December 2002

SOFTWARE REUSE WITH ANALYSIS PATTERNS

Andreas Geyer-Schulz
Universität Karlsruhe (TH)

Michael Hahsler
Wirtschaftsuniversität Wien

SOFTWARE REUSE WITH ANALYSIS PATTERNS

Andreas Geyer-Schulz
Universität Karlsruhe (TH)
andreas.geyer-schulz@em.uni-karlsruhe.de

Michael Hahsler
Wirtschaftsuniversität Wien
michael.hahsler@wu-wien.ac.at

Abstract

The purpose of this article is to promote reuse of domain knowledge by introducing patterns already in the analysis phase of the software life-cycle. We propose an outline template for analysis patterns that strongly supports the whole analysis process from the requirements analysis to the analysis model and further on to its transformation into a flexible and reusable design and implementation. As an example we develop a family of analysis patterns in this paper that deal with a series of pressing problems in cooperative work, collaborative information filtering and sharing, and knowledge management. We evaluate the reuse potential of these patterns by analyzing several components of an information system, that was developed for the Virtual University project of the Vienna University of Economics and Business Administration. The findings of this analysis suggest that using patterns in the analysis phase has the potential to reducing development time significantly by introducing reuse already at the analysis stage and by improving the interface between analysis and design phase.

Introduction

The Internet still grows exponentially. And with the growth of the Internet also the need for information systems that help to store, organize and present information exploded. The rapid change of technology in today's networked world and the current transition to the information society presents a great challenge for system developers in terms of ever shorter time for development of high quality custom information systems. Implementing every system from scratch does not scale up to the demand and, equally important, does not provide the required quality (see (Gillies 1992) for details on software quality). Reusing code in the form of libraries and components, and reusing design through frameworks (Fach 2001) and design patterns (Gamma et al. 1995) already became a widely accepted strategy to cope with these challenges. Recently, research in software reuse moved on to focusing on domain knowledge (Moore 2001) known from domain analysis (Neighbors 1984). Discovering common elements across a domain provides the possibility to reuse non-code artifacts such as specifications, use-cases, analysis models and design, and thus enables us to produce more reliable and flexible systems at significantly reduced time-to-market. In this article we present and evaluate analysis patterns as a means to facilitate domain knowledge reuse.

The paper is structured as follows: First we give a very brief overview over analysis patterns and we propose a suitable template structure to describe analysis patterns. Then we present examples for analysis patterns for cooperative work and information sharing which we have developed in a series of virtual university projects from 1997 to 2001. Finally, we analyze two information systems that are based on an implementation of the analysis patterns presented in this paper. For these examples we analyze code reuse resulting from reused domain knowledge and we estimate the reduction in effort and development time using the well-known *Constructive Cost Model* by Boehm (1981).

Analysis Patterns

The term analysis pattern has been coined by Martin Fowler (1997) for patterns which capture conceptual models in an application domain in order to allow reuse across applications. Analysis Patterns, in contrast to Design Patterns, focus on organizational, social and economic aspects of a system, since these aspects are central for the requirements analysis and the acceptance and usability of the final system.

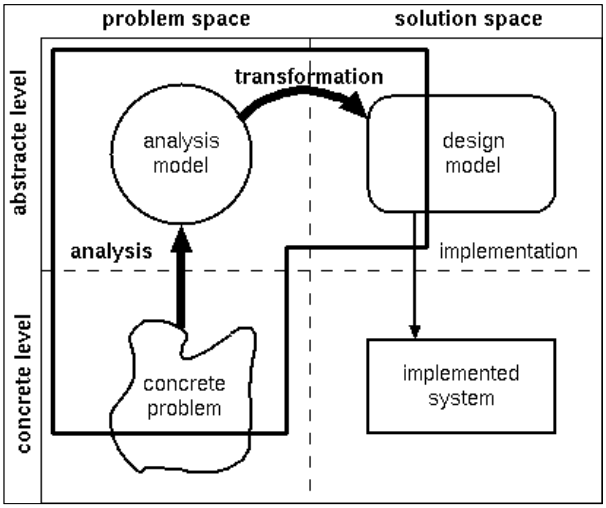

Figure 1. Analysis Patterns in the Software Development Process

Figure 1 shows the two main tasks where analysis patterns contribute to the software development process. First, analysis patterns speed up the development of abstract analysis models that capture the main requirements of the concrete problem by providing reusable analysis models with examples as well as a description of their advantages and limittions. Second, analysis patterns facilitate the transformation of the analysis model into a design model. This transformation from the problem space to the solution space is a complicated and time-consuming process as described by Champeaux et al. (1992) and Kaindl (1999). Analysis Patterns support this process by suggesting design patterns and reliable solutions for recurring constructs in the problem space.

