Abstract—Software developers often use design patterns for effectively developing their applications. Design pattern documents are available in various formats, like printed text books and electronic formats. Software engineers entrust these documents for choosing a design pattern and subsequently use the pattern to their design problem. Selecting an appropriate design pattern is extremely valuable for building error free software and this requires lot of effort (time and human hr). However, experienced people can identify suitable patterns with minimal effort. The current work in this paper will assist the developers (both novice and experienced) in choosing the right design pattern for their problem situation. Every day many people write different problem scenarios in the Internet bogs and seek solutions. This motivated us to develop current research project. The objective of research is to develop a methodology to find a suitable pattern to the user. The present paper describes Methodology, Data model, Results, and Tools & Techniques.

Keywords—IR (Information Retrieval Techniques), XML, semi structured database, parsing, UML, Artificial Intelligence, Fuzzy logic, and Recommender Systems.

I. INTRODUCTION

A pattern is a common solution to a common problem in a given context [12]. Software developers are frequently using GoF design patterns for the software development. Basically, the concept of design pattern evolved from software design principles (e.g. Coupling, Cohesion etc.) [3]. Hence, many Universities in India introduced software design patterns as a subject in the engineering courses. The introductory knowledge about patterns had been made available to the public by two sets of authors, namely, GoF (Popularly called Gang-of-Four: E.Gamma, R.Helm, Ralpha Johnson and J.Vlissides) and Christopher Alexander. The basic differences between the concepts of these two authors are, The Gang-of-Four had described 23 design patterns and modelled these patterns in UML (Unified Modelling Language) whereas the author Christopher Alexander had described patterns at several levels of abstraction, from architecture to design patterns to low-level idioms. In the fall of 1994, the Gang-of-Four developed patterns such as Creational, Structural and Behavioural whereas the authors Christopher Alexander and his team had grouped patterns based on appropriate criteria’s such as interaction and adoptable systems, organization of work, communication and access control. This approach has been named as “Pattern-Oriented Software Architecture”. Also Pattern Almnac published in the year 2000 has provided over 700 patterns organized into 70 categories. Before moving on to the further description, it is essential to review the definitions of Architectural pattern, Design Pattern and Idiom.

A. Architectural pattern

An architectural pattern describes a fundamental structural schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, includes rules and guidelines for organizing the relationships between them [5].

B. Design pattern

A design pattern describes commonly-recurring structure of communicating components that solve general design problem in a particular context [5].

C. Idiom

An idiom is a low-level pattern specific to a programming language. An idiom describes how to implement particular aspects of components or the relationships between them using the features of the given language [5]. From both the author’s groups, pattern is defined as “problem to the solution in a given context”. The context is nothing but a problem scenario. If one can evaluate scenarios properly, then finding the pattern to their problem becomes easier. So, careful attention is required for scenarios parsing. Hence, our goal of the research is to evaluate the scenarios effectively using information retrieval techniques and subsequently recommends pattern using recommender system techniques. MacManus [10] had given four approaches to recommendations. They are, personalized recommendation – recommend things based on the individual’s past behaviour, Social recommendation – recommend things based on the past behaviour of similar users, Item recommendation – recommend things based on the item itself and a combination of the three approaches above."

At present, we are using social recommendations only. The
remaining paper has been organized as follows: Section II describes common template of patterns, section III describes proposed methodology, section IV describes data model, Section V describes brief details of algorithms, and section VI describes results and discussion. Finally, section VII concludes with summary of research and future directions.

II. COMMON TEMPLATE

Please Patterns are documented in a template format. The template format make the readers understand patterns easily. Also, the template captures the details about the design pattern and related information. The template describes only pattern information and implementation of pattern in general. But it does not give any specific solution. The implementation is dependent on the features of programming language used. However, ultimately the problem in context should be solved. The general and most frequently used template (According to Gang of Four authors) is given the table 1.

<table>
<thead>
<tr>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern Name</td>
<td>Describes the essence of the pattern</td>
</tr>
<tr>
<td>Intent</td>
<td>Describes what the pattern does</td>
</tr>
<tr>
<td>Also known as</td>
<td>List any synonyms for the pattern</td>
</tr>
<tr>
<td>Motivation</td>
<td>Motivation Provides an example of a problem and how the pattern solves that problem</td>
</tr>
<tr>
<td>Applicability</td>
<td>Lists the situations where the pattern is applicable</td>
</tr>
<tr>
<td>Structure</td>
<td>Set of diagrams of the classes and objects that depict the pattern</td>
</tr>
<tr>
<td>Participants</td>
<td>Describes the classes and objects that participate in the design pattern and their responsibilities</td>
</tr>
<tr>
<td>Collaborations</td>
<td>Describes how the participants collaborate to carry out their responsibilities</td>
</tr>
<tr>
<td>Consequences</td>
<td>Describes the forces that exist with the pattern and the benefits, trade-offs, and the variable that is isolated by the pattern</td>
</tr>
</tbody>
</table>

