ENHANCING REUSABILTY IN AGILE SOFTWARE DEVELOPMENT USING OBJECT ORIENTED PATTERNS

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering

in

Software Engineering

By:

Ruchika

(800831022)

Under the supervision of:

Ms. Ashima Singh

Assistant Professor(CSED)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004

July 2010

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Enhancing Reusability in Agile Software Development using Object Oriented Patterns"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Ms. Ashima Singh* and refers other researcher's works which are duly listed in the bibliography section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

(Ruchika)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

15/7/10

(Ms. Ashima Singh)
Assistant Professor
Computer Science and Engineering Department
Thapar University
Patiala

Countersigned by:

16/07/10

(Dr. Rajesh Bhatia)
Head
Computer Science & Engineering, Department
Thapar University
Patiala

29.7.10

(Dr. R.K. Sharma)
Dean (Academic Affairs)
Thapar University
Patiala

Acknowledgement

I would like to start by thanking GOD ALMIGHTY for providing me with faith, self-confidence, ability and strength to complete this thesis.

I would like to place on record my deepest gratitude to Ms. Ashima Singh, Assistant Professor(CSED), Thapar University, Patiala for her valuable advice and guidance in carrying out the thesis. I appreciate her unlimited help and patience with her students. I will always remain indebted to her for the moral support, encouragement and the infectious zeal and enthusiasm for work that she imbibed upon me.

With great pleasure and acknowledgement, I extend my profound thanks to Dr. Rajesh Bhatia, Assistant Professor and Head (CSED), Thapar University, Patiala who has been a constant source of inspiration for me throughout this work.

I am also thankful to all the staff members of the Department for their full cooperation and help.

Thanks are also due to my friends, my husband , my mother-in-law for boosting me with their constant encouragement and confidence and bearing me, day and night throughout the thesis work.

Lastly, I am thankful to my parents for their blessings and moral support without which this thesis would not have been possible.

Ruchika

Abstract

Software Engineering Discipline is continuously gaining momentum from past two decades. In last decade, tremendous growth had been observed in the software industry. New process models are introduced time to time in order to keep pace with multidimensional demands of the industry. New software development paradigms are finding its place in industry such as Agile Software Development, Reuse based Development and Component based Development. But unfortunately, different software development models fail to satisfy the needs of software industry. As aim of all the process models is same i.e. to get quality product, reduce time of development, productivity enhancement and reduction in cost. Still, no single process model is complete in itself.

Software industry is moving towards Agile Software Development. Essence of Agile Software Development is rapid development and less cost. Thus, it somewhere compromises with quality and also unable to provide reusability of its developed components. Agile Software Development provides specific solutions whereas Reuse and Component based Development believe in generalized solutions. Both have same target but different approach. Three layered OO-framework Reusability-Pro Agile Software Development Model is proposed. The first layer depicts Reusability-Pro Agile Software Development Cycle and the second layer elaborates its Reusability-Pro Agile Software Processes. Third layer has Agile Repository. This model incorporates generalization as well as specialization due to proposed UML's extend and uses based design patterns. These object oriented features of the proposed model gives the Agile Software Development the essence of Reusability.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Agile Software Development

Agile Software Development (ASD) methods and techniques are being followed in the industry from the last decade to get quality product and to reduce development time. Rapid development and accommodate changes at any level of development gives the competitive advantage to the agile processes over traditional processes. But to get best and being new in software engineering, research on agile processes is going on as to combine light-weight processes and other processes. Recent research shows that only limitation of ASD is its inability to reuse components those are developed through agile processes. On the whole rapid software development ignores reusability.

[1] reveals that Japanese projects also exhibited higher levels of reuse while spending more time on product design as compared to American teams which spend more time on actual coding and concludes that Indian firms are doing great job in combining conventional best practices, such as specification and review, with more flexible techniques that should enable them to respond more effectively to customer demands. If such a trend is replicated across the broader population, it suggests the Indian software industry is likely to experience continued growth and success in future.

1.1.1 Evolution

Figure 1.1 shows the evolution of software development processes. Waterfall model was being followed where requirements are fixed and the next phase starts when the earlier one finished. It's the representative of the traditional methods. To overcome the limitations of waterfall model evolutionary model and spiral model comes into picture where prototype is first made and then that is converted to the working software. But all have one common limitation that no process could handle the change of requirements at later phases.

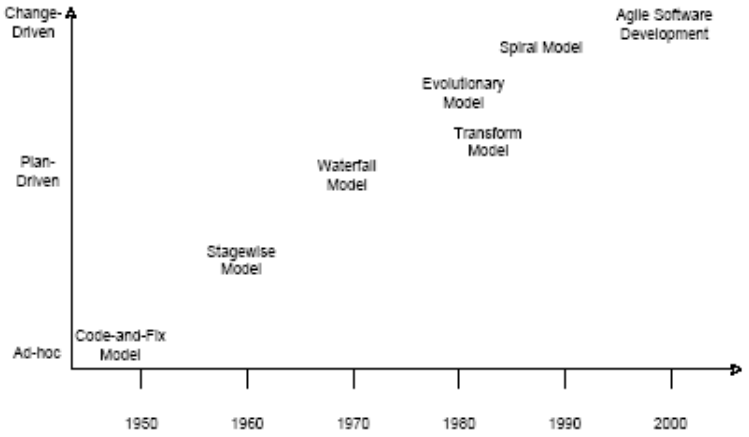


Figure 1.1: The Evolution of Software Process Models [2]

Agile development which includes many methodologies as XP, SCRUM, Lean Software Development, Feature Driven Development (FDM), Test Driven Development etc. is being accepted in industry because of adaptation to change even at the later stages of the development and also for rapid development.

1.1.2 Agile Values and Principles

For any method to be agile, the values and principles of the Agile Manifesto[35], which set out the central elements of agility should be followed. "*We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to values:*

Individuals and interactions over *processes and tools*

Working software over comprehensive documentation

Customer collaboration over *contract negotiation*

Responding to change over *following a plan*

That is, while there is value in the items on the right , we have the items on the left more." [35]

The twelve principles of agile software development are:

- The highest priority is to satisfy the customer through early and continuous delivery of valuable software,
- the welcoming of changing requirements, even late in development, for the benefit of the customer's competitive advantage,
- frequent delivery of working software, the release cycle ranging from a couple of weeks to a couple of months, with a preference for a shorter timescale,
- daily collaboration of business people and developers throughout the project,
- building of projects around motivated individuals by offering them an appropriate environment and the support they need, and trusting them to get the job done,
- emphasis on face-to-face conversation for conveying information and within a development team,
- working software is the primary measure of progress,
- agile processes promote a sustainable development pace for the sponsors, developers, and users,
- continuous attention to technical excellence and good design enhances agility,
- simplicity is essential for maximising the amount of work not having to be done,
- self-organising teams give best results in terms of architectures, requirements, and designs,
- regular reflection of teams on how to become more effective, and tuning and adjusting its behavior accordingly.

1.1.3 Agile Methodologies

Extreme Programming (XP), one of the agile software development models, brings together some new software principles to develop software based on 'user stories' in a vague and constantly changing environment. In XP projects, the software development life cycle is characterized by small iterations. Iterations each address a collection of stories, for each story an acceptance test is written, then a solution is designed and coded. XP centers on small teams realizing four values (communication, simplicity, feedback, and courage) and implementing twelve practices. These practices are: the planning game, small releases, metaphor, simple design, test first, re-factoring, pair programming,

collective ownership, continuous integration, 40 hour week, on-site customer, and coding standards.

SCRUM development assumes and addresses changing environmental and technical variables. Its main focus is on the organization of a software team to influence software projects' success in a changing environment. SCRUM's software project life cycle mirrors a rugby game, where there are three phases: Pre-game, Development, and Post-game. The pre-game is characterized by planning; early plans produce prioritized requirements, the system architecture, and a high-level design. Development follows in iterative cycles called 'sprints'. In each sprint a part of the system is expected to be delivered. The post-game follows once there is full customer-team agreement on the system's functionality. In this phase, no additional functionality is presented. Scrum projects involve regular project management activities aimed at removing obstacles which can influence deficiencies.

Feature-Driven Development (FDD) emphasizes software process management in the early stages of development, focusing its agility primarily on the design and building phases .The main focus in FDD software development is on quality and rapid delivery. This is rarely achieved without careful monitoring of project processes explain that the FDD project life cycle has five phases: overall development planning phase, building features list phase, features planning phase, design by feature phase, and build by feature phase. Planning and process management are particularly evident in the first three phases, agile development then follows in the remaining phases, implementing a life cycle resembling that of XP.

1.1.4 Agile Approaches

Refactoring: Refactoring is a fundamental to Agile Development. Refactoring is a development process for restructuring an existing code, altering its internal structure without changing its external behavior. It's a process of improvement to an existing software artifact. It improves the design of the software and makes software easier to

understand. It helps to avoid errors and to maintain, and modify a program with more accuracy and speed.

Various approaches include extract, pull up, and push down, form templates, parameterization, move etc. to convert bad smells to structured code. Code duplication is the main cause for bad smells in code. Other bad smells founded are Large classes, nested if statement, switch statement, dependency cycle, long parameter list etc. Various development tools have refactoring feature. Ecllipse (java), Jbuilder (Java), ReSharper (.NET), Refactor for Visual Basic are a few names.

TDD (Test Driven Development): TDD is one of the most profound agile development practices. It TDD is an evolutionary approach to development which instructs to have TFD (Test First Design) intent.

Basically, it starts by writing a test and then coding to elegantly fulfill the test requirements. TDD actually helps to meet deadlines by eliminating debugging time, minimizing design speculation and re-work, and reducing the cost and fear of changing working code.

TDD starts with writing a test to fail and then coding is being done to pass that test. If written code pass the test then code refactoring is being done otherwise again the code is being written and tested. It's a way to do unit testing. The cycle is repeated till the dead end. xUnit framework family of open source tools is a very common used tool support for TDD in agile development.

TDD is basically the combination of Test First Design and Refactoring [3]. Figure 1.2 elaborates steps of Test first Design

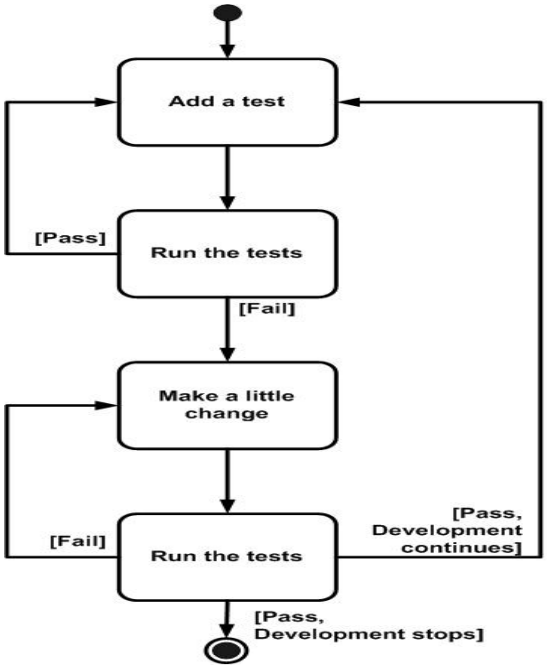


Figure 1.2: Steps of Test First Design [3]

1.1.5 Critical Success Factors in ASD

Following are the Critical Success Factors and its associated entities shown in Table 1.1

Factors	**Entities**
Project	Project Type Project Schedule
Process	Project Management Process
Organizational	Team Environment Management Environment
Technical	Agile Software Techniques Delivery Strategy
People	Team Capability Customer Involvement
Product	Specialized

Table 1.1: Critical Success Factors

Here, Figure 1.3 shows how all these factors taken collectively can boost the rapid development.