In contrast to Fowler (1997) who uses a very free and informal format for pattern writing, we adopt a uniform and consistent format for describing analysis patterns. Vlissides (1998) and other authors stress that adhering to a structure for writing patterns is essential since structured information is easier to teach, learn, compare, write, and use, once the structure has been understood. In table 1 we show the template which we propose for writing analysis patterns. It is used for the 4 example patterns in the following sections.

Note, however, that our proposal preserves the typical context/problem/solution structure of patterns. With three exceptions (Forces, Solution, Design) the used sections originate from (Gamma et al. 1995), albeit with a slight shift in meaning as required in the analysis phase and illustrated in table 1. The section on Forces is in the spirit of Alexander's patterns for architecture (Alexander 1979), to which the usage of patterns in software engineering can be traced back. The section is used to discuss the forces and tensions which should be resolved by the pattern including social and economic conflicts.

Table 1. Template for Analysis Patterns

Pattern Name (Gamma et al. 1995; Buschmann et al.1996) A pattern's name expresses the essence of a patterns precisely. It becomes part of the vocabulary used in analysis.
Intent (Gamma et al. 1995) What does the analysis pattern do and what problem does it address?
Motivation (Gamma et al. 1995) A scenario that illustrates the problem and how the analysis pattern contributes to the solution in the concrete scenario.
Forces and Context (Alexander 1979) Discussion of forces and tensions which should be resolved by the analysis pattern.
Solution (Buschmann et al. 1996) Description of solution and of the balance of forces achieved by the analysis pattern for the scenario in the motivation section. Includes all relevant structural and behavioral aspects of the analysis pattern.
Consequences (Gamma et al. 1995; Buschmann et al. 1996) How does the pattern achieve its objectives and what trade-off exist?
Design [New] How can the analysis pattern be realized by design patterns? Sample design suggestions.
Known Uses (Gamma et al. 1995; Buschmann et al. 1996) Examples of the pattern found in real systems.

The solution part of a pattern is from (Buschmann et al. 1996). It shows how to solve the problem and how a balance of forces is achieved by the pattern. It contains all diagrams which graphically describe the relevant structural and behavioral aspects of the pattern. We use the notation of the Unified Modeling Language (UML).

The design part of an analysis pattern contains a description of a possible realization of the analysis pattern with one or several design patterns. The motivation for this section is to reduce software development time by providing design examples for the analysis pattern. In the best case, the design phase can be completely eliminated by reusing such a proposed design solution.

A key challenge in the use of patterns in general is the requirement that prospective authors and users of patterns share a common vocabulary which is a prerequisite to achieve efficient communication and establish mutual understanding about the purpose of analysis patterns. While an investigation of this problem is outside the scope of this paper, we address the problem as follows: First, we establish a common vocabulary and a mutual understanding by intensive project meetings of the development team. In this we rely on group dynamics to weed out misunderstandings of what an analysis pattern really means. Second, as indicated by our use of UML as notation we too revert to a certain degree to more formal specification languages. However, a word of caution is necessary here: This requires that these formal specification languages are understood by authors and users. This requirement can not taken to be granted, because knowledge diffusion takes considerably more time than expected.

Examples: The Evolution from a Simple Pinboard to a Virtual Library with Active Agents

Next, we describe the rationale of the evolution from a simple pinboard to a virtual library. The driving forces of this evolution process are transaction cost reduction, scalability and performance improvements.

First, consider a *simple pinboard*. Simple pinboards constitute an efficient informal communication channel for small groups as long as the number of messages posted remains small. As soon as either the number of group members or the number of messages grows, group members need more and more time to keep track of the pinboard, searching specific information soon becomes an expensive, time-consuming task.