The team Gang-of-Four has divided patterns into three groups based on their problems attained during software design [8]. In our current work we have considered these three groups of patterns. The brief details of these patterns are as follows:

A. Creational patterns

Creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation. The basic form of object creation could result in design problems and increases complexity to the design. Creational design patterns solve this problem by somehow controlling this object creation. The names of creational pattern are as follows:
- Abstract Factory
- Builder
- Factory Method
- Prototype
- Singleton

B. Structural patterns

Structural Design Patterns are Design Patterns that ease the design by identifying a simple way to realize relationships between entities. The names of creational pattern are as follows:
- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

C. Behavioral patterns

Behavioral design patterns are design patterns that identify common communication patterns between objects and realize these patterns. These patterns increase flexibility in the communication between objects. The list of behavioral patterns are given below,
- Chain of Responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- State
- Strategy
- Template Method
- Visitor
- Observer

III. METHODOLOGY

The methodology that has been proposed in this paper is described as follows:

A. Methodology

**Step1:** Parse query and find execution scenario

Pattern search is the vital operation in our work. The search begins with a query. A design pattern seeker submits a query text (problem scenario) comprising collection of words. In our current work, there are two possible search scenarios for finding the suitable pattern. Soon after submitting the query, first, the system executes scenario2, if pattern is not found, then scenario1 would be executed. Scenario1 finds the pattern in question and answers session. Scenario1 takes more time than scenario2. At present, this recommendation system records the level of each user along with query. Currently
these levels are categorized as follows: Novice, Beginner, Advanced beginner, Expert, and Wizard. These categories are useful to know the acceptance level of various users and thereby performance. The highlighted steps of both scenarios are given below –

**Scenario 1**

- **Step 1:** Find intent of the query
- **Step 2:** Retrieve knowledgeable questions related to the intent from the pattern repository and read user choice to the questions.
- **Step 3:** Find the score points while answering questions. Based on the score points, the system gives its recommendations. The recommendation can be either “strongly recommended” or “satisfactorily recommended” or “not recommend”.

**Scenario 2**

- **Step 1:** Search for similar query from the history (previous pattern database) database. If similar query is found and satisfies minimum thresholds of Support and Confidence then suggest that pattern, else continue with scenario 1 from step 1. The details of scenario 1 are given below

  - **Step 1.1:** Parse and analyze the text to find the matching pattern intents to the given query.
    
    Once the query text is submitted, the search tool parses the given query text and performs analysis. The analysis involves the most common tasks like, removing stop words (stop words are common words like a, an, the, and, etc.) lowercasing the text and extracting tokens. Techniques from open source search Lecone API[6] were used to do this analysis. From Lucene API, The tool, Standard analyzer is used for analyzing the given text. It can identify email IDs, address and any other special text (non-standard text) effectively and removes them from the input text. Generally, very few users can specify their requirements (query) clearly. Reasons can be many. Hence, queries specified by the end users fail to describe completely what those users want [6]. So directly parsing the query and retrieving the desired pattern suiting to the requirements is a daunting task. Hence, instead of retrieving the pattern directly from the repository, the suitable query-pattern intent text is retrieved. Thereafter, question and answer session would begin. The success of pattern intent retrieval depends on two important jobs. i.e. Index and Search. The details of index and search are described in the step 1.2 and 1.3

- **Step 1.2 Index Creation**
  
  About key words
  
  - The first step we perform before starting the application is validating the xml files. As of now, we have two sets of xml files:
  
  - Actual pattern xml files: These are the files contained in "/conf/patterns" directory. We validate these xml files against the given dp.xsd schema definition file. And if any xml is against/violating the rules defined in xsd, we report error and stop process of starting the server.
  
  - Configuration.xml: This is actual application level configuration file, we also validate this at the time of startup of application against "configuration.xsd". This file contains the application level configuration options like, items to be indexed, the final result html page to be rendered and the Structured Query Image ile path per pattern etc.,
  
  - We get the elements to be indexed from the application level configuration file and create index on these selected elements. (AS of now we are indexing based on problem statement, forces, applicability’s and intent tags). The index building is done using “XmlIndexer.buildIndex()”, we use the StandardAnalizer given by Lucene to build the Index, the generated index files are stored in the preconfigured location configured as shown below:

  
  `<conf:indexDirectoryPath>./index</conf:indexDirectoryPath>`.selected.

