4. A GENETIC ALGORITHM-BASED ASSOCIATION RULE MINING APPROACH FOR IDENTIFYING IMPORTANT HUMAN FACTORS THAT IMPACT SOFTWARE QUALITY

Over the years, the production of quality software has become a big challenge to software organizations, developers and managers. Software engineers and managers grapple with a lot of challenges in their endeavor to engineer quality software.

4.1 SOFTWARE QUALITY

The American Heritage Dictionary defines quality as “an attribute or characteristic of something”. Often these attributes are measurable – like cost, color and length. But given the nature of software, it is difficult to characterize software through measurable properties like cyclomatic complexity and lines of code do exist. According to Pressman[70], there are two kinds of quality:

- Quality of Design – characteristics that designers specify for an item.
- Quality of Conformance – the degree to which design specifications are followed during manufacturing.

But software quality is not that easy. According to Glass[28], User Satisfaction=compliant product + good quality + delivery within budget and schedule.

Glass[28] view encompasses the notion that despite the importance of quality, if the user is not satisfied, nothing else matters. In DeMarco’s[19]
view, a product’s quality is a function of how much it changes the world for the better. Pressman[70] defines quality as “conformance to explicitly stated functional and performance requirements, explicitly documented development standards and implicit characteristics that are expected of all professionally developed software”.

4.1.1 Total Quality Management

Total quality management encompasses the following four phases:

- **Kaizen** – a system of continuous process improvement. The goal is to develop a visible, repeatable and measurable process.

- **Atarimae Hinshitsu** – examines intangibles that affect the process and attempts to optimize their impact.

- **Kansei** – leads to improvement in the product itself by examining ways in which the user applies the product.

- **Miryokuteki Hinshitsu** – attempts to uncover new applications that are an outgrowth of existing ones.

4.1.2 Software Quality Assurance

Software Quality Assurance (SQA) is defined by IEEE as “the planned and systematic pattern of actions carried out to provide adequate confidence that the product meets its requirements”. The history of SQA parallels the history of quality in manufacturing. Gone are the days where quality was the sole responsibility of the craftsman who engineered the product. Quality is the responsibility of many people – software engineers, managers, customers and
individuals within the SQA group. The following are the responsibilities of the SQA group which serves as the customer’s in-house representative and sees the software from the customer point of view.

- Preparing an SQA plan for the project;
- Participating in the development of the project’s software process description;
- Reviewing activities to ensure conformance with the defined software process;
- Auditing designated software work products to verify compliance with those defined as part of the software process;
- Ensuring that deviations in work products are documented and handled according to a documented procedure;
- Recording any non-compliance and reporting the same to senior management.

4.2 SOFTWARE RELIABILITY

Software reliability is an important aspect of software quality and is defined as “the probability of failure-free operation of a computer program in a specified environment for a specified time”. A simple measure of reliability of computer-based systems is the Mean Time Between Failure (MTBF) where \( MTBF = MTTF + MTTR \), where MTTF is the Mean Time to Failure and MTTR is the Mean Time to Repair. Pressman[70] clearly differentiates between error count and reliability as in the context of reliability only those
errors that contribute to system failure are of consideration. Availability is the probability that a program is operating according to requirements at a given point of time and is defined as Availability = \(\frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}\)\]*100%.

### 4.3 QUALITY FACTORS

Pressman[70] categorizes the factors that affect software quality into two main groups - those that can be measured directly and those that can be measured only indirectly. Examples of factors that can be directly measured include defects per function point. Examples of factors that can be measured only indirectly are usability and maintainability.