The evolution from a simple pinboard to a *structured pinboard* addresses these scalability problem by providing an environment for structured messages basically by extending the presentation interface and the message representation. These changes allow users to focus their attention only on small parts of the pinboard entries and improve the search capabilities considerably. However, there is a price to pay: properly using an information structure requires that either the user understands the semantic of the fields of the form he has to fill or that a professional librarian must classify the messages. In short, the cost, of the administration of such a pinboard rise as it grows and time goes by.

Moreover, frequently it is of advantage to have a pinboard for distributed information objects, e.g. because the owner of an information object prefers personal control or because the owner changes the information object often or because a distributed object-store provides better load-balancing under heavy network traffic and a better performance with lower network-bandwidth. All of these factors reduce the cost of an organization's network and computing infra-structure. Clearly, the next evolution step is motivated by these issues. The *virtual library* allows for distributed information objects with weak consistency checks.

Analysis Pattern: A Simple Pinboard

Intent
How can a dispersed group share information efficiently?

Motivation
You are working for a large multinational company. This company has work-groups with its members dispersed over various countries. This groups have to work together to fulfill their jobs, therefore efficient communication and information sharing is a vital necessity for them.

It is not possible that all members of a work-group meet frequently, since this would be too time consuming. A work-group needs to exchange messages, communicate asynchronously and build up a common base of knowledge. Furthermore, to prevent information overload, the members of a work-group need an easy way to retrieve the needed information for a specific task. Unfortunately, existing communication media like letters, phone, fax and e-mail can not comply with all this needs.

A pinboard-like system (figure 2) is used for the work-group. All members can read and compose messages. So every group member can use the messages when he needs them.

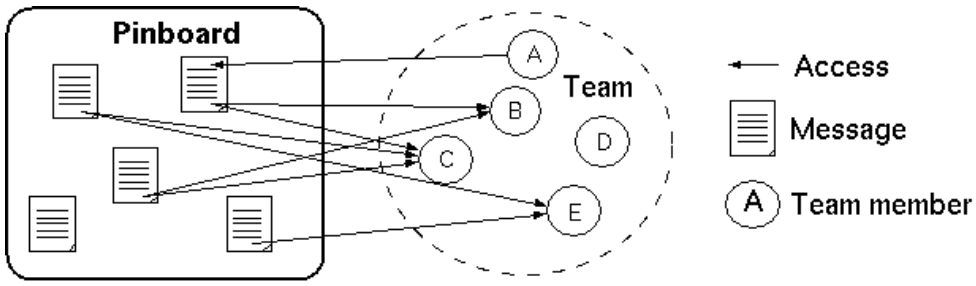

Figure 2. Simple Pinboard for Intra-Team Communication

Forces
- Efficient group communication is vital.
- Frequent meetings in person are not feasible.
- The risk of information overload is present.

Solution
Use a pinboard for group communication. It stores messages from all group members and makes them available. For simple pinboards a message consists only of a textual body and the name of its author. To let members of the work-group choose the needed messages, full-text search over all messages can be provided (e.g. search for messages posted from a specific group member or which contains a special keyword).

Figure 3 shows that the pinboard consists of three main components, the database for storing and retrieving messages, the interface providing access to users, and the control unit responsible for managing the whole pinboard.

The following actors use the system:

1. *User:* Retrieves messages from the pinboard.
2. *Information Provider:* Adds new messages to the pinboard.
3. *Administrator:* Is responsible for operating the Pinboard (e.g. deleting old messages from the database).

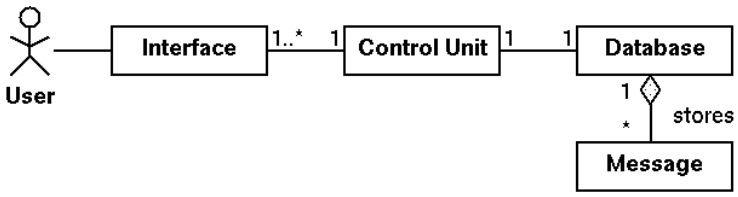

Figure 3. Structure of a Simple Pinboard

Consequences
The benefits and liabilities of the pinboard pattern are:

1. *Reduction of information overload.* The user actively searches for the information he needs. A problem occurs, however, if a messages has to be delivered quickly to a specific person. In this case alternative means of communication (e.g. electronic mail) should be used.
2. *Asynchronous communication.* Work is not permanently interrupted by incoming electronic mails or phone calls.
3. *Automatic generation of a collaborative information source.* Messages can be collected and archived to build a *collective dynabase* (Press 1992).
4. *Confidential information.* If confidential information is shared via a pinboard, sufficient security precautions have to be taken.
5. *Lifetime of messages.* When is a message out of date and who deletes such a message? A pragmatic solution could be to automatically archive messages older than a defined age.

