# Model Cloning: A Push to Reuse or a Disaster?

Maria Rigou[1,2], Spiros Sirmakessis[1], Giannis Tzimas[1,2]

[1]Research Academic Computer Technology Institute, 61 Riga Feraiou str., GR-26221 Patras, Hellas
[2]University of Patras, Computer Engineering & Informatics Dept., GR-26504, Rio Patras, Hellas
{rigou, syrma, tzimas}@cti.gr

**Abstract.** The paper focuses on evaluating and refactoring the conceptual schemas of Web applications. The authors introduce the notion of model clones, as partial conceptual schemas that are repeated within a broader application model and the notion of model smells, as certain blocks in the Web applications model, that imply the possibility of refactoring. A methodology is illustrated for detecting and evaluating the existence of potential model clones, in order to identify problems in an application's conceptual schema by means of efficiency, consistency, usability and overall quality. The methodology can be deployed either in the process of designing an application or in the process of re-engineering it. Evaluation is performed according to a number of inspection steps. At first level the compositions used in the hypertext design are evaluated, followed by a second level evaluation concerning the manipulation and presentation of data to the user. Next, a number of metrics is defined to automate the detection and categorization of candidate model clones in order to facilitate potential model refactoring. Finally, the paper proposes a number of straightforward refactoring rules based on the categorization and discusses the aspects affecting the automation of the refactoring procedure.

## 1 Introduction

Modern web applications support a variety of sophisticated functionalities incorporating advanced business logic, one-to-one personalization features and multimodal content delivery (i.e. using a diversity of display devices). At the same time, their increasing complexity has led to serious problems of usability, reliability, performance, security and other qualities of service in an application's lifecycle.

The software community, in an attempt to cope with this problem and with the purpose of providing a basis for the application of improved technology, independently of specific software practices, has proposed a number of modeling methods and techniques that offer a higher level of abstraction to the process of designing and developing Web applications. Some of these methods derive from the area of hypermedia applications like the Relationship Management

Methodology (RMM) (Isakowitz et al. 1995), Araneus (Atzeni et al. 1998) and HDM (Garzotto et al. 1993), which pioneered the model-driven design of hypermedia applications. HDM influenced several subsequent proposals for Web modeling such as HDM-lite (Fraternali and Paolini, 1998), a Web-specific version of HDM, Strudel (Fernandez et al. 1998) and OOHDM (Schwabe and Rossi 1998). Extensions to the UML notation (Booch et. al. 1998) to make it suitable for modeling Web applications have been proposed by Conallen (1999a; 1999b). Finally, Web Modeling Language - WebML (Ceri et al. 2000) provides a methodology and a notation language for specifying complex Web sites and applications at the conceptual level and along several dimensions.

Most of the above methodologies are based on the key principle of separating data management, site structure and page presentation and provide formal techniques and means for an effective and consistent development process, and a firm basis for re-engineering and maintenance of Web applications.

Deploying a methodology for the design and development of a Web application enhances effectiveness, but does not guarantee optimization in the design process, mainly due to the restricted number of available extreme designers/programmers (Boehm 1981). Moreover, most applications are developed by large teams, leading to communication problems in the design/development process, often yielding products with large numbers of defects. In most of the cases due to the lack of time, designers reuse their previous work and experience without trying to fully adapt it to the requirements of the project at hand, resulting to "bad" cases of reuse. This situation stresses the need for restructuring/refactoring applications, even in their conceptual level, and the fact that effective modeling must be treated as a first class citizen and be considered from the very early and during all stages of the design process.

In the past, a number of research attempts have been conducted in the field of refactoring applications based on their design model. Most of them focus on standalone software artifacts and deploy UML to perform refactoring (Mens et al., 2003). But despite the popularity of model-driven methodologies, there is an absence of assessment/analysis throughout the design and development process.
On the other hand, there are cases where the use of a methodology may cause a number of problems leading to design processes that make application development less effective and productive, particularly in the cases of automatic code generation. For this reason, there is a need for techniques appropriate for analyzing conceptual schemas so as to discover potential problems at the early phases of the design process and thus allow for fast recovery with the less possible effort. The goal of this paper is to argue the need to approach all aspects concerning effective design from the beginning in the Web application's development cycle. Since internal quality is a key issue for an application's success, it is important that it is dealt with through a design view, rather than only an implementation view.

## 2 Field Background and the notion of Model Cloning

One of the intrinsic features of real-life software environments is their need to evolve. As the software is enhanced, modified, and adapted to new requirements, the code becomes more complex and drifts away from its original design, thereby lowering the quality of the software. Because of this, the major part of the software development cost is devoted to software maintenance (Coleman et al. 1994; Guimaraes 1983). Improved software development methodologies and tools cannot resolve this problem because their advanced capabilities are mainly used for implementing new requirements within the same time slot, making software once again more complicated (Mens and Tourwe 2004). To cope with this increased complexity one needs techniques for reducing software complexity by incrementally improving the internal software quality.

Restructuring, refactoring and code cloning are well known notions in the software community. According to the reverse engineering taxonomy of (Chikofsky and Cross 1990), *restructuring* is defined as: "… the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior (functionality and semantics). A restructuring transformation is often one of appearance, such as altering code to improve its structure in the traditional sense of structured design. While restructuring creates new versions that implement or propose change to the subject system, it does not normally involve modifications because of new requirements. However, it may lead to better observations of the subject system that suggest changes that would improve aspects of the system."

In the case of object-oriented software development the definition of *refactoring* is basically the same: "… the process of changing a software system in such a way that it does not alter the external behavior of the code, yet improves its internal structure" (Fowler et al. 1999). The key idea here is to redistribute classes, variables and methods in order to facilitate future adaptations and extensions.