- **Step 1.3 Search operation**

  Search operations have been implemented with the open source Jakarta’s search project, Lucene API. Lucene has powerful search capability that uses Artificial Intelligence Fuzzy logic to locate indexed items [4]. It is a popular search and index component [4].

  It is a basic framework, upon which we build full-featured search. Lucene scoring uses a combination of the Vector Space Model (VSM) of Information Retrieval and the Boolean model to determine how relevant a given Document is to a User's query [1]. In general, the idea behind the VSM is the more times a query term appears in a document relative to the number of times the term appears in all the documents in the collection, the more relevant that document is to the query. It uses the Boolean model to first narrow down the documents that need to be scored based on the use of Boolean logic in the Query specification. Lucene also adds some capabilities and refinements onto this model to support Boolean and fuzzy searching, but it essentially remains a VSM based system at the heart. If a user enters a generic search term, the system parses the search term with Lucenes Standard analyzer by passing Custom stop words file (stop word file contains frequently appeared words like a, an, the, then etc.) to remove unwanted strings. We used Lucene MultifieldQueryParser to parse and search the Query string within multiple fields of the Pattern xml repository, upon which the index is built (for instance in this case it is based on problem statement, forces, applicability and intent).

  Lucene supports a wide array of possible searches including AND, OR and NOT, fuzzy searches, proximity searches, wildcard searches, and range searches. The retrieval of matching intent documents to the given query text is done using Lucene’s built in formula [4]. The formula includes, for each document (d) matching each term (t) in a query (q) is given in equation (1),
\[ \sum \text{df(t)} \times \text{idf(t)}^2 \times \text{boost(t, fieldid)} \times \text{lengthNorm(t, fieldid)} \times \text{coord(q, d)} \times \text{querynorm(q)} \quad (1) \]

**Step 2:** Retrieve knowledge based questions related to the intent.

In the step 1, the retrieval system retrieves possible matching pattern intents to the given query. Once intents are displayed, user has to choose one matching intent out of available intents. In this step 2, the retrieval system finds pattern intent matching questions and records user answers.

All the questions should be answered. While answering, the system calculates \text{pattern_weighted_score} (numeric). In our current paper, the \text{pattern_weighted_score} is one of the measures for pattern selection. Generally, higher the score stronger the pattern selection.

**Step 3:** Find the score points while answering questions and gives comments: strongly recommended, or satisfactorily recommended or not recommended. The score details are as follows: The maximum score for each pattern is 100. For selecting an ‘intent’, a score of 25 will be added to the \text{score_counter} variable [\text{score_counter} is a variable initially set to 0 (zero)]. Next, pattern knowledgeable questions would be asked. For example, if there are 5 pattern knowledgeable questions to be answered, then for answering each question, a score 12 will be added to the \text{score_counter} (60/5=12). Otherwise, a score 12 will be deducted from the \text{score_counter}. Finally, there would be question related to the pattern structure. At present one pattern structure based question is asked. If a user answers for this question, a constant score 15 will be added to the \text{score_counter} otherwise 15 will be deducted from the \text{score_counter}. All these scores have been decided by using our expertise.

**Scenario 2**

**Step 1:** Search for similar query from the history (previous pattern database) database. If similar query is found and satisfies minimum thresholds of Support and Confidence then suggest the same pattern, else continue with scenario 1 from step 1. In this scenario 2, the system finds the pattern based on previous users queries by finding associations between queries and patterns. For example, for a query Q0, the recommended pattern may be P0 [Q0->P0] and there are chances that the same query or near similar query may have suggested pattern, P1 [Q0->P1]. All associations belong to two-item sets only.

### IV. Data Model

Pattern repository has been built on XML (extensible markup language) technology. First, we have designed a pattern schema. The schema describes Meta data, elements (simple & complex), constraints, relations between elements and references. The reasons for choosing XML are, XML format gives better indexing and searching capabilities, platform independence, Interoperability and Internationality (Unicode) and W3C standard [9]. The details of the schema are given in Table 2 and subsequently its meta is described:

<table>
<thead>
<tr>
<th>Table II</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Patterns</strong></td>
</tr>
<tr>
<td>[e] Functional domain [0..*] functionaldomain</td>
</tr>
<tr>
<td>[e] Force [0…*] force</td>
</tr>
<tr>
<td>[e] Consequence [0…*] consequence</td>
</tr>
<tr>
<td>[e] pattern [1…*] pattern</td>
</tr>
<tr>
<td>[e] variation [0..*] variation</td>
</tr>
<tr>
<td>[e] relatedpattern [1..*] relatedpattern</td>
</tr>
</tbody>
</table>

Every pattern balances certain forces, consequences etc. These things are modeled as complex elements. The list of complex elements in our pattern schema are described as follows:

#### A. Functional domain

Patterns can be applied in different functional domains. For example, Banking, Insurance etc. The cardinalities are zero/many. Each functional domain contains two elements, name and summary. It is a complex type.

