Patterns 2.0: a Service for Searching Patterns*

Aliaksandr Birukou†
DISI - University of Trento, Italy

Michael Weiss
SCE - Carleton University, Canada

January 22, 2010

Abstract

With ever-increasing number of patterns in the literature and online repositories, it can be hard for non-experts to know about new patterns and select patterns appropriate to their needs. We argue that a systematic way for searching patterns is required and we present the Patterns 2.0 service, a composite software service for facilitating pattern search and selection. The service combines several pattern-related services with a recommendation service that allows users to share their experiences in using patterns. The contributions of the paper are: the overview of existing services related to the problem of pattern selection, the definition of Patterns 2.0 service, and description of its possible uses.

1 Introduction

Given the steadily growing number of patterns in the literature and online repositories, it can be hard for non-experts to select patterns appropriate to their needs, or even to be aware of the patterns that exist. In this paper, we present an overview of existing software services related to pattern selection and propose a composite software service that facilitates pattern selection. The service can combine existing pattern retrieval services with a recommendation service that allows users to share their experiences in using patterns. We also provide different usage scenarios of that composite service.

Almost fifteen years ago, the GoF stated the problem of selecting patterns: "With more than 20 design patterns in the catalog to choose from, it might be hard to find the one that addresses a particular design problem, especially if the catalog is new and unfamiliar to you" [8]. As time passed, patterns have become an integral part of many development approaches. However, the problem of selecting patterns still exists. If anything, it has become more critical, as the number of documented patterns has continually increased: for instance, Rising's *Pattern Almanac* [17]

†This research is partly supported by the EU FP7 projects COMPAS and LiquidPub

lists more than 1200 patterns. In the past nine years since the publication of the almanac, many new patterns and books on patterns have been published. The domains containing more than ten patterns, the problem of choosing the appropriate pattern is particularly hard to solve for inexperienced programmers [18]:

> Only experienced software engineers who have a deep knowledge of patterns can use them effectively. These developers can recognize generic situations where a pattern can be applied. Inexperienced programmers, even if they have read the pattern books, will always find it hard to decide whether they can reuse a pattern or need to develop a special-purpose solution.

This quote also suggests that experienced software engineers rely on their knowledge to select patterns to apply in a given context. Over time, they build up a good understanding of which patterns apply to their domain. However, they also tend to be less aware of more recently documented patterns. (This becomes very clear when we consider that for many developers the notion of patterns still stops at the GoF book.) Developers with less experience may also ask for advice from friends or colleagues. However, such interactions are highly personalized and rarely documented, that is, this knowledge remains tacit. May [14] observes that patterns have made design knowledge explicit, the process of applying patterns has become itself new tacit knowledge. Several tools for assisting in the process of pattern selection have been developed to make the knowledge underlying the application of patterns explicit.

Although the problem of pattern selection can be considered a particular instance of the general problem of retrieval of relevant information from large document collections [5], it requires specialized tools for a number of reasons: (i) patterns are structured documents where different parts express very different types of information; (ii) they are often linked to each other in a pattern language; and (iii) design patterns accumulate the experience of developers in dealing with design problems. Therefore, besides search engines for patterns such as PatternSeer[1], tools for managing pattern catalogs (see Deng [4] for an overview) and wikis such as PatternForge[2] and Planet[3], existing approaches for supporting pattern selection include case tools [9], expert systems [13], recommendation systems [3], and formal frameworks that help reuse knowledge about patterns (see Weiss [19] for an overview of several such systems).

However, existing approaches that support pattern users in the selection of patterns have several shortcomings: (i) they usually require additional effort during the authoring and selection process (e.g. authors need specify metadata about their patterns); (ii) pattern repositories require effort in maintaining and updating information; (iii) they often targeted at developers, helping them to select architectural or design patterns, while there are also patterns on organizing conferences or meetings, computer-mediated interaction patterns, which are used by non-developers[4]; (iv) they rarely support collaboration and personalized recommendations.