*Code cloning* or the act of copying code fragments and making minor, non-functional alterations, is a well known problem for evolving software systems leading to duplicated code fragments or *code clones*. Code cloning can be traced by *code smells* that is, certain structure in code that suggests the possibility of refactoring (Fowler et al. 1999).

Roundtrip engineering has reached a level of maturity that software models and program code can be perceived as two different representations of the same artifact. With such an environment in mind, the concept of refactoring can be generalized to improving the structure of software instead of just its code representation.

There is a variety of techniques for supporting the process of detecting code cloning in software systems. Some of them are based on string and token matching (Johnson 1993; Manber 1994; Johnson 1994; Baker 1995; Rieger and Ducasse 1998; Ducasse et al. 1999; Kamiya et al. 2002), some others on comparing sub-trees and sub-graphs (Kontogiannis 1997; Baxter et al. 1998; Krinke 2001), while others are based on metrics characterization (Mayrand et al. 1996; Lague et al. 1997; Kontogiannis 1997; Balazinska et al. 1999; Antoniol et al. 2001). Moreover,

there are a large number of tools that mine clones in source code and support a variety of programming languages such as C, C++, COBOL, Smalltalk, Java, and Python.

Clone mining in Web applications was first proposed by Di Lucca et al. (2001) who study the detection of similar HTML pages by calculating the distance between page objects and their degree of similarity. Static page clones can also be detected with the techniques proposed by Boldyreff and Kewish (2001) and Ricca and Tonella (2003), with the purpose of transforming them to dynamic pages that retrieve their data form a database. Despite that, the decreasing percentage of Web sites that merely publish static content and the current shift towards Web applications with high degree of complexity, lead to the need to introduce new techniques, capable of coping with the problem. In the specific domain of Web application modeling, the notion of cloning has not yet been introduced. Even though a few model-level restructuring techniques have been proposed, this certain issue remains open for the scientific community (Mens and Tourwe 2004). Moreover, the existing techniques are based exclusively on UML as the modeling language and there is no technique based on one of the rest of Web application modeling languages and methods.

In this work we extend the notion of code cloning to the modeling level of a Web application. Analogously to code cloning, we introduce the notion of *model cloning* as the process of duplicating, and eventually modifying, a block of the existing application's model that implements a certain functionality. This ad-hoc form of reuse occurs frequently during the design process of a Web application. We will attempt to capture *model smells* that is, certain blocks in the Web application's model that imply the possibility of refactoring. Both notions were initially introduced in a previous work (Sakkopoulos et al., 2005) addressing the specific domain of personalization in Web applications. The approach we follow in the current paper is much broader and refers to the generalized process of conceptual modeling.

More specifically, in this paper we provide a methodology in order to evaluate the conceptual schema of an application, by means of the design features incorporated in the application model. We try to capture cases (i.e. model clones) which have different design, but produce the same functionality, thus resulting in inconsistencies and ineffective design and may have been caused by inappropriate forms of model reuse.

The evaluation of the conceptual schema is performed in two steps of inspection: a first level evaluation of the hypertext compositions used in the hypertext design, and a second level evaluation of data manipulation and presentation to the user. We provide metrics for the quality evaluation of the application's conceptual schema and a number of restructuring/refactoring rules to improve the final applications quality.

Up to date, modeling of Web applications is mainly deployed during the early life cycle stages as a system analysis guide, with little attention allocated to the use of specifications at a conceptual level during application evaluation, maintenance and evolution analysis (Fraternali et al. 2004b). The proposed methodology can be deployed either in the process of designing an application or

in the process of re-engineering it. In this work, WebML has been utilized as design platform for the methods proposed, mainly due to the fact that it supports a concrete framework for the formal definition of data intensive Web Applications and the fact that it is supported by a robust CASE tool called WebRatio (WebRatio 2005). Most of the work presented here can be generalized and applied to applications utilizing other modeling languages, such as OOHDM or HDM-lite, with slight straightforward modifications.

The remaining of this paper is structured as follows: Section 3 provides a short overview of WebML. Section 4 describes in detail the methodology for mining model clones in the conceptual schema of an application, while section 5 introduces metrics for the evaluation of the clones. Finally, section 6 provides refactoring suggestions and section 7 concludes the paper and discusses future steps.

## 3 WebML: A Brief Overview

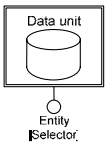
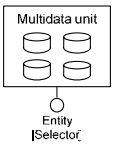
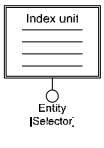
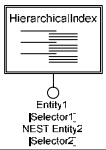
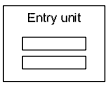
WebML is a conceptual model for Web application design (Ceri et al. 2002). It offers a set of visual primitives for defining conceptual schemas that represent the organization of the application contents and of the hypertext interface. These primitives are also provided with an XML-based textual representation, which allows specifying additional detailed properties that cannot be conveniently expressed in terms of visual notation. The organization of data is specified in WebML by exploiting the E-R model, which consists of entities (defined as containers of data elements) and relationships (defined as semantic connections between entities). WebML also allows designers to describe hypertexts for publishing and managing content, called *site views*. A site view is a specific hypertext, which can be browsed by a particular class of users. Within the scope of the same application, multiple site views can be defined.

Site views are internally organized into hypertext modules, called *areas*. Both site views and areas are composed of *pages*, which in turn include containers of elementary pieces of content, called *content units*. Typically, the data published by a content unit are retrieved from the database, whose schema is expressed by the E-R model. The binding between content units (and hypertext) and the data schema is represented by the *source entity* and the *selector* defined for each unit. The source entity specifies the type of objects published by the content unit, by referencing an entity of the E-R schema. The selector is a filter condition over the instances of the source entity, which determines the actual objects published by the unit. WebML offers predefined units, such as data, index, multidata, scroller, multichoice index, and hierarchical index (some of them are presented in Table 1), that express different ways of selecting entity instances and publishing them in a hypertext interface.

