

PATTERNS APPROACH TO BUILDING SOFTWARE SYSTEMS

Satish Chandra
Consultant
Satyam Learning Center
Satyam Computer Services Ltd.
Hyderabad – INDIA

Satyendra Bhattaram
Technical Leader
Knowledge Initiative Group
Satyam Computer Services Ltd.
Hyderabad – INDIA

{satish_chandra, satyendra_bhattaram}@satyam.com

ABSTRACT

This position paper suggests an approach for building software systems using patterns, right from business architecture to software architecture. Further, the approach incorporates a concurrent and iterative development process to ensure that the business architecture and software architecture are aligned, end to end. Usage of patterns leads to reuse of various artifacts, involved in the software development life cycle.

1. INTRODUCTION

It is highly desirable to start the development of a software system with most of the system requirements being captured in the form of use cases. However, in practice systems evolve. New requirements crop up when the business on which the system is based keeps growing/changing. Also, development of a system gives rise to new ideas, which could be incorporated in the system. Thus, in practice, activities take place concurrently, rather than sequentially.

Nuseibeh [15] mentions that there are compelling economic arguments why an early understanding of stakeholders' requirements leads to systems that more closely meet these stakeholders' expectations. There are equally compelling arguments why an early understanding and construction of a software system architecture provides a basis for discovering further system requirements and constraints, for evaluating alternative design solutions and states that, in practice, software development starts from either requirements or software architecture [15].

It is obvious that the requirements, both functional and nonfunctional, are derived from business architecture. The underlying business architecture may be either implicit or explicit. Hence, we propose that the approach suggested by Nuseibeh could be extended to

business architecture, as well. It is apparent that the effect on requirements would have a cascading effect on the business architecture, and vice-versa. Sometimes, the effect of the software architecture, which led to modification of requirements, could even lead to business process re-engineering.

This article illustrates the linkages that would be there between various artifacts used in the development of a software system. We also suggest a concurrent and iterative process to software development.

2. BUSINESS ARCHITECTURE

The role of architecture in building any type of structure is well defined. A well-designed architecture makes it possible to thoroughly understand the structure being built, to plan the actual construction, and to estimate costs; it serves as the basis for the blueprints of the structure. Once construction has been completed, a good architecture remains as the documentation of the process and the result, making it possible to understand, maintain and, if so desired, to extend the structure [18].

An architecture captures the vital parts of a structure in an organized manner and is a practical tool for managing a complex system, such as a software system or a business. Business Architecture defines the business structure, so modeling this architecture is key to understanding the business and how it functions [18].

Eriksson and Penker [18] propose a business architecture description using four views: business vision, business process, business structure, and business behavior.

They state "The knowledge and information in a business architecture is used to define the software architecture. This isn't a one-to-one mapping, and there is no simple algorithm to convert the business model into a software model. They are two different models that serve different purposes. The business model describes a business or a specific part of a business; not all of the business goes into the software systems. To

define a software architecture, the business architecture is used to:

- Identify suitable support systems.
- Identify functional requirements.
- Identify nonfunctional requirements.
- Act as basis for analysis and design.
- Identify suitable components.”

Creating a business model before the software models, then using the information in that business model for the creation of software models, will increase the quality of the software systems. Systems that better support the business of which they are a part will be the result [18]. They have also catalogued some business patterns.

At times, it may so happen that while establishing the business goals it may be realized that some of the business processes may have to be changed, event to the extent of having a business process reengineering.

3. SOFTWARE ARCHITECTURE

A Software Architecture is a description of the subsystems and components of a software system and the relationships between them. Subsystems and components are typically specified in different views to show the relevant functional and non-functional properties of a software system. The software architecture of a system is an artifact. It is the result of the software design activity [8].

Buschmann [25] states: “According to its definition, a pattern system for software architecture should support building concrete software systems with help of patterns. Fortunately, many well-described patterns for software architecture already provide steps and guidelines that specify their implementation [GHJV95] [POSA1] [POSA2]. Many such patterns also provide information about their refinement and combination with other patterns. Whenever another pattern is referenced, its implementation steps can be applied: they thus complement and complete the implementation steps of the original pattern.” He also sees some drawbacks in application of patterns to certain areas like partial design and suggests improvements in the application of patterns in building software architecture.

4. FROM BUSINESS ARCHITECTURE TO SOFTWARE ARCHITECTURE

UML is a de-facto as well as de-jure standard for modeling object-oriented systems. The business architecture could be developed by using UML, as suggested by Eriksson and Penker. The software architecture could also be modeled using the UML or any Architecture Description Language. Between the two ends we could consider the use cases for capturing and documenting requirements, the class diagrams to capture the analysis models as well as design models. The other diagrams of the UML could be used to support the development process.

This paper suggests that the conceptual class diagrams derived from the use cases be refined. It is suggested that the conceptual class diagrams be refined using the classes that can be derived from the analysis patterns of the specific domain or the generalized analysis patterns which apply to the domain under consideration. It may be mentioned that a good number of analysis patterns have been documented covering domains like insurance, virtual libraries, oil & refinery and so on [21,22,23].

