neighbour classifier). We use the testing metrics as dependent variables and the software metrics as independent variables of our prediction system. We define a system to calculate the testing metrics using the values of software metrics (i. e., we may know the values of the testing metrics without performing a testing phase of the application under analysis).
   b. We have performed a statistical analysis using a set of applications in order to validate the prediction system and define its accuracy. To this aim, we have used two different measures of accuracy, the MMRE-mean magnitude of relative error-which represents the mean of absolute

percentage errors and the Pred25 which representsthe percentage of predictions that fall within 25 percent of the current value. We naturally think that the choice of accuracy evaluation is strictly related to the goals of the prediction system. For example, Pred25 is a very fast method to identify a system with high percentage of good predicted values but it may not identify a system that is occasionally inaccurate. In our case, to compare the two systems (statistical and analogies based) we decide to use both MMRE and Pred25 in order to define a comparison result more accurate and comparable. Generally speaking, the predictions by analogies tend to be more accurate but occasionally it may be wildly inaccurate, while prediction by regression may be less accurate but it is more conservative with a bias against overestimates. For example, reliability is composed of defects density and reliability-rate metrics. In particular, we know that in a software system the reliability-rate metric has more influence on software reliability than defects density. Generally speaking, this is due to the type of errors/bugs differently considered in metrics.

4. We have selected a set of quality rules defining the relationships among quality model elements. These quality rules let us interpret the results of software measurements and predictions.
   a. We have defined the set of quality rules describing the relationships between quality factors and testing metrics. Our quality rules are the following:

      Testability = (0.6 (1 - TE) + 0.2 Tcov + 0.2 Fcov) / 3
      Error Proneness = (0.6 (1 - DD) + 0.4 (1 - Ddu)) / 2
      Reliability = (0.4 (1 - DD) + 0.6 Rr) / 2
      Fault Tolerance = (0.5 (1 - DD) + 0.5 (1 - Ddu)) / 2

      Where the testing metrics are normalized in order to define the factors as $\in [0, 1]$.
   b. We have studied the relationships among quality factors and defined a relationship that groups them in a final application *quality index* thatmay be used as general value to define a structural quality of a web application from testability point of view. Our quality index rule is:

      Quality = (0.38 Testability + 0.15 (1 - Error Proneness) + 0.32 Reliability + 0.15 Fault Tolerance) / 4

5.2. User Layer

To show the step-by-step user layer of our framework applied in the WAAT project, we have used our quality model with an existing web application (we use the quality model so we are *quality model user*) in order to study its structure from the testing point of view. Thus, we have performed the following steps:

1. *MailSending* is the application selected as case study to applying the WAAT quality model. It is a small web application written in PHP that consists in a web interface to send mail.
2. From web application source code and its reverse engineered UML model, we have measured software metrics (Table 1 shows fragments of measured results). Thus, every software component (unit or group of them -named *concerns*-) is represented through its specific measured value.
3. (L1) through the prediction system based on software analogies, we have calculated the values of testing metrics for every software component (Table 2 shows fragments of the calculated values). We compute these values without performing the testing phase, these are predicted values.
4. (L2) through the predicted testing metrics and the quality rules of the model, we have calculated the quality factors value for MailSending. Then we have classified software components using these calculated factors to define a set of colour-based tables similar to the one in Figure 2. These tables show the trend of factors on MailSending components and let us identify components with low quality from testability-related point of view (components that needed to be analyzed again). For example, Figure 2 shows that the component named CO3 (corresponding to a group of software units) has low value in term of test-effort, while CO2 has high value of test-effort.
5. (L3) through the calculated quality factors and the quality index rule defined in the model, we have computed the quality index for every component or group. These indexes may be considered the general values defining the structural quality level of the MailSending software components. (Table 3 shows fragments of the calculated indexes with their colour-map).
6. We have analyzed the calculated values obtained for every level (i. e., L1, L2, L3) to evaluate the quality of the application through the defined quality factors.
   a. For every level, we have grouped the application components based on their calculated values (i. e., for L1 based on testing metrics, for L2 based on quality factors, and for L3 based on quality index). In this step, we have used a clustering algorithm (see the next section for more details about the algorithm) that helps us to define

groups of components based on their different LX-related values (i. e., distance among components). We are interested in these clusters because the "border" clusters (generally, the first and the last ones) contain components with low and/or high quality values (that are the most interesting software components for us). Table 4 shows fragments of the clustering results. In particular, the table contains the values calculated for test-effort metrics, for testability quality factors and for two different types of quality indexes (in fact, the second is an alternative version composed of a set of sub-terms of the first one).