McCall et al.[61] provide a categorization of quality factors by focusing on the three dimensions - the operational characteristics of the software, the ability of the software to undergo change and the adaptability of the software to new environments. The factors along the operational dimension include:

- **Correctness** - the extent to which the program satisfies the specification and fulfills customer’s mission objectives;
- **Reliability** - the extent to which the program can be expected to perform its intended function with required precision;
- **Efficiency** - amount of resources and code required by a program to perform its functions;
- **Integrity** - extent to which access to software data by unauthorized persons can be controlled;
- **Usability** - effort required to learn, operate and use the software.
The factors along the revision dimension include:

- Maintainability - effort required to locate and fix an error in a program;
- Flexibility - effort required to modify an operational program;
- Testability - effort required to test a program to ensure that it performs its intended functions.

The factors along the transition dimension include:

- Portability - effort required to transfer the program from one hardware/software platform to another;
- Reusability - extent to which the program or parts of it can be reused in other applications;
- Interoperability - effort required to couple one system with another.

The quality factors developed by Hewlett-Packard, given the acronym FURPS encompasses (GRA87):

- Functionality - assessed by evaluating the feature set and capabilities of the program;
- Usability - assessed by considering human factors, overall aesthetics and documentation;
- Reliability - assessed by measuring the frequency and severity of failures;
- Performance - measured by processing speed, response time, throughput and efficiency;
• Supportability - combines extensibility which is the ability to extend the program, adaptability and serviceability.

The quality factors developed by ISO 9126 include:
• Functionality - degree to which the software satisfies stated needs;
• Reliability - amount of time that the software is available for use;
• Usability - degree to which the software is easy to use;
• Efficiency - degree to which software makes optimal use of system resources;
• Maintainability - ease to which repair can be made to the software;
• Portability - ease with which software can be transported from one environment to another.

All the above literature point to one fact - quality of software has been an issue daunting software engineers and researchers for years. Ensuring the quality of software delivered has been and will continue to be an issue to be addressed by the software engineering research community.

4.4 HUMAN FACTORS IN SOFTWARE QUALITY

While addressing the issue of software quality, the role played by the human factors seems to be the most ignored of all. Human factors engineering pertains to the application of scientific knowledge concerning human behavior to the design of jobs and products. The dearth of available literature pertaining to the role of human factors in software quality stands testimony to this fact. While a lot of work identifies the human factors involved in software quality
and productivity, little has been done in the application of scientific methodologies to draw meaningful and valid conclusions pertaining to human factors. McConnel[62] goes to the extent of stating that people issues have the biggest impact on software quality.

White and Lab[91] addresses the people factors that have an impact on productivity of software. The author suggests that it is important to have the right people, people who are knowledgeable, skilled and satisfied, project manager who works well with people to ensure quality. The author also opines that staffing, motivation and the work environment are the most important people issues. The author also draws attention to the often overlooked issues in recruitment – selecting the candidates who fit with the work environment, who have the potential to contribute to the team framework. Care also needs to be given to the team composition given the nature of software development – it being a team work. The project managers need to be cognizant of the factors that would motivate the different team members. The top motivators for programmers are identified as opportunities for professional growth, achievement and challenging work. The author also showcases the importance of the work environment with regard to quality and productivity.

Fairley[24] states seventeen different quality and productivity factors. Out of the 17 factors, the following are those pertaining to people:

- Individual Ability – encompasses both the general competency of the individual and familiarity with the particular application area;
• Team Communication – the ability of programmers to communicate well have a direct bearing on the quality;

• Required Skills – software engineering requires a variety of skills;

• Adequacy of Training – the degree to which the training requirements of personnel are met;

• Management skills – the differences in the nature of software projects compared with the others dictate the need for new skills for managing software projects.

John et al.[46] opine that since software is developed for people and by people, human and social factors have a very strong impact on the success of software development endeavors and the resulting system. They reinforce the opinion expressed above that surprisingly, much of the software engineering research in the last decade has been technical ignoring the people aspect. Grisham and Perry[31] attempt to derive the conditions under which extreme programming – an agile method – can lead to user satisfaction. They analyze whether on-site customer can always lead to better relationships.