[1] http://doc-it.fe.up.pt/aaguiar/space/Projects/ PatternSeer

[2] http://www.patternforge.net/wiki

[3] http:// patternlanguagenetwork.org

[4] Therefore in the following we use a more general term “pattern user”

Thus, as May [14] says, much of the information how patterns are selected by users remains tacit despite the existence of these tools.

In this paper we present an overview of existing services for pattern search and selection and propose the Patterns 2.0 [5] service, a composite service for facilitating pattern selection. The service combines existing pattern retrieval, tagging, and recommendation services. The core contribution is improving pattern search and providing assistance for pattern selection by combining the services and integrating the recommendation service for tracking pattern usage history. The latter provides support for social factors (tacit knowledge about how patterns are used within an organization), collaboration (potential for linking users) and personalization (who prefers which patterns or domains).

The proposed service address the outlined shortcomings in existing solutions for pattern selection in the following ways: (i) it improves searching by using tagging, usage history; (ii) using community-generated content allows for minimizing the effort in maintaining and updating information in pattern repositories; (iii) the service is orthogonal to the format and domain of patterns, and can be used by different communities either collaboratively (sharing usage data between different communities), or in isolation (each community consumes recommendations based on its own usage data).

The primary audience for this paper are developers of pattern repositories and pattern retrieval systems, as well as researchers on the application of patterns.

The paper has the following structure: in Section 2 we review existing approaches for pattern search and selection. In Section 3 we describe the proposed service and requirements on the services it invokes, while in Section 4 we discuss limitations and possible extensions of our approach. We conclude the paper in Section 5.

2 Approaches to pattern selection

Recently, there have been several efforts in making patterns available in online pattern repositories, where they can be browsed and searched by various criteria. An early example was the Pattern Almanac [17], which is also available in electronic form[6]. More recent examples are the PatternShare[7] site hosted by Microsoft between 2006-2007, Yahoo Design Pattern Library[8], Sun collection of J2EE patterns[9], and computer-mediated interaction patterns[10]. In this section, we review existing approaches for selecting patterns stored in such repositories.

In order to store patterns in a repository, a structured pattern representation must be adopted. There have been several proposals for structural pattern representation, most notably the Pattern Language Markup Language (PLML) [6] and Entity Meta-Specification Language (EML) [20].

[5] 2.0 in the name is from Web 2.0, because the service uses tagging and other community-generated content

[6] `www.smallmemory.com/almanac`

[7] `patternshare.org`

[8] `http://developer.yahoo.com/ypatterns/`

[9] `http://java.sun.com/blueprints/patterns/`

[10] `http://www.cmi-patterns.org/`

Existing online repositories rarely contain personalized features, although they can provide customizable pattern properties for enhancing search [10]. To the best of our knowledge, most of them remain oblivious to the advent of Web 2.0 and list content defined by the repository creator and provide no tagging, bookmarking and other social features. The sad thing about this is no matter how heavily the repository is used for searching patterns, it does not change and improve over time, if not maintained. However, several wiki-based repositories such as PatternForge were created recently trying to overcome such shortcomings and to use the power of the community in order to enrich repositories with tags, links and other user-generated content.

There are several search engines for patterns. PatternSeer is an ongoing project that aims at delivering a system that crawls and indexes pattern descriptions on the Internet and makes them accessible to users via keyword-based search. The problem with current solutions is their limited coverage of patterns. This reminds one of the problems early Internet had – just eleven years ago it was better to use several search engines to get more different results for a query.

Several approaches exploit past user experience in order to suggest suitable patterns. The ReBuilder [9] framework adopts a case-based reasoning approach, where cases represent situations in which a pattern was applied in the past to a software design. ReBuilder supports the retrieval and adaptation of patterns. Cases are described in terms of class diagrams. Cases are retrieved based on a combination of structural similarity between the current design and a pattern, as well as the semantic distance between class names and role names in the pattern.

The authors developed a recommendation system for pattern selection [3] which is complementary to systems like ReBuilder: in this system, patterns are selected on the basis of previous actions by other users. Also, while the use of the relations in a class diagram provides additional information about the desired pattern, such diagrams are not always available. However, since our system uses textual descriptions, and does not require an object model, it has a wider range of potential applications. However, it probably cannot compete with ReBuilder in domains where class diagrams are available. Finally, our system implements a collaborative approach to pattern selection, facilitating experience sharing among users.

