An Empirical Analysis of Defect Prone Design Pattern

Mamoona Jalil[1], Javed Farzand[2] and Muhammad Ilyas[3]

**[1]Department of Computer Science and Information Technology, University of Sargodha
Sargodha, Pakistan**

**[2]Department of Computer Science and Information Technology, University of Sargodha
Sargodha, Pakistan**

**[3]Department of Computer Science and Information Technology, University of Sargodha
Sargodha, Pakistan**

Abstract

Design patterns are problem-solution pairs that provide proven solutions for commonly occurring design problems. They are used to increase maintainability, reusability, comprehensibility and code quality. However, some studies have indicated relationship between design patterns and defects that doubts the claimed benefits of design patterns. In this paper we present an empirical study to evaluate the error proneness of design patterns. We extract the design patterns from open source software and map these patterns to post-release defects. Information on defects is extracted from version control repositories and bug databases. We have applied Mann-Whitney test to find the design patterns that are more error-prone than others.

Keywords: *Design Patter, Code Quality, Reusability, Comprehensibility, Error-Prone Modules.*

1. Introduction

Patterns have been developed in different discipline of software engineering. Major objective of patterns is to produce high-quality solutions by considering limitation of time. Patterns yields towards reusability and support to simplify analysis and design processes by choosing existing solutions of different problems. Patterns can simplify analysis and design processes by reusing existing solutions of particular problems. A comprehensive collection of patterns that are broadly used in software analysis and design is design pattern [10].

Studies of Design patterns started in 1980's [5] and gained popularity in early 90's [4]. Software design pattern have become very popular in object oriented paradigm, as it shows relationship and interaction between objects and classes. This relationship or interaction do not cope the final specification of objects and classes. Basic occurrence of design pattern is to help out designers to keep focus on different aspects like, How to interact with design, How to improve and transfer knowledge through design patterns, How to improve the software documentation, How to encapsulate the experience, How to improve the common vocabulary of software to reduce domain related barrier, etc.[1][2][3].

A good quality design pattern mostly focuses problems of object oriented software design and present a solution which apparently improve (or proportional to) its quality with respect to (aspects like) reusability, maintainability, comprehensibility and flexibility to changes [6][7]. However, some common arguments regarding the use of design patterns also often related to defects [8][9].

In this research, we have discussed different design patterns and defects incorporated in those design patterns. We performed a case study on different software projects. During the study, we perform a comparative analysis among different patterns and try to find out solution, that which design pattern is more error prone with respect to a particular scenario.

In this research, we have discussed different design patterns and defects incorporated in those design patterns. We have proposed an approach in which error-prone design patterns are identified and extracted. For this purpose, we have taken five open source systems and among those, java files which contain bugs are observed for pattern occurrence. Four design patterns have been considered for evaluation; Singleton, Factory, Composite and Adapter. We have analyzed that which design pattern among pairs of patterns is likely to produce more errors than its partner. Different hypothesis are established and statistical tests are performed. It has been evaluated that Adapter pattern is more error prone as compare to other patterns.

The rest of the paper is organized as follows: Section 2 reviews the related work. In section 3, the proposed framework is provided. Hypothesis studies are results are covered in Section 4. The final section gives conclusion of research work.

2. Related Work

Design pattern describe good solutions to commonly recurring problems in software design [10]. Gamma et al. [4] has described design patterns in details and classified them on the basis of two parameters. The first criterion is known as *Purpose*, which tells what a pattern does. Patterns may have three kinds of purposes; creational, structural and behavioral. The second criterion is called *Scope,* which determines whether the pattern mainly applies to classes or objects. Class patterns conduct inter-relationships of classes and their sub-classes. These patterns are based on inheritance that is why they are static fixed at compile time [10][4].

Design pattern proposes good solutions to commonly occurring problem in software design. During the maintenance phase of software system, the correctness of design of software system must be checked according to some criteria. This is done because of checking that defects exist in design and if they exist, corrective measures are performed. Design pattern defects are poor or bad solutions to mostly occurring problem in software design [9].

Lerina et al. [7] has also highlighted design pattern defects. Naouel Moha et al. have explained different types of design defects. Design patterns defects contain problems at different levels of granularity starting from architectural problems i.e anti-patterns to low level problems i.e. code smells [11][12]. An anti-pattern suggests that solution to design problem generate negative or incorrect results. They have explained the following different types of design defects.