#### B. Author

Author xml document creator. It is a token data.

#### C. Applicability

Patterns are applied in various domains in various situations. The possible set of scenarios is given in this element. The applicability ensures whether pattern is right for our problem or not. It is of type content.

#### D. Variation

Every pattern has different variations. For example proxy pattern has many variations, like virtual proxy, remote proxy etc. It is of type variation and contains two elements, name and summary.

#### E. Related pattern

One Pattern has relationship with other patterns. For example, abstract factory is often implemented with factory methods and many patterns Could be implemented with singleton. It is of type relationship. It contains two elements, related type and relationship type.

#### F. Generic Question

These questions are pattern related questions.

#### G. Structured question

The structured question is related the Structure of the pattern

### V. Algorithms

In our current work, the following algorithms have been implemented so far.
A. Algorithm 1: Query to Document Matching

**Purpose:** The purpose of this algorithm is to parse the given query text and find the relevant document matching to the given query. The algorithm returns list of pattern intents matching to the given query.

B. Algorithm 2: Weighted Pattern Selection

**Purpose:** The purpose of this algorithm is to find suitable questions to the user query, and captures user answers. While capturing answers, the system calculates numeric score. Based on the score obtained, the system gives comments either strongly recommended or satisfactorily recommended, or not recommended.

C. Algorithm 3: Finding query similarity and pattern selection based on association measures [Support, Confidence].

**Purpose:** This algorithm focuses on finding the pattern based on previous user’s queries and recommended patterns.

VI. RESULTS AND DISCUSSION

Users play critical role of this kind of system. Ultimately, user’s acceptance is important. First, we have approached our students to use our system. Then working professionals (advanced beginner, expert and wizard) from a software development company (Agiliance Technologies, Hyderabad, India). All the students are pursuing Master’s Degree in Engineering at our institute (ISQUAREIT-IGNOU Center of excellence FOR ADVANCECD EDUCATION AND RESEARCH, PUNE, INDIA). The system stores the pattern selection details in a logfile (applog.rtf). This applog file contains variables: Query, weighted_pattern_score, Lucene_score, category of user, user acceptance (yes/no) and total time (seconds). These details are useful for finding the following measures,

- Average time taken for each category of users.
- Statistical Correlation (Pearson correlation coefficient) between categories of user with time.
- Minimal statistics (Minimum, Maximum, Standard deviation, number of observations) for all the numeric variable, Lucene_score, and weighted_patter_score (pw_score).
- Statistical Correlation (Pearson correlation coefficient) between categories of user with Lucene_score etc.
- Variation in user categories and their responses can also be measured.

These measures are useful for evaluating the performance of the recommendation system. The frequency table 4 shows, so far there are total 86 suggestions were given by the system. Out of which 18 (20.93%) were rejected and 68 (79.07%) were accepted by the different categories of users.

Where in the table 4, 0 indicates rejected user, and 1 indicates accepted user. The following table 5 shows minimal statistics for each category of user.

First, there are 19 advanced beginners who have submitted their queries seeking for pattern suggestion. Their mean (average) time is 38.842150 seconds. Also it is observed, the average time is higher for expert and wizard category of users when compared with other category of users. Because, they read each and every question carefully and then gave answers to the questions. Whereas, novice users gave answers without reading the questions properly. Hence, their average time is less when compared to other users.

<table>
<thead>
<tr>
<th>TABLE III</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>User type</td>
</tr>
<tr>
<td>advanced beginner</td>
</tr>
<tr>
<td>beginner</td>
</tr>
<tr>
<td>Expert</td>
</tr>
<tr>
<td>Novice</td>
</tr>
<tr>
<td>Wizard</td>
</tr>
</tbody>
</table>

VII. CONCLUSION

Basically, we followed prototype methodology to develop this research work. At present, the system supports search for GoF patterns only. The search can be extended to any category of software design patterns (e.g. Anti patterns, GRASP etc). Because the underlying schema is same. Also, we are in the process of designing an algorithm to recommend an alternative pattern based on pattern similarity characteristics. Also, we would like to do various data analyses to measure the performance of our recommendation system.

ACKNOWLEDGMENT

I, sincerely thank my co-author, Prof. Dr.M.M.Naidu for his valuable suggestions to carry out my work. Also, I thank Prof.Dr.Rahul Athale for his inputs during algorithm design.

REFERENCES