Several approaches propose adding formal semantics to pattern descriptions (see Weiss [19] for an overview). As most patterns are organized in pattern languages, some approaches target the selection of pattern(s) from such languages, thus handling relations between patterns, not only individual pattern descriptions. Zdun [21] proposes an approach for pattern selection based on desired quality attributes and pattern relations. The approach requires formalizing the pattern relationships in a pattern language grammar and annotation of the patterns with their effects on quality goals. As a result, the search space is narrowed down and the time spent evaluating alternatives is decreased. Mussbacher et al. [15] present a goal-oriented requirement language that formalizes the forces of patterns and relations between patterns.

Most of the existing approaches require additional efforts, such as specifying additional information about patterns or their relations, creating a knowledge base, or organizing the collection in a specific way. On the contrary, our system can handle different repositories and pattern engines and provide recommendations and other social features. As we show in the

sections below, our pattern recommendation service addresses some of the shortcomings of existing approaches for pattern selection by combining several pattern-related services and enhancing them with social features such as recommendations and tagging. The only additional (and optional) effort required is that of providing feedback, but, as we show, in some cases this can be automated.

3 Patterns 2.0: a composite service for pattern selection

The Patterns 2.0 service composes other existing services for delivering pattern-related content to the user. It is intended to be used by pattern users (including developers) and pattern writers. In the following subsections, we present the architecture and use cases of the Patterns 2.0 service.

3.1 Architecture

The Patterns 2.0 service is essentially a combination of existing services related to pattern selection. It takes as input user queries, forwards them to the appropriate services and integrates the results. The aim of the service is to improve search of patterns and to provide assistance for pattern selection. The nature and the purpose of the services composing the Patterns 2.0 is explained in this section.

The Patterns 2.0 service combines the following services as shown in Figure 1:

- *Pattern retrieval service*. A service that provides such functionalities as searching and retrieving patterns from a *pattern repository* that stores pattern descriptions. Planet is an example of such a service. It is wiki that allows authors to contribute patterns.

- *Pattern engine service*. A service with functionalities similar to those of a pattern retrieval service, but with the key difference that the system only maintains a *meta-index* to patterns described in full elsewhere. PatternSeer is an example of such a service.

- *Pattern recommendation service*. A service that provides recommendations[11] about patterns using a database of *pattern usage history* collected from past user interactions with the service. An example of such service is IC-Service, a general-purpose recommendation service [2], whose application to the problem of pattern selection is described in [3]. Other examples are described in [9, 13].

- *Pattern tagging service*. A service with functionalities similar to those existing on many pattern wikis: users can annotate patterns via tags, and patterns can be retrieved by an external service based on tags. Examples include PatternForge and Planet.

[11] By recommendations we mean hints on patterns that may be on interest to the user, considering the submitted query, something the user may (as opposed to must) find relevant

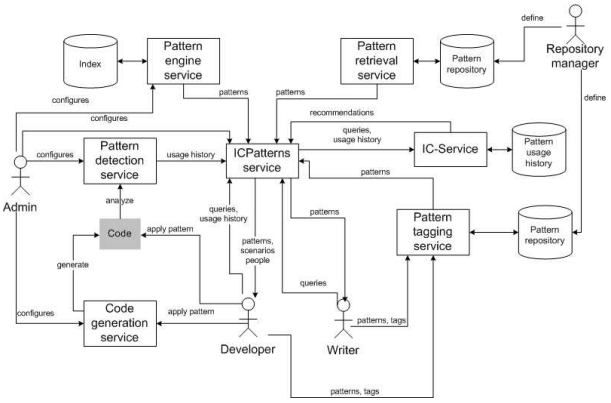

Figure 1: Architecture of the Patterns 2.0 service. The arrows denote the invocation flow and the labels denote information passed

Note, that we do not restrict pattern repository to GoF patterns. Instead, we assume that several repositories that use different pattern representation formats (PLML, etc.) and domains (patterns about security, organizing meetings, architectural patterns [12]).

