

Software Pattern Communities: Current Practices and Challenges

Scott Henninger, Victor Corrêa
Computer Science and Engineering
University of Nebraska-Lincoln
256 Avery Hall
Lincoln, NE, USA
+1 (402) 472-2401

{scotth, vcorrea}@cse.unl.edu

ABSTRACT

Software pattern designers and users have few resources available to support pattern-based development practices. Patterns are currently disseminated in disjoint collections in various publishing mediums with little or no technology support. As the number of patterns and diversity of pattern types continue to proliferate, pattern users and developers are faced with difficulties of understanding what patterns already exist and when, where, and how to use or reference them properly. This defeats the very purpose of patterns as a medium to encapsulate and disseminate recurring design experiences. In this paper, an initial study among a set of pattern collections is performed to better understand the difficulties related to improve pattern-based support for software development activities. Based on the empirical survey, challenges are identified that define impediments to the federation of software patterns into an interconnected body of knowledge. A Semantic Web ontology is presented as an initial attempt at solving some of these issues through the use of Web-based ontologies.

Categories and Subject Descriptors

D.2.m [Software Engineering]: Miscellaneous – *Reusable Software*; K.6.3 [Management of Computing and Information Systems]: Software Management – *Software Development*.

General Terms

Design, Documentation, Management, Measurement, Experimentation.

Keywords

Design Patterns, Software Pattern Collections, Pattern-Based Software Development, Semantic Web, Ontologies.

1. SOFTWARE PATTERNS IN PRACTICE

The Software patterns encapsulate proven solutions extracted from the experiences of software developers that address recurring

Preliminary versions of these papers were workshopped at Pattern Languages of Programming (PLoP) '07 September 5-8, 2007, Monticello, IL, USA. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission. Copyright is held by the authors. ISBN: 978-1-60558-411-9.

problems within a context [25]. The concept of using patterns to disseminate and document design knowledge derives from Alexander's notion of design patterns for Architecture [4]. The main intention of design patterns has dual connotations: 1) provide a common vocabulary by which people can succinctly communicate well-known solutions to recurring problems; and 2) create a systematic language for developing holistic solutions by composing patterns at different levels of abstraction [3]. While the former concept of patterns as vocabulary has been widely embraced by the software patterns community, far less attention has been paid to meeting the challenge of achieving pattern languages for systematic design. While this problem has been recognized for some time [2, 37], little progress has been reported to date.

It can be argued that the informal use of software patterns have become ubiquitous in software development research and practice [24], at least with respect to an awareness of the topic and collective knowledge of a few well-known patterns. Current design pattern practices have focused on identifying and describing patterns and patterns collections/languages (see Section 2.2 for further discussion on pattern languages and collections). These patterns are designed for human consumption alone. Pattern users (software designers, etc.) are expected to study patterns in a collection and add them to their cognitive repertoire of techniques. This representation must be preserved, as most pattern collections are described at a level of abstraction that requires human interpretation of pattern contents and adaptation to the implementation context. But this practice is limited by human cognitive ability, that can only master a few patterns [1, 37] that add to an already burgeoning tool mastery burden [11] facing software developers. Patterns now exist for a wide range of software development topics, from process patterns to code patterns at various levels of abstraction to maintenance patterns. The scale of published software patterns is reaching a point where it is becoming infeasible to know all potentially relevant patterns, let alone understand when a given pattern should be applied to a specific context. The need for tools to help people find, understand, and apply patterns is becoming a critical need.

In addition, free text representations severely limit the potential of tool support for pattern-based design methods. More formal specifications for pattern languages enhance machine processing capabilities [17, 34], such as search and automated translation to code or models, but lose the human readability aspects that are critically important to the utility of software patterns. *Representations and tools are needed that both retain human*

readability while enhancing automated processing and search/browsing capabilities.

The overall objective of this research is to describe the current state of software patterns and enumerate existing barriers for using patterns as a more effective software development tool. We begin by surveying currently available pattern collections, focusing on the scale, diversity, and other factors that characterize current software pattern practice. Drawing on this empirical data, we then identify a number of challenges for transitioning from current practices to realizing the potential of patterns as a unified (federated) body of knowledge. We conclude by briefly describing our plans to utilize Semantic Web technologies as a promising technical solution that meets many of the challenges we identify.

2. SURVEYING SOFTWARE PATTERN COLLECTIONS

The overall goal of the software pattern community has been to build a body of literature to support software design and development efforts. This culture of focusing on documenting sound design principles and cataloging best practices are a first step toward codifying software design knowledge. This has in turn led to the development of a number of patterns across a wide range of topics.

To better understand the scope and content of current available patterns, where “availability” is defined as being either in a published form (books, journals, proceedings) or in Web pages, we have conducted a survey. Thus far, we have sampled 170 pattern entities (collections and individual patterns not in a collection) with a total of 2,241 patterns. Although “patterns” (in the Alexandrian sense) have been created for a number of disciplines, we focused solely on those related to software development and the software development process, including topics such as software project management. The patterns surveyed ranged widely from those that were closely related to programming activities and could potentially be used in automated code development to process and management patterns that are strictly informational. The following sections explain our findings in detail, but we should be clear that our purpose is not to simply enumerate the different patterns available, but to analyze our findings to find current trends in pattern practices. Since pattern collections normally represent a coherent domain of interest, we focused on “collections”, sets of patterns (although some are single patterns) gathered in a single location, and the types of patterns these collections contain.

2.1 Method

The definition of a pattern is particularly troublesome. One could say it is a structured text representation consisting of attributes describing a ‘problem’, a ‘solution’ and a ‘context’ and/or ‘forces’. But there are many known software patterns that use freeform text and others that use concepts that do not map well to problem/solution/context/forces. In addition, due to the lack of clearly defined standards, any author can claim that their text is a pattern. It takes some community debate and a considerable amount of expertise to state “definitively” whether the text is indeed a pattern.

Given these issues we adopted the following attributes: *Problem AND Solution AND (Forces OR Context)* as our least common denominator for considering the contents of a collection as

patterns. Where possible, we sampled the contents of collections, looking for evidence that these attributes, whether in structured or free-form text, were present. This was not possible for all collections. For example, we cannot reasonably have access to all pattern books and other printed publications. In these cases, we made conjectures about inaccessible pattern collection contents based on tables of contents, Web pages, and other sources. Appendix A shows a full listing of all collections used in this (evolving) study.

We began with some well-known pattern collection *portals* such as Hillside [28], Appleton’s Web page [6], the Portland Pattern Repository [16], etc. The collections referenced in these portals sometimes pointed to other collections that were included as well. Note that both the searches and pointers to collections referenced books and published literature as well as other pages (see Section 2.8). There were also a number of other well-known pattern *collections* that were used as initial references. Amongst these are the GoF book [25], PLoP proceedings, POSA [12, 30, 42], Fowler’s Analysis Patterns [23], the Amsterdam Usability patterns [50], Tidwell’s UI Patterns [48], the J2EE Patterns [46], the Yahoo! Pattern Library [51], and a handful of others. These collections also provided a set of exemplars for considering other pattern collections and often provided references to other collections.

When these links were exhausted, Google was used to search for pattern collections using key phrases such as “software pattern”, “pattern collection”, etc. Once a collection was found, we sampled patterns from the collection or carefully considered whether the assertion that the reference contained software patterns was sound. In some cases an informal reputation criteria was used. I.e. if the collection was referred to as a pattern collection by known pattern collections, then it was considered a pattern collection for our study.

In the end, it is impossible to tell whether something is a “pattern” or not and could be the source of considerable debate. Our approach was to make careful but practical conjectures on what we felt could be named a pattern/pattern collection. As this research proceeds, we will seek input from the patterns community, both authors and users, to help refine our conjectures for accuracy and continuously refine our listing, the current version of which is found in Appendix A.

2.2 Patterns and Pattern Collections

The definitions we used are as consistent as possible with current software pattern literature. *Patterns* are considered as structured entities that address a commonly recurring problem within a context. For this study, we do not make any value judgments on the validity or quality of patterns, whether they have been properly vetted, or whether they were duplicates (although see Section 2.5). *Pattern collections* are loosely coupled patterns located in a common location (repository, paper, book, Web site). Most collections address a fairly homogeneous set of topics and consistently use a common *pattern form*, a set of attributes used to describe the collection’s pattern, although pattern form vary widely between collections.