b. We have analyzed the trend of the grouped components using the quality rules expressed in term of if-then-else rules in order to analyze the "border" clusters and define the contained set of Software components that need to be analyzed again (via testing, reengineering, refactoring, source code analysis, and so on). Table 5 shows fragments of clusters interpretation. For example, we may consider the testability (at unit level). Quality rules for testability are the following: Software quality increases if testability increases, thus (*rule1*) high testability is desirable; high testability (*rule2*) is based on high testing coverage measures (Tcov and Fcov testing metrics) and (*rule3*) low test effort (TE metric). Moreover, the clusters centroids are calculated as: For cluster $cl1_{centroids}$ = 0,143 while for $cl2_{centroids}$ = 0,577. Thus, considering the quality rules for testability, we may analyze the centroids evolution using *rule1*. In this case, we have $cl1_{centroids} < cl2_{centroids}$ thus we have found cluster named cl2 as cluster with high quality, while cl1 as low quality. Therefore, in this case components associated with cluster cl2 needed to be analyzed in more detail (e. g., through more accurate testing, or via redesign, and so on) in order to increase their quality. Furthermore, for example, we may focus MailSending analysis on application components that needed more test effort. Thus, at unit level, components in clusters named cl1 (composed of C1i and C3s) need high effort (i. e., more attention in testing); while at concern level, components in cl2 (composed of CO1 and C02) need high effort. For another example, we may analyze the total quality of components (*qI* in Table 5, it is composed of the entire set of metrics and quality factors). In this case (considering *qI*), components with low level of quality are grouped in cluster named cl2, units-level, and cl1, at concerns-level.

Table 1. Software metrics statistics for MailSending.

Metric	Type	Aver.	Std. Dev	Min	Max
InC	Unit	1,333	0,816	0	2
dPC	Unit	5,667	7,967	0	21
CBC	Unit	2,5	1,225	1	4
DIT	Unit	0,167	0,408	0	1
CMC	Unit	1,167	0,983	0	2
CFA	Unit	0,5	0,548	0	1
RFM	Unit	2,167	0,983	1	3
LOC	Unit	19,333	21,768	6	63
WOC	Unit	2,5	2,950	0	7
MCo	Concern	4,667	1,155	4	6
DS	Concern	25,03	43,3	0,05	75
Cc	Concern	3,333	0,57	3	4
….					

Table 2. Testing metrics predictions for MailSending.

		Aver.	Std. Dev	Min	Max
Unit	TE	40,12	55,67	1,74	139,49
	Rr	1,43	1,30	0,13	3,3
	DDu	0,49	0,81	0	1,92
Concern	TE	188,47	70,85	134,4	268,67
	Tcov	0,55	0,17	0,42	0,75
	Fcov	6,5	2,09	4,83	8,84
	Rr	0,8	0,31	0,57	1,15
	DDu	1,32	0,14	1,22	1,47
	DD	0,05	0,02	0,04	0,08

ANALOGIES

	TE	Tcov	Fcov	Rr	DDu	DD
CO3						
CO1						
CO2						

	TE	Rr	DDu
C1i			
C2f			
C3s			
C4c			
C5t			
C6l			

Figure 2. Classification maps of quality factors.

Table 3. Software quality index for MailSending.

Units		qI (R + EP)	qI
	C1i	0,23	0,10
	C2f	0,31	0,14
	C3s	0,25	0,06
	C4c	0,12	0,10
	C5t	0,09	0,09
	C6l	0,10	0,10
Concerns	CO3	0,15	0,04
	CO1	0,22	0,08
	CO2	0,19	0,07