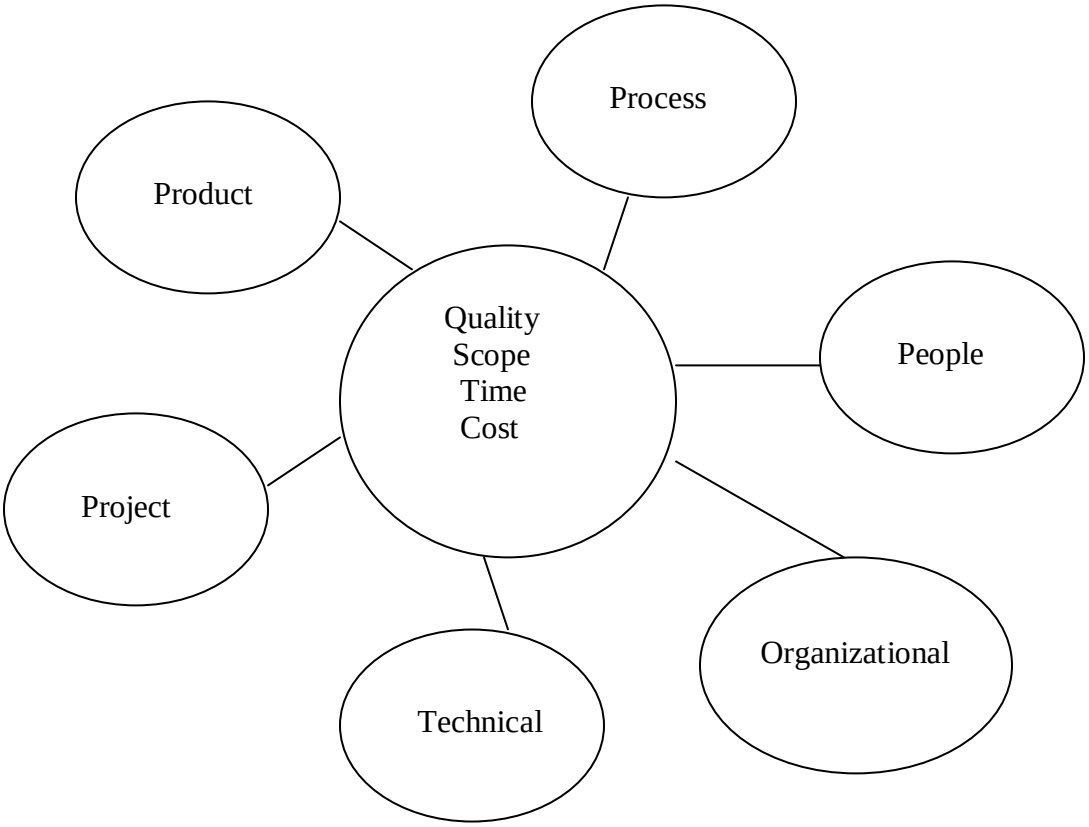


Figure 1.3: Association of Critical Success Factors and Constraints

1.1.6 Limitations and Bottlenecks in Agile Development

These days extensive research is being carried out to get best of agile development as [4] discusses that no agile process follows all the principles. Lean software development has five bottlenecks, XP itself has two, SCRUM has two and FDD also has seven bottlenecks.

As many authors say though agile development becomes industry standard but agile processes also have limitations as [5] discusses the limitations of agile on its 11 assumptions which says none of the agile processes is a silver bullet to fit all these assumptions.

Among the limitations of agile methods mentioned in [6] one is the lack of attention to design and architectural issues. Boehm in an article pointed out that there is a risk of architectural mistakes that cannot be detected easily by external reviewers due to lack of documentation in agile development.

1.2 Agile Software Development and Other Techniques

To get best traditional approach and agile approach has to combine. [7] says that the companies quite expertly combine agile and traditional practices and adjust their practices according to the situation at hand. Figure 1.4 shows the effects (benefits or drawbacks) in both the methods and also insists on the cumulative methods development since there has been a movement from no methods, via traditional method to agile method.

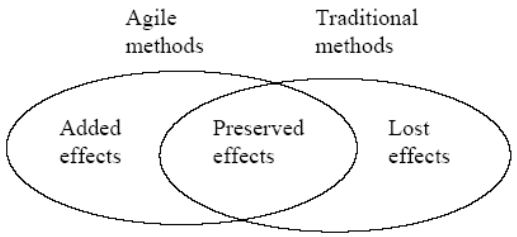


Figure 1.4: Relation between Identified Effects [8]

Research is going on to combine agile with other processes and models .One is in [9] which concludes through a case study that Software Product Line Engineering (SPLE) and agile software development are complementary to each other.

Following Table 1.2 shows a risk based approach to develop a balanced development strategy. [10] discusses about the process which appears to be generic i.e. amenable to use for building any type of system, including web applications; in a context where risk analysis is important.

Step 1.	Rate the project's environmental, agile, and plan-driven risks. If uncertain about ratings, buy information via prototyping, data collection, and analysis.
Step 2a.	If agility risks dominate plan-driven risks, go Risk-based Plan-driven.
Step 2b.	If plan-driven risks dominate agility risks, go Risk-based Agile.
Step 3.	If parts of the application satisfy 2a and others 2b, architect the application to encapsulate the agile parts. Go Risk-based Agile in the agile parts, and Risk-based Plan-driven elsewhere.
Step 4.	Establish an overall project strategy by integrating individual risk mitigation plans
Step 5.	Monitor progress and risks/opportunities, readjust balance and process as appropriate.

Table 1.2: Summary of Risk Based Method [11]

1.3 Agile Software Development and SPI (Software Process Improvement)

One key research area related to agile processes is in software process improvement. [12] reflects in Table 1.3 the differences between software improvement initiatives of Traditional and Agile development.

	Traditional software development and SPI	Agile software development and SPI
Software development process	Universal approach and repeatable solution to provide predictability and high assurance	Flexible approach adapted with collective understanding of contextual needs to provide faster development times, responsiveness to rapid changes, increased customer satisfaction, and lower defect rates.
Process control	Control on organizational level	Self-organizing teams
Primary means of knowledge transfer	Document based knowledge transfer	Face-to-face communication
Immediate focus of process improvement	Improvement of organizational software development processes/(future projects)	Improvement of daily working practices of ongoing project

Table 1.3: Underlying differences of Traditional and Agile Software Development and SPI [12]

1.4 Thesis Outline

The Chapter 1 highlights Agile Software Development, its related methodologies, approaches, critical success factors and inherent limitations.

In Chapter 2, Literature Review of Object Oriented Patterns and Architectures has been carried out. Relationship between Reusability and Agility is studied. Risk analysis of integration of reusability and agility is done.

Chapter 3 concentrates on problem statement i.e. what is the actual problem and how it can be solved. Here objectives of study are set and the need of study is found.

In Chapter 4, three layered Reusability-Pro Agile Software Development Model is proposed.

Chapter 5 is the implementation of Reusability-Pro Agile Software Development Model through a case study of Hotel Reservation System.

Chapter 6 conveys the conclusion and the future scope of our work.

Chapter 2
Literature Review

2.1 Patterns

A Pattern is a known solution to a recurring problem. Patterns are about design and interaction of objects, as well as providing a communication platform concerning elegant, reusable solutions to commonly encountered programming challenges.

Many fields use patterns in various ways: In music and literature, a pattern is the coherent structure or design of a song or book. In art, a pattern is the composition or plan of a work of graphic or plastic art. In architecture, a pattern is an architectural design or style. In archeology, a pattern is a group of phases having several distinguishing and fundamental features in common. In linguistics, a pattern is the manner in which smaller units of language are grouped into larger units. In dressmaking, a pattern is a pleasing shape that is applied repeatedly. In decorating, a pattern is a design or figure appearing in furniture or an accessory. In manufacturing, a pattern is the shape or style of a manufactured form. In aviation, a pattern is a collection of approaches, turns, and altitudes prescribed for an airplane that is coming in for a landing. In broadcasting, a pattern is a standard diagram for testing television circuits. In numismatics, a pattern is a specimen of a proposed coin or coin design. With each pattern, small piecework is standardized into a larger chunk or unit. Patterns become the building blocks for design and construction. Finding and applying patterns indicates progress in a field of human endeavor. Patterns are more than just the smallest elements in an endeavor [13].

2.1.1 Design Patterns

A design Pattern shows the interaction among various components and objects. Design Patterns are “descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context”. Patterns of the Gang of Four (GoF) pattern catalogue are generally considered the foundation for all other patterns.

These are 23 ***Object Oriented design patterns*** categorized in three groups: Creational, Structural, and Behavioral [14] has shown in table 2.1.

Pattern can be implemented by any Object Oriented language like C#, JAVA, NET, PHP. Other pattern catalogues are POSA, SOA, Enterprise, and Head First. Java's own design patterns can be found in J2EE pattern catalogue like business delegate, DAO, MVC etc. Patterns can be related to each other but mind it not every pattern can be related to each one.

Developers can find their own patterns and can make their own catalogues. To find a pattern is a complex task but sometimes it becomes a matter of judgment and experience too. Name of the pattern should be understandable.

Creational Patterns	
Abstract Factory	Creates an instance of several families of classes
Builder	Separates object construction from its representation
Factory Method	Creates an instance of several derived classes
Prototype	A fully initialized instance to be copied or cloned
Singleton	A class of which only a single instance can exist

Structural Patterns	
Adapter	Match interfaces of different classes
Bridge	Separates an object's interface from its implementation
Composite	A tree structure of simple and composite objects
Decorator	Add responsibilities to objects dynamically
Façade	A single class that represents an entire subsystem
Flyweight	A fine-grained instance used for efficient sharing
Proxy	An object representing another object

Behavioral Patterns	
Chain of Resp.	A way of passing a request between a chain of objects
Command	Encapsulate a command request as an object
Interpreter	A way to include language elements in a program
Iterator	Sequentially access the elements of a collection
Mediator	Defines simplified communication between classes
Memento	Capture and restore an object's internal state
Observer	A way of notifying change to a number of classes
State	Alter an object's behavior when its state changes
Strategy	Encapsulates an algorithm inside a class
Template Method	Defer the exact steps of an algorithm to a subclass
Visitor	Defines a new operation to a class without change

Table2.1 Gang of Four Patterns Catalogue

Example of Abstract Factory Pattern: Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

Participants: The classes and/or objects participating in this pattern are;

Abstract Factory: declares an interface for operations that create abstract products

Concrete Factory: implements the operations to create concrete product objects

Abstract Product: declares an interface for a type of product object

Product: defines a product object to be created by the corresponding concrete factory. Implements the abstract product interface

Client: Uses interfaces declared by abstract factory and abstract product classes.

Examples will make it clear how the patterns increase the reusability.

Class Diagram:

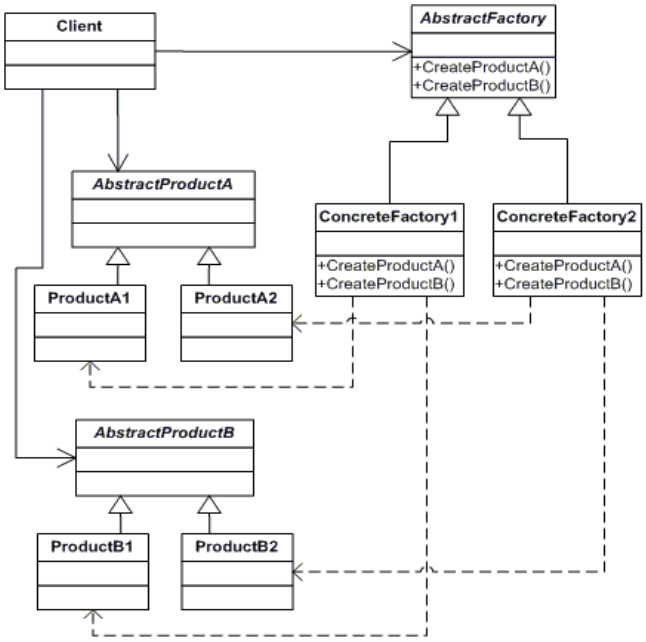


Figure 2.1: Class Diagram of Abstract Factory Pattern

Case Study 1: FullAddressFactory

Abstract Factory: FullAddressFactory
Concrete Factory1: PbFullAddressFactory
Concrete Factory 2: MhFullAddressFactory
Abstract ProductA: Address
Abstract ProductB: PhoneNo
ProductA1: PbAddress
ProductA2: MhAddress
ProductB1: PbPhoneNo
ProductB2: MhPhoneNo
Client : AddressProcessor

Case Study 2: FinanceFactory

Abstract Factory: FinanaceFactory

Concrete Factory1: PbFinanceFactory

Concrete Factory2: MhFinanaceFactory

Abstract ProductA: TaxCal

Abstract ProductB: ShipmentCal

ProductA1: PbTaxCal

ProductA2: MhTaxCal

ProdcutB1: PbShipmentCal

ProductB2: MhShipmentCal

Client: OrderProcessor

2.1.2 Architectural Patterns

An architectural pattern is a description of element and relation types together with a set of constraints on how they may be used . [15] Its basically an integration of components and are at higher level than design patterns and have larger scope, usually describing an overall pattern followed by the entire system. Sometimes combination of patterns is also used to construct architecture for a system. There are some styles like Pipe and filter, Layered, Blackboard, and Implicit Invocation etc. In most cases architectural patterns are considered in close connection with object orientation. Object Oriented language constructs like abstract classes or inheritance, which support the architectural pattern idea in a very elegant way.[16]

Architectural pattern is being decided on the non-functional requirements of the product. Single pattern or combination of patterns is being used to design the architecture by the architect by keeping in mind the hindrance to each of the non-functional properties because of one another.

Architectural patterns are very few while comparatively design patterns are available in bundle. Both are used for design purposes but architectural patterns are used in large scale and design patterns are used in small scale design solutions and are mainly localized.

2.1.3 Benefits of using Patterns

As patterns are already fully tested and can be easily adapted and enhances the reusability. Other noticeable benefits are:

1. Communication: Patterns promote a common vocabulary among designers
2. Patterns provide reuse at the design level
3. Community review: Pattern implementations tend to be standardized
4. Patterns simplify documentation
5. Patterns give beginners immediate "experience"
6. Patterns help control "essential complexity"

2.2 Architecture

Software architecture of a program or computing system is the structure or structures of the system which comprise software elements, the externally visible properties of those elements and the relationships among them [15].