2.1 Intra-Class Design Defects

Intra-class design defects includes design defect concerned with internal structure of a class. Methods with so many invocations are error-prone and tough to maintain.

2.2 Behavioural Design Defects

It includes defects concerned with application semantics. One more example of this kind of defects concern changes in the system environment.

2.3 Inter-Class Design Defects

This classification contains design defect concerned with external structure of classes and their relationships. All design defects related to architecture belong to this category, e.g. mixing of different algorithms within a single data structure is a defect related to architecture.

On the other hand Gamma et al. [4] has classified defects that tell the nature of design pattern defects as follows:

Creational design pattern defects: These patterns are linked with creational design patterns, i.e. Abstract, Factory, Builder, Singleton etc.

Structural design pattern defects: Structural design pattern defects are linked with structural patterns including Composite, Decorator, and Façade etc.

Behavioral design pattern defects: These patterns are linked with behavioral patterns i.e. Command, Iterator and Visitor etc.

There are four kinds of design pattern defects named missing, in excess, deformed and distorted. In [13] they distinguish among four kinds of design pattern defects.

- An approximate or deformed design pattern is a design pattern that is not implemented properly according to the GOF patterns but there is no error in it.
- A distorted or degraded design pattern is distorted form of design pattern that is dangerous for the code quality.
- A missing design pattern generates poor designs.
- An excess design pattern is related to excessive use of design patterns [13].

Naouel Moha et al. has argued that detection of design defects is needed for enhancing the quality of software systems. They presented a validation of their previously presented DÉCOR method. Using four design defects and their detection in 10 reverse engineered designs. DÉCOR is a method that specifies design defects, automatically generate detection algorithm and detect design defects. They have also described that design defects are bad solutions to commonly occurring problems in object-oriented programming. Quality of programs can be accessed by using design principles like low coupling and high cohesion. They propose an approach established on the combined use of metrics and FCA to propose corrections to design defects in object-oriented programs. A case study of a specific defect, the Blob, which is depicted from the Azureus project exemplify their approach.

3. Proposed work

Design patterns are problem-solution pairs that can be used to solve recurring design problems. In this section, we will describe our methodology, project under study, hypothesis and results.

3.1 Methodology

We have selected some open source software systems and studied their statistics which includes total number of java files in the specific project. Design patterns have been identified from source code and all required information has been putted in some table and name of design pattern has also been mentioned in table. Then a bug repository of each project has to be checked. In bug reports, we have tried to know that which bugs are there and how programmers have removed them in different versions. Then mapping of bugs to design patterns has been done. The error producing design patterns has been highlighted.

3.2 Project Selection

For this study, five projects are selected from the open source repository (sourceforge.net). These systems belong to the category of ‘Software Development’. All projects are written in the Java language. The reason of selection for these projects is the easy availability. The category “Software Development” is selected because of its reasonable size having 6923 projects available. So selection on the basis of number of downloads and recommendations are quite easy.

A brief description of the selected projects is given below:

JEdit: JEdit is a programmer's text editor. It uses the Swing toolkit for the GUI and can be configured as powerful IDE through the use of its plug-in architecture.

Eclipse Checkstyle Plug-in: The Eclipse Checkstyle plug-in integrates the Checkstyle Java code auditor into the Eclipse IDE. The plug-in provides real-time feedback to the user about violations of rules that check for coding style and possible error prone code constructs.

JSmooth: JSmooth creates standard Windows executable files (.exe) that smartly launch java applications. It makes java deployment much smoother and user-friendly, as it is able to find and run Java VMs by itself, or help the user get one if none are available.

JACOB-JAVA COM Bridge: JACOB is a JAVA-COM Bridge that allows you to call COM automation components from Java. It uses JNI to make native calls to the COM libraries.

Hibernate: Hibernate is an Object/Relational Mapping tool for Java environments. The term Object/Relational Mapping (ORM) refers to the technique of mapping a data representation from an object model to a relational data model with a SQL-based schema. Hibernate - Relational Persistence for Idiomatic Java.