Many collections are referred to as *pattern languages*. It can be argued that many of these languages, which in Alexander’s vision were connected by a kind of “grammar” that supported the composition of patterns from large to small scale [4], lack the

means to systematically compose patterns into holistic design and therefore are not “languages”. In the very least, one would expect the pattern language to show clear dependency relationships between the patterns, such as one pattern being required before another pattern can be applied. We will leave it as a topic for further debate in the community and/or future research define precise differences between languages and collections [41]. For the purposes of this study, *we have opted to use the term “collection” to refer to any body of patterns, whether considered a language or not.* The overall criterion we want to communicate is that individual patterns should be seen as a piece of a larger puzzle that together sheds light on a body of design knowledge. Indeed, the objective of our future work is to provide the means to put these pieces together in a meaningful way.

2.3 Scale and Availability of Software Patterns

Even before 2000, when Rising published a catalog of over 1000 individual patterns [40], it was stated that “...there are now so many patterns it is very difficult to remember them all” [14] and that “the increase in the number of Design Patterns makes a common vocabulary unmanageable” [1]. Since then, the number of patterns has more than doubled and has been created for an increasing diverse set of software development topics. Figure 1 shows our current sampling in terms of the year they were created (we could not determine the year of origin for 9 patterns).

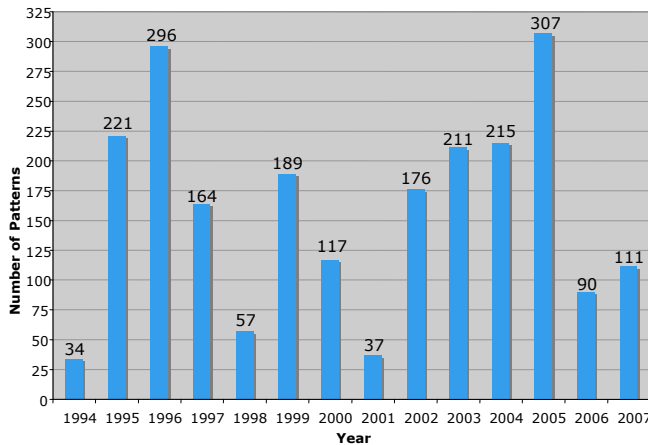


Figure 1. Number of Patterns, 1994 - 2007.

The size of collections ranges from 1 (which really isn’t a collection) to 146. Note that with the exception of a down year in 2006, the number of patterns developed have been rather consistent at over 200 patterns per year since 2002 (2007 has partial data as the survey was performed in April 2007) Figure 2 reveals that collections tend to be small. Excluding the 46 individual patterns, 70 of 121 collections (58%) have between 2 and 10 patterns. The mode is 5 patterns in a collection and the average is 18, being skewed by a collection with 146 and two with over 90 patterns. The pattern listing in Appendix A is sorted by the number of patterns in the collection.

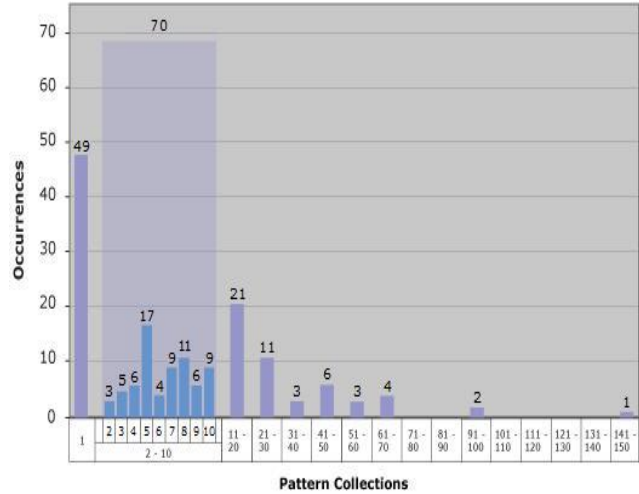


Figure 2. Number of Patterns within Collections.

2.4 Types of Software Patterns

The development of pattern languages addressing holistic solutions for software requires patterns that address a wide variety of topics. Table 1 shows a subset of these topics that are related to technical (software-oriented) domains. The largest number of patterns is User Interface design patterns. OO Design has the largest number of collections, meaning that UI design collections tend to be larger in size. Not all patterns address software development technologies. Forty-one of the collections (546 patterns) do not fall under the 17 categories shown in Table 1. Many of these patterns address specific application domains, leading to even more diverse pattern types.

Table 1. Pattern Diversity by Technical Domain

Type	#Collections	#Patterns
User Interface	14	425
Programming Languages	14	243
Architecture	11	231
OO Design	33	161
Workflow	11	149
Systems	14	140
Communication	11	91
Database	5	54
Frameworks	4	51
Components	3	47
Parallelization	3	35
Security	2	16
Management	2	12
Concurrency	7	11
Networking	3	11
Information Integrity	1	10
Fault Tolerance	1	8

Another measure of pattern diversity is the ability to address various software development issues, both process and lifecycle. Out of all patterns found, about 86% of the patterns addressed software development issues. Figure 3 shows the distribution of patterns across types of software development activities. Design and Architecture patterns constitute a majority of the types of development patterns (65%). The types of patterns available are quite broad, although testing patterns, in particular, seem underrepresented relative to the amount of effort that goes into testing techniques. Thirty of the collections, containing a total of 315 patterns, were not classified as *software development* patterns and do not appear in Figure 3.

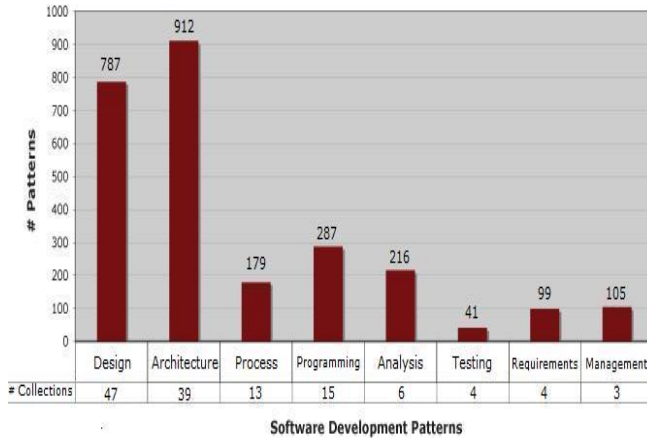


Figure 3. Types of Software Development Patterns.

2.5 Variants and Duplicates

In our investigations, we have found few instances of direct duplication. For example, there are at least four instances of the “Breadcrumbs” usability pattern [13, 43, 50, 51], one of which uses the (more appropriate) name “Homeward Bound” [13] (which includes a study showing that Breadcrumbs does not solve the problem – enhancing navigation in Web sites). But pattern variants are much more common. For example, Dyson and Anderson split the GoF State pattern into a set of intra-related patterns forming a language of the overall GoF State pattern [5]. Variants of the GoF Observer pattern include the “Extended Observer” [49] and “The Middle Observer” [29]. GoF Patterns have also been combined to make new aggregate patterns such as the Managed Observer, which combines the Observer and Mediator patterns [34].

There are many other examples that seem to be valid by Alexander’s definition that a good pattern describes “the core of the solution to that problem in such a way that you can use the solution a million times over without doing it the same way twice” [3]. Others are more oriented toward specific implementations. For example, the GoF Iterator pattern has documented variants including patterns that follow the Iterator and Enumeration classes in Java [18]. Some of these implementation-oriented patterns may not be considered as distinct from the original Iterator pattern by many pattern experts.

There are often good reasons for these variants, and they therefore not only need to be embraced, but represented in terms of how and when the variants should be used. This also adds a dimension

of semantic complexity to the problem of finding appropriate patterns. I.e. once appropriate patterns are found, a secondary task arises to choose which variant is best suited to the task at hand.

2.6 Pattern Relationships

Perhaps most concerning for the development of systematic pattern-based methodologies is that patterns tend to be defined in isolation from other pattern collections, having no inter-collection links or relationships. While many pattern collections either have explicit references to “related patterns” or embed pattern relationships within pattern descriptions, most relationships are intra-collection, i.e. between patterns within the collection. Cross-collection (inter-collection) relationships are rarely found, and most references are to a minority of collections, notably the GoF or POSA patterns. Out of 170 collections, we were able to find only one instance that lists URL references to patterns in other collections, the Web patterns collection [43]. However, the URLs in this collection are listed in plain text and not hyperlinks.

Even within pattern collections, intra-collection relationships are not always represented explicitly through a “related patterns” or other attributes. Even rarer are instances in which machine-processable links, such as URLs, are used. As stated, some links between patterns in the collection are found in the pattern text, a reasonable way to describe a pattern and its overall context with other patterns. Nonetheless, the lack of explicit links between patterns to define relationships between patterns, whether inter- or intra-collection, remains an impediment for computational pattern language support.