Object-Oriented architecture is an architecture in which everything (processes, files, I/O operations, etc.) is represented as an object. Objects are data structures in memory that may be manipulated by the total system (hardware and software); they provide a high-level description that allows for a high-level user interface [17]. An object-oriented software architecture is one which is based on an object and which contains the global behavior of the system and centralizes all the interactions between the software objects [18].

Class type Architecture[19] which is based on common architectural strategy, some might call it a pattern, is to layer the architecture of a system into several layers. Some strategies simply define N layers stacked on top of each other where layer J interacts only with layers J-1 and J+1 . This approach first used object-oriented (OO) technology, but now these days it is used for component-based architectures, service oriented architectures (SOAs), and combinations thereof. To implement the layers, OO technology is being used i.e. classes within the layers.

2.3 Reusability

To share code between different applications is considered to be the reusability but a variety of assets can be reused across software development processes as shown in Table 2.2.

Intermediate Artifact	**Implemented Artifact**	**Project Management and Quality Assurance Artifact**
Requirements	(Sub) Systems	Process Models
Architectures	Frameworks, Components, Modules, Packages	Planning Models
Designs	UML Models, Interfaces, Patterns	Cost Models
Algorithms	Libraries	Review and inspection Forms e.g. checklists
Documentation (including templates)	Test Cases	Analysis Models e.g. Performance ,reliability
	Classes, Procedures, Routines, Functions, Methods, Source Code , Data	Design and Coding Conventions

Table 2.2: Possible Assets for Reuse [20]

Reusability increases not only the productivity of the developers but also the reliability and maintainability of the software products. Many software companies have repository to support the reusability. Object-Orientation also offers reusability.

No of techniques are available to support reusability. Considerable research and development is going on in reuse; industry standards like CORBA have been created for component interaction; and much domain specific architecture , toolkits, application generators and other related products that support reuse and open systems have been developed [21]. Reuse Landscape as shown in figure 2.2 depicts the different reuse approaches.

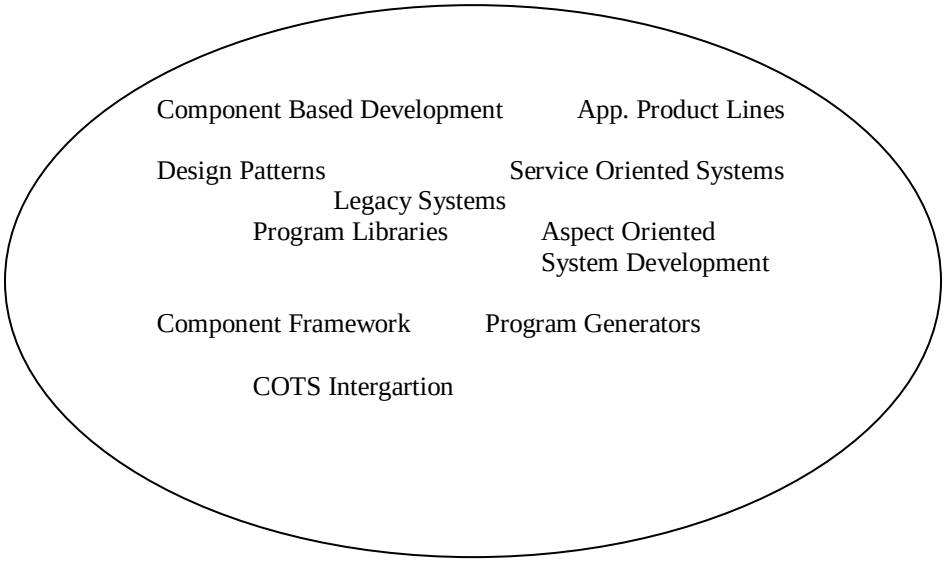


Figure 2.2: Reuse Landscape [22]

2.4 Reusability in Agile Software Development

There are three ways or technologies discussed one by one below, by which reusability can be incorporated in agile software development.

2.4.1 Component Based Development (CBD)

CBD is a reusability approach that can be found in Microsoft.NET Framework and J2EE (Java2 Enterprise Edition). Component Based Software Engineering(CBSE) process identifies not only candidate components but also qualifies each components' interface, adapts components to remove architectural mismatches, assembles components into selected architectural style, and updates components as requirements for the system change [20].

2.4.2 Refactoring to Design Patterns

To provide a software system quality in terms of reusability, flexibility and extendibility, refactoring is a significant solution.

Tool	Approach	Pattern Category	
		Structural	Behavioral
JIAD[23]	Intent Aspects		
[24]	Complex Refactoring		
Ptidej[25]	Algorithmic	Yes	
Crocopat[26]	Parsing+RSF	Yes	
DPRE[27]	Parsing	Yes	
DPVK[28]	Parsing+DB	Yes	Yes
PINOT[29]	Lightweight Static Program Analysis Tech	Yes	Yes
DP Miner[30]	Weight And Matrix	Yes	Yes

Table 2.3: List of Tools, Patterns Discovered and Approaches used by Respective Researchers

Use of design patterns in an application increases reusability and maintainability.One new emerging approach that refactoring to design patterns seems to have promising future in reusability discipline. Research is going on in the field of pattern mining as to find new approaches and new tools.

As the many approaches and tools are being proposed but still more research is needed as no tool is complete in itself. Refactoring has been gained much more attention in the object oriented software development. Though there are several ways to refactor OO Software system, refactoring using design patterns has been more of research interest.

2.4.3 Reusable Architectures

Reusable architectures can be developed from reusable architectural patterns [31] as in FIM architecture[32] which operates at three different levels of reuse: Federation, domain and application.

[33] Focuses on how non-functional property reusability relates to the software architecture of a system. [34] presented a suggested software process model for reuse based software development approach .

2.4.4 Risk Analysis

From the past we learnt that any new development has some associated risks. Some risks if occur have great impact and some have minor impact. Risks which are being identified are described in the paragraphs mentioned below.

The major risk is where the decisions are made on the vision and on the future scope of the project. Although its probability is low because we are considering that the team members are experts but its impact is high on project and on the organization itself.

Another risk is technical risk which is a common to all of the organizations that what if the technology changes. A new technology comes in the market.

Risk related to design patterns is of time as sometimes more than one design pattern seems to be the solution. In deciding which is the right solution time consumption could be large as compared to direct development. If this risk encountered even then its impact is very low.

Risk related to repository is that the assets are kept there which are not being used since a long time. It's related to the maintenance of the repository. The impact is low if the repository is small but as the repository grows the impact will also grow as the time to find the asset will increase.

2.4.5 Conclusion

From the overall literature survey, we conclude that agile development which has promising future in the software industry and is capable of fulfilling the demands of the industry. It will be accepted widely if pattern based architecture designing, design patterns, UML based analysis and designing is incorporated. Pattern based architecture oriented agile development and use of OO patterns as refactoring to design patterns will make a space for reusability and reusable artifacts. Reuse based software engineering and agile development is an open research area in rapid development.

Chapter 3

Problem Statement

3.1 Problem Definition

Software Architecture, a topic of steadily growing importance within the field of software engineering, focuses on overall structure of a software system. Focus of study is on reusability in Agile Software Development (ASD). Reusability finds limited scope in ASD and therefore adding reusability to ASD is a challenge. Essence of Agile Software Development is rapid software development and less cost. Thus, somewhere it compromises with quality and also unable to provide reusability of its developed components. Agile Software Development provides specific solutions whereas Reuse and Component based Development believe in generalized solutions. Three layered OO-framework Reusability-Pro Agile Software Development Model is proposed.

3.2 Objectives

1. To study Agile Software Development
2. To find the critical success factors
3. Risk analysis of introducing reusability in agile software development
4. To propose a model

3.3 Need for Study

Agile methodologies came into picture to overcome the limitations of the traditional methods. In the rapid development at lesser cost the agile methodologies somewhere compromises on the quality. The systems developed by following agile methodologies is not easy to maintain as once the delivery is made no one knows what it is. Other methods like reuse based , component based software development are also used in the industry. But still No single process is complete and fulfilling the demands of the industry though all the processes are aimed towards quality product, improvement in development time and enhanced productivity.

Thus, here it becomes the need to study that how agile development is being adopted in the industry and what the research areas for agile development for improvement are as there is always a space for improvement and how can we get best from agile methodologies.

Chapter 4

Reusability-Pro Agile Software Development Model

4.1 The Proposed Model

As discussed in previous chapter about incorporating reusability in Agile Software Development. Here, we will see how it can be achieved.

The proposed model named Reusability-Pro Agile Software Development Model is a Three Layered Object Oriented Framework.

- First layer depicts Reusability-Pro Agile Software Development Cycle and
- Second layer elaborates its Reusability-Pro Agile Software Processes.
- Third layer has Agile Repository.

Reusability-Pro Agile Software Development Model is a framework that succeeds to incorporate generalization as well as specialization of its components. These components and artifacts generated while working on the software project are stored in Agile Repository. Therefore, these once developed components can be reused further. This only became possible due to proposed UML's extend and uses based design patterns. These design patterns are object oriented in nature. Thus, the proposed framework gives the Agile Software Development the essence of Reusability.

The Reusability-Pro Agile Software Development Framework is proposed in a way so that it can enhance reusability, generalization and quality in Agile Software Development using four different techniques.

(i) Reference architectures based on OO techniques which are industry domain specific are suggested for Agile Software Development.

(ii) UML's extend and uses based design pattern is proposed and standard design patterns stored in design pattern repository.

(iii) Components storage in the Compository.

(iv) Refactoring to design patterns.

Figure 4.1 shows the Reusability-Pro Agile Software Development Model and Figure 4.2 shows its Reusability-Pro Agile Software Processes.

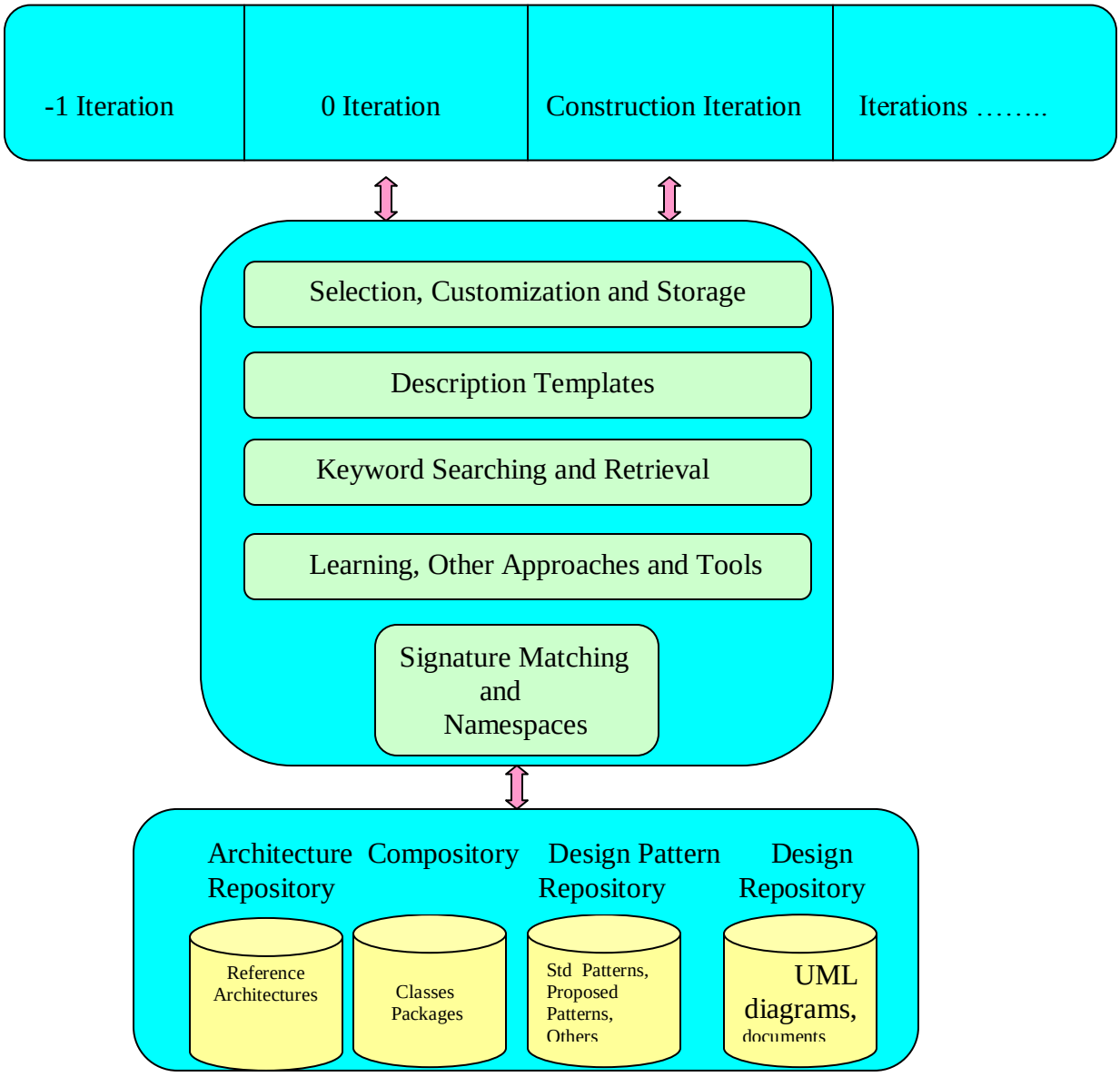


Figure 4.1: Reusability-Pro Agile Software Development Model

By incorporating architecture, components and design patterns in Agile Software development, we have directly aimed at quality and generalization of the product. Overall, Reusability is now become inherent property of our new proposed model.

