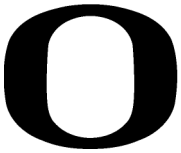
Presented to the Interdisciplinary Studies Program:



UNIVERSITY OF OREGON
APPLIED INFORMATION MANAGEMENT

Applied Information Management
and the Graduate School of the
University of Oregon
in partial fulfillment of the
requirement for the degree of
Master of Science

Adopting Software Design Patterns in an IT Organization: An Enterprise Approach to Add Operational Efficiencies and Strategic Benefits

CAPSTONE REPORT

John Knox
Software Developer
The Regence Group

University of Oregon
Applied Information
Management
Program

July 2011

Continuing Education
1277 University of Oregon
Eugene, OR 97403-1277
(800) 824-2714

Approved by

_____

Dr. Linda F. Ettinger
Senior Academic Director, AIM Program

Adopting Software Design Patterns in an IT Organization: An Enterprise Approach to Add Operational Efficiencies and Strategic Benefits

John Knox

The Regence Group

Spring 2011

Abstract

Software design patterns are recognized as a valuable part of good engineering practices (Buschmann, 2005). Literature published between 1995 and 2011 is examined in order to provide IT professionals with definitions, classifications, and benefits of software design patterns. Design patterns capture experience and provide multiple perspectives on design problems. They support improved communication, flexibility and extensibility, and collaborative and mentoring opportunities. Adoption and implementation are required enterprise wide to realize benefits and efficiencies.

Keywords: design patterns, software development, software design, design pattern frameworks, object-oriented development, software quality, software development lifecycle.

Table of Contents

Introduction to the Annotated Bibliography

Problem

As businesses become increasingly reliant on information technology (IT) to improve operational efficiency and provide strategic benefits, IT organizations must learn to design software in a way that increases software quality while speeding up software delivery (Bhatt & Grover, 2005; Bleistein, Aurum, Cox, & Ray 2003). The earliest design decisions made during the software development life cycle can have a significant impact on software quality; they can also be the most costly to revoke (Folmer & Bosch, 2007). A *design pattern* describes a proven solution to a design problem with the goal of assuring reusable and maintainable solutions (Garcia, Sant'Anna, Figueiredo, Kulesza, Lucena, & VonStaa, 2005). For example, security design patterns are meant to eliminate the accidental insertion of vulnerabilities into code or to mitigate the consequences of vulnerabilities (Dougherty, Sayre, Seacord, Svaboda, & Togashi, 2009). Dougherty further explains that if a security design pattern is not implemented it can leave a vulnerability that later becomes an operational problem; what may seem like a solid two-way remote procedure call during the design phase may cause a chain of weakness from an external threat in the final product.

According to Duell (1999), design patterns make a system more amenable to change by allowing some aspect of the system to vary independent of other aspects. Ram (2003) further explains that when patterns are used as the building blocks of architecture, they allow change to evolve in a way that prevents an erosion of the software design. This design centric view of change mitigates defects in the earliest phases of the design which reduces a burden on testing groups, enterprise support centers, security, change management, and operations (Ram, 2003). That's important, because the costs of fixing system vulnerabilities and the risks associated with

vulnerabilities after system deployment are high for both developers and end users (Dougherty, 2009).

However, all product creation has to balance speed, quality, and cost (Project Triangle, 2011). As noted by McKenzie and Wheaton (2010), if the use of a design pattern makes the software development effort more complicated than required, then the implementation of the solution may be out of balance. The question underlying this study is: How can software information technology (IT) organizations incorporate design patterns in a way that speeds software delivery while reducing software defects?

Significance

Using a pattern is not unique to software development. The study of patterns is well established in many other fields including architecture, anthropology, music, and sociology (Schmidt, Fayad, & Johnson, 1996). The concept of producing fewer defects in less time comes from taking advantage of previous knowledge, embedded in the design pattern (Gross, 2001). Design patterns constitute an important tool for improving software quality by providing reusable solutions for recurring design problems (Shlezinger, Reinhartz-Berger, & Dori, 2010). As noted by Lauder and Kent (1998), software design patterns capture the distilled experience of expert designers. The longer a pattern has been used successfully, the more valuable it is (Schmidt,1996).

According to Yacoub (2000), a pattern-oriented design is built by gluing together what he terms *constructional* design patterns, used as building blocks for various levels of design abstraction. These abstract building blocks can specify the core interface of the design solutions required without revealing the concrete details of the system architecture (Ahlgren, Penttila, & Markkula, 2005). This separation of functional requirements, or the "what," from technical

specifications, or the "how," provides a more focused way for business to communicate with their IT organization. From a software implementation perspective, the value of a design pattern comes from the codification of its specification (Ackerman & Gonzalez, 2007).

To maximize the benefit of a design pattern, it should be applied throughout the organization, and expectation should be managed carefully (Cline, 1996). This application means that patterns are successful because people take the time to read them, learn them, and use them (Schmidt, 1996*)*. Manolescu's (2007) research indicates that only half of the developers and architects in an organization use design patterns. To be effective, a consolidated effort is required to implement patterns throughout an IT organization because patterns have a context in which they apply (Schmidt, 1996).

Purpose

The purpose of this study is to develop a scholarly annotated bibliography that examines commonly used software design patterns. The intent is to provide information to IT leaders so that they are better able to justify and promote the use of these patterns throughout an IT organization. Specifically this study examines how the implementation of software design patterns throughout an IT organization can achieve two goals: (a) increased operational efficiency or (b) delivery of strategic benefits.

The assumption underlying this study is that through the use of software design patterns, IT organizations will be able to deliver software more quickly and with relatively few defects. As noted by Johnson (1997), developers who share a set of patterns and a common vocabulary for describing their designs can reuse solutions and accomplish these goals. Additionally, software development is rarely an individual effort and generally involves teams and diverse collaboration to generate good reliable code (Amrit & Hillegersberg, 2010). Design patterns capture the

essential properties of the software architecture by expressing the structure and collaboration of participants at a higher level of abstraction than source code. According to Schmidt (1996), this higher level of abstraction bridges the communication gap that exists between software developers and other members of a cross-functional development team.

The annotated bibliography is designed around the following research questions:

Main question. What factors must be considered when selecting and implementing software design patterns for use in an IT organization?

Sub-questions.

1. Why should an IT organization use design patterns?
2. What are the potential strategic benefits that an IT organization can realize by using design patterns in the software delivery process?
3. What are the potential gains in operational efficiency that might accrue by using design patterns in the software delivery process?
4. What are the most common software design patterns in use for IT application development?
5. What is the role of the IT leader in relation to promoting, implementing, and managing design patterns throughout an IT organization?

Audience

The audience for this annotated bibliography includes software developers and their supporting operational roles in IT. Developers, security analysts, database managers, systems administrators, software testers, and solution architects can communicate their respective domain expertise using a standard vocabulary during the software review process (see Definitions for

descriptions of these roles). As noted by Cline (1996), design patterns provide this type of a standard vocabulary for use among developers and provide a list of things to look for during a design review. According to Yacoub (2000), a lot of effort is expended in cataloging and discovering patterns in various domains for this very purpose. Some of the most common design elements include searching, retrieving, caching, persistence, and presentation. And while design patterns exist to help with these common elements, there is a significant learning curve within the organization that has to be balanced with a return on investment. The question must be asked if it makes sense to use a complex design pattern if the larger organization has not adopted it. This kind of question is best addressed using an enterprise level architectural assessment, which is essential in the design of high quality systems (Folmer & Bosch, 2007).

Delimitations

Time frame. The references provided in this study are published between 1995 and 2011. The oldest reference is the design patterns book released in 1995 by Gamma, Helm, Johnson and Vlissides. These four authors, who are often referred to as the Gang of Four (GoF), created a software development milestone when they created their design pattern book. While the inventory of patterns has not changed radically since this time frame, the analysis of the potential benefits and the descriptions of implementation continue to be developed.

Types of sources. The literature for this study is primarily selected from academic and professional journals. The academic literature provides theoretical guidance based on research while the professional journals include both research and case studies. The journals are diverse in that they range from the highly technical Software Engineering Institute and IEEE Communications magazine to the more business focused Information and Software Technology journal. According to Creswell (2009), journals often include articles that have undergone

rigorous reviews by editorial boards. All journal articles are either peer reviewed and/or the author is affiliated with an accredited university.

Exclusions. Literature on how to use design patterns is not included in this study. While implementing design patterns is part of the focus, the literature selected for this study only addresses this aspect from a management perspective. Besides an inventory of design patterns, the detail of design pattern use and detailed technical merit is not within the scope of this study.

Reading and Organization Plan Preview

The plan for reading the literature selected for use in this study starts with re-reading the abstract and introduction of each collected reference to grasp concepts and ensure the literature can support or add to the context of this research. The bibliography of each reference is then scanned in an attempt to ensure adherence to the Evaluation Criteria employed in this study. Finally, the reference is scanned for relevant keywords as identified in Research Parameters and Definitions portion of this study. The goal of this preliminary reading plan is to identify literature that address selecting and implementing software design patterns for the benefit of IT software delivery. Literature that meets the selection and evaluation criteria is then read in depth, in order to examine key concepts described in the set of research questions.

The plan for organizing and presenting literature in the Annotated Bibliography includes re-phrasing the research question into three main content areas including (a) design pattern benefits, (b) design pattern inventory, and (c) design pattern management. Each content area provides a brief description followed by a list of the selected pertinent annotated references. The goal of this organization is to provide full coverage of the research questions while allowing specific audience members to focus on specific content areas as needed.

Definitions

The following definitions provide specific meaning to terms as they apply to the use of design pattern in an IT organization, including a description of the key roles involved in this work. According to Amrit (2010), a collaboration of roles must be governed by technical task structure. The roles described are critical in implementing design patterns at an enterprise level. These roles, along with the definition of terms, provide a base layer of information for this study.

Database Manager: This role is required to design, install, monitor, maintain, and performance tune production databases while ensuring high levels of data availability. This individual is also responsible for developing, implementing, and overseeing database policies and procedures to ensure the integrity and availability of databases and their accompanying software (Info-Tech research group, 2011).