2.7 Pattern Forms

One issue that may contribute to the lack of cross-reference (inter-collection) relationships is the lack of consistency between pattern forms. Most pattern collections use a common pattern form, consisting of a set of named attributes that describe collection patterns, to describe all patterns within the collection, although some collections use a flat-text format. Almost every pattern collection we surveyed used a *different* pattern form. Table 2 shows some of the complexities involved through three example pattern forms. Even where the attributes have the same meaning, different terms are used, such as “also known as” and “alias”. Others are more subtly similar, such as “motivation” (GoF) and “problem” (POSA), which may be misaligned enough to not be used as the exact same attribute.

Standard formats have been proposed to incorporate a wide variety of pattern forms. PLML is specified as a DTD schema where none of the elements are required so that free-text forms can be accommodated [21]. This allows flexibility, but still does not accommodate all pattern forms, as shown in Table 2. Not all pattern form attributes are appropriate for all pattern types. For example, the GoF ‘collaborations’ and ‘participants’ attributes refer to specific object-oriented design constructs and will not be appropriate for other design methodologies or other pattern types. Any standard form will need to be both flexible and able to accommodate a wide variety of pattern types while retaining a degree of formal representation for computational queries and browsing.

2.8 Pattern Publishing Mediums

Patterns are available in a number of publishing mediums, from books to proceedings to Web sites. Figure 4 shows the distribution of patterns across these mediums. Much of the distinction is between printed and electronic mediums. Although 31% of the patterns are electronically inaccessible in book format (proceedings, journal, book), 69% are electronically accessible in the Web. However, less than half (44%) of the Web-accessible patterns are represented using structured text such as HTML (10% of patterns), or XML (1 collection of 120 patterns). The other 57% are available through PS/PDF/Word files. In Figure 4, “Hardcopy” means any printed form, such as books, proceedings, and journals. Patterns in the “Hardcopy & PS/PDF/Word” category means that the patterns were published in hardcopy *and* all patterns in that publication are also available in a downloadable form. For example, the GoF patterns are available in book form only and therefore appear in the “Hardcopy only” category. As an example, although various GoF patterns are available on the Web from third-person authors, the original book is not available electronically. PLoP proceedings are hardcopy but can be downloaded in PDF format. Therefore, they are placed in the “Hardcopy & PS/PDF/Word” category. The same is true for the “Hardcopy & HTML”, although some Web pages for books have only a subset of their patterns online. These are divided into their respective categories. For example, suppose we have a printed collection of 24 patterns, 10 are in the publication’s Web page. Then 10 would be used for the “Hardcopy & HTML” category, and 14 (24-10) appear in the “Hardcopy only” category.

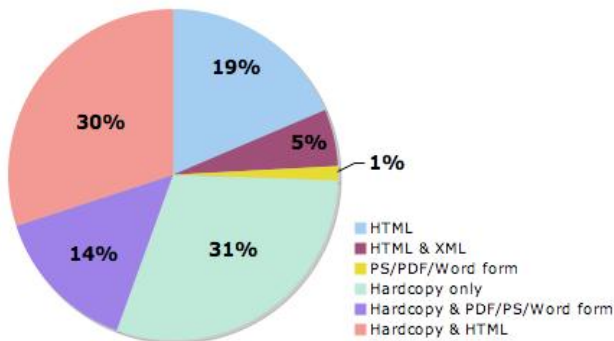


Figure 4. Types of Electronic Accessibility.

3. TOWARDS PATTERNS AS A UNIFIED BODY OF KNOWLEDGE

There is a great potential for software patterns to become a medium for defining knowledge about best practices for software development and about domains of expertise in software development. In many respects, this is already happening. The process of vetting patterns through shepherding processes is a peer review process that ensures a degree of quality. In addition, most patterns define structured knowledge representations (pattern forms) that can be utilized to search for relevant patterns by different attributes – problem, solution, context, author, etc.

But software patterns have yet to receive the widespread use commensurate with the potential of the technique. As shown in our study, the scale and diversity of patterns has reached the point

where tools are needed to help pattern users and developers find and discover potentially relevant patterns. Critical to the issue of tool support is utilizing existing patterns and defining the infrastructure for new pattern development and refinement. Given the haphazard way in which patterns have been created thus far, many issues need to be addressed before software patterns become an integral part of software development practices.

Table 2. Mappings Between Three Pattern Forms.

GoF	POSA	PLML
name	name	name
author	author	author
implementation	implementation	implementation
consequences	consequences	
known uses	known uses	
structure	structure	diagram
motivation	problem	problem
applicability	context	context
related patterns	see also	related-patterns
intent		
collaborations		
participants		
sample code		
also known as		alias
	summary	synopsis
	solution	solution
	example	example
	example resolved	
	dynamics	
	variants	
		forces
		evidence
		rationale
		literature
		confidence
		management
		illustration
		pattern-link
		creation-date
		credits
		last-modified
		revision-number

3.1 Six Challenges for Federating Software Patterns

Through our empirical work, we have identified a set of challenges for federating the currently disconnected realm of

pattern collections into an interconnected body of knowledge. Our challenges are heavily biased toward federating currently heterogeneous patterns in a distributed electronic format utilizing Web technologies. In addition, the development of communities that build on their collective intelligence in a “network effect” [10] is crucial to the realization of this vision. To achieve these goals, the following challenges must be met.

3.1.1 *Electronic Accessibility*

A wide variety and large number of software patterns are available in electronic form. While all of these can be accessed through the Web, about a quarter of these are available in HTML and XML, a total of 537 patterns in the collections we surveyed. Many more are available in PDF or other txt-based document formats. The challenge is to turn these patterns into formats that can be searched and browsed through pattern attributes. XML formatting is most amenable to this and other forms of machine computation. HTML and other file formats will either need to be converted into some XML or database form or have some kind of wrapper that supports attribute-based querying. While this involves some effort, the benefit of interconnecting the patterns may prove worthwhile.

3.1.2 *Lack of Standard Pattern Forms*

The pattern forms in Table 2 are indicative not only of the heterogeneous pattern forms available, but also the complexities involved in reconciling the attributes of forms to support querying and browsing. The lack of formal and widely adopted standards adds a rather cumbersome barrier to develop patterns in a way that can be meaningfully communicated and inter-linked. However, it is neither possible nor desirable to create a single pattern form that meets the needs of all types of patterns. Different pattern types may require different types of attributes. Techniques are needed to create relationships between pattern attributes such that different collections in different forms can be used as a federated whole while accommodating necessary differences for different pattern types.

3.1.3 *Inter-Pattern Relationships*

Defining intra-pattern relationships within collections, which is not a universal practice for pattern collections, is clearly only a first step towards understanding how patterns can and should be used together. Defining inter-pattern relationships is far less common, to the point that the practice does not exist at all. Not only does this make it difficult to federate pattern collections, but larger, more damaging, implications can be found when considering the severe paucity of knowledge about the interrelationships between patterns – for novices and experts alike. Software patterns and collections tend to be written to solve specific problems with little to no regard about how the pattern could or should be used with other patterns. This makes it all the more difficult to understand the interdependencies, potential side-effects, or benefits of using pattern combinations.

There have been some attempts to define standard relationship types between patterns. Noble defined three “Primary Relationships”, Uses, Refines, and Conflicts, and a number of “Secondary Relationships” (expressed in terms of the primary relationships), Used by, Refined by Variant, Variant Uses, Similar, Combine, Requires, Tiling, Sequence of, and Elaboration

[36]. These are good starting points for defining pattern relationship semantics, but are by no means a complete list, and have certainly not become an integral part of defining patterns. The lack of infrastructure (relationships types, semantic links, etc.) for defining inter-collection relationships makes it extremely difficult to devise a true pattern “languages” that integrate different kinds of knowledge for a holistic solution.

3.1.4 *Software Pattern Validation*

Very little work has been done to capture pattern validation efforts. With the exception of the “confidence” and “evidence” attributes in PLML [21], patterns and pattern forms do not explicitly represent information about pattern validation. While patterns in PLoP proceedings undergo a rigorous shepherding process through Writer’s Workshops [38], this and subsequent validation information is lost. Information associated with validation and empirical evaluation efforts for patterns and issues associated with the patterns need to be captured and associated with the patterns to help designers make informed decisions on how and when to use the pattern. Pattern usage information is also crucial to the effective application and evolution of patterns. Information such as how a pattern was applied to different context, caveats, etc., is all critical information for the pattern user.

