Journal of Web Engineering, Vol. 7, No. 1 (2008) 070-092

A SITUATIONAL METHODOLOGY FOR ADDRESSING THE PRAGMATIC QUALITY OF WEB APPLICATIONS BY INTEGRATION OF PATTERNS

PANKAJ KAMTHAN

Department of Computer Science and Software Engineering, Concordia University, Montreal
kamthan@encs.concordia.ca

Received December 8, 2006
Revised February 3, 2008

The development and evolution of Web Applications is viewed from an engineering perspective. A Pattern-Oriented Web Engineering Methodology (POWEM) for deploying patterns as means for assuring the quality of Web Applications is presented. POWEM consists of a sequence of steps including setting goals, identifying stakeholder types, following a suitable development process model, identifying relevant quality attributes, and acquiring, selecting and applying suitable patterns. The feasibility issues involved in each step are examined. The use of patterns during macro- and micro-architecture design of a Web Application is illustrated. Finally, some directions for future research, including extensions to POWEM, are outlined.

Key words: Feasibility, Iterative Development, Pattern, Quality Model, Semiotics, Stakeholder
Communicated by: B. White and M. Gaedke

1 Introduction

The Web has opened new vistas for many sectors of society including education, businesses, and government. The success of the Web, however, has also come with its share of failures, many of which are attributed to issues related to quality (Pertet & Narasimhan, 2005).

A Web Application aims to serve its stakeholders by providing some value. This value is potentially threatened if the quality of the Web Application is compromised. In order to elevate, restore, and/or sustain the confidence of stakeholders, it is incumbent upon the providers to ensure the quality of Web Applications, and to do so in a viable manner.

In this paper, the quality of Web Applications as viewed by the stakeholders is addressed. To do that, Web Applications are considered as end-products of a systematic and feasible approach that, during development, incorporates a form of a posteriori knowledge, namely patterns (Buschmann, Henney, & Schmidt, 2007).

The organization of the rest of the paper is as follows. In Section 2, the background and state-of-the-art necessary for the discussion that follows is outlined, and the position taken by the author is stated. This is followed by the presentation of a Pattern-Oriented Web Engineering Methodology (POWEM) for systematically addressing the pragmatic quality (Lindland, Sindre, & Sølvberg, 1994) of Web Applications in Section 3. POWEM consists of a non-linear sequence of steps including the identification of stakeholder types, suitable development process model, decomposition of pragmatic quality into relevant attributes, and systematic selection and application of suitable patterns. Specifically, with regard to organizational, social, or technical decisions, the feasibility of each step in POWEM is considered. In Section 4, some directions for future research and challenges in pursuing them are highlighted. Finally, in Section 5, the concluding remarks are presented.

2 Background and Related Work

This section presents a synopsis of the need for a systematic approach to the development of Web Applications from the perspective of quality and the role of patterns. For the sake of this paper, a Web Application is viewed as an interactive software system specific to a domain that will typically have a large-size and require a non-trivial infrastructure for development, operation, and maintenance.

2.1 Understanding and Addressing the Quality of Web Applications

It is critical to all stakeholders involved that a Web Application exhibit "high" quality. If unaddressed, there is a potential, for example, for a resource in a Web Application to be rendered unreadable on a user agent of a customer, be inaccessible to someone who is visually impaired, or be prohibitive to adaptive maintenance by an engineer.

There have been various initiatives for understanding and addressing the quality of Web Applications (Brajnik, 2001; Offutt, 2002; Mich, Franch, & Gaio, 2003; Kappel et al., 2006; Mendes & Mosley, 2006). They cover a range of topics such as furthering the understanding of the concept of quality by suggesting a quality model that consists of (a tree of) quality attributes; discussing the significance of each quality attribute, thereby rationalizing its inclusion in the quality model; the impact of a quality attribute on a class of stakeholders for specific Web Applications; and so on.

However, these efforts are restricted by one or more of the following issues: the view towards quality is at times not explicitly stated; the relationships, including trade-offs, among quality at-tributes relevant to Web Applications are not always indicated; or in addressing the quality at-tributes, the focus, if at all, is less on assurance and more on evaluation.

2.2 Patterns for Engineering Web Applications

The reliance on past experience and expertise is critical to any development. A pattern is commonly defined as a proven solution to a recurring problem in a given context (Buschmann, Henney, & Schmidt, 2007). For an expert, patterns are means to share knowledge and act as a reference "handbook"; for a novice, patterns can be useful as source of guidance, learning, and exploration.

From a structural viewpoint, a pattern is typically described using a set of ordered elements labeled as (pattern) *name*, *author*, *context*, *problem*, *forces*, *solution*, *examples*, and *related patterns*. At times, the labels of these mandatory elements may vary in the literature. Furthermore, optional elements, such as those related to metadata, may be included to enrich the description. In the rest of the paper, the elements of a pattern are highlighted in italics.

It is seldom that a pattern exists in isolation, and is often implicitly or explicitly related to other patterns. Indeed, related patterns are often organized in a pattern catalog, in a pattern system, or in a pattern language.

Patterns and Process

There have been some initiatives previously for incorporating patterns in a development process of a Web Application.

It has been suggested that the patterns for navigation design (Rossi, Schwabe, & Lyardet, 1999) could be integrated in the Object-Oriented Hypermedia Design Method (OOHDM). However, there are several outstanding issues with OOHDM including that it does not address feasibility concerns, does not explicitly and systematically approach quality concerns, and lacks broad tool support for some of the design notations it suggests.

The Web Modeling Language (WebML) has been used to express the *solution*s of certain design patterns (Fraternali, Matera, & Maurino, 2002) as conceptual models. However, the precise impact of

these patterns on quality attributes and the relationship between patterns and quality attributes is not shown.

A pattern-based software development process based on case-based reasoning (CBR) has been discussed and applied to an Internet chat application (Wentzlaff & Specker, 2006). However, besides *problem* and *solution*, it does not take other mandatory elements of a pattern into consideration, and it does not explicitly suggest any implications towards quality.

Patterns and Quality

There have been some efforts earlier in connecting quality-related concerns of Web Applications and patterns.

There are some patterns discussed from the perspective of maintainability of Web Applications (Weiss, 2003). However, the *solution*s of some patterns are highly technology-specific, the mandatory elements of a pattern are not always appropriately documented, and the integration of patterns into any development process is not mentioned.

There are also some patterns discussed from the perspective of usability of Web Applications (Perzel & Kane, 1999; Lyardet & Rossi, 2001; Graham, 2003). However, these efforts are restricted by one or more of the following issues: usability is viewed as an atomic (non-decomposable) concept, the patterns are strongly oriented towards user interface design, the mandatory elements of a pattern are not always suitably documented, the feasibility of using patterns is not discussed, or the integration of patterns into any (say, user-centered) development process is not outlined explicitly.

3 A Methodology for Integrating Patterns in Web Engineering

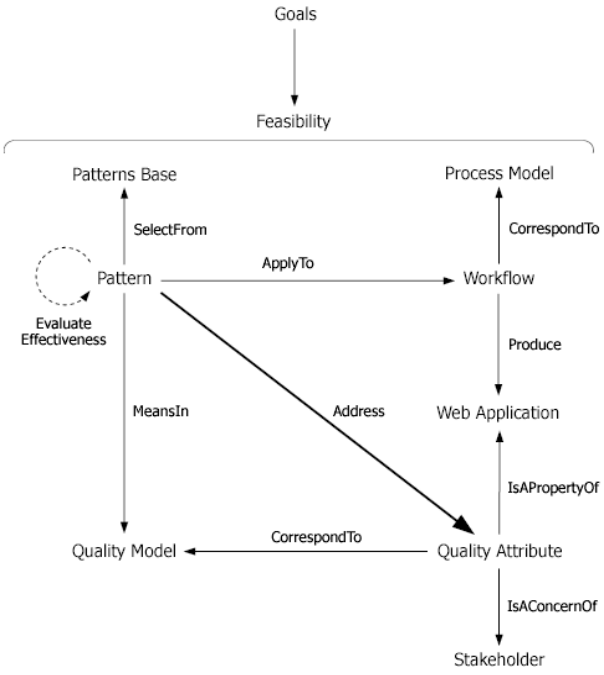

Figure 1. A high-level view of a pattern-oriented, stakeholder-quality-centric, and feasibility-sensitive approach to engineering Web Applications.

This section proposes a methodology, namely POWEM, inspired by the notions of patterns and situational method engineering (Kumar & Welke, 1992), for addressing the quality of Web Applications. POWEM rests on the following interrelated hypothesis:

- **Hypothesis 1.** The aim of developing "high" quality Web Applications is a *first-class* consideration in the organization.
- **Hypothesis 2.** A preventative approach to quality is at least as significant as a curative approach (Dromey, 2003).