Army and Raylene[8] explain how agile practices can be applied to address social factors like knowledge sharing, motivation and customer collaboration. They consider the impact of different working environmental factors, minimal documentation, and strategies for introduction of new team members. Umarji and Seaman[86] focus on challenging personal factors like
fear of adverse consequences and degree of control over personnel working processes.

John et al.[46] also opine that there is a belief that agile practices and guidance-centered management practices can improve quality and productivity. They also perceive that with development teams becoming more globalized and culturally dissimilar, new organizational challenges will be encountered.

4.5 ASSOCIATION RULE MINING

Data mining is generally considered as the process of discovering hidden, non-trivial and previously unknown patterns from a large collection of data. Association rule mining is an important component of data mining. According to the Indian Agricultural Statistical Research Institute (IASRI), association rule mining is perhaps the most widely studied model by the data mining community. Potential examples of the utility of the mined association rules abound in almost every discipline. Mata[60] Organizations, particularly, in the retail segment are interested to use association rule mining to discover customer buying patterns that can aid them in major decision-making regarding which products can be promoted together. Medical domain is no exception. Association rules can help doctors to find the factors most likely to cause a particular disease. This is especially significant for non-communicable diseases for which the importance of various contributing factors is relatively unknown. Association rule mining can also be applied to agricultural databases to survey data from agricultural research.
The basic objective of association rule mining is to find all co-occurrence relationships called associations. The classic application of association rule mining is the market basket analysis which aims to discover how items purchased by customers are associated. An association rule is of the form \( X \rightarrow Y \) where \( X \) and \( Y \) are collections of attributes whose intersection is null. For example, every customer who purchased a computer (\( X \)) also purchased a printer(\( Y \)). \( X \) is called the antecedent and \( Y \) is called the consequent. Since the number of possible association rules can be huge, often the interest is on those rules which satisfy some constraints. The most common of these constraints include support and confidence.

Formally, let \( I = \{i_1, i_2, \ldots, i_m\} \) be a set of items. Let \( T = \{t_1, t_2, \ldots, t_n\} \) be a set of transactions where each transaction \( t_i \) is a set of items. An association rule is of the form \( X \rightarrow Y \), where \( X \subset I, Y \subset I \) and \( X \cap Y = \phi \). \( X \) (and \( Y \)) is a set of items called an itemset. The support of a rule \( X \rightarrow Y \) is the percentage of transactions in \( T \) that contains \( X \cup Y \) and can be stated as \( \text{Pr}(X \cup Y) \) which is the estimate of probability. If \( n \) is the number of transactions then the support of the rule \( X \cup Y \) is given as \( (X \cup Y).\text{count}/n \). Confidence of a rule \( X \rightarrow Y \) is the percentage of transactions in \( T \) that contain \( X \) also contain \( Y \). It can be stated as \( \text{Pr}(Y|X) \) – the conditional probability. Confidence = \( (X \cup Y).\text{count}/X.\text{count} \)

Yan et al.[96] designed a genetic algorithm-based strategy for identifying association rules without specifying actual minimum support. Hadian et al.[32] presented a new algorithm, Cluster-Based Multi-Objective
Genetic Algorithm (CBMOGA) which optimizes the support counting phase by clustering the database. Nath et al.[64] described a comprehensive survey on the state-of-the-art algorithms for association rule mining, especially when the data sets used for rule mining are not static. Liu and Li[54] given a hybrid association rule mining method for characterizing network traffic behavior. Al-Maolegi and Arkok[2] indicated the limitation of the original Apriori algorithm of wasting time for scanning the whole database searching on the frequent itemsets, and presented an improvement on Apriori by reducing that wasted time depending on scanning only some transactions.

4.6 ASSOCIATION RULE MINING ALGORITHMS

Over the years, many algorithms have been presented for mining association rules, and the algorithms vary in terms of their computation efficiency and applicability. The most frequently studied is the Apriori algorithm that is explained below.