3.1.5 *Tracking Software Pattern Variants and Duplicates*

Closely related to pattern validation and the need for community-based control of pattern creation is the need to track pattern variants and duplicates. Duplicates should be allowed – people may want to express the patterns differently and should be allowed a certain degree of expression. Variants are more difficult, as there are many types of valid variants, some examples of which were described in Section 2.5. There is currently no mechanism for tracking such variants. Some means is needed by which a community of experts can comment on and arrive at a consensus on whether a pattern is a duplicate, an implementation, a refinement, specialization, etc. Tracking these types of variants will not only provide the means to browse and query distributed patterns, it will provide the means for a greater understanding of the knowledge behind the patterns for both pattern creators and users alike.

3.1.6 *Updating Software Pattern Knowledge*

Patterns are currently written and disseminated in a static form. Once the pattern is published, changes become difficult to track and enforce, with the possible exception of edits performed by the authors of patterns disseminated in Web mediums. In some respects, this is expected, as the pattern should be “timeless”. But with the rapid pace of change in technology in the software field, this rule may not hold. Improved patterns could be created, refinements may become more useful than the original or other variants, etc. Allowing these refinements can lead to more accurate and up-to-date knowledge. Some form of version and change control may also become necessary. Usage data, instances where one or more patterns are used can also be captured, leading to information on how useful a pattern is would also be a valuable source of validation information.

All of these issues involve viewing patterns not as isolated collections of information, but as an interconnected corpus of patterns. Furthermore, the creation of pattern languages will be facilitated to the extent that patterns are defined with meaningful relationships between them.

4. UTILIZING INTERCONNECTED SOFTWARE PATTERNS

Our study leads to the inevitable conclusion that the volume, diversity, and disconnected nature of current software pattern practices have become significant barriers to the effective use of software patterns in the software development process. Informal success stories lend credence to the value of patterns and pattern language as a potentially valuable software development technique. But it is surprising how few developers know about and/or use patterns [27]. The sparse success stories need to be broadened to more ubiquitous practices used and known by the majority of software developers. A central contention of our research is that *loosely coupled and isolated collections of patterns, however well specified and/or catalogued, cannot alone provide significant improvement for software design productivity and quality.* Current informally practiced techniques, particularly given the failure to include cross-collection relationships, fall far short of the original vision of pattern languages as organized collections of patterns informed by their context of use [4].

Widely adopted standards are necessary but face significant problems with reconciling diverse pattern forms, many of which have domain-specific attributes that are necessary to properly define patterns of that type. An alternative approach is to construct formal models of software patterns that support translations and/or transformations between forms. In addition, formal specification of design patterns can enhance the understanding of their semantics [47], for example by explicitly showing how a pattern solution is associated with a design problem (perhaps via explicit forces) within a context. This can help users decide which patterns are most appropriate for a given design problem and how the patterns can be combined. Formalization can also support a wide range of pattern-based tools, from finding instances of patterns in programs and fine-tuning them to meet pattern specifications [20] to helping designers find and adapt relevant patterns.

4.1 Web-Based Ontologies

Building on our survey results, we are investigating the use of Semantic Web ontologies [9, 35] to formally define patterns and semantic relationships between patterns that can be distributed across collections in the World-Wide Web. The use of ontologies to represent pattern languages is a marriage of two complementary philosophies. An objective of pattern languages is to provide the means for professionals to use a common *vocabulary* about design and other issues [25]. An ontology, often defined as a “formal, explicit specification of a shared conceptualization” [26, 45], consists of a *vocabulary* of concepts, relationships, and axiomatic definitions. Ontologies are therefore a natural extension to the essential design pattern goal of providing a common vocabulary to communicate design concepts. For these reasons, we are investigating the feasibility of using Web-based ontologies to formally represent shared vocabularies that can be used as a framework for pattern languages.

We are in the early stages use a semi-formal approach that defines pattern relationships using formal Description Logic [7] implemented in the Web Ontology Language (OWL) recommendation from W3C. OWL defines a frame-based knowledge representation language with axiomatic constructs for logic-based expressivity that can be distributed over multiple files in the World-Wide Web [33]. OWL includes vocabulary for describing properties and classes that support the construction of class taxonomies and relationships between class properties and class instances. OWL Description Logic (OWL-DL) is founded on decidable fragments of first order logic and axiomatic definitions that can be used by reasoners to infer new facts and to check the consistency of resulting ontologies [8]. OWL properties are predicates that operate on subjects (domains) and map to objects (range). Range values can be restricted through various axiomatic class construction operators.

4.2 Ontology-Based Pattern Representations

Figure 5 shows a screen images from the OWL ontology editor Protégé [44] displaying very early work in creating Web-based ontologies for pattern forms. The figure shows a set of pattern forms arranged in an inheritance hierarchy, including the Pattern Forms in OWL (PFOWL – pronounced *fowl*) form, our ontology-based pattern form derived from the PLML standard [21].

OWL is designed to be compatible with XML technologies. The `plm:`, `gof:`, `posa:` and `pfowl:`, prefixes that appear in the left-hand window of and elsewhere are XML namespace abbreviations [31]. These indicate that the constructs come from different OWL files that can be distributed across the WWW and federated into a single location for computational purposes (search, reasoning, etc.). In our example, the namespaces represent common pattern forms located in different files and federated through the OWL import mechanism into our PFOWL file. The `plm:` namespace defines our essential form (Problem AND solution AND (Forces OR Context)), the Coplien form [15], and the “canonical” form [6]. The `gof:` namespace the Gang of Four [25] form. Note that the `plm:` namespaces build on each other by inheriting properties, while the `gof:GoF_Form` starts from the base (empty) PLForm (“Pattern Language” Form). The `posa:` namespace represents the Pattern-Oriented Software Architecture [42] form. This form inherits from the EssentialForm and adds new properties as defined by the POSA form.

The EssentialForm pattern form properties (pattern form attributes) are shown in the top-left window of Figure 5 (follow 1). This defines four main types of properties, Problem, Solution, Forces and Context, along with the pattern name and author. The UsabilityPatternCollection specializes the PFOWL form for use in usability patterns (see 2). This form builds on the other forms (note the namespaces – for example, `hasImplementation` comes from the `gof:` namespace) to add a number of properties defined in the PLML standard. In addition, the universal quantifiers restrict the range of values for a property to a class. This enables consistency checking and inferencing while allowing reuse of concepts.

Note that each of the concepts representing pattern forms is intermixed within the inheritance hierarchy. This is a degree of flexibility not afforded with other computational formats such as XML and provides a powerful distributed framework for defining

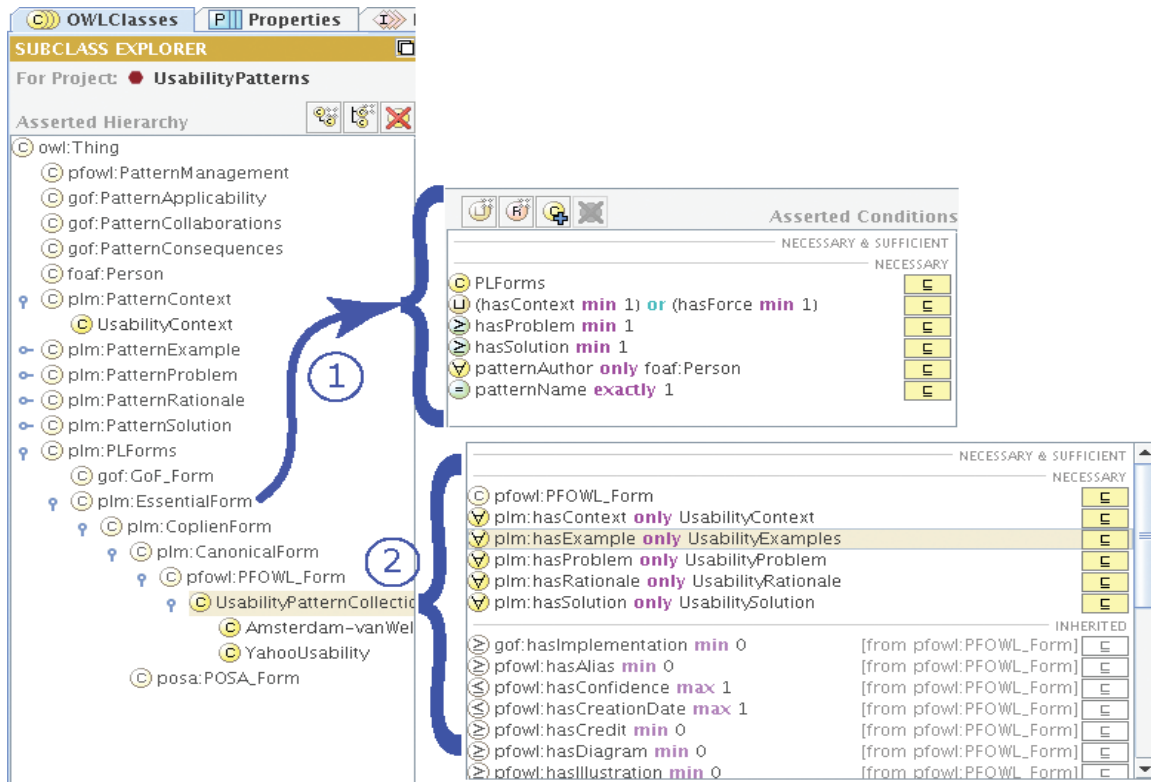


Figure 5. Pattern Forms in PFWL.

