Reusability and Agile Pattern Mining

Kavitha C.R
Dept of Computer Applications
SNGIST
N Paravur

Jino P. J
Dept of Computer Applications
SNGIST
N Paravur

Shidha M.V
Dept of Computer Applications
SNGIST
N Paravur

ABSTRACT

The main objective of software companies is to develop quality software at a high speed with fewer errors and to deliver the software to the customer at the right time. To achieve the above said objectives, many companies have adopted agile software development methodologies, by which the software can be rapidly developed with less cost and time. And also reusable components play a major role in developing reliable software faster. This paper emphasizes the importance of using patterns to achieve reusability in agile software development and also describes about the PDOT - Pattern Document Online Tool. PDOT is developed in Java platform with MySQL as the database.

General Terms

Software engineering, Data Mining

Keywords

Agile Software Development**,** Reusability**,** Patterns, Online Tool, PDOT

1. INTRODUCTION

Agile software development (ASD) which is an iterative and incremental development of software has become more popular during the last few years. These methods aim to deliver software faster and ensure that the software meets customer's changing needs and expectations. Agile methodologies focus on skills, communication and community clarifying the roles of customers, managers and developers for more satisfying and productive relationship.

1.1 AGILE MANIFESTO

The agile methodologies also known as the lightweight methodologies consist of development techniques designed to deliver products on time, on budget, and with high quality and customer satisfaction. There are different agile methodologies available today. The Agile Manifesto is a statement of the principles that underpin agile software development. It was drafted from 11 to 13 February 2001, at The Lodge at the Snowbird ski resort in the Wasatch Range of mountains in Utah, where representatives of various new methodologies (such as Extreme Programming, Scrum, DSDM, Adaptive Software Development, Crystal, Feature Driven Development, and Pragmatic programming) met to discuss the need for lighter alternatives to the traditional heavyweight methodologies.

The Agile Manifesto is a formal proclamation of four key values and 12 principles to guide an iterative and people-centric approach to software development. Agile software development focuses on keeping code simple, testing often and delivering functional bits of the application as soon as they're ready. The four core values of agile software development as stated by the Agile Manifesto emphasize Individuals and interactions over processes and tools, Working software over comprehensive documentation, Customer collaboration over contract negotiation and Responding to change over following a plan.

The 12 principles laid down in the Agile Manifesto are described below:

1. Customer satisfaction through early and continuous delivery of valuable work.
2. Breaking big work down into smaller components that can be completed quickly.
3. Working as a self-organizing team.
4. Motivating individuals with the environment and support they need and trust them to get the job done.
5. Creation of processes that promote sustainable efforts.
6. Maintaining a constant pace for completed work.
7. Changing requirements accepted, even late in a project.
8. Organizing the project team and business owners on a daily basis throughout the project.
9. Team reflects upon how to become more effective, then tuning and adjusting behavior accordingly.
10. Measurement of progress by the amount of completed work.
11. Continuously seeking excellence.
12. Utilizing change for competitive advantage.

2. REUSABILITY IN ASD

2.1 Features of ASD

The main characteristics of Agile Software Development are rapid software development and less cost, but 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.

2.2 Software Reusability

Software reusability is the quality of a piece of software that enables it to be used again in another application, be it partial, modified or complete. In other words, software reusability is a measure of the ease with which previously acquired concepts and objects can be used in new contexts. Reusable software is software (specification, design, code, etc.) made with reuse in mind. Reusable software may have been extracted from software that was not written with reuse in mind, but this is not often the case.

Software reuse can be defined as all activity that has using previously written software as its goal. Software

specifications, designs, tests cases, data, prototypes, plans, documentation, frameworks, and templates are all candidates for reuse. Reuse is a matching between new and old contexts. Whenever matching succeeds partially or completely, reuse is possible. Software reuse is not only about the re-application of a software component in another context or environment; it is also about the reuse of knowledge. This knowledge can include mathematical, engineering, and any other competencies that were needed for the construction of the reused software. Software reuse is getting considerable attention and research in methodologies, tools and techniques to promote reuse is prolific. The general opinion is that software reuse is good and that one should try to incorporate reuse of software in the development process where possible

Reasons for software reuse are:

- Increase software productivity.
- Shorten software development time.
- Improve software system interoperability.
- Develop software with fewer people.
- Move personnel more easily from project to project.
- Reduce software development and maintenance costs.
- Produce more standardized software.
- Produce better quality software and provide a powerful competitive advantage.