Let us define possible API for the described services, in order to give more details about what is expected from them. The API are defined using pseudo code.

Patterns 2.0 service:

`Pattern[] getPatternsByProblem(problemDescription:String, context:Context)` - this function returns patterns relevant to the specified `problemDescription` possibly in a given `context`.

`Pattern[] getPatternsByKeywords(keywords:String, context:Context)` - this function returns patterns relevant to the specific `keywords`, possibly in a given `context`.

`saveFeedback(pattern:Pattern, keywords:String, accepted:Boolean)` - this function saves feedback: which `pattern` was accepted/rejected for a set of `keywords`.

Patterns retrieval service:

`Pattern[] getPatterns(keywords:String)` - this function returns patterns relevant to the specified `keywords`.

Patterns engine service:

`URL[] getPatterns(keywords:String)` - this function returns URLs to patterns relevant to the specified `keywords`.

Patterns recommendation service:

`Pattern[] getPattern(keywords:String)` - this function returns URLs to patterns relevant to the specified `keywords`.

`saveFeedback(pattern:Pattern, keywords:String, action:Action)` - this function saves feedback: which `pattern` was found relevant for a set of `keywords`. Here `action` refers to the feedback action, which depends on the type of user. For instance, in case of a pattern user, such actions can be applying pattern or marking it as not relevant. In case of writer, an example of feedback action could be rating the pattern.

Patterns tagging service:

`Pattern[] getPatternsByTags(tags:String)` - this function returns tagged with all specified `tags`.

The goal of the Patterns 2.0 service is to process the query submitted by a pattern user or a pattern writer

and transform it into several ad-hoc queries that will be forwarded to the appropriate service. For instance, to get results from the pattern tagging service, the query should be transformed into relevant tags and the `getPatternsByTag` function should be invoked. Furthermore, the feedback provided by the user is propagated to the recommendation service, to be stored in the usage history.

A query user submits to the Patterns 2.0 service includes a description of the problem and an optional context. The problem is described by a set of keywords, optionally restricted to specific elements of the pattern description, e.g. context or problem statement. An example of such a description could be "improve access control", or, in case of restriction: "improve access control, CONTEXT, SOLUTION". An example of the context could be the set of patterns already deployed in the project where the problem is encountered, e.g. "Authorization, Authenticator, TrustedProxy". The use of context can be used by the recommendation service in order to improve recommendations. For instance, to find in the usage history the situation, which is the most similar to the current search, the IC-Service calculates the similarity between users in terms of their past actions (queries and feedback). We believe that the personalization and contextualization of the query should allow for providing more relevant results than when only the simple keyword-based search already supported by pattern engine and pattern retrieval services is used. However, since the recommendation service does not host pattern descriptions, but only actions users performed on patterns, one or several other services (pattern retrieval service, pattern engine service, and pattern tagging service) are required for answering user queries.

There are three types of recommendations supported by the recommendation service:

- **Recommending patterns.** Recommending patterns suitable for solving a specific problem. Patterns matching a specific problem are returned in response to a query.
- **Recommending key patterns in a specific area.** Suggesting a list of patterns essential to a certain class of problems, or to the understanding of a particular repository of patterns (i.e. what is the best order to read the patterns in order to learn to use them).
- **Recommending pattern sequences.** Similar to recommending patterns, but recommendations consist of sequences of patterns to apply in a given situation. This takes relations between patterns into account. Sequences can also be mined from pattern usage.

The algorithms used for producing the recommendations are outside the scope of this paper and can be found elsewhere [3]. The purpose of this paper is to describe an architecture that embeds the recommendation service as part of an integrated system for pattern selection and application.

3.2 Use cases

The Patterns 2.0 service can be used in different ways: as a component of an ad-hoc pattern management system within an organization, on online pattern sites, as a plug-in into an IDE for developers, and so on. In the following, we describe general use of Patterns 2.0, classify potential users of the Patterns 2.0 service, and discuss example scenarios.