3.3 Design Pattern

There are total 23 design patterns described in [9]. We have selected five design patterns based on their common use and availability in literature. Following is the brief description of the selected design patterns:

- Singleton pattern fall under the category of Creational Patterns. The intent is to ensure a class has only one instance and provide a global access point to it.
- Factory Method falls under the category of Creational Patterns. The intent is to define an interface for object creation, but subclasses will decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
- Adapter pattern comes in the category of Structural Pattern. The intent is to convert the interface of a class into another interface expected by client.
- Composite Pattern comes under the category of Structural Pattern. The intent is to compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.
- Observer Pattern is a part of Behavioral Pattern. The intent is to define a one-to-many dependency between objects so that when state of one object changes, all its dependents are notified and updated automatically.

3.4 Extraction of Design Patterns

We have manually extracted the design patterns from source code. The reason for choosing the manual method is to achieve high precision and accuracy for pattern extraction. Most of the automated methods have poor precision and accuracy, so manual extraction will overcome this problem. To extract the design patterns, we have reverse engineered the source code to design diagrams. From these diagrams, patterns are identified according to the pattern templates. Figure 1 shows a simple class diagram that is used for the identification of design patterns. We make similar diagrams for all the classes

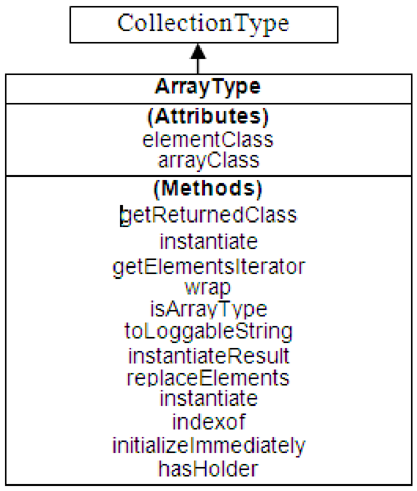


Fig. 1 Java Class ArrayType (org.hibernate.type).java

3.5 Mapping of Bugs

We take the bug information from the bug databases of respective projects. The bug databases provide information on the type of bug, description, component and module details, status and the developer who is assigned to fix the bug. Such information can be easily extracted using bug database servers like bugzilla. We extract this information and store into a local database for further use.

In order to map the bugs to source code locations, we use version control systems. These systems provide the information on changes made to the source code. For this study, subversion is used to extract the log and modifications data. The subversion log holds information about revision number, date of modification, number of lines added or deleted, developer and a comment describing the modification. This information is helpful in identifying the revisions in which a bug was fixed. To locate the origin of bugs, modification data is required which can be obtained using a differencing tool. We take the revision in which a bug was fixed and take difference with the previous revision. Difference between each two consecutive revisions is taken until the source of bug is found. The revision containing the source of bug is scanned for the presence of a design pattern. We extracted this information and store into a local database.

4. Hypothesis

We have established the following hypothesis to support our idea.

4.1 First Hypothesis

Null Hypothesis:

The pair of patterns (Singleton, Factory) will have same central tendency (Median) to produce errors.

$\text{Median}_S = \text{Median}_F$ (where S = Singleton and F= Factory)

Alternate Hypothesis:

The pair of patterns (Singleton, Factory) will have different tendency (Median) to produce errors.

$\text{Median}_S \neq \text{Median}_F$ (where S = Singleton and F= Factory)

4.2 Second Hypothesis

Null Hypothesis:
The pair of patterns (Composite, Adapter) will have same central tendency (Median) to produce errors.

$\text{Median}_C = \text{Median}_A$ (where C = Composite and A= Adapter)

Alternate Hypothesis:
The pair of patterns (Composite, Adapter) will have different tendency (Median) to produce errors.

$\text{Median}_C \neq \text{Median}_A$ (where C = Composite and A= Adapter)

4.3 Third Hypothesis

Null Hypothesis:
The pair of patterns (Singleton, Composite) will have same central tendency (Median) to produce errors.

$\text{Median}_S = \text{Median}_C$ (where S=Singleton and C = Composite)

Alternate Hypothesis:
The pair of patterns (Singleton, Composite) will have different tendency (Median) to produce errors.

$\text{Median}_S \neq \text{Median}_C$ (where S=Singleton and C = Composite)

4.4 Fourth Hypothesis

Null Hypothesis:

The pair of patterns (Singleton, Adapter) will have same central tendency (Median) to produce errors.