and maintain ontologies. For example, another pattern collection designer may want to create a hybrid form that adds `inCollection`, `hasKnownUse`, and `hasImplementation` to the `EssentialForm`. This can be easily done through an ontology editor that imports the `EssentialForm` and `PFWL` ontology files. The new pattern form would be created by constructing a subtype of `plm:EssentialForm` and adding the properties `powl:inCollection`, `plm:hasKnownUse`, and `gof:hasImplementation`.

A key element of our approach to pattern representation is the ability to federate distributed pattern collections. Pattern designers retain local control over their patterns while continuing to use pattern forms that are convenient for them. Federating distributed pattern collections involves two distinct problems that are addressed by Semantic Web technologies: 1) patterns can be located on different machines distributed throughout the Web while retaining unique identities; and 2) different pattern forms can be used together as a unified whole to the extent that semantic matches exist between attributes in the forms.

Due to space constraints and the objectives of this paper, we are only able to provide this small glimpse into how OWL and Semantic Web technologies can be utilized to federate heterogeneous and distributed patterns. This continues to be ongoing work and future papers will provide further details on how this approach works and how it can be utilized to create an *infrastructure for creating semantically interconnected pattern languages*.

4.3 Related Work

Our ontology-based approach is similar in scope to some formal approaches for specifying patterns. Previous research in this area

all build on formal specifications of object-oriented languages and have focused on a subset of the GoF design patterns. LePUS (Language for Pattern Uniform Specification) uses first-order logic to describe structural properties of design patterns [19] through formula-based mechanisms and visual representations. LePUS is based on ‘fragments’, which are abstractions of design elements, such as classes, patterns, methods, and code that contain roles or slots which are filled by other fragments to produce an interconnected architecture [22]. An extension of LePUS (extended LePUS or eLePUS) broadened the range of patterns by adding representations for intent, applicability, and collaborations [39]. DisCo (*Distributed Co-operation*) uses a form of Temporal Logic of Actions (TLA) [32] to formally describe constraint interactions for reactive systems [34]. While LePUS efforts focus on the static aspects of patterns, DisCo is primarily concerned with the behavioral aspects. BPSL (Balanced Pattern Specification Language) combines both approaches into a language designed to specify the ‘solution’ element of GoF design patterns [47].

All of these formal methods are based on models of object-oriented systems and therefore do not scale to other types of patterns such as process or usability patterns. In addition, while these approaches all have reasonable formal representations of patterns, none have adequately examined the types of rigorous reasoning enabled by their techniques, focusing instead on representations only. Nor have they been particularly clear on why the formal descriptions are needed and how the benefits of formally defined patterns can be utilized to outweigh the obvious costs of describing patterns using formal notations.

5. FUTURE WORK

A survey such as this one is only a representative example of the actual data that exists. In our case, there are many patterns we were probably not able to find, and absolute completeness will run into a point of diminishing returns that will make further efforts infeasible. Our central claim is that we have captured a sufficient breadth and depth of the currently available patterns to make valid statements about software pattern that have currently been created.

Nonetheless, the data presented here is seen only as the beginning of a dialog to both inform the community of existing patterns and allow the community to tell us what collections and patterns have been missed, need updating, etc. We plan to develop a simple interface to the overall data built on OWL data and integrated into a Wiki structure for collaborative editing. The objective would be to continuously refine our knowledge of existing patterns by drawing on the collective knowledge of the community while providing a search-and-browse interface to explore pattern collections and some of the data presented here.

The ontology-based pattern forms is in its formative stages. We believe that Web-based ontologies have the potential to address the challenges presented in this paper and will work to address each of the challenges. Work will continue to both refine the ontology and add pattern collections as instances in the federated data. Some pattern collection owners have agreed to allow us to represent their collections in our ontology. Through these efforts, we will refine and build the ontologies to suit different patterns and pattern forms while creating the added value of semantically interconnected patterns.

Relationships between patterns in different collections currently do not exist, much less semantic relationships. We will continue to explore refinements to Noble's pattern relationship types [36]. In addition, relationship between pattern instances must be researched and created. We hope to open a dialog with the patterns community on this issue, which has barely been explored thus far. Again, Wiki structures and cultivating a community interested in creating inter-collection pattern relationships will be critical to ensure accuracy and approach completeness.

6. CONCLUSIONS

The dual goals of pattern languages, to provide a common vocabulary of succinct communication concerning design problems and the creation of a systematic language for composing holistic design problems, has the potential for significant impact on software development practices. Unfortunately, significant barriers exist for the realization of these goals. With over 2200 patterns available, no coordination between isolated pattern collections, complex pattern variants and a lack of standards (flexible or otherwise) for creating patterns, patterns risk being lost in a babble of disconnected voices.

As an initial inquiry into the current state of software pattern practices, we have surveyed published pattern collections to draw conclusions on current challenges for taking patterns to the next level as a viable software development practice. The good news is that the body of knowledge collectively represented by patterns is vast and increasing. The bad news is that it has reached the point where it is difficult to find and select relevant design patterns, particularly when the differences are subtle.

While a focus on tools has astutely been avoided in favor of creating pattern content, the problem is reaching, or has already reached, the point where we can no longer require software professionals to read a couple of books on software patterns and expect that their "cognitive toolbox" will sufficiently cover a sufficient range of known patterns. Tools are needed, not just to search for patterns, but to create an awareness of existing patterns, browse pattern collections, collect relevant patterns for specific efforts, create systematic pattern languages for design, etc.

7. ACKNOWLEDGMENTS

This research is funded by CCF 0613985 of the National Science Foundation.

8. REFERENCES

- [1] Agerbo, E., Cornils, A. 1998. How to preserve the benefits of Design Patterns. *OOPSLA '98*, Vancouver, Canada, 134-143.
- [2] Alexander, C. 1996. The Origins of Pattern Theory: the Future of the Theory, and the Generation of a Living World. *OOPSLA 1996 Keynote Address*, Available at: <http://www.patternlanguage.com/archive/ieee/ieeetext.htm>.
- [3] Alexander, C. 1979. *The Timeless Way of Building*. Oxford Univ. Press, New York, NY.
- [4] Alexander, C., Ishikawa, S., Silverstein, M. 1977. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, New York, NY.
- [5] Anderson, P., Dyson, P. State Patterns, 1998. In *Pattern Languages of Program Design 3*, R. Martin, D. Riehle, F. Buschmann, F., Eds. 125-142.
- [6] Appleton, B. 2000. Patterns and Software: Essential Concepts and Terminology. Available at: <http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html>, Updated: Feb., 2000.
- [7] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., Schneider, P. P. 2003. *The Description Logic Handbook*. Cambridge Univ. Press.
- [8] Baader, F., Horrocks, I., Sattler, U. 2003. Description Logics as Ontology Languages for the Semantic Web. In *Lecture Notes in Artificial Intelligence*, vol. LNCS 2605, Springer, 228-248.
- [9] Berners-Lee, T. 1998. Semantic Web Roadmap. *W3C Semantic Web Vision Statement*. Available at: <http://www.w3.org/DesignIssues/Semantic.html>.
- [10] Berners-Lee, T., Fischetti, M., Dertouzos, M. L. 2000. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*. Harper Business.
- [11] Brooks, F. P. 1987. No Silver Bullet: Essence and Accidents of Software Engineering. In *Computer*, 20(4), 10-19.
- [12] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. 1996. *Pattern-Oriented Software Architecture: A System of Patterns*. Wiley.
- [13] Clemens, A. 2007. The Diemen Repository of Interaction Design Patterns. Available at: <http://www.visiblearea.com/cgi-bin/twiki/view/Patterns/Home>, Updated: Aug. 18, 2007.