The apriori algorithm works in two steps:

- Generate all frequent itemsets – a frequent itemset is an itemset that has transaction support above the minimum support;
- Generate all confident association rules from frequent itemsets – a confident association rule is a rule with confidence above minimum confidence.

The most interesting aspect of the apriori algorithm is that it relies on the downward closure property which states that if an itemset has a minimum
support, then every non-empty subset of this itemset also has minimum support. For a formal specification of the Apriori algorithm[2], the following notations are adopted.

- $L_k$ – set of large $k$-item-sets (those with minimum support). Each member of this set has two fields: a) itemset and b) support count
- $C_k$ – set of candidate $k$-item-sets. Each member of this set has two fields: a) itemset b) support count

With these notations, the algorithm is sketched below:

\[
L_1=\{\text{large 1-itemsets}\};
\]

for (k=2; $L_{k-1} \neq \phi; k ++$) do begin

\[
C_k = \text{apriori-gen}(L_{k-1});
\]

forall transactions $t \in D$ do begin

\[
C_i = \text{subset}(C_k, t);
\]

forall candidates $c \in C_i$ do

\[
c. \text{count}++;\]

end

\[
L_k = \{c \in C_k \mid c.\text{count} \geq \text{min sup}\}
\]

end

Answer=$\bigcup_k L_k$ ;
4.6.1 Candidate Generation

The procedure for generating candidate itemsets apriori-gen is shown below. It takes $L_{k-1}$, the set of large (k-1) itemsets.

```
insert into $C_k$ select $p.item_1, p.item_2, ..., p.item_{k-1}, q.item_{k-1}$ from $L_{k-1}p, L_{k-1}q$
```

where $p.item_1 = q.item_1, ..., p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1}$;

In the prune step, all itemsets $c \in C_k$ such that some (k-1) subset of $c$ is not in $L_{k-1}$ are deleted.

```
forall itemsets $c \in C_k$ do
    forall (k-1) subsets $s$ of $c$ do
        if ($s \notin L_{k-1}$) then
            delete $c$ from $C_k$;
```

Many other association rule mining algorithms have been given and studied in the literature. Most of these algorithms attempt to discover interesting association rules with less time overhead. Most notable among them are Frequent Pattern (FP) Growth Algorithm, Pincer Search Algorithm, Border Algorithm and the like.

4.7 FP GROWTH ALGORITHM

One of the main drawbacks of the apriori algorithm is the time overhead incurred in candidate generation. Han et al.[34] describe a novel Frequent Pattern (FP) tree structure and use this tree for mining in a method called as “FP Growth”.

101
The algorithm basically works in two phases:

- Frequent Pattern Tree Design and Construction
- Mining frequent patterns using FP-Tree

**Phase 1 – FP Tree Construction**

Let \( I = \{a_1, a_2, \ldots, a_m\} \) be a set of items and a transaction database \( DB = \langle T_1, T_2, \ldots, T_n \rangle \) where \( T_i (i \in [1..n]) \) is a transaction which contains a set of items in \( I \). The support of a pattern \( A \) which is a set of items is the number of transactions containing \( A \) in \( DB \). \( A \) is a frequent pattern if \( A \)’s support is no less than a pre-defined minimum support threshold.

An FP Tree is constructed based on the following observations:

- Only the frequent items will play a role in frequent pattern mining. One scan of \( DB \) is performed to identify the set of frequent items.
- The set of frequent items for each transaction are stored in a compact tree structure thereby avoiding repeated scans of \( DB \).
- In case of multiple transactions sharing identical frequent itemset, these are merged. If the frequent items in transactions are maintained in a sorted order, it is easy to check if two sets are identical.
- For transactions sharing a common prefix, the shared parts are merged into one prefix structure. If the frequent items are sorted in descending order of their frequencies, the scope of sharing the prefix strings increases.
Han et al.[34] define an FP tree as follows:

- It consists of one root labeled null, a set of item prefix sub trees as the children of the root, and a frequent item header table.