$\text{Median}_S = \text{Median}_A$ (where S = Singleton and A= Adapter)

Alternate Hypothesis:
The pair of patterns (Singleton, Adapter) will have different tendency (Median) to produce errors.

$\text{Median}_S \neq \text{Median}_A$ (where S = Singleton and A= Adapter)

4.5 Fifth Hypothesis

Null Hypothesis:
The pair of patterns (Factory, Composite) will have same central tendency (Median) to produce errors.

$\text{Median}_F = \text{Median}_C$ (where F=Factory and C = Composite)

Alternate Hypothesis:
The pair of patterns (Factory, Composite) will have different tendency (Median) to produce errors.

$\text{Median}_F \neq \text{Median}_C$ (where F=Factory and C = Composite)

4.6 Sixth Hypothesis

Null Hypothesis:
The pair of patterns (Factory, Adapter) will have same central tendency (Median) to produce errors.

$\text{Median}_F = \text{Median}_A$ (where F = Factory and A= Adapter)

Alternate Hypothesis:
The pair of patterns (Factory, Adapter) will have different tendency (Median) to produce errors.

$\text{Median}_F \neq \text{Median}_A$ (where F = Factory and A= Adapter)

4.7 Mann-Whitney U Test

For the purpose of checking independency of each possible pairs of design patterns (6 pairs in this case), Mann-Whitney U test in SPSS is performed. The variable 'Bug Count' is selected as "Test Variable" and 'Pattern' is selected as "Group Variable". There are four patterns i.e. "Singleton, Factory, Composite, Adapter" and their possible pairs are shown below:

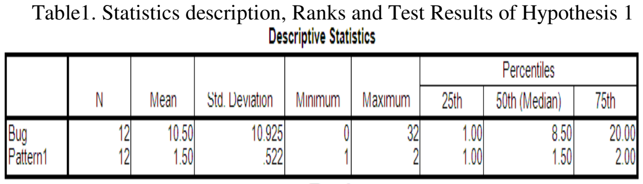
1. SF (Singleton, Factory)
2. CA (Composite, Adapter)
3. SC (Singleton, Composite)
4. SA (Singleton, Adapter)
5. FC (Factory, Composite)
6. FA (Factory, Adapter)

4.8 Results

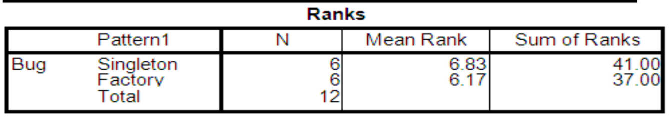
After applying Mann-Whitney U test, following results have been found.

1). (Pattern1 (Singleton, Factory)

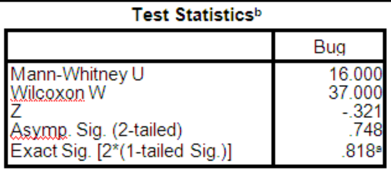
Table1. Statistics description, Ranks and Test Results of Hypothesis 1


Descriptive Statistics

	N	Mean	Std. Deviation	Minimum	Maximum	Percentiles		
						25th	50th (Median)	75th
Bug	12	10.50	10.925	0	32	1.00	8.50	20.00
Pattern1	12	1.50	.522	1	2	1.00	1.50	2.00


Ranks

	Pattern1	N	Mean Rank	Sum of Ranks
Bug	Singleton	6	6.83	41.00
	Factory	6	6.17	37.00
	Total	12		


Test Statistics[b]

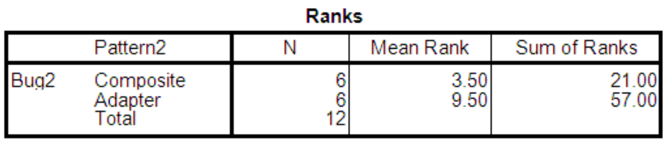
	Bug
Mann-Whitney U	16.000
Wilcoxon W	37.000
Z	-.321
Asymp. Sig. (2-tailed)	.748
Exact Sig. [2*(1-tailed Sig.)]	.818[a]

Since (p-value= 0.748 > 0.05 = α), the null hypothesis can't be rejected

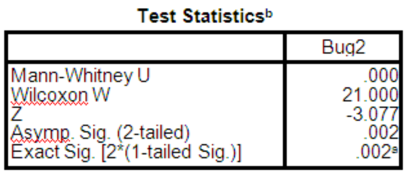
Conclusions: At the α = 0.05, there is enough evidence to conclude that there is no difference in the median bug count of the two patterns.

