Viewing Change History of Design Patterns

BY

Anila

M.Sc. COMPUTER SCIENCE

2012-2014

DEPARTMENT OF COMPUTER SCIENCE

QUAID-I-AZAM UNIVERSITY

ISLAMABAD

بسم الله الرحمن الرحيم

In the Name of Allah, Most Compassionate, Ever-Merciful

A dissertation submitted to the

Department of Computer Science,

Quaid-i-Azam University, Islamabad

As a partial fulfillment of the requirements

For the award of the degree of

Master in Computer Science

DEDICATION

My every good deed firstly dedicated to **ALLAH** Almighty for making me capable of doing this, my loving parents specially my mother who put her trust in me.

DECLARATION

I hereby declare that this report is my own work and effort and that it has not been submitted anywhere for any award. Where other source of information has been used, they have been acknowledged.

Anila

ACKNOWLEDGEMENT

Thanks to Almighty Allah for giving me knowledge, power and strength to complete this project. I have learned a lot during this project and this will certainly help me in my upcoming life. I really thankful to the help and support of **Dr. Onaiza Maqbool** who is my project supervisor in this project. She helped me in all phases of this project. Her supervision helped me a lot during the time of difficulties. In the end i would like to thank my teachers, parents and specially Bhai and Bhabhi g because without their love and affection during studies, this project was not achievable.

Anila

ABSTRACT

The project is related to development of a unified environment/tool that extracts Patterns information from different successive versions of a software, and provides support for visual display in the form of diagrams based on extracted information of any software system. The tool shows the version to version growth of Patterns in the system in statistical form and also shows the trends in architecture. These trends help users to predict the growth of patterns in the software.

Contents

Chapter # 1

Introduction

1.1 Introduction

This chapter provides an overview of design patterns and project objective and scope.

1.1 Design patterns overview

It is essential to understand design patterns concepts in order to understand the objectives of this project.

1.1.1 Introduction to Design Patterns

A design pattern is a repeatable solution to a software engineering problem. Design patterns are widely used by software developers because patterns give software designers the ability to design a robust solution rapidly by using proven techniques [1]

1.1.2 Origin

The term “Pattern" was first used by an architect Christopher Alexander in the late 1970's. In [2] the concept of the pattern is described as

“Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a billion times over, without ever doing it the same way twice."

In [3], the concept of pattern is modified as:

“Each pattern is a three part rule, which expresses a relation between a context, a problem and a solution."

1.1.3 Role of Design Patterns in Software Design

A software design pattern is a general solution to a common problem in software design. It is a description or template for how to solve a problem that can be used in different situations. A design pattern typically shows relationship and interaction between classes or objects, without specifying final application classes or objects that are involved. Patterns identify and specify abstractions that are above the level of single classes and instances [1]

1.2 Problem Definition

Design patterns have been used in software systems to incorporate quality. They make our code easy to understand and debug. This leads to faster development and new members of team understand the system easily. Design patterns generally change and improve over time as software is updated.

Manually checking these changes in different versions is a difficult task and we cannot determine what and where changes occur in different versions of software.

1.3 System Definition

The proposed system, is an application that will read details about the detected design patterns and then present them in a meaningful manner for comparison between versions and among various software systems. So researchers or architects/designers can predict the growth of patterns systems and do corrective effort on architecture/design.

1.3 Scope

System will provide the following functionality to the user

- System will allow the user to select different versions of a software
- System will automatically detect design patterns from different categories
- System will show statistical and trend analysis for comparison
- System will allow user to view change history in different ways
- System will allow user to apply search

1.3.1 Input

- Versions of software

1.3.2 Output

- Statistical and graphical form of software design pattern history

I have used 12 design patterns in this project. Their detail is provided below

1.4 Catalog of Design Pattern

The design pattern catalog by Gamma et al. [1] Contains different Design Patterns from which I chose 12 design patterns from the Creational, structural and behavioral categories.

1.4.1 Creational patterns

These design patterns are about class instantiation. These patterns can be further divided into class-creation patterns and object-creational patterns. While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.

- **Factory Method**

Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

- **Prototype**

Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

- **Singleton**

Ensure a class only has one instance, and provide a global point of access to it.

1.4.2 Structural patterns

These design patterns are about Class and Object composition. Structural class-creation patterns use inheritance to compose interfaces. Structural object-patterns define ways to compose objects to obtain new functionality.

- **Command**

Encapsulate a request as an object, thereby letting one to parameterize clients with different requests, queue or log requests, and support undoable operations.

- **Composite**

Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

- **Decorator**

Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to sub classing for extending functionality.

1.4.3 Behavioral patterns

These design patterns are about Class's objects communication. Behavioral patterns are those patterns that are most specifically concerned with communication between objects.

- **Observer**

Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

- **State**

Allow an object to alter its behavior when it’s internal state changes. The object will appear to change its class.

- **Strategy**

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

- **Template Method**

Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

- **Visitor**

Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.

1.5 Benefits of design Patterns

Below are listed some benefits of design patterns [4]

1. Enhances code readability

Design patterns help to speed up software development by providing tested development techniques. Software design involves considering small issues that might become visible later during the implementation process. Reusable design patterns help to correct problems and enhance code readability.

2. Robust

Using design patterns promotes reusability that leads to more robust and highly maintainable code.

3. Solutions to specific problems

They give the developer a selection of tried and tested solutions to work with.

4. Simplify the coding process

If user have knowledge about basic design patterns.it will be easier to divide a task into pieces, each with its own specific responsibilities.

5. Enhances software development

Design patterns helps to enhance software reusability and development. In other words, a design pattern is the main component in software development. Better understanding of how design patterns relate to each other will help enhance software development

Chapter # 2

Requirements Specification

2. System Requirements Specification

Software requirement specification is the process of analyzing the requirement of the system. It includes all the requirements of the stakeholders. This chapter describes the user's specification and use cases and Domain model.

2.1 Stakeholder & Interest

Stakeholder	Researchers
Interests	Researchers want to extract Design patterns and see the changes in any software system in visual format (Diagrams) for the maintenance and understanding of design of open source systems. They want to predict the growth in design patterns of open source systems.

Stakeholder	Architects/Designers
Interests	Architects/Designers also want to see trend analysis of any open source system. They also want to predict the growth in design of open source systems to make corrective effort on architecture.

Stakeholder	Open source community developers
Interests	Open source community developer also wants to see the design changes between different versions in open source system.

2.2 User Goals

Researchers	Extract Patterns.
Designers	View Trend Analysis.
Open source community developers	Visualize Difference in versions.