To compute its content, a unit may require the “cooperation” of other units, and the interaction of the user. Making two units interact requires connecting them with a *link*, represented as an oriented arc between a source and a destination unit.

The aim of a link is twofold: permitting navigation (possibly displaying a new page, if the destination unit is placed in a different page), and enabling the passing of parameters from the source to the destination unit.
Finally, WebML models the execution of arbitrary business actions, by means of operation units. An operation unit can be linked to other operation or content units. WebML incorporates some predefined operations (enabling content management) for creating, modifying and deleting the instances of entities and relationships, and allows developers to extend this set with their own operations.

**Table 1. Some basic WebML elements. The complete set is listed in (Ceri et al. 2002)**

|  |  |  |  |  |
|---|---|---|---|---|
| Displays a set of attributes for a single entity instance. | Displays a set of instances for a given entity. | Displays a list of properties of a given set of entity instances. | Displays index entries organized in a multi-level tree. | Displays forms for collecting input data into fields |

# 4 A Methodology for Mining Model Clones

In what follows, we present a methodological approach for mining potential model clones at the conceptual level of a Web application. The methodology comprises three distinct phases.

- In the first phase, we transform the Web application's conceptual schema into a number of directed graphs, representing the navigation structure and the distribution of content among the areas and pages of the application. This forms the basis for an information extraction mechanism required for the next phase.
- In the second phase, we extract potential model clones and information related to the navigation and semantics of the application by utilizing graph mining techniques,
- Finally, in the third phase we provide a first level categorization of the potential model clones according to a number of criteria.

## 4.1 Conceptual Schema Transformation

In this phase the application's conceptual schema is preprocessed in order to provide the means for the potential model clones extraction. Assuming an application comprising a number of site views, we construct a first set of graphs representing the navigation, the content presentation and the manipulation mechanisms of the application. More specifically, we define a site view as a

directed graph of the form $G(V, E, f_V, f_E)$, comprising a set of nodes $V$, a set of edges $E$, a node-labeling function $f_V: V \rightarrow \Sigma_V$, and an edge-labeling function $f_E: E \rightarrow \Sigma_E$. Function $f_V$ assigns letters drawn from an alphabet $\Sigma_V$ to the site view nodes, whereas $f_E$ has the same role for links and the edge alphabet $\Sigma_E$. $\Sigma_V$ has a different letter for each different WebML element (content units, operations, pages, areas, etc). Correspondingly, $\Sigma_E$ consists of all the different kinds of links (contextual, non contextual, transport & automatic). Besides the predefined WebML links, we introduce a special kind of edge (labeled *'c'*) in order to represent the *containment* of content units or sub-pages in pages, as well as pages, sub-areas and operation units in areas. Note that there can be arbitrary containment sequences. A transformation example is depicted in Fig. 1 (Transformation A), where we transform a page containing several content units, interconnected by a number of contextual links.

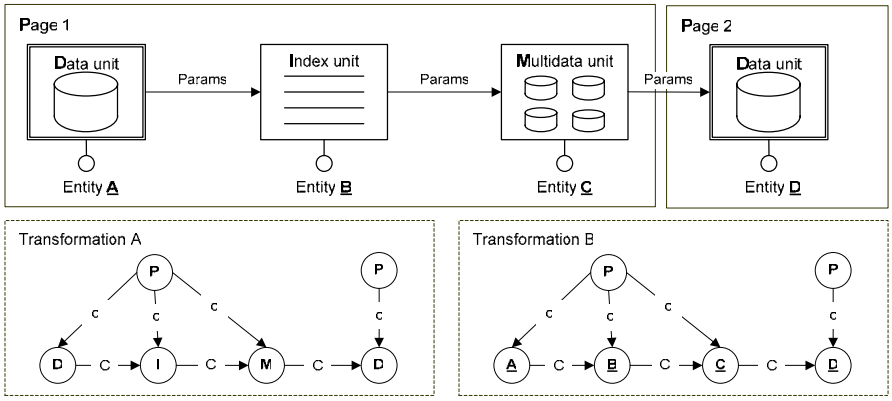



**Fig. 1. Transformation of a WebML hypertext composition to its graph equivalents**

Following a similar procedure, for every site view of the hypertext schema we create a second graph representing the data distribution within each area, sub-area and page, thus constructing a second set of graphs. In this case we define a site view as a directed graph of the form $Q(N, L, f_N, f_L)$, comprising a set of nodes $N$, a set of edges $L$, a node-labeling function $f_N: N \rightarrow \Sigma_N$, and an edge-labeling function $f_L: L \rightarrow \Sigma_L$. Function $f_N$ assigns letters drawn from an alphabet $\Sigma_N$ to the site view nodes, whereas $f_L$ has the same role for links and the edge alphabet $\Sigma_L$. $\Sigma_N$ has a different letter for each different source entity used by the WebML elements comprising the hypertext schema, as well as for the pages, sub-pages, areas and sub-areas of the site view. $\Sigma_L$ comprises all the different kinds of WebML links in order to model the context navigation within the hypertext schema. As in the previous transformation, we also introduce edges denoting containment. A transformation example is depicted in Fig. 1 (Transformation B).

### 4.2 Potential Model Clones Selection

Having modeled the navigation, content presentation and manipulation mechanisms of the application, as well as the data distribution within each site view, the next step is to capture model smells.