Every artifact stored in the repository is reusable. There is also the provision of integrating a learning module with repository. This provision will make the repository to store more and more new learned patterns.
First layer is Reusability-Pro Agile Software Development Cycle and is discussed in 4.1.1.

Second Layer defines the processes and has five processes as follows

- Selection, Customization and Storage
- Description Templates (described in 4.1.3) ,
- Keyword Searching and Retrieval
- Learning ,Other approaches and Tools and
- Signature Matching and Namespaces.

The third layer represents repository called *Agile Repository* and is discussed in 4.1.2.

Selection, Customization and storage module is common for the agile repository and represents that needed artifact can be selected, used and can be modified and restored in the repository for further use.

Keyword Searching and Retrieval module defines the technique to extract the artifacts from the agile repository i.e. respective artifacts can be searched and hence retrieved by keywords. Keyword can be any word as name of pattern, object and architecture.

Learning , Other approaches and Tools module represents the future scope for storage and retrieval of artifacts. Any pattern matching algorithm can be used for learning. Or new techniques like machine learning or neural networks can also be used.

Signature matching and Namespaces module is restricted to only compository module of the agile repository. Signature matching technique is used to retrieve the methods and namespaces technique is used to retrieve the packages from the compository.

4.1.1 Reusability-Pro Agile Development Cycle

First layer represents the Reusability-Pro Agile Development Cycle.

- All the tasks like project selection, feasibility study, project approval are taken in –1 iteration .
- 0 iteration is where initial requirements are discussed and initial architectural envisioning is done by developers.
- Construction cycle is characterized by iteration planning, refactoring and Test Driven Development (TDD) .
- Next, further iterations continue like Release and Maintenance.
- In the existing cycle, some improvements are made in 0 iteration and construction iteration as explained in process model in 4.2 section.

4.1.2 Agile Repository

The Agile Repository is divided into three sections. Each one is explained in detail as follows.

Architecture Repository ***contains the reference architectures. These reference architectures are proposed by taking into consideration the most used technologies by different software industry working in different software systems and applications. These architectures are described in detail in section 4.1.4.***

Compository ***contains the classes and packages .***

Design Pattern Repository contains the standard design patterns, proposed design patterns and other patterns. Standard design patterns are Gang of Four (GoF) patterns and are already discussed in section 2.1.1.

Proposed OO Design Patterns is UML's <<extends>> and<< uses>> based Design Patterns.

Design Repository ***contains the UML diagrams and documents related to the design of the corresponding systems.***

4.1.3 Description Templates

Description Templates are the structure of the artifacts, which helps in the storage and retrieval of these artifacts. These templates also help to understand about the artifacts. These templates contain the detailed information of the corresponding artifacts. Description Templates for architecture repository, proposed patterns, standard patterns, object and methods are as follows.

Description Template of Architecture Repository

Sr. No.	Project Name	Initial Requirement Envisioning	Initial Architectural Envisioning	Reference Architecture	Proposed Architecture

Description Template of Proposed Patterns

Pattern Name	
Structure	UML Diagram

Description Template of Standard Patterns

Pattern Name and Classification	
Intent	What does the pattern do?

Also Known As	Other Names
Motivation	Scenario
Applicability	Situations
Structure	Graphical Representation
Participants	Classes and Objects
Collaborations	
Consequences	
Implementation	
Sample Code	
Known Uses	
Related Patterns	

Description Template of Object

Name	Attributes	Code

Description Template of Method

Name	Number of Parameters	Type of Parameters	Code

4.1.4 Reference (OO) Architectures

All the architectures are software industry domain-specific (like embedded systems, real time systems, mobile applications, web based applications, artificial intelligence systems, scientific systems etc.) architectures. Therefore different Reference Architectures based upon prevailing trends in different areas of software development are stored in the architecture

repository. Now, developer can easily refer to these stored available architectures.

(i) Hibernate based Architecture: Hibernate is an open source tool for Java platform. It provides powerful, ultra-high performance ORM (Object Relational Mapping) persistence and query service for Java. Here JDBC denotes Java Data Base Connectivity and RMI/IOP is the extension of Java RMI (Remote Method Invocation). Here DB(Data Base) could be any Java compatible DB. Figure 4.2 depicts Hibernate based Architecture.

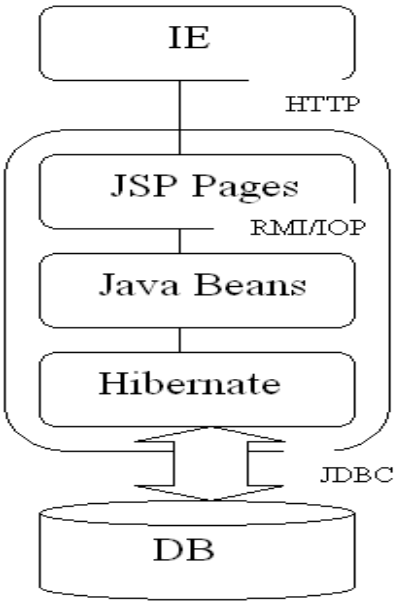


Figure 4.2: Hibernate based Architecture

(ii) Microsoft Technology based Architecture: Every layer of this architecture uses the Microsoft driven technology. ADO.NET Entity Framework is the OR persistence in .NET framework and is also shown in figure 4.3.

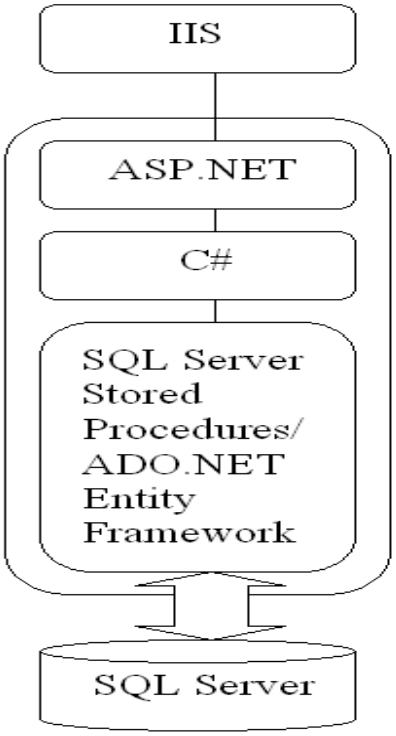


Figure 4.3: Microsoft Technology based Architecture

(iii) **Enterprise Java Beans (EJB) based Architecture:** This architecture uses Enterprise Java Beans for business classes. Use of RMI and JDBC makes this architecture simple. Figure 4.4 shows the Enterprise Java Beans (EJB) based Architecture.

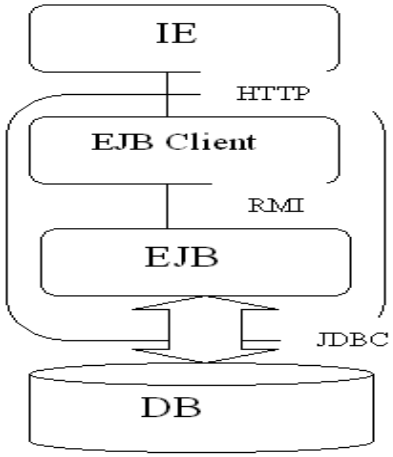


Figure 4.4: Enterprise Java Beans (EJB) based Architecture

(iv) Web Service based Architecture: Web services are typically APIs(Application Programming Interfaces) or Web APIs that are accessed via HTTP and executed on a remote system hosting the requested services. WSDL(Web Services Definition Language) is an XML based language that provides a model for describing web services. Figure 4.5 illustrates Web Service based Architecture.

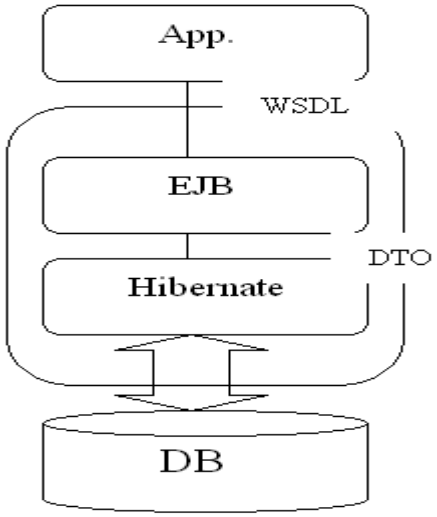


Figure 4.5: Web Service based Architecture

(v) JSQL based Architecture: JSQL is Java embedded SQL (Structured Query Language) which is subset of SQL languages, can be used to select objects instances according to selection condition. JSQL uses notation more popular for object-oriented programming than for relational database. Table rows are considered as object instances and the table as class of these objects. Unlike SQL, JSQL is oriented on work with objects instead of SQL tuples. So the result of each query execution is a set of objects of one class. JSQL based Architecture is elaborated in figure 4.6.

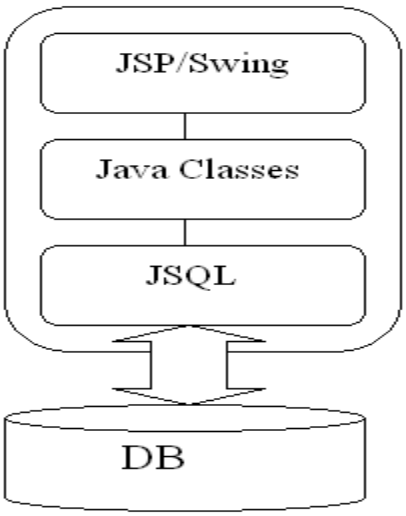


Figure 4.6: JSQL based Architecture

(vi) ASP.NET based Architecture: Every layer of this model has ASP. For persistence ADO.NET Entity Framework is used as described in figure 4.7.

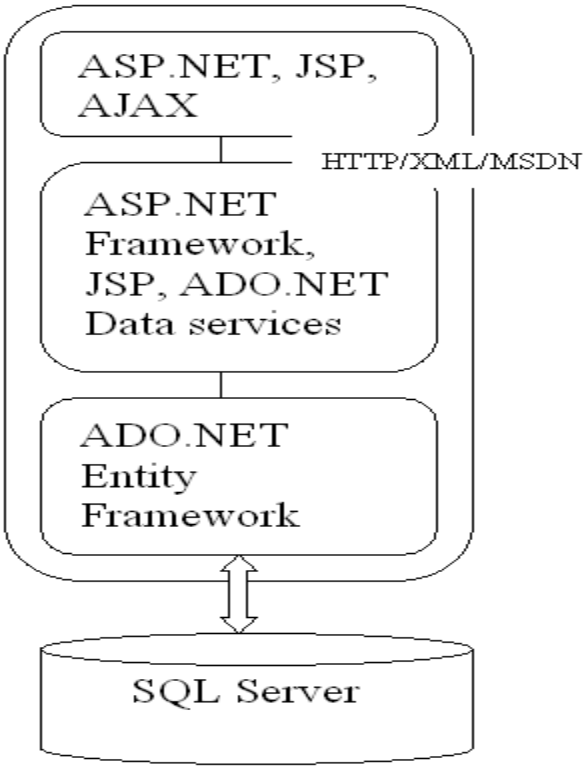


Figure 4.7: ASP.NET based Architecture

(vii) Spring and JBOSS based Architecture: Spring is an open source framework and has layered architecture, which allows the selection of its components to be used. It provides a consistent framework for J2EE application development. JBOSS is an open source Java EE based application server. Because it is Java based, the JBOSS operates cross-platform. The architecture is shown in figure 4.8.

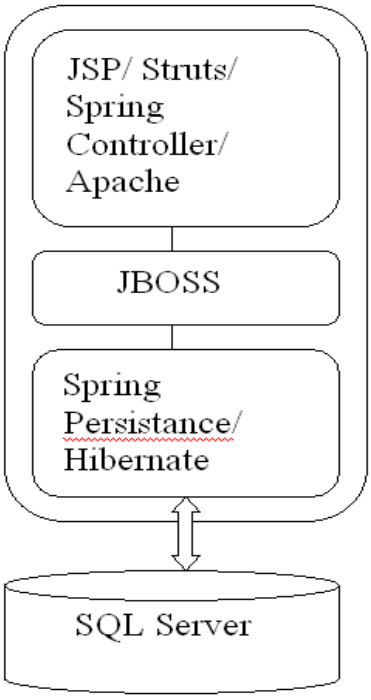


Figure 4.8: Spring and JBOSS based Architecture

(viii) Struts Framework based Architecture: Struts is an open source framework, which is based on MVC (Model-View-Controller) design paradigm. It stores application routing information and request mapping in a single core file, *struts-config.xml.* The

struts framework, itself, only fills in the View and Controller layers. The Model layer, which is application state, is left to the developer. The architecture is shown in figure 4.9.