2.3 Usecase diagram

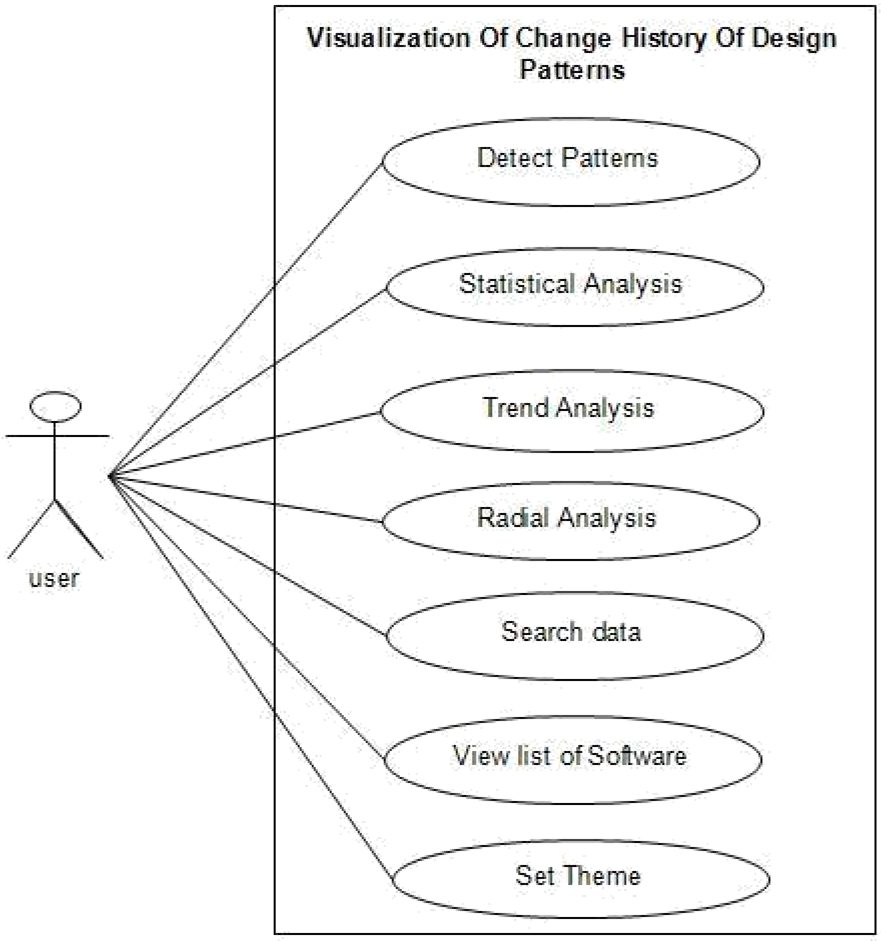


Fig 2.3 **Use Case Diagram**

Use case description

Use Case UC1: Detect Patterns

Primary actor: user

Stakeholder and interest: Researchers /Architects/Designers/ Open source community developers: Detect the Design patterns of many successive versions of the software.

Success Guarantee:

- Patterns of all the versions detected successfully.
- All required reports are successfully generated or exported in specific directory.

Basic Flow (Main Flow of Events)

User Request	System Response
1-This use case begins when user wishes to detect patterns in successive versions of any open source software.	
2-The user indicates to the system that he/she wants to detect the patterns in different version of software.	3- System asks the user to browse the path of the software.
4- User browses the path of the software.	5- System validates the path of the software.
	6- System asks the user to start the detection.
7- User starts detection.	8- System starts the patterns detection on different versions.
	9- System detects the patterns information.
	10- System detects the instance under each pattern.
	11- System detects the roles under each instance.
Step 8-11 repeat until design information of all successive versions of the given software detect.	
	12- System define a directory and saves the detected results in it.
	13-System start extraction (System Function Extract Data)
	14- System shows the patterns detection confirmation.

<u>Extension</u>:

a*.Any time while detection when error occurs:

- System Stop the detection process and shows or display a message to user, indicating that the browse software contain some incorrect file.

b*. User browses invalid directory structure software:

- System prompts the user to browse the defined directory structure software.

<u>Special Requirement:</u>

- Successive versions of the software should be present in specific directory.
- Successive versions files of the software should be in java source code form.
- Successive versions files of the software should be present in specific structure like that
    - Software
        - Version_1
        - Version_2
        - Version_3
- User must select the root directory i.e. software.

Use Case UC2: Trend Analysis

Primary actor: user

Stakeholder and interest:

Researchers /Architects/Designers/ Open source community developers: They want to see trend analysis and predict the growth in design of open source systems to make corrective effort on design.

Pre-condition:

- Design patterns of all the successive versions of the software must be extracted successfully.

Success Guarantee:

- All type of trend analysis shown successfully.
- Different views of trend (line and dot view or bar chart view)

Basic Flow (Main Flow of Events):

User Request	System Response
1-This use case begins when user wishes to see trend analysis on extracted design of versions of open source software.	
2-The user indicates to the system that he/she Wants to see trend analysis on extracted design of the successive versions of open source software.	
	3- Using extracted results system displays trend analysis graphically.
4-User view the environment.	
	5-System allows the user to see the trend analysis on pattern and instances
6- User view the trend on instances and patterns.	
	7- System allows the user to see different view of trend analysis (dot and line or bar chart view)
8-User selects the trend analysis view.	
	9- System displays the trend analysis view.

Use Case UC3: Statistical Analysis

Primary actor and interest: System allows the user to see the extracted data in statistical form.

Pre-condition:

- Design elements of all the successive versions of the software must be extracted successfully.

Success Guarantee:

- System will show statistical view successfully.

Basic Flow (Main Flow of Events):

User Request	System Response
1-This use case begins when user wishes to see Statistical analysis on extracted design of versions of open source software. 2-The user indicates to the system that he/she wants to see Statistical analysis on extracted design of the successive versions of open source software. 4-user see the Statistical view.	3-System will show statistical data.

Use Case UC4: Search data

Primary actor and interest: system allow user to apply search on different version of a software.

Pre-condition:

- System will show statistical view.

Success Guarantee:

- System will show search statistical successfully.

Basic Flow (Main Flow of Events):

User Request	System Response
1-This use case begins when user wishes to see apply search on extracted design of versions of open source software. 2-The user indicates to the system that he/she Wants to apply search on extracted design of the successive versions of open source software. 4-User selects different option for making query	3-System will show search environment allow user to make query. 5-System show the searched results.

Use Case UC5: Show Radial View

Primary actor and interest: System allow user to see data in radial view.

Pre-condition:

- Intermediate results file must be present in specific directory.

Success Guarantee:

- System will show Radial view successfully.

Basic Flow (Main Flow of Events):

User Request	System Response
1-This use case begins when user wishes to see visual analysis on extracted design of versions of open source software. 2-The user indicates to the system that he/she Wants to see Radial view on extracted design of the successive versions of open source software. 4-User can zoon in/zoom out the view. 6-User sees the Radial view.	3-Using intermediate results system show the Radial view. 5-System zoom in/zoom out the view.

Use Case UC6: View list of Software

Primary actor and interest: User wants to see the software list along with different versions on which different operations will be performed.

Pre-condition:

- Different software and versions must be present in specific directory.

Success Guarantee:

- List of software and versions shown successfully.

Basic Flow (Main Flow of Events):

User Request	System Response
1-This use case begins when user wishes to see the list of software along versions. 2-The user indicates to the system that he/she wants to see list of software. 6-User view list of software.	3-System checks that default directory exists. 4.System defines directory if not exist 5.System displays list of software along with versions

Extension:

3. a. Verification of path of defined directory:

- System prompts the user to set the path of define directory in which software present.

Special Requirement:

Software along version must be present in specific directory

Use Case UC7: Set theme

Primary actor: user

Stakeholder and interest: User wants to change the default environment and set different themes.

Success Guarantee: New theme applied successfully.

Basic Flow (Main Flow of Events):

User Request	System Response
1-This use case begins when user wishes to change the default theme.	
2-The user indicates to the system that he/she wants to set the theme.	
	3-System display the environment for theme settings.
4-User will select different options.	
	5-System will apply changes and display the view.
6-User views the environment.	

System Function: Extract data

Level: System goal

Primary actor: System extracts the data of many successive versions of the software.

Pre-condition:

- Detection results of successive versions of the software should be present in the defined directory Structure.

Success Guarantee:

- Patterns, instances and roles of all the versions extracted successfully.
- Intermediate result file generated successfully.

System Function
1- After detection of the successive versions next step is to extract data from the detection files of different version of any open source software. 2- System starts the patterns extraction on a version. 3- System extract the patterns information. 4- System extract the instance under each pattern. 5-System extract the roles under each instance.
Step 2-5 repeat until information of all successive versions of the given software extracted.
5- On the basis of theses extraction system generate an intermediate result file. 6- System saves the extraction results.