Figure 1 gives an overview of POWEM with the placement of (abbreviated versions of) steps in it. POWEM consists of six mandatory steps, namely (1) *Setting Goals*; (2) *Identifying Stakeholders*; (3) *Selecting the Development Process Model*; (4) *Identifying and Organizing Quality Concerns*; (5) *Acquiring and Selecting Patterns*; and (6) *Applying Patterns*. These could be followed by an optional step of *Evaluating Effectiveness*.

Remark 1. POWEM bears some resemblance to the Goal/Question/Metric (GQM) method (Van Solingen & Berghout, 1999), a rigorous means to conduct a quality improvement program. For example, step 1 could be refined into sub-goals which could then be used to ask questions that in turn can help towards the steps involving the selection of the development process model, relevant quality concerns, and suitable patterns.

3.1 Characteristics of POWEM

Before divulging into details, certain distinctive characteristics of POWEM need to be briefly pointed out.

First, POWEM is essentially independent of the final product, namely the Web Application under development. This makes POWEM broadly applicable.

Second, from a practical standpoint of integrating patterns in Web Applications, the following must be feasible: the personality types of the stakeholders, the development process, the expectations of improving the pragmatic quality concerns, and the adoption/deployment of patterns as means for addressing these concerns. Each of these is discussed in detail in the sections that follow. The feasibility study could be a part of the overall Web Application project management planning activity. Further discussion of this aspect is beyond the scope of this paper.

Third, a pattern can be viewed as *knowledge* that is *conceptually* reusable. To that regard, the *dynamics* of knowledge creation and transfer in steps 5 and 6 in POWEM is based upon the interaction between tacit and explicit knowledge. Indeed, following the approach of the Socialization, Externalization, Combination, Internalization (SECI) model (May & Taylor, 2003), steps 5 and 6 in POWEM fall into the interplays of Internalization (explicit-to-tacit knowledge) and Combination (explicit-to-explicit knowledge).

In the following sections, the work involved in each of these steps is discussed in detail.

3.2 Setting Goals

A goal is the result towards which an effort is directed. It has been hypothesized that a project without clear goals will not achieve its goals clearly (Gilb, 1988). A POWEM approach must include high-level organizational and/or technical goals that need to be accomplished.

For instance, an organizational goal could be to achieve success (improve visibility, sustain and/or enhance consumer satisfaction, persuade new consumers, and so on, in the long-term) and a technical goal for the overall development strategy could be to rely on organizational memory (like patterns) to produce Web Applications that meet the stakeholder expectations of quality.

Feasibility of Goals

A goal needs to be operationalized to be in realistic. This is evidently related to feasibility. A goal could either be *hard* or *soft*. A hard goal is either *satisfied* or not satisfied. A soft goal can only be satisfied to a certain degree, that is, *satisficed* (Simon, 1996).

A high-level goal is useful to set the overall picture from which the rest of the steps will follow. It may, however, be difficult to attain a high-level goal directly. If a goal is at a level that is deemed too high, it may need to be decomposed into manageable sub-goals. As an example, sub-goals of the aforementioned technical goal of an engineer could be to find a development process model that embraces the inclusion of patterns and find pattern(s) that contribute to modifiability of a specific Web Application.

3.3 Identifying Stakeholders

A stakeholder is a person or organization who influences a software system or who is impacted by that system. In our case, there are two broad classes of stakeholders that can be identified and form a partition with respect to their *roles* in relationship to a Web Application: a *producer* (such as an owner, legislator, manager, engineer, or maintainer) is the one who owns, manages, develops or maintains the Web Application, and a *consumer* (such as novice or expert user) is the one who interacts with the Web Application for some purpose.

There are a few systematic approaches for identification and refinement of stakeholder classes. The use cases (Jacobson et al., 1992) and Viewpoint-Oriented Requirements Definition (VORD) (Kotonya & Sommerville, 1998) give us two established ways of classifying aforementioned stakeholders of interactive systems in a client-server environment such as Web Applications. From a use case perspective, a consumer is a human actor. From a VORD perspective, owner and manager are indirect viewpoints, while the engineer and maintainer are direct viewpoints in their relationship to the Web Application.

Feasibility of Stakeholders

The selection and application of patterns is essentially a human activity involving *problem solving*, and therefore the constraints of human characteristics pertaining to it need to be taken into account.

The studies in human psychology have shown that variations among people with respect to their predispositions, abilities, and knowledge, need to be accepted and managed (Keirsey, 1998). A common measure of (variations in) personality is the Myers-Briggs Type Indicator (MBTI) that measures an individual's preference on four bipolar dimensions: introversion/extraversion (I/E), intuition/sensing (I/S), feeling/thinking (F/T), and judgment/perception (J/P). An MBTI personality type consists of a four-letter code, such as ESTJ (Extraverted Sensing Thinking Judging), to indicate the personality type of an individual. Based on controlled empirical experiments, it has been hypothesized (Boreham, 1987) that in problem solving, people with S or T in their personality type are more likely to perform better than people with E or F, respectively. Since one of the views of patterns is that they are a problem solving approach, it is assumed in the following that the producers (especially, engineers and maintainers) have compatible personality types with respect to appropriate steps in POWEM.

3.4 Selecting the Development Process Model

The inclusion of patterns in the development of Web Applications *cannot* be ad-hoc or an afterthought, and should be carried out within the auspices of a suitable process model. The underlying assumption is that an improvement in the *process* for development can bring about improvement in the *product*, namely the Web Application itself.

Indeed, the use of POWEM within the framework of an *existing* process environment with an implicit or explicit support for patterns is recommended. Furthermore, the following properties of the process environment are desirable: it is *human-centric* as the Web Applications are inherently interactive, customer involvement is critical, and user base is often broad; it is *flexible* (for example, its workflows can proceed iteratively and incrementally) as the requirements of commercial Web Applications are particularly susceptible to change (for example, due to market demands); and it has *broad community and tool support* in order to keep the cost to a minimum and the learning curve low.

To that regard, the adoption of one of the following process models is suggested. Extreme Programming (XP) (Beck & Andres, 2005) is a broadly-used and well-tested agile methodology for software development. XP places "lightweight" demands on resources and is suitable for small projects. The Unified Process (UP) (Jacobson, Booch, & Rumbaugh, 1999) is an archetype of model-based and use case-driven process *framework*, instances of which are the Rational Unified Process (RUP), and its customization for educational environment, the Unified Process for EDUcation (UPEDU). RUP requires heavy modeling and documentation and is especially suited for large projects. Both XP and RUP have been "tailored" to Web Applications (Kappel et al., 2006). There is some provision of the use of patterns during macro- and micro-architecture design in both XP and RUP and, by reference, their extensions for Web Applications.

Feasibility of Development Process Model

The process model for the development of a Web Application and the realization of the activities recommended therein will evidently depend on the level of organizational process maturity such as the appropriate level in the Capability Maturity Model for Software (SW-CMM) (Paulk et al., 1995). This in turn is determined by several factors, including emphasis on quality in the organization to which the constraints of Hypothesis 1 and 2 apply.

There have been relatively few studies that conclusively relate XP or RUP to SW-CMM. The precise mapping of the use of patterns in an organization to SW-CMM remains an open question but is likely to be at least Level 2.

3.5 Identifying and Organizing Quality Concerns

There are different possible views of quality (Wong, 2006). The viewpoint for quality of Web Applications for the purpose of this paper is *semiotics* (Stamper, 1992). Among the proposals for semiotic quality, the treatment in one of the well-known approaches (Lindland, Sindre, & Sølvberg, 1994) is adopted and extended.

Semiotic Level	**Quality Attributes**	**Means for Quality Assurance**
Pragmatic	[Tier 3] Maintainability, Usability	Patterns
	[Tier 2] Comprehensibility, Performance, Reliability	
	[Tier 1] Aesthetics, Availability, Efficiency, Familiarity, Readability	

Table 1. A model for the pragmatic quality of Web Applications.

The steps of the construction, summarized in Table 1, are as follows:

1. **Semiotic Levels.** From a semiotics viewpoint, the representation of a resource in a Web Application (that is, a sign) can be viewed on six interrelated levels: physical, empirical, syntactic, semantic, pragmatic, and social. This paper is restricted to the *pragmatic* level, which is responsible for the relation of signs to their interpreters. The interpreters in our case are the stakeholders as given in Section 3.3.
2. **Quality Attributes.** Since pragmatic quality is a multi-dimensional concept, it is decomposed into granular levels that consist of known attributes that can be addressed directly or indirectly. These quality attributes could, for example, manifest themselves as non-functional requirements (NFRs) of a Web Application. For the definitions of these quality attributes, one could refer to the IEEE Standard 1061-1998, the ISO 9241-11:1998 Standard, and the ISO/IEC 9126-1: 2001 Standard.
3. **Means.** Finally, among the possibilities, patterns as means for improving the quality attributes are assigned.