Design Patterns: A written document that describes a general solution to a design problem that recurs repeatedly in many projects. In object oriented programming, a pattern can contain the description of certain objects and object classes to be used, along with their attributes and dependencies, and the general approach to how to solve the problem (SearchSoftwareQuality, 2011).

Developer: This role is required to design, code, test, and analyze software programs and applications. This includes researching, designing, documenting, and modifying software specifications throughout the production lifecycle. The software developer also analyzes and amends software errors in a timely and accurate fashion and provide status reports where required (Info-Tech research group, 2011).

Operational Efficiency: Operational efficiency is about doing more with less. It is about business process improvements that lead to improved productivity and reduced operating costs (Operational Efficiency, 2011).

Object Oriented Programming (OOP): A programming language model organized around *objects* rather than *actions* and data rather than logic (SearchSOA, 2011).

Pattern Framework: A collection of patterns to solve a specific problem (SearchSoftwareQuality, 2011).

Software Development Lifecycle (SDLC): The process of creating or altering systems, and the models and methodologies that people use to develop these systems (Software Development Life Cycle, 2011).

Security Analyst: Performs two core functions for the enterprise. The first is the day-to-day operations of the in-place security solutions while the second is the identification, investigation and resolution of security breaches detected by those systems. Secondary tasks may include involvement in the implementation of new security solutions, participation in the creation and or maintenance of policies, standards, baselines, guidelines and procedures as well as conducting vulnerability audits and assessments. The IT Security Analyst is expected to be fully aware of the enterprise's security goals as established by its stated policies, procedures and guidelines and to actively work towards upholding those goals (Info-Tech research group, 2011).

Software Application Tester: The Systems/Software Application Tester's role is to plan, design, architect, and deploy effective test suites and regimes for in-house product development, software application development, information system launches, and operations systems enhancements. This individual applies proven analytical and problem-solving skills to help validate, verify, communicate, and resolve systems/software application issues through careful

testing in order to maximize the benefit of IT investments and initiatives level (Info-Tech research group, 2011).

Solutions Architect: This role is required to strategically design and implement in-house information systems and networked software architectures that support core organizational functions, and assure their high availability. This individual gains organizational commitment for all systems and software plans, as well as evaluates and selects all technologies required to complete those plans. In addition, the Systems Architect provides technical leadership across the organization, from strategic decision making down to the project planning level (Info-Tech research group, 2011).

Strategic Benefit: Benefits that focus on an organization's mission, vision and objectives (Strategic management, 2011).

Systems Administrator: This role is required to manage and tune in-house computer software systems and network connections to ensure high levels of availability and security of the supported business applications. This individual also participates in the planning and implementation of policies and procedures to ensure system provisioning and maintenance that is consistent with company goals, industry best practices, and regulatory requirements (Info-Tech research group, 2011).

Research Parameters

This section describes how the literature review is conducted. The search strategy report describes the detailed search strategy including databases accessed and key terms used as search criteria. Evaluation criteria outlines a framework to determine relevancy and credibility of the references that are included in this study. The documentation approach describes the methods used to record pertinent information about each reference. The full reading and organization plan provides detailed procedures for reading and organizing the material into key content areas that are applicable for the audience specified for this study.

Search Strategy Report

The selection of references to support this study focuses on the following areas of literature: (a) benefits of the use of design patterns in the software delivery process; (b) design pattern definition and inventory; and (c) operational efficiencies to be gained through implementation and management of design patterns in an IT organization. Google Scholar and UO Libraries catalog are used to search for journal articles as well as for finding alternate search engines and databases. References for this research study are also selected from ACM, IEEE Computer Science Digital Library, EBSCO HOST, Computer Source, Business Source, and Web of Science research databases. The search terms are mined during the initial searches and their value is validated after yielding results from the search engines.

Key Search Terms

- Design patterns
- Software development
- Software quality

- Software design
- Software architecture
- Object oriented development
- Design pattern frameworks
- Managing design patterns
- Software development lifecycle
- Software development patterns

Search Engines/Databases

- ACM Digital Library
- EBSCO HOST
- Google search
- Google Scholar
- IEEE Computer Science
- UO Libraries Catalog

Literature Evaluation Criteria

Chosen literature is first retrieved based on their abstract and their perceived authority as described by the delimitation section. This first pass includes using what Bell and Smith (2011) describes as the importance of authority and currency of a reference. The references are further evaluated for relevancy and quality by scanning both the body and the reference section looking for content that is applicable to the research questions. This pass includes using what Bell and Smith (2011) describe as the importance of objectivity, quality, and coverage of the work.

The relevancy of a reference is determined by counting and rating the mined elements from the abstract, body, and any other content the reference provides. Counting the content elements is done to ensure balanced coverage in relation to the number of references selected to address each of the sub-questions. A quality rating is also given to each reference based on how well it matches or supports at least one research question and the described needs of the audience. This approach provides a quantifiable and repeatable approach to evaluating the collected references with respect to the delimitation and research questions.

Literature is also included that defines and validates the terms and concepts presented in this study, including one book (Gamma, 1995) and several web references. While these references are not included in the Annotated Bibliography section of this document, they are of value and are included in the References.

The researcher's knowledge of the topic and audience plays a part in the evaluation of literature for use in this study. This knowledge does not apply to the authority of the content as described in the delimitation, but pertains to how to parse, objectively reason, and rate the content in relation to domain knowledge and audience needs.

The frequency of referencing by other literature is another criterion for consideration. For example, the design pattern book by Gamma et al. (1995) is often cited and it is included in the References for this paper.

Documentation Approach

Once selected, copies of each reference, along with mined elements that match the research question, are saved electronically. References and notes for this study are captured and organized using the following techniques:

- Electronic copies of each piece of literature, along with their respective URL, are saved to a computer file folders that is labeled with one of the one of the three content areas described by the Annotated Bibliography. The references in these folders are named with a rating followed by the author's last name. This combination of folder and naming convention provides a simple technique to sort and find references using the *detail list* perspective in Microsoft Explorer. References that rate lower, as specified in the Literature Evaluation Criteria, are sorted to the bottom using this file naming convention. Only the top 10 rated references in each area of content are selected for potential use in the Annotated Bibliography section of this study.

- All notes and markups are recorded electronically using the portable document format (PDF) annotation capability in Tracker Software's PDF-Xchange viewer. Any content that is not in PDF format is rendered to PDF format. Content mined from the literature are highlighted in yellow. Additionally, an electronic *sticky note* is created and placed beside selected content to detail any paraphrasing, coding, or details that apply to the research problem. These annotations are printed and filed by author/page providing a hard copy for quick reference.

Reading and Organization Plan

A lot of reference material has been published on design patterns since the publication of the Gamma book in 1995. However, only the reference material that supports or adds to the context of this research, as described in the Literature Evaluation Criteria, are selected for this study. This selected literature is then read in depth, in order to examine key concepts described in the set of research questions. Key concepts include (a) design pattern benefits, (b) design pattern inventory, and (c) design pattern management. According to Creswell (2009), organization of

literature enables the audience to understand how the research adds to, extends or replicates research already completed. The key concepts are used as a way to define the larger parameters of the study for a specified audience while the more detailed criteria ensure the research questions are examined as clearly as possible.

The review of the selected literature starts with the electronic annotation process described in the Documentation Approach. These annotated "chunks" of information are then validated and organized using the key concepts and criteria described below.

Design pattern benefits. Literature about design pattern benefits is collected to define the concept of a design pattern and to explain the integration of a design pattern as it applies to software development. These benefits can apply to anyone new to design patterns or for leaders that want to promote the use of design patterns. The benefits are identified, as they apply to an IT organization's software development lifecycle. The primary focus of this collection is on identification of benefits that increase operational efficiency and strategic benefit of an IT organization through:

- Improved delivery speed
- Reduced defects
- Improved business workflow and security
- Improved IT process and communication

Design pattern inventory. Literature about design pattern inventory helps to build and reinforce a vocabulary of design patterns that are commonly used across an IT organization. While the reference material is highly technical, the focus is on providing a cross-functional survey of design patterns. According to Manolescu (2007), the typical design pattern user

doesn't understand how to leverage patterns and applies only a small subset of design patterns centered on the 23 patterns from Gamma's (1995) Design Patterns book. Examination of this concept provides developers, and other technical supporting roles, a list of patterns used for the common challenges an IT organization faces in these areas:

- General purpose design patterns.
- Design patterns for common IT domains.
- Organization of design patterns.

Design pattern management. Literature about design pattern management defines the importance of implementing a design pattern in a way that will provide synergy to cross-functional IT teams. The focus is on examples that illustrate design pattern implementation knowledge of the past 16 years as it applies to improved operational and strategic benefits. IT software development and architecture leaders are the primary audience for this collection of literature that includes:

- Promoting design patterns.
- Implementing and managing design patterns.
- Best practices and lessons learned.

Annotated Bibliography

This annotated bibliography contains 30 key references selected to address the purpose of the study as defined by the research questions. The literature is organized into three content areas that are closely aligned with the research questions of this paper. The three content areas include:

(a) *IT benefits of design patterns* and how these are useful to anyone in IT who is new to design patterns or for IT leaders who want to promote the use of design patterns (9 references).

(b) *Design pattern definition and inventory* in order to provide developers, and other technical supporting roles, a list of patterns used for the common IT challenges ranging from general purpose to specific workflow and security patterns (9 references).

(c*) Managing design patterns* as a way to illustrate best practices, examples, and general principles learned over the past 16 years that IT software development and architecture leaders can apply (12 references).

Each annotation in this bibliography consists of three elements: an excerpt from the published abstract, an assessment of the credibility of the reference, and a summary of the content relevant to this study. Ideas presented in the summaries are paraphrased and/or quoted directly from the references.

IT Benefits of Design Patterns

Ackerman, L., & Gonzalez, C. (2007). The value of pattern implementations. *Dr. Dobb's Journal: The World of Software Development*, *32*(6)*,* 28-32. Retrieved Mar 30, 2011 from Computer Source.

Abstract. The article explores the benefits of implementing patterns in software designs. The concept of pattern specification consists of having a detailed description of the pattern, a context, and the forces that it addresses. The value of pattern implementation is that it allows the codification of a pattern specification and automates its application in a particular environment.