2.5 Domain Model

"A domain model in problem solving and software engineering is a conceptual model of all the topics related to a specific problem. It describes the various entities, their attributes, roles, and relationships, plus the constraints that govern the problem domain" [5]

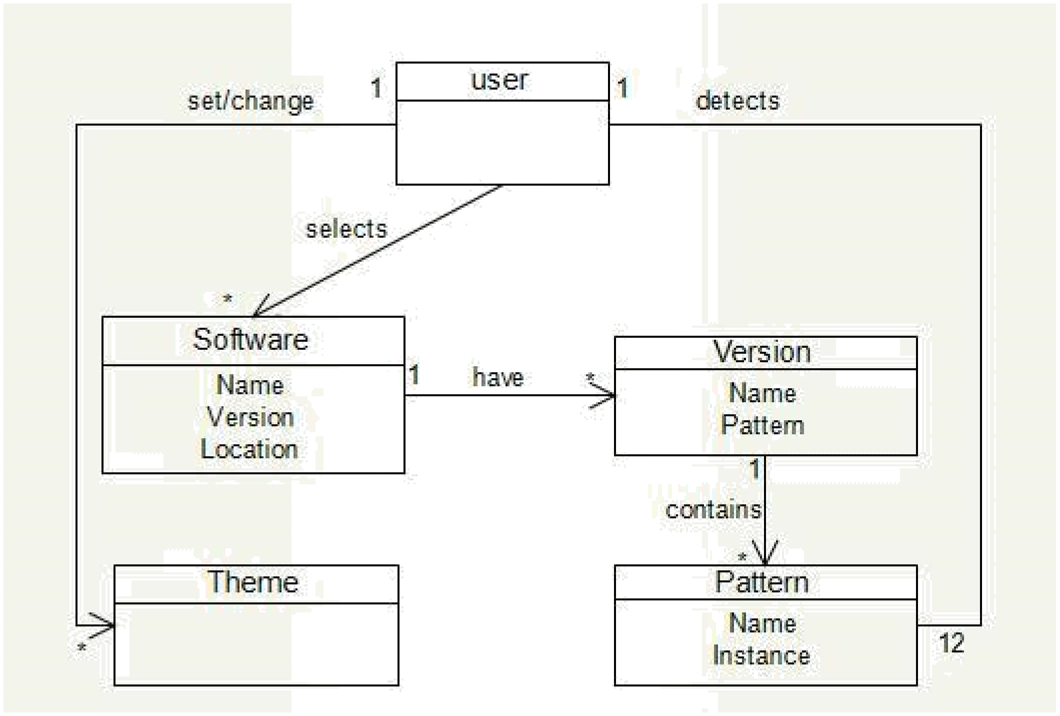


Fig 2.5 **Domain Model**

Chapter # 3

System Design

3. System Design

Once requirements are gathered next phase of software engineering is to design the system. Design is the first step in the development phase for any engineering product or system. It is the process of applying different techniques and rules for defining the system details to clear it.

The design of the system is explained with the help of the following diagrams:

- System Architecture
- System Flow Diagram
- Detailed Design
    - Sequence diagram
    - Class diagram

3.1 System Architecture

System Architecture defines the infrastructure of system, that is, it tells that what elements are related to one another in what manner. How the communication inside a system is being handled? This system is developed on the "Three Layer Architecture". Use of "Three Layer Architecture" make the application more understandable and easy to maintain and easy to modify.

3.1.1 Three layers architecture:

Three layer Architecture indicates a physical separation of components. The system is divided into three layers (parts); Backend Database System (or some directory in which software along versions are present), Middle layer related to business logic layer and Front End Application.

3.1.2 Architecture Diagram

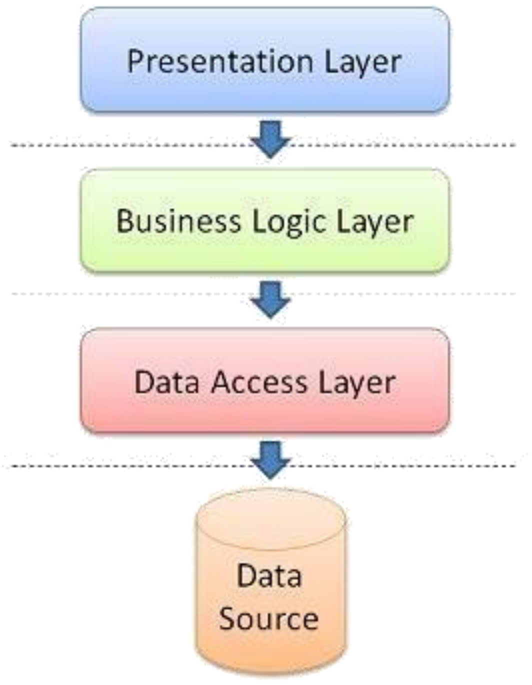


***Fig 3.1.2* Architecture diagram**

3.1.2 Architecture Overview

In three layer architecture, each layer runs independently of the others and performs a specific role in the application.

The ***Presentation Layer*** is responsible for human interaction with the application. Its primary concern is data visualization and searching.

The ***Business Logic Layer*** processes requests from the client to detect patterns and export detected results in some directory. Extract data and to perform other functions.

The layer's primary role is to shield direct access to the data from unwanted and invalid reads and updates. This helps to ensure the long-term integrity and security of the data.

The ***Data Access Layer*** is responsible for durable storage of the application layer.

3.2 System Flow Diagram

Flow diagram is a collective term for a diagram representing a flow or set of dynamic relationships in a system. Flow diagrams are used to structure and order a complex system, or to reveal the underlying structure of the elements and their interaction. [5]

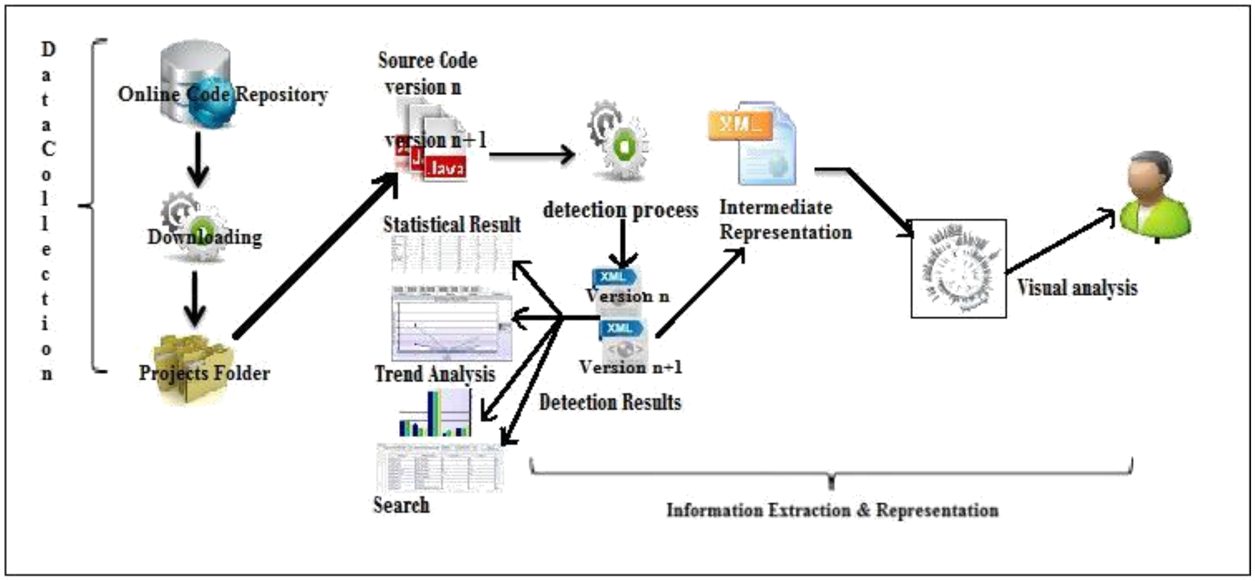


Fig 3.2 **System Flow Diagram**

1. User first download software along versions.
2. Place all the software in specific directory and specific format as described above.
3. System will detect patterns from different versions and store results respectively.
4. Using these detected results system will show Statistical, Trend, and graph view.
5. Using these detected results system will also generate an intermediate file which can be further used for radial analysis.