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 the Table 1.

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. Reuse Landscape as shown in figure 1 depicts the different reuse approaches.

There are many ways or technologies as shown in the figure1 by which reusability can be incorporated in agile software development. This paper deals mainly with the different types of patterns which help to achieve reusability in ASD.

Table 1: Possible Assets for Reuse

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

3. ABOUT PATTERNS

According to the architect Christopher Alexander, "A Pattern describes a problem, which occurs over and over again in the environment, and then describes the core of the solution to that problem in such a way that one can use the solution a million times over".

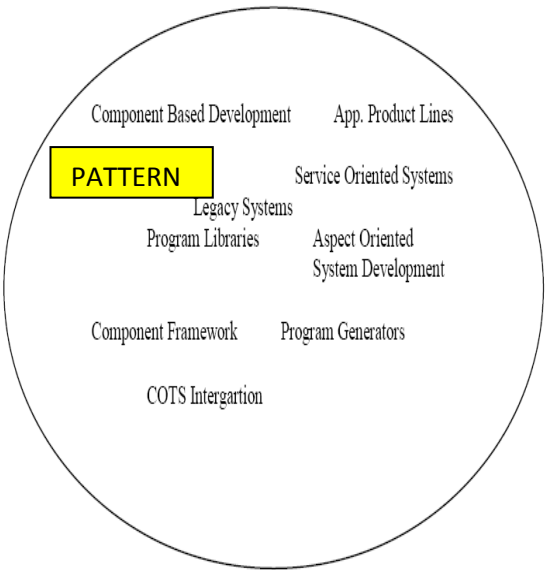

Figure 1: Reuse Landscape

Pattern is a recurring solution for recurring problem. Pattern is a format for defining problem solution pairs and giving contextual information where and how to apply the pattern. In this framework patterns are used for documenting principles and practices. The pattern has a name to facilitate discussion and the information it represents. The main characteristics of a good pattern include:

1. It solves a problem. Patterns capture solutions, not just abstract principles or strategies.

2. It is a proven concept. Patterns capture solutions with a track record, not theories or speculation.

3. The solution is not obvious. The best patterns generate a solution to a problem indirectly—a necessary approach for the most difficult problems of design.

4. It describes a relationship. Patterns do not just describe modules, but describe deeper system structures and mechanisms.

5. The pattern has a significant human component.

6. All software serves human comfort or quality of life; the best patterns explicitly appeal to aesthetics and utility.

3.1 TYPE OF PATTERNS

There are many types of design patterns which includes the following:

1. **Algorithm strategy patterns:** It addresses concerns related to high-level strategies describing how to exploit application characteristics on a computing platform.
2. **Computational design patterns:** It addresses concerns related to key computation identification.
3. **Execution patterns:** It addresses concerns related to supporting application execution, including strategies in executing streams of tasks and building blocks to support task synchronization.
4. **Implementation strategy patterns:** It addresses concerns related to implementing source code to support
   i. program organization, and
   ii. the common data structures specific to parallel programming.
5. **Structural design patterns:** It addresses concerns related to high-level structures of applications being developed.

3.2 ADVANTAGES OF PATTERNS

Design patterns can speed up the development process by providing tested, proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Reusing design patterns helps to prevent subtle issues that can cause major problems, and it also improves code readability for coders and architects who are familiar with the patterns. Design patterns provide general solutions, documented in a format that does not require specifics tied to a particular problem. Patterns encapsulate a design expert's time and expertise to solve a software problem.