Design
Although a pinboard seems to be easily implemented, several issues have to be considered in order to ensure reusability and extendibility:

1. *Separation between the three components of the system.* For such a simple system like the pinboard it is tempting to implement all parts in one module. This reduces implementation time since no interfaces between the components have to be specified, but it also makes later changes very expensive. For example, changing the interface from a proprietary client to a Web-based interface is very easy, if only the interface component has to be adapted.
2. *Extendibility of the control unit.* To ensure that new functions can easily be added to the control unit, the best choice is to implement it using the *Interpreter* pattern (Gamma et al. 1995) with an extendable instruction set.
3. *Choice of the database.* There are several possibilities to store messages, ranging from database management systems (object-oriented or relational) to simply using the file system. The decision should be based on search performance, license cost and implementation effort. However, if the database component is separated from the rest of the system, later change using the *Facade* pattern (Gamma et al. 1995) is possible.
4. *Choice of the interface.* Here we have to choose between an interface for a proprietary client or a standard client like a Web-browser. The enormous advantage of Web technology is the availability of clients. It is common that they are distributed and preinstalled with every new operating system. The only reason for a proprietary client could be security considerations, but the built-in security features (like Netscape's Secure Socket Layer) are improving constantly and even a proprietary interface can be implemented e.g. using a Web-browser with a client side Java applet.

Known Uses
- News Network Transfer Protocol (Kantor and Lapsley 1986).
- Bulletin Boards (e.g. for groupware applications like the BSCW system (Bentley et al. 1997)
- Guest books of Web-sites.

Analysis Pattern: Structured Pinboard

Intent
Collect, group and present structured information.

Motivation
For your work-group collecting simple text messages is not sufficient to find the needed information within reasonable time. This is due to the size of the group and thus the number of messages available as well as to the variety of different topics the group has to deal with.

It is important, that a user can decide quickly if the message contains what he is searching for. Also grouping by specific criteria (e.g. by several categories) makes finding related information easier. The simple Pinboard only supports unstructured text, but different people use different terms to describe the same concept. Therefore all problems of information retrieval are present. We use a pinboard that supports structured messages to address these problems. Additionally to the textual body you can use separate searchable fields for the subject, keywords and several predefined categories (e.g. the information's language, or a special classification scheme). To compose messages, predefined forms are used.

Forces
- Big groups with a great number of messages and/or topics make a simple pinboard confusing.
- It takes more time to read unstructured text.
- Unstructured text is hard to read and very hard to process automatically.
- People tend to forget important aspects when they compose a new message.

Solution
Build a pinboard with structured messages. With such a system that provides several semi-structured message types, *intelligent information sharing* is possible (Malone et al. 1987).

Figure 4 shows, that structured messages are obtained by composing the Message-objects of several Field-objects. To provide a more flexible interface, the Interface delegates presentation issues to several subclasses using the Strategy or Template pattern (Gamma et al. 1995). While Interface A in figure 4 could be a simple search interface, Interface B could provide access via e.g. a hierarchical classification tree, like NAICS (NTIS 1997) provides for industries.