- Each node in the item prefix sub tree consists of three fields: item-name, count and node-link, where item-name is the item the node represents, count is the number of transactions represented by the portion of the path reaching this node and node-link refers to the next node in the FP Tree carrying the same item-name or null if there is none.

- Each item in the frequent-item header table contains two fields: 1) item name 2) head of node link which points to the first node in the FP Tree carrying the item-name.

The FP Tree construction algorithm is sketched below:

Input: A transaction database DB and a minimum support threshold $\zeta$

Output: FP Tree

Method:

- Scan the transaction DB once. Collect the set of frequent items $F$ and their supports. Sort $F$ in support descending order as $L$, a list of frequent items.

- Create a root of an FP Tree labeled “null”. For each transaction $\text{trans}$ in the DB do:
  - Select and sort the frequent items in $\text{trans}$ according to the order of $L$. Let the sorted frequent item list in $\text{trans}$ be $[p|P]$, where $p$
is the first element and \( P \) is the remaining list. Call \( \text{insert_tree}([\text{p}|P], T) \).

The function \( \text{insert_tree}([\text{p}|P], T) \) works as follows:

- If \( T \) has a child \( N \) such that \( N\text{.item-name}=\text{p}\text{.item-name} \) then increment \( N \)'s count by 1; else create a new node \( N \) and set its count to 1, its parent link be linked to \( T \) and its node-link be linked to the nodes with the same item-name via the node-link structure. If \( P \) is non-empty, call \( \text{insert_tree}(P,N) \) recursively.

<table>
<thead>
<tr>
<th>Item</th>
<th>Head of node-links</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4.1 The Constructed FP-Tree**
Table 4.1 Transaction Database

<table>
<thead>
<tr>
<th>Transaction ID</th>
<th>Items Bought</th>
<th>(Ordered) Frequent Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>f, a, c, d, g, i, m, p</td>
<td>f, c, a, m, p</td>
</tr>
<tr>
<td>200</td>
<td>a, b, c, f, l, m, o</td>
<td>f, c, a, b, m</td>
</tr>
<tr>
<td>300</td>
<td>b, f, h, j, o</td>
<td>f, b</td>
</tr>
<tr>
<td>400</td>
<td>b, c, k, s, p</td>
<td>c, b, p</td>
</tr>
<tr>
<td>500</td>
<td>a, f, c, e, l, p, m, n</td>
<td>f, c, a, m, p</td>
</tr>
</tbody>
</table>

Phase 2 – Mining frequent patterns using the FP Tree

The procedure for mining frequent patterns from the FP Tree as described by Han et al.[34] is sketched below for completeness:

Input: FP Tree constructed as above

Output: the complete set of frequent patterns

Method:

Procedure FP-growth(Tree, α)

{ 
    IF tree contains a single path P
    THEN FOR EACH combination of the nodes in the path P DO
        Generate pattern β∪α with support=minimum support of nodes in β;
    ELSE FOR EACH aᵢ in the header of tree DO{
        Generate pattern β=aᵢ ∪ α with support = aᵢ.support;

}
Construct β’s conditional pattern base and then β’s conditional FP Tree Treeβ:

If Treeβ!=null

THEN CALL FP Growth(Treeβ,β) }

Han et al.[34] also demonstrate the superiority of the approach in terms of performance.

4.8 EVALUATING THE QUALITY OF THE MINED ASSOCIATION RULES

Besides support and confidence, many other measures have been given to mine association rules. The following are some of the measures. Given an association rule, $A \rightarrow B$, the following notations are used:

- $n=|E|$ the total number of records
- $n_a = |A|$, the number of records satisfying A
- $n_b = |B|$, the number of records satisfying B
- $n_{ab} = |A \cap B|$, number of records satisfying both A and B
- $n_{a\overline{b}} = |A \cap \overline{B}|$, number of records satisfying A but not B