3.3 Detailed Design

In Detailed design, flow of the system is described using sequence diagrams. Also classes are defined and their relationship is shown using class diagram.

3.3.1 Sequence Diagram

Sequence diagrams in the UML are primarily used to model the interactions between the actors and the objects in a system and the interactions between the objects them-selves. As the name implies, a sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance.

Detect Pattern

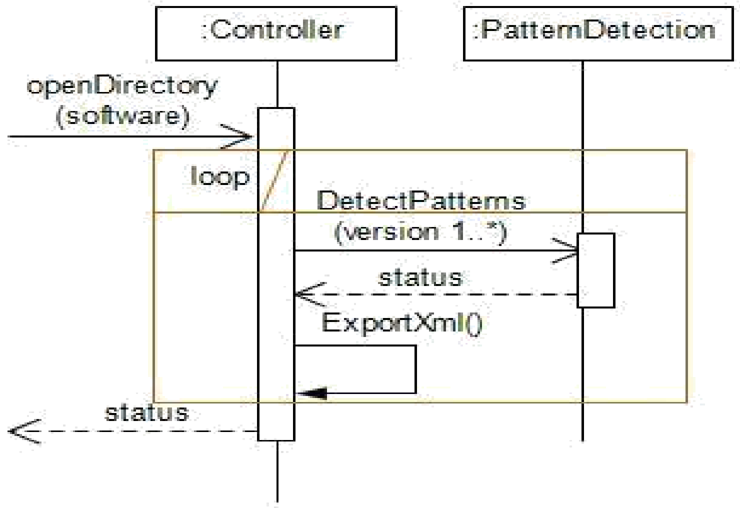


Fig 3.2.1

Trend Analysis

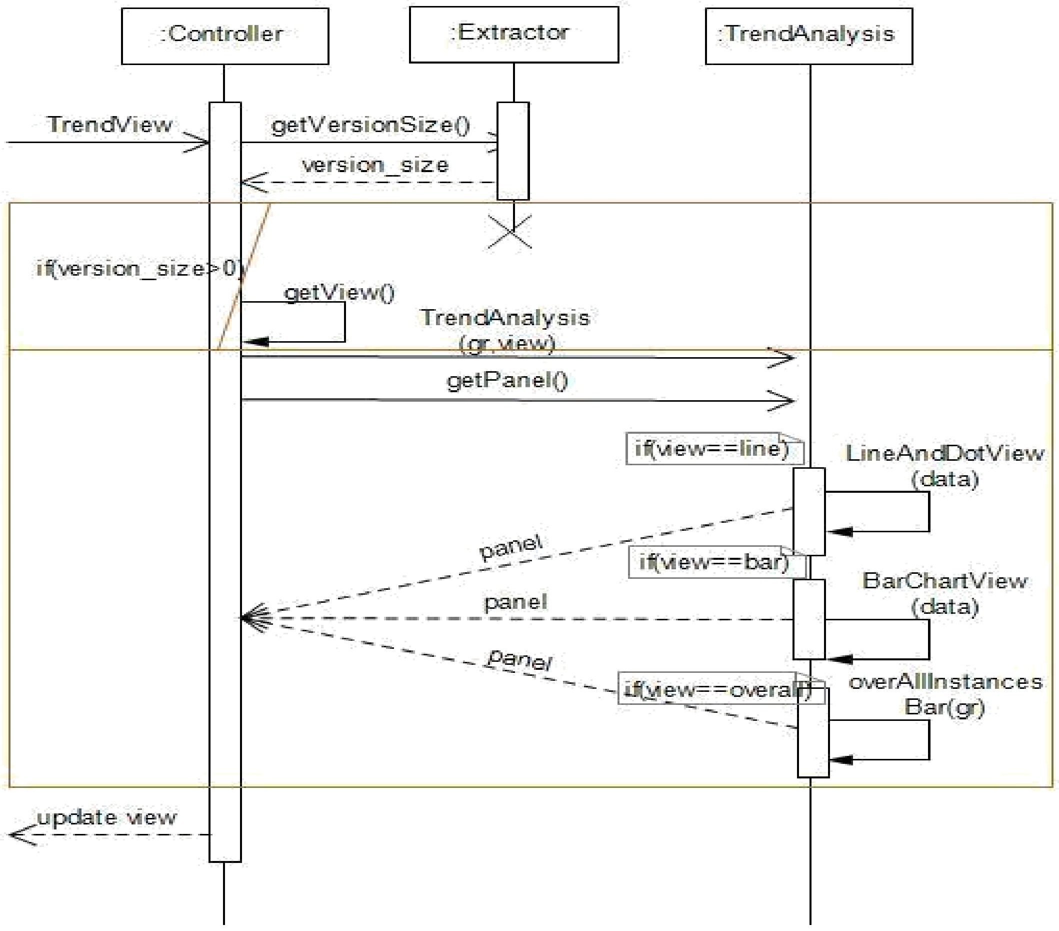


Fig 3.2.2

Search data

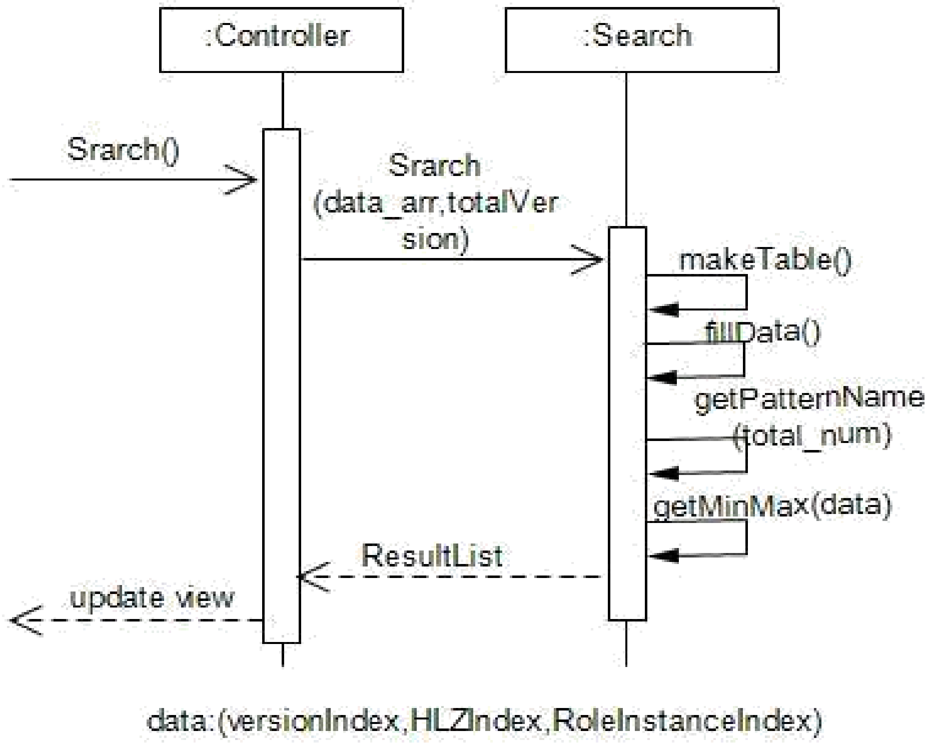

:Controller
:Search
Srarch()
Srarch (data_arr,totalVersion)
makeTable()
fillData()
getPatternName (total_num)
getMinMax(data)
ResultList
update view
data:(versionIndex,HLZIndex,RoleInstanceIndex)