The pattern user does not need to know how to design a pattern, but good pattern documentation is needed for the pattern applier to locate, select, and apply a pattern. The user needs to know what problem the pattern solves, how it is solved and the consequences of applying it. Creating and using patterns promotes software reuse; a pattern is designed once and is used many times. Reuse of patterns potentially lowers production costs and saves time by eliminating redesign. Reuse affords higher software reliability and continuity to code design.

1. Design patterns helps in analyzing the more abstract areas of a program by providing concrete, well-tested solutions.
2. Design patterns helps in writing code faster by providing a clearer picture of how one can implement the design.
3. Design patterns encourage code reuse and accommodate change by supplying well-tested mechanisms.
4. Design patterns encourage more legible and maintainable code by following well-understood paths.
5. Design patterns increasingly provide a common language and jargon for programmers.
6. The patterns address activities performed by software engineers and project managers who are accustomed to using well structured information like pattern definitions.
7. Patterns describe individual practices in a general enough way to be applied in different situations. Therefore they can be easily tried out and included in definitions of new processes. Adopting a small set of new practices gives a more profound understanding of the practices themselves and of the benefits from applying them together, which facilitates the continuous process improvement.
8. As each pattern is selected and adapted as to best fit a project and organizational context, the whole process will be more suitable for that context than any general one.

3.3 PATTERN TEMPLATE

The object oriented design patterns are described by using Gamma's patterns which are described in a consistent form by natural language. An explanation on Gamma's pattern form is given below.

Pattern Name: A meaningful name that conveys the essence of the pattern.

Intent: What design issue does the pattern address?

Also Known As: Synonyms, if any.

Motivation: An example design problem, and the way it is solved by the pattern.

Applicability: The context to which the pattern can be applied.

Structure: An OMT class diagram of the pattern

Participants: The roles of the different classes in the pattern.

Collaborations: Dynamic behavior of the participants (interaction).

Consequences: Trade-offs and benefits of using the pattern.

Implementation: Hints and techniques for implementing the pattern.

Sample Code: Examples of implementing the pattern in C++ or Smalltalk.

Known Uses: References to real systems where the pattern is found.

Related Patterns: To what patterns it is related.

The design pattern form used in Buschmann, et al. as shown below.

Name: The name and a short summary of the pattern

Problem: The problem the pattern addresses, including

a discussion of its associated forces.

Example: A real-world example demonstrating the existence of the problem and the need for the pattern

Context: The situations in which the pattern may apply

Solution: A resolution of the problem stated in terms that could be applied in many situations

Implementation: The fundamental solution principle underlying the pattern

***Technological Assumptions*:** What infrastructure must be in place for the implementation to be practical?

Variants: A brief description of variants or specializations of a pattern

Consequences: The benefits the pattern provides, and any potential liabilities

See Also: References to patterns that solve similar problems.

3.4 USING PATTERNS IN ASD

Agile developers face most challenges same as in traditional methods and the frequent change in user requirements is also pertinent in Agile. This makes it easier for agile developers to look for existing frameworks that implement design patterns or to create one for their use. Agile believes in just developing the functionality that is required. Re-factoring of patterns can also be used in agile development to deliver the product frequently within short development cycle time. Most well-formed solutions to recurrent problems are integrated into frameworks and used extensively in Agile development methodology. Whatever programming language is used for development, frameworks that include design patterns are the essential building blocks. Developers apply these patterns in their routine work as they incorporate frameworks in their code

Many of the researchers and the developers who are working in the agile system development environment have discovered many design patterns of which some of the patterns has been described here.

Klaus Marius Hansen in his paper has described the following patterns that provide a set of starting points for relating experience in agile software development facilitation. The patterns are divided into four categories:

Pattern Name:	Team Chooses
Problem:	One need to provide the best working environment for the development team. Each project is unique
Solution:	The team decides how their work environment should be and sets rules for how development should be facilitated
Pattern Name:	One War, One Room
Problem:	Developers and stakeholders on the project need to communicate as effectively and efficiently as possible
Solution:	Place developers in the same room
Pattern Name:	Customer Close By
Problem:	Stakeholder requirements are initially vague and are constantly prone to change
Solution:	Place the customer and developers close by and make room for that
Pattern Name:	Writing on the Wall
Problem:	Developers and customers should have immediate access to writing material to be used for discussions and for thinking
Solution:	Place whiteboards and paper for writing in development rooms
Pattern Name:	Simple Artifacts
Problem:	One wants to make sure developers focus on producing working software instead of, e.g., just working with software
Solution:	The simplest tool that solves a problem should be used
Pattern Name:	Flexible Furniture
Problem:	One wants Space for Pairs, Space for Groups, and want to allow for adaptation to the current project situation
Solution:	Place developers in the same room
Pattern Name:	Public and Private Spaces
Problem:	One wants to facilitate group and pair communication while still catering for the needs for private communication and thoughts
Solution:	Provide a mix of public and private spaces
Pattern Name:	Space for Groups
Problem:	Communication with customers and brainstorming among developers is essential
Solution:	Make space for groups. Make the space visible to all
Pattern Name:	Space for Pairs
Problem:	Developers need to communicate, collaborate, and coordinate effectively also in the small
Solution:	Make room for pairs so that pair discussions and pair programming are facilitated
Pattern Name:	Perks

Problem:	One wants to keep moral high and want the support of lateral thinking in the project group
Solution:	Give developers perks in the form of food and toys

Teodora Bozheva and Maria Elisa Gallo in their paper 'Framework of Agile Patterns' have described a few examples of patterns which are described below:

CodeIntegrator

Intent: To have working code at the end of every day. In a software development environment with collective code ownership, the idea is to build the system every day, after a very small batch of work has been done by each of the developers.

Origin: LD: Synch and Stabilize (Daily build and smoke test); XP: Integrate often

Category: Implementation & Testing

Application scenario: After implementing a piece of code

Roles: Developers

Activities

· Check out source code from the configuration management system.

· Put together the newly implemented and the existing code.

· Check to see, if anyone else has made changes to the same code, and if so, resolve the conflicts by applying codeImplementer.

· Apply AcceptanceTester.

· Check in the new code.

Tools: Version control tools support this activity.

Guidelines:

· Continuous integration avoids or detects compatibility problems early. If changes are integrated in small batches, it will be infinitely easier to detect and fix problems.

· A single integration point (computer) has to be defined.

· Every developer is responsible for integrating his/her own code always when there is a reasonable break. This could be when all the unit tests run at 100% or some smaller portion of the planned functionality is finished. Only one developer integrates at a given moment and after only a few hours of coding.

· All the tests have to pass successfully after integrating the system. Each integration results in a running system. Integration happens every 1-5 hours, at least once a day.

The other mentioned patterns described in their paper are given below:

Code Implementer: Implement code

FDDCoder: Implement defect-free code following FDD

XPCoder: Implement defect-free code following XP

SoftwareInspector; Find out defects in project work products

UnitTester: Define and execute in an automated way unit tests for a module of code.

AcceptanceTester: Define and execute in an automated way tests written by the customer to show that her requirements have been implemented correctly

AgileContractCreator. Define a contract that provides a software provider some degree of flexibility in any development parameter: schedule, cost, scope or quality.

AgileDocumenter. Create and maintain models and design documentation.

Communicator. Maintain good communication between project stakeholders

CustomerFeedbackIncreaser. Increase the feedback from the customers to development teams by holding a customer focus group at the end of each iteration

Designer. Design a software product in an iterative and incremental way.

DesignRefactorer. Improve the design of a software product.

DevelopmentFeedbackIncreaser. Increase the feedback from the development team to the management.

FeatureDesigner. Identify and specify classes to be involved in the design of a feature

FeedbackIncreaser. Increase feedback from developers and customers

IterationReviewer. Review an application, find, record and document customer changes requested to be implemented in the next iteration.

Iterative & Incremental Modeller. Perform iterative and incremental modeling.

Iteration Planner. Make a plan of the requirements to be implemented in one iteration

InTeamModeller. Enable effective teamwork and communication within the development team and with the project stakeholders.