We traverse the first set of graphs constructed in the previous phase, in order to locate identical configurations of hypertext elements (subgraphs) along with their variants, either within a graph representing a single site view or among graphs representing different site views. The variants include all the alternatives in which the configuration retrieved starts and terminates (that is, all the nodes or sets of nodes in the graph passing the context and receiving context from the hypertext configuration). For every instance of the configuration we also retrieve the respective variants. The recovery of the various configurations can be achieved using graph mining algorithms.

Intuitively, after modeling the site views as directed graphs the task is to detect frequently occurring induced subgraphs. The problem in its general form is synopsized to finding whether the isomorphic image of a subgraph exists in a larger graph. The latter problem has proved to be NP-complete (Garey and Johnson 1979). However, quite a few heuristics have been proposed to face this problem with the most prominent such approaches being the *gSpan* (Yan and Han 2002), the *CloseGraph* (Yan and Han, 2003) and the *ADI* (Wang et al. 2004). Any of the above approaches can be utilized for extracting the hypertext configurations.

Likewise, employing the same graph mining techniques, we traverse the second set of graphs in order to locate identical configurations of data elements (source entities) along with their variants.

Finally, we try to locate compositions of identical hypertext elements referring to exactly the same content interconnected with different link topologies. Ignoring the edges in the first set of graphs, except from those representing containment, we mine identical hypertext configurations within a graph or among graphs. Then, we filter the sets of subgraphs acquired utilizing the information represented in the second set of graphs (source entities), and keep those compositions that refer to the exact same data.

### 4.3 Potential Model Clones Categorization

In this phase, we categorize all the retrieved subgraph instances (potential model clones), in order to facilitate the quality evaluation procedure of the overall application conceptual schema, presented in the next section.

More precisely, for every instance of the hypertext configurations mined in the first case of graphs, we make a first level categorization according to the source entities and attributes that the WebML elements of the configurations refer to. To accomplish that, we utilize the information provided by the XML definition of each site view, where there is a detailed description of the source entities and the selectors of each element included in the site view (Ceri et al. 2002). For a specific

configuration retrieved, we categorize its instances in the various site views of the application, as follows:

Configurations constituted by WebML elements referring to:

- exactly the same source entities and attributes,
- exactly the same source entities but different attributes (in the worst case the only common attribute is the object identifier(OID),
- partially identical source entities (i.e. Fig. 2),
- different source entities.

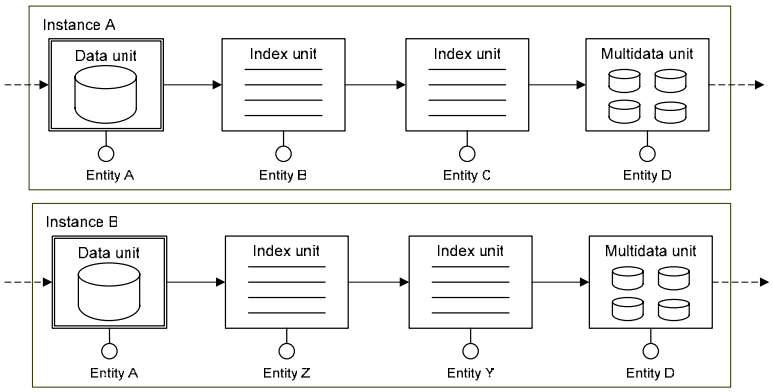



**Fig. 2. Categorizing potential model clones at the Hypertext Level**

We also categorize (exploiting the XML definition of site views) every instance of the data element configurations acquired by the graphs representing the data distribution as:

Configurations constituted by source entities utilized by:

- different WebML elements,
- similar WebML elements, that is elements of the same type such as composition or content management (e.g. in Fig. 3), entity A is utilized by two composition units, a multidata and an index).
- identical WebML elements.

The last category captures exactly the same hypertext configurations as the first case of the previous categorization.

Finally, potential model clones are the sets of hypertext configurations retrieved in the third step of the previous phase, where compositions of identical WebML elements referring to common data sources, utilizing different link topologies, have been identified (Fig. 4).