Summary. This article introduces developers and architects to the idea of a pattern implementation. A pattern implementation goes further than simply using pattern specifications as blueprints. The pattern implementation is an artifact that allows the codification of a pattern specification for a specific environment. Development teams can get the abstraction benefits of pattern specifications with the domain benefits of an implementation. The author indicates that these implementations are specific enough to be useful to a specific domain and yet general enough to be portable for modeling, coding, and scripting. In addition, pattern implementations can be created and used for different phases in the software development lifecycle, such as requirements gathering, as well as different levels of an application. The author uses an abstract factory pattern to illustrate the transformation modeling and to illustrate benefits. This concept can be used by a development team that wants an artifact that is both abstract yet covers the specific domain implementation in question.

Credibility. The authors have background in publishing articles on design patterns and have received accolades from highly respected peers on both their website and a book titled Patterns-

Based Engineering. This article is published in Dr. Dobb's Journal which is a popular professional journal that focuses on software development topics.

Chang, C., Lu, C., & Hsiung, P. (2011). Pattern-based framework for modularized software development and evolution robustness. *Information Software Technology, 53*(4). Retrieved April 20, 2011 from Business Source.

Abstract. As developers are increasingly asked for higher quality software systems from industries, there is insufficient guidance from the methodologies and standards of software engineering to provide assistance to the rapid development of qualified business software. In this paper, the authors discuss the advantages of the pattern-based software development. They verify the benefits of using a pattern-based software framework and a corresponding system design architecture that is intended for the rapid development of web applications.

Summary. This article is useful for developers and architects who want to learn about standardization, integration and pattern based frameworks. A background on standardization and integration technologies includes object-oriented technologies, xml, design patterns, frameworks, struts, spring, and hibernate. These technologies range from general building blocks to implementation of patterns that provide domain value. Pattern-based frameworks are introduced that can be used for retrieval and adaptation of patterns along with the open source software framework (OS2F) to integrate struts, spring, and hibernate to develop web applications. The concept is to separate the following layers:

- Data access layer (hibernate)
- Business layer (spring)
- Control layer (struts)
- Presentation layer (struts)

While this framework is specific to web development, it illustrates a model-view-controller design pattern that can be used by developers and architects for building robust applications with any architecture.

Credibility. Chang is an assistant professor at Hsiuping Institute of Technology and received his PhD in Computer Science from Geng Chia University. His research interests include design patterns and other software development concepts. Lu is an associate professor and the chair in the Department of Information Management, Hsiuping Institute of Technology. Lu received his PhD in information engineering and computer science from Geng Chia University and researches software engineering and software reuse. This article is published in a peer-reviewed journal.

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. Retrieved Mar 30, 2011 from Business Source.

Abstract. This article focuses on the pros and cons of adopting and applying design patterns, which are a valuable tool for practicing software professionals. Design patterns provide a standard vocabulary among developers. They provide a list of things to look for during a design review and a list of things that must be taught in a course on object-oriented design. They communicate information between designer, and programmer at a significantly higher level than individual classes or functions. To maximize the benefit of a design pattern, it should be applied uniformly throughout the organization, and and expectation should be managed carefully.

Summary. The author presents the practical benefits of design patterns as well as inhibitors to pattern applications. This article is beneficial to IT professionals who want to learn more or desire to promote the use of design patterns. The benefits of design patterns include:

- Design patterns coordinate the entire process and community through a common vocabulary.
- Design patterns can be used reactively and proactively through fragmenting and abstraction of design.
- Design patterns can be used to provide a software hinge or adaptability point.
- Design patterns can turn a trade-off into a win-win situation by allowing multiple facets of quality that are often viewed mutually exclusive.
- Design patterns constrain maintenance programmers by reducing the chance of breaking a design patterns adaptability point (or software hinge).

- Design patterns let management reward self-directed designers, as these designers require making an extra effort to understand and implement patterns.

Inhibitors to pattern applications include:

- Design patterns have been oversold and require investment to reap gains.
- Some design patterns are unnecessarily difficult to learn.
- Design pattern classifications are not yet useful for the average developer.

The author stresses that it takes time for a design pattern approach to mature. However, if the software development organization infuses the patterns into the software development lifecycle and invests in training and mentoring, the benefits can be realized in the short run.

Credibility. This article is published in a peer-reviewed journal. Cline is the author of two object-oriented language books and has taught programming design and analysis to professionals in both industrial and academic settings. Cline is President of MT Systems Company, which develops object-oriented systems for large corporations. He received a PhD in Electrical and Computer Engineering from Clarkson University.

Gross, D. & Yu, E. (2001). From non-functional requirements to design through patterns. *Requirements Engineering*, *6*(1). Retrieved May 1, 2011 from http://www.springerlink.com/content/luclrv4jlc5wye4p/

Abstract. This paper examines how design patterns play a critical role in the non-functional requirements (NFRs) of a software design solution and proposes a systematic treatment of NFRs to organize, analyze, and refine these requirements. The authors provide guidance and reasoning support when applying patterns during the design of a software system. Focus is on three design patterns to illustrate this approach in relation to how a developer or architect can reason, organize, analyze, and refine a design from a non-functional, high-quality, perspective.

Summary. This paper defines many of the design quality ingredients found in software development as non-functional requirements. The author states that system qualities are often expressed as non-functional requirements and include such requirements as reliability, usability, maintainability, cost, and development time. These attributes are critical for the success of a system.

The author describes software patterns as having a three-part rule, which expresses relations between a context, a problem, and a solution. The author lists the following four benefits that developers and architects can consider when using design patterns:

1. Clarifying the role of non-functional requirements in design pattern. This makes them a “first class citizen” with respect to the analysis and structure of quality added in the design.
2. Providing structure for characterizing each pattern. This includes characterizing and evaluating patterns for a solution with respect to quality.

3. Systematic support for the application of patterns during design. This provides better addressing of system wide non-functional requirements when retrieved, selected, and then applied.
4. Supporting forward engineering and traceability. Keeping track of how non-functional requirements drive the design and retain how the system structures evolve.

Credibility. This article is published in a peer-reviewed journal and has 33 scholarly references. The authors are faculty of Information Studies, University of Toronto.

Johnson, R. E. (1997). How frameworks compare to other object-oriented reuse techniques. Frameworks = (Components + paterns). *Communications of the ACM*, October 1997, *40*(10). Retrieved April 3, 2011 from http://www.inf.ufsc.br/~vilain/framework-thiago/p39-johnson.pdf.

Abstract. Frameworks can be thought of as a concrete form of a pattern in that they reuse both design and code. This paper describes the benefits and relationships between many types of object-oriented reuse techniques.

Summary. This paper provides clarity to technical staff between the various object-oriented reuse technologies and terminologies. For example, the model/view/controller is a user-interface framework that is often described as a pattern. The author explains how developers who share a set of patterns have a common vocabulary for describing their design, and also a way of making design tradeoffs explicit. Being able to descriminate between patterns and other technologies provides developers with the insight needed to take advantage of the appropriate tool set.

The author explains how a single framework typically contains many patterns, so patterns are a subset of frameworks. The author indicates that many patterns were discovered by examining a framework, and were chosen as being representative of reusable, object-oriented software. The explanation allows developers and architects to gain a clear understanding of how these object-oriented reuse techniques relate to each other and provide design benefits that will permeate the entire software development lifecycle.

Credibility. The author is the Coordinator of Project Design Activity in the Department of Computer Science at the University of Illinois-Urbana. This article is published in a peer-reviewed journal and is supported by 12 scholarly references.

Moynihan, G. P., Suki, A., & Fonseca, D. J. (2006). An expert system for the selection of software design patterns. *Expert Systems*, *23*(1)*,* 39-52. Retrieved Mar 30, 2011 from Computer Source.

Abstract. This paper describes benefits of design patterns and focuses on the development of a prototype expert system for the selection of design patterns that are used in object-oriented software. Design patterns provide a method of software reuse, which supports the goal of improved software development productivity. This article illustrates the feasibility of using expert system technology to aid in design pattern selection.

Summary. The authors describe how design decisions directly affect 65% of the work done in subsequent activities and how software design is an interactive process that must be transformed into a model for constructing software. The net results of creating patterns include a more flexible, elegant, and ultimately reusable component that is based on prior experience. The authors discuss how an expert system can benefit anyone who is implementing design patterns. The goal of this kind of system is to guide the designer through the pattern selection process via a targeted inquiry regarding the nature of the specific design problem. It also provides a means of browsing patterns and viewing the relationships between them. The expert system does this by prompting developers with questions relating to the selection based on the concept of a "better similar pattern." This results in a subset of patterns that become candidates for activation, which are then tested for applicability and consequences testing screen. This expert system can help automate the process of selecting and implementing design patterns by representing domain knowledge from human experts, using a user interface that helps guide a developer, in which design pattern combinations are suitable to solve specific problems.

Credibility. This article is published in a peer-reviewed journal. Moynihan and Fonseca are professors with the Department of Industrial Engineering at the University of Alabama. Their prior research includes manufacturing systems, artificial intelligence, and expert systems. Suki received his MS degree in Industrial Engineering from the University of Alabama.

Rising, L. (2010). The benefits of patterns. *IEEE software, 27*(5). Retrieved May 8, 2011 from Computer Source.

Abstract. Rising asks, does the real power of patterns lie not in the extraordinary solutions an expert provides or in the pattern writer's ability to capture deep insight into the simple, ordinary, basic structures that support good decision-making? This article provides two examples of one person's experience in uncovering the benefit of patterns.

Summary. This article describes the power of patterns as a way to remember the simple, ordinary, basic solutions that we know but forget in the heat of battle. Developers can learn from the author’s experience that patterns are handy tools and not just a way to create exotic solutions. The author analyzes an example from a small development team as it discovers that a novice pattern, called the Mediator, is a perfect fit for the design challenges that they had just spent hours battling.

Credibility. This article is published in a peer-reviewed journal. The author has a PhD from Arizona State University in object-based design metrics. She is an independent consultant and has published four books, including two on software design patterns.

Shlezinger, G., Reinhartz-Berger, I., & Dori, D. (2010). Modeling design patterns for semi-automatic reuse in system design. *Journal of Database Management*, *21*(1)*,* 29-57. Retrieved Mar 30, 2011 from Computer Source.