3.2.1 General description of use of Patterns 2.0

Figure 1 depict the invocation flow, described in this subsection. A user accesses the service by submitting a query via the user interface. We assume that the Patterns 2.0 service can be accessed in a number of ways: from a browser, from a plug-in to an IDE, or similar. After the Patterns 2.0 service receives a query, the service forwards it to the recommendation service. The problem description part of the query is forwarded

to the pattern engine and pattern retrieval services. Tags extracted from the query are forwarded to the pattern tagging service. Each invoked service responds to the Patterns 2.0 service with a list of patterns, which the Patterns 2.0 service combines in the results sent back to the user. In case of a pattern user, i.e. is searching for patterns to solve a specific problem, they can also get descriptions of situations where other users have used the pattern or a list of those users so that it is possible to discuss the problem with them.

At some point after getting the results, the pattern user applies one or several patterns. We provide the possibility for submitting this information to the Patterns 2.0 service as part of the usage history, i.e. feedback actions in connection with the previously submitted query. Such feedback is passed to the pattern recommendation service. In case a writer performs a search, the feedback contains relevance of the results, i.e. whether returned patterns are indeed related to the queried topic. In case of pattern user, the feedback contains information about which patterns were applied to which queries. Obviously, the key problem lies in the "observability" of the users' feedback actions, i.e. actions of using a pattern for a problem, or finding it relevant to the topic. In case of the pattern user, they can explicitly indicate that the pattern X has been selected for the problem A, where the problem corresponds to a search in the history of searches. The key challenge is providing them with motivation of doing this additional action. However, learning from Web 2.0 lessons, there should be a way for convincing users to provide feedback if they see value in recommendations they get as the result of this collaborative effort.

In case of the developer, the following two options can be also considered for observing feedback actions:

1. A pattern detection service (such as [1]) processes project documents and code, but in this case it is still non-trivial to link the detected patterns back to the query.
2. A case tool which supports (semi-)automatic implementation of patterns (such as [16]) could be used. The actions of the tool would provide the required trace. However, this case is limited to widely known patterns, such as GoF patterns.

In case of the writer, implicit relevance feedback (i.e. the feedback not requiring additional actions from the writer), such as clickthrough rate, time spent reading pattern description, and how the search session ended (see [7] for an overview) can be used.

3.2.2 Users of Patterns 2.0

Potential users of the Patterns 2.0 service can be classified in several groups:

- Pattern users who lack experience in applying patterns such as students, trainees or interns, people who are on their first architect/developer job, or have rarely used patterns before.
- People who have experience in using patterns, and, for instance, know how and when to apply the GoF patterns, but are unfamiliar with a specific pattern collection.
- Pattern writers who would like to find patterns related to patterns they are authoring.

Concerning the first and second group of users, the proposed tool could be very effective for organizations who maintain a pattern repository and infer which patterns are most useful under which conditions from the users' interaction with the repository.

With respect to the general architecture, users can play several roles when interacting with the various services:

- **Admin.** Configures services for a specific group of users.
- **Repository manager.** Defines the collection of patterns in a pattern repository.

– **Pattern user.** Interacts with the Patterns 2.0 service to get recommendations.

– **Pattern writer.** Interacts with the service to find patterns on a specific topic.

3.2.3 Possible uses of Patterns 2.0

To clarify the main uses of the Patterns 2.0 service, we consider the following specific "profiles" of the proposed architecture:

– **Poogle**[12]**.** In this profile, Patterns 2.0 is used to search patterns using pattern engine and pattern retrieval services.

– **Plickr**[13]**.** The use of Patterns 2.0 for tagging patterns and searching them using tags.

– **PatternLens**[14]**.** The use of Patterns 2.0 as a recommendation system, where users get recommendations about patterns to apply for solving a specific problem.

3.3 Example

In this section, we use an example to illustrate a possible use of the Patterns 2.0 service.

Let us assume that the Patterns 2.0 service uses a repository of security patterns, as configured by the Admin. The patterns in the repository defined by Repository Manager and are from the collection of security patterns previously hosted in patternshare.org [11]. Let us consider a developer who needs to improve access control in a system that offers multiple services. Suppose that for an experienced developer it is apparent to use the Single Access Point pattern.