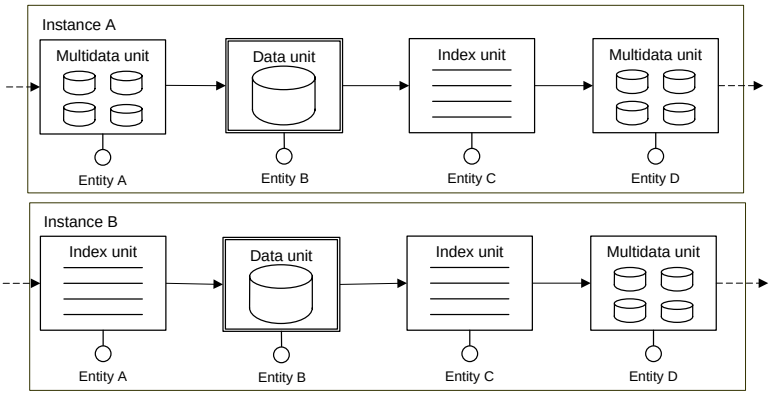



**Fig. 3. Categorizing potential model clones at the Data Level**

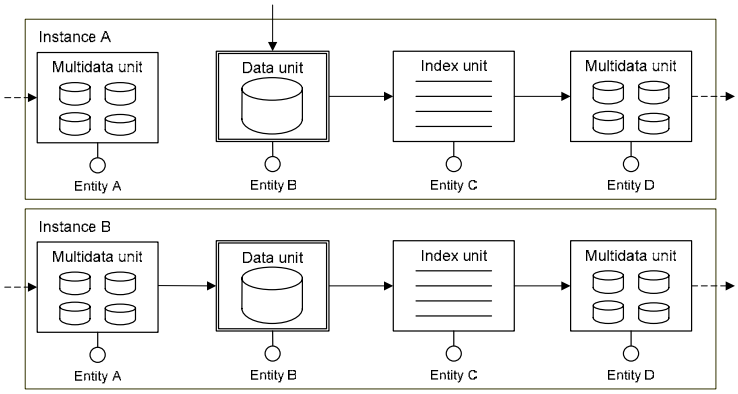



**Fig. 4. Potential model clones based on link topology**

## 5 Metrics and Quality Evaluation

Having identified the potential model clones at the hypertext and data representation level, we provide metrics for categorizing the application conceptual schema through a quality evaluation which specifies factors (referring to the identified clones) denoting possible need for refactoring. The evaluation is threefold: First, we evaluate the model clones by means of consistency, based on their variants throughout the conceptual schema. Second, we quantify the categorization presented in section 4.3 based on the similarity among potential model clones and third we make an evaluation based on the topology of links.

### 5.1 Consistent Use of Model Clones

Fraternali et al. (2002) have introduced a methodology for the evaluation of the consistent application of predefined WebML design patterns, within the conceptual schema of an application. We utilize this methodology and extend it, in order to introduce metrics depicting the consistent application of design solutions (i.e. model clones) retrieved during the first step of the procedure described in section 4.2. To achieve that, we introduce two metrics that represent the statistical variance of the occurrence of *N* termination or starting variants of the identified model clones, normalized according to the best case variance. These metrics are called *Start-Point Metric (SPM*) and *End-Point Metric (EPM)* respectively. Following there is the formal definition of SPM (EPM is defined in an analogous way):

$$SPM = \sigma^2 \Big/ \sigma_{BC}^2 \tag{1}$$

In equation 1 $\sigma^2$is the statistical variance of the starting variants occurrences, which is calculated according to the following formula:

$$\sigma^2 = (\frac{1}{N})\sum_{i=0}^{N}(p_i - \frac{1}{N})^2 \tag{2}$$

where $N$ is the number of variants addressed by the metric, while $p_i$ is the percentage of occurrences for the *i-th* configuration variant. $\sigma_{BC}^2$ is the best case variance, and it is calculated by equation 2, assuming that only one variant has been coherently used throughout the application.

The last step in the metrics definition is the creation of a measurement scale, which defines a mapping between the numerical results obtained through the calculus method and a set of (predefined) meaningful and discrete values. According to the scale types defined in the measurement theory (Roberts 1979), the SPM metric adopts an ordinal nominal scale; each nominal value in the scale expresses a consistency level, corresponding to a range of numerical values of the metrics as defined in Table 2. The same scale covers the EPM metric as well.

**Table 2. The SPM metric measurement scale**

| Metric's Range | Measurement Scale Value |
|---|---|
| 0 <= SPM<0.2 | Insufficient |
| 0.2<= SPM<0.4 | Weak |
| 0.4<= SPM<0.6 | Discrete |
| 0.6<= SPM<0.8 | Good |
| 0.8<=SPM<=1 | Optimum |

### 5.2 Similarity of Model Clones

To evaluate the similarity between two hypertext configuration instances (potential model clones either with the same entities or the same WebML elements), we adopt the vector model and compute the degree of similarity (Baeza-Yates and Ribeiro-Neto 1999). To convert the initial configurations into vectors in the vector space model we define the vector $\vec{d}_i = (x_{i1}, x_{i2}, x_{i3} \ldots x_{in})$, where the compounds comprise all distinct WebML elements or entities of all configurations under consideration (for instance in figure 2, there are six distinct entities, while in figure 3, there are four). These compounds are considered as unigrams and are weighted by the frequency $\varphi$ of each respective unigram. In the case of entities that appear with different attributes, their unigram is transformed to a fraction of frequency φ proportionate to the number of the participating attributes, as opposed to the number of all designed attributes. The vector space model proposes to evaluate the degree of similarity of a configuration $d_m$ and a configuration $d_\mu$ (with the same entities or WebML elements) as the correlation between vectors $\vec{d}_m$ and $\vec{d}_\mu$. This correlation can be quantified by the cosine angle between these two vectors, that is:

$$\text{similarity}(d_m, d_\mu) = \cos(\vec{d}_m, \vec{d}_\mu) = \frac{\vec{d}_m \cdot \vec{d}_\mu}{\left|\vec{d}_m\right| \times \left|\vec{d}_\mu\right|} \Rightarrow$$

$$\text{similarity}(d_m, d_\mu) = \frac{\sum_{t=1}^{t} w_{i,m} \times w_{i,\mu}}{\sqrt{\sum_{t=1}^{t} w_{i,m}^2} \times \sqrt{\sum_{t=1}^{t} w_{i,\mu}^2}}, \in [0,1] \quad (3)$$

where $\left|\vec{d}_m\right|$ and $\left|\vec{d}_\mu\right|$ are the norms of the two configurations.

Potential clones can be categorized as relevant or not, using the vector space model (that ranks them according to the degree of similarity between them).
In the following, we propose the different thresholds and corresponding categorization based on the level of clones *similarity*($d_m$, $d_\mu$). This second stage of our evaluation approach uses as input the instances of the hypertext configurations retrieved. The goal of this stage is to check whether instances can be actually considered clones, and identify the opportunities of refactoring. Results of this step are depicted in ordinal scaled categories, as presented below.

Initially, configuration instances retrieved by the first set of graphs are classified according to the clone classification scheme shown in Table 3.