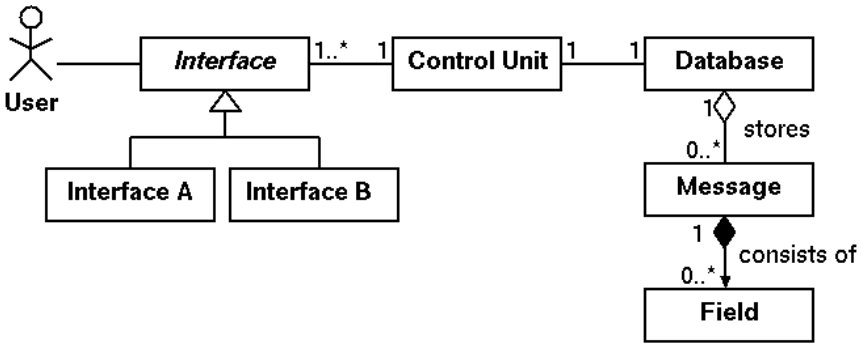

Figure 4. Structure of a Structured Pinboard

Consequences
The benefits and liabilities of the Structured Pinboard pattern are:

1. *Structure.* Understanding information with a logical structure is easier and takes less time than reading unstructured text.
2. *Automatic processing.* It is possible to use fields whose content is restricted to a predefined set of values. With such fields automatic processing can be employed without complex information recognition techniques.
3. *Completeness of information.* Presenting the user a form with all mandatory fields marked, prevents him from omitting essential information.
4. *Incompatibility between different types of messages.* Using a fixed form to compose messages has the drawback that all possible fields have to be predefined. No new fields, that would improve the readability of the message, can be added by the user while writing it. And if several different forms with different logical structures are provided, one has to find a strategy how to deal with polymorph message types.

Design
Additional to the issues presented for the simple Pinboard the following points are important:

1. *Development of the message form.* The implementation of the form requires a detailed analysis of the information needed. The form has to be usable intuitively (without looking into the manual), otherwise the users will loose time thinking about which information should go into which field. If some fields could be easily misunderstood, their purpose has to be explained in the form.
2. *Extension of the Interface.* With structured messages it is possible to improve the interface considerably. E.g. the message can be presented by the subject and the author only, therefore more messages can be displayed on one screen. Another improvement is possible by providing an interface that lets you search only in a specific field or lets you restrict the search to messages that contain a specific value in one field (e.g. the field containing the date of a message could be restricted to dates not older then one week while searching for a keyword).

Known Uses
- The simple Pinboard pattern is a special case of the Structured Pinboard with only one field, the textual body. Therefore all applications for simple Pinboards can also be realized using a Structured Pinboard.
- The Information Lens system (Malone et al. 1987).
- Broker services on the Internet (e.g. on-line job offers of an employment agency).

Analysis Pattern: Virtual Library

Intent
A Virtual Library is used to provide easy access to distributed information sources.

Motivation
At a university much research and teaching material is available on-line, but most of it is hard to find since it is dispersed all over the campus network. Typically some material is accessible from departments' home pages or from the lecturers' personal home

pages. Unfortunately, often material is only available from pages deep in the Web-space or it is not linked at all. Even though most universities provide search engines for their Web-site, presenting all material in an organized way would improve accessibility and thus increase its usage and value.

A common approach to cope with the problem of dispersed information is to centralize it, for example, in a digital library. A digital library is a depository for storing and retrieving documents. However, there are several reasons why the introduction of a digital library can be problematic or even fail.

- Documents that tend to change frequently (topical lecture material) produce additional administrative overhead for updating the document in the central storage every time it is changed.
- Multimedia supported documents are hard to store in a digital library while preserving their functionality (e.g. lecture material that uses server side programs to visualize a model manipulated by the user).
- In organizations with many virtually independent units it is hard to convince all that a central storage of information is favorable for every member of the organization and not just a means of centralizing power.

We use a central inventory as shown in Figure 5 that provides uniform access to material and information resources available on the campus network (or the Internet). The inventory consists of electronic index cards which additionally to meta-information (e.g. the information object's name, the author, keywords, a description ... see (Weibel et al. 1998) for standard metadata for the Web) also contain a reference (e.g. a hyperlink) to the real information object they represent.

Use separate fields of the Structured Pinboard's message class to store the references and its associated meta-information. An important issue is that references are only valid as long as the referenced information objects are available. So it is necessary to implement a mechanism to ensure that all references are checked frequently for consistency.

Forces

- Need for a central access point to dispersed information.
- Independent information provider with different objectives and using different technologies.
- High change rate of the provided information.

Solution

The virtual library uses the structure of the structured pinboard. The pinboard's messages are used to store meta-information about the information objects, as well as a reference to them.