Remark 2. The quality attributes in Table 1 are necessary, but there is no explicit claim of their sufficiency. In this sense, Table 1 follows the "Open World Assumption."

The Quality-Stakeholder Contract

For the sake of this paper, pragmatic quality is viewed as a *contract* between a Web Application and a stakeholder. The relevance of quality attributes in Table 1 varies with respect to stakeholder types. For the sake of simplicity, the discussion of (not necessarily mutually exclusive) stakeholders here is limited to those of the type *end-user* and *engineer*:

- **Pragmatic-Tier 1.** The quality attributes of direct concern to an end-user are aesthetics, availability, familiarity, and readability. The quality attributes of direct concern to an engineer are efficiency and readability.
- **Pragmatic-Tier 2.** The quality attributes of direct concern to an end-user are comprehensibility, performance, and reliability. The quality attribute of direct concern to an engineer is comprehensibility.
- **Pragmatic-Tier 3.** The quality attributes of direct concern to an end-user is usability. Furthermore, accessibility can be viewed as a special case of usability (Mendes & Mosley, 2006). The quality attribute of direct concern to an engineer is maintainability. Moreover, modifiability, portability, and reusability can be viewed as special cases of maintainability (Buschmann et al., 1996).

Pragmatic Quality Attribute	Producer Concern	Consumer Concern
[Q01] Aesthetics		X
[Q02] Availability		X
[Q03] Comprehensibility	X	X
[Q04] Efficiency	X	
[Q05] Familiarity		X
[Q06] Maintainability	X	
[Q07] Performance		X
[Q08] Readability	X	X
[Q09] Reliability		X
[Q10] Usability		X

Table 2. The stakeholder types and corresponding quality attributes of concern.

Table 2 summarizes the mapping between stakeholder types and quality attributes. The identification labels [Q01] to [Q10] have been associated for later reference.

Remark 3. The quality attributes in Table 1 are not "absolute." The significance and associated priority of quality attributes will likely vary across the different domains targeted by Web Applications. For example, the performance and reliability needs of a Web Application providing information on stock market indices will vary from the one providing information on classic movies.

Relationship among Quality Attributes

The quality attributes in Table 1 can depend on each other in a favorable or unfavorable manner. Indeed, the quality attributes in Table 1 are not necessarily mutually exclusive in two different ways: (1) within the same tier, and (2) across tiers.

1. **Intra-Tier Quality Attribute Relationships.** The quality attributes within the same tier not necessarily mutually exclusive. For example, the steps taken towards improving reliability (say, fault tolerance) may lead to redundant source code or data (that can be unfavorable to maintainability) but enable ease-of-use (that can be favorable to usability). Similarly, efforts towards improving efficiency (say, resource utilization) could lead to a detriment of readability. For example, saving screen estate on a user interface of a Web Application by reduction of white space increases (spatial) efficiency but at the expense of readability.
2. **Inter-Tier Quality Attribute Relationships.** The quality attributes in Tier 3 depend on that in Tier 2, which in turn depend on Tier 1. For example, if a user cannot read, he/she cannot comprehend the information in a Web Application, and thereby cannot use it to its full potential. Similarly, for a Web Application to be reliable, it must be available.

Using a simple scheme (Wiegers, 2003), the pairwise relationships between the quality attributes are pointed out in Table 3 where necessary. Moreover, three possible determinations can be made: (1) a "+" sign in a cell indicates that an improvement in the attribute in the corresponding row has a favorable (non- negative) effect on the attribute in the column, (2) a "–" sign in a cell indicates that an improvement in the attribute in that row has, in general, a detrimental (non-positive) effect on the attribute in the column, and (3) an empty cell indicates that an improvement in the attribute in the corresponding row has no (known) bearing on the attribute in the column or that, due to the existence of several exceptions, conclusion in either direction can not be drawn decisively.

	[Q1]	**[Q2]**	**[Q3]**	**[Q4]**	**[Q5]**	**[Q6]**	**[Q7]**	**[Q8]**	**[Q9]**	**[Q10]**
[Q01]			+		+			+		+
[Q02]						–			+	+
[Q03]						+				+
[Q04]						–		–		
[Q05]			+			+		+		+
[Q06]										
[Q07]						–				+
[Q08]						+				+
[Q09]			–				–	+		+
[Q10]										

Table 3. A matrix of pairwise relationships between the pragmatic quality attributes.

Remark 4. From Table 3, it is evident that the rows for [Q06] maintainability and [Q10] usability are empty, and the matrix is (obviously) not symmetric.

Feasibility of Quality Attributes

The expectations of improving the quality attributes of a Web Application must be feasible in order to be practical.

The pragmatic level quality concerns are in general soft goals. That is, the quality attributes in Table 1 cannot (at least mathematically), with respect to stakeholders, be completely satisfied but can only be satisficed. For example, an a priori guarantee that a Web Application will be usable to *all* users at *all* times in *all* situations that the users can find themselves in, is simply not viable. The exception to this could for example be performance-related concerns that could be expressed as hard goals.

The interdependency of the quality attributes as discussed earlier also places inevitable feasibility constraints. From Table 3, the improvement in one attribute may need to be balanced with respect to its impact on the others as the steps taken to address one quality attribute can lead to a compromise of the other. There are well-known techniques such as the Analytical Hierarchy Process (AHP) and Quality Function Deployment (QFD) for carrying out multi-criteria decision making. Further discussion of this aspect is beyond the scope of this paper.

3.6 Acquiring and Selecting Patterns

The characteristics of reflection, realization, and prevention make patterns especially suitable as means for addressing quality-related concerns:

- **Reflection.** A pattern is not invented in isolation. To reach the candidacy of a pattern, an initial submission by an author often needs to go through an arduous process that includes a comprehensive review (via "shepherding" or "writers' workshops") by other experts in the domain.
- **Realization.** A pattern, when appropriately described and documented, provides the *process* of arriving at an abstract but broadly applicable proven *solution* specific to a *problem* in a given *context*, including the reasoning and trade-offs behind it. In other words, the *solution* of a pattern is the most optimal solution to a problem under the given circumstances. This often makes patterns more practical in their applicability compared to, for instance, guidelines (Vanderdonckt, 1999).
- **Prevention.** A pattern, unlike inspections or testing that focus on evaluation, is means of early prevention of quality-related concerns.

The need for selection of patterns manifests itself by *necessity* as the relationship between the set of quality attributes and the set of patterns, in general, is many-to-many. Although there are preliminary results on automation such as an expert system-based decision analysis (McPhail & Deugo, 2001), finding suitable patterns appropriate for a task largely remains a manual process.

Feasibility of Patterns

There are a few challenges (that could be viewed as risks) in the patterns' selection process. They stem from a variety of factors:

- **Availability of Patterns.** For an adoption of a pattern-oriented approach to the development of Web Applications, there need to be analysis and synthesis patterns that can sufficiently "map" the problem and solution space, respectively. However, there is no *a priori* guarantee that for every quality related problem there will be suitable pattern(s) available to solve it.
- **Findability of Patterns.** It is relatively straightforward to detect and find necessary patterns when they are organized in a catalog, system, or language: in these cases the *name*s of patterns and relationships among patterns are made explicit. However, patterns are not always organized as

such, and there are still a few challenges in finding desirable patterns that have been publicly documented. There is currently no de facto meta-index of patterns. In general, patterns currently available are not classified by quality attributes (such as those in Table 1). The number of patterns has also grown significantly in the last decade or so. It has been reported (Buschmann et al., 1996) that the ease of finding pattern(s) appropriate for a task is inversely proportional to the number of documented patterns. The name of a pattern is often a means for finding patterns. However, there are patterns in different collections that have similar or same *name*s but semantically different functionality, or have similar intent or functionality but different *name*s. For example, the COMPOSITE pattern from one collection (Gamma et al., 1995) is similar to the WHOLE-PART pattern from another collection (Buschmann et al., 1996). These can be prohibitive to both navigating and searching for suitable patterns.

- **Cost of Deploying Patterns.** There is inevitable cost in terms of time, effort, and resources of learning and adapting any reusable knowledge, including patterns. For example, there is a learning curve involved in aspects such as understanding the pattern description, checking if and how accurately the *context* of the pattern matches with that of the *problem* of the Web Application development at hand, and the constraints under which the *solution* suggested by the pattern exists. There is also an associated cost of bringing the *solution* to a realization. For example, it has been shown (Rosado, Fernández-Medina, & Piattini, 2006) that the cost of implementing the CHECK POINT pattern (Yoder & Barcalow, 1997) is "high."

