A Concerns-based Metrics Suite for Web Applications

ALESSANDRO MARCHETTO

Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano
Via Comelico 39, 20135 Milano, Italy
Alessandro.Marchetto@unimi.it

Abstract.

Web applications have become very complex and crucial, especially when combined with areas such as CRM (Customer Relationship Management) and BPR (Business Process Reengineering). The scientific community has focused attention to Web applications design, development, analysis, and testing, by studying and proposing methodologies and tools. This paper proposes a metrics suite to measure Web software modelled via Object-Oriented techniques, in particular, the proposed suite is useful to analyze software designed via feature concerns approach, based on Multi-Dimensional Separation of Concerns (MDSOC). This suite lets us analyze existing software relevant to specific concern functionality, and/or relevant to functionalities integration. The measurement approach was developed in the context of WAAT (Web Applications Analysis and Testing) project.

Keywords: Web Applications, Separation of Concerns, Object-Oriented, Software Metrics

(Received June 27, 2005 / Accepted August 12, 2005)

1 Introduction

Web applications' quality, reliability and functionality are important factors because software glitches could block entire businesses and determine strong embarrassments. These factors have increased the need for methodologies, tools and models to improve Web applications (design, analysis, testing, and so on).

The approach presented in this article focuses on 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. This paper describes a metrics suite to help the user define quantitative system to measure existing Web software and to analyze its quality factors via structural properties. Nowadays the existing metrics 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 [20] for Web metrics roadmap). In this paper, we focus not only on Web specific measures, but, more generally, on measures related to software in general (such as OO, Web, AO, and so on) but applied to Web applications. Several techniques exist in literature to design Web applications via OO approaches (see Section Related Works). These approaches are used to increase software quality in software modelling (i.e., using UML, see [18]), testing ([35]), and analysis of existing software. In this paper we define a metrics suite to analyze Web software modelled via OO techniques (defined *a priori* or *a posteriori* through reverse engineering techniques). Separation Of Concerns (SOC) refers to the ability to identify, encapsulate and manipulate the software parts (code fragments) relevant to a particular concern. The use of a good SOC policy is useful to increase software quality and decrease the effort to test, maintain, understand, reuse and document software. In the project WAAT (Web Application Analysis and Testing) under development at our university laboratory, we have defined an approach to apply MDSOC to Web applications [9]. Moreover we have defined a method [6] based on feature concerns identification to slice applications via MDSOC and Concept Analysis. The defined slices are useful to test applications functionalities. Modelling Web applications via MDSOC or via concerns (in this paper named *COD*, Concern Oriented Design) lets the user define a software structure more flexible and modular than with OO design methods (*OOD*). In case of concern modelling,

defining a system to analyze software based on OO metrics (see [42]) may not be very useful, because concerns based modularization lets the user define a set of software decomposition and not only the class-based decomposition (the tyranny of dominant decomposition, see [40]). Furthermore, introducing concepts such as concerns, aspects in application design, the structural properties may change their relationship with software quality factors. For example, software components relationships requires a more specific analysis, e.g., in aspect oriented design (AOD) is introduced an implicit coupling between the aspects and the modules in the principal decomposition, in that the latter may be unaware of the presence of aspects that intercept their execution and/or modify their structure On the other hand, some new metrics systems defined to study AOD software (see [16], [38]) are also not adequate, because they study the aspect oriented programming, and they measure aspects properties (i.e., [29]) such as advices, point-cuts, and so on. These aspect properties may not be very important in case of concerns modelling, because in MDSOC aspects are only a case of possible concerns modelling via MDSOC, and there are several other types of concerns (i.e., functionality, and so on).

To improve the above cited approaches we propose to define a quantitative system to study structural properties of the software to analyze application quality factors at system and concerns level, and in particular at feature concerns level. This system is based on metrics applied in a Web application modelled by OO techniques via concern oriented software modularization. A metrics system, to be effectively useful in the analysis of software modularized via feature concerns, must analyze software units (such as classes, aspects, and so on), but also clusters of units (such as feature concerns defining a groups of units relevant for a particular task or functionality). The proposed metrics suite inquires the analysis of software unit and concerns (i.e., clusters of units).

This paper is organized as follows. Section 2 describes a general state of the art about Web modelling, aspect oriented design, and software metrics measurements. Section 3 describes our OO applications modelling, reverse engineering approach to model recovery and Section 4 describes our concerns mining technique. Section 5 introduces our concerns-based metrics suite software measurement system. Section 6 analyzes a sample case study. Finally, Section 7 ends this paper.

2 Related Works

Several Web applications modelling approaches are presented in literature (see [7]), and some of these are UML based, such as Conallen's UML extensions [18]. Our WebUml [7] is tool to reverse engineering Web applications to describe them with UML models. Other reverse engineering tools are WARE [22], ReWeb [35], and Rational Rose Web Modeler [3]

More details about **Aspect Oriented programming** are in [29], while [1] presents the famous AspectJ software. [2],[40] describe the MDSOC and HyperJ tool, while [32] studies the relations between quality factors and MDSOC, while [39] the relations between MDSOC and testing. [9] describes our approach to apply Multi-Dimensional Separation of Concerns (MDSOC) theory at Web applications. [34] describes SOC used to reduce the complexity of Web applications. [33] presents an approach to separate Web navigation concerns and application structure. [9] describes our approach to apply Multi-Dimensional Separation of Concerns (MDSOC) theory at Web applications.