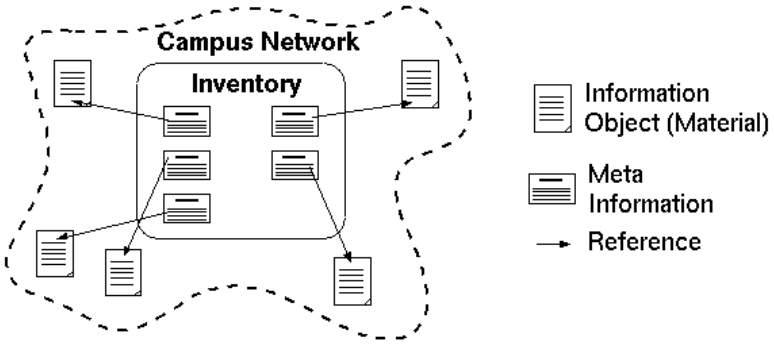

Figure 5. A Virtual Library

Consequences

In addition to the benefits and liabilities known from the structured pinboard pattern, the virtual library pattern has the following consequences:

1. *Information objects physically stay with their owners.* Since only meta information with a reference is stored centrally, the owner of the real object can manipulate it without any restriction. This is of enormous advantage, if frequent updates of the objects are necessary.
2. *Consistency problems.* Information objects can be deleted or moved to other locations without notifying the central directory. Therefore all references and the meta information have to be checked frequently in order to preserve system consistency. Moreover, repairing invalid references can be a very expensive task.

Design

The Virtual Library uses a Structured Pinboard which has an extendable control unit. Therefore, adding new functions (e.g. consistency check, other administrative function) is possible without changes. A consistency check of references to information objects as well as additional functions (e.g. automatic keyword extraction) can be implemented, as shown in Figure 6, using the *Visitor pattern* (Gamma et al. 1995).

Known Uses

- Directory based services (e.g. the WWW Virtual Library, Yahoo!).
- The Virtual Library system developed for the Living Lectures-Virtual University Project (http://vu.wu-wien.ac.at/virlib/).

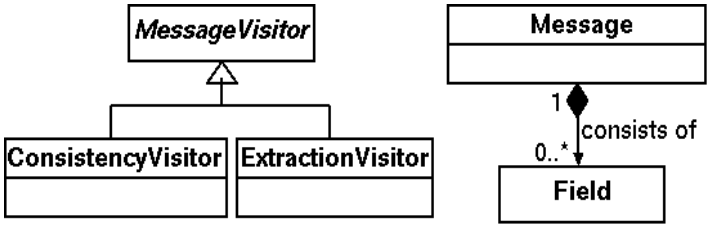

Figure 6. Structure of Additional Functions for a Virtual Library

Benefits of Analysis Patterns

We used the analysis patterns described in this paper to realize several information systems for the *Virtual University project at the Vienna University of Economics and Business Administration* (see: http://vu.wu-wien.ac.at). First, we implemented the analysis pattern *virtual library* in the programming languages PERL and HTML, and then we reused the resulting software to build several information systems for different tasks within the virtual university project.

To quantify the benefit of the used analysis patterns we compare the actual effort needed to implement two information systems for the project reusing the implementation of the patterns with an estimate of the effort necessary to build these systems from scratch. For the estimate we use the well-known *Constructive Cost Model (COCOMO)* (Boehm 1981; Boehm et al. 2000) to convert *Lines of Code (LOC)* into effort measured in man months (MM). We use the parameters for the basic model of COCOMO suggested by Boehm in the *Organic Mode of Software Development* since the analyzed information systems were developed by small teams and the technical specifications were not fully fixed at the beginning of the development. This provides us with the simple formula shown in equation 1, where 2.4 and 1.05 are the parameters of the model, *KDSI* is the number of delivered source instructions (a similar measure like LOC) in 1000, and *MM* is the estimated amount of man months needed to develop the system.

$$MM = 2.4 \cdot KDSI^{1.05} \quad (1)$$

The analyzed information systems of the project, their size, the size of the development teams and the actual effort needed to implement them are shown in table 2. For the calendar of events we changed the interface of the virtual library, the structure of its meta-information and we added an automatic archiving function for events that already took place. For the digital library we had to extend the virtual library with a repository for documents, implement support for polymorph meta-information and we had to change and add many administrative functions (e.g. upload of documents, management of documents in several formats, full text search in the documents).