2). Pattern2 (Composite, Adapter)

Table 2. Test Rank and Results of Hypothesis 2


Ranks

	Pattern2	N	Mean Rank	Sum of Ranks
Bug2	Composite	6	3.50	21.00
	Adapter	6	9.50	57.00
	Total	12		


Test Statistics[b]

	Bug2
Mann-Whitney U	.000
Wilcoxon W	21.000
Z	-3.077
Asymp. Sig. (2-tailed)	.002
Exact Sig. [2*(1-tailed Sig.)]	.002[a]

Since (p-value= 0.002 < 0.05 = α), the null hypothesis can't be accepted.

Conclusions: At the α = 0.05, there is enough evidence to conclude that there is difference in the median bug count of the two patterns.

3). Pattern3 (Singleton, Composite)

Table 3. Test Rank and Results of Hypothesis 3

Ranks

	Pattern3	N	Mean Rank	Sum of Ranks
Bug3	Singleton	6	9.00	54.00
	Composite	6	4.00	24.00
	Total	12		

Test Statistics[b]

	Bug3
Mann-Whitney U	3.000
Wilcoxon W	24.000
Z	-2.678
Asymp. Sig. (2-tailed)	.007
Exact Sig. [2*(1-tailed Sig.)]	.015[a]

Since (p-value= 0.007 < 0.05 = α), the null hypothesis can't be accepted.

Conclusions: At the α = 0.05, there is enough evidence to conclude that there is difference in the median bug count of the two patterns.

4). Pattern4 (Singelton, Adapter)

Table 4. Test Rank and Results of Hypothesis 4

Ranks

	Pattern4	N	Mean Rank	Sum of Ranks
Bug4	Singleton	6	4.17	25.00
	Adapter	6	8.83	53.00
	Total	12		

Test Statistics[b]

	Bug4
Mann-Whitney U	4.000
Wilcoxon W	25.000
Z	-2.242
Asymp. Sig. (2-tailed)	.025
Exact Sig. [2*(1-tailed Sig.)]	.026[a]

Since (p-value= 0.025 < 0.05 = α), the null hypothesis can't be accepted.

Conclusions: At the α = 0.05, there is enough evidence to conclude that there is difference in the median bug count of the two patterns.

5). Pattern5 (Factory, Composite)

Table 5. Test Rank and Results of Hypothesis 5

Ranks

	Pattern5	N	Mean Rank	Sum of Ranks
Bug5	Factory	6	9.00	54.00
	Composite	6	4.00	24.00
	Total	12		

Test Statistics[b]

	Bug5
Mann-Whitney U	3.000
Wilcoxon W	24.000
Z	-2.678
Asymp. Sig. (2-tailed)	.007
Exact Sig. [2*(1-tailed Sig.)]	.015[a]

Since (p-value= 0.007 < 0.05 = α), the null hypothesis can't be accepted.

Conclusions: At the α = 0.05, there is enough evidence to conclude that there is difference in the median bug count of the two patterns.

6). Pattern6 (Factory, Adapter)

Table 6. Test Rank and Results of Hypothesis 6

Ranks

	Pattern6	N	Mean Rank	Sum of Ranks
Bug6	Factory	6	4.17	25.00
	Adapter	6	8.83	53.00
	Total	12		

Test Statistics[b]

	Bug6
Mann-Whitney U	4.000
Wilcoxon W	25.000
Z	-2.242
Asymp. Sig. (2-tailed)	.025
Exact Sig. [2*(1-tailed Sig.)]	.026[a]

Since (p-value= 0.025 < 0.05 = α), the null hypothesis can't be accepted.

Conclusions: At the α = 0.05, there is enough evidence to conclude that there is difference in the median bug count of the two patterns.

5. Conclusions

Design patterns are reusable solutions to frequently occurring problems in software design. The expert designers like to use previously made solutions for their problems instead of spending time on recreating the solutions. But, obviously these solution are not completely error free. For this, we have conducted this study to find out which patterns is more error prone. The Mann-Whitney U Test' results reject our null hypothesis in all the cases except first one (i.e. Singleton, Factory). Therefore it is concluded that each pattern has independent tendency to produce errors. Now for evaluating that which pattern is more error prone, the results of Mann-Whitney U Test are analyzed. For this purpose, the Mean Rank value of each pattern in each group is highlighted and their average has been calculated.