The authors of [14] evaluate AOD code quality influence and present an approach for reverse engineering aspects, based on concern verification and aspect construction. [26] introduces an approach to reengineer AOSD software. [15] evaluates the suitability of clone detection as a technique for the identification of crosscutting concerns via manual concern identification. [19] introduces aspect mining and identification in OO. [27] another paper introducing aspect mining and refactoring. [11],[41] show an approach to aspect mining based on dynamic analysis technique via program traces investigation, to search recurring execution relations. In [31] three different separation of concerns (SOC) mechanisms (HyperJ, AspectJ, and a lightweight lexically based approach) are applied to separate features in the two software packages. This paper studies effects that various mechanisms have on code-base structure and on restructuring process required while performing separations. [6] presents our approach to apply MDSOC and Concept Analysis to identify and extract feature concerns on Web applications.

Generally speaking, for a 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. Several metrics-papers goal 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 Chidamber and Kemerer (C&K) OO metrics suite [17], and [36]). Other papers (for example see [12]) 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 faults detection in system classes during testing). [24] defines a software metrics roadmap for OO systems. [28] studies Web metrics definition and analysis, while [21] proposes a Web metrics roadmap. Some papers study metrics for specific software quality aspect. [30] defines a metrics-based approach for detecting design problems (well-known design flaws). [37] defines metrics to promote and assess software reliability. [13] studies the relationship between a set of OO metrics and class test effort. [10] studies machine learning models applied to software effort prediction. [23] introduces an approach to software reliability prediction based on Markov chains.

New metrics suites are derived from OO to analyze aspectized software, for example [43] analyzes coupling for classes and aspects in AOD software, and it investigates the different type of relationship type between classes and aspects. [42] describes the effect of AOD on Chidamber and Kemerer (C&K) OO metrics suite [17]. [38] defines an OO derived metrics suite and a quality model useful to measure the reusability and maintainability degrees of aspect-oriented systems. [16] defines an OO derived metrics suite to analyze aspect oriented code and to investigate the trade-off between advantages and disadvantages obtained by using the AOP approach.

3 Web Application Model Recovery

Our approach ([7] ,[8]) to model recovery is composed of: **application behavior** analysis, application **model building**, and **model validation**.
- Application behavior analysis; it is performed through static and dynamic analysis. Static and dynamic analysis treat static and dynamic application components using source code and on-line interactions with the Web server. For example, for static pages, we use traditional source code analysis based on a language parser. While, for a single server page generating multiple client pages, we apply dynamic analysis to try to determine a meaningful number of client pages (through mutation analysis and application executions). Then, the dynamically generated client side pages are analyzed (with traditional source code analysis) to build diagrams. [7] describes our used techniques.
- Model building; with the information extracted by the previous phase we build an application OO model (such as described in [35], [22]) using UML class and state diagrams. We have defined a UML meta-model usable to describe applications [7]. 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 "mutation" generated model may contain more information than 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 (without mutants) via an execution path. Since mutation may define a model with a superset of behaviors, we need a pruning technique. Our validation technique is essentially based on Web server log files analysis and "Visual Navigation validation" performed with user help. [8] describes our used techniques.

Model construction is automatic via mutation analysis, while model validation is quite user dependent. The traditional way to analyze existing Web software focuses on applications source code analysis of control flow expressions to identify the representative page input values. The input values are used to define the application feasible behaviors. The use of mutation analysis decreases user interactions needed to build application models, and simplify them, because mutation lets the user change the analysis perspective, from source code analysis to application analysis. When the model is built a user should analyze it to delete spurious information (via Model Validation approach).

4 Web Application Concerns Extraction

The goal of our algorithm is to extract features from existing Web applications. More formally, we would like to define a Separation Of Concern (SOC) procedure to identify and encapsulate software concerns. "Software concerns" are essentially software functionalities that may be transversal to software structure (software functionality for the user, features used by other features to implement external functionality, etc.). Then these concerns may be used in our WAAT project to increase the quality of software testing or analysis.

Our approach [6] to extract concerns from existing Web applications is based on MDSOC Hyperspaces definition and Formal Concept Analysis (FCA, see [25]) applied to a set of pre-defined software artifacts. In particular, we use artifacts such as software variables and methods, to define a hyperspace. Then, we elaborate this hyperspace via FCA into hierarchical groups (e.g.,

artifacts sharing def-use relationships) and to define a concept lattice describing software modularization. Finally, we elaborate the lattice to define core-concerns dependencies and to find feature concerns. This approach let us slice applications through their artifacts relationships. To apply this technique, source code analysis is needed.

