CHAPTER 5

Handling Stakeholder Conflict by Agile Requirement Prioritization Using Apriori Technique

5.1 Introduction

Business is emerging with new technologies and approaches every day according to the economic conditions, expectations, emerging trend, market needs, etc. From the beginning, the main objective of any product development is to enhance the features of a system which in turn is valuable to the user (Inayat et al. 2015). The common issue faced by any software development organization is to deliver the system on time and within the budget. In order to compete effectively in a competitive environment, any problem-solving technique should be novel and efficient which is instantly available as and when expected. Rather, to deliver the system in expected time without compromising the quality, an optimized development methodology is highly important. In general, the system is said to be a quality one if and only if it satisfies the needs of a customer. The First thing that comes to our mind when we want to develop a software or product in a frequently changing business trends is agile process development. It is an alternate approach to developing software. Agile is an interval boxed, an iterative method to software delivery that builds software incrementally from the project kick-off, instead of trying to supply it all, once at the end.

Agile process is capable of managing changing requirements. In agile methodology, product releases happen in iterative and incremental work pieces called sprints. The features which have to be released are prioritized according to the customer needs and change requests (Logue and McDaid 2008). The agile process can easily adjust to their project selections due to changing business priorities. Agile projects deliver working software on a consistent schedule usually once or twice in a week (Yu and Petter 2014). These practices hugely help to reduce the risk of dependency on resources that may move away from the project at any point in time. The first phase in developing any software or product is Requirements gathering. The requirements play a key role in any process. Among the software development phases requirement analysis is said to be the most important phase since it is an initial step in
the development and any requirement misunderstanding would lead to chaos in the later stages of software development. The requirements are treated like where the stakeholder gathers together and exchange their ideas, interact with one another, share knowledge and data. The requirements gathered are further developed to a software design followed by coding, which are the downstream artifacts. Each requirement has to undergo a feasibility study to see if they can be implemented or not with the current software hardware environment. The success of process or the failure of the process also depends on the requirements, so they have to be gathered correctly. When there are many requirements there is a chance of project going away from the scope of meeting the needs. So they are to be gathered according to the needs and have to be prioritized properly.

Various requirement prioritization methods are available and it can be chosen as per the projects need which are listed below,

1. Moscow - Must have, Should have, Could have, Wont have.
2. Business value - Based on the business value that will be generated by the company.
4. Validated learning - Developing any feature that has not experimented yet and it is released to get the customer feedback and if it is successful applying it to the next release.
5. Walking skeleton - Requirements which can be completed in a very short span of time and minimal end to end features are selected.

Using these methods one can prioritize the requirement only to some moderate extent which will not consider the stakeholders and customers point of view in every constraint. To satisfy the customer needs, one needs an effective technique to list out the order of requirements. Selection of requirements or features for every release is highly important and can never be taken light. Requirement prioritization plays an important role. The process of prioritizing the requirements provides support for following activities as:

(i) The clients or stakeholders have to identify and freeze the core requirements of the system (Tessem 2014).

(ii) Planning and categorizing the group of requirements in order to implement these requirements in specific releases.

(iii) To deliver a system that satisfies the customer even if partial requirements are selected for implementation.
To predict or expect the customer satisfaction (Brhel et al. 2015).

To reduce stakeholder conflict.

The aim is to prescribe an effective method for prioritizing requirements. Here, Apriori algorithm has been used to generate frequently asked requirements, which are implemented first in order to conquer the stakeholder considerations and to ensure customer satisfaction. To find out the most frequently asked requirements, the Apriori algorithm is applied which in turn helps in decreasing the stakeholder conflict. Apriori algorithm is used widely in order to extract frequent item set from the database. Apriori algorithm is architected such that it operates on databases which contain transactions. Based on the threshold value (we call it minimum support value, this algorithm identifies the frequent item sets which are subsets of transactions in the database. The same technique is applied to prioritize requirements that are frequently asked by stakeholders so that the needs of the stakeholders or customers are met in an efficient and an effective way (Xu and Dong 2013).