Abstract. Design patterns are reusable solutions that provide an important tool for improving software quality. This article focuses on an approach to modeling the different aspects of design patterns and semi-automatically utilizing these models to improve software design**.** The author shares her evaluation of this approach on commonly used design patterns and a case study of an automatic application for composing, taking, checking, and grading analysis and design exams. Her findings include that the suggested approach successfully locates the main design problems modeled by the selected design patterns.

Summary. This article takes some of the common patterns and categories and constructs a three-layered framework for modeling design patterns. The layers are the meta-design pattern layer, the design pattern layer, and the system layer. It illustrates this with standard modeling language as it uses factory, builder, chain of responsibility, composite, template method, decorator, observer, and command patterns as examples. The meta-design pattern layer includes specifications of commonalities among design patterns. The design pattern layer hosts design pattern models in a complete and coherent way. The system layer is the most concrete one, as it includes design models of different systems and applications. Evaluating the approach on eight commonly used design patterns and several case studies provides a better understanding for those who need to organize patterns. As Schmidt (1996) states, patterns are successful because people take the time to read them, learn them, and use them.

Credibility. This article is published in a peer-reviewed journal. The authors, Schlezinger and Dori, are faculty of Industrial Engineering and Management, Technion-Israel Institute of

Technology. The author Reinhartz-Berger is affiliated with the Department of Management Information Systems, University of Haifa.

Yacoub, S. A., Ammar, H. H., & Mili A. (2000). Constructional design patterns as reusable. *Software reuse: Advances in software reusability, 1844*, 369-387. Retrieved April 16, 2011 from Computer Source.

Abstract. This article focuses on the benefits of deploying proven design patterns early in the development lifecycle. Design composition and abstraction techniques are critical to providing the needed communication between classes and to construct applications. Examples of patterns are used to illustrate the benefits of this approach.

Summary. The article focuses on interfaces and developing *constructional* design patterns as building blocks. Developers can learn about the abstraction benefits of design patterns. The benefits include understanding the dependencies and collaboration between participating patterns while hiding implementation details. The pattern level view expresses the interface while the detailed pattern level view describes connectivity between interfaces. Additionally, this article describes the following positions and motivations that can benefit any development team:

- The most difficult part of building software is not coding, it's the decisions we make early at the design level.
- Early design decisions are crucial to quality.
- Promote pattern-based development.
- Develop systematic approaches to glue patterns.
- Facilitate design learning.
- Solve the traceability problem.

Credibility. The authors are affiliated with the Department of Computer Science and Electrical Engineering, West Virginia University. This article is published in a peer-reviewed journal and has 33 scholarly references.

Design Pattern Definition and Inventory

Bertrand, M., & Arnout, K. (2006). Componentization: The visitor example. *Computer*, *39*(7). Retrieved April 3, 2011 from http://se.ethz.ch/~meyer/publications/computer/visitor.pdf.

Abstract. Is it possible to go beyond patterns by componentizing them - turning them into components? The authors build a component library which answers this question positively for a large subset of the best-known design patterns. They summarize these results and analyze the componentization process through the example of an important pattern, Visitor, showing how to take advantage of object-oriented language mechanisms to replace the design work of using a pattern by mere "ready-to-wear" reuse through an application program interface.

Summary. All developers who use patterns build components from the common design elements described in this paper. The authors introduce and explain principles about how to build a self-contained component library for a pattern. This componentization effort leads to dividing patterns into the following categories: (a) fully componentized, 48%; (b) partially componentized, 4%; (c) Wizard or Library Support, 26%; and (d) not componentized, 9%.

The result of the componentization example of the Visitor pattern is a single class in a code library. By using the basic language mechanisms of genericity, tuples, and agents, the design pattern becomes an application program interface component that developers can invoke directly. The key criteria for this component includes faithfulness, completeness, usability, ease of learning, type safety, and performance. A client application can directly use the design element by instantiating the class and accessing the respective routines and actions. The interface can be learned by a developer in a few minutes by simply creating

the Visitor object and then passing in the desired actions to be executed on every object to be visited.

Credibility. This article is published in a peer-reviewed journal and is supported by 12 scholarly references. Meyer is a professor of software engineering at ETH Zurich and Chief Architect of Eiffel Software in California. Arnout is a software engineer at AXA Rosenberg in Orinda, California. She holds a PhD from ETH Zurich and her research interests include design patterns.

Dougherty, C., Sayre, K., Seacord, R., Svaboda, D., & Togashi, K. (2009). Secure design patterns. *Software Engineering Institute* , Retrieved April 3, 2011 from http://www.cert.org/archive/pdf/09tr010.pdf.

Abstract. This paper describes a set of secure design patterns, which are descriptions or templates describing a general solution to a security problem that can be applied in many different situations. The patterns are derived by generalizing existing best security design practices and by extending existing design patterns with security-specific functionality. They are categorized according to their level of abstraction: architecture, design, or implementation.

Summary. This paper describes secure design patterns and the rationale for using them. The cost of fixing system vulnerabilities and the risk associated with vulnerabilities after a system deployment are high for both developers and end users. This paper lists and describes an inventory of patterns for each of the following general classifications:

- Architectural-level patterns - This includes software running in higher privilege.
- Design-level patterns – This includes credentials and ensuring safe interfaces.
- Implementation-level patterns – This includes secure logging and data visibility.

Each pattern listed under the larger general pattern classification is explicated by a description that includes intent, motivations, applicability, structure (plus diagram), participants, consequences, implementation, known uses, and sample code. The paper will benefit any developer who wants a deeper understanding of the benefits of secure design patterns.

Credibility. The authors are affiliated with the Carnegie Mellon University's Software Engineering Institute. Dougherty is the team leader for the Vulnerability Analysis team in the CERT Coordination Center. This article is published in the Software Engineering Institute, a federally funded research and development center sponsored by the U.S. Department of Defense.

Fortis, A. F. & Fortis, F. (2008). Workflow patterns in process modeling. *Annals Computer Science Series, 6*(1). Retrieved April 16, 2011 from http://anale-informatica.tibiscus.ro/download/lucrari/6-1-07-Fortis.pdf

Abstract. This paper provides terminology, descriptions, and modeling for workflow based design patterns. This includes the definition of the process, process management (choreography), and the effective course of the process (orchestration).

Summary. Developers who model business processes will be interested in reading this paper that describes the high level concepts and lower level modeling of workflow patterns. The paper begins with definitions of common terminology used in workflows and describes and models each of the following patterns that range from simple to complex:

- **S**equence Pattern
- Parallel Split Pattern
- Synchronization Pattern
- Exclusive Choice Pattern
- Simple Merge Pattern
- Multiple Choice Pattern
- Synchronizing Merge Pattern
- Multiple Merge Pattern
- The Discriminator Pattern

These patterns provide generalized names and actions for the actions in today's business process management software. Integration and orchestration of software is implemented through these patterns and this article provides a good background for developers and other IT solution builders.

Credibility. This article is published in a peer-reviewed journal, founded in 2003 by the collective of researchers of Computers and Applied Computer Science Faculty in “Tibiscus” University of Timisoara, Romania. At the time of publication, both authors were assistant professors at the Universitatea de Vest Timisoara.

Fowler, M. (2003). Patterns. *IEEE software, 20*(2). Retrieved May 8, 2011 from http://www.se.rit.edu/~se362/Misc/FowlerOnPatterns-IEEESoftware-Mar-2003.pdf

Abstract. Patterns provide a mechanism for rendering design advice in a reference format. Software design is a massive topic, and when faced with a design problem, one must be able to focus on something as close to the problem as possible. Patterns can help by trying to identify common solutions to recurring problems. The solution is really what the pattern is, yet the problem is a vital part of the context. Patterns are half-baked, meaning one always has to finish them oneself and adapt them to one's own environment

Summary. This article communicates clearly about how patterns and libraries are associated. First, a library typically combines things so extracting specific content into a well-written set of patterns can help explain these concepts. Second, people often move between programming environments, and patterns provide this level of portability to allow solutions to be implemented in a new environment. Finally, libraries make decisions on implementation and a developer may need to know more about the implementation strategies than a library provides. Design patterns provide the transparency to view these implementation details.

The author explains the value of patterns as teaching tools. Written patterns help educate other team members for building and reviewing software. Some developers declare patterns as good or bad instead of appropriate or not. In the case of a simple application, having several design patterns implemented in the source code usually means that the benefit realized is not high enough compared to the complexity added. All developers could benefit from an ability to discern when patterns are applicable.

Credibility. This article is published in a peer-reviewed journal. The author is the chief scientist for ThoughtWorks, an Internet systems delivery and consulting company.

Garcia, A., Sant'Anna, C., Figueiredo, E., Kulesza, U., Lucena, C., & VonStaa, A. (2005). Modularizing design patterns with aspects: A quantitative study. *Lecture Notes in Computer Science,* doi:10.1007/11687061_2. Retrieved April 3, 2011 from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.5392&rep=rep1&type=pdf.

Abstract. This paper analyzes the most common design patterns using a highly technical set of assessment criteria. The study takes the most well known design patterns and applies computer engineering attributes such as separation of concerns, coupling, cohesion, and size to verify whether aspect-oriented approaches support improved modularization for crosscutting concerns.

Summary. This paper provides an inventory of patterns and analyzes them from a computer engineering perspective. For example, the Mediator pattern is illustrated graphically and in code. An abstract aspect encapsulates the common part to all potential instantiations of the pattern and creates public extensions for the pattern. Metrics are then gathered to evaluate the pattern implementation. This kind of quantitative analysis of patterns is useful for understanding both flexibility and cost. Developers can learn more details about the implementation of a pattern as well as how to improve pattern quality by taking in knowledge about these software engineering principles. This paper provides a more comprehensive view of common patterns a developer will use. Common patterns analyzed include: abstract factory, adapter, bridge, builder, chain of responsibility, command, composite, decorator, façade, factory method, flyweight, interpreter, integrator, mediator, memento, observer, prototype, proxy, singleton, state, strategy, template method, and visitor.

Credibility. This paper is published in a computer science technical journal and is supported by 23 scholarly references. The authors are affiliated with the Software Engineering Laboratory, Computer Science Department, Pontifical Catholic University of Rio de Janeiro.