5 Metrics suite

Quality Factor	*Internal Attribute*
Maintainability	
- Understandability	SOC, coupling, size
- Flexibility	SOC, coupling, cohesion
- Modularity	SOC, coupling, size
- Testability	SOC, coupling, size, complexity
- Expandability	SOC, coupling, size
Reusability	
- Modularity	SOC, coupling, size
- Understandability	SOC, coupling, size
- Adaptability	coupling, size
Reliability	
- Fault tolerance	coupling, size, complexity
- Error proneness	coupling, size, complexity
Functionality	
- Flexibility	SOC, coupling, cohesion
- Efficiency	SOC, coupling, size
- Understandability	SOC, coupling, size

Table 1: Quality Model

In this section we describe our metrics suite. Our suite is essentially derived from: [17], [38], and [16]. We extended some of the metrics to make them useful to analyze and describe *COD* software modularization, and in particular to analyze software modularized via feature concerns. Through the concept-lattice defined via our concern mining algorithm, we may analyze existing Web software to re-modularize it based on their essential features and sub-features, moreover we may isolate the application source code fragments related to every feature. This new software modularization is composed of classes, aspects, and concerns. Generally speaking, we may define software units as core-concerns (classes or aspects derived from lattice concepts) while aggregates of core-concerns to implement single features as feature concerns. In this section we use the term "module" to indicate any of the three modularization units. Furthermore, in this section we introduce a set of OO derived metrics to evaluate the goodness of a software modularity in term of separation of concerns, and to evaluate a goodness of the internal structure of an application (components and components interactions) in terms of several quality factors (see Table 1).

We have used the GQM approach [5] to describe a simple quality model to analyze *COD* modularization specific properties (i.e., separation of concerns degree, concerns coupling, and so on). Due to lack of space we do not present here the entire GQM approach used but we introduce its result: the quality model described in Table 1. Our quality model is composed of four different elements: (i) quality factors, (ii) quality sub-factors, (iii) internal attributes, and (iv) software metrics. Table 1 describes the relationships between quality factors, sub-factors, and internal software attributes. In particular, it presents: the quality that we want to primarily observe in the software system (reusability, maintainability, reliability, and functionality); the quality sub-factors that are secondary quality attributes that influence our primary qualities (testability, understandability, and so on); and the internal software properties (related to well-established software engineering principles) that are useful to increase (or to study) the quality in their internal factors. Table 1 is useful for metrics-data interpretation. Furthermore, studying the internal attributes, we have defined the set of related metrics presented in this section. Metrics are used to measure some different internal quality factors (SOC, size, complexity, coupling, cohesion), and are used at some different level of abstraction (classes, aspects, concerns, SOC, system).

The metrics composing our suite may be divided into four categories based on the internal software attributes that are influenced by them (coupling, cohesion, size & complexity, SOC). In particular, in the following, we briefly present every defined metrics: metric; metric level; brief metric definition.

Coupling is the degree to which the elements in a design are connected. The coupling degree impacts on system quality such as maintainability (modifying a given module may require the modification of some of its connected modules), understandability (a very connected module is very hard to understand), reusability (the more independent a module is, the easier it is to be reused in another application), testability (a fault in a module may cause failures in its connected modules), modularity (low coupling between modules improves software modularity), and efficiency (strong coupling between modules complicates a system). Thus, a common good programming principle is to minimize the coupling. Coupling metrics are:

- Coupling between components (CBC): for a given module (or system) defines the number of other modules to which it is coupled. It counts module relationships with other software units. For example, it counts modules that are used in attribute

declarations, or components declared return types parameters.
For a given concern in the SOC level, CBC_sum metric is defined as the sum of the CBC for every module contained in the concern.
A Module with high CBC value is harder to understand, change, reuse and test.

- Depth of Inheritance Tree (DIT): for a given module, DIT is defined as the maximum length from a given node to the root of the tree. It counts how far down the inheritance hierarchy a module is declared. High DIT value increases module complexity, and furthermore, module understandability, testability and maintainability is harder.

- Number of Children (NOC): NOC counts the number of immediate sub-modules of a given module. The number of children of a module indicates the proportion of modules potentially dependent on properties inherited from the given one. High NOC number decrease testability and reusability.

- Coupling on Method call (CMC): CMC counts the number of modules declaring methods that are possibly called by a given module. Usage of a high number of methods from many different modules indicates that the function of the given module cannot be easily isolated from the others. CMC is related to CBC metric. High CMC is associated with high dependencies from the functions in other modules.

- Coupling on filed access (CFA): CFA counts the number of modules declaring fields that are accessed by a given module. CFA measures the dependencies of a given module on other modules, in terms of accessed fields, instead of methods. In OO systems this metric is usually close to zero. CFA is related to CBC metric. High CFA is associated with high attributes dependencies in other modules.

- Response for a Module (RFM): RFM counts the methods potentially executed in response to a message received by a given module. It measures the potential communication between the given module and the others. It counts the number of methods of a given module, and the number of methods invoked by the module-methods. High RFM decreases testability, understandability, maintenance and reusability.

- Number of Stub methods (NsM): NsM counts the methods of other modules called by every module. If a module uses a method in other module, this method may be a stub method, i.e., in a testing phase. High NsM decreases testability and reusability.

- In-Cyclical Dependencies (iCd): iCd counts the number of cyclical dependencies of the software system containing a given module. High iCd decreases module testability, reusability, and maintenance.

- Cyclical Dependencies (Cd): for a given concern (or system) counts the number of cyclical dependencies in the system. High Cd decreases testability, reusability, modularity, and maintenance.

Cohesion is the degree to which software elements within a module are related to one another and work together to provide well-bounded behavior. High cohesion indicates good module subdivision. Low cohesion increases complexity, thereby increasing the likelihood of errors during the development process. Cohesion metric is:

- Lack of Cohesion in Methods (LCOM): LCOM measures the degree of similarity of methods by data input variables or module attributes. A technique to measure LCOM is based on module methods analysis. Class methods are more similar if they operate on the same set of attributes. LCOM counts the number of disjoint sets produced from the intersection of the sets of attributes used by the methods. This metric influence modularity, functionality, reusability, and testability.