6. Clustering Algorithm

In the user-layer of our framework, we have suggested the use of a clustering algorithm in combination with the pre-defined quality rules to analyze the predicted/computed data. As clustering algorithm, we use a genetic-based approach (such as [6, 8, 17]). This type of clustering technique automatically decides the best number of clusters needed to group the current data (i. e., in our case the predicted values) using an optimization function. In fact, this is very interesting in our framework because the quality model interpretation is not influenced by the experience or the

knowledge of the quality model user. Nowadays the most used clustering algorithms are dependent from human interaction and thus influenced by human choices. This may be an intrinsic limitation of the use of a clustering-based approach for the quality model interpretation, so through a genetic algorithm we may control this limitation. For example, the use of a k-means clustering forces the user to choose the *k* value (number of cluster) and then the groups are defined based on this human choice. In this case, the clustering may not be optimal and controlled by human choice. Generally speaking, our ad-hoc clustering implementation is inspired by the algorithm presented in [17], but it is applied in combination with a k-means clustering technique. We use a genetic algorithm to define the most adequate number of clusters (*k*) for the k-means clustering of our current data. In our genetic algorithm, an individual (that is a feasible solution in genetic algorithm language) is a group of clustered data elements. The *selection* operation selects the chromosomes according to the fitness function. An individual with high fitness has high probability to be in the new population. The *crossover* (we use the single-point crossover) operation exchanges features of two individuals to produce two new individuals (the children). The exchange of features may produce other good individuals. The mutation operation changes a single element in the chromosome representation. This operator should happen with very low probability. Finally, the fitness function defines the goodness of every individual (feasible solution) in the problem domain and defines the probability of the solution survival in the evolution process. In our case, the fitness is inspired to the function used in [17]. The fitness for clustering is defined as a linear combination of the intra-cluster homogeneity and inter-cluster separation as following: $F = H(G) + \mu * S(G)$, where μ ($0 \leq \mu$) is a pre-defined scale factor. Through this function the clustering problem is represented as a direct maximization of trade-off between H and S independently of number of cluster k, yet as μ varies a control on k will be indirectly achieved. The following limit cases exist:

$$\mu \rightarrow 0 \quad \text{then} \quad f = H(G) \quad \text{then} \quad k \rightarrow \infty$$
$$\mu \rightarrow \infty \quad \text{then} \quad f = S(G) \quad \text{then} \quad k \rightarrow 1$$

Notice that the scale factor is not strictly related to the specific optimization problem and to the dataset under analysis, thus we may use it to indirectly control the cluster algorithm. In our case, it may be interesting to control the density of the clustered software components. For example, if the user has a limited resource to test the application, he/she may desire to select a limited number of software components to test, thus he/she may use the scale factor to control the clustering operation.

The Genetic Algorithm:

```
{
    Randomly population initialization;
    Fitness evaluation;
    While (termination is false)
        {
          Selection
          Crossover and Mutation;
          Fitness evaluation;
        }
}
```

Genetic algorithms accept as input a finite length string (the chromosome). Each of the elements in the chromosome is a gene, and each gene has an allele value. A population is randomly defined, then a fitness function is calculated for every chromosome, and then a set of genetic operators (selection, crossover, mutation) are applied to generate the new population. The process is stopped when the population is stabilized or after a pre-defined maximum number of iterations. Moreover, we have defined a procedure to determine the most adequate scale factor based on the cluster density to obtain the best clusters considering the user necessities in term of density of the "border" clusters.

Table 4. Software clusters samples for MailSending.

		TE	Cluster	*Test.*	Cluster
Units	C1i	0,524	1	0,286	1
	C2f	0,064	2	0,562	2
	C3s	1	1	0	1
	C4c	0,025	2	0,585	2
	C5t	0,059	2	0,564	2
	C6l	0	2	0,6	2
Centroids	cl1		0,716		0,143
	cl2		0,037		0,577
Concerns	CO3	1	2	0,133	1
	CO1	0,208	1	0,192	2
	CO2	0	1	0,2	2
Centroids	cl1		0,104		0,133
	cl2		1		0,196
		qI		*qI**	
Units	C1i	0,104	2	0,232	1
	C2f	0,141	1	0,313	3
	C3s	0,063	2	0,252	1
	C4c	0,104	2	0,118	2
	C5t	0,095	2	0,09	2
	C6l	0,101	2	0,099	2
Centroids	cl1		0,141		0,242
	cl2		0,093		0,102
	cl3				0,313
Concerns	CO3	0,038	2	0,15	2
	CO1	0,076	1	0,221	1
	CO2	0,073	1	0,186	1
Centroids	cl1		0,074		0,203
	cl2		0,036		0,15

*qI = qI computed using only Reliability and errorProneness.