Table 2. The Analyzed Projects

Project	Size in LOC	Team Size	Actual Effort in MM
Calendar of Events	4021	1	0,5
Digital Library	7616	2	6

Table 3. Code Reuse for the Calendar of Events

	Unit	Total Code	Reused Code	Code Reuse
PERL	LOC	3743	3502	93,56
HTML	LOC	278	192	69,06
Sum	LOC	4021	3694	91,67
Sum	MM	10,35	9,46	91,40

The calendar of events. The on-line calendar of events is a software for the collaborative collection of Web-sites for events, which perfectly is covered by the analysis pattern *virtual library.* Therefore, we could reuse most of the implementation of the pattern. In table 3 the percentage of the code reuse is summarized. More than 93% of the LOC in PERL and 69% of the LOC in HTML (with embedded PERL statements) could be reused unchanged. By using COCOMO to estimate man months from LOC,

this reuse resulted in a reduction of the total development effort of more than 9 man months or over 90%.

The development of a digital library. For the development of the digital library based on the implementation of the analysis pattern *virtual library,* several changes were necessary. Virtual libraries only manage meta-information and references to documents. Therefore, the design of the virtual library had to be augmented significantly by components necessary to store and manage the actual documents. Figure 7 shows the main changes.

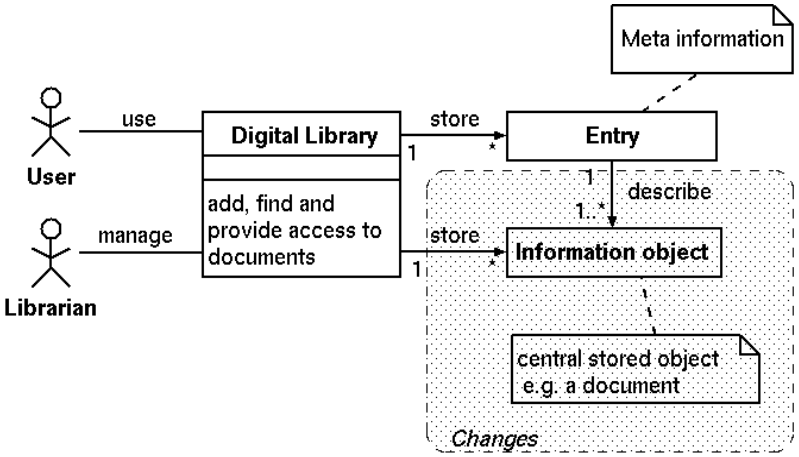

Figure 7. Changes from a Virtual Library to a Digital Library

But not only the design changed, also the analysis presented in the pattern *virtual library* only applies partial to a digital library. Virtual libraries are used to provide access to distributed information sources, in contrast, digital libraries provide a central storage for information in form of documents. This leads to a different set of forces and consequences for a digital library. In fact, digital library should become its own analysis pattern.

Table 4. Code Reuse for the Development of a Digital Library

	Unit	Total Code	Reused Code	Code Reuse
PERL	LOC	6954	3687	53,02
HTML	LOC	662	224	33,84
Sum	LOC	7616	3911	51,35
Sum	MM	20,23	10,05	49,68

However, analysis patterns, as all patterns, are supposed to lead to clearer, and more flexible and reusable design and code. Thus, the implementation of the virtual library should be easy to change and extend to build a digital library. Table 4 summarizes the results of the code reuse for building a digital library from the virtual library. More than 53% of the LOC in PERL and more than 33% of the LOC in HTML could be reused unchanged. This resulted in a total code reuse of over 51% or an estimated reduction of effort of 10 man month, which is about 50% of the total development effort.

Only a small part of the potential benefits of analysis patterns could be shown by the simple analysis of code reuse presented in this section. Analysis patterns provide a tool to save time and effort along the whole software development process, which only can be correctly measured by comparing the overall costs due to all development phases including changes after the system is deployed with and without using patterns. However, this first evaluation of code reuse provides strong evidence, that patterns already used in the analysis phase, accompanied by design patterns have the potential to significantly reduce time-to-market as well as decrease costs in the long run through more robust and flexible systems.