Size & Complexity Software size metrics measure the size of systems, typically by counting modules contained within. For example: the number of operations in a class, the number of classes in a package, and so on. Size is traditionally used to effort estimates for implementation, review, testing, or maintenance activities. A good practice is to avoid containing too much big entities in a module, because big entities may be problematic, and they can be really difficult and complex to understand. The modules with both a high complexity and a large size tend to have the lowest reliability. Modules with low size and high complexity are also a reliability risk because they tend to contain very terse code, which is difficult to change or modify. High system size value for a software system decreases its understandability, reusability, adaptability, and testability. Size/Complexity metrics are:

- System size (SyS): SyS counts the number of system modules, i.e. the number of classes and as-

pects into the system. This metric measures the system components size.

- Lines of code (LOC): LOC counts the lines of code of every module or concern. In this case LOC does not include comment lines.

- Number of Attributes (NA): NA metric counts the number of attributes of each module. Inherited attributes are not included in the count.

- Weighted Operations per Component (WOC): it measures the complexity of a module in terms of the sum of its operations complexities. The operation complexity measure is obtained by counting the number of parameters of the operation, assuming that an operation with more parameters than another is likely to be more complex.
  For SOC level we define WOC_sum as the sum of modules complexity.

- Reuse Ratio (RR): RR measures the module reuse[1] in terms of $RR = \frac{\#super-modules}{\#total-modules}$. An RR near to 1 describes a linear hierarchy structure (poor design), while RR near to 0 describes a shallow depth structure.

- Specialization Ratio (SR): SR measures the specialization ratio [1] in term of module and super-module: $SR = \frac{\#sub-modules}{\#super-modules}$. An SR near to 1 describes a poor designed structure, while a high SR describes a good abstraction in super-modules.

Metrics for **Separation Of Concerns** (SOC) define a set of structural software measures of concerns such as: size and complexity of concerns in a software system, or internal and external concerns/modules coupling. Quality factors such as: understandability, adaptability, reusability, maintainability, testability, flexibility, modularity, and modifiability may be influenced by a SOC level metrics. We suggest to treat every feature concern as a unique module such as a class or aspect. This intuition let us apply some existing metrics to describe concerns and separation of concerns degree. SOC metrics are:

- Modules for Concern (MCo): MCo counts the number of modules for a given concern.

- Operations for Concern (OCo): OCo counts the number of operations in modules of a given concern.

[1]Reuse is in OO definition, where it is strongly connected with inheritance and specialization hierarchy

- LOC for Concern (LOCCo): LOCCo counts the number of code lines for each concern as the sum of its modules LOC.

- Degree of SOC (DS): DS measures the separation degree of a given concern from the system in term of its modules. $DS = \frac{sharedModules}{modules}$, where: *modules* is the number of concern modules, and *sharedModules* is the number of modules of the concern shared with other concerns. For a given concern, DS values near to 1 describes concerns with shared (with other system concerns) modules, thus poor separation of concerns.
  For the entire system, DS is defined by the sum of every concern DS as: $DS = (\frac{sum}{sumMax}) * 100$, where: *sum* is the sum of actual DS value for every concern, and *sumMax* is the max of possible sum. It corresponds to $sumMax = 1 * \# \, concerns$.
  DS is an indicator of concern understandability, testability, adaptability, and reusability.

- Concern coupling (Cc): Cc counts the number of modules common to more than one concern. To define common modules for a given concern, its sub-concerns are considered, while concerns aggregating the given concern are not considered. Cc defines a coupling measure for a given concern, in term of shared components between concerns or system.

- Number of Inner-Concerns (IC): IC counts the number of sub-concerns contained in a concern. IC defines a measure for concerns composition. IC defines an internal-coupling measure for a given concern, in terms of its sub concerns.

- Number of In-different-concerns (InC): InC counts the number of different concerns to which a given module is participating. A module present in more than one concern, but in the same concern hierarchy, is counted only once. InC may be an indicator of separation of concerns degree in a given system.

- Number of Module point-cuts (mPC): mPc counts the number of potentially definable point-cuts in a given module and potentially accessed by other modules. A point-cut is a module point potentially called by other modules. For example, methods or attributes declarations may be point-cuts. mPC defines a potential diffusion degree of a module in a system. mPC defines a coupling measure, in terms of module diffusion.

- Number of Degree diffusion point-cuts (dPC): dPc counts the number of modules that access point-

cuts defined in a given module. dPC defines the aspect degree diffusion on modules. It measures the number of modules depending on point-cuts defined in this module. dPC defines a potential coupling measure, in term of module diffusion.

6 Sample

This section shows the actual computation of the metrics we propose on a small Web application.

For every different modularization, we may perform a metrics based software evaluation to define the goodness of components structure in term of the quality of the modularization itself. In literature there are several empirical experiments defining various set of quality rules useful to analyze software structure with OO derived metrics. For example, a set of rules are:

1. The average method size should be less than 10. Bigger averages indicate OO design problems (i.e. function-oriented coding).

