A Survey on Software Design Pattern Tools for Pattern Selection and Implementation

S.Saira Thabasum[1] and Dr.U.T.Mani Sundar[2]
Research Scholar, Mother Teresa Women's University[1] , Co-ordinator , University of Madras[2]
Email :naas2009@rediffmail.com[1],utmanisundar@yahoo.co.in[2]

Abstract

Designing object oriented software is hard and designing reusable object oriented software is even harder. A design should be specific to the problem at hand but also general enough to address future problems and requirements. Experienced object oriented designers make good designs whereas new designers are overwhelmed by the options available and tend to fall back on non-object oriented techniques they have used before. It takes a long time for novices to learn about good object oriented design. Expert designers don't solve a problem from the scratch. Rather, they reuse solutions that have worked for them in the past. This expertise can be captured in the form of patterns that can be used effectively by people. A design pattern can solve many problems by providing a framework for building an application. Design patterns make the design process cleaner and more efficient and also to reuse successful designs and architectures. It can help a designer to get a design right faster. But, a large amount of knowledge is associated with design patterns. Thus, expertise is needed to determine the suitable patterns for selection. Design patterns are a valuable tool for the practicing software professional. As the benefits of design patterns are becoming more and more evident, the use of design patterns to develop elements of reusable object-oriented software has become an emerging trend. This paper presents a survey on various tools proposed earlier in selecting design patterns suitable to the problem at hand. In addition, it provides some fundamental idea for future research that may help in eliminating the problems associated with the present tools for pattern selection and implementation.

Keywords:
Object-oriented, Software design, Design Patterns, Framework, Reusability

1. INTRODUCTION

Software development is a very complex process that at present is primarily a human activity. Design and Implementations are the two main phases of software engineering. In the software development process, design decisions have a significant impact on the quality of the product. These decisions can directly affect 65% of the work done in subsequent activities. Reckless execution of the engineering process can have a dramatic impact on the efficiency and quality of the final product. It is imperative for success that engineers consider all aspects of the challenge set before them and then craft a solution capable of handling any possible contingency. The solution must not be able to expect the unexpected, but do so in a timely manner and in a fashion useful to the final user of the system. A brilliant design that takes an hour to compute a solution is of little use to a user who needs the information in minutes. Carefully fashioned and well defined designs are the foundations of good engineering. Today object oriented languages are rapidly becoming the de facto solution for most general purpose programming problems. Object oriented methodologies are accepted as the standard analysis and design technique for new large scale software systems. While object oriented design methodologies and languages are in ever increasing use, it is becoming recognized that it is harder to become an expert Object Oriented programmer than it is to become an expert in traditional structured techniques. This is partly due to the fact that Object Oriented programming builds upon structured techniques and adds additional programming language and design features that must be comprehended and large libraries that must be learned. The unique nature of object oriented design lies in its ability to build upon four important software design concepts (Data Abstraction, Encapsulation, Information Hiding, and Modularity). Only OOD provides a mechanism that enables the designer to achieve all four with less complexity and compromise. Though OOD seems to be easier than alternative approaches, designing reusable software is a difficult task regardless of technique. A good reusable flexible design cannot

be achieved in the first attempt. Experienced designers may produce but, it is hard for a novice designer. Design patterns are a promising technique for achieving widespread reuse of software architectures.

Design patterns capture the static and dynamic 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. Patterns aid the development of reusable components and frameworks by expressing the structure and collaboration of participants in a software architecture at a level higher than source code or object-oriented design models that focus on individual objects and classes. Thus, patterns facilitate reuse of software architecture, even when other forms of reuse are infeasible (e.g., due to fundamental differences in operating system features).

The remainder of this paper is organized as follows. Section II discusses some of the earlier proposed research work on development and use of design pattern tools for effective design process. Section III provides a fundamental idea on which the future research work focuses on. Section IV concludes the paper with fewer discussions.

2. RELATED WORK

There are numerous works that suggests the benefits of knowing and using design patterns in the development process. Patterns are proven solutions for building software systems. This section of the paper discusses some of the relevant work proposed earlier in the literature illustrating the use of the software design pattern tools in development to achieve greater productivity.