Table 7. Patterns and Mean Rank value of each pattern

Pattern Name	Average Value
Singleton	6.66
Factory	6.44
Composite	3.83
Adapter	9.50

These values show that more error prone pattern among the four patterns (Singleton, Factory, Composite, Adapter) is Adapter pattern. One other reason of being more error prone is that it is excessively used in all the projects especially Hibernate Project. So when a pattern will be excessively used then its tendency of producing errors would be more for sure.

References

[1]. K. N. Loo, S. P. Lee, "Representing design pattern interaction roles and variants", ICCET, 2010 2nd International Conference on , vol.6, no., pp.V6-470-V6-474, 16-18 April 2010

[2]. D. Budgen, Software Design, Second ed. Essex, England: Pearson Education Limited, 2003.

[3]. E. Agerbo, A. Cornils, "How to preserve the benefits of Design Patterns," ACM SIGPLAN Notices, vol. 33, pp. 134-143, 1998.

[4]. E. Gamma, R. Helm, R. Johnson et al., "Design Patterns: Elements of Reusable Object-Oriented Software". Addison-Wesley. ISBN 0-201-63361-2.

[5]. K. Beck, W. Cunningham, "Using Pattern Languages for Object-Oriented Program". OOPSLA '87 Retrieved 2006-05-26.

[6]. R. Ferenc, A. Beszedes, L. Fulop et at.,"Design Pattern Mining Enhanced by Machine Learning", 25-30 September 2005, Budapest, Hungary

[7]. L. Aversano, L. Cerulo, M.D. Penta, "The relationship between design patterns defects and crosscutting and cross cutting concern scattering degree", Software, IET, vol.3, no.5, pp.395-409, October 2009

[8]. M. Vokac, "Defect frequency and design patterns: an empirical study of industrial code," Software Engineering, IEEE Transactions on , vol.30, no.12, pp. 904- 917, Dec. 2004

[9]. Y.G. Gue´he´neuc, H. Albin-Amiot, "Using Design Patterns and Constraints to Automate the Detection and Correction of Inter-Class Design Defects," Proc. 39th Int'l Conf. and Exhibition Technology of Object-Oriented Languages and Systems, pp. 296-305, 2001.

[10].J. Dong, Y. Zhao, Y. Sun." A Matrix-Based Approach to Recovering Design Patterns", IEEE Transaction on systems and cybernetics-part A: systems and humans, vol 39, NO 6.November 2009.

[11].N. Moha, Y. Gueheneuc, L. Duchien et al.,"Discussion on the Results of the Detection of Design Defects". ECOOP workshop on Object-Oriented Reengineering, July--August 2007. Springer-Verlag

[12].N. Moha, Y. Gueheneuc, L. Duchien et al.,"On the Automatic Detection and Correction of Software Architectural Defects in Object-Oriented Designs". ECOOP Workshop on Object-Oriented Reengineering, July 26, 2005, Universities of Glasgow and Strathclyde, Glasgow, UK

[13]. N. Moha, D. Huynh, Y. Guéhéneuc., "A Taxonomy and a First Study of Design Pattern Defects". Proceedings of the STEP International Workshop on Design Pattern Theory and Practice, September 25-30, 2005, Budapest, Hungary.

Mamoona Jalil was awarded a Bachelor's degree in Computer Sciences from the University of Sargodha and currently doing M.S. in Computer Sciences from the same university. Her research interests include Software Engineering, Design Pattern and other topics.

Javed Farzand received a Master degree in Computer Sciences in 2003 and a Doctor of Informatics from Technical University of Graz, Austria in 2009. His research interests include Design Pattern, Software Testing and e-learning. He is currently an assistant professor in the Department of Computer Science and Information Technology at the University of Sargodha, Pakistan.

Muhammad Ilyas received a Master degree in Software Project Management in 2004 from National University of Computer and Emerging Sciences, Lahore and a Doctor of Informatics from Johannes Kepler University, Linz Austria in 2010. His research interests include Software Engineering, Design Pattern and knowledge base systems. He is currently an assistant professor in the Department of Computer Science and Information Technology at the University of Sargodha, Pakistan.