It would involve some experience or training to understand that a transaction would give birth to an association class, which has to be identified and documented as an analysis class. Similarly, analysis patterns help in identification of additional classes. This would help in refining the conceptual class diagrams.

The other suggestion is with regard to the design class diagrams. The design class diagrams which are identified in the system could be refined using the design patterns. Apart from the design patterns documented by Gamma and others there are design patterns specific to technologies like EJB, J2EE and so on. These would help in refining the class diagrams.

Taking into consideration the existing architectural patterns/styles could refine the identified software architecture.

Thus, the software architecture, which is a composition of patterns, is derived from the business architecture.

Each of the artifacts would affect the other, as shown in figure 1 (Appendix IV).

5. SOFTWARE DEVELOPMENT PROCESS

The Unified Process describes process workflows as: business modeling, requirements, analysis & design, implementation, test and deployment. It describes configuration & change management, project management and environment as the supporting workflows.

The waterfall methodology describes the various phases of software development cycles as; requirements gathering, analysis, design, coding, testing and maintenance.

The phases of the waterfall model or the workflows suggest sequencing of activities. However, in practice we experience that activities in the phases or workflows happen concurrently.

We strongly feel that the concurrent and iterative development approach presented in [14] and [15], using the Twin Peaks, is close to reality. We suggest that the approach be extended to Three Peaks, with the addition of the business architecture as the third peak. The modification is as given in figure 2 (Appendix V).

6. A Case Study

We had studied a knowledge management system, developed in-house, to identify and discover analysis and design patterns in the system [1]. The identified and discovered analysis and design patterns are presented in Appendix II and Appendix III respectively.

Further, study of the system in identifying the business patterns led to the identification of the business patterns presented in Appendix I.

As regards the solution architecture of the system, our study shows that the System follows the MVC architecture. We have used the Microsoft's ASP technology.

7. CONCLUSIONS

We advocate the continuing of use cases for capturing requirements. However, use case driven approach leads to identification of classes, with boundary, control and entity stereotypes. The conceptual class diagram, which has thus been arrived at, would be similar to a design class diagram without the application of design patterns. Hence, the analysis patterns of the domain or generalized analysis patterns could be used to refine the conceptual class diagram. This would enable creation of rich conceptual class diagrams. The conceptual diagrams could lead to the questioning of business processes and requirements.

The design class diagrams could be refined using design patterns. This would lead to development of systems, which would address the nonfunctional attributes like flexibility, scalability, maintainability, etc..

Some of the issues that need to be addressed are:

Training the various stakeholders.

Analysis patterns for more domains have to be developed.

This methodology has to be validated.

8. REFERENCES

1. Satish Chandra, Suri Ponnada and Satyendra Bhattaram: Pattern Oriented Architecture for a Knowledge Management System – In proceedings of Informatica 2003, Hyderabad, India, January, 2003.
2. Geyer-Schulz and Hahsler: Software Reuse with Analysis Patterns – In proceedings of AMCIS 2002, Dallas, TX, August 2002
3. Fowler, M. *Analysis Patterns: Reusable Object Models*, Object Technology Series. Addison-Wesley Publishing Company, Reading, 1997.
4. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional Computing Series, Addison-Wesley Publishing Company, New York, 1995.
5. Martin Fowler: Patterns of Enterprise Application Architecture. www.martinfowler.com
6. Moore, M.M., "Software Reuse: Silver Bullet?" *IEEE Software* (18:5), September/October 2001, p. 86.
7. Ulrich Frank, Knowledge Management Systems: Essential Requirements and Generic Design Patterns. Proceedings of the International Symposium on Information Systems and Engineering, ISE'2001, Las Vegas: CSREA Press 2001.
8. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. *Pattern-Oriented Software Architecture, A System of Patterns*, John Wiley & Sons Ltd, Chichester, 1996.
9. Al Williams, Design Patterns for Web Programming Do you need MVC?, New Architect June, 2002.
10. Dick Stenmark, Turning Tacit Knowledge Tangible. Proceedings of the 33rd Hawaii International Conference on System Sciences – 2000.
11. Neighbors, J.M., "The Draco Approach to Constructing Software from Reusable Components," *IEEE Transactions of Software Engineering* (10:5), September 1984, pp. 564-574.
12. Francisco Montero, María Lozano, Pascual González, Isidro Ramos: Designing Web Sites by Using Design Patterns.
13. Paola Inverardi & Henry Muccini & Patrizio Pelliccione: Checking consistency between architectural models using using SPIN. STRAW 2001.
14. Jon G. Hall, Michael Jackson, Robin C. Laney, Bashar Nuseibeh and Lucia Rapanotti: Relating Software Requirements and Architectures using Problem Frames. STRAW 2001.
15. Bashar Nuseibeh : Weaving the Software Development Process Between Requirements and Architectures. STRAW 2001.
16. Paul Evitts : UML Pattern Language. Techmedia publications.
17. Martin Fowler & Kendall Scott :UML Distilled. Addison Wesley.
18. Hans-Erik Eriksson: Magnus Penker Business Modeling with UML: Business Patterns at Work. John Wiley & Sons. 2000.