Table 5. Clusters samples interpretation for MailSending.

Regression					
TE		Test	*Test*		qI
Units	cl1	++	units	cl1	++
	cl2	+		cl2	+
Concerns	cl1	+	concerns	cl1	++
	cl2	++		cl2	+
Analogies					
qI		qI	*qI**		qI*
Units	cl1	+	units	cl1	++
	cl2	++		cl2	+++
				cl3	+
Concerns	cl1	++	concerns	cl1	+
	cl2	+		cl2	++

qI*	qI (Reliability + errorProneness)

7. Discussion

Referring to the two roles defined in our approach, we may note that the administrator must be an expert user, because he/she builds quality systems and our framework needs some specific interaction. For example, we may consider the choice of the set of software/testing metrics, or the definition of the quality rules (from a literature analysis). On the other hand, the quality model user doesn't need to be an expert user and the usage/interpretation of the quality model is entirely automatic. The described approach to apply our framework to an existing web application is based on a prediction system, clustering of values calculated/predicted and on the use of the pre-defined quality rules, thus it is entirely automatic. The automation level may be a very useful factor to define the model effectiveness, because its usage may be composed of several heavy and repetitive tasks and steps applied in all software components and group of them. Thus, in very large and complex web applications, an automatic quality model may be useful to analyze the software and to extract information reusable in testing, redesign, reengineering, refactoring, and so on.

Furthermore, this paper focuses on the framework description and on the use and interpretation of quality model that we may build with our framework. Often, in literature more effort is dedicated to construction, description and evaluation of a quality model rather than the description of its usage, interpretation and customization. We think that this may be a limitation on the use of quality models to increase the application structural quality, because often model are very difficult to understand, comprehend and use.

8. Conclusions

We have presented the framework used in our WAAT project to build quality models for web applications. Then, we have shown the use of this framework to build a quality model based on a combination of Object-Oriented and web metrics and focused on testability quality factor to analyze structural quality of web software. Our model is based on two different layers, the *administrator* and the *user*. In the first, we have described the steps performed by the users that need to build a new quality factors or to customize an existing model. While, in the other layer we have described the steps that the quality-model user needs to perform to apply the quality model built by the administrator.

References

[1] Basili V., Briand L., and Melo W., "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Transaction on Software Engineering*, vol. 22, no. 10, pp.751-761, October 1996.

[2] Basili V., Caldiera G., and Rombach D., *GQM Paradigm, Computer Encyclopedia of Software Engineering*, John Wiley and Sons, 1994.

[3] Bellettini C., Marchetto A., and Trentini A., "Dynamical Extraction of Web Applications Models Via Mutation Analysis," *Journal of Information - An International Interdisciplinary Journal - Special Issue on Software Engineering*, vol. 8, no. 5, pp. 673-682, September 2005.

[4] Boetticher G., "Using Machine Learning to Predict Project Effort: Empirical Case Studies in Data-Started Domains," *in Proceedings of 1st International Workshop on Model-based Requirements Engineering*, San Diego, USA, pp.17-24, 2001.

[5] Botafogo R., Rivlin E., and Shneiderman B., "Structural Analysis of Hypertexts: Identifying Hierarchies and Useful Metrics," *ACM Transaction on Information Systems*, vol. 10, no. 2, pp. 142-180, 1992.

[6] Boudjeloud L. and Poulet F., "Attribute Selection for High Dimensional Data Clustering," *in Proceedings of the International Symposium on Applied Stochastic Models and Data Analysis*, Brest, France, 2005.

[7] Briand L., Morasca S., and Basili V., "Defining and Validating High-Level Design Metrics," *Computer Science Technical Report Series*, 1999.

[8] Casillas A., González de Lena M. T., and Martínez R., *Document Clustering into an Unknown Number of Clusters Using a Genetic Algorithm*, Text, 2003.

[9] Chidamber S. and Kemerer C., "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 176-493, June 1994.

[10] Conallen J., *Building Web Applications with UML*, Addison-Wesley, 2000.

[11] Denaro G., Morasca S., and Pezzè M., "Deriving Models of Software Fault Proneness," *in*

Proceedings of 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'2002), Italy, pp.361-368, July 2002.