However, our user does not know this, and therefore submits a query with the following problem description: "complex security control". The Patterns 2.0 service obtains the Single Access Point and Role Based Access Control patterns as results from the other services, discovers that other users previously used Single Access Point for similar problems and sends this information to the user. The developer then submits the feedback on the recommended patterns to the system.

Let us now consider a pattern Writer, who is preparing a pattern language on security in mobile applications, and would like to find related patterns and pattern languages. The user can submit a query on "secure mobile applications" to the Patterns 2.0 service and it will search for related patterns in pattern repositories and pattern search engines. The developer can browse through the list of results to see if there are related patterns to cite in his language. One can imagine an extension of this scenario, where query consists of patterns already present in the language, similarly to query-by-example approach.

4 Discussion

Since the Patterns 2.0 service does not store pattern descriptions, there is no copyright issues involved.

[12]This name combines "patterns" and "Google", one of the most popular search engines

[13]This name combines "patterns" and "Flickr", one of the most popular services for sharing (and tagging) photos

[14]One of the first systems for recommending movies was named "MovieLens", followed by several systems with similar names in different domains

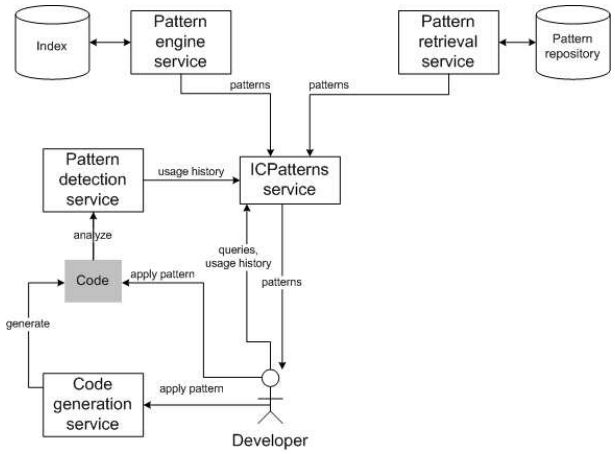

Figure 2: Using Patterns 2.0 for applying and detecting patterns in code.

The quality of Patterns 2.0 results obviously depends on the quality of its component services. One of possible future work directions can be investigation if Patterns 2.0 can perform better than the component services.

We can consider one additional use case for Patterns 2.0, shown in Figure 2. In this case, ad-hoc tools are adopted for automation of applying patterns and detecting them in the code. This Figure introduces two new services:

- *Code generation service*. A service that can generate code templates implementing the selected pattern. An example is described by O'Cinneide and Nixon [16].
- *Pattern detection service*. A service that analyses code to determine instances of patterns that occur in the code. The PTIDEJ tool [1] provides such functionality.

Such services can be also used by the Patterns 2.0 service for gathering feedback on chosen patterns and for providing pattern usage history, correspondingly. However, in the architecture of the Patterns 2.0 service, described in this paper, we leave out the description of tools that automatically generate code implementing patterns (code generation services) and of tools that automatically detect patterns in the code. The reasons why we do not include these tools are: (i) Even though several approaches for automatic detection and code generation have been proposed, the tools remain at the stage of prototypes and were not taken by the mainstream of IDE tools; (ii) such tools are usually limited to GoF patterns, while our solution is more general and does not depend on the repository; (iii) we would like to beyond software patterns, recommending also other types of patterns (e.g., patterns for organizing meetings or conferences), and such patterns are not related to code.

In this work we do not consider pattern languages that organize patterns in collections. The use of such languages can provide additional information for improving pattern selection, especially in the case of recommending sequences of patterns. The inclusion of pattern languages can be one of future work directions.

5 Conclusion

We presented the architecture and use cases of a composite service for facilitating the selection of patterns. A unique aspect of this service is its use of different types of services and sharing experiences among pattern users. The service composes other existing services (such as existing pattern repositories or pattern tagging services) for delivering pattern-related content to the user. Future work will include the definition of standard APIs for the services composing Patterns 2.0 and implementation of the service. This goal requires a collaborative effort of the creators of pattern management tools, and progress towards this end has been made over the last year. Information about the further development of this work will be available at `http://disi.unitn.it/~birukou/pattern_selection`.