**Table 3. Classification of potential model clones based on the hypertext model**

| Level | Configurations of WebML elements referring to: |
|---|---|
| 1 | the same source entities and attributes. |
| 2 | source entities that are identical up to 75%. |
| 3 | source entities that are identical up to 50%. |
| 4 | source entities that are identical up to 25%. |
| 5 | totally different source entities. |

Classification follows an ordinal scale based on the degree of similarity between the potential clones. When the similarity degree increases, the probability having detected an actual model clone decreases. Therefore, the higher the level's value is, the smallest the probability to have identified a potential model clone gets.

Next, the configuration instances retrieved by the second set of graphs are classified according to the clone classification scheme as depicted in Table 4.

**Table 4. Classification of potential model clones based on the distribution of data**

| Level | Configurations constituted by source entities utilized by: |
|---|---|
| 1 | WebML elements of the same type, which are identical. |
| 2 | WebML elements of the same type, which are identical up to 75%. |
| 3 | WebML elements of the same type, which are identical up to 50%. |
| 4 | WebML elements of the same type, which are identical up to 25%. |
| 5 | totally different WebML elements. |

Likewise, the classification follows an ordinal scale based on the degree of similarity between the candidate model clones. Checking proceeds from level 1 to 5 and stops as soon as the adequate level has been recognized. Again, the probability to have identified a model clone increases as the level's value decreases.

### 5.3 Evaluation of the Link Topology

To further refine the evaluation procedure, we examine the hypertext configurations of candidate model clones within a page, based on the topology of their link connectivity. Intuitively, similarity of link topology at a local (within a page) or a broader (within an area or site view) hypertext level implies similarity in the business logic. When the differences are broader, then either the business logic is changed or potential misconfigurations are detected. We define *link topology similarity* as the cosine angle between vectors of the form $\vec{v}_i = (x_{i1}, x_{i2}, x_{i3} \ldots x_{in})$, where $n$ is the number of different link connections in the hypertext model of candidate clones with the same WebML elements (in terms of instances and total number). The unigram compounds are assigned value "1" when the link

is of the same direction, value ‘0” if the link does not exist at all, and value “-1” if the link creates an opposite direction element connection. As a result, we classify the hypertext configurations retrieved in the third step of the mining phase, in accordance to their link topology similarity as depicted in Table 5.

**Table 5. Classification of potential model clones based on the links topology**

| Level | Description |
|---|---|
| 1 | Very high similarity in terms of link topology (equivalent). |
| 2 | 75% similarity in terms of link topology (high). |
| 3 | 50% similarity in terms of link topology (semi). |
| 4 | 25% similarity in terms of link topology (low). |
| 5 | Totally different link topology (disjoint). |

In this case also, we follow an ordinal scale for the classification scheme.

## 6 Refactoring Opportunities

After the classification of potential model clones according to their similarity, the hypertext architect has a first set of metrics pointing out “hot spots” for possible improvements in the conceptual schema of the application. The potential model clones are ranked according to their similarity level and their size (number of elements constituting the configuration[1]), thus forming a valuable tool providing quality improvement guidelines for the overall model. The methodology can be extended to support automatic model refinement, but there are a number of issues that need to be considered first. If an automated model refactoring process is to be deployed, checks for the syntactic and semantic correctness (e.g. conflicts, racing conditions, deadlocks) should be performed. Moreover, since personalization may be delivered in various forms and various levels of the design process (e.g. navigation or content personalization) automatic model refactoring could lead to inconsistencies with respect to initial functional requirements of the application. Nevertheless, a number of refactoring suggestions are straightforward.

Based on the first classification, the existence of two or more model clones belonging to Level 1 implies a very high refactoring opportunity. The designer should keep only one of the model clones, utilizing the “change siteview” unit if the clones are in different site views, or reconsider the link topology if the clones are in the same site view. In case that the WebML elements of two or more clones refer to exactly the same entities and differ to some of the attributes, merging all the attributes in every element should be considered (provided that content personalization is not affected). If the clones are categorized from levels 2 to 4, the designer should consider refactoring the hypertext schema by merging pages or areas.

[1] Larger configurations denote a higher probability of having identified a model smell.

On the other hand, the configurations categorized as level 5, might lead to capturing cases of effective application of a design solution or a design pattern. These configurations imply effective use of previous experience that leads to a high degree of consistency within the application and promotion of usability. These are cases where the application gains in terms of quality. In this category of model clones, refactoring should be performed when SPM and/or EPM metrics values are under 0.6, by using where possible the variant having the larger occurrence frequency.

The various similarity levels computed during the second classification denote cases where the same data are presented to users (or manipulated by them) by means of different presentation (or content management) mechanisms. High similarity between clones implies "hot spots" where the designer should examine redesigning the hypertext in order to accomplish consistency in the presentation level and enhance usability of the final application.

Likewise, the last classification identifies cases in which the link topology of segments of the application model should be reconsidered, in order to accomplish consistency in the navigation and presentation level. Model clones belonging to the lower categorization levels should be considered for possibility of being merged by reordering links. Table 6 summarizes the refactoring proposals based on the categorization of model clone configurations that were presented in the previous sections.

## 7 Conclusions & Future Work

In this paper we have illustrated a methodology that aims at capturing potential problems caused by inappropriate reuse, within the conceptual schema of a Web application. We have introduced the notions of model cloning and model smells, and provided a technique for identifying model clones within an application's hypertext model by mapping the problem to the graph theory domain.

Moreover, we have specified a set of evaluation metrics quantifying the inappropriate reuse of clones and proposed refactoring recommendations. Applying the methodology can result in higher consistency and usability levels in the target application, as well as in a productivity increase in the application design and maintenance process.

Even though the quality of conceptual schemas depends at a high degree on the selected modeling language, the proposed methodology may be used by a series of languages with minor straightforward adjustments.