Heer, J. & Agrawala, M. (2006). Software design patterns for information visualization. *IEEE Transactions on Visualization and Computer Graphics, 12*(5). Retrieved May 8, 2011 from Computer Source.

Abstract. This article presents a series of design patterns for the domain of information visualization. The article discusses the structure, context of use, and interrelations of patterns spanning data representation, graphics, and interactions.

Summary. Developers who focus on the presentation tier of software will benefit from the inventory of patterns in this article. The authors assert that by representing design knowledge in a reusable form, patterns can be used to facilitate software design, implementation and evaluation, improve developer education, and communication. The paper describes a set of twelve design patterns for information visualization software. Each pattern has a name, description, examples, and a standardized model for spatial representation. The patterns that are described in detail include: (1) Reference Model, (2) Data Column, (3) Cascaded Table, (4) Relational Graph, (5) Proxy Tuple, (6) Expression, (7) Scheduler, (8) Operator, (9) Renderer, (10) Production Rule, (11) Camera, and (12) Dynamic Query Binding.

The paper describes how the patterns in this domain can be combined to gain a greater value. Using a diagram, this article shows the class relationships between the visual patterns presented in this paper with the more common general purpose patterns. These specialized visual patterns provide an extremely valuable tool for developers who focus on the presentation layer and want to build highest quality user interfaces.

Credibility. This article is published in a peer-reviewed journal, and is supported by 23 scholarly references. Agrawala is an associate professor and the University of California, Berkley and has published articles on visualization, computer graphics, and human interactions.

Lauder, A., & Kent, S. (1998). Precise visual specification of design patterns. *E. Jul (Ed.): ECOOP'98, LNCS 1445*, 114-134. Retrieved April 3, 2011 from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.18.3268&rep=rep1&type=pdf.

Abstract. This paper describes two challenges with design patterns including having too many specific details and having natural language annotations. This paper addresses these problems by separating the specification of patterns into three models (role, type, and class).

Summary. In this paper, the authors propose the following three-model presentation of patterns:

- *Role-model* - The most abstract and depicts only the essential spirit of the pattern, excluding inessential application-domain-specific details.
- *Type-model* - Constrains the role-model with abstract state and operation interfaces forming a (domain-specific) refinement of the pattern.
- *Class-model* - Realizes the type-model, thus deploying the underlying pattern in terms of concrete classes.

The authors explain that purity, precision, and expressiveness are achieved by adopting this three-model layering of pattern descriptions; the essential spirit of the pattern is represented as a role-model, further refined by a type-model, and implemented by a class-model. Architects can use this concept when creating and maintaining an inventory of design pattern definitions. Additionally, the precise visual pattern specification will enable clear communication between domain experts and pattern writers (and ultimately pattern users) in order to operate at a higher level of abstraction without ambiguity. The article illustrates this layered approach using standard modeling and by decomposing the abstract factory design pattern.

Credibility. This article is published in a peer-reviewed journal, and is supported by 20 scholarly references. The authors are affiliated with the Divisions of Computing, University of Brighton, UK.

Tichy, W. (1998). A catalogue of general-purpose software design patterns. UIUC Patterns Group Version Feb 12. Retrieved April 11, 2011 from http://www.laputan.org/pub/patterns/tichy/catalogue.pdf

Abstract. Knowledge of software design patterns increases designers' abilities, leads to cleaner and more easily maintained software, speeds up implementation and test, and helps programmers document and communicate their designs. This paper catalogues over 100 general-purpose design patterns.

Summary. This paper concentrates on the problems solved by patterns and creates a catalog with the following top-level categories:

- Decoupling - Dividing a software system into independent parts in such a way that the parts can be built, changed, replaced, and reused independently. These patterns include repository, manager, iterator, sandwich, facade, mediator, bridge, adapter, proxy, decorator, facet, pipe, catch and throw, callback, event loop/channel, propagator, and observer.
- Variant Management - Treating different objects uniformly by factoring out their commonality. These patterns include super-class, visitor, template method, and abstract factory.
- State Handling - Generic manipulation of object state. These patterns include singleton, prototype, flyweight, and memento.
- Control – Control-of-execution and method selection. These patterns include blackboard, command, chain-of-responsibility, strategy, control state, master/slave, and process control.
- Virtual Machines - Simulated processors. These patterns include Interpreter patterns.

- Convenience Patterns - Simplified coding. These patterns include method, class, default class, and null object.
- Compound Patterns- Patterns composed from others, with the original patterns visible. These patterns include model/view/controller, bureaucracy, and active bridge.
- Concurrency - Controlling parallel and concurrent execution. These patterns include semaphore, critical region, monitor, double-checked locking, thread management, event loop, send/receive, reactor, rendezvous, listener, daemon, readers/writers, bounded buffer, active object, asynchronous completion token, half-sync/half-async, transaction and rollback,
- Distribution -Problems germane to distributed systems. These patterns include protocol stack, remote procedure call, router, acceptor and connector, gateway, forwarder/receiver, client/server, client/dispatcher/server, broker, and distributed state.

Credibility. The author has published five additional software development articles in the IEEE conference proceedings and journals and he is affiliated with the University of Karlsruhe, Germany. This article is published as an IEEE Computer Society invited paper and is supported by 17 scholarly references.

Zimmer, W (1994). Relationships between design patterns. Retrieved May 8, 2011 from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.37.8779&rep=rep1&type=pdf

Abstract. This paper organizes relationships of common design patterns into different category layers. The results simplify the understanding of the overall structure of the catalogue, thereby making it easier to classify other design patterns, and to apply these design patterns to software development.

Summary. The paper is useful for architects and developers who are working on an inventory of patterns. It expresses how patters use other patterns (lines connecting), how patterns are similar to other patterns (dotted lines) and how they can be layered together.

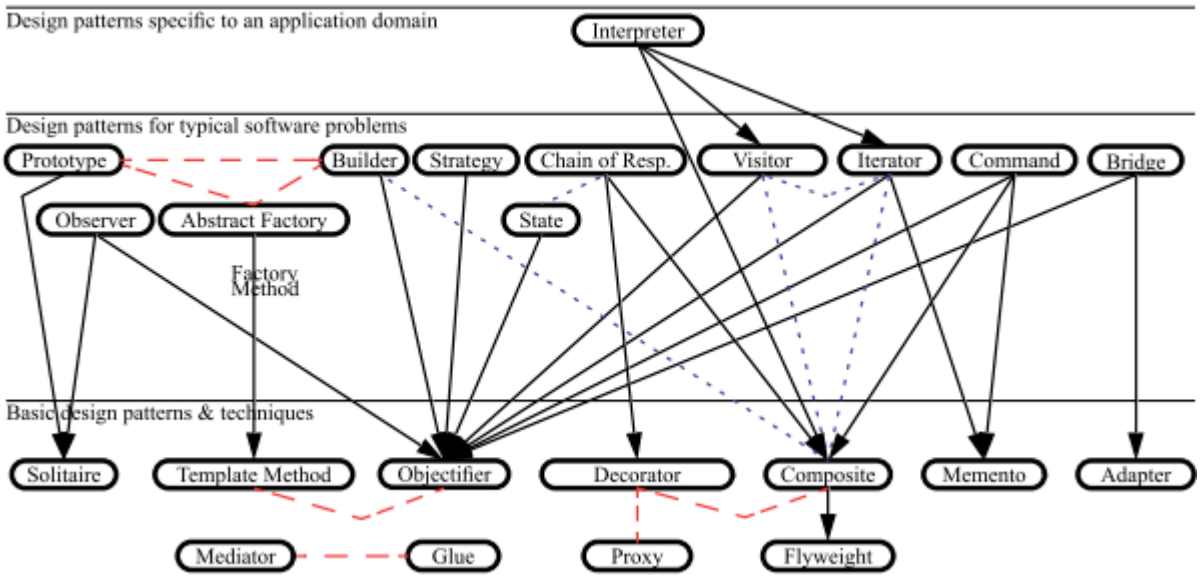


Figure 1. Arrangement of design patterns in layers

The paper provides a description and purpose of some of the common design patterns. The middle layer comprises design patterns which are used for more specific problems in the design of software. These design patterns are not used in design patterns from the basic layer, but in patterns from the application specific layer, and possibly from the same layer. This lower layer contains the design patterns, which are heavily used in the design patterns of higher layers and in

object-oriented systems in general. While many other arrangements and organizations exist, these layers and relationships can provide insight into the process of creating and classification of a design pattern inventory.

Credibility. The author is affiliated with Forschungszentrum Informatik, Bereich Programmstrukturen and has published three other papers on software design. This paper is supported by 22 scholarly references.

Managing Design Patterns

Ahlgren, R. P., Penttila J., & Markkula, J. (2005). Applying patterns for improving subcontracting management. *On the move to meaningful internet systems, 3762, 572-581.* Retrieved April 16, 2011 from Computer Source Complete.

Abstract. This paper evaluates software patterns as they pertain to the communication and co-operation needed in software subcontracting. The focus is on how software patterns are proposed and found to be suitable means for efficient, systematic, and explicit communication in managing subcontracting relationships.

Summary. This paper provides research for IT management to leverage patterns to overcome the communication challenges that exist in subcontracting management. While the communication challenges focus on subcontracting management, the solutions that are analyzed in this paper can apply to the silos and cross functional communication that is common in IT. As the article states, the players need efficient co-operation to be able to produce applications and services in a timely fashion. The paper emphasizes that communication of design information is critical to ensuring quality and that patterns provide a means to collect, store, and distribute design information about successful and verified solutions.

According to the article, patterns provide an abstraction that can be used to help track design alternatives and implementation trade-offs. They also provide traceability between high-level design and the lower-level implementation details. IT leaders can use this traceability to provide early warnings when something doesn't go as planned.

This paper stresses the importance of adequate levels of training for individuals and adequate visibility for an organization, including pattern repositories. IT leaders can apply this kind of pattern awareness to any software development organization.

Credibility. This literature is deemed credible by its publication in the peer-reviewed journal, Information Systems Management. This literature includes an extensive bibliography and the author has an affiliation with the University of Twente, Enschede, Netherlands.