2. The average number of methods per class should be less than 20. Bigger averages indicate too much responsibility in too few classes.

3. The average number of instance variables per class should be less than 6. More instance variables indicate that one class is doing more than it should.

4. Class hierarchy nesting level should be less than 6. Deeply nested classes are more complex, inherently due to inheritance.

5. The number of class-to-class relationships within a subsystem should be relatively high. A class or group of classes (e.g., a framework) with a low amount of coupling to other classes will be more reusable.

6. The number of subsystem-to-subsystem relationships should be less than the average number of class-to-class relationships within a subsystem.

7. High cyclical dependencies in a system increases the difficulty to test components integration.

For every metric, it is possible to define a set of experience-derived quality rules, useful to evaluate the structure of a system. Instead, in this section, we focus on metrics suite analysis through the study of two different software modularizations of the same application. In particular, we compute metrics on *OOD* modelled software structure, and then we apply our concerns mining technique to extract application feature concerns and to define a new software modularization based on them. Then, we compute metrics on this new *COD* modelled software structure. Finally, we perform an analysis on measurements results.

"ImgViewer" (see Figure 1 for Home Page) is the application selected as a case study for experiments. ImgViewer is a small (see Table 2) images-viewer Web application composed of fifteen PHP/HTML files and few flat file databases. ImgViewer functionalities are: images viewer, reserved area (login + password), sponsor links manager, users-browser identification.

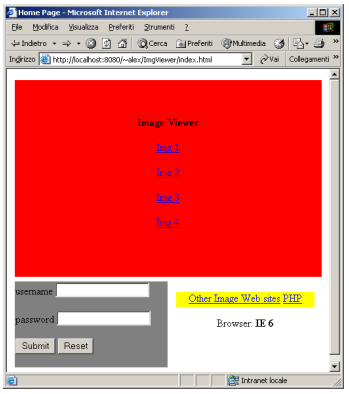

Figure 1: ImgViewer Home Page

By examining application source code and model (extracted via our reverse engineering techniques) we compute metrics measurement on the OOD-modelled *ImgViewer* components structure. Table 2 shows application SOC and system metrics values and statistics for module level metrics.

In this case, the IC metric is estimated by an expert developer based on his/her application know-how. The expert has estimated IC=3, where 3 are the main application functionalities. Moreover in this *OOD* modelled structure, there is only one concern that matches the entire system.

After measurement computation for the *OOD* modelled system, we may calculate the Pearson's correlation (r) for Module level metrics. This correlation measure is the most used and common measure to define the existence of a statistical linear correlation between two variables (i.e., see [4]). The statistical correlation between two variables reflects the statistical degree to which the variables are related. Moreover, for every computed correlation value we may also calculate its related p-value to define the statistical significance of the computation. The p-value is used to test the hypothesis of no correlation against the alternative that there is a non-zero correlation. Due to lack of space we omit the Pearson's correlation matrix compiled for

SOC Metrics:				
Concern	OCo	DS	Cc	IC
F-*all*	22	0,89	16	4

System Metrics:							
DS	Cd	SyS	LOC	RR	SR2	WMC_sum	CBC_sum
89%	5	18	166	0,11	2,5	30	46

Statistics of Module Metrics:													
Stat.	dPC	LOC	InC	mPC	CBC	DIT	NOC	CMC	CFA	RFM	NsM	NA	WMC	iCd
Mean	1,39	9,33	1,00	2,72	2,56	0,28	0,28	1,56	0,17	2,78	1,11	1,50	1,67	0,83
Max	3,00	25,00	1,00	8,00	7,00	1,00	3,00	8,00	1,00	10,00	3,00	7,00	5,00	3,00
Min	1,00	1,00	1,00	1,00	1,00	0,00	0,00	0,00	0,00	1,00	0,00	0,00	0,00	0,00
Std.Dev	0,70	7,51	0,00	2,35	,85	0,46	0,83	1,79	0,38	2,24	0,90	2,15	1,37	0,99
Median	1,00	6,50	1,00	1,00	2,00	0,00	0,00	1,00	0,00	2,00	1,00	0,00	0,00	0,50

Table 2: ImgViewer OOD metrics

Module metrics and the related p-value matrix. Generally speaking, we consider: strong positive correlation if r>0,7; weak positive correlation if 0,3<r<=0,7; strong negative correlation if r<-0,7; weak negative correlation if -0,3<r<=-0,7; and no correlation if -0,3<r<0,3; while if the p-value is small (i.e., less than 0,05 for 95% of significantly), then the correlation is significantly different from zero (i.e., there is correlation). The interpretation of statistical correlation does not imply that correlation equals causation. Moreover, a narrow data sample could lead to a deflation in the correlation.

In our case, the analysis of Pearson's correlations and p-values matrices for Module level metrics shows that there are strong relationships between point-cuts metrics (i.e., dPc and mPC) and attributes and methods size metrics (i.e.,NA or WMC). Moreover, strong relationships also exist among coupling metrics (such as between RFM or CMC and CBC or NOC), while there are no relationships among every other couple of metrics. Furthermore, we notice that lnC metric is not significant for *OOD* (in our case, it is a constant value).