Budinsky et al. (1996) in [1] developed a computer based tool that automates the implementation of design patterns. For a given pattern, the user of the tool inputs application-specific information, from which the tool generates the entire pattern prescribed code automatically. The tool incorporates a hypertext rendition of design patterns to give designers an integrated online reference and developmental tool. This automatic code generation enhances the utility of design patterns. This tool provides no help in the selection of design patterns, but once a pattern is selected, it can automate its implementation.

Kramer and Prechelt (1996) [8] have developed a system to improve maintainability of existing software. They have incorporated the structural design patterns identified by Gamma et al. (1995). In their approach, Kramer and Prechelt (1996) note that design information is extracted directly from Cþþ header files and stored in a repository. The patterns are expressed as PROLOG rules and the design information is translated into facts (Kramer & Prechelt 1996). A single PROLOG query is then used to search for all patterns. This tool demonstrates an artificial intelligence approach for discovering design patterns in existing software.

Eden et al. (1997) in [4] have presented a prototype tool that supports the specification of design patterns and their realization in a given program. The prototype automates the application of design patterns without obstructing modifications of the source code text from the programmer. The programmer may edit the source code text at will (Eden et al., 1997). The authors have described a prototype that supports the application of pattern specification language routines by the principles of the Meta programming approach. The automating tool maintains a library of routines and supports their modification, application and debugging. The tool also recognizes the need of the programmer for manual editing of the object program's source code.

Cinneide and Nixon (1999) in [2] have automated the transformations required to introduce design patterns in re-engineering legacy code. They have presented a methodology for the design pattern transformations and have constructed a prototype software tool, called DPT (Design Pattern Tool) that applies design pattern transformations to Java programs. Their methodology has been applied successfully to structure-rich patterns, such as Gamma et al.'s (1995) creational patterns.

Khriss et al. (2000) [7] have presented a pattern based approach to the correct stepwise refinement of UML static and dynamic design models by application of refinement schemas. They have also described an approach that allows for the construction of intelligent computer-aided software engineering tools that can provide automatic support for the selection, management and application of design patterns for the refinement of design models.

Tokuda and Batory [15] present an approach in which patterns are expressed in the form of a series of parameterized program transformations applied to software code.

Gu´eh´eneuc and Jussien developed an application of explanation-based constraint programming for the identification of design patterns in object-oriented source code.

Paulo Gomes et al in [12] developed REBUILDER a CASE tool that provides new functionalities based on CBR. They presented an approach to the selection and application of software design patterns in an automated way. Using CBR and WordNet they are able to store situations where design patterns were applied. These situations, called cases, can then be reused in similar situations to guide design pattern selection and application. It is based on the idea that a system can learn to select and to apply design patterns if it can store and reuse experiences that encode the situation in which patterns are used. It centralizes the corporation's design knowledge, and provides the software designer with a design environment capable of promoting software design reuse. This is achieved with CBR as the main reasoning process, and with cases as the main knowledge pieces. Experiments were performed to evaluate the performance of REBUILDER. They used a case-base of 60 DPA cases, each one presenting an application of a software design pattern to an UML class diagram. Each DPA case was generated from a different class diagram. For these experiments five software design patterns were used, the names of the patterns accordingly to [5] are: Abstract Factory, Builder, Composite, Singleton and Prototype. Each of these patterns is implemented in REBUILDER along with the participant's definition and operators. An advantage of this approach is the complete automation of the application of design patterns. One limitation of this approach is that the system performance depends on the quality and diversity of the case library. Another limitation, is that the range of case application is always restricted, and it does not outperform a software designer ability to identify which pattern to apply

ESSDP (Expert System for Suggesting Design Patterns) is a tool presented by Kung et al [10] that selects a design pattern through dialog with the software designer to narrow down the choices. They have implemented the twenty-three design patterns in Gamma et al's book. ESSDP engages the user in a question-answer session that helps narrow down the selection process. At the end of the process a suitable design pattern is suggested with a certainty. The ESSDP is a selection type of expert system and they have used a rule-base as the knowledge base. ESSDP is aimed to help a beginner to decide on which pattern to use for a particular design situation. It validates a designer's choice of a particular design pattern. The evaluation results of ESSDP have proved that it is more effective although many improvements are required to improve its success rate.