Amrit, C. H. & Hillegerberg, J. (2010). Detecting coordination problems in collaborative software development environments. *Information Systems Management, 25*(1). Retrieved April 16, 2011 from http://arxiv.org/ftp/arxiv/papers/1006/1006.1243.pdf

Abstract. This paper examines the importance of team collaboration when it comes to generating a reliable software product. Four different classifications of patterns are discussed; the focus is on what is termed *socio-technical* patterns. The importance of an alignment of design and team interactions is stressed.

Summary. This paper details some of the dynamics found in today's cross-functional IT teams. If teams involved in software production have shortcomings in their interpersonal relationships, the resulting technical architecture of the software is likely to be flawed. Collaboration of the developers, designers and testers must be related to and governed by the technical task structure. These collaboration practices are handled in what this paper describes as *socio-technical patterns*.

IT leaders who want to improve collaboration may benefit from this paper's emphasis on best practices and design patterns; the problem with the use of best practices is that they are not precisely formalized and are often difficult to apply. Software design patterns, which are proven solutions to recurrent software development problems, provide a specific design solution rather than attempting to solve a problem from high-level principles. The insights this paper provides on how cross-functional teams collaborate with project and functional leaders can be applied to any IT organization.

Credibility. This article is published in a peer-reviewed journal. Chintan Amrit is a PhD student while Jos van Hillegersberg is a professor and chairman of the Information Systems and Change Management Department at the School of Management and Governance, University of

Twente, the Netherlands. Both authors have a background in software development, research, and training.

Bleistein, S. J, Aurum, A., Cox, K., & Ray, P. K. (2003). Linking requirements goal modeling techniques to strategic e-business patterns and best practices. *AWRE, 3*. Retrieved May 22, 2011 from http://www.caeser.unsw.edu.au/publications/pdf/Tech03_2.pdf

Abstract. This paper focuses on using patterns as a way to facilitate the requirements discovery process for strategic-level information systems initiatives in business. The paper asserts that best practices in business strategy are based on recurring business models.

Summary. This paper examines a business driven approach to design patterns. IT leaders responsible for keeping team goals in-line with strategic objectives can study several domain modeling approaches shown in the following diagram:

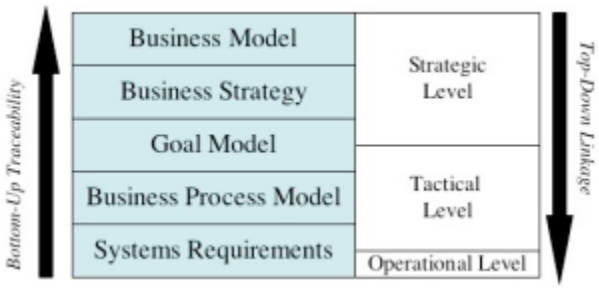


Figure 2. Proposed framework for strategy-oriented model.

This article illustrates how modular patterns can be derived from business research. IT leaders can promote the concept of design patterns by emphasizing the high level abstraction benefits in terms of an organization's operational and strategic goals.

Credibility. This article is published in a peer-reviewed journal and provides extensive references to prior work. The authors are affiliated with the School of Information Systems, Technology and Management, National ICT Australia and have published several articles on requirements engineering and e-Business best practices.

Booch, G. (2009). Software abundance in the face of economic scarcity, part 2. *IEEE Software* 26(6). Retrieved May 8, 2011 from Computer Source.

Abstract. This editorial, from a well known software leader, makes the case that software-intensive systems are a necessary element in helping us operate, innovate, and thrive in the face of lean economic times. Booch states that software-intensive systems are subject to the usual business forces (cost, schedule, and mission) and notes that the enterprise's architecture is the collection of engineering decisions and artifacts that steer the organization through the forces acting upon it and guide it toward its mission.

Summary. This editorial provides an argument for investing in software development and states that design patterns can bring economics of scale. The author acknowledges that while we are in a period of material scarcity, we are also graced with a cognitive surplus, an abundance that can fuel the production of software intensive systems, which is only limited by human imagination and labor. This perspective can help IT leaders provide an argument for promoting and implementing design patterns by seizing software as a "strategic weapon." The article emphasizes focusing on standardized technology, the formation of an optimized core, and business modularity.

Credibility. The author is an IBM Fellow and one the leading authorities in software modeling and design; he has written many software modeling and design books that have become a standard in the industry. The editorial is published in a peer-reviewed journal.

Bottoni, P., Guerra, E. & de Lara, J. (2010). A language-independent and formal approach to pattern-based modeling with support for composition and analysis. *Information & Software Technology, 52*(8). Retrieved April 20, 2011 from Business Source.

Abstract. This paper provides a language-independent formalization of the notion of pattern, allowing application to different modeling languages and tools, as well as generic methods to enable pattern discovery, instantiation, composition, and conflict analysis.

Summary. IT can formalize patterns from different domains, providing methods to analyze conflicts and dependencies that usually are expressed only in textual form. Software engineering promotes quality, standardization, reusability, and maintainability of software artifacts. However, the full realization of power is hindered by the lack of a standard formalization of the notion of pattern. This paper describes a language-independent modeling language that offers the ability to provide inter-pattern synchronization and constraints across several diagrams. The examples range from the basic singleton pattern to more specialized workflow patterns. IT leaders can leverage this information in order to integrate diverse domain patterns into their existing domain models for the sake of consistency and reuse.

Credibility. This article is published in a peer-reviewed journal. Bottoni is from the Computer Science Department, “Sapienza” Università di Roma, Italy. Guerra is from the Computer Science Department, Universidad Carlos III de Madrid, Spain. De Lara is from the Polytechnic School, Universidad Autónoma de Madrid, Spain.

Buschmann, F., Henney, K., & Schmidt, D. C. (2007). Past, present, and future trends in software patterns. *IEEE Software, 24*(4). Retrieved May 8, 2011 from Computer Source.

Abstract. This article examines how patterns provide a vocabulary for expressing architectural visions and clear, concise representative designs and detailed implementations. Patterns let developers communicate more effectively, with greater conciseness and less ambiguity. Examination of past, present, and future trends can help developers improve their projects through analysis.

Summary. This article is for IT leaders who would like to know more about where patterns are today and be able to predict their value for tomorrow. Patterns capture experience, and for newer domains and technologies, developers must first gain, evaluate and codify this experience before they can document effective patterns. While this article is four years old, the trends discussed are consistent with the more recent literature in this review. This article touches on many of the existing patterns from the Gang of Four's book (Gamma, 1995) as well as new patterns that are generating a lot of attention including business orchestration, web2.0, and service oriented architecture (SOA). Web2.0 is changing the various presentation and transport protocols on the internet, while SOA is a style of organizing and using distributed capabilities that different organizations or groups control and own.

Credibility. The article is published in a peer-reviewed journal and the three authors have extensive experience research, reviewing, and authoring publications on design patterns. Buschman is a principal engineer at Seimens Corporate Technology in Munich and has co-authored four volumes of pattern architecture books. Henney is a consultant and trainer in information services, member of ACM, and has co-authored two volumes of software pattern books. Schmidt is a computer science professor, chair of Vanderbilt University's Computer

Science program, chief technology officer for Prism Tech, and has extensive background in design patterns research and publishing.

Folmer, E., & Bosch, J. (2007). A pattern framework for software quality assessment and tradeoff analysis. *International Journal of Software Engineering & Knowledge Engineering* , *17*(4), 515-538. Retrieved Mar 30, 2011 from http://www.worldscinet.com/ijseke/17/1704/S021819400700332X.html

Abstract. This paper presents a framework that formalizes some of the relationships between software architecture and software quality; it compiles existing design knowledge (quality improving patterns) in a format suitable for architecture assessment.

Summary. This paper provides IT management and software architects a framework that can proactively improve quality by compiling existing design knowledge through quality improving patterns. The authors assert that this framework can improve efficiency and reduce costs for systems that fall under the respective architectures. By compiling a list of existing domain knowledge, the framework can remove the impact of change and defects at the earliest point, while they are the least costly to revoke. A pattern format is provided that will help pattern authors organize content, including: (a) quality, (b) authors, (c) problem, (d) when to use, (e) solution, (f) rationale (why use it), and (g) quality principles. Concrete examples from security are provided that help assist in reasoning on the pattern format and possible architectural tradeoffs. While this article is more general than most of the literature in this collection, the pattern format, pattern organization, and architecture implications can assist IT leaders with implementing design patterns.

Credibility. This article is published in a peer-reviewed journal and includes 50 references. Folmer has a PhD from the University of Groningen where he is an assistant professor at the Department of Mathematics and Computer Science, University of Groningen, The Netherlands.

Bosch is from the Software and Application Technologies Laboratories, Nokia Research Center, Helsinki Finland.

Manolescu, D., Kozaczynski, W., Miller, A., & Hogg, J. (2007). The growing divide in the patterns world. *IEEE software, 24*(4). Retrieved May 8, 2011 from Computer Source.

Abstract. This article analyzes information from the Microsoft patterns & practices group's usage and surveys customer satisfaction metrics to provide unique insights into how practitioners use patterns. The authors address key challenges with design patterns and states that bridging the gap between the patterns expert community and the typical pattern user is critical to achieving the full benefits of software patterns.

Summary. This article emphasizes practical use of design patterns; the authors explain that one of the primary reasons for the survey report that indicates a low adoption of software patterns is because there is currently no authoritative index. Practitioners seeking patterns for a particular problem space fall back on search engines to find them. Education is another challenge in the adoption of patterns. Ninety percent of respondents had not taken any pattern courses. Practitioners who describe their own patterns don't share them, which is problematic because pattern authoring should be collaborative. IT leaders can use this information to remove roadblocks and gain adoption of patterns by facilitating discovery, education, and collaboration.

Credibility. This article is published in a peer-reviewed journal. All three authors work at Microsoft's pattern and practices group. Manolescu is an architect and member of IEEE, earned his PhD in computer science from the University of Illinois, and was lead editor of Pattern Languages of Program Design 5. Kozacynski is an architect and earned his PhD in information sciences from the Technical University of Wroclaw. Miller is a software development lead and received his PhD in physics from the University of Southampton.

Ram, D. J., & Rajasree, M. S. (2003). Enabling design evolution in software through pattern oriented approach. *9th International conference Object-oriented information systems 2817: 179-190*. Retrieved April 16, 2011 from Computer Source.