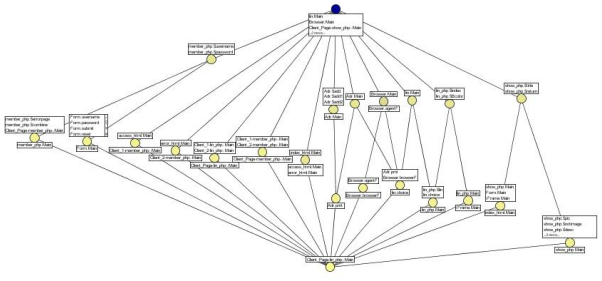

Figure 2: ImgViewer lattice

We have applied our concerns mining technique (see Section 4) to automatically define feature concerns out of our application and to define a new software modularization. Figure 2 gives the rough idea about the concept lattice for concerns extraction in ImgViewer application, while Table 3 shows every feature concerns/subconcerns found. F13 is the concern that identifies the entire software system. We may define an ordering between these concerns to define concerns that contain other concerns, as following: F3, F4 < F6 < F1 < F2; and F14; and F8, F9 < F10 < F11; and F12 < F11.

F-Concern	Feature
F2	Reserved area
F1	- Server-side
F6	- Response Pages
F8	Sponsor Link Manager
F9	User Browser Identification
F10	Control Sponsor-Browser
F11	Interface Sponsor-Browser
F12	- Client-side
F14	Image Viewer
F13	*..all features....*

Table 3: ImgViewer feature concerns

Now we may compute metrics for the new software modularization introduced by concerns extraction. Table 4 shows application SOC and system metrics values, and statistics for module level metrics. In this case, the new software modularization focuses on functionalities and it is described by some feature concerns (see Table 3). This software structure has a SOC level more defined than the previous one. SOC level metrics are very important to describe and to evaluate the separation of concerns degree.

For this *COD* software modularization we perform the Pearson's correlation and p-value analysis on the Module and SOC level metrics, to measure the statistical correlation between them. In the case of Module level, we find strong relationships among some coupling metrics (such as NsM, CBC, CMC, and RFC),

SOC Metrics:								
Concern	MCo	OCo	LOCCo	DS	Cc	IC	WMC_sum	CBC_sum
F1	5	4	23	0	0	3	6	12
F2	6	5	28	0.83	5	4	9	15
F6	3	3	11	0	0	2	2	4
F8	3	2	8	0	0	0	1	4
F9	3	4	21	0	0	0	4	4
F10	9	8	39	0,44	4	2	9	23
F11	13	9	46	0,77	10	4	12	32
F12	6	4	10	0,66	4	0	4	13
F14	4	3	53	0	0	0	7	5
F13	18	22	166	0,89	16	4	30	46

System Metrics:				
DS	Cd	SyS	LOC	RR
30%	3	27	175	0,26
SR2	WMC_sum:		CBC_sum:	
2,71	29		65	

Statistics of Module Metrics:														
Stat.	dPC	LOC	InC	mPC	CBC	DIT	NOC	CMC	CFA	RFM	NsM	NA	WMC	iCd
Mean	1,16	6,56	0,92	2,52	2,60	0,36	0,52	0,88	0,28	1,68	0,88	0,80	1,16	0,48
Max	2,00	24,00	2,00	7,00	6,00	1,00	2,00	3,00	1,00	4,00	3,00	4,00	3,00	2,00
Min	1,00	1,00	0,00	1,00	1,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Std.Dev	0,37	6,87	0,57	1,56	1,22	0,49	0,87	0,78	0,46	1,03	0,83	1,32	0,94	0,82
Median	1,00	5,00	1,00	2,00	3,00	0,00	0,00	1,00	0,00	2,00	1,00	0,00	1,00	0,00

Table 4: ImgViewer *COD* metrics

while there are no relationships between every other couple of metrics. Instead, in the case of SOC level, we find that there are strong relationships among some size metrics (i.e., OCo, WMC_sum, and MCo), while there are no relationships among every other couple of metrics.

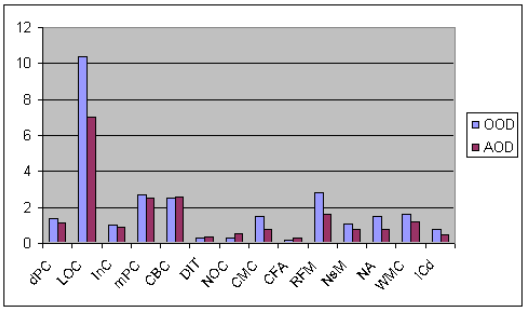

Figure 3: *OOD* & *COD* Module Metrics

Now we may compute the metrics suite for two different software modularizations of the same Web application. Then we analyze the measurement results (see Figures 3,4,5).

Figure 3 shows a graphical viewer of the average values measured in *COD* design at module level metrics, it shows a general improvement (than *OOD* design) in some metrics (dPC, LOC, lnC, mPC, CMC, RFM, NsM, NA, WMC, and lCd), while some worse values in other metrics (CBC, DIT, NOC, and CFA). For DIT, and NOC the worse values is due to the new software components organization, more hierarchical than

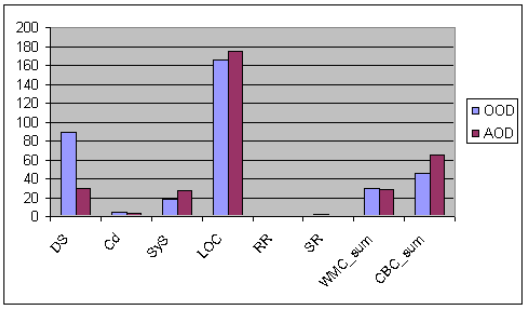

Figure 4: *OOD* & *COD* System Metrics

in *OOD*. Instead, CBC and CFA average values increase because *COD* contains classes but also aspects and concerns. In this case it is important to notice that the CBC value increases only a little bit, while for other coupling measures (i.e., CMC, RFM) the decreasing is significant. This coupling "phenomenon" is essentially due to the aspects/concerns overlapping of class method executions (e.g., AOP coupling).