Conclusion

Productive software engineering without reuse in the form of libraries, components, frameworks, design patterns, and other similar methods is unthinkable today. However, recently the much broader approach of reusing domain knowledge, as the sum of the information used in the whole software development process from specification to code, attracts attention.

In this paper we introduced analysis patterns as a way to document domain knowledge in a structured form and therefore make the knowledge reusable. With several examples we showed how domain knowledge in the form of analysis patterns can facilitate reuse in all major phases of the software development process. To provide evidence that analysis patterns can improve productivity, we analyzed code reuse for two applications of the analysis pattern *virtual library,* presented in this paper. Even though the evaluated applications are quite different (a calendar of events and a digital library) the applied pattern provides a significant code reuse potential (between 49 and 91%).

The results of this paper are quite promising. However, further and more detailed research on the benefits of patterns as a means to document and reuse domain knowledge is needed. Some important open research questions include:

- How analysis patterns can help to provide a common vocabulary for system developers and users?
- How analysis patterns can help to classify and organize documented domain knowledge to make it easily accessible for reuse?
- How analysis patterns can help to teach effective analysis strategies?
- How analysis patterns can help to describe and understand large systems?

References

Alexander, C. *The Timeless Way of Building*, Volume 1 of *Center for Environmental Structure Series*. Oxford University Press, New York, 1979.

Bentley, R., Appelt, W., Busbach, U., Hinrichs, E., Kerr, D., Sikkel, S., Trevor, J., and Woetzel, G. "Basic Support for Cooperative Work on the World Wide Web," *International Journal of Human-Computer Studies* (46:6), June 1997, pp. 827-846.

Boehm, B.W., Abts, C., Brown, A.W., Chulani, S., Clark, B.K., Horowitz, E., Madachy, R., Reifer, D.J., and Steece, B. *Software Cost Estimation with COCOMO II*, Prentice Hall PTR, Upper Saddle River, NJ, 2000.

Boehm, B.W. *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. *Pattern-Oriented Software Architecture, A System of Patterns*, John Wiley & Sons Ltd, Chichester, 1996.

Champeaux, D., Lea, D., and Faure, P. "The process of object-oriented design," *ACM SIGPLAN Notices* (27:10), October 1992, pp. 45-62.

Fach, P.W., "Design Reuse through Frameworks and Patterns," *IEEE Software* (18:5), September/October 2001, pp. 71-76.

Fowler, M. *Analysis Patterns: Reusable Object Models*, Object Technology Series. Addison-Wesley Publishing Company, Reading, 1997.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional Computing Series, Addison-Wesley Publishing Company, New York, 1995.

Gillies, A.C., *Software Quality: Theory and Management,* Chapmann & Hall, London, 1992.

Kaindl, H. "Difficulties in the Transformation from OO Analysis to Design," *IEEE Software* (16:5), September/October 1999, pp. 94-102.

Kantor, B., and Lapsley, P. *Network News Transfer Protocol*, Request for Comments 977, Network Working Group, February 1986.

Malone, T.W., Grant, K.R., Turbak, F.A., Brobst, S.A., and Cohen, M.D. "Intelligent Information-Sharing Systems," *Communications of the ACM* (30:5), May 1987, pp. 390-402.

Moore, M.M., "Software Reuse: Silver Bullet?" *IEEE Software* (18:5), September/October 2001, p. 86.

Neighbors, J.M., "The Draco Approach to Constructing Software from Reusable Components," *IEEE Transactions of Software Engineeing* (10:5), September 1984, pp. 564-574.

NTIS. *North American Industry Classification System (NAICS) - United States*, National Technical Information Service, Springfield, 1997. ISBN-0-934213-57-7.

Press, L. "Collective Dynabases," *Communications of the ACM* (35:6), June 1992, pp. 26-32.

Vlissides, J., *Pattern Hatching: Design Patterns Applied,* Addison Wesley Longman, Reading, 1998.

Weibel, S., Kunze, J., Lagoze, C., and Wolf, M. *Dublin Core Metadata for Resource Discovery*, Request for Comments 2413, Network Working Group, September 1998.