- [14] Cline, M. P. 1996. The pros and cons of adopting and applying design patterns in the real world. *Communications of the ACM*, 39(10), 47-49.
- [15] Coplien, J. O. 1996. *Software Patterns*. SIGS Press.
- [16] Cunningham, W. 2006. Portland Pattern Repository. <http://c2.com/ppr/>, Updated: Sept., 2006.
- [17] Deng, J., Kemp, E., Todd, E. G., 2005. Managing UI pattern collections. In *Proceedings of the 6th ACM SIGCHI New Zealand Chapter's Int'l Conf. on Computer-Human Interaction (CHINZ '05)*, 2005, New Zealand, 31-38.
- [18] Dietrich, J., Elgar, C. 2005. Towards a Web of Patterns. In *Proceedings of the Semantic Web Enabled Software Engineering (SWESE)*, 2005, Galway, Ireland, 117-132.
- [19] Eden, A., Yehudai, A., Gil, J. 1997. Precise specification and automatic application of design patterns. In *Proceedings of the Automated Software Engineering Conference*, 143-152.
- [20] Eden, A. H., Hirshfeld, Y. 2001. Principles in formal specification of object oriented architectures. *CASCON '01*.
- [21] Fincher, S., CHI 2003 Workshop Report - Perspectives on HCI patterns: concepts and tools (introducing PLML). In *Interfaces*, 56, 26-28, Available at: <http://www.bcs-hci.org.uk/interfaces.html>.
- [22] Florijn, G., Meijers, M., van Winsen, P. 1997. Tool support for object-oriented patterns. *11th European Conf. on Object Oriented Programming (ECOOP'97)*, Springer-Verlag.
- [23] Fowler, M. 1997. *Analysis Patterns*. Addison-Wesley.
- [24] Gamma, E. 2002. Design Patterns Ten Years Later. In *Software Pioneers: Contributions to Software Engineering*. Springer-Verlag, New York, NY, 688-700.
- [25] Gamma, E., Helm, R., Johnson, R., Vlissides, J. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA.
- [26] Gruber, T. 1995. Towards principles for the design of ontologies used for knowledge sharing. *Int'l Journal of Human-Computer Studies*, 43, 907-928.
- [27] Hahsler, M., 2005. A quantitative study of the adoption of design patterns by open source software developers. In *Free/Open Source Software Development*, S. Koch, Ed., Idea Group Publishing, 103-123.
- [28] Hillside. 2005. Patterns Library, <http://hillside.net/patterns/>.
- [29] Iaria, P., Chenini, 1998. Refining the Observer Pattern: The Middle Observer Pattern. *Pattern Languages of Programming (PloP 98)*, 1998. Available at: http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/.
- [30] Kircher, M., Jain, O. 2004. *Pattern-Oriented Software Architecture, Volume 3: Patterns for Resource Management*. Wiley.
- [31] Klein, M. 2001. XML, RDF, and Relatives. *IEEE Intelligent Systems*, 15(2), 26-28.
- [32] Lamport, L. 1994. The temporal logic of actions. *ACM Trans. Programming Languages and Systems*, 16(3), 872-923.
- [33] McGuinness, D. L., van Harmelen, F. 2004. OWL Web Ontology Language Overview. *W3 Consortium*. Available at: <http://www.w3.org/TR/owl-features/>, Updated: February 10, 2004.
- [34] Mikkonen, T. 1998. Formalizing Design Patterns. *Int'l Conf. Software Engineering*, 115-124.
- [35] Miller, E., Hendler, J. 2007. Web Ontology Language (OWL). *W3 Consortium*. Available at: <http://www.w3.org/2004/OWL/>, Updated: April 24, 2007.
- [36] Noble, J. 1998. Classifying relationships between object-oriented design patterns. *Australian Software Engineering Conference (ASWEC)*, 98-107.
- [37] Noble, J. 1998. Towards a Pattern Language for Object-Oriented Design. In *Proceedings of Technology of Object-Oriented Languages and Systems*, 28, IEEE Computer Society Press, 2-13.
- [38] PLoP, 2005. PatternLanguagesOfPrograms. <http://hillside.net/plop/>.
- [39] Raje, S., Chinnasamy, S. 2001. eLePUS-A Language for Specification of Software Design Patterns. In *Proceedings of the 2001 ACM Symp. Applied Computing*, 600-604.
- [40] Rising, L. 2000. *The Pattern Almanac 2000*. Addison-Wesley.
- [41] Salingaros, N. A. 2000. The Structure of Pattern Languages. *Architectural Research Quarterly*, 4, 149-161.
- [42] Schmidt, D. C., Stal, M., Rohnert, H., Buschmann, F. 2000. *Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects*. Wiley.
- [43] Snow, K., Marks, M., Hong, D., Dennis, T. 2006. Web Patterns Project. *U.C. Berkeley School of Information*. Available at: http://harbinger.sims.berkeley.edu/ui_designpatterns/webpatterns2/webpatterns/home.php.
- [44] Stanford University. 2006. Protégé Project. *Stanford Medical Informatics*, <http://protege.stanford.edu/>.
- [45] Studer, R., Benjamins, V. R., Fensel, D. 1998. Knowledge Engineering: Principles and Methods. *Data and Knowledge Engineering*, 25, 161-197.
- [46] Sun Microsystems Inc. 2002. Core J2EE Patterns: Patterns index page. Available at: <http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html>.
- [47] Taibi, T., Ling Ngo, D. C. 2003. Formal Specification of Design Patterns - A Balanced Approach. *Journal of Object Technology*, 2(4), 127-140.
- [48] Tidwell, J. 2005. UI Patterns and Techniques. <http://time-tripper.com/uipatterns/>, Updated: May, 2005.
- [49] UIUC, 2005. Pattern Stories Wiki. *Univ. of Illinois at Urbana-Champaign*. <http://wiki.cs.uiuc.edu/PatternStories>.
- [50] van Welie, M. 2007. Patterns in Interaction Design. <http://www.welie.com/patterns/index.php>.
- [51] Yahoo! 2007. Yahoo! Design Pattern Library. <http://developer.yahoo.com/ypatterns/>.

APPENDIX A

Title	Source	# of Patt.	Year
Patterns in Interaction Design	http://www.welie.com/	146	2005
"Analysis Patterns: Reusable Object Models"	"Analysis Patterns: Reusable Object Models"	95	1996
"Designing Interfaces: Patterns for Effective Interaction Design"	http://www.mit.edu/~jtidwell/common_ground_onefile.html	94	2005
Ajax Design Patterns	http://ajaxpatterns.org	70	2006
"Requirements Patterns and Antipatterns: Best (and Worst) Practices for Defining Your Requirements"	http://www.tabletuml.com/RPandAP/default.aspx	69	2007
"Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions"	http://www.eaipatterns.com/toc.html	65	2003
Yahoo! Design Pattern Library	http://developer.yahoo.com/ypatterns/	63	2005
"Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects"	"Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects"	55	2004
"J2EE Antipatterns"	"J2EE Antipatterns"	52	2003
"Patterns of Enterprise Application Architecture"	http://www.martinfowler.com/eaCatalog/	51	2002
"Object Oriented Reengineering Patterns"	http://www.iam.unibe.ch/~scg/OORP/book.html	49	2002
A Generative Development-Process Pattern Language	http://users.rcn.com/jcoplien/Patterns/Process/index.html	48	1995
UML Pattern Language	http://www.ncc.up.pt/~zp/aulas/0607/es/geral/bibliografia/UML Pattern Language.pdf	46	2000
"Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems"	Addison Wesley Professional	44	2002
"AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis"	John Wiley & Sons	42	1998
WikiPatterns	http://www.wikipatterns.com/	42	2007
"Patterns for Effective Use Cases"	Addison Wesley Professional	32	2002
"Enterprise Solution Patterns Using Microsoft .NET Version 2.0: Patterns & Practices"	Microsoft Press	32	2004
"Remoting Patterns: Foundations of Enterprise, Internet and Realtime Distributed Object Middleware"	John Wiley & Sons	32	2004
XML Design Patterns	http://www.xmlpatterns.com/	28	2000