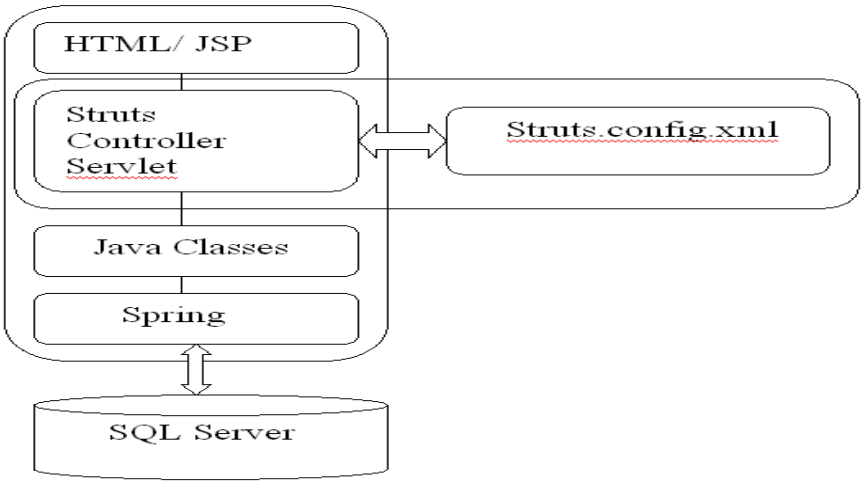


Figure: 4.9 Struts Framework based Architecture

4.2 Reusability-Pro Agile Software Development Process Model

The model describes the whole process divided into two major parts according to the iterations. In the 0 iteration firstly initial requirements are gathered from the customer and accordingly the team member who is also playing a role of architect prepares a logical model. Then its architectural design is prepared with prior experience stored in the architecture repository.

The next iteration is construction iteration, first of all iteration planning is being done where new requirements are collected and finalized for the iteration. To build or reuse decision is also made here for the components after referring the compository. After planning, coding is started and then TFD (Test First Development) which is completed with Refactoring and Test First Development with the help of pattern repository. To make the decisions like when refactoring has to start and until when refactoring has to be done a technical expert panel is setup, which review the whole process and suggest to go with refactoring or stop after some iteration. The refactored code is stored as component for future references and as reusable component.

The corresponding artifacts of the whole process are being stored in the respective repositories for future use. The figure 4.10 of Reusability-Pro Agile Software Development Process Model is as shown on the next page.

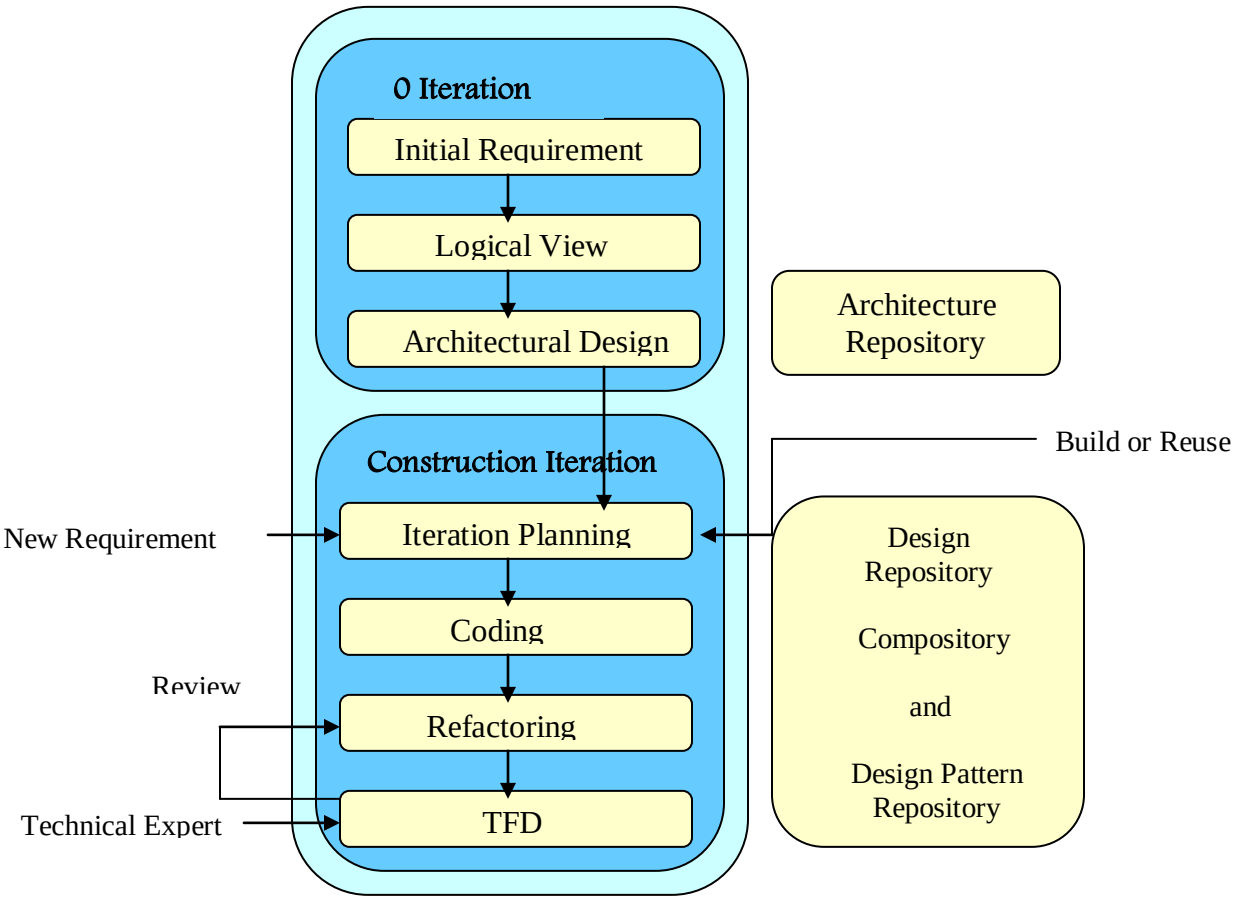


Figure 4.10: Reusability-Pro Agile Software Development Process Model

4.2.1 Decidability for the software to build a new or reuse previously stored components

There are many factors, which is kept in mind while taking the decision whether the component should be developed or should reuse. For decidability we are doing an analysis and its results are been stored in the Table 4.1.

Some strategic factors at the business level (money, policies) also count at the time of decision. Major organizational factors are skills and maintenance, which are to be considered. The build category has skill and core as its two high-order considerations. Skill refers to the expertise that is required to develop an asset. This might be a simple binary decision: Do you have the skill or not? To develop a component the organization must have the required skills and if it has then support and maintenance factors are considered, which are in favor of the building a component i.e. it is easy to maintain the component, which is developed at the organization. Another factor is delivery time of the project. Its been observed that if the components are reused its easy and at lesser cost delivery is made if one gets success at the integration and customization of the component.

For reuse, it is also worth stating that some development effort might be required in order to achieve reuse of an asset; that is, it is not simply a matter of reusing an asset. The main factor from where the analysis starts or can say contribute in the analysis most is the requirements. When reuse decision is made, there could be somewhat compromise on the requirements and up to some level customization has to be done and up to what level it depends upon need and the component both. In the build decision it's a favorable point that all the requirements will match and there is no compromise.

Consolidation describes the desire to minimize the number of assets that deliver similar functionality. This might be to achieve a variety of outcomes that range from reduction of overall cost to delivering data sources that present a single version of the truth.

Finally, there has to be the correct culture within the organization to achieve success when looking to reuse assets. This might result in the ability to compromise in order to meet the greater needs of the business, instead of completely fulfilling the needs of a specific project.

	Favorable Factors	Non Favorable Factors	Factors to be Considered
Reuse	Delivery time Complexity Cost	Requirements compromise Maintenance	Consolidation Culture
Build	Requirements match Maintenance Support	Delivery time Complexity Cost	Skill Technology Core

Table 4.1 Summary of Decidability Analysis

4.3 UML's <<*extends*>> *and* <<*uses*>> based Proposed Design Pattern

As our proposed model has the Repository which contains UML Diagrams (for different application domains) and other design patterns meant for reuse in alike software systems.

The <<uses>> pattern enhances the generality of the system and <<extends>> maintains the specialization of the system as it depicts the optional behavior. The choice of behaviors i.e to reuse the generic component or specific component depends upon the designer's will. Here in Figure 4.2, use of UML's <<extends>> and<< uses>> shows how incorporation of these two Object Oriented design patterns make reusability of different components, artifacts and design patterns according to the designers or developers will.

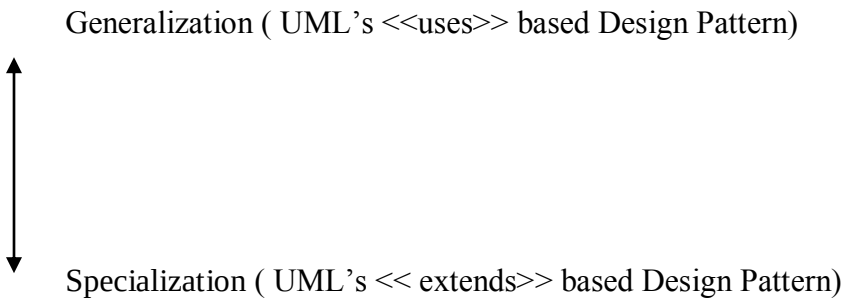


Figure 4.2: Effect of <<uses>> and <<extends>> Design Patterns

It tends to increase the reusability in Agile Software Development System. It is described in more detail in section 4.3.

Other new design patterns and architectures are get learned with the help of learning

<<*uses*>> Relationship with Generic Behavior

The generalization relationship is good old fashioned inheritance. It represents a taxonomic (eg "is a") relationship. In this case, the specializing use case provides more specific actions that correspond to the actions of the use case, but are special cases of those actions.

<<*extends*>> Relationship with Specialised Behavior

The extend relationship exists where one use case optionally has its behavior inserted into another. The extend relationship defines one or more extension points within the extended use case and a condition.

When the first extension point is reached, the condition is evaluated and the decision is made to insert the actions of the extending use case or not.

Difference between extend and uses is comparable to the difference between inheritance and composition of objects, just applied to use cases. An extending use case may be reusable in ways that don't depend on the extended case. For example, a retailer might extend a "make payment" use case with "apply for credit card". The latter use case need not be tied solely to the "make payment" use case.

For example "make payment" generalizing "make payment by check", "make payment by credit card", "make payment by gift certificate" etc... Each of these specializations accomplishes the actions of "make payment".

Generalization use cases are very useful, because they allows the software architect to outline a generic set of actions before enumerating special cases that need special treatment. To use an extend relationship, the extended use case has to identify the extensions, including which use cases, the extension points, and the conditions. Not so with a generalization. If tomorrow the system requirements change so that payment via paypal is accepted, I do not have to rewrite the "make payment" use case.

To make it more clear, Let’s take another example.

In the **figure 4.12**, uses is used twice, once from Check Out Item to Swipe Code Reader. The other is from Inventory Item to Swipe Code Reader. The Swipe Code Reader use case describes the behavior of the actor and the system when the actor swipes the Code reader across the bar codes on a product. Swipe Code Reader. The other is from Inventory Item to Swipe Code Reader. The Swipe Code Reader use case describes the

behavior of the actor and the system when the actor swipes the Code reader across the bar codes on a product. The system behavior is shown in following UML diagram.

The Swipe Code Reader behavior needs to occur within the Check Out Item description. However, since inventory clerks also swipe bar codes while counting objects on the shelves, this same behavior must be part of the Inventory Item use case. Rather than write the description for this behavior twice; we can employ the «uses» relationship to show that it belongs in both use cases.

The other interesting relationship is the «extends» relationship between Check Out Item and Check Out "21" item. In many stores, sales clerks under the age of 21 are not allowed to check out liquor.