Mission Creator Create and document project mission. The related artifacts need to answer three questions: What is this project about? Why should we do this project? How should we do this project?

MissionValuesSharer. Building a shared vision and responsibility for achieving the project mission.

ModellingMotivator. Better understand and communicate the models. Compare design alternatives to identify the simplest solution that meets the requirements.

ModelValidator. Validate the design work by considering how it will be tested and proving it by code.

ProductFeedbackIncreaser. Increase development team's feedback about the product

ProductivityIncreaser. Optimise the work of a team as to increase its productivity.

ProjectCloser. Help people learn from experience and anticipate the future.

Project Planner. Make a plan of the requirements to be implemented in a project

SimplicityModeller. Enable simplicity within the modeling effort. That means keeping the actual content of the models

(requirements, architecture, design) as simple as possible, depicting models simply, using simple tools .

SoftwareInspector. Find defects and accelerate team learning

StandardApplicator. Model and code according to agreed standards. This ensures that the models and the code communicate as clearly as possible.

TeamFeedbackIncreaser. Increase the feedback within the team.

ValueMapper. Create a flow diagram of an activity granulating it to atomic tasks and specifying the time /effort/expenses spend for the performance of each atomic task.

ValueTracker. Identify the points in a process or activity, where effort is spent without generating value for the customer.

WasteEliminator. Reduce the activities, which take part in a process, but do not generate value for the customer, i.e. do not contribute to the final result.

Many different patterns identified by different developers and researches in the agile software development can be identified and can be recorded in a repository. The above mentioned patterns can be permanently stored in a repository using an online pattern document tool which can be used for further references. This tool helps the developers to capture the design patterns and also helps the new developers to easily browse and find out similar patterns by which they are able to solve their software development problems.

4. PDOT

PDOT (Pattern document Online Tool) is an online tool which is used to document the design patterns in an agile software development. This tool is used to collect all those patterns which really works, found during the different stages of software development life cycle. Registered users can only add patterns to this repository. Also through this tool, developers can easily search for the patterns either by category wise or by the name of the pattern which will be helpful to them in solving their problems while they develop the software. The data mining techniques has been used for implementing the search for the patterns from the repository. The table 2 given below shows the comparison between the traditional pattern tool and PDOT (Pattern documentation online tool).

The template used for describing the patterns is of the following form:

1. Pattern Name: Each pattern has a simple and meaningful name
2. Objective :Brief description about the purpose
3. Category : Type to which the pattern belongs i.e. requirement analysis, design, coding, testing, software development facilitation etc.
4. Content :Scenario in which the pattern is to be applied
5. Roles: People involved in carrying out the pattern and their responsibilities.
6. Tools: Support for executing the pattern
7. Guidelines: Hints for performing the activities

Table 2: Comparison between a traditional tool and PDOT

Criterion	TP tool	PDOT
Readability	Good	Excellent
Understandability	Easy	Easy
Language Support	All languages	All languages
Ease of development	medium	High
Reusability	medium	high
Searching	medium	high

4.1 Features of PDOT

The system is developed using the front end Java and the back end MySQL At present this tool helps the end users to perform the following:

1. Registered users can add patterns to the database
2. End users are able to search the patterns database by entering the particular pattern name or the end user is able to search for patterns by category wise.

The software architecture and the sample screenshot of the online tool have been given below.