- **Bayes Factor** – $\frac{n_{ab} n_b}{n_a n_{a\overline{b}}}$

- **Centred Confidence** – $\frac{n_{ab}}{n_a} - \frac{n_b}{n}$

- **Conviction** – $\frac{n_a n_{a\overline{b}}}{nn_{a\overline{b}}}$
- Information Gain - \( \log \left( \frac{\text{n}_{\text{ab}}}{\text{n}_a \text{n}_b} \right) \)

- Laplace – \( \frac{n_{ab} + 1}{n_a + 2} \)

- Lift - \( \frac{n_{ab}}{n_a n_b} \)

4.9 GENETIC ALGORITHMS

Genetic algorithms belong to the class of evolutionary computation and attempt to mimic the natural process of evolution to uncover solutions to problems. The basic ideas behind them were presented in the previous chapter. The pseudo-code is sketched here for completeness.

1. Initialize the population by generating an initial set of chromosomes representing solutions;

2. Calculate fitness of the individuals in the population based on some criteria;

3. Select two individuals from the population at random based on the fitness so that the more fit individuals get more chance for selection;

4. Perform cross over by selecting one or more cross points;

5. Select chromosomes at random and mutate them with some probability;

6. If the terminating condition is met, stop and display the best solution obtained. Else repeat steps 2 – 5 using the newly generated solutions
Genetic algorithms have been extensively applied in various domains and are very promising in yielding solutions to complex problems involving a lot of uncertainty.

4.10 MINING ASSOCIATION RULES BETWEEN HUMAN FACTORS AND QUALITY OF SOFTWARE

The importance of human factors in assuring software quality was elaborated earlier. The research aims at deriving association rules that would aid in discovering the crucial human factors that have a significant impact on the quality of the software engineered.

4.10.1 Human Factors Considered

The list of human factors involved in software engineering and those that impact the quality of the software tends to be very huge and most of the factors may also be correlated. For the purpose of the research, the following set of human factors is considered. The list is by no means complete and can be extended to accommodate many more.

**Table 4.2 List of Human Factors Considered**

<table>
<thead>
<tr>
<th>Human Factor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Competency in Technology</td>
<td>The skill in the particular technology</td>
</tr>
<tr>
<td>Experience in the technology</td>
<td>The experience of the candidate in the technology</td>
</tr>
<tr>
<td>Experience in the domain</td>
<td>The experience in the domain of the project</td>
</tr>
<tr>
<td>Ability to work in a team</td>
<td>The ability of the individual to contribute in a team</td>
</tr>
<tr>
<td>Communication Skills</td>
<td>The ability of the individual to comprehend requirements and communicate problems faced</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Motivation Level</td>
<td>The level of motivation of the individual</td>
</tr>
<tr>
<td>Commitment</td>
<td>The commitment of the individual toward the success of the project</td>
</tr>
<tr>
<td>Responsibility</td>
<td>Learn from past mistakes</td>
</tr>
</tbody>
</table>

### 4.10.2 Methodology

A project manager working in a local software organization and who has 8 years of experience in software development and 7 years in software project management is asked to rate the team leaders of 50 teams that have delivered around 60 web sites. The team leaders are rated in the scale of 0 to 3 on all the above factors where 0 indicates that the attribute in question is least and 3 indicate that the attribute in question is maximum. For example, a rating of 0 for experience the domain factor means the individual has no experience in the domain, and a rating of 3 for motivation level means that the individual is motivated to the greatest extent. Despite the subjectivity involved in the rating of the team leaders, the experience of the manager is expected to increase the objectivity to a reasonable level and thus improve the quality and utility of the mined association rules. The quality information pertaining to the 50 web sites is available. For simplicity, every web site is classified either as quality or non-quality web site based upon the number of issues faced post delivery for the website. Out of the 60 web sites, 43 are classified as “quality” web sites and the
remaining 17 as “not-quality” sites. This categorization is done solely based on the number of issues reported so far in the web sites and by comparing this with a threshold value. A quality value of 1 indicates a quality web site and a 0 indicates a no-quality web site.