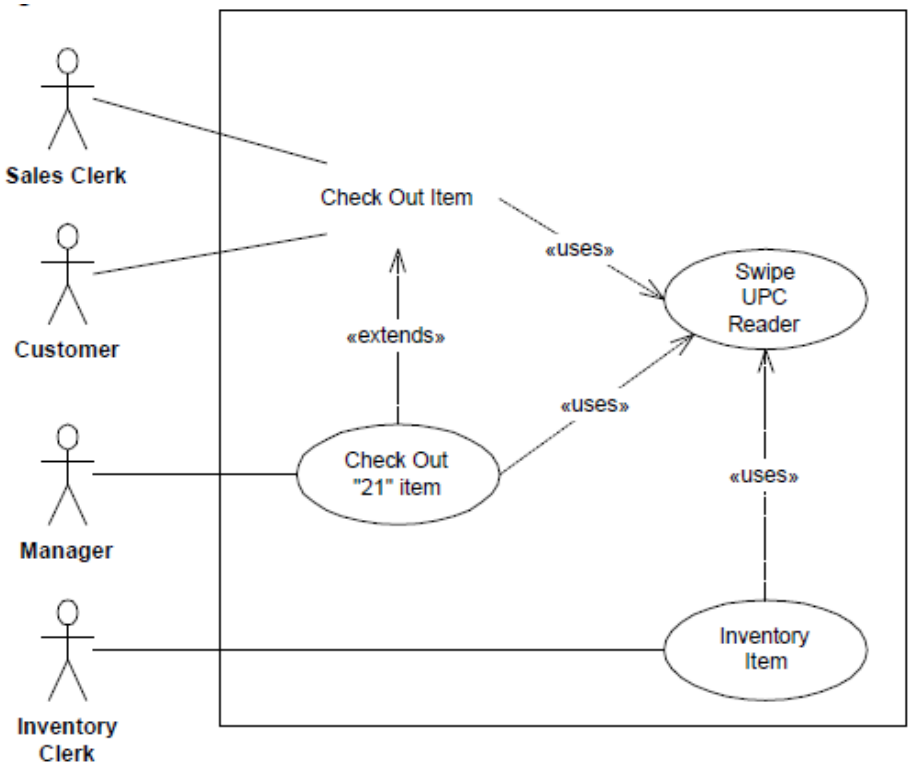


Figure 4.11 Use Case Diagram of Swipe Code Reader

We propose design pattern using extend in UML. It's a reservation design pattern and is represented in form of use case diagram and class diagram.

The Reservation Design Pattern

There are many reservation systems like ticket reservation, hotel reservation, venue reservation, book reservation system. Our aim is to enhance generalization and reusability.

Using this reservation pattern the, ASD moves somewhat from specialization to generalization. Individual systems like ticket reservation, hotel reservation, venue reservation, book reservation system are the specialized systems But incorporating all these in one reservation design pattern will make the system generics as well as specific depending upon the requirement of the system. All other specific systems will be extended from the single reservation system. The core reservation system or say reservation pattern will have all the general attributes and methods needed for any kind of reservation system. According to the software project requirement, Developer can choose from a given set of specialized domains without rethinking and redesigning its software components i.e. he only needs to reuse and add more functionality to the design pattern if only desired.

In the use case diagram shown in figure 4.13, the central reservation pattern shows the generic behavior, while other specific systems shown as collaboration so that in future at any time developer can make selection of the system according to the need.

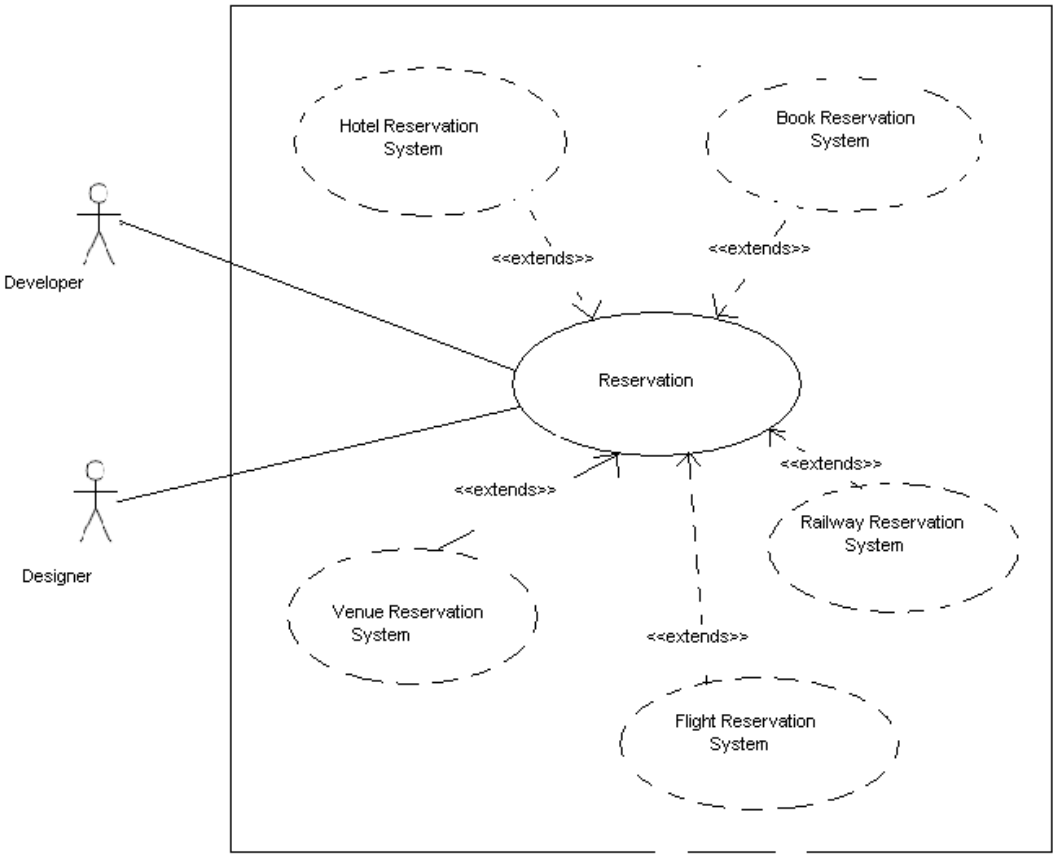


Figure 4.13 Use Case Diagram of the Proposed Reservation Pattern

Corresponding classes for the above use case diagram are:

1. Reservation
2. Hotel
3. Venue
4. Book
5. Railway
6. Flight

The resultant class diagram with attributes and related functions is as shown in figure 4.13. It shows the relationship as 1 to 0…1 that is a mapping of optional or <<extend>> relationship in UML's use case and class diagrams relationship.

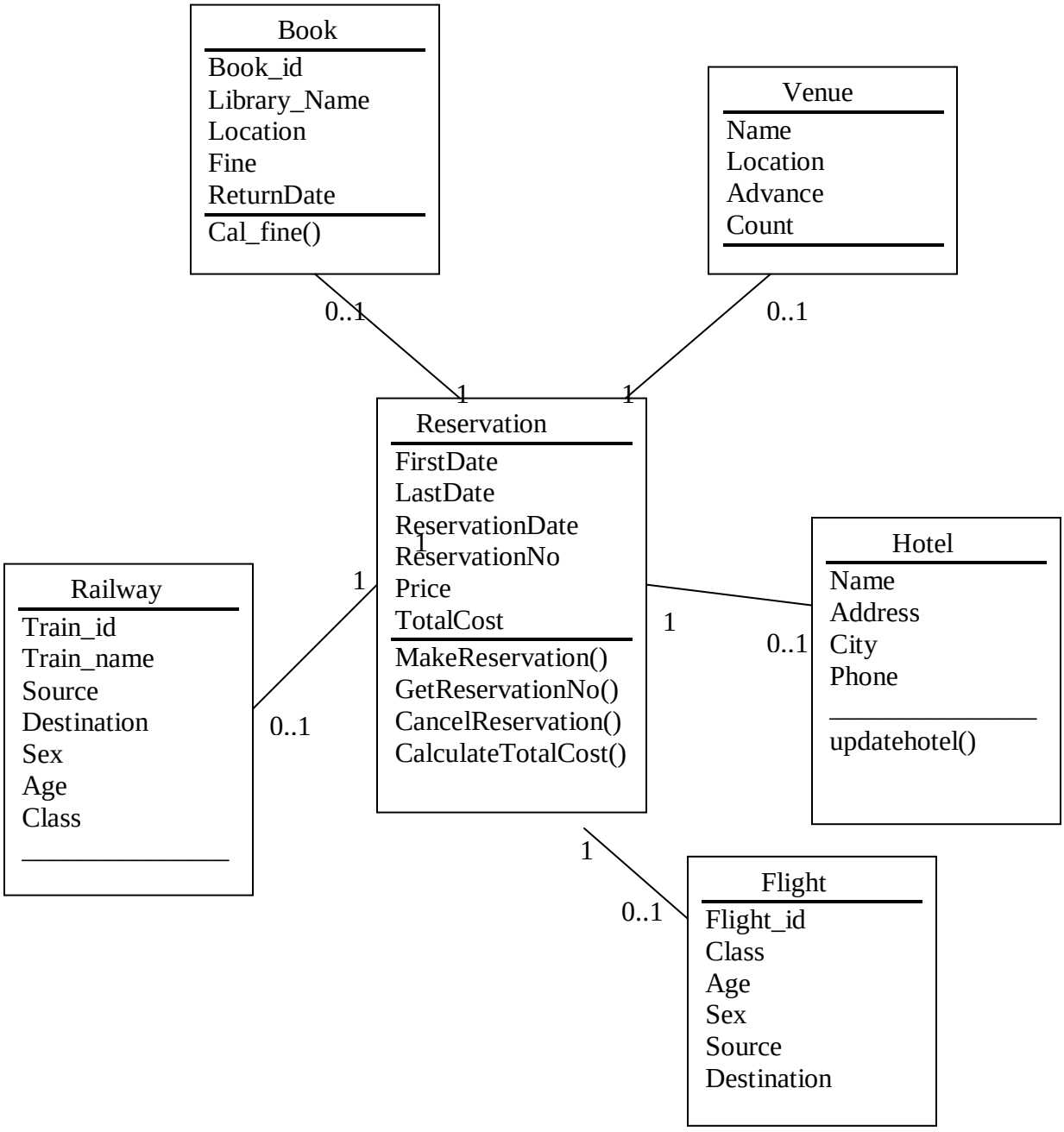


Figure 4.14: Class Diagram of Proposed Reservation Pattern

The developer if included one specialized system then the relationship is one to one (reservation design pattern to specialized design pattern) and if not included the system then the relationship is one to zero.

The next level of the proposed design pattern is as shown in figure 4.15. It represents the design pattern as 2nd level inheritance pattern. Here Venue class becomes the parent class and the other classes Marriage, Party, Presentation and Bar are the child classes. Same with the Hotel class as it becomes the parent class and room becomes its child class.

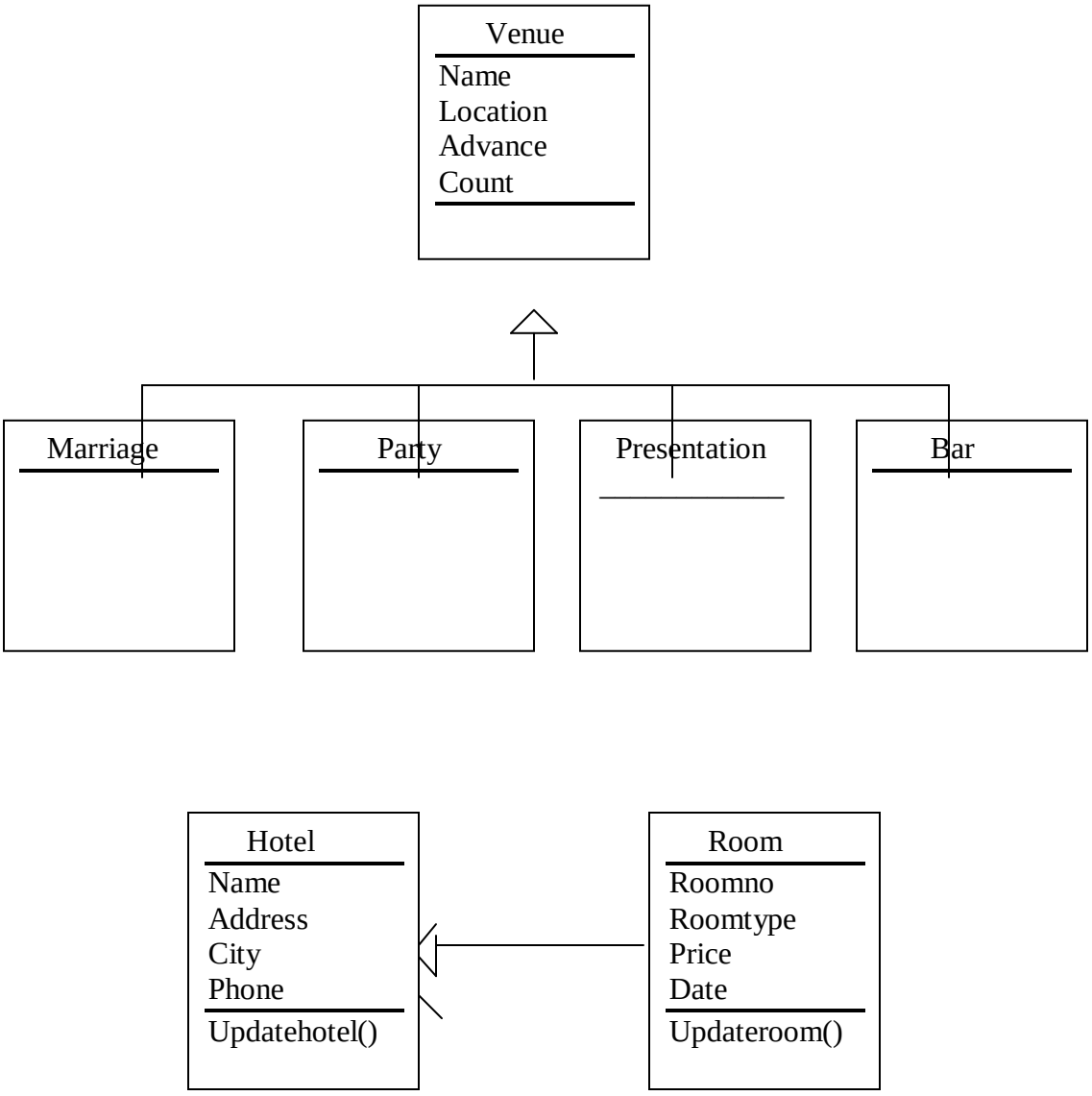


Figure 4.15: 2nd level Inheritance Design Pattern

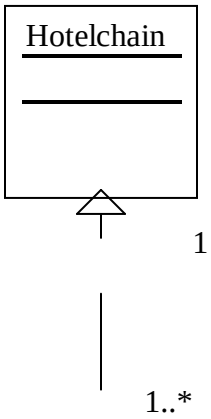
Chapter 5

Hotel Reservation System: A Case Study

Hotel Reservation System

For implementation case study of hotel reservation system is considered. Step by Step illustration of Reusability-Pro Agile Software Development Processes is depicted as follows:

1. Get all the *initial requirements* from the customer, which he has on his top of mind so that we can make a logical view of the project. Here, We find that the customer wants an on-line hotel reservation system, so that user can book the hotel by using net. Customer has a chain of hotels and he wants the facility to make selection.
2. So accordingly we prepare a *logical model* as shown in Figure 5.1.