Fig 3.2.4

System Function:

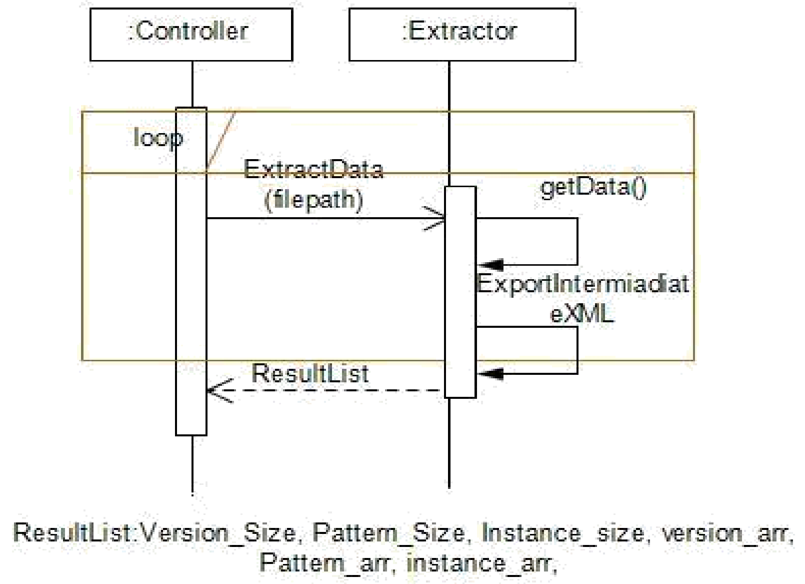

:Controller
:Extractor
loop
ExtractData (filepath)
getData()
ExportIntermiadiateXML
ResultList
ResultList:Version_Size, Pattern_Size, Instance_size, version_arr, Pattern_arr, instance_arr,

Fig 3.2.7

Remaining sequence Diagrams are in **Appendix.**

3.2.2 Class Diagram

"Class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations and the relationships among the classes." [6]

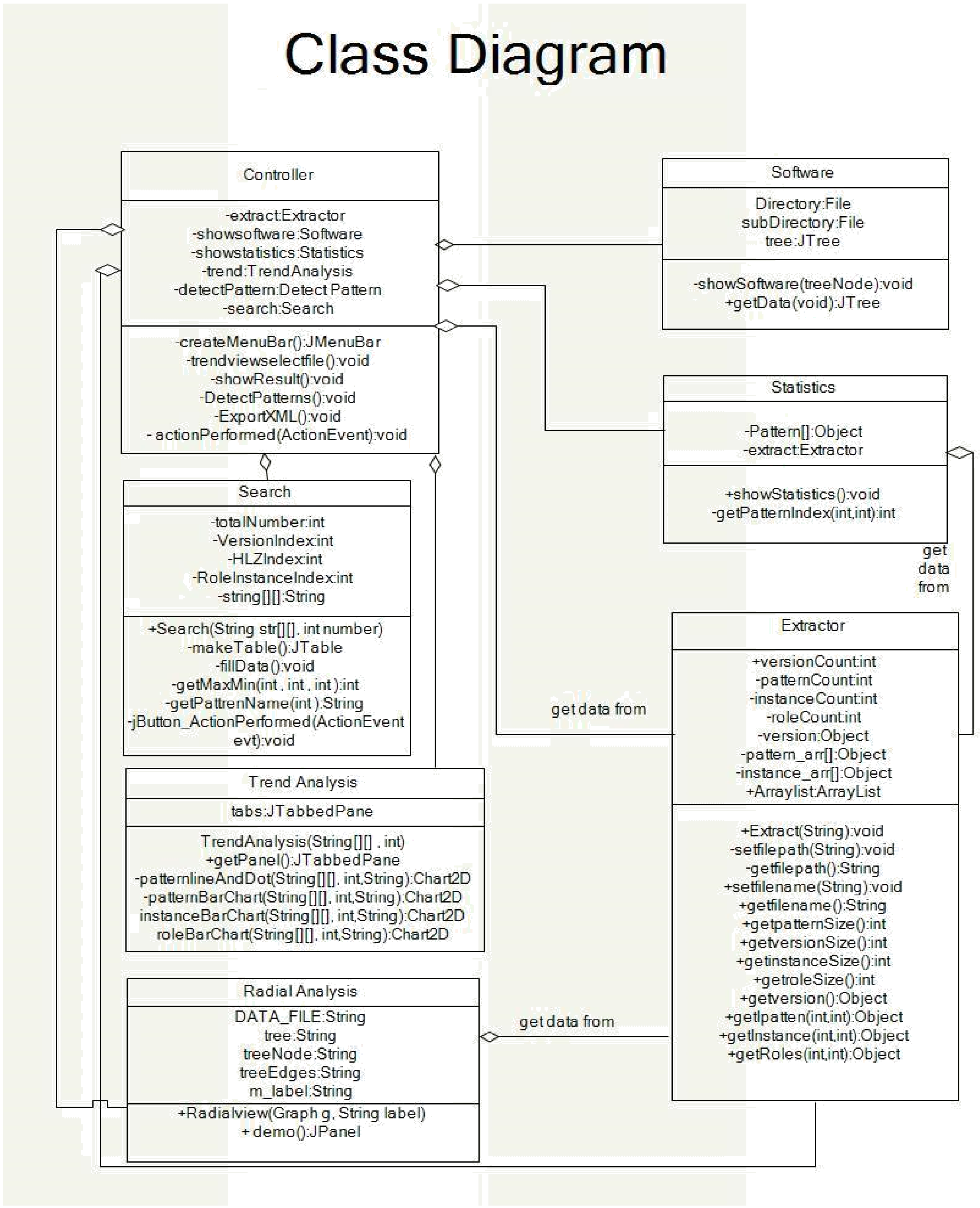


Fig 3.2.2 **Class Diagram**

Chapter # 4

System Implementation

4. System Implementation

After design phase there is an implementation phase. At this stage you create an executable version of the software. Implementation may involve developing programs in high- or low-level programming languages or tailoring and adapting generic, off-the-shelf systems to meet the specific requirements of an organization.

4.1 Methodology

I applied following methodology to complete this system.

1. Design patterns were studied.
2. Tool for design pattern detection was searched.
3. Selecting one of those design pattern detection tool, as each of them had different limitations.
4. The limitation of the selected tool was that user has to select one version of software at a time and then select the pattern form a list and its results are then exported. The user has to select the next version of the software again and do the same.
5. In proposed system, the user selects the whole directory having different versions of the software and the design patterns detection are applied on all versions and results are saved.
6. XML parser is applied to extract data from these results one by one and that data is used for analysis views.
7. The analysis views were generated by using many libraries.
8. A list of themes was provided for analysis views.

4.2 Framework Selection

This system is developed using Java with Net Beans IDE. It is a desktop based application created in Java language and is very easy to use.

4.3 Language Selection

After the selection of framework the next step is to decide a suitable language for our project. Suitable means that the language must be flexible enough to support design. Things that should be taken care of while selecting a language is:

- Design is object oriented or structured
- System is Web based or Desktop based

For my project, the system is **object oriented** and **Desktop-based**.

Java

I use Java to implement my project because the detection tool that I used for detection of patterns is in java language.

Java is an elegant language that enables a developer to build a variety of applications that can run on any platform. It is platform independent means the code that runs on one machine does not need to recompile to run on another.

4.4 Detection Tool

DPD

DPD is used to detect patterns from different versions of the software [7].

User must first select the root of the project package. Only class files (bytecode files) are necessary for the project analysis. Designs patterns are detected using similarities basis and then detected results are saved in xml format.