The proposed methodology extends the analytical application framework as presented by (Comai et al. 2002; Fraternalli et al. 2002, 2004a, 2004b, 2004c; Lanzi et al. 2004), as apart from evaluating the consistency level in using design patterns, it also detects a number of factors that may cause design and quality problems at all levels of application modeling.

**Table 6.** Refactoring proposals based on the similarity levels of the model clones

| | Level | Description | Refactoring Proposals |
|---|---|---|---|
| Configurations of WebML elements referring to: | 1 | The same source entities and attributes. | Only one of the model clones should be kept, utilizing the "change siteview" unit if the clones are in different site views, or reconsider the link topology if the clones are in the same site view. |
| | | Exactly the same entities with differences at some of the attributes. | Merging all the attributes in every element should be considered and deletion of the remaining clones. |
| | 2 | Source entities identical up to 75%. | Merging of pages and areas should be considered. Priority should be given to clones belonging to lower levels. |
| | 3 | Source entities identical up to 50%. | |
| | 4 | Source entities identical up to 25%. | |
| | 5 | Different source entities. | Consider refactoring when SPM and/or EPM metrics values are under 0.6, by using where possible the variant having the larger occurrence frequency. |
| Configurations constituted by source entities utilized by: | 1 | WebML elements of the same type, which are identical. | Same as for level 1 in the previous case. |
| | 2 | WebML elements of the same type, up to 75% similar. | Examine redesigning the hypertext in order to accomplish consistency in the presentation level and enhance usability of the final application. Priority should be given to clones belonging to lower levels. |
| | 3 | WebML elements of the same type, up to 50% similar. | |
| | 4 | WebML elements of the same type, up to 25% similar. | |
| | 5 | Different WebML elements. | No refactoring should be considered. |
| Link Topology Similarity: | <2 | | Examine the possibility of merging clones by restructuring the link topology. |

In the future we plan to apply the methodology to a large number of Web application conceptual schemas, in order to refine it and fine-tune the model clone evaluation metrics. We will also consider the distribution and effect of design patterns within a conceptual schema in accordance with the process of model clones identification. Moreover, we plan to investigate the representation of a conceptual model at a higher level of abstraction based on the existence of design patterns and model clones. Finally, we intend to extend the methodology in order to support automatic model refinement assuring at some level syntactic and semantic correctness, based on the evaluation results.

## References


Antoniou G, Casazza G, Di Penta M, Merlo E (2001) Modeling Clones Evolution Through Time Series In the Proceedings of the International Conference on Software Maintenance, Florence, Italy, pp 273-280

Atzeni P, Mecca G, Merialdo P (1998) Design and Maintenance of Data-Intensive Web Sites In the Proceedings of the 6th International Conference on Extending Database Technology – EDBT'98, pp 436-450

Baeza-Yates R, Ribeiro-Neto B (1999) Modern Information Retrieval. Addison Wesley, ACM Press

Baker BS (1995) On Finding Duplication and Near-Duplication in Large Software Systems. In the Proceedings of the Second Working Conference on Reverse Engineering, Toronto, Canada, pp 86-95

Balazinska M, Merlo E, Dagenais M, Lague B, Kontogiannis K (1999) Measuring Clone Based Reengineering Opportunities. In the Proceedings of the 6th IEEE International Symposium on Software Metrics, Boca Raton, USA, pp 292-303

Baxter ID, Yahin A, Moura L, Santa Anna M, Bier L (1998) Clone Detection Using Abstract Syntax Trees. In the Proceedings of the International Conference on Software Maintenance, Washington DC, USA, pp 368-377

Boehm B (1981) Software Engineering Economics. Prentice Hall PTR

Boldyreff C, Kewish R (2001) Reverse Engineering to Achieve Maintainable WWW Sites. In the Proceedings of the 8th Working Conference on Reverse Engineering (WCRE'01), Stuttgart, Germany, pp 249-257

Booch G, Jacobson I, Rumbaugh J (1998) The Unified Modeling Language User Guide. The Addison-Wesley Object Technology Series

Ceri S, Fraternali P, Bongio A (2000) Web Modeling Language (WebML): a Modeling Language for Designing Web Sites. In the Proceedings of WWW9 Conference, Amsterdam

Ceri S, Fraternali P, Bongio A, Brambilla M, Comai S, Matera M (2002) Designing Data-Intensive Web Applications. Morgan Kauffmann

Chikofsky E, Cross J (1990) Reverse engineering and design recovery: A taxonomy. IEEE Software 7(1): 3-17

Coleman DM, Ash D, Lowther B, Oman, PW (1994) Using Metrics to Evaluate Software System Maintainability. Computer 27(8): 44-49

Comai S, Matera M, Maurino A (2002) A Model and an XSL Framework for Analysing the Quality of WebML Conceptual Schemas. In Proc of IWCMQ'02 - ER'02 International Workshop on Conceptual Modeling Quality, Tampere, Finland, October 2002

Conallen J (1999) Modeling Web application architectures with UML. Communications of the ACM 42(10): 63–70

Conallen J (1999) Building Web Applications with UML. Addison-Wesley, Reading MA

Di Lucca GA, Di Penta M, Fasolino AR, Granato P (2001) Clone Analysis in the Web Era: an Approach to Identify Cloned Web Pages. In the Proceedings of the 7th IEEE Workshop on Empirical Studies of Software Maintenance, Florence, Italy, pp 107-113

Ducasse S, Rieger M, Demeyer S (1999) A language independent approach for detecting duplicated code. In the Proceedings of the International Conference on Software Maintenance, IEEE Computer Society Press, Oxford, UK, pp 109–118

Fernandez MF, Florescu D, Kang J, Levy AY, Suciu D (1998) Catching the Boat with Strudel: Experiences with a Web-Site Management System. In the Proceedings of ACM-SIGMOD Conference, pp 414-425