KARACAs (Knowledge Acquisition with Repertory Grids and Formal Concept Analysis for Dialog System Construction) is a knowledge acquisition tool developed by Hilke Garbe et al [16] that recommends software design patterns by asking critical questions. They have described a new knowledge acquisition tool that enabled them to develop a dialog system recommending software design patterns by asking critical questions. Their assistance system is based on interviews with experts. For the interviews they adopted the repertory grid method and integrated formal concept analysis. The repertory grid method stimulates the generation of common and differentiating attributes for a given set of objects. Using formal concept analysis they control the repertory grid procedure, minimizing the required expert judgments and built an abstraction based hierarchy of design patterns, even from the judgments of different experts. Based on the acquired knowledge they semi-automatically generate a Bayesian Belief Network (BBN), that is used to conduct dialogs with users to suggest a suitable design pattern for their individual problem situation. Integrating these different methods into the knowledge acquisition tool KARaCAs enables to support the entire knowledge acquisition and engineering process. They have used KARACAs with three design pattern experts and derived approximately 130 attributes for 23 design patterns.

Aliaksandr Birukou et al [11] presented a multiagent system SISC (Systems for Implicit Culture Support) that supports developers in choosing patterns that are suitable for a given design problem. The system implements an implicit culture approach for recommending patterns to developers based on the history of decisions made by other developers regarding which patterns to use in related design problems. They have implemented their approach within the IC-Service [4], which provides recommendations on patterns. The recommendations are created using a history of previous user interactions with the system. The viability of their approach is proved by the experimental systems and the system can be further enhanced to provide support for complex recommendation scenarios.

Muhammad Ali Babar and Ian Gorton [14] developed a tool PAKME (Process centric Architecture knowledge Management Environment) to support a framework for capturing and using architectural knowledge to improve the architecture process. They

developed PAKME to act as a knowledge source for those who need rapid access to experience based design decisions to assist in making new decisions or discovering the rationale for past decisions. PAKME serves as a repository of an organisation's architecture knowledge analogous to engineers' handbooks. PAKME captures design options as contextualized cases from literature or previous projects. A design option case consists of problem and solution statements, patterns and tactics used, rationale, and related design options. Rationale for each design option are captured in a separate template, which is designed based on practitioners' opinions about rationale reported in [18] and templates proposed in [19, 20]. By capturing design options as cases, PAKME enables architects to follow a case-based approach and supports human intensive case-based reasoning [21].

Gary P. Moynihan et al [13] developed a prototype expert system for the selection of design patterns that are used in object-oriented software. Their prototype expert system guides the designer through the pattern selection process via targeted inquiry regarding the nature of the specific design problem. The system also provides a means of browsing patterns and viewing the relationships between them. The prototype employs the rule-based method of knowledge representation within an object-based environment. The prototype incorporates five design patterns, which are considered to be a frequently encountered subset of patterns (Gamma et al., 1995). The experimental results of the prototype proves that it provides users with detailed guidance on which design pattern or which combination of patterns is suitable to solve the specified problem. A reasonable extension of this prototype is to enlarge the group of candidate design patterns.

J.Rajesh and D. Janakiram [17] presented a tool JIAD (Java Based Intent Aspects Detector) to infer design patterns in refactoring. They address the refactoring issues such as the scope for applying Design Patterns in the code and the appropriate selection of Design Patterns. Their tool automates the identification of Intent-Aspects that helps in applying suitable design patterns while refactoring the java code. Automating the process of identifying Intent Aspects enables rapid development of software systems. Also, their tool minimizes the number of possible errors while inferring the suitable Design Patterns to apply refactoring.

3. FUTURE ENHANCEMENT

Even though considerable advancement has been made to improve the software development process and the overall quality of systems and applications over the past decade, much remains to be done. A reasonable extension of this research is to enlarge the group of candidate design patterns. The increased number of design patterns will also raise the complexity of the knowledge base exponentially. Another future development is to develop more cognitive tools that could help the designer in her/his task, trying to perform tasks that are hard for the human, leaving more time for creative tasks.

4. CONCLUSION

Patterns in general are an appropriate means of reusing previously gained knowledge and facilitate the development process. Proper use of design patterns in software development can greatly increase the efficiency of the coding process. Patterns capture the static and dynamic aspects of successful solutions to problems that commonly arise when building software systems. If software is to become an engineering discipline, these successful practices and design expertise must be documented systematically and disseminated widely. Patterns are important tools for documenting these practices and expertise, which traditionally existed primarily in the minds of expert software architects. Over the next few years a wealth of software design knowledge will be captured in the form of strategic and tactical patterns that span disciplines such as client/server programming, distributed processing, organizational design, software reuse, real-time systems, game development, business and financial systems, and human interface design. Continuing research in design patterns promises to help application developers to build better applications.