3.7 Applying Patterns

There are three main non-mutually exclusive concerns in the application of patterns: an understanding of the patterns, the order in which patterns are applied, and the result upon the composition of patterns.

1. **Understanding Patterns.** For a given pattern, an understanding of the underlying *problem* and the *context* in which it occurs, and understanding the *forces* and consequences of the proposed *solution*, are necessary for applying a pattern effectively. For example, a pattern whose *solution* suggests the use of color will not be applicable in situations when the underling device lacks support for it or if the user is color blind.
2. **Order of Applying Patterns.** The order in which patterns are applied does matter and needs to be taken into consideration. For example, in the design phase, the patterns for high-level design are applied first, followed by the patterns for low-level design. If ignored, it may not be at all possible to apply some patterns (as they conflict) and/or lead to a non-optimal result. This order is pre-determined if all the patterns being selected are from the same pattern system or a pattern language. In such as case, they may also be available as a "map" or more formally as a directed acyclic graph (DAG) to assist navigation. Otherwise, an appropriate pattern description will have *context* and *related patterns* elements that can give an indication of the order to be followed.
3. **Composing Patterns.** Even when a certain order in application of patterns is followed, it is the result upon the composition of patterns that should appear suitable. Whether the result is acceptable could, for example, be revealed during a low- or high-fidelity prototyping. If not, one needs to revert back to the POWEM step 5, and revise the selection.

3.8 Patterns for Pragmatic Quality in the Design of Web Applications

This section addresses the role of patterns elicited from external sources in the design phase of a typical Web Application. The selection of patterns is based on their generality, neutrality with respect to any specific application domain, broad availability, parity to the quality attribute at hand, suitability of the *context* and the *forces* (where available), and the public reputation of the authors. In order to distinguish the patterns from the main text, their *name*s are indicated in uppercase.

Macro-Architecture Design of Web Applications

The macro-architecture design is the place where high-level design decisions, independent of any implementation paradigm or technology, are made. To that regard, some general considerations are in order.

A Web Application will implicitly or explicitly target some domain such as education, commerce, entertainment, government, and so on. (These can obviously be granularized further to arbitrary levels.) The choice of the domain name (such as `.net` or `.org`) does not always or automatically reveal the nature of the domain. There are patterns available for certain common genres like EDUCATIONAL FORUMS (for educational institutions), NONPROFITS AS NETWORKS OF HELP (for non-profit organizations), and SELF-SERVICE GOVERNMENT (for municipal, state, or federal government agencies) (Van Duyne, Landay, & Hong, 2003). The application of such genre-specific patterns can increase user familiarity with the Web Application. Furthermore, the organization owning a Web Application may wish to serve (potential) consumers in diverse cultural and/or geopolitical situations (such as, in different countries and using different natural languages). This could be done using the LOCALE HANDLING pattern (Busse, 2002). However, the trade-off in deploying this pattern is that the maintenance responsibilities increase.

Next, some technical considerations can be emphasized. The macro-architecture design patterns suggested are based on the fact that Web Applications are a class of distributed request-response-type interactive systems. Specifically, the applicable patterns are the CLIENT-SERVER pattern (Schmidt et al., 2000) followed by the APPLICATION SERVER pattern (Manolescu & Kunzle, 2001), which in turn is followed by the MODEL-VIEW-CONTROLLER (MVC) pattern (Buschmann et al., 1996; Fowler et al., 2003).

The CLIENT-SERVER pattern supports modifiability and reusability. For example, a server or resources on the server-side could be modified without impacting the client. Also, a single server can support multiple clients simultaneously, or a client could make simultaneous requests for resources residing on multiple servers. For instance, an Extensible Markup Language (XML) document could be located on one server, while an ECMAScript/JavaScript script on another server, and a Cascading Style Sheet (CSS) or Extensible Stylesheet Language Transformations (XSLT) document on yet another.

The APPLICATION SERVER pattern also supports maintainability: it isolates the Web Application from other aspects on the server-side such that the communication between the application itself and the Web server takes place via the SINGLE POINT OF ACCESS (Yoder & Barcalow, 1997) pattern. This separation allows the Web Application to evolve independently.

The separation of structure of information in a markup document from its presentation is one of the principles of Web Architecture (Jacobs & Walsh, 2004) and is practically realized by the SEPARATE CONTENT FROM PRESENTATION pattern (Weiss, 2003). Adopting this principle and pattern along with an appropriate use of MVC leads to a separation of semantically-different aspects, namely of data, rendering, and processing, into three sets of components: M = {model}, V= {view}, and C = {controller}, where $| M | = 1$, $| V | \geq 1$, $| C | \geq 1$, and $| \cdot |$ denotes the cardinality. This (at least theoretically) minimizes the coupling between these aspects. Thus, modifications to one component are localized and lead to minimal propagation of changes to other components. This improves the modifiability of components of a Web Application. A model is normally not aware of the views and controllers attached to it. This allows the MVC pattern to follow the "single source" approach: the same model in MVC could be used with multiple views and multiple controllers. For example, the same information could be adapted (repurposed or transformed) and delivered to different situations (like user agent environments or consumer needs). This improves the reusability of components of a Web Application.

Figure 2 presents an abstract view of the aforementioned macro-architecture design patterns.

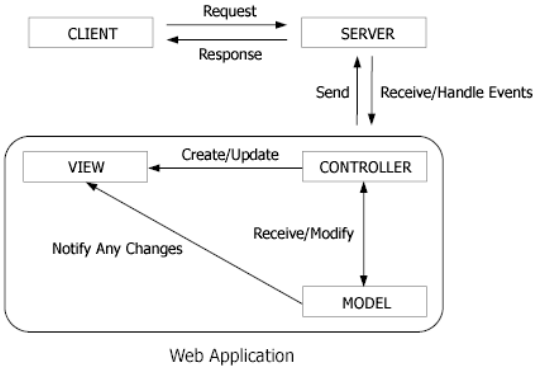

Figure 2. A view of the macro-architecture design patterns in the development of Web Applications.

MVC is an example that illustrates the fact that patterns often do not exist in isolation. Indeed, other patterns may be needed to help solve the problems encountered in the use of MVC in specific situations. A few such cases are considered next.

The actual separation of model, view, and controller in an object-oriented environment is practiced with the help of micro-architecture design patterns (Gamma et al., 1995) such as OBSERVER, COMPOSITE, and STRATEGY.

A Web Application can become increasingly sophisticated with functionalities such as dynamic delivery of resources, personalization, and so on. MVC needs to adapt to these situations, and in doing so, it faces a number of challenges that can be tackled with the use of other special-purpose patterns (Fowler et al., 2003). For example, to invoke different user interface styles in a Web Application, there may be a need to have multiple controllers. As the number of controllers increase, this can lead to redundancy that in turn is prohibitive to maintainability. In such a case, the common logic could be extracted via the APPLICATION CONTROLLER pattern. In response to a request to a Web Application, there may be several tasks that need to be performed and these tasks may have overlapping aspects. The distribution of the behavior of the input controller across multiple objects can lead to redundancy, which in turn is exorbitant to maintainability. The FRONT CONTROLLER pattern suggests a consolidation of all request-handling by channeling requests through a single handler object.

Reliability Design

For addressing reliability (specifically, availability) concerns, the macro-architecture design of server-side components of a Web Application could use a number of available patterns (Manolescu & Kunzle, 2001; Ahluwalia & Jain, 2006). For example, extra measures to support the availability of a Web Application (unrelated to the functionality of the Web Application) could be included by using the INTRODUCE REDUNDANCY pattern. One way to introduce redundancy is to have a cluster of multiple servers such as suggested by the FAIL-OVER THROUGH CLUSTERING pattern, where if one (primary) server fails, the other (secondary) server takes over the responsibility. However, redundancy also increases maintenance responsibilities.

If and when the need arises, a failure message could be relayed directly using the FAILURE NOTIFICATION pattern or indirectly using the HEARTBEAT pattern (where an engineer is informed via periodic broadcasts that a specific Web server is available; the absence of such a message would then imply its unavailability.)

Micro-Architecture Design of Web Applications

The micro-architecture design is the place where low-level design decisions that are to be implemented are cast.

In the following, the focus is only on the design aspects that impact pragmatic quality. As such, the attention is geared more towards client-side rather than server-side concerns.

Interaction Design

Interaction design is an approach to the design of interactive systems that focuses on the human as well as the computer aspects, and is perhaps the most crucial client-side concern of Web Applications.