4.5 Libraries Used

As to increase functionality and decrease the development time of the application following libraries are used in the system

Chart2D

Chart2d is the open source java library that is used for visualization of quantitative data. It provides chart types: pie, line, bar, scatter plot (or dot), combination (dot and line). It supports setting of maximum, minimum, and preferred sizes and colors, and auto-calculation of pref. size and color of different categories.

Prefuse

Prefuse is a Java-based extensible software framework for creating interactive information visualization applications. It can be used to build standalone applications. The intends to use Prefuse is to simplify the processes of visualizing, handling and mapping of data, as well as user interaction.

Prefuse features:

- Graph, and tree data structures supporting arbitrary data attributes, data indexing, and selection queries.
- Components for layout, color, size, and zooming.

Look And Feel

Libraries including elements such as colors, shapes, layout, and typefaces , as well as the behavior of dynamic elements such as buttons, boxes, and menus are called look and feel libraries. Different libraries are used for this purpose.

4.6 Limitation of system

1. System will only detect patterns from java based software.
2. System will only detect patterns from .java files present in the version’s folder.
3. System will detect patterns if software versions are placed in specific path.
4. System will only detect twelve design GOF patterns.

Chapter # 5

Testing and Conclusions

5.1 Testing

Testing is the process of detecting errors. Testing plays a critical role in assuring quality and reliability of software. The results of testing are used later on during maintenance also.

5.1.2 Psychology of Testing

The basic purpose of testing phase is to detect the errors that may be present in the program.

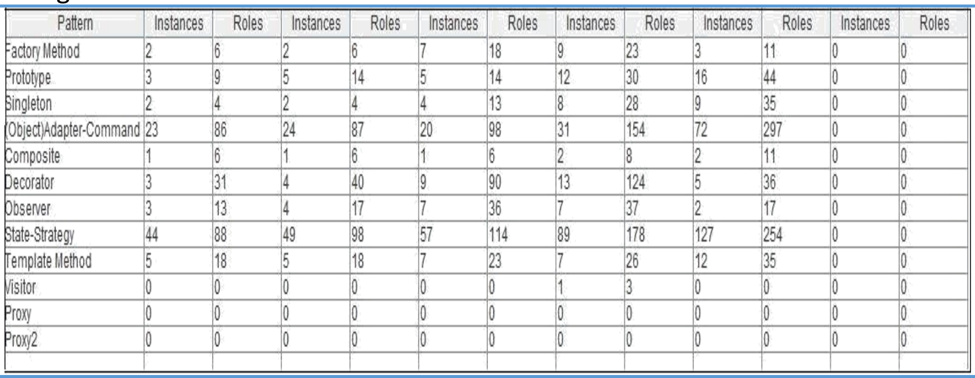
5.1.2 Acceptance Testing

The last phase of software development process is User Acceptance Testing (UAT). Acceptance Testing is done by real software users to test that the software meets its requirements. Different test cases are made keeping in mind the use cases of the system and is assigned to a user to test it. User runs the software according to the given instructions to see if the software fulfils its expected output.

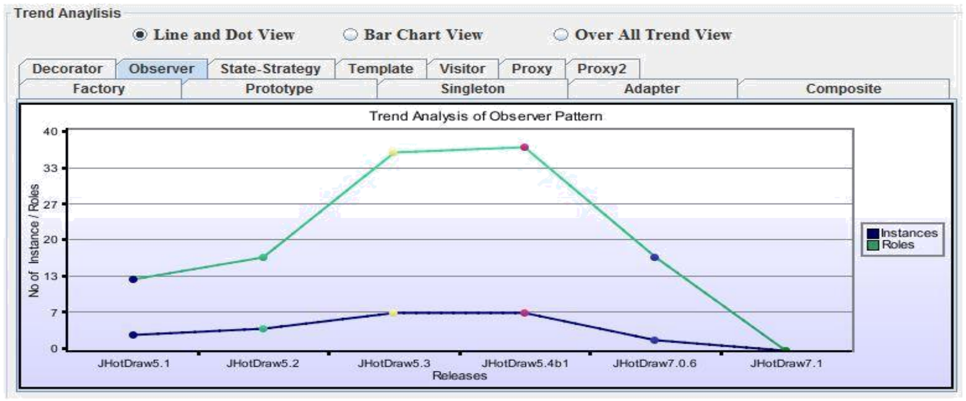
5.1.3 Test Cases

The tests cases performed during the project development which are verified successfully are as follows:

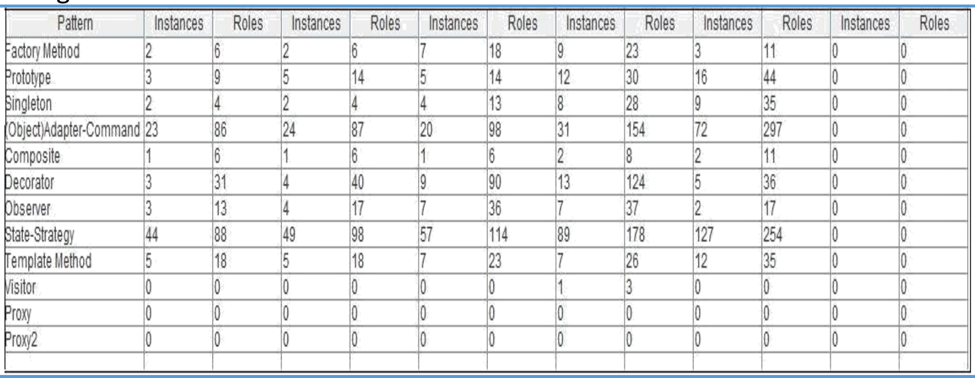
Test ID	T001
Use case	Detect pattern
Description	Verification of software path in detection of patterns, that user must select the root directory of a software.
Setup	Directory contains: JhotDraw JhotDraw5.2 JhotDraw7.3 JhotDraw7.3.1
Pre-Requisite	User must know the path of the software
Instruction No 1	Select Extract data from File Browse the path Select the root directory of the software i.e. JhotDraw Select open
Expected Result 1:	System should start the Extraction process
Observed Result 1:	System starts the Extraction process
Verdict	PASS

Test ID	T002
Use case	Trend Analysis
Description	User want to see trend analysis on different patterns against successive versions of a software.
Pre-Requisite	Design of the software must be extracted first.
Setup	

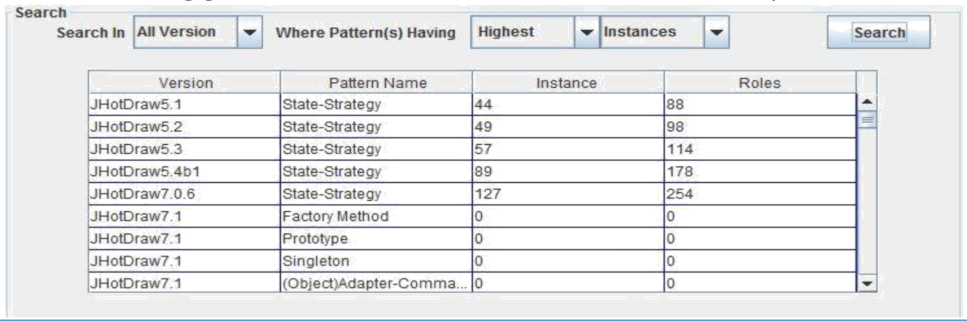
Pattern	Instances	Roles	Instances	Roles	Instances	Roles	Instances	Roles	Instances	Roles	Instances	Roles
Factory Method	2	6	2	6	7	18	9	23	3	11	0	0
Prototype	3	9	5	14	5	14	12	30	16	44	0	0
Singleton	2	4	2	4	4	13	8	28	9	35	0	0
(Object)Adapter-Command	23	86	24	87	20	98	31	154	72	297	0	0
Composite	1	6	1	6	1	6	2	8	2	11	0	0
Decorator	3	31	4	40	9	90	13	124	5	36	0	0
Observer	3	13	4	17	7	36	7	37	2	17	0	0
State-Strategy	44	88	49	98	57	114	89	178	127	254	0	0
Template Method	5	18	5	18	7	23	7	26	12	35	0	0
Visitor	0	0	0	0	0	0	1	3	0	0	0	0
Proxy	0	0	0	0	0	0	0	0	0	0	0	0
Proxy2	0	0	0	0	0	0	0	0	0	0	0	0

Instructions	Press Trend view from menu. Select pattern(e.g. Observer) Select view(e.g. Line and Dot) System will show result.
Expected Result	Trend against different patterns should be shown.
Actual Result	Trend against different patterns is show successfully. 