REFERENCES

[1] BUDINSKY, F., M. FINNIE, J. VLISSIDES and P. YU(1996) Automatic code generation from design patterns, IBM Systems Journal, 35 (2), 151–171.

[2]. CINNEIDE,M. and P. NIXON (1999) A methodology for the automated introduction of design patterns, in Proceedings of the IEEE International Conference on Software Maintenance, New York: IEEE, 463–472.

[3]. CLINE, M. (1996), The pros and cons of adopting and applying design patterns in the real world, Communications

of ACM, 39(10), 47–49.

[4]. EDEN, A., J. GIL and A. YEHUDAI (1997), Automating the application of design patterns, Journal of Object Oriented Programming, 10 (2), 44–46.

[5]. GAMMA, E., R. HELM, R. JOHNSON and J. VLISSIDES (1995) Design Patterns: Elements of Reusable Object Oriented Software, Boston, MA: Addison-Wesley.

[6]. [tich02a] Tichy W. F., "Essential Software Design Patterns", ttp://wwwipd.ira.uka.de/˜tichy/ patterns/overview.html, 2002.

[7]. KHRISS, I., R. KELLER and I. HAMID (2000), Pattern based refinement schemas for design knowledge transfer, Journal of Knowledgebased Systems, 13 (6), 403–415.

[8]. KRAMER, C. and L. PRECHELT (1996), Design Recovery by Automated search for structural design patterns in object oriented software, Proceedings of the Third Working Conference on Reverse Engineering, New York: IEEE, 208–215.

[9]. [jacob99a] I Jacobson, Booch G. and Rumbaugh J., "The Unified Software Development Process." Addison-Wesley, Reading, MA, 1999.

[10]. Kung, D. C., H. Bhambhani, R. Shah, and G. Pancholi. 2003, An expert system for suggesting design patterns: a methodology and a prototype. In Software Engineering With Computational Intelligence, ed. T. M. Khoshgoftaar. Kluwer Int.

[11]. Birukou, A., E. Blanzieri, and P. Giorgini. 2006, Choosing the right Design Pattern: an Implicit culture approach. http://eprints.biblio.unitn.it/archive/00000960/. Technical report.

[12]. Paulo Gomes, Francisco C. Pereira and Paulo Carreiro, 2002. Using CBR for Automation of Design Patterns: Volume 2416 Springer Berlin

[13]. Gary P. Moynihan, Abhijit Suki and Daniel J. Fonseca. An expert system for the selection of software design patterns: Expert System Journal ,Vol 23, Issue 1.

[14]. Muhammed Ali Babar, Andrew Northway, Ian Gorton and Paul Heuer. Introducing Tool Support for Managing Architectural Knowledge: An Experience Report: Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, IEEE Computer Society.

[15]. Lance Tokuda and Don Batory : Evolving Object-Oriented Designs with Refactorings,Journal Automated Software Engineering,Vol 8, 2001.

[16]. Hilke Garbe, Claudia Janssen, Claus Möbus and Holger de Vries, Knowledge Acquisition with Repertory Grids and Formal Concept Analysis for Dialog System Construction, Springer Berlin, Volume 4248, 2006.

[17]. J.Rajesh and D. Janakiram, JIAD: a tool to infer design patterns in refactoring, Proceedings of the 6th ACM SIGPLAN international conference on Principles and Practice of Declarative programming,, 2004.

[18]. A. Tang, M. Ali-Babar, I. Gorton, and J. Han, A Survey of Architecture Design Rationale, Journal of Systems and Software, 2006. 79(12): pp. 1792-1804.

[19]. J. Tyree and A. Akerman, Architecture Decisions: Demystifying Architecture, IEEE Software, 2005. 22(2): Pp.19-27.

[20]. P. Clements, et al., Documenting Software Architectures:Views and Beyond. 2002: Addison-Wesley.

[21]. J.L. Kolodner, Improving Human Decision Making through Case-Based Decision Aiding, AI Magazine, 1991. 12(2): pp. 52-68.

[22]. James Rumbaugh, Object Oriented Modeling and Design. Prentice-Hall, Englewood Cliffs, NJ, 1991.