[12] Dhyani J., Keong W., and Bhowmick S., "A Survey of Web Metric," *ACM Computing Surveys*, vol. 34, no. 4, pp. 469-503, 2002.

[13] Di Lucca G. A., Fasolino A. R., Pace F., Tramontana P., and De Carlini U., "WARE: A Tool for the Reverse Engineering of Web Applications," *in Proceedings of 6th European Conference on Software Maintenance and Reengineering (CSMR'2002)*, Hungary, pp.230-240, March 2002.

[14] Durand J. and Gaudoin O., "Software Reliability Modelling and Prediction with Hidden Markov Chain," *INRIA-Rhone-Alpe Technical Report*, February 2003.

[15] Fenton N. and Neil M., "A Critique of Software Defect Prediction Models," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp.675-689, 1999.

[16] Fenton N. and Neil M., "Software Metrics: Roadmap," *in Proceedings of the International Conference on Software Engineering (ICSE'2000)*, Ireland**,** pp.359-370, June 2000.

[17] Gagliardi F., "An Evolutionary Computational Model of Prototype-Based Categorization: An Application on Clinical Semeiotics," *in Proceedings of the XXVII Annual Conference of the Cognitive Science Society*, Italy, pp. 732-737, 2005.

[18] Herder E., "Metrics for the Adaptation of Site Structure," *in Proceedings of the German Workshop on Adaptivity and User Modeling in Interactive Systems (ABIS02)*, pp. 22-26, 2002.

[19] Marchetto A. and Trentini A., "Web Applications Testability Through Metrics and Analogies," *in Proceedings of the 3rd International Conference on Information and Communication Technology (ICICT'2005)*, Egypt, pp. 751-780, 2005.

[20] Marinescu R., "Detecting Design Flaws via Metrics in Object-Oriented Systems," *in Proceedings of the 39th Technology of Object-Oriented Languages and Systems (TOOLS'2001)*, CA, USA, , pp. 103-116, 2001.

[21] Nageswaran S., *Test Effort Estimation Using Use Case Points*, Quality Week 2001, CA, USA, June 2001.

[22] Offutt J., Wu Y., and Du X., "Modeling and Testing of Dynamic Aspects of Web Applications," *Technical Report*, George Mason University, USA, 2004.

[23] Ricca F. and Tonella P., "Building a Tool for the Analysis and Testing of Web Applications: Problems and Solutions," *in Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2001)*, Italy, pp. 25-34, April 2001.

[24] Rosenberg L., Hammer T., and Shaw J., "Software Metrics and Reliability," *in Proceedings of the 9th International Symposium on Software Reliability Engineering*, Germany, pp.109-125, 1998.

[25] Schwabe D., Pontes R., and Moura I., *OOHDM-Web: An Environment for Implementation of Hypermedia Applications in the WWW*, SigWEB Newsletter, 1999.

[26] Shan Y., McKay R., Lokan C., and Essam D., "Software Project Effort Estimation Using Genetic Programming," *in Proceedings of the International Conference on Communications, Circuits and Systems (ICCCAS'2002)*, China, pp. 1108-1112, July 2002.

[27] Shepperd M., Schofield C., and Kitchenham B., *Effort Estimation Using Analogy*, ICSE-18, Germany, pp. 170-178, March 1996.

[28] Systä T., Understanding the Behavior of Java Program," *in Proceedings of the 7th Working Conference on Reverse Engineering (WCRE'2000)*, Australia, pp.214-223, 2000.

Alessandro Marchetto is a PhD student in computer science in the Department of Information and Communication at the University of Milano, Italy. He graduated with a degree in computer science from the University of Milano, Italy, in 2003. His research interests include software engineering, software modeling, reverse engineering techniques, and software analysis and testing, in particular, for web systems.

Andrea Trentini is an assistant professor at the Diparimento di Informatica e-Comunicazione Universita' di Milano-Bicocca, Italy. Formerly, researcher at Departimento di Informatica Sistemistica e-Comunicazione, Universita' di Milano-Bicocca, Italy. His research interests include software architectures, object-oriented design and object-oriented languages. His research applies to free (as in freedom) software. He teaches Java, UML, OOA&D, and operating systems.