In the following, the four most critical interaction design aspects of Web Applications, namely information design, navigation design, search design, and presentation design, independent of any specific domain, are considered. These four aspects are not mutually exclusive.

Information Design

The information in Web Applications has increasingly become heterogeneous. It is often the case that the information presented on a single document is aggregated from several sources. For example, consider, from user-perspective, the entry point to a Web Application, namely the "Home Page," which can be realized using the HOME PAGE pattern (Graham, 2003). Then, the "Home Page" of a news organization served from the main source may include a latest news ticker from one server and weather information from another server, the currency and stock market information from a financial Web Service, and periodically changing advertisements from yet another source.

This can be systematically realized by the use of the WHOLE-PART pattern (Buschmann et al., 1996), which enables a hierarchical organization of objects. Since each of these objects can be modified or replaced independently, the WHOLE-PART pattern supports maintainability. Also, since a "part" can correspond to more than one "whole," the WHOLE-PART pattern also supports reusability. However, multiple indirections stemming from client requests and responses for fulfilling them can lead to a loss of performance, particularly when each "part" itself is structured as WHOLE-PART.

Next, the classification of information, which is a conventional approach by humans for understanding information, is considered. The information organization patterns (Van Duyne, Landay, & Hong, 2003), when use appropriately, aid readability, comprehensibility, and usability. For example, the GRID LAYOUT pattern that suggests the organization of information in a single document ("Web Page") into a grid of rows and columns where every atomic information element is made to fit within this grid. The WHAT'S NEW PAGE pattern provides newly added information to a Web Application, and could include the CHRONOLOGICAL ORGANIZATION pattern. A document in a Web Application that provides event proceedings could contain a list of publications and/or their authors based on the ALPHABETICAL ORGANIZATION pattern.

The users of a Web Application can vary in their capabilities and preferences, and may find one view of information to be more usable than another. The MIRRORWORLD pattern (German & Cowan, 2000) provides two or more views of the same information. Specifically, information in these views could be presented (Tidwell, 2006) in TWO-PANEL SELECTOR pattern when there are two different views that are to be presented simultaneously, or CLOSABLE PANELS or CARD STACK patterns when there are several different views to be presented in such as way that only one view is visible at a time in each panel or stack, respectively.

Tables are often used to structure information in two dimensions. However, the presence of many columns (or multiple lines per row) can adversely affect readability, as it becomes increasingly hard to separate the entries visually. The ROW STRIPING pattern (Tidwell, 2006) suggests the use of two similar shades to alternately color the backgrounds of the table rows.

Now, documents in a Web Application may contain images for presenting some information such as the corporate logo or product pictures. The FAST-DOWNLOADING IMAGES pattern (Van Duyne, Landay, & Hong, 2003) suggests creation of images optimized for color and size in an appropriate format, and thus aids accessibility and performance. The REUSABLE IMAGES pattern (Van Duyne, Landay, & Hong, 2003) suggests caching images that appear at multiple places in a Web Application, and thereby aids performance.

To support usability, there should be a provision in the information design to support internal locus of control (thereby provide options to a user) and for users to recover (say, from inadvertent errors). The MULTI-LEVEL UNDO pattern (Tidwell, 2006) provides a way to easily reverse a series of actions performed by the user in a Web Application that can track user session and maintain state.

Navigation Design

Navigation is traversal in information space for some purpose such as casual or targeted browsing for information or complementing a reading sequence (like in electronic books). Both the intra- and the inter-document navigation within the context of Web Application are realized by the use of hypermedia.

The navigation patterns, when use appropriately, aid usability. For example, the BREADCRUMBS pattern (Van Duyne, Landay, & Hong, 2003) could be used to inform the user of location of the resource being viewed relative to the "Home Page" of a Web Application. The Yahoo! Directory was one of the earliest users of the BREADCRUMBS pattern. However, the use of the BREADCRUMBS pattern must be in *context*: it will not scale well for Web Applications with deep hierarchies ($L_1 > L_2 > \ldots > L_n$, for large n, will occupy quite a bit of space on a document) or for information architectures of Web Applications that are not based on a hierarchical design. The CLEAR ENTRY POINTS pattern (Tidwell, 2006) presents only a few entry points into the interface, which can restrict the navigation to a specific category and make it task-oriented.

The FLY-OUT MENU pattern (Marks & Hong, 2006) could be used to present content organized in a compound menu where each menu item itself has a sub-menu that expands only upon interaction and when the user desires. This enables large amount of navigation information to be abstracted from the user and presented only upon interaction by the user, thereby improving both (spatial) efficiency and readability. The FLY-OUT MENU pattern could itself be arranged horizontally or vertically as suggested by the HORIZONTAL NAVIGATION or VERTICAL NAVIGATION patterns (Marks & Hong, 2006), respectively. The choice of the menu items themselves could be inspired by those that are known to be most frequently selected by users (via some form of statistical analysis like checking server log files): the guidance for this is provided by the DIRECT PATH pattern (Casteleyn, Garrigós, & Plessers, 2004).

Any navigation design must take exceptional behavior into consideration to support usability. The SESSION pattern (Weiss, 2003) can help maintain the state of the Web Application in the event of an interruption of navigation flow. The MISSING LINK pattern (German & Cowan, 2000) informs the user that certain hyperlink does not exist and suggests alternatives.

There are navigation design patterns that enable efficient use of space and aid comprehensibility (Tidwell, 2006). For example, the WIZARD pattern leads the user through the interface step by step for carrying out tasks in a prescribed order. It can also be used to implement a context-sensitive help on a given functionality. The RESPONSIVE DISCLOSURE pattern starts with a very minimal interface, and guides a user through a series of steps by showing more of the interface as the user completes each step. These two patterns could, for example, be used for carrying out a registration process. Now, during such a process, the user may have to be presented with several options (such as multiple mailing addresses or credit cards). However, the designer is faced with the problem that the information on all of them would not fit on a single panel and a user does not have to see the details of

all the options simultaneously. In such a case, the CARD STACK pattern could be used where one option is visible by default and the others are hidden.

Search Design

The goal of searching is finding information. Searching is not native to Web Applications, but has become ever more challenging as the amount of information to be searched through increases.

The searching patterns, when used appropriately, aid comprehensibility, efficiency, performance, and usability. The use of a SIMPLE SEARCH INTERFACE pattern (Lyardet, Rossi, & Schwabe, 1999) or an INTUITIVE SEARCH INTERFACE pattern (Wellhausen, 2005) that suppresses technical details can contribute towards comprehensibility. Specifically, this can be realized by the STRAIGHTFORWARD SEARCH FORMS pattern (Van Duyne, Landay, & Hong, 2003) that requires minimal technical background on part of the user. In case when there are clearly defined user types such as novice and expert, the SIMPLE AND EXPERT SEARCH DIALOG pattern (Wellhausen, 2005) suggests a simple search dialog that contains only the most basic features, while additionally provide an expert search dialog that supports all features available. This user-sensitivity of searching allows not sacrificing the effectiveness of searching for the sake of comprehensibility.

The use of SEARCH BAR pattern (Wellhausen, 2005) allows adding a small search bar to the Web Application that provides space for entering a value for one search parameter, and thereby contributes to efficiency.

The following patterns can all improve the effectiveness of the searching activity: the use of SELECTABLE SEARCH SPACE pattern (Lyardet, Rossi, & Schwabe, 1999) or ONLY HERE! pattern (Schmettow, 2006) that can restrict the search to a specific category; the REFINE RESULTS pattern (Wellhausen, 2005) that enables iterative refinement of past search results, including reverting back to modify earlier decisions; the SELECTABLE KEYWORDS pattern (Lyardet, Rossi, & Schwabe, 1999) or EXPLOIT SOCIAL INFORMATION pattern (Schmettow, 2006) that can suggest keywords for improving subsequent search results based on the past user experience or behavior; the ORGANIZED SEARCH RESULTS pattern (Van Duyne, Landay, & Hong, 2003) or RANK BY AUTHORITY pattern (Schmettow, 2006) that organize, rank, and present a summary of the most "relevant" search results; and the SAVE SEARCHES pattern (Wellhausen, 2005) that allows the users save, load, manage, and re-execute their search requests. The Google search engine and CiteSeer publication server implement some of these search patterns.

When search results are too numerous, they could cognitively overload a user. In such a case, searching and navigating can *complement* each other. Using the PAGING pattern (Marks & Hong, 2006), the search results could be split into multiple sections that could be navigated sequentially.