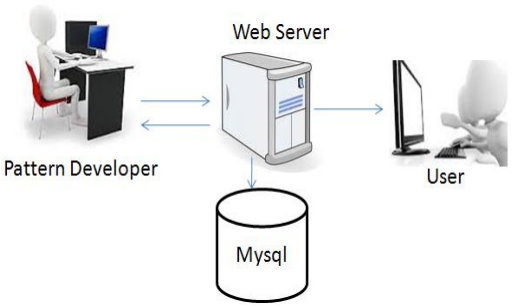

Figure 2: Architecture of the System

Figure 3: Screen shot of the home page

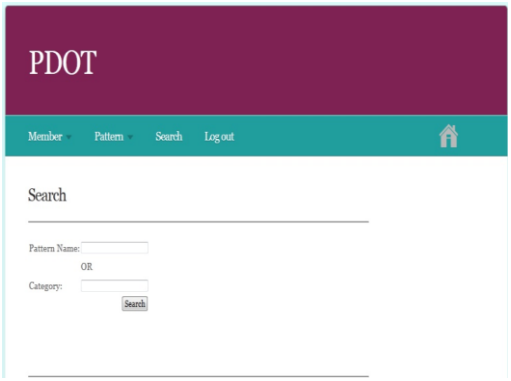

Figure 4: Screen shot of the search page

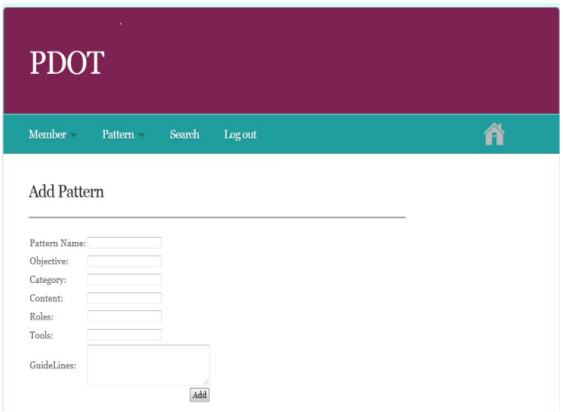

Figure 5: Screen shot of the Add Pattern page

5. CONCLUSION

ASD emerged as a need to respond to the rapidly changing market. Many ASD methods were developed and successfully implemented in different organizations. But these methods have bottlenecks that should be carefully monitored while implementing the method. From the overall literature survey, we conclude that agile software development has a promising future in the software industry and is capable of fulfilling the demands of the industry. Thus we can say that reusability can be achieved in agile software development by using pattern based architecture designing, design patterns, UML based analysis and designing. The use of patterns in agile development will make a space for reusability and reusable artefacts. PDOT tool will be helpful for the developers to record the different patterns which they might come across during the software development and also helps in searching for the patterns from the repository. Further, searching for patterns can be improved by means of advanced pattern matching techniques, neural network etc.

6. REFERENCES

[1] Jaspreet Singh, Ashima Singh, Agile software development and reusability**,** IJREAS Volume 2, Issue 2 (February 2012) ISSN: 2249-3905

[2] http://www.win.tue.nl/~mchaudro/cbse2007/managing%20CBSE%20 and %20reuse.pdf (cited on 1/1/2013)

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

[4] Klaus Marius Hansen, 'Agile Environments – Some Patterns for Agile Software Development Facilitation'

[5] Michael R Blaha,, James R Rambaugh, Object - Oriented Modeling And Design With Uml, 2/E, Dorling Kindersley(India) Private Ltd

[6] Fowler, Martin Analysis Patterns: Reusable Object Models, Pearson education, ISBN: 0-201-89542-0

[7] Gomaa, Hassan Software Modeling and Design: UML, Use Cases, Patterns, and Software architectures, ISBN: 978-0-521-764148

[8] James O. Coplien, Gertrud Bjørnvig Lean Architecture: for Agile Software Development, Wiley

[9] Roger Lee, Naohiro Ishii Software Engineering Research, Management and Applications 2009

[10] James O. Coplien, Neil Harrison Organizational patterns agile software development, Pearson Prentice Hall, 2005

[11] Craig Larman: "Agile and Iterative development: a Managers Guide", Addison-Wesley, 2004

[12] David J. Anderson: "Agile Management for Software Engineering. Applying the Theory of Constraints for Business Results", Prentice Hall, 2006

[13] Jim Highsmith: "Agile Software Development Ecosystems", Addison-Wesley, 2002

[14] Boehm B., Turner R.: Balancing Agility and Discipline: A Guide for the Perplexed, Addison Wesley (2003)

[15] Scott W. Ambler: Agile Modelling, John Wiley&Sons, Inc. (2002)

[16] Gamma E., et al: Design Patterns, Addison-Wesley (1995)