Hypermedia Design Patterns Repository	http://www.designpattern.lu.unisi.ch/index.htm	28	1997
Embedded Design Patterns	http://www.eventhelix.com/RealtimeMantra/Patterns/	28	2004
"Small Memory Software: Patterns for Systems with Limited Memory"	http://hillside.net/patterns/books/Details/056.htm	27	2001
A Pattern Language for Pattern Writing	http://hillside.net/patterns/writing/patternwritingpaper.htm	26	1997
Experiences -- A Pattern Language for User Interface Design	http://www.maplefish.com/todd/papers/Experiences.html	26	2003
Data Access Patterns: Database Interactions in Object-Oriented Applications"	http://helloworld.siteburg.com/content/databases/db2/0131401572_toc.html	25	2003
GoF Patterns	http://www.vico.org/pages/PatronsDisseny.html	23	1995
Caterpillar's Fate: A Pattern Language for the Transformation from Analysis to Design	http://c2.com/ppr/catsfate.html	21	1995
User Interface Design Patterns	http://www.cs.helsinki.fi/u/salaakso/patterns/index.html	21	2003
Workflow Patterns	http://www.workflowpatterns.com/patterns/index.php	21	2000
Patterns for System Testing	"Pattern Languages of Program Design 3"	20	1997
Web Design Patterns Library	http://harbinger.sims.berkeley.edu/ui_designpatterns/webpatterns2/webpatterns/home.php	20	2006
A Pattern Language for Writers' Workshops	http://users.rcn.com/jcoplien/Patterns/WritersWorkshop/	19	1999
"Patterns for Parallel Programming"		19	2004
"Microsoft Integration Patterns"	http://download.microsoft.com/download/a/c/f/acf079ca-670e-4942-8a53-e587a0959d75/IntPat.pdf	18	2004
Patterns Systems for Hypermedia	http://www-di.inf.puc-rio.br/schwabe//papers/PlOp97.pdf	18	1997
POSA 1 Patterns	http://www.vico.org/pages/PatronsDisseny.html	17	1996
POSA 2 Patterns	"Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects "	17	2000
RAPPeL: A Requirements-Analysis-Process Pattern Language for Object-Oriented Development	http://www2.umassd.edu/SWPI/ATT/pattern/rapel.html	17	1995
Understanding and Using the ValueModel Framework in VisualWorks Smalltalk	http://c2.com/ppr/vmodels.html	17	1994
An Input and Output Pattern Language: Lessons from Telecommunications	http://hillside.net/plop/plop98/final_submissions/P31.pdf	17	1999
New Clients with Old Servers: A Pattern Language for Client/Server Frameworks	http://citeseer.ist.psu.edu/156837.html	16	1995
Lazy Optimization: Patterns for Efficient Smalltalk	"Pattern Languages of Program Design 2"	16	1996

Programming			
EPISODES: A Pattern Language of Competitive Development	http://c2.com/ppr/episodes.html	16	1996
"Data Model Patterns: Conventions of Thought"	http://www.tdan.com/i005fe03.htm	15	1995
"Core J2EE Patterns: Best Practices and Design Strategies"	http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html	15	2003
Prioritizing Forces in Software Design	"Pattern Languages of Program Design 2"	13	1996
C++ Idioms	http://www.laputan.org/pub/sag/coplien-idioms.pdf	13	1999
Capable, Productive, and Satisfied: Some Organizational Patterns for Protecting Productive People	http://hillside.net/plop/plop98/final_submissions/P54.pdf	11	1999
SCRUM: A Pattern Language for Hyperproductive Software Development	http://citeseer.ist.psu.edu/397129.html	11	1999
"Use Cases: Patterns and Blueprints"	http://www.awprofessional.com/articles/article.asp?p=353171&seqNum=2&rl=1	11	2004
POSA 3 Patterns	"Pattern-Oriented Software Architecture: Patterns for Resource Management"	10	2004
G++: A Pattern Language for Computer-Integrated Manufacturing	http://citeseer.ist.psu.edu/134161.html	10	1995
The CHECKS Pattern Language for Information Integrity	http://c2.com/ppr/checks.html	10	1994
Selecting Locking Designs for Parallel Programming	http://citeseer.ist.psu.edu/493802.html	10	1996
A Pattern Language for Improving the Capacity of Reactive Systems	"Pattern Languages of Program Design 2"	10	1996
Customer Interaction Patterns	http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/P11/P11.htm	10	1999
"Java Testing Patterns"		10	2004
Patterns of Cooperative Interaction	http://www.comp.lancs.ac.uk/computing/research/cseg/projects/pointer/patterns.html	10	2001
Process Patterns	"Process Patterns"	10	1998
A Generative Pattern Language for Distributed Processing	"Pattern Languages of Program Design 1"	9	1995
Patterns for Evolving Frameworks	http://st-www.cs.uiuc.edu/~droberts/evolve.html	9	1997
Tropyc: A Pattern Language for Cryptographic Object-Oriented Software	http://citeseer.ist.psu.edu/62190.html	9	1999
Finite State Machine Patterns	"Pattern Languages of Program Design 4"	9	1999
"Analysis Patterns 2"	http://www.martinfowler.com/ap2/index.html	9	
Evolving Frameworks: A Pattern Language for Developing Object-Oriented	http://st-www.cs.uiuc.edu/users/droberts/evolve.html	9	1996

Frameworks			
Patterns for Software Architectures	http://citeseer.ist.psu.edu/shaw96some.html	8	1995
MOODS: Models for Object-Oriented Design of State	http://www.soberit.hut.fi/tik-76.278/alex/plop95.htm	8	1996
Crossing Chasms: A Pattern Language for Object-RDBMS	"Pattern Languages of Program Design 2"	8	1996
Transactions and Accounts	http://c2.com/cgi-bin/wiki?TransactionsAndAccounts	8	1996
Some Patterns for Software Architecture	http://www.cs.cmu.edu/afs/cs.cmu.edu/project/vit/ftp/pdf/PLoP95.pdf	8	1996
Fault-Tolerant Telecommunications System Patterns	http://users.rcn.com/jcoplien/Patterns/PLoP95_telecom.html	8	1996
Accessing Relational Databases	http://citeseer.ist.psu.edu/90550.html	8	1997
High-Level and Process Patterns from the Memory Preservation Society: Patterns for Managing Limited Memory	http://jerry.cs.uiuc.edu/plop/plopd4-submissions/P54.doc	8	1999
A Collection of History Patterns	hillside.net/plop/plop98/final_submissions/P63.pdf	8	1999
Display Maintenance: A Pattern Language	hillside.net/plop/plop98/final_submissions/P15.pdf	8	1999
More Process Patterns	"More Process Patterns"	8	1999
A Pattern Language for Tool Construction and Integration Based on the Tools and Materials Metaphor	http://www.riehle.org/computer-science/research/1994/plop-1994-tools.pdf	7	1995
Stars: A Pattern Language for Query-Optimized Schemas	http://c2.com/ppr/stars.html	7	1994
Reusability Through Self-Encapsulation	"Pattern Languages of Program Design 1"	7	1995
Partitioning Smalltalk Code into ENVY/Developer Components	http://c2.com/ppr/envy/	7	1996
State Patterns	http://citeseer.ist.psu.edu/396622.html	7	1997
The Selfish Class	http://www.joeyoder.com/papers/patterns/Selfish/selfish.html	7	1997
Architectural Patterns for Enabling Application Security	http://st-www.cs.uiuc.edu/~hanmer/PLoP-97/Proceedings/yoder.pdf	7	1999
Big Ball of Mud	http://www.laputan.org/mud/	7	1999
The Diemen Repository of Interaction Design Patterns	http://www.visiblearea.com/cgi-bin/twiki/view/Patterns/Patterns_repository	7	2003
Implementation Patterns for the Observer Pattern	"Pattern Languages of Program Design 2"	6	1996
Accountability and Organizational Structures	"Pattern Languages of Program Design 2"	6	1996
Smalltalk Scaffolding Patterns	"Pattern Languages of Program Design 4"	6	1999
Parallel Patterns for Synchronization on Shared-	http://c2.com/ppr/mutex/mutexpat.html	6	1995