Any search design must take exceptional behavior into consideration to support usability. A keyword-based search necessitates input from the user, and is therefore prone to errors. The use of AUTO COMPLETE pattern (Yahoo! Design Pattern Library) can circumvent this problem. The CORRECT ME IF YOU CAN pattern (Schmettow, 2006) suggests informing the user about possible spelling errors and suggesting a correction.

Presentation Design

It has been shown in surveys (Tractinsky et al., 2006) that users value the aesthetics or the "look-and-feel" of a Web Application. The elements of presentation apply to all aspects of design discussed previously.

For the purpose of aesthetics, a variety of patterns could be used. For example, using the DEEP BACKGROUND pattern (Tidwell, 2006), an image or gradient could be placed in the background of a document in such a way that it visually recedes behind the foreground elements. As another example,

often in traditional shops, certain products that are seasonal, for special occasions, on sale or otherwise recommended, are often displayed in a special manner. This selected list of products could be highlighted via the FEATURED PRODUCTS pattern (Van Duyne, Landay, & Hong, 2003).

It is known that colors can have a positive impact both cognitively and aesthetically if used appropriately, in particular taking into the considerations of the color blind (Rigden, 1999). Often, organizations have their favorite color(s) that are prominently reflected in their logo, and placed on their business cards and office supplies. Using patterns like FEW HUES, MANY VALUES or COLOR-CODED SECTIONS (Tidwell, 2006) a Web Application could be given a unique "identity." It is known that the user interface must reflect the state of the software. So, for example, the option in a navigation bar selected by a user could be reflected uniquely upon visitation by the user. As shown below, C_2 is the color of the option that corresponds to the resource that the user is currently viewing and is different from the others:

$$[\ C_1\ ] \quad [\ C_2 \neq C_1\ ] \quad [\ C_1\ ] \quad [\ \ldots\ ] \quad [\ C_1\ ]$$

There are of course other presentation issues that would impact usability such as the choice of fonts and their properties, layout and positioning of user interface components, use of white space, and so on. These could also be addressed via an appropriate use of patterns.

Example

Figure 3 gives an abstract illustration of the solutions of some of the interaction design patterns mentioned previously. As a concrete example, the "News Page" at IBM has used these interaction design patterns.

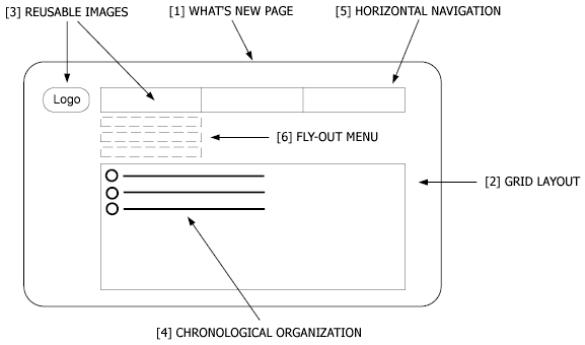

Figure 3. A confluence of solutions of interaction design patterns in the development of Web Applications.

The numbers indicate the order of application and the FLY-OUT MENU items are shown in dashed lines to exemplify non-permanence. Figure 3 does not show all possible behaviors of solutions but rather presents only one possible view.

The Appendix summarizes the quality attributes and patterns mentioned in this paper. The list of patterns is by no means complete and is subject to evolution. This is because the use of each pattern places a Web Application in a new state (*context*) that can lead to new *problem*(s). To solve these problem(s) we may have to apply new set of pattern(s), assuming that such pattern(s) are indeed available.

Remark 5. The above discussion reiterates the fact that there is no unique classification scheme for patterns. Indeed, a pattern proposed in one context can be applicable to other contexts, irrespective of the original intention of the author, including placement in some known classification scheme.

Remark 6. Many of the patterns mentioned in this section can be implemented using an assortment of technologies. The use of technologies needs to be conservative so as to be reachable to a variety of user agents. Indeed, from previous discussion, two cases are relevant. The MVC pattern can be realized using the Java implementation of frameworks like Asynchronous JavaScript and XML (AJAX) and Rails where XML and XSLT can be used for storage and repurpose of text-based information, and Extensible HyperText Markup Language (XHTML) along with CSS were used for presentation of information. In case of the FLY-OUT MENU pattern, the menu can be populated with menu items using ECMAScript (JavaScript), the expanding and collapsing of menu can be realized using ECMAScript, highlighting of a menu item being pointed to can be realized using CSS, and the navigation part can be realized via support for hypertext in XHTML.

3.9 Evaluating Effectiveness

There are metrics for theoretically evaluating the effectiveness of applying guidelines for the purpose of improving certain quality attributes (like accessibility or usability) of Web Applications (Ivory, 2001). However, to the author's knowledge there are no such corresponding metrics for patterns. Furthermore, existing collection of metrics may prove insufficient to evaluate patterns (Masuda, Sakamoto, & Ushijima, 1999). Therefore, there is much work to be done in the area of measuring designs that make use of patterns.

For an empirical evaluation, a "lightweight" manifestation of POWEM was followed at Concordia University in a project for managing patterns titled *Patterns for Web, Web for Patterns (P4W4P)*. Specifically, P4W4P was given in multiple undergraduate- and graduate-level courses, each spanning about four months, and was carried out by multiple teams of size 10.

The complementary but interrelated goals for the development of the P4W4P Web Application were:

- **Goal 1.** P4W4P would make publicly available, from both print and/or electronic sources, a cohesive and carefully selected collection of patterns for designing Web Applications. This collection, namely Patterns for Web Applications (PWA), would acknowledge original sources, be navigable, searchable, and evolvable. Users of P4W4P would have context-dependent help and opportunities for feedback available to them at all times. P4W4P would also provide maintenance facilities such as modifying the information of an existing pattern and adding/deleting a pattern. The access to these administrative services would be restricted.
- **Goal 2.** P4W4P would apply PWA to itself.

Thus, the purpose of P4W4P was that it would help PWA, and in turn, PWA would help P4W4P. Figure 4 illustrates the symbiosis resulting from the interplay of these goals.

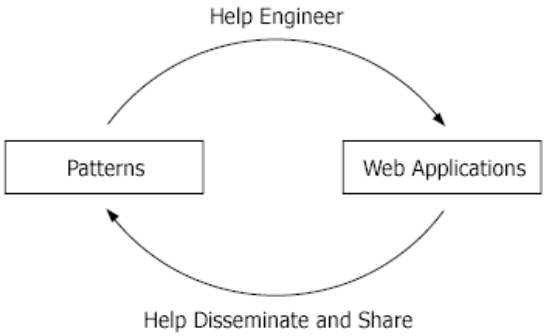

Figure 4. The synergy between Patterns and Web Applications.

An initial, albeit informal and elementary, feasibility study for P4W4P was performed. The knowledge and skill set of students were key determinants in forming groups within a single team. To manage costs, the activities of modeling, documentation, and implementation were all carried out in an Open Source Software (OSS) environment according to available guidance (Kamthan, 2007).

The development of P4W4P was both iterative and incremental with periodic deliverables. Based on instructor's formative assessment (feedback and suggestions for improvement, including questions), some of these deliverables were revisited and revised for compliance with future submissions. In particular, P4W4P followed the UPEDU with minor variations such as more emphasis on early modeling in the Unified Modeling Language (UML), NFRs, and on interaction design, and less on lengthy documentation. Many of the patterns mentioned in Section 3.8 were included in PWA and were used during the development of P4W4P.

In general, the students found the project experience rewarding. In the following, the (admittedly mixed) reactions and results by students using POWEM in P4W4P for achieving the aforementioned goals are outlined. They are based on the reports submitted by the teams at the conclusion of the project and have been limited only to the context of this paper.

- **Quality Assurance.** The students appeared to be more interested in quality attributes of concern to consumers than to producers. For example, there was much more emphasis on patterns for usability than for maintainability. This could be explained in two ways: most students did not have a formal background in maintenance and the students knew at the outset that once the course ended they are unlikely to maintain P4W4P in the future.
- **Acquisition, Selection, and Application of Patterns.** The students initially found the notion of a pattern rather abstract and hard to understand, particularly when it was introduced in a domain-independent fashion, but intriguing once exposed to real-world examples. Furthermore, they found the selection from a large number of (at times, seemingly similar) patterns on certain topics to be overwhelming. On the other hand, this made them appreciate the necessity of a well-designed navigation and precise searching in their own effort towards P4W4P. The students also found that, in some cases, the descriptions of certain patterns were inadequate (such as absence of certain mandatory elements) in order for them to make an informed decision about their selection. In retrospect, this also helped them "reengineer" certain key aspects such as the *context* and *forces* of a pattern.
- **Quality Evaluation.** The task of formally inspecting the design artifacts, including verifying the presence and use of patterns, was carried out in rotation (for example, Team A reviews the design of Team B, Team B reviews the design of Team C, and so on). The students found the inspections carried out to be useful for improving their own designs and for instilling a competitive yet professional spirit.