Acknowledgement

We would like to thank our shepherd Klaus Marquardt for his many insightful comments.

References

[1] H. Albin-Amiot, P. Cointe, Y.-G. Gueheneuc, and N. Jussien. Instantiating and detecting design patterns: Putting bits and pieces together. In *International Conference on Automated Software Engineering*, pages 166–173, 2001.

[2] A. Birukou, E. Blanzieri, V. D'Andrea, P. Giorgini, N. Kokash, and A. Modena. IC-Service: A service-oriented approach to the development of recommendation systems. In *Proceedings of ACM Symposium on Applied Computing. Special Track on Web Technologies*, 2007.

[3] A. Birukou, E. Blanzieri, P. Giorgini, and M. Weiss. A multi-agent system for choosing software patterns. Technical Report DIT-06-065, University of Trento, 2006.

[4] J. Deng, E. Kemp, and E. G. Todd. Managing UI pattern collections. In *ACM SIGCHI New Zealand Chapter's International Conference on Computer-Human Interaction*, pages 31–38, New York, NY, USA, 2005. ACM.

[5] W. Fan, M. Gordon, and P. Pathak. On linear mixture of expert approaches to information retrieval. *Decision Support Systems*, 42(2):975–987, November 2006.

[6] S. Fincher. PLML: Pattern language markup language. report of workshop held at CHI, Interfaces, 56 (pp. 26-28). Technical report, 2003.

[7] S. Fox, K. Karnawat, M. Mydland, S. Dumais, and T. White. Evaluating implicit measures to improve web search. *ACM Trans. Inf. Syst.*, 23(2):147–168, 2005.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[9] P. Gomes, F. C. Pereira, P. Paiva, N. Seco, P. Carreiro, J. L. Ferreira, and C. Bento. Using CBR for automation of software design patterns. In *European Conference on Advances in Case-Based Reasoning*, pages 534–548, London, UK, 2002. Springer-Verlag.

[10] S. L. Greene, P. M. Matchen, L. Jones, J. C. Thomas, and M. Callery. Tool-based decision support for pattern assisted development. In *CHI 2003 workshop on HCI Patterns: Concepts and Tools*, 2003.

[11] M. Hafiz, P. Adamczyk, and R. E. Johnson. Organizing security patterns. *Software, IEEE*, 24(4):52–60, 2007.

[12] N. Harrison, P. Avgeriou, and U. Zdun. Architecture patterns as mechanisms for capturing architectural decisions. *IEEE Software*, July-August 2007.

[13] D. C. Kung, H. Bhambhani, R. Shah, and G. Pancholi. An expert system for suggesting design patterns: a methodology and a prototype. In T. M. Khoshgoftaar, editor, *Software Engineering With Computational Intelligence*, Series in Engineering and Computer Science. Kluwer International, 2003.

[14] D. May and P. Taylor. Knowledge management with patterns. *Communications of the ACM*, 46:94–99, 2003.

[15] G. Mussbacher, M. Weiss, and D. Amyot. Formalizing architectural patterns with the Goal-oriented Requirement Language. In *Nordic Pattern Languages of Programs Conference*, September 2006.

[16] M. O'Cinneide and P. Nixon. Automated software evolution towards design patterns. In *International Workshop on Principles of Software Evolution*, pages 162–165. ACM, 2001.

[17] L. Rising. *The Pattern Almanac*. Addison-Wesley Longman Publishing Co., Inc., 2000.

[18] I. Sommerville. *Software engineering*. Addison-Wesley, Boston, MA, USA, 7th edition, 2004.

[19] M. Weiss. Patterns and their impact on system concerns. In *European Conference on Pattern Languages of Programs*, 2008.

[20] L. Welicki, J. M. C. Lovelle, and L. J. Aguilar. Patterns meta-specification and cataloging: Towards a more dynamic patterns life cycle. In *International Workshop on Software Patterns: Addressing Challenges* at *COMPSAC 2007*, 2007.

[21] U. Zdun. Systematic pattern selection using pattern language grammars and design space analysis. *Software: Practice and Experience*, 37(9):983–1016, 2007.