4.10.3 Applying GA to Mine Association Rules

To apply Genetic Algorithm to any problem, the first issue that needs to be addressed is encoding of the solutions. Solutions have to be represented as chromosomes. When applying Genetic Algorithm to mine association rules, there are basically two approaches for solution representation: The Pittsburgh approach where each chromosome represents a set of rules and the Michigan approach where each chromosome represents a single rule. This research uses a modified Michigan approach given by Ghosh and Nath[27]. The basic idea is to associate 2 bits with each attribute. If these 2 bits are 00, then the following attribute appears in the antecedent part and if these are 11, then the following attribute appears in the consequent part. The remaining 2 combinations – 01 and 10 imply the absence of the following attribute in the rule. Since the position of various attributes in the chromosomes is fixed and the attributes are numeric, the values of the attributes can be encoded in their binary form. The values to be encoded are in the range of 0-3 for the ratings and 0-1 for quality attribute. 8 human factors would need 2 bits each and the quality attribute needs 1 bit. For all the 9 attributes 2 tag bits are required as described above. The total space requirement for association rules of arbitrary length turns out to
be $8 \times 2 + 1 + 9 \times 2 = 35$ bits. The fitness evaluation is done using an approach analogous to the one adopted by Wakabi-Waiswa and Baryamureeba[90]. Five complementary metrics are used for fitness evaluation.

- **Support** – the support $\sigma(X)$ of an itemset $X$ is defined as the proportion of transactions in the dataset that contains the itemset.

- **Confidence** – it is defined as the conditional probability of the consequent given the antecedent. $\text{confidence} = \frac{\sigma(X \cup Y)}{\sigma(X)}$

- **Rule Antecedent Interestingness (RAI)** is calculated as

$$1 - \left[ \frac{\sum_{i=1}^{n} \text{InfoGain}(A_i)}{\log_2(|G_i|)} \right]$$

where InfoGain is the information gain

- **Rule Consequent Interestingness (RCI)** is calculated as

$$\left(1 - \Pr(G_{i \mid l})\right)^{1/\beta}$$

where $G_{i \mid l}$ is the prior probability of the goal attribute value, $\beta$ is a user defined parameter and $1/\beta$ is a measure for reducing the influence of the rule consequent interestingness in the value of the fitness function.

- **Interestingness of the rule** is defined as $\text{RAI} + \text{RCI} / 2$

- **J-Measure** is calculated as

$$J_M = f(x) \left[ f(y \mid x) \ln \left( \frac{f(y \mid x)}{f(y)} \right) + (1 - f(y \mid x)) \ln \left( \frac{1 - f(y \mid x)}{1 - f(y)} \right) \right]$$
• Lift is the ratio of the observed support to that expected if X and Y were statistically independent and is calculated as

\[
lift(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X) \cdot \sigma(Y)}
\]

Finally, the fitness function is calculated as

\[
f(x) = \frac{w_s \cdot S + w_c \cdot C + w_l \cdot I + w_j \cdot L + w_j \cdot J_M}{w_s + w_c + w_l + w_i + w_j}
\]

where S denotes Support, C denotes Confidence, I denotes Interestingness, L denotes Lift and J denotes the J-Measure. \( w_s, w_c, w_l, w_i, w_j \) are the weight values.

The mutation probability was set at 0.01. The number of generations is capped at 150.

4.10.4 Results and Discussion

The best performing individuals in the 150 generations are sorted according to their fitness values and from the resulting 150 rules, rules for which the consequent is some attribute other than the quality attribute are pruned. The top 5 association rules that were mined are shown in the Table 4.2.