3.10 POWEM in Perspective

It is evident that POWEM is not "absolute." The discussion on the feasibility of each of the steps in POWEM places constraints and determines the scope of POWEM.

POWEM also does not come with an a priori guarantee of an improvement in quality. Specifically, for a given quality attribute, the non-existence of a suitable pattern; the inability to find a pattern within a given time constraint to make optimal use of it; not being able to use a pattern due to organizational policies; or misinterpretation of the *context* of a pattern leading to an undesirable result, are all potential limitations of a pattern-oriented approach to Web Applications such as POWEM.

4 Some Directions for Future Research

This section briefly discusses the different directions in which the work presented in this paper can be extended and may be worth pursuing.

4.1 Extensions of POWEM

Table 1 could benefit from formalization of the relationships between quality attributes and patterns. Another possible extension of Table 1 is increasing the granularity of the quality attributes at the level Pragmatic-Tier 1, and thereby adding another level (say, Tier 0) underneath it. In that case, for example, fault tolerance and recoverability could be candidate quality attributes that belong to the level Pragmatic-Tier 0. Some patterns for fault tolerance and recoverability have been suggested (Ahluwalia & Jain, 2006). Similarly, consistency (on which familiarity depends) could be a candidate for the level Pragmatic-Tier 0.

4.2 Integration of Patterns in Web Engineering Education

There appears to be a potential for deployment of patterns Web Engineering education, both inside and outside the classroom. For example, during lectures, teachers could demonstrate the use of patterns in the design of Web Applications. The students could be asked to compare and contrast between a given set of competing patterns for a specific problem. They could also be, as described in Section 3.9, asked to make use of a pre-selected collection of patterns in a course project like P4W4P.

However, a systematic integration of patterns in Web Engineering education will need to be in line with the adopted teaching strategies and learning theories. There is currently no standard effort towards defining the knowledge areas and the basic body of knowledge in Web Engineering education where patterns could be given a place. There are pedagogical models for Web Engineering education (Hadjerrouit, 2005) but the support for patterns in them is also lacking.

4.3 Adoption of Patterns in Standards for Web Engineering

It is known that based on consensus, standards provide a common ground and, when applied well, can contribute towards improvement of productivity and communicability across project teams. There exist initiatives for standardizing the development of Web Applications such as the IEEE Standard 2001-2002 that do signify the role of quality and means for evaluation. However, the adoption of patterns in such standards efforts will be crucial for a widespread acceptance and use of patterns in Web Engineering.

5 Conclusion

A disciplined approach towards the development of Web Applications is necessary for both their longevity and for acceptance by their stakeholders. A systematic and lasting view towards pragmatic quality, and viable means for addressing it, is integral to this. Patterns provide one practical means for addressing the quality of Web Applications, if they are located, adopted and applied with care, within a systematic development process, and by taking feasibility issues into consideration. This paper presents a step in that direction.

Still, for patterns to continue being useful, they must be adequately described, be openly available, be readily findable, and evolve with the Web and the needs of Web Applications. In other words, pattern authors and Web Engineers *share* the responsibility in the deployment of patterns in future Web Applications. P4W4P only reinforces the significance of this commitment.

In conclusion, by investing in a pattern-oriented and stakeholder-quality-centric approach such as POWEM, the producers of Web Applications can further the spirit of Web Engineering by bringing more structure and predictability in the development process. A transition towards a preventative

approach to the quality of Web Applications equipped with patterns is not free but the benefits can outweigh the costs in the long-term.

Acknowledgements

The author would like to thank the reviewers for their feedback and suggestions for improvement.

References

[1] Ahluwalia, K. S., & Jain, A. (2006). High Availability Design Patterns. The 13th Conference on Pattern Languages of Programs (PLoP 2006), Portland, USA, October 21-23, 2006.

[2] Buschmann, F., Henney, K., & Schmidt, D. C. (2007). Pattern-Oriented Software Architecture, Volume 5: On Patterns and Pattern Languages. John Wiley and Sons.

[3] Beck, K., & Andres, C. (2005). Extreme Programming Explained: Embrace Change (Second Edition). Addison-Wesley.

[4] Boreham, N. C. (1987). Causal Attribution by Sensing and Intuitive Types during Diagnostic Problem Solving. Instructional Science, 16, 123-136.

[5] Brajnik, G. (2001). Towards Valid Quality Models for Web Sites. The 7th Conference on Human Factors and the Web (HFWeb 2001), Madison, USA, June 4-6, 2001.

[6] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). Pattern Oriented Software Architecture, Volume 1: A System of Patterns. John Wiley and Sons.

[7] Busse, D. (2002). Usable Web Design Patterns for World-Ready E-Commerce Sites. CHI 2002 Workshop on Patterns in Practice: A Workshop for UI Designers, Minneapolis, USA, April 21, 2002.

[8] Casteleyn, S., Garrigós, I., & Plessers, P. (2004). Pattern Definition to Refine Navigation Structure in Hypermedia/Web Applications. The IADIS International Conference WWW/Internet 2004 (ICWI 2004), Madrid, Spain, October 6-9, 2004.

[9] Dromey, R. G. (2003). Software Quality - Prevention Versus Cure? Software Quality Journal, 11(3), 197-210.

[10] Fowler, M., Rice, D., Foemmel, M., Hieatt, E., Mee, R., & Stafford, R. (2003). Patterns of Enterprise Application Architecture. Addison-Wesley.

[11] Fraternali, P., Matera, M., & Maurino, A. (2002). WQA: An XSL Framework for Analyzing the Quality of Web Applications. The 2nd International Workshop on Web-Oriented Software Technology (IWWOST 2002), Malaga, Spain, June 10-11, 2002.

[12] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

[13] German, D. M., & Cowan, D. D. (2000). Towards a Unified Catalog of Hypermedia Design Patterns. The 33rd Hawaii International Conference on System Sciences (HICSS 2000), Maui, USA, January 4-7, 2000.

[14] Gilb, T. (1988). Principles of Software Engineering Management. Addison-Wesley.

[15] Graham, I. (2003). A Pattern Language for Web Usability. Addison-Wesley.

[16] Hadjerrouit, S. (2005). Designing a Pedagogical Model for Web Engineering Education: An Evolutionary Perspective. Journal of Information Technology Education, 4, 115-140.

[17] Ivory, M. Y. (2001). An Empirical Foundation for Automated Web Interface Evaluation. PhD Thesis, University of California, Berkeley, USA.

[18] Jacobson, I., Booch, G., & Rumbaugh, J. (1999). The Unified Software Development Process. Addison-Wesley.

[19] Jacobson, I., Christerson, M., Jonsson, P., & Övergaard, G. (1992). Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley.

[20] Kamthan, P. (2007). On the Prospects and Concerns of Integrating Open Source Software Environment in Software Engineering Education. Journal of Information Technology Education, 6, 45-64.

[21] Kappel, G., Pröll, B., Reich, S., & Retschitzegger, W. (2006). Web Engineering. John Wiley and Sons.

[22] Keirsey, D. (1998). Please Understand Me II. Prometheus Nemesis Book Company.

[23] Kotonya, G., & Sommerville, I. (1998). Requirements Engineering: Processes and Techniques. John Wiley and Sons.

[24] Kumar, K., & Welke, R. J. (1992). Methodology Engineering: A Proposal for Situation-Specific Methodology Construction. In: Challenges and Strategies for Research in Systems Development. W. W. Cotterman & J. A. Senn (Eds.). John Wiley and Sons, 257-269.

[25] Lindland, O. I., Sindre, G., & Sølvberg, A. (1994). Understanding Quality in Conceptual Modeling. IEEE Software, 11(2), 42-49.

[26] Lyardet, F., & Rossi, G. (1998). Patterns for Designing Navigable Information Spaces. The 5th Conference on Pattern Languages of Programs (PLoP 1998), Monticello, USA, August 11-14, 1998.

[27] Lyardet, F., & Rossi, G. (2001). Web Usability Patterns. The 6th European Conference on Pattern Languages of Programs (EuroPLoP 2001), Irsee, Germany, July 4-8, 2001.

[28] Lyardet, F., Rossi, G., & Schwabe, D. (1999). Patterns for Adding Search Capabilities to Web Information Systems. The 4th European Conference on Pattern Languages of Programming and Computing (EuroPLoP 1999), Irsee, Germany, July 8-10, 1999.

[29] Manolescu, D., & Kunzle, A. (2001). Several Patterns for eBusiness Applications. The 8th Conference on Pattern Languages of Programs (PLoP 2001), Monticello, USA. September 11-15, 2001.