Abstract. Architectural erosion in software systems results due to the drifting of code from the initial design. This drifting happens because the changes in code (supporting evolving requirements) are not reflected back in the design. The authors propose a pattern-oriented approach to software development that uses patterns as building blocks of architecture in order to prevent software erosion.

Summary. This paper illustrates ways to implement software using a pattern-oriented approach. The author stresses that successful evolution of software system depends on the traceability of artifacts produced in each stage of the software development lifecycle (SDLC). Project and functional managers who are currently incurring a large cost in software sustainment efforts can benefit from this article because, as this article reiterates, requirement changes in terms of patterns provide a substitution that keeps the integrity of the design.

Credibility. This paper is a research report, published in the OOIS 2003 conference proceedings. The authors are affiliated with the Department of Computer Science and Engineering, Indian Institute of Technology.

Schmidt, D., Fayad, M., & Johnson, R. (1996). Software patterns. *Communications of the ACM* , *39*(10), 36-39. Retrieved Mar 30, 2011 from UO library search.

Abstract. The article describes software patterns and successful solutions to common contextual software problems. The authors state that all solutions have a cost, and pattern descriptions should state the costs clearly. Finding a pattern is a matter of discovery and experience, and the longer a pattern has been used successfully, the more valuable it tends to be. These experts assert that patterns are successful because people take the time to read them, learn them, use them, and write them.

Summary. IT leaders can cultivate a positive culture by promoting what the authors describe as *motivators* to encourage people to document design patterns including:

- Discovery and experience are keys to success instead of novelty.
- Naming and describing patterns put an emphasis on writing clarity.
- Appreciate and reward the creative process of expert developers.
- Good patterns arise from practical experience that should be shared.
- Recognize the importance of human dimensions in software development.

Credibility. This article is published in a peer-reviewed journal. Schmidt is a computer science professor, chair of Vanderbilt University's Computer Science program, and the chief technology officer for Prism Tech. He is a member of IEEE and has co-authored several books and publications on design patterns.

Schmidt, D. (1995). Experience using design patterns to develop reuseable object-oriented communication software. *Communications of the ACM* , *38*(10). Retrieved April 11, 2011 from GoogleScholar.

Abstract. This paper describes how design patterns help to enhance software quality by addressing fundamental challenges including communication of architectural knowledge among developers, accommodating new design paradigms or architectural styles, and avoiding development traps and pitfalls that are usually learned only by experience. This author provides insights and recommendations gained from several years of experience.

Summary. The following lessons learned can help IT leaders promote and implement a design pattern solution that works effectively:

- Patterns improve communication within and across software development teams.
- Pattern names should be chosen carefully and used consistently.
- Patterns enable wide-spread reuse of software architecture.
- Pattern descriptions should contain concrete examples.
- Patterns explicitly capture knowledge that experienced developers already know.
- Focus on developing patterns that are strategic to the domain and promote reuse.
- Institutionalize rewards for developing patterns.
- Patterns are validated by experience rather than by testing.
- Directly involve pattern authors with application developers and domain experts.
- Integrating patterns into a software development process is human-intensive.
- Carefully document the contexts where patterns apply and do not apply.
- Patterns facilitate training of new developers.
- Successful pattern descriptions capture both structure and behavior.

- Managing expectations is crucial to using patterns effectively.

Credibility. The author is a computer science professor, chair of Vanderbilt University's Computer Science program, and the chief technology officer for Prism Tech. He is a member of IEEE and has co-authored several books and publications on design patterns. This article is published in a peer-reviewed journal.

Wendorff, P. (2001). Assessment of design patterns during software reengineering: Lessons learned from a large commercial project. *IEEE Computer Society.* Retrieved May 1, 2011 from http://www.ptidej.net/teaching/ift6251/fall05/presentations/050919/Wendorff.pdf

Abstract. This paper describes how software patterns have been applied inappropriately due to lack of experience. Instead of an increase in software quality, severe problems resulted when a large commercial project used patterns in what is best described as an uncontrolled implementation.

Summary. This paper provides IT leaders with knowledge that can balance the potential benefits of using design patterns with risks associated with a poor implementation. There are many critical success factors and many software patterns are simply over-engineering a solution. For example, the post mortem on a project that used one of the more popular patterns, called the Command Pattern, indicates that the developer did not clearly understand the rationale behind the pattern and added complexity without the benefits of promised flexibility. Due to the complexity, removing and reengineering the pattern was no longer a viable option. This article provides many examples of lessons learned and how an unconstrained deployment of patterns can provide just enough knowledge to be dangerous. The author also makes an argument that much of the industry has this type of uncontrolled implementation. IT leaders and developers can learn from this article when defining and implementing the critical success factors needed to avoid these design pattern pitfalls.

Credibility. This paper is published in a peer-reviewed journal. Wendorff received a degree in electrical engineering / technical informatics from Duisburg University, Germany, and a Masters degree in software engineering from the University of Westminster, England. He is co-

founder of ASSET GmbH, a company that does consulting around software productivity issues. Wendorff has 19 years of professional experience as an IT consultant and has a background in academic research and publishing.

Conclusion

This paper identifies commonly used software design patterns and examines how the expert knowledge these patterns contain can help IT organizations increase operational efficiency and deliver strategic benefit. Literature from the past 15 years is categorized to provide knowledge on the definitions, classifications, applicability, benefits, and efficiencies that can result from the use of software design patterns. The goal is to better prepare IT professionals to make wise decisions when it comes to adopting design patterns in software designs and across software development organizations. The selected literature provides an overview of patterns that extends well beyond the initial 23 object-oriented design patterns identified in Gamma's (1995) Design Patterns book. The literature includes many non object-oriented patterns as well as domain specific patterns such as security and workflow. IT organizations that span across diverse functional areas will need patterns for all sorts of problem domains including architectural styles, object-oriented frameworks, domain models, usability, reliability, safety, maintainability, security and performance patterns (Folmer, 2007). The research in this paper provides insight to cross-functional development teams using a broad survey of design patterns, designed to reveal strategic benefits and operational efficiencies.

A theme that permeates throughout the selected literature is the concept of capturing experience and providing multiple-perspectives on design problems. As noted by Moynihan, Suki, & Fonseca (2006), there is a remarkable difference between the approaches of experienced and inexperienced designers toward solving a software design problem as it relates to flexibility, elegance, and reusability. Software design patterns are used to collect, store, and distribute design information about successful and verified solutions. However, to gain efficiency, these software patterns must be utilized by a software business that is formed by networks of

participants who operate efficiently and cooperatively to be able to produce the products, applications, and services on time (Ahlgren, 2005).

According to Gross (2001), many of the system quality requirements (non-functional requirements) are determined during the design phase of the development process. A good designer has learned from experience how to efficiently address a range of quality requirements including reliability, usability, maintainability, cost, and development time. Folmer (2007) suggests that adding quality improvement solutions during late stages of development is difficult because it can affect many parts of the software and require a system to be completely reworked. By addressing quality early in the design, an experienced developer can reduce defects and design retrofits that are often a costly problem in the later stages of the software development lifecycle (Folmer, 2007).

Strategic Benefits

Because design patterns capture distilled experience, they can provide a communication tool throughout the software development lifecycle and across diverse communities of designers and programmers (Cline, 1996). This *improved communication* among software developers is a benefit that can empower less experienced developers to produce high-quality designs. According to Fowler (2003), an expert on a team can use written design patterns to help educate other team members as they work through software requirement, design, and review. According to Schmidt (1996), design patterns capture the structures and collaborations of components in successful solutions to problems that arise when building software in domains like business data processing, telecommunications, graphical user interfaces, databases, and distributed communication software. Design patterns provide a standard vocabulary among developers as

well as an abstract cross-functional uniformity that is an important component of a software development project (Cline, 1996).

Two related benefits presented throughout the selected literature are *flexibility* and *extensibility*. Tichy (1998) describes the concept of flexibility as how design patterns can improve the structure of software, speed up implementation, simplify maintenance, and help avoid architectural drift. Cline (1996) describes flexibility points or *hinges* where the software is designed to bend/adapt. Future changes (extensibility) consistent with one of these hinges are relatively inexpensive, but forcing the software to change in other ways is like bending your elbow backwards; the system normally breaks. However, design pattern flexibility often comes at a price of complexity as dynamic, highly parameterized software is harder to understand and document (Wendorff, 2001). Wendorff's research indicates that the poor judgments of individual software developers often add complexity without benefit which results in designs that become difficult to alter or remove. According to Rising (2010), the real power of patterns is not to provide exotic solutions, but to find and document simple, basic solutions that may be overlooked in a given circumstance.

The educational benefits to be gained from learning to classify and organize design patterns include *discovery*, *collaboration*, and *mentoring*. According to Zimmer (1994), organizing design patterns can simplify the understanding of the overall structures, which makes it easier to apply design patterns to software development and to further classify other design patterns. Tichy (1998) describes a more purpose-based classification that combines many of the "classic" design patterns that have been around for decades with many of the more modern general purpose design patterns. This approach to classifications promotes discovery and learning with an aim to make it easier for designers to use well-known and successful designs

developed from expert experience (Chang, 2011). According to Rising (2010), the debate continues about the merits of providing developers with formal training versus simply having access to a large searchable repository within which to search for a pattern to address some sticky problem.

Operational Efficiencies

The literature included in this paper provides IT leaders with insight, reasoning, and techniques to promote and implement design patterns in order to gain operational efficiency and provide strategic benefit for their organization. While learning and organizing design patterns provides an important step, Fowler (2003) states that all developers must finish and adapt design patterns to their own environment and that implementing a pattern for a particular purpose is one of the best ways to learn about it. According to Bleistein (2003), engineers find themselves under increasing pressure to deliver solutions with a high degree of quality in a timely manner. As the Project Triangle (2011) describes, project management can only focus on two of the following three facets of a development effort: (a) speed, (b) quality, and (c) cost. The literature in this paper describes how delivery speed and software quality improve through the use of design patterns. However, the selected literature makes it clear that it requires an investment in education and adoption that crosses functional boundaries of IT. Without this adoption and education, the inappropriate application of patterns can backfire (Wendorff, 2001).