5.2 Related Work

Any product a customer is willing to buy depends on how well the product meets the customer needs. Here the major deal is prioritizing the user requirements which yield good results as well as customer satisfaction. Prioritization of requirements is all about making the user stories in the order of its importance and business value. It still depends on many factors like project constraints, stakeholder characteristics, and the environment under which the product is being developed. Further, these requirements are developed as iterations and the requirement with the highest priority is implemented first.

The major stakeholders are the developers of the product who are the most influential people in the requirement prioritization process. Prioritization process includes both developers and customers. The complication begins while involving the multiple stakeholders where each stakeholders idea differs from one another and there the problem arises in the prioritization of requirements (Maiti and Mitropoulos 2015). Here the need of effective and efficient algorithm for requirement prioritization arises.

In Agile development, to carry out proper requirement prioritization, choosing the right stakeholders is the important criteria. The stakeholders should have some experience in agile development and good skill in capturing exact customers needs. They should have direct communication with one another and joined learning experience of both implicit and explicit knowledge.

Selecting the requirements accomplishes the effective prioritization technique, where the selected requirements are sent to the project backlog. From here, the requirements
at the top of the list are taken to the sprint backlog. When all the requirements cannot be developed in one single iteration, few of them are sent to the initial backlog, where the reprioritization occurs along with the newly added requirements. By doing this, project time is extended which may also lead to the loss of product quality. Here also arises the need of effective prioritization technique.

Requirement prioritization is based on some of the factors which include project constraints, Stakeholders characteristics and the environment. These factors play a major role in identifying the requirements for every project.

Using these methods we can prioritize the requirement only to some extent which will not consider the stakeholders and customers point of view in every constraint. To satisfy the customer needs we need an effective technique to list out the order of requirements. The focus was shifted on to stakeholder opinion because in agile we will be releasing products in many different versions. If the first module of the project which the developers or organization members feel important is released, it may not be much effective or much needed to the customer. As the customer may have some basic need with the product, if he is not satisfied with the release, it may lead to many disturbances like project failure or down of organization name in the market etc. So, considering this factor is also important in agile development, a new approach is needed to handle the constraint (Inayat and Salim 2015).

Considering the factors of the project before the requirement prioritization is a good way to get the defined order but still, the effective technique or algorithm is needed for the exact and efficient prioritization of requirements which yields the better quality product. This eventually satisfies the customer needs and expectations. So, in the next part, we discuss the prioritization algorithm called Apriori algorithm which gets the requirements from multiple stakeholders and carries out the process and produces the order of requirements (Serrador and Pinto 2015). This algorithm also solves the problems while involving multiple stakeholders.

Including the existing work like considering the factors and putting in the new prioritization algorithm is expected to yield the efficiently prioritized requirements. Based on the factors and the new algorithm a conceptual framework is to be developed (AL-Taani and Razali 2013).

5.3 Objective

The objective of this chapter is to provide an efficient algorithm for prioritizing the requirements in agile development. For this purpose, we have used Apriori association rule mining technique from data mining.
5.4 Framework to prioritise Requirements

In agile methodology, we are supposed to deliver the product in an iterative and incremental way. The first delivery product should be in such a way that the basic need of creating the product should be met. Later, we add other requirements and release enhanced product. We consider many factors in prioritizing the requirements and a few factors are listed below (Daneva et al. 2013).

(i) The need to hold change.

(ii) Should depend on project constraint.

(iii) The scope of satisfying stakeholder needs.

(iv) Project staffs.

Fig. 5.1 Step by step process of prioritizing the requirements

The conceptual framework includes four modules Fig 5.1. First module emphasizes on the environment under which the prioritization process takes place. The environment has three main factors namely stakeholder view, project constraints and the nature of requirements. These factors will have sub factors like knowledge, learning experience, authority, cost, human resources, risk, complexity, business value, dependencies etc. After the consideration of these factors, the requirements are taken and then we apply Apriori process to prioritize the requirements.

Apriori algorithm is used widely in order to extract