Figure 4 shows that *COD* increases quality of the separation of concerns at system level (DS). Moreover, the presence of aspects and concerns defines more modules in *COD* than in *OOD*, thus increasing system size (Sys,LOC) and the coupling between internal system components (CBC_sum). It also decreases system module complexity (WMC_sum) and does not change other metrics (RR, SR, Cid).

In Figure 5, the F13 concern represents the entire system in *OOD* design, while it represents only one

type of concern in *COD* design. In the last case, metrics show that *COD* increases the quality of separation of concerns (more concerns identified in *COD* than in *OOD*). We may view a *COD* system as composed of a set of feature concerns interacting one another, in contrast with *OOD* where the system may be viewed as a black-box system implementing a list of functionalities. A good level of SOC is very useful to understand, reuse, document, maintain, and test software. In our WAAT project we are using this kind of software structure analysis to increase testing quality. In this case, a *COD* system may be analyzed by some point of views: units, clusters (or feature concerns, that are composed of units to realize a specific functionality), clusters integration (of functionalities integration), and system. This set of views may be used to focalize and prioritize test cases definition.

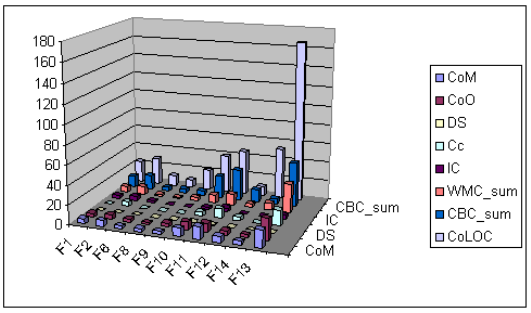

Figure 5: *COD* SOC Metrics

7 Conclusions

We proposed a metrics suite to measure Web software structural properties to analyze a set of Web software quality factors, such as testability, reusability, maintenance, understandability, and so on. Furthermore, this suite may be used with our feature concerns mining technique to evaluate SOC quality on existing Web software. The suite is composed of traditional OO metrics and by new metrics tailored to Web software.

We are currently investigating the empirical validation of the proposed metrics composing the suite. Our investigation focuses on the validation of the relationship between metrics and Web software quality factors, and, in particular, for our WAAT project, we analyze the relationship between metrics and testability.

This validation is needed because we propose the use of OO metrics (or derived) to measure traditional structural properties of Web software, albeit modelled through reverse engineering.

We are also working on some kind of mapping between "normal|standard|average" values for metrics applied on real OO software and the same metrics applied to Web software (i.e., Number of methods for class $\cong$ 20; max value of DIT $\cong$ -10; Number of classes with children $\cong$ 30%, and so on).

References

[1] Aspectj. *http://eclipse.org/aspectj/*.

[2] Hyperj. *http://www.research.ibm.com/hyperspace*.

[3] Rational Rose Web Modeler. *http://www.rational.com*.

[4] Basili, V., Briand, L., and Melo, W. A validation of Object-Oriented design Metrics as Quality Indicators. *IEEE Transaction on Software Engineering*, October 1996.

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

[6] Bellettini, C., Marchetto, A., and Trentini, A. An Approach to Concerns and Aspects Mining for Web Applications. *International Journal of Information Technology (IJIT)*, 2005.

[7] Bellettini, C., Marchetto, A., and Trentini, A. WebUml: Reverse Engineering of Web Applications. *19th ACM Symposium on Applied Computing (SAC 2004)*, Nicosia, Cyprus. March 2004.

[8] Bellettini, C., Marchetto, A., and Trentini, A. Validation of Reverse Engineered Web Application Models. *2nd International Conference on Software and Knowledge engineering (SKE 2005),*, Turkey, 2005.

[9] Bellettini, C., Marchetto, A., and Trentini, A. Applying MDSOC Web Applications. *9th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI 2005)*, USA, 2005.

[10] Boetticher, G. Using Machine Learning to Predict Project Effort: Empirical Case Studies in Data-Starved Domains. *First International Workshop on Model-based Requirements Engineering*, San Diego, USA. 2001.

[11] Breu, S. and Krinke, J. Aspect Mining Using Event Traces. *19th. Conference on Automated Software Engineering 2004 (ASE 04)*, Linz, Austria. September 2004.

[12] Briand, L., Wüst, J., Daly, J., and Porter, D. Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems. *20th The Journal of Systems and Software*, May 2002.

[13] Bruntink, M. and van A., D. Predicting Class Testability using Object-Oriented Metrics. *4th IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'04)*, 2004.

[14] Bruntink, M., van Deursen, A., and Tourwé, T. An Initial Experiment in Reverse Engineering Aspects from Existing Applications. *11th IEEE Working Conference on Reverse Engineering (WCRE 04)*, Netherlands. November 2004.