Verdict	PASS

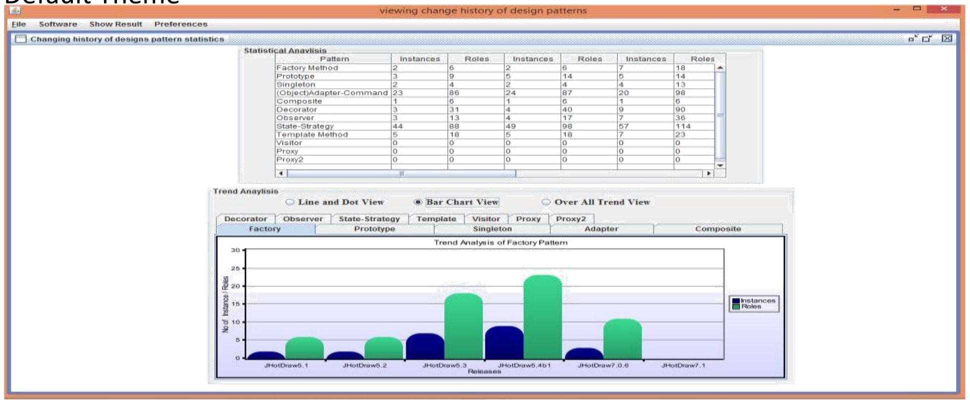
Test ID	**T003**
Use case	Search data
Description	User selects different option for making query and get their required result
Pre-Requisite	Design of the software must be extracted first.
Setup	

Pattern	Instances	Roles	Instances	Roles	Instances	Roles	Instances	Roles	Instances	Roles	Instances	Roles
Factory Method	2	6	2	6	7	18	9	23	3	11	0	0
Prototype	3	9	5	14	5	14	12	30	16	44	0	0
Singleton	2	4	2	4	4	13	8	28	9	35	0	0
(Object)Adapter-Command	23	86	24	87	20	98	31	154	72	297	0	0
Composite	1	6	1	6	1	6	2	8	2	11	0	0
Decorator	3	31	4	40	9	90	13	124	5	36	0	0
Observer	3	13	4	17	7	36	7	37	2	17	0	0
State-Strategy	44	88	49	98	57	114	89	178	127	254	0	0
Template Method	5	18	5	18	7	23	7	26	12	35	0	0
Visitor	0	0	0	0	0	0	1	3	0	0	0	0
Proxy	0	0	0	0	0	0	0	0	0	0	0	0
Proxy2	0	0	0	0	0	0	0	0	0	0	0	0

Instructions	Press search from menu. Select search in option (e.g. All Versions). Select option of search (e.g. highest). Select Instances from drop down menu Press Search.
Expected Result	Statistics having greater number of instances should be show.
Actual Result	Statistics having greater number of instances is show successfully. 

Version	Pattern Name	Instance	Roles
JHotDraw5.1	State-Strategy	44	88
JHotDraw5.2	State-Strategy	49	98
JHotDraw5.3	State-Strategy	57	114
JHotDraw5.4b1	State-Strategy	89	178
JHotDraw7.0.6	State-Strategy	127	254
JHotDraw7.1	Factory Method	0	0
JHotDraw7.1	Prototype	0	0
JHotDraw7.1	Singleton	0	0
JHotDraw7.1	(Object)Adapter-Comma...	0	0

Verdict	PASS

Test ID	T004
Use case	Apply theme
Description	To verify that theme apply on software
Set up	Default Theme 

viewing change history of design patterns

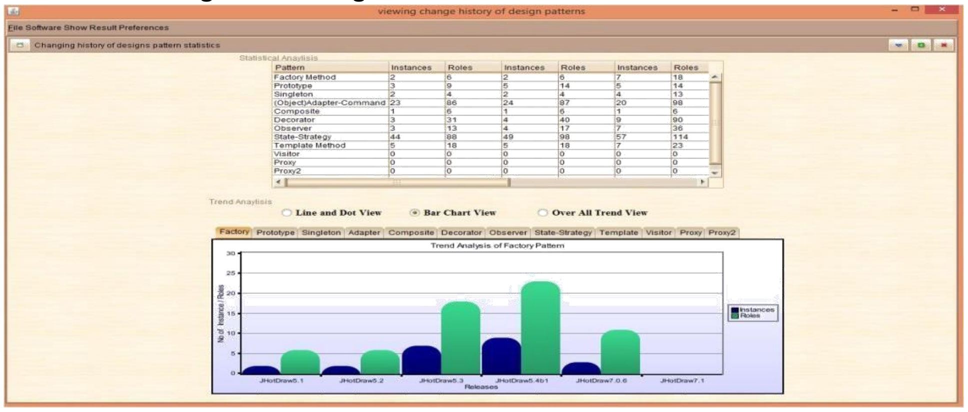
Changing history of designs pattern statistics

Statistical Anaylisis

Pattern	Instances	Roles	Instances	Roles	Instances	Roles
Factory Method	2	6	2	6	7	18
Prototype	3	9	5	14	5	14
Singleton	2	4	2	4	4	13
(Object)Adapter-Command	23	86	24	87	20	98
Composite	1	6	1	6	1	6
Decorator	3	31	4	40	9	90
Observer	3	13	4	17	7	36
State-Strategy	44	88	49	98	57	114
Template Method	5	18	5	18	7	23
Visitor	0	0	0	0	0	0
Proxy	0	0	0	0	0	0
Proxy2	0	0	0	0	0	0

Trend Anaylisis

Line and Dot View [✓ Bar Chart View] Over All Trend View

Instructions	Press set Theme Select theme from list
Expected Result	Interface should be changed according to selected theme.
Actual Result	Interface is changed according to selected theme. 

viewing change history of design patterns

Changing history of designs pattern statistics

Statistical Anaylisis

Pattern	Instances	Roles	Instances	Roles	Instances	Roles
Factory Method	2	6	2	6	7	18
Prototype	3	9	5	14	5	14
Singleton	2	4	2	4	4	13
(Object)Adapter-Command	23	86	24	87	20	98
Composite	1	6	1	6	1	6
Decorator	3	31	4	40	9	90
Observer	3	13	4	17	7	36
State-Strategy	44	88	49	98	57	114
Template Method	5	18	5	18	7	23
Visitor	0	0	0	0	0	0
Proxy	0	0	0	0	0	0
Proxy2	0	0	0	0	0	0

Trend Anaylisis

Line and Dot View [✓ Bar Chart View] Over All Trend View

Verdict	PASS

5.2. Conclusion and Future Enhancements

This chapter also summarizes the system regarding what functionality system is currently providing and what should be future enhancements to make it more effective.

5.2.1 Conclusion

Design patterns are solutions to well-known design problems. They enable us to implement tried and tested solutions to problems, thus saving time and effort during the implementation stage of the software development lifecycle. By using well understood and documented solutions, the final product will have a much higher degree of comprehension. If the solution is easier to comprehend, then, it will also be easier to maintain.