Table 4.3  Top 5 Rules with Quality Attribute as the Consequent (in terms of fitness)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Fitness Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motivation Level=3 ^ Experience in the domain=3  \rightarrow  Quality=1</td>
<td></td>
</tr>
<tr>
<td>Experience in the domain =2 ^ Ability to work in team =3  \rightarrow  Quality=1</td>
<td></td>
</tr>
<tr>
<td>Experience in Technology = 3 ^ Experience in domain = 2 ^ Motivation = 1  \rightarrow  Quality=0</td>
<td></td>
</tr>
<tr>
<td>Experience in Technology = 0 ^ Ability to work in team = 1  \rightarrow  Quality=0</td>
<td></td>
</tr>
<tr>
<td>Commitment = 3 ^ Responsibility = 3  \rightarrow  Quality=1</td>
<td></td>
</tr>
</tbody>
</table>
The top 3 rules in terms of fitness imply that a high motivation and high domain experience are likely to yield good quality software. Interestingly, the competency in the technology attribute is not present in the top 5 rules even though it is generally perceived as highly important for quality. Another interesting rule that was mined as Motivation = 3 $\rightarrow$ Commitment = 3 $\wedge$ Responsibility = 3. This rule, though was not considered as it gives no indication of quality, reveals an interesting correlation that highly motivated people tend to be more responsible and committed. The third rule gives an insight into one important aspect of human factors pertaining to quality. Notwithstanding the experience in the domain and technology, less motivated individuals tend to impact the quality negatively. Experience seems to offer no guarantees of quality. 78.21% of the mined rules that had quality = 1 as the consequent had a motivation level of 2 and 3 in the antecedent. This is in line with the intuition that motivated individuals to overcome all other obstacles; they might otherwise pertain to experience and competency.

4.10.5 Comparison with Apriori and FP Growth

To demonstrate the potential of GA, the same experiment was done with the Apriori algorithm and the FP Growth algorithm. The following Table 4.3 gives a comparative statement about the performance of the algorithms in terms of execution time and average quality. In terms of time, Apriori seemed to be the worst performer and this is expected given the time required for generating candidate itemsets by the algorithm. Though the difference between GA and FP
Growth in terms of time is very small, there is a considerable improvement in the quality of the rules generated. GA seems to generate more quality rules than FP Growth. This reinforces the opinion that as the complexity of the problem and uncertainty involved increases, trends shift in favor of GA.

**Table 4.4 Comparison of Apriori, FP Growth and GA**

<table>
<thead>
<tr>
<th></th>
<th>Time (in secs)</th>
<th>Average Quality (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apriori</td>
<td>2.09</td>
<td>78.91</td>
</tr>
<tr>
<td>FP Growth</td>
<td>1.81</td>
<td>80.11</td>
</tr>
<tr>
<td>Genetic Algorithm</td>
<td>1.78</td>
<td>84.12</td>
</tr>
</tbody>
</table>

The comparative performances of Apriori, FP Growth and genetic algorithms in terms of time taken are shown in Figure 4.2

**Figure 4.2 Comparison of Apriori, FP Growth and GA in terms of time**
Figure 4.3 compares the average quality obtained by Apriori, FP Growth and GA graphically.

![Comparison of Apriori, FP Growth and GA in terms of Average Quality](image)

**Figure 4.3  Comparison of Apriori, FP Growth and GA in terms of Average Quality**

4.11 SUMMARY

A genetic algorithm was proposed to mine association rules to discover interesting associations between human factors and software quality. Human factors have a great impact on software quality and a careful identification of the significant human factors contributing to high quality is of utmost importance for project managers and organizations in their endeavor to develop quality software. Several association rules are mined, and the mined association rules indicate that motivation is one of the crucial human factors impacting quality. Experience in domain, commitment and responsibility seem to be the other dominating factors in ensuring high quality. The research
Attempts to throw light on the prospects of utilizing association rule mining and genetic algorithm in identifying the crucial human factors impacting quality. Given the strong correlation between human factors and quality, this can be a starting step of an attempt that can lead to perceivable improvements in the quality of the software developed. Managers can reap great benefits by paying attention to the crucial human factors identified. In future, the list of human factors considered can be extended to include more and interesting associations discovered between them and software quality.