Fowler M, Beck K, Brant J, Opdyke W, Roberts, D (1999) Refactoring: Improving the Design of Existing Code. The Addison-Wesley Object Technology Series

Fraternali P, Paolini, P (1998) A Conceptual Model and a Tool Environment for Developing More Scalable, Dynamic, and Customizable Web Applications. In the Proceedings of the 6th International Conference on Extending Database Technology – EDBT'98, pp 421-435

Fraternali P, Lanzi PL, Matera M, Maurino A (2004) Exploiting Conceptual Modeling for Web Application Quality Evaluation. In the Proceedings of WWW 2004 Alternate Tracks and Posters, New York, USA, May 2004

Fraternali P, Lanzi PL, Matera M, Maurino A (2004) Model-Driven Web Usage Analysis for the Evaluation of Web Application Quality. Submitted for publication to Journal of Web Engineering, April 2004

Fraternali P, Matera M, Maurino A (2002) WQA: an XSL Framework for Analyzing the Quality of Web Applications. In the Proceedings of the 2nd International Workshop on Web-Oriented Software Technologies - IWWOST'02, Malaga, Spain, pp 46-61

Fraternali P, Matera M, Maurino A (2004) Conceptual-Level Log Analysis for the Evaluation of Web Application Quality. In the Proceedings of LA-Web Conference, IEEE Press, Cile, November 2004

Garey MR, Johnson DS (1979) Computers and Intractability: A guide to NP-Completeness. New York: Freeman

Garzotto F, Paolini P, Schwabe D (1993) HDM - A Model-Based Approach to Hypertext Application Design. ACM Transactions on Information Systems 11(1): 1-26

Guimaraes T (1983) Managing Application Program Maintenance Expenditure. Communications of the ACM 26(10): 739-746

Isakowitz T, Sthor EA, Balasubranian P (1995) RMM: a methodology for structured hypermedia design. Communications of the ACM 38(8): 34-44

Johnson JH (1993) Identifying Redundancy in Source Code using Fingerprints. In the Proceedings of the CAS Conference, Toronto, Canada, pp 171–183

Johnson JH (1994) Substring Matching for Clone Detection and Change Tracking. In the Proceedings of the International Conference on Software Maintenance, Victoria, Canada, pp 120-126

Kamiya T, Kusumoto S, Inoue K (2002) CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code. IEEE Transactions On Software Engineering 28(7): 654–670

Kontogiannis K (1997) Evaluation Experiments on the Detection of Programming Patterns Using Software Metrics. In the Proceedings of the 4th Working Conference on Reverse Engineering, Amsterdam, The Netherlands, pp 44-54

Krinke J (2001) Identifying Similar Code with Program Dependence Graphs. In the Proceedings of the 8th Working Conference on Reverse Engineering, Stuttgart, Germany, pp 301-309

Lague B, Proulx D, Mayrand J, Merlo EM, Hudepohl J (1997) Assessing the Benefits of Incorporating Function Clone Detection in a Development Process. In the Proceedings of the International Conference on Software Maintenance, Bari, Italy, pp 314-321

Lanzi PL, Matera M, Maurino A (2004) A Framework for Exploiting Conceptual Modeling in the Evaluation of Web Application Quality. In the Proceedings of the International Conference of Web Engineering-ICWE'04, LNCS 3140, Springer Verlag Publ, Munich, Germany, July 2004

Manber U (1994) Finding Similar Files in a Large File System. In the Proceedings of the USENIX Winter 1994 Technical Conference, San Francisco, USA, pp 1-10

Mayrand J, Leblanc C, Merlo EM (1996) Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics. In the Proceedings of the International Conference on Software Maintenance, Monterey, USA, pp 244-254

Mens T, Demeyer S, Du Bois B, Stenten H, Van Gorp P (2003) Refactoring: Current research and future trends. In the Proceedings of the 3rd Workshop on Language Descriptions, Tools and Applications (LDTA 2003), April 6, Warsaw, Poland

Mens T, Tourwe, T (2004) A survey of software refactoring. IEEE Transactions on Software Engineering, 30(2): 126-139

Ricca F, Tonella P (2003) Using Clustering to Support the Migration from Static to Dynamic Web Pages. In the Proceedings of the 11th International Workshop on Program Comprehension, Portland, USA, pp 207-216

Rieger M, Ducasse S (1998) Visual Detection of Duplicated Code. In the Proceedings of the Workshop on Experiences in Object-Oriented Re-Engineering, Brussels, Belgium, pp 75-76

Roberts D (1999) Practical Analysis for Refactoring. Ph.D. thesis, University of Illinois at Urbana-Champaign

Sakkopoulos E, Sirmakessis S, Tsakalidis A, Tzimas G (2005) A Methodology for Evaluating the Personalization Conceptual Schema of a Web Application. in the Proceedings of the 11th International Conference on Human-Computer Interaction (HCI International 2005), July 22-27, Las Vegas, Nevada, USA

Schwabe D, Rossi G (1998) An object-oriented approach to web-based application design. Theory and Practice of Object Systems (TAPOS) 4(4): 207-225

Wang C, Wang W, Pei J, Zhu Y, Shi B (2004) Scalable Mining of Large Disk-based Graph Databases. In the Proceedings of ACM KDD04, pp 316-325

WebRatio (2005), available at: http://wwwwebratiocom

Yan X, Han J (2002) gSpan: Graph-based substructure pattern mining. In the Proceedings of International Conference on Data Mining (ICDM'02), Maebashi, pp 721-724

Yan X, Han J (2003) CloseGraph: mining closed frequent graph patterns. In the Proceedings of ACM KDD03, pp 286-295