[30] Marks, M., & Hong, D. (2006). Web Design Patterns Collection Technical Design. Center for Document Engineering Technical Report CDE2006-TR09. University of California, Berkeley, USA.

[31] Masuda, G., Sakamoto, N., & Ushijima, K. (1999). Evaluation and Analysis of Applying Design Patterns. International Workshop on the Principles of Software Evolution (IWPSE 1999), Fukuoka, Japan, July 16-17, 1999.

[32] McPhail, J. C., & Deugo, D. (2001). Deciding on a Pattern. The Fourteenth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE 2001), Budapest, Hungary, June 4-7, 2001.

[33] Mendes, E., & Mosley, N. (2006). Web Engineering. Springer-Verlag.

[34] Mich, L., Franch, M., & Gaio, L. (2003). Evaluating and Designing Web Site Quality. IEEE Multimedia, 10(1), 34-43.

[35] Montero, F., López-Jaquero, V., & Molina, J. P. (2003). Improving e-Shops Environments by Using Usability Patterns. The 2nd Workshop on Software and Usability Cross-Pollination, Zürich, Switzerland, September 1-2, 2003.

[36] Offutt, J. (2002). Quality Attributes of Web Software Applications. IEEE Software, 19(2), 25-32.

[37] Pertet, S. M., & Narasimhan, P. (2005). Causes of Failure in Web Applications. PDL Technical Report PDL-CMU-05-109. Carnegie Mellon University, Pittsburgh, USA.

[38] Perzel, K., & Kane, D. (1999). Usability Patterns for Applications on the World Wide Web. The 6th Conference on Pattern Languages of Programs (PLoP 1999), Monticello, USA, August 15-18, 1999.

[39] Paulk, M. C., Weber, C. V., Curtis, B., & Chrissis, M. B. (1995). The Capability Maturity Model: Guidelines for Improving the Software Process. Addison-Wesley.

[40] Rosado, D. G., Fernández-Medina, E., & Piattini, M. (2006). Comparison of Security Patterns. The International Journal of Computer Science and Information Security, 6(2B), 139-146.

[41] Rossi, G., & Koch, N. (2002). Patterns for Adaptive Web Applications. The 7th European Conference on Pattern Languages of Programs (EuroPLoP 2002), Irsee, Germany, July 3-7, 2002.

[42] Rossi, G., Schwabe, D., & Lyardet, F. (1999). Improving Web Information Systems with Navigational Patterns. The 8th International World Wide Web Conference (WWW8), Toronto, Canada, May 11-14, 1999.

[43] Schmettow, M. (2006). User Interaction Design Patterns for Information Retrieval Systems. The 11th European Conference on Pattern Languages of Programs (EuroPLoP 2006), Irsee, Germany, July 5-9, 2006.

[44] Schmidt, D. C., Stal, M., Rohnert, H., & Buschmann, F. (2000). Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects. John Wiley and Sons.

[45] Simon, H. (1996). The Sciences of the Artificial (Third Edition). The MIT Press.

[46] Stamper, R. (1992). Signs, Organizations, Norms and Information Systems. The 3rd Australian Conference on Information Systems, Wollongong, Australia, October 5-8, 1992.

[47] May, D., & Taylor, P. (2003). Knowledge Management with Patterns. Communications of the ACM, 46 (7), 94-99.

[48] Tidwell, J. (2006). Designing Interfaces: Patterns for Effective Interaction Design. O'Reilly Media.

[49] Tractinsky, N., Cokhavi, A., Kirschenbaum, M., & Sharfi, T. (2006). Evaluating the Consistency of Immediate Aesthetic Perceptions of Web Pages. International Journal of Human-Computer Studies, 64, 1071-1083.

[50] Vanderdonckt, J. (1999). Development Milestones towards a Tool for Working with Guidelines, Interacting with Computers, 12(2), 81-118.

[51] Van Duyne, D. K., Landay, J., & Hong, J. I. (2003). The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience. Addison-Wesley.

[52] Van Solingen, R., & Berghout, E. (1999). The Goal/Question/Metric Method: A Practical Method for Quality Improvement of Software Development. McGraw-Hill.

[53] Van Welie, M., & Van Der Veer, C. (2000). A Structure for Usability Based Patterns. CHI 2000 Workshop on Pattern Languages for Interaction Design: Building Momentum. The Hague, The Netherlands. April 2-3, 2000.

[54] Weiss, M. (2003). Patterns for Web Applications. The 10th Conference on Pattern Languages of Programs (PLoP 2003), Urbana, USA, September 8-12, 2003.

[55] Wellhausen, T. (2005). User Interfaces for Searching - A Pattern Language. The 10th European Conference on Pattern Languages of Programs (EuroPloP 2005), Irsee, Germany, July 6-10, 2005.

[56] Wentzlaff, I., & Specker, M. (2006). Pattern Based Development of User Friendly Web Applications. Workshop on Model-Driven Web Engineering (MDWE 2006), Palo Alto, USA, July 10, 2006.

[57] Wong, B. (2006). Different Views of Software Quality. In: Measuring Information Systems Delivery Quality. E. Duggan & J. Reichgelt (Eds.). Idea Group, 55-88.

[58] Yoder, J., & Barcalow, J. (1997). Architectural Patterns for Enabling Application Security. The 4th Conference on Pattern Languages of Programs (PLoP 1997), Monticello, USA, September 3-5, 1997.

Appendix

Table 4 summarizes (in a lexicographical order) the patterns mentioned in this paper within the context of pragmatic quality of Web Applications. A symbol of (+) associated with a pattern *name* reflects a positive impact on the corresponding quality attribute, whereas a (−) reflects a negative impact.

Pragmatic Quality Attribute	Patterns
Aesthetics	COLOR-CODED SECTIONS (+) DEEP BACKGROUND (+) FEATURED PRODUCTS (+) FEW HUES, MANY VALUES (+)
Availability	FAIL-OVER THROUGH CLUSTERING (+) FAILURE NOTIFICATION (+) HEARTBEAT (+) INTRODUCE REDUNDANCY (+)

Comprehensibility	ALPHABETICAL ORGANIZATION (+) CHRONOLOGICAL ORGANIZATION (+) GRID LAYOUT (+) INTUITIVE SEARCH INTERFACE (+) PAGING (+) RESPONSIVE DISCLOSURE (+) SIMPLE AND EXPERT SEARCH DIALOG (+) SIMPLE SEARCH INTERFACE (+) STRAIGHTFORWARD SEARCH FORMS (+) WIZARD (+)
Efficiency	CARD STACK (+) FLY-OUT MENU (+) ORGANIZED SEARCH RESULTS (+) SEARCH BAR (+)
Familiarity	EDUCATIONAL FORUMS (+) NONPROFITS AS NETWORKS OF HELP (+) SELF-SERVICE GOVERNMENT (+)
Maintainability	APPLICATION CONTROLLER (+) APPLICATION SERVER (+) CLIENT-SERVER (+) FRONT CONTROLLER (+) INTRODUCE REDUNDANCY (−) LOCALE HANDLING (−) MODEL-VIEW-CONTROLLER (+) SEPARATE CONTENT FROM PRESENTATION (+) WHOLE-PART (+)
Performance	ACCOUNT SETUP (+) EXPLOIT SOCIAL INFORMATION (+) FAST-DOWNLOADING IMAGES (+) ONLY HERE! (+) RANK BY AUTHORITY (+) REFINE RESULTS (+) SAVE SEARCHES (+) SELECTABLE KEYWORDS (+) SELECTABLE SEARCH SPACE (+) SIMPLE AND EXPERT SEARCH DIALOG (+) WHOLE-PART (−)
Readability	FLY-OUT MENU (+) GRID LAYOUT (+) HORIZONTAL NAVIGATION (+) ROW STRIPING (+) VERTICAL NAVIGATION (+)
Reliability	FAILURE NOTIFICATION (+) INTRODUCE REDUNDANCY (+)
Usability	AUTO COMPLETE (+) BREADCRUMBS (+) CARD STACK (+) CORRECT ME IF YOU CAN (+) CLEAR ENTRY POINTS (+) CLOSABLE PANELS (+) DIRECT PATH (+) FAST-DOWNLOADING IMAGES (+) HOME PAGE (+) LOCALE HANDLING (+) MIRRORWORLD (+) MISSING LINK (+) MULTI-LEVEL UNDO (+) SESSION (+) TWO-PANEL SELECTOR (+) WHAT'S NEW PAGE (+)

Table 4. Pragmatic quality attributes of a Web Application and corresponding patterns along with their ratings.