As software evolves, solutions to new design problems are added or removed constantly. The basic purpose of our project is to provide a platform for analysis of changing history of design patterns that, which pattern is added more frequently and which are removed over time this helps to predict the growth of patterns in open source projects. System clearly shows the version to version growth of pattern in design of the system and also show analysis in different ways.

Software interface is kept simple and easily understandable for users.

5.2.2 Future Enhancements:

- Other GOF patterns can be detected.
- More analysis views can be added.

References

1. Erich Gamma (1995), *Design Patterns Elements of Reusable Object-Oriented Software.*
2. Alexander, C. (1979). The Timeless Way of Building, volume 1. Oxford University Press.
3. Alexander, C., Ishikawa, S., and Silverstein, M. (1977). A Pattern Language: Towns, Buildings, Construction, volume 2. Oxford University Press.
4. http://benefitof.net/benefits-of-design-patterns
5. http://en.wikipedia.org
6. Craig Larman, (2002) *Applying UML and Patterns*, Prentice Hall Professional.
7. http://java.uom.gr/~nikos/pattern-detection.html

Appendix

Statistical Analysis

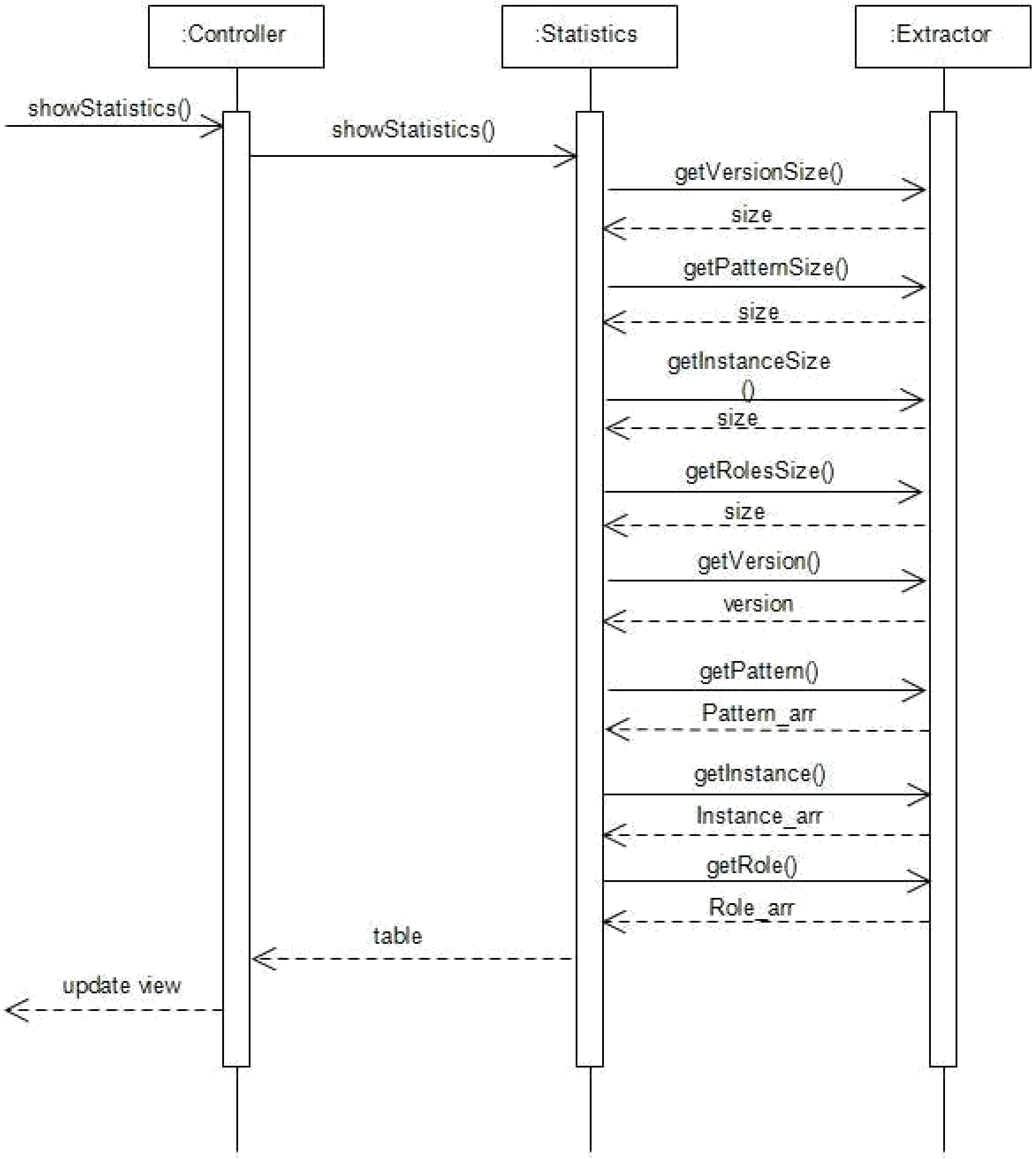


Fig 3.2.3

View list of software

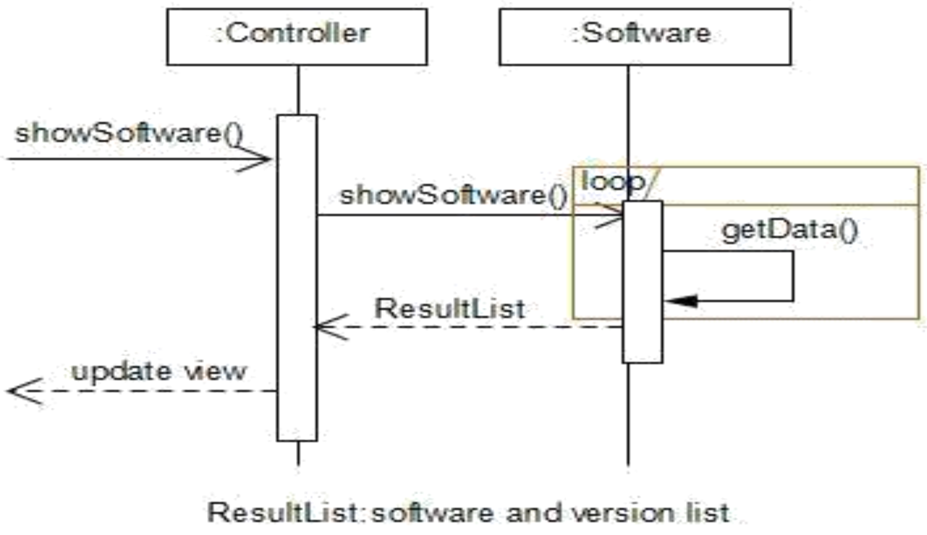


Fig 3.2.5

Set Preference

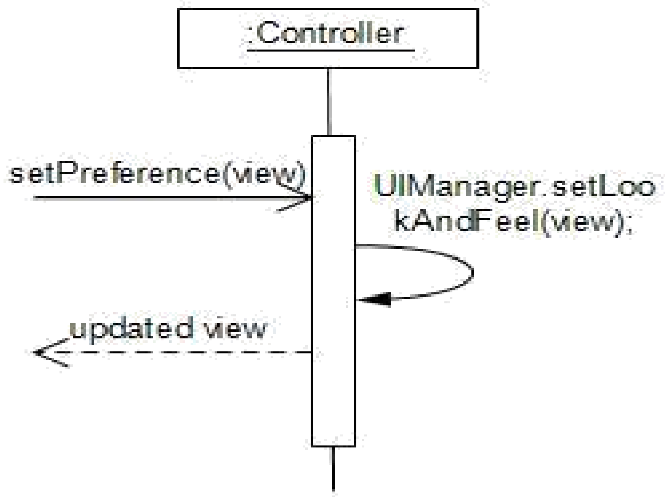


Fig 3.2.6