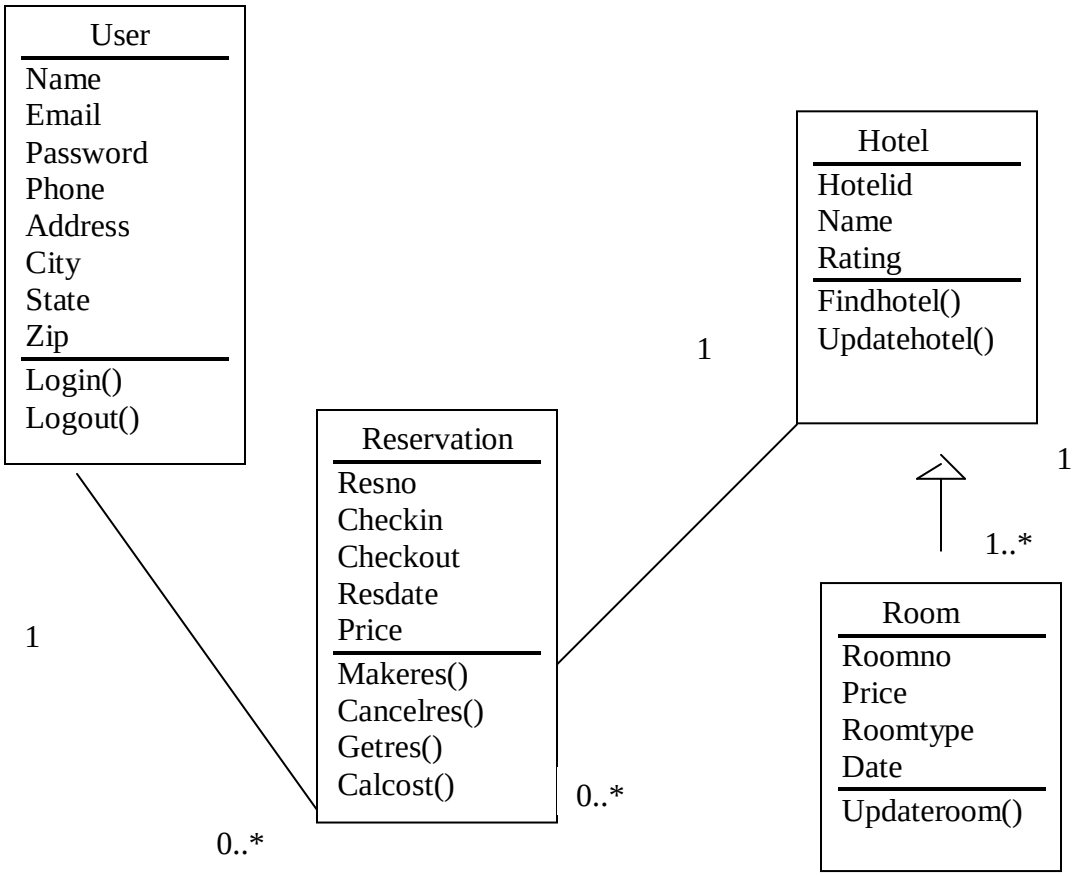


Figure 5.1 Logical View

3. Client's choice of technology and according to reference architectures in the architecture repository we choose Microsoft technology based architecture, which is shown in Figure 5.2. The *architectural design* is based on Layered Style.

1. Presentation Layer
2. Middel Layer
3. Database Layer

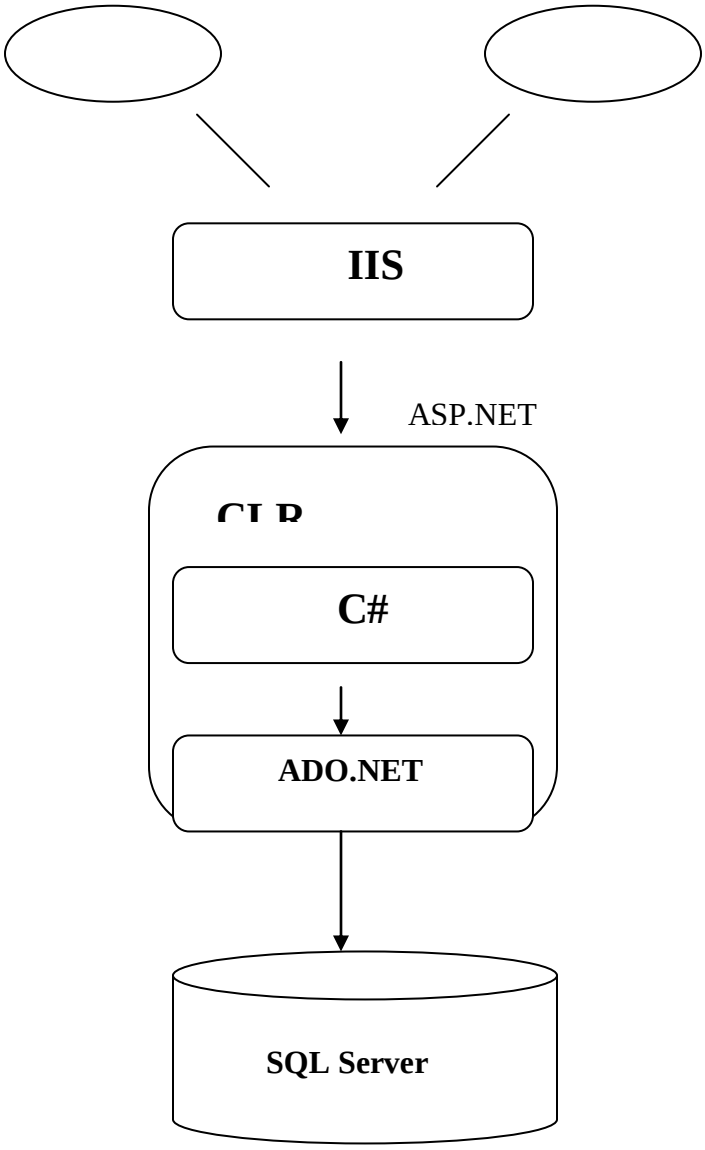


Figure 5.2 Architectural Design

4. We have done with the iteration 0. In the next iteration i.e. construction iteration, first step is to *plan the iteration*. Here, Design repository will be referred so if any existing artifact can be used. As we searched the repository, we found the reservation pattern as shown in figure 4.13. We used it to add the general behavior of reservation system
5. *Coding of related system can be undertaken* .
6. R*efactoring can be carried out depending upon expert judgement*.
7. After using the Reservstion Design Pattern, *Test First Design* must be applied for the review.

   After doing all this we get a refined design of the system, which is shown in Figure 5.3 and we start with this in the next construction iteration planning.

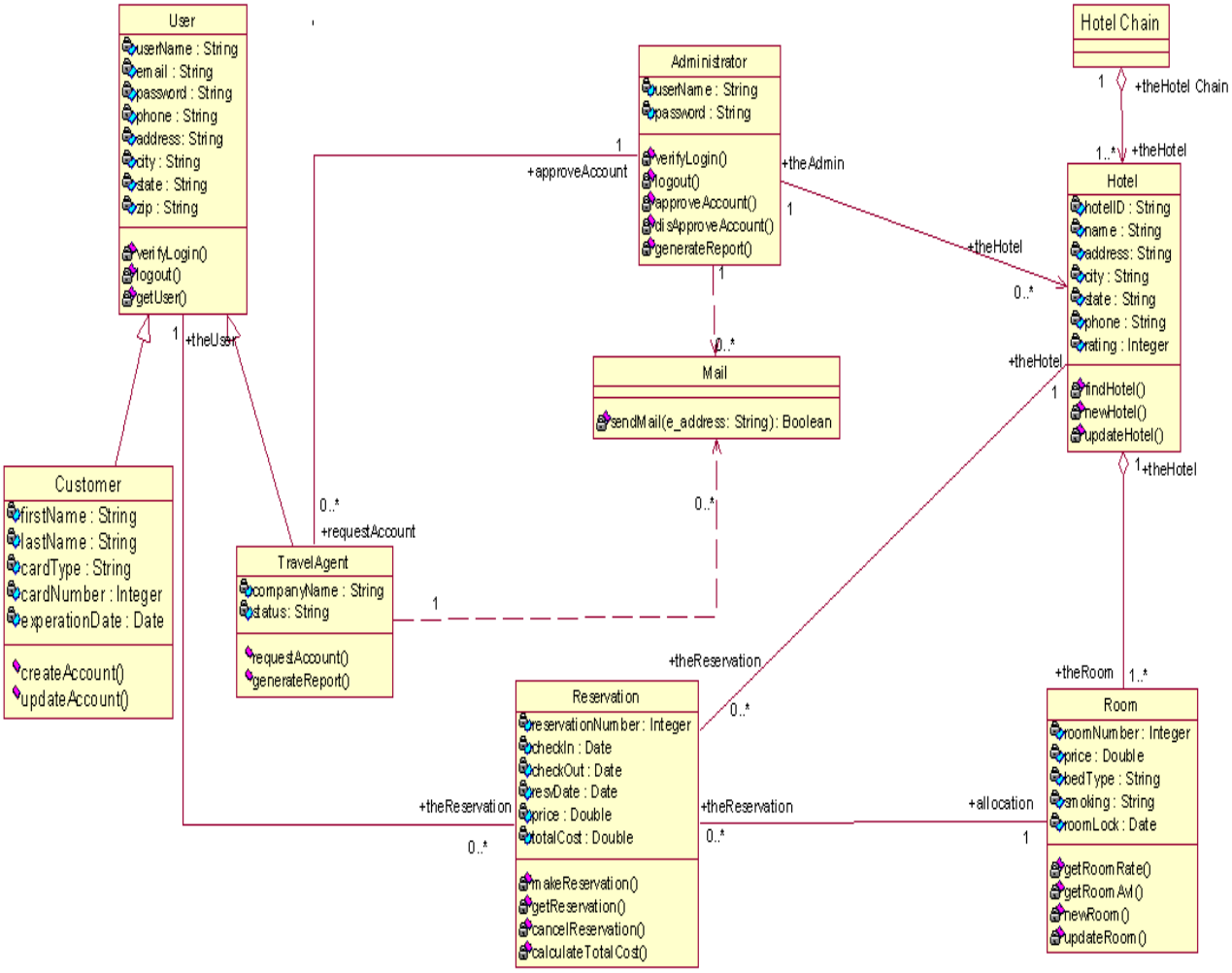


Figure 5.3 Refined Design of Hotel Reservation

In the class diagram, the 1------0..1, 0..* mappings show the selection. Where these mappings show the <<extend>> behavior and the parent-child relation shows the <<uses >> behavior.

In the figure 5.4 below, if see clearly there is one pattern hidden inside it as *acknowledge pattern*. When the user gets reservation, it gets reservation no as acknowledgment. Here customer first send the request for the reservation of the room, system will check for the room and will send the room no as acknowledgement to the Customer. This pattern gets stored in the repository and as the learning module and the

ability of the developer works together.