Memory Multiprocessors			
Lifecycle and Refactoring Patterns That Support Evolution and Reuse	http://www.laputan.org/Lifecycle.html	5	1995
Discovering Patterns in Existing Applications	"Pattern Languages of Program Design 1"	5	1995
Patterns for Encapsulating Class Trees	http://citeseer.ist.psu.edu/riehle95patterns.html	5	1996
Decision Deferral and Capture Pattern Languages	"Pattern Languages of Program Design 2"	5	1996
Organizational Patterns for Teams	"Pattern Languages of Program Design 2"	5	1996
Object-Oriented Design Patterns in Reactive Systems	http://citeseer.ist.psu.edu/426489.html	5	1996
A Pattern Language for Developing Form-Style Windows	"Pattern Languages of Program Design 3"	5	1997
The Points and Deviations Pattern Language of Fire Alarm Systems	http://www.cs.wustl.edu/~schmidt/PLoP-96/molin.ps.gz	5	1997
Patterns for Designing in Teams	http://www.charlesweir.com/papers/teamwork.pdf	5	1997
Basic Relationship Patterns	http://citeseer.ist.psu.edu/38872.html	5	1999
Creating Reports with Query Objects	http://www.joeyoder.com/papers/patterns/Reports/	5	1999
Patterns for Designing Navigable Information Spaces	http://www.inf.puc-rio.br/~schwabe/papers/PLoP98.pdf	5	1999
Composing Multimedia Artifacts for Reuse	http://hillside.net/plop/plop98/final_submissions/P38.pdf	5	1999
Patterns for Designing Navigable Information Spaces	http://www-di.inf.puc-rio.br/schwabe//papers/PLoP98.pdf	5	1998
Patterns for Adding Search Capabilities to Web Information Systems	http://www-di.inf.puc-rio.br/schwabe//papers/Europlop99.pdf	5	1999
Patterns for E-commerce Applications	http://www-di.inf.puc-rio.br/schwabe/papers/Europlop00.pdf	5	2000
The Risk Management Catalog	http://members.aol.com/acockburn/riskcata/risktoc.htm	5	1996
Patterns for Generating a Layered Architecture	"Pattern Languages of Program Design 1"	4	1995
Pattern-Based Integration Architectures	"Pattern Languages of Program Design 1"	4	1995
Patterns of Events	"Pattern Languages of Program Design 1"	4	1995
Organizational Multiplexing: Patterns for Processing Satellite Telemetry with Distributed Teams	http://citeseer.ist.psu.edu/berczuk96organizational.html	4	1996
Improve Responsiveness in Interactive Applications Using Queues	"Pattern Languages of Program Design 2"	4	1996
Bridging Patterns: An	Information and Software Technology, 48, pp 69-89	4	2005

approach to bridge gaps between SE and HCI			
Localized Ownership: Managing Dynamic Objects in C++	"Pattern Languages of Program Design 2"	3	1996
Evolution, Architecture, and Metamorphosis	http://www.laputan.org/metamorphosis/metamorphosis.html	3	1996
Patterns for Logging Diagnostic Messages	http://www.cs.wustl.edu/~schmidt/PLoP-96/harrison.ps.gz	3	1997
Business Patterns of Association Objects	http://www.riehle.org/computer-science/programming/patterns/association-objects/index.html	3	1997
Temporal Patterns	http://www.hillside.net/plop/plop98/final_submissions/P09.pdf	3	1999
Design Patterns for Object-Oriented Hypermedia Systems	http://www.cs.colorado.edu/~kena/classes/7818/f99/patterns.pdf	2	1996
Default and Extrinsic Visitor	"Pattern Languages of Program Design 3"	2	1997
A Pattern Language of Transport Systems (Point and Route)	www.cs.wustl.edu/~schmidt/PLoP-96/zhao.ps.gz	2	1997
Functionality Ala Carte	"Pattern Languages of Program Design 1"	1	1995
Flexible Command Interpreter: A Pattern for an Extensible and Language-Independent Interpreter System	"Pattern Languages of Program Design 1"	1	1995
Half-object + Protocol [HOPP]	"Pattern Languages of Program Design 1"	1	1995
The Master-Slave Pattern	http://www.vico.org/pages/PatronsDisseny/Pattern%20Master%20Slave/index.html	1	1995
Account Number: A Pattern	http://citeseer.ist.psu.edu/wake95account.html	1	1995
A Systems of Patterns	"Pattern Languages of Program Design 1"	1	1995
Implementing Patterns	http://www.codefarms.com/publications/papers/patterns.html	1	1995
Streams: A Pattern for "Pull-Driven" Processing	"Pattern Languages of Program Design 1"	1	1995
The Pipes and Filters Architecture	http://www.vico.org/pages/PatronsDisseny.html	1	1995
Client-Specified Self	"Pattern Languages of Program Design 1"	1	1995
A Pattern for Separating Assembly and Processing	http://citeseer.ist.psu.edu/berczuk95pattern.html	1	1995
Reactor: An Object Behavioral Pattern for Concurrent Event Demultiplexing and Event Handler Dispatching	http://citeseer.ist.psu.edu/schmidt95reactor.html	1	1995
Command Processor	http://vico.org/pages/PatronsDisseny/Pattern%20Command%20Processor/index.html	1	1996
The Proxy Design Pattern Revisited	"Pattern Languages of Program Design 2"	1	1996
Shopper	"Pattern Languages of Program Design 2"	1	1996
Detachable Inspector/Removable cout: A Structural Pattern for Designing Transparent	http://citeseer.ist.psu.edu/201036.html	1	1996

Layered Services			
Backup Pattern: Designing Redundancy in Object-Oriented Software	"Pattern Languages of Program Design 2"	1	1996
Reflection	http://www.vico.org/pages/PatronsDisseny/Pattern%20Reflection/index.html	1	1996
Half-Sync/Half-Async: An Architectural Pattern for Efficient and Well-Structured Concurrent I/O	http://www.cs.wustl.edu/~schmidt/PDF/PLoP-95.pdf	1	1996
Resource Exchange: A Behavioral Pattern for Low-Overhead Concurrent Resource Management	"Pattern Languages of Program Design 2"	1	1996
The Client-Dispatcher-Server Design Pattern	http://www.vico.org/pages/PatronsDisseny/Pattern%20ClientDispatcherServer/index.htm	1	1996
Active Object: An Object Behavioral Patterns for Concurrent Programming	http://citeseer.ist.psu.edu/lavender96active.html	1	1996
Null Object	http://www.cs.oberlin.edu/~jwalker/nullObjPattern/	1	1997
Manager	http://www.cs.wustl.edu/~schmidt/PLoP-96/sommerlad.ps.gz	1	1997
Product Trader	http://www.riehle.org/computer-science/research/1996/plop-1996-product-trader.pdf	1	1997
Type Object	http://citeseer.ist.psu.edu/133930.html	1	1997
Sponsor-Selector	http://cns2.uni.edu/~wallingf/patterns/sponsor-selector.html	1	1997
Extension Object	http://citeseer.ist.psu.edu/160815.html	1	1997
Acyclic Visitor	http://www.objectmentor.com/resources/articles/acv.pdf	1	1997
Recursive Control	http://citeseer.ist.psu.edu/181638.html	1	1997
Bureaucracy	http://www.riehle.org/computer-science/research/1996/europlop-1996-bureaucracy.html	1	1997
Acceptor and Connector	http://www.cs.wustl.edu/~schmidt/PDF/Acc-Con.pdf	1	1997
Bodyguard	http://ei.cs.vt.edu/~cs6704/bodyguard.ps	1	1997
Asynchronous Completion Token	http://www.cs.wustl.edu/~schmidt/PDF/ACT.pdf	1	1997
Object Recovery	http://www.cs.wustl.edu/~schmidt/PLoP-96/silva.ps.gz	1	1997
Serializer	http://www.riehle.org/computer-science/research/1996/plop-1996-serializer.pdf	1	1997
Double-Checked Locking	http://www.cs.wustl.edu/~schmidt/PLoP-96/DC-Locking.ps.gz	1	1997
External Polymorphism	http://citeseer.ist.psu.edu/181874.html	1	1997
Abstract Class	http://st-www.cs.uiuc.edu/~hanmer/PLoP-97/Proceedings/woolf.pdf	1	1999
Role Object	http://st-www.cs.uiuc.edu/~hanmer/PLoP-97/Proceedings/riehle.pdf	1	1999
Essence	hillside.net/plop/plop98/final_submissions/P10.pdf	1	1999
Object Recursion	http://www.industriallogic.com/patterns/P21.pdf	1	1999
Prototype-Based Object System	"Pattern Languages of Program Design 4"	1	1999
Abstract Session: An Object Structured Pattern	http://www.doc.ic.ac.uk/~np2/patterns/session.ps.gz	1	1999
Object Synchronizer	http://citeseer.ist.psu.edu/216930.html	1	1999
Proactor	http://www.cs.wustl.edu/~schmidt/PDF/proactor.pdf	1	1999
Feature Extraction: A Pattern for Information Retrieval	http://micro-workflow.com/PDF/plop98.pdf	1	1999

Identify the Champion: An Organizational Pattern Language for Program Committees	http://hillside.net/plop/plop98/final_submissions/P07.pdf	1	1999
--	---	---	------