[15] Bruntink, M., van Deursen, A., van Engelen, R., and Tourwé, T. An Evaluation of Clone Detection Techniques for Identifying Cross-Cutting Concerns. *IEEE International Conference on Software Maintenance (ICSM 04)*, 2004.

[16] Ceccato, M. and Tonella, P. Measuring the Effects of Software Aspectization. *1st Workshop on Aspect Reverse Engineering (in WCRE'04)*, 2004.

[17] Chidamber, S. and Kemerer, C. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engieneering*, June 1994.

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

[19] Deursen, A., Marin, M., and Moonen, L. Aspect Mining and Refactoring. *First International Workshop on REFactoring: Achievements, Challenges, Effects (REFACE03)*, Canada. November 2003.

[20] Dhyani, J., Keong, W., and Bhowmick, S. A Survey of Web Metrics. *ACM Computing Surveys*, 2002.

[21] Dhyani, J., Keong, W., and Bhowmick, S. A Survey of Web Metrics. *ACM Computing Surveys*, 2002.

[22] 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. *6th European Conference on Software Maintenance and Reengineering (CSMR 2002)*, Budapest, Hungary. March 2002.

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

[24] Fenton, N. and Neil, M. Software Metrics: Roadmap. *International Conference on Software Engineering (ICSE 2000)*, Limerick, Ireland. June 2000.

[25] Ganter, B. and Wille, R. Formal Concept Analysis. *Springer-Verlag, Berlin, Heidelberg*, New York, 1996.

[26] Garcia, V., Lucrédio, D., Prado, A., Almeida, E., Alvaro, A., and Meira, S. L. Towards an Approach for Aspect-Oriented Software Reengineering. *7th International Conference on Enterprise Information Systems (ICEIS'2005)*, Miami, USA. May 2005.

[27] Garcia, V., Piveta, E., Lucrédio, D., Almeida, E., Prado, A., and Zancanella, L. Manipulating Crosscutting Concerns. *4th Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP'2004)*, Brazil. 2004.

[28] Herder, E. Metrics for the Adaptation of Site Structure. *German Workshop on Adaptivity and User Modeling in Interactive Systems (ABIS02)*, 2002.

[29] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., and Irwin, J. Aspect-Oriented Programming. *11th Europeen Conf. Object-Oriented Programming*, Springer Verlag. 1997.

[30] Marinescu, R. Detecting Design Flaws via Metrics in Object-Oriented Systems. *39th Technology of Object-Oriented Languages and Systems (TOOLS USA 2001)*, Santa Barbara, CA, USA. July-August 2001.

[31] Murphy, G., Lai, A., Walker, R., and Robillard, M. Separating Features in Source Code: An Exploratory Study. *23rd International Conference on Software Engineering*, Toronto, Canada. May, 2001.

[32] Noda, N. and Kishi, T. On Aspect-Oriented Design - Applying Multi-Dimensional Separation of Concerns on Designing Quality Attributes -. *First Workshop on Multi-Dimensional Separation of Concerns in Object-oriented Systems (OOPSLA'99)*, November 1999.

[33] Reina, A. and Torres, J. Analysing the Navigational Aspect. *Second International Workshop on Aspect Oriented Programming for Distributed Computing Systems*, Viena, Austria. July, 2002.

[34] Reina, A., Torres, J., and Toro, M. Aspect-Oriented Web Development vs. Non Aspect-Oriented Web Development. *Workshop of Analysis of Aspect-Oriented Software (AAOS 2003)*, University of Darmstadt, Germany. July 2003.

[35] Ricca, F. and Tonella, P. Building a Tool for the Analysis and Testing of Web Applications: Problems and Solutions. *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'200)*, Genova, Italy. April 2001.

[36] Rosenberg, L. Applying and Interpreting Object Oriented Metrics. *Software Technology Conference*, 1998.

[37] Rosenberg, L., Hammer, T., and Shaw, J. Software Metrics and Reliability. *9th International Symposium on Software Reliability Engineering*, Germany. November 1998.

[38] Sant'Anna, C., Garcia, A., Chavez, C., Lucena, C., and Staa, v. A. On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework. *17th Brazilian Symposium on Software Engineering*, 2003.

[39] Stanley, J. and Sutton, M. Multiple Dimensions of Concern in Software Testing. *First Workshop on Multi-Dimensional Separation of Concerns in Object-oriented Systems (OOPSLA'99)*, November 1999.

[40] Tarr, P., Ossher, H., Harrison, W., Stanley, J., and Sutton, M. N-degrees of separation: Multi-Dimensional Separation of Concerns. *21st International Conference on Software Engineering*, IEEE Computer Society Press, 1999.

[41] Tonella, P. and Ceccato, M. Aspect Mining through the Formal Concept Analysis of Execution Traces. *11th IEEE Working Conference on Reverse Engineering (WCRE 04)*, Netherlands. November 2004.

[42] Zakaria, A. and Hosny, H. Metrics for Aspect-Oriented Software Design. *3th International Workshop on Aspect-oriented Modeling*, 2003.

[43] Zhao, J. Measuring Coupling in Aspect-Oriented Systems. *10th International Software Metrics Symposium (Metrics 04)*, 2004.