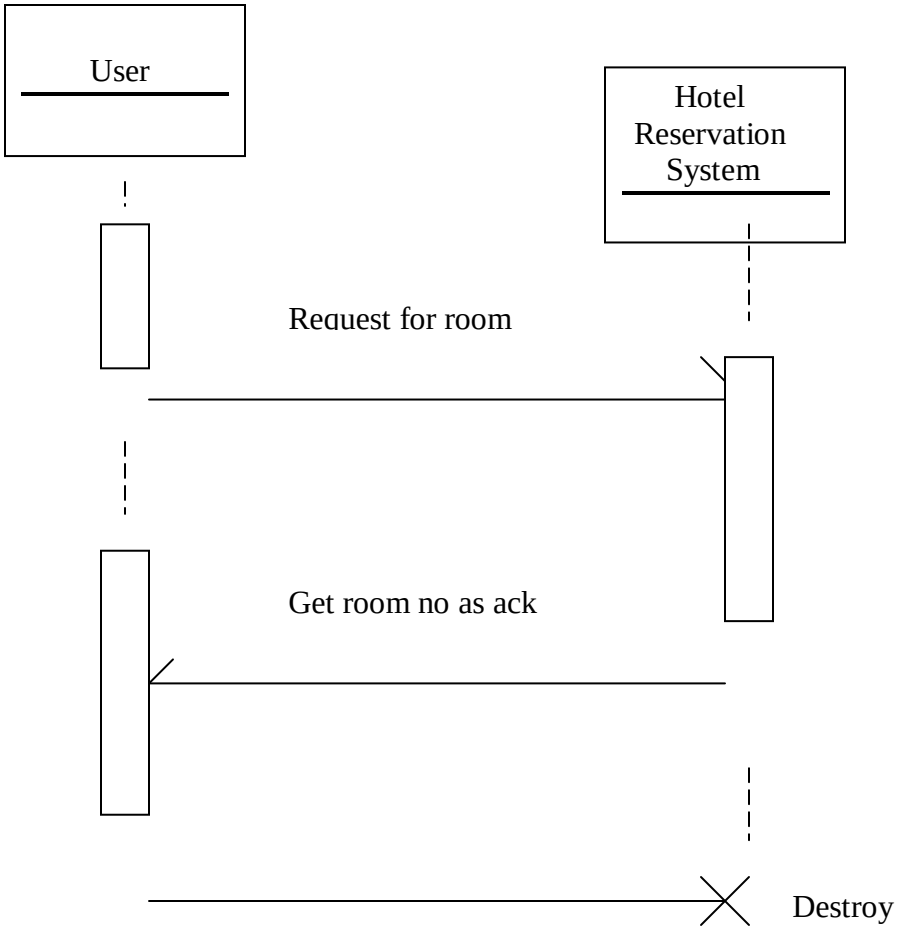


Figure 5.4 Acknowledge Pattern

Figure 5.5 is showing the general behavior of this pattern, as user will send the request for the reservation. Then the reservation system will respond accordingly as by sending the reservation number as the acknowledgement.

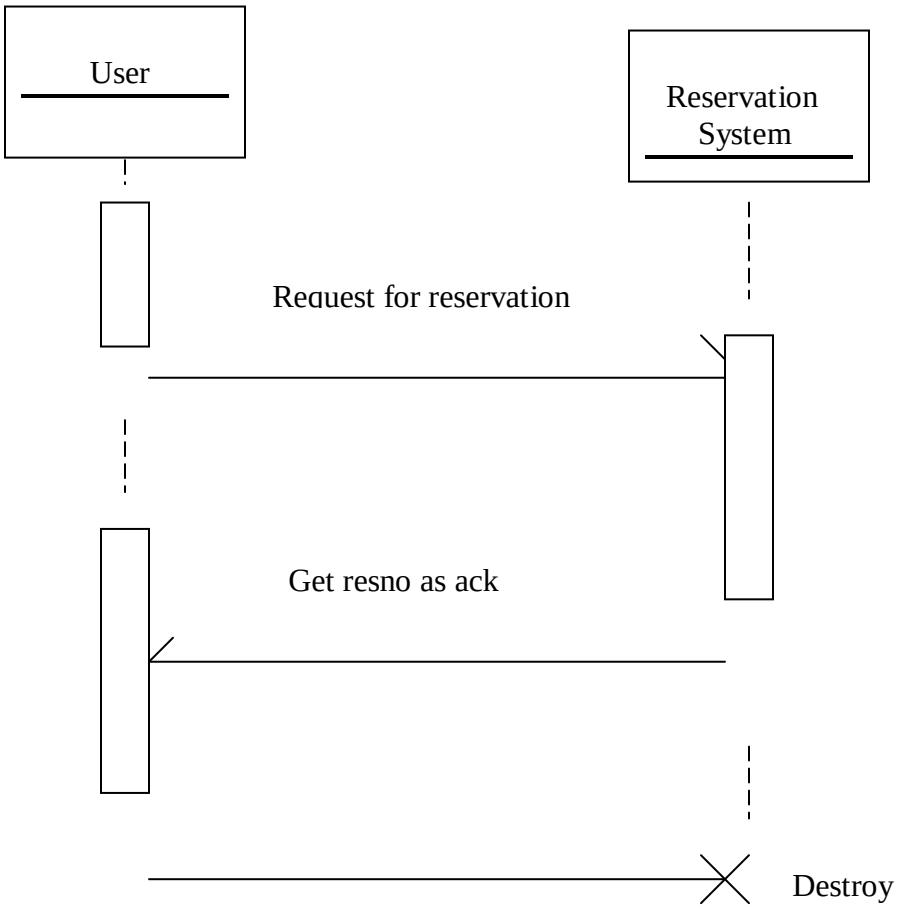


Figure 5.5 General Behavior of Acknowledge Pattern

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

The proposed model named Reusability-Pro Agile Software Development Model is a 3 layered Object Oriented framework. It succeeds to incorporate generalization as well as specialization of its components. Proposed Object Oriented Patterns i.e. UML's <<extend>> and <<uses>> based design patterns can incorporate reusability in ASD. More artifacts can be produced and reused from the design pattern repository, compository and architecture repository. It will enhance the reusability, maintainability, productivity in ASD. It will also reduce the developer's time and will improve the quality.

6.2 Future Scope

The proposed work is not automated yet. A tool that would be capable of learning from components and design patterns stored in agile repository can be designed using pattern matching techniques, neural networks or machine learning algorithms. And it can be further developed to make th whole or some part of the process automated. Also, incorporating little bit of object oriented designing can create much scope of reusability in ASD.

Bibliography

[1] Cusumano M., MacCormack A., Kemerer C.F., and Crandall W., "A Global Survey of Software Development Practices", available at http://ebusiness.mit.edu/research/papers/178_Cusumano_Intl_Comp.pdf.

[2] Salo O., "Enabling Software Process Improvement in Agile Software Development Teams and Organizations", ESPOO 2006, VTT Publications 618, pp149 +app.96 pp.

[3] Zannier C., "Tool Support for Refactoring to Design Patterns ", Conference on OOP System Language and Applications, Companion of the 17th annual ACP SIGPLAN Conference on OOP Systems, Languages and Applications, Seattle, Washington, 2002, pp. 122-123.

[4] Murauskaite A., Adomauskas V., "Bottlenecks in Agile Software Development using Theory of Constraints (TOC) Principles", Master's Thesis, Gothenburg, Sweden 2008.

[5] Turk D., France R., and Rumpe B., "Limitations of Agile Software Processes", 3rd International Conference on XP and Agile Processes in Software Engineering (XP 2002), May 2002.

[6] Dyba T., and Dingsoyr T., "Empirical Studies and Agile Software Development: A Systematic Review", Information and Software Technology, 2008, vol. 50, pp. 833-859.

[7] Hansson C. , Dittrich Y., Gustafsson B. ,and Zarnak S. , "How Agile are Industrial Software Development Practices?", Journal of Systems and Software , Sept. 2006, vol. 79, issue 9, pp. 1295-1311.

[8] Cronholm S., "Using Agile Methods? Expected Effects", 17th International Conference on Information Systems Development (ISD 2008), Paphos, Cyprus , Aug 25-27,2008.

[9] Hanssen G. K. , and Faegri T.E., "Process Fusion : An industrial Case Study on Agile Software Product Line Engineering", Journal of Systems and Software, 2008, vol. 81, pp. 843-854.

[10] Ge X., Paige R. F., Polack F.A.C., Chivers H., and Brooke P.J., "Agile Development of Secure Web Applications", Proceedings of the 6th International Conference on Web Engg., 2006, pp. 305-312.

[11] Boehm B., and Turner R., "Balancing Agility and Discipline: Evaluating and Integrating Agile and Plan-Driven methods", ICSE 2004, pp. 718-719.

[12] Salo O., and Abrahamsson P., "An Iterative Improvement Process for Agile Software Development" available at www.agile-itea.org/public/papers/SPIP.pdf.

[13] Coad P., "Object Oriented Patterns", Communications of the ACM, Sept. 1991, vol. 35, no. 9, pp. 152-158.

[14] Gamma E., Helm R., Johnson R., and Vissides J., "Design Patterns:Elements of Reusable Object Oriented Software", Addison Wesely,1994.

[15] Bass L., Clements P., and Kazman R., "Software Architecture in Practice", 2nd Edition, Addison Wesely,1997.

[16] Barbosa M. Antonio de Castro, "Components, Connectors and Architectural Patterns" available at http://www.di.uminho.pt/pg/sddi2003/artigos/marco .

[17] http://www.encyclopedia.com/doc/1011-Ooarch.html

[18] Salvio J.D., StAphane J., and Faculdade C.D., "Object Oriented Software Architecture Design based on UML/PeriNet Approach for Deadlock Prevention and Real Time Systems", Journal of Computational Methods in Science and Engineering Sept. 09, 2005, vol. 5, pp. 67-83.

[19] Class Type Architecture: A Strategy for Layering Software Application available at www.ambysoft.com/essays/ClassTypeArchitecture.html .

[20] www.win.tue.nl/~mchaudro/cbse2007/managing%20CBSE%20and %20reuse.pdf

[21] Garlen D., Allen R., and Ockerbloom J., "Architectural Mismatch : Why Reuse is So Hard", IEEE Software, November1995, vol. 12, no 6, pp 17-26.

[22] Pressman R. S., "Software Engineering" , 7th edition, McGraw Hill Education,2009.

[23] J. R., and D. J., "JIAD: A Tool to Infer Design Patterns in Refactoring", International Conference on Principles and Practice of Declarative Programming, Proceedings of the 6th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming, Verona, Italy , 2004, pp. 227 - 237 .

[24] Zannier C., " Tool Support for Refactoring to Design Patterns", Conference on OOP System Language and Applications, Companion of the 17th annual ACP SIGPLAN Conference on OOP Systems, Languages and Applications, Seattle, Washington, 2002, pp. 122-123.

[25] Gueheneuc Y. G., "Ptidej:A Flexible Reverse Engineering Tool Suite" available at ieeexplore.ieee.org/iel5/4362596/4362597/04362684.pdf .

[26] Reinersdorff A.V., "On Recognizing Design Patterns with Crocopat, Bachelor Thesis" available at http://arendvr.com/wp-content/uploads/On-Recognizing-Design-atterns-with-Crocopat.pdf.

[27] Lucia A.D., Deufemia V., Grauino C., Risi M., and Tortora G., "An Ecllipse Plug-in for Design Pattern Recovery" available at http://eit09.unibg.it/pdfs/99990041.pdf.

[28] Wang W., and Tzerpos V., "An Ecllipse Plug-in to Detect Design Patterns in Eiffel Systems", Electronic Notes in Theoretical Computer Science (ENTCS), Dec. 2004, vol. 107 , pp. 71-86.

[29] Shi N., and Olsson R.A., **"**Reverse Engineering of Design Patterns from Java Source Code", Automated Software Engineering Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering, 2006, pp. 123 – 134.

[30] Dong J., Lad D.S., and Zhao Y., "DP-Miner: Design Pattern Discovery using Matrix", Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, 2007, pp. 371-380.

[31] Gomma H. and Farrukh G.A., "Composition of Software Architectures from Reusable Architecture Patterns", Foundations of Software Engineering, Proceedings of 3rd International Workshop on Software Architecture, Orlando, Florida, US, 1998, pp. 45-48.

[32] Gomma H., and Farukh G. A., "A Reusable Architecture for Federated Client/Server Systems", Proceedings of the 1999 Symposium on Software Reusability, Los Angeles, California, US,1999, pp. 113-121.

[33] Paulisch F. , Siemens AG, " Software Architecture and Reuse – an Inherent Conflict?", 3rd International Conference on Software Reuse, Nov. 1994, pp. 214.

[34] K.S. J., and Dr. Vasantha R., "A New Process Model for Reuse based Software Development Approach", Proceedings of the World Congress on Engineering, London U.K, July 2008 , vol. 1.

[35] www. Agilealliance.org.

List of Papers

1. Ruchika, Ashima Singh, "Reusability in Agile Software Development- A Review" **Communicated** at International Conference on Advances in Computer Science, Trivandrum, Kerela.