Design patterns implementation is not limited in scope to the individual developer using object-oriented design patterns. Wendorff (2001) clearly describes costly lessons learned from implementations that treat design patterns in this way. The literature in this paper describes how design patterns add strategic value by increasing collaboration of software architecture at a higher level (Bleistein, 2003; Schmidt, 1996). IT leaders can use this research to promote design

patterns as a tool for innovation. As Booch (2009) explains, organizations that use software development as a strategic weapon will be increasingly focused on standardized technology, the formalization of an optimized core, and the emergence of architecture of business modularity.

Over the past 15 years, the design patterns pattern community has been successful in promoting good software engineering practices through mining, documenting, and applying patterns (Buschmann, 2007). However, according to Manolescu (2007), adoption rates are still low for IT organizations due to lack of discovery and limited education around how to apply design patterns to specific domain contexts. A survey from Manolescu (2007) indicates that only half of the developers and architects in IT organizations use patterns. Additionally, ninety percent of respondents that claim to be pattern practitioners hadn't taken any educational courses on design patterns. Manolescu attributes this low adoption rate to the fact that finding patterns relevant to a particular problem isn't trivial and because the design pattern world doesn't have an authoritative index. This challenge is, in part, due to the nature of how patterns often match a problem domain and each domain needs a distinct approach (Bleistein, 2003).

According to Buschmann(2007), domain-specific patterns are a work in progress and developers must first gain, evaluate, and codify this experience before they can document effective patterns. IT leaders can learn from the selected literature in this paper by fostering a culture that leverages design patterns as a vehicle for improving software development team collaboration and mentoring. The research shows that only through adoption and implementation of software design patterns at an enterprise level can IT fully realize the benefits and efficiencies.

References

Ackerman, L., & Gonzalez, C. (2007). The value of pattern implementations. *Dr. Dobb's Journal: The World of Software Development* , 28-32. Retrieved Mar 30, 2011 from Computer Source.

Ahlgren, R. P., Penttila J., & Markkula, J. (2005). Applying patterns for improving subcontracting management. *On the move to meaningful internet systems. 3762: 572-581.*

Amrit, C. H. & Hillegerberg, J. (2010). Detecting coordination problems in collaborative software development environments. *Information Systems Management, 25(1).* Retrieved April 16, 2011 from http://arxiv.org/ftp/arxiv/papers/1006/1006.1243.pdf

Bell, C., & Smith, T. (2009). Critical evaluation of information sources. *University of Oregon Libraries*. Retrieved May 1, 2011 from http://libweb.uoregon.edu/guides/findarticles/credibility.html

Bertrand, M., & Arnout, K. (2006). Componentization: The Visitor example. Retrieved April 3, 2011 from http://se.ethz.ch/~meyer/publications/computer/visitor.pdf.

Bhatt, G. D., & Grover, V. (2005). Types of informational technology capabilities and their role in competitive advantage: an empirical study. *Journal of Management Information Systems, 22*(2).

Bleistein, S. J, Aurum, A., Cox, K., & Ray, P. K. (2003). Linking requirements goal modeling techniques to strategic e-business patterns and best practices. *AWRE, 3*. Retrieved May 22, 2011 from http://www.caeser.unsw.edu.au/publications/pdf/Tech03_2.pdf

Booch, G. (2009). Software abundance in the face of economic scarcity, part 2. *IEEE Software 26*(6). Retrieved May 8, 2011 from Computer Source.

Bottoni, P., Guerra, E., & de Lara, J. (2010). A language-independent and formal approach to pattern-based modelling with support for composition and analysis. *Information & Software Technology, 52*(8), 821-844.

Buschmann, F., Henney, K., & Schmidt, D. C. (2007). Past, present, and future trends in software patterns. *IEEE Software 24(4)* . Retrieved May 8, 2011 from Computer Source.

Chang, C.-H., Lu, C.-W., & Hsiung, P.-A. (2011). Pattern-based framework for modularized software development and evolution robustness. *Information & Software Technology. 53*(4).

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. Retrieved Mar 30, 2011 from Business Source.

Creswell, J. W. (2009). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches.* Thousand Oaks, California: SAGE Publications, Inc.

Dougherty, C., Sayre, K., Seacord, R., Svaboda, D., & Togashi, K. (2009). Secure design patterns. *Software Engineering Institute* , Retrieved April 3, 2011 from http://www.cert.org/archive/pdf/09tr010.pdf.

Duell, M. (1999). Managing change with patterns. *IEEE communications magazine, 37*(4). Retrieved April 16, 2011 from Computer Source Complete.

Folmer, E., & Bosch, J. (2007). A pattern framework for software quality assessment and tradeoff analysis. *International Journal of Software Engineering & Knowledge Engineering*, Aug, *17*(4), 515-538. Retrieved Mar 30, 2011 from http://www.worldscinet.com/ijseke/17/1704/S021819400700332X.html

Fortis, A. F. & Fortis, F. (2008). Workflow patterns in process modeling. *Annals Computer Science Series, 6*(1). Retrieved April 16, 2011 from http://anale-informatica.tibiscus.ro/download/lucrari/6-1-07-Fortis.pdf

Fowler, M. (2003). Patterns. *IEEE software, 20*(2). Retrieved May 8, 2011 from http://www.se.rit.edu/~se362/Misc/FowlerOnPatterns-IEEESoftware-Mar-2003.pdf

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software.* Addison-Wesley.

Garcia, A., Sant'Anna, C., Figueiredo, E., Kulesza, U., Lucena, C., & VonStaa, A. (2005). Modularizing design patterns with aspects:A quantitative study. *Software Engineering Laboratory - Computer Science Department Pontifical Catholic University of Rio de Janeiro* , Retrieved April 3, 2011 from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.5392&rep=rep1&type=pdf.

Gross, D. & Yu, E. (2001). From non-functional requirements to design through patterns. Requirements Engineering, *6*(1). Retrieved May 1, 2011 from http://www.springerlink.com/content/luclrv4jlc5wye4p/

Heer, J. & Agrawala, Maneesh (2006). Software design patterns for information visualization. *IEEE Transactions on Visualization and Computer Graphics, 12*(5). Retrieved May 8, 2011 from Computer Source.

Info-Tech research group. (2011). Retrieved April 25, 2011 http://www.infotech.com/research/tools-and-resources/top/job-descriptions

Johnson, R. E. (1997). How frameworks compare to other object-oriented reuse techniques. Frameworks = (Components + paterns). *Communications of the ACM* , October 1997, *40*(10). Retrieved April 3, 2011 from http://www.inf.ufsc.br/~vilain/framework-thiago/p39-johnson.pdf.

Lauder, A., & S., K. (1998). Precise visual specification of design patterns. *E. Jul (Ed.): ECOOP'98, LNCS 1445* , 114-134. Retrieved April 3, 2011 from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.18.3268&rep=rep1&type=pdf.

Manolescu, D., Kozaczynski, W., Miller, A., & Hogg, J. (2007). The growing divide in the patterns world. *IEEE software, 24*(4). Retrieved May 8, 2011 from Computer Source.

McKenzie, C., & Wheaton, P. (2010). When design patterns are evil. *TheServerSide.com*. Retrieved April 16, 2011 from http://www.theserverside.com/tip/Evil-Design-Patterns-When-Design-Patterns-Become-the-Problem.

Moynihan, G. P., Suki, A., & Fonseca, D. J. (2006). An expert system for the selection of software design patterns. *Expert Systems* , Feb, *23*(1), 39-52. Retrieved Mar 30, 2011 from Computer Source.

Operational Efficiency (2011). Greenfield Software. Retrieved May 8, 2011, from http://www.greenfieldsoft.com/operational-efficiency.html

Project triangle. *Wikipedia (2011)*. Retrieved May 1, 2011, from http://en.wikipedia.org/wiki/Project_triangle

Ram, D. J., & Rajasree, M. S. (2003). Enabling design evolution in software through pattern oriented approach. *9th International conference Object-oriented information systems 2817: 179-190*. Retrieved April 16, 2011 from Computer Source.

Rising, L. (2010). The benefits of patterns. *IEEE software, 27*(5). Retrieved May 8, 2011 from Computer Source.

Shlezinger, G., Reinhartz-Berger, I., & Dori, D. (2010). Modeling design patterns for semi-automatic reuse in system design. *Journal of Database Management* , *21*(1), 29-57. Retrieved Mar 30, 2011 from Computer Source.

Schmidt, D. (1995). Experience using design patterns to develop reuseable object-oriented communication software. *Communications of the ACM*, *38*(10). Retrieved April 11, 2011 from GoogleScholar.

Schmidt, D., Fayad, M., & Johnson, R. (1996). Software patterns. *Communications of the ACM* , Oct, *39*(10), 36-39. Retrieved Mar 30, 2011 from UO library search.

SearchSOA. (2011). Retrieved April 25, 2011 from http://searchsoa.techtarget.com/definition/object-oriented-programming

SearchSoftwareQuality. (2011). Retrieved April 25, 2011, from http://searchsoftwarequality.techtarget.com/definition/pattern

Shlezinger, G., Reinhartz-Berger, I., & Dori, D. (2010). Modeling design patterns for semi-automatic reuse in system design. *Journal of Database Management* , Jan-Mar, *21*(1), 29-57. Retrieved Mar 30, 2011 from Computer Source.

Software development life cycle. *Wikipedia (2011)*. Retrieved May 8, 2011, from http://en.wikipedia.org/wiki/Systems_Development_Life_Cycle

Strategic management. *Wikipedia (2011)*. Retrieved May 8, 2011, from http://en.wikipedia.org/wiki/Business_strategy

Tichy, W. (1998). A catalogue of general-purpose software design patterns. UIUC Pattern Group Version Feb 12. Retrieved April 11, 2011 from http://www.laputan.org/pub/patterns/tichy/catalogue.pdf

Yacoub, S. A., Ammar, H. H., & Mili A. (2000). Constructional design patterns as reusable. *Software reuse: Advances in software reusability, 1844*, 369-387.

Wendorff, P. (2001). Assessment of design patterns during software reengineering: Lessons learned from a large commercial project. *IEEE Computer Society.* Retrieved May 1, 2011 from http://www.ptidej.net/teaching/ift6251/fall05/presentations/050919/Wendorff.pdf

Zimmer, W (1994). Relationships between design patterns. Retrieved May 8, 2011 from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.37.8779&rep=rep1&type=pdf