19. Grady Booch; James Rumbaugh; Ivar Jacobson : The UML User Guide. Addison Wesley. 1999. of understanding and optimizing this communication.
20. Philippe Kruchten: The Rational Unified Process, An Introduction. Second Edition. Addison Wesley. 2000.
21. Prashant Jain and Michael Kircher: Leasing. PLoP 2000.
22. Wolfgang Keller: Some Patterns for Insurance Systems. PLoP 98.
23. Lei Zhen and Guangzhen Shao: Analysis patterns for oil refineries. PLoP 2002.
24. Ivar Jacobson, Martin Griss, Patrik Jonsson: Software Reuse – Architecture, Process and Organization for Business Success. ACM Press. Pearson Education Asia. Addison Wesley Longman. 2000.
25. Frank Buschmann. Building Software with Patterns. EuroPloP 99.

APPENDIX I - Business Patterns

Resource and Rules Patterns:

1. Actor-Role: Provides guidelines for using actor and role concepts, including how they should be separated and how they can be combined.
2. Organization and Party: Used to create flexible and qualitative organizational (processes)charts in object-oriented models.
3. Product Data Management: All businesses have many products and/or documents that must be organized and structured. Capturing the structure of the relationship between documents and products is a difficult but common problem in all businesses.
4. Thing – Information: Eliminates the focus-shifting that occurs during the modeling process by referring to two frequently used foci(thing focus and information focus) in business modeling and how they are related to each other.
5. Title-Item: Helps modelers to simplify the design process for systems that involve objects that exist in multiple copies or instances. It separates the information about the title from the information about individual instances of that title.
6. Type-Object-Value: Models the relationships between a type, its Object, and value.

Goal patterns:

7. Business Goal – Problem: Used to identify the connection between business goals and their related problems in order to correct the problems and achieve the goals

Process Patterns:

8. Action Workflow: A tool for analyzing communication between parties, with the purpose

APPENDIX II - Analysis Patterns

The following analysis patterns have been identified in the System:

1. Recurring event pattern: (Fowler [3]).
2. Individual instance method (Fowler [3]).
3. Effectivity analysis pattern (Fowler [3]).
4. Range analysis pattern (Fowler [3]).
5. Structured Pin Board (Hahsler [2]).

The following analysis patterns have been discovered in the System:

1. Push Pull Analysis Pattern and
2. Collaborative Problem Solving Analysis Pattern.

APPENDIX III - Design Patterns

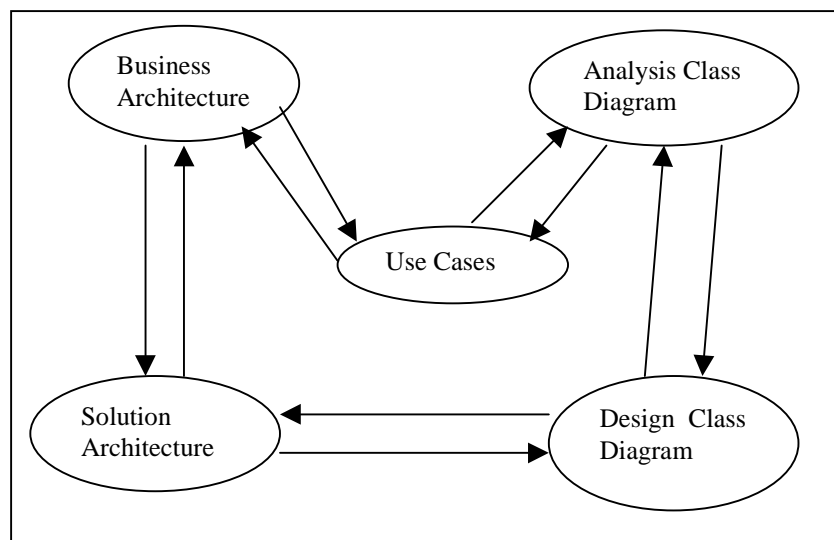
During our literature survey, we had come across two types of design patterns. The first dealing with the user interface and the second dealing with the functionality of the System. We have used all the UI Design Patterns mentioned in [12]. We have used 3 design patterns of GoF [4] and 12 design patterns from Fowler [5].

The identified design patterns from the Gang of Four are: Decorator, Iterator and Facade.

The identified design patterns from Fowler are: Front Controller, Two Step View, Server Session State, Gateway, Mapper, Service Layer, Recordset, Data Access Object, Transaction Script, Domain Model, Table Module and Active Record.

APPENDIX IV

Figure 1 – Relationship between the various artifacts:



APPENDIX IV

Figure -2: Three Peaks – a model of the concurrent development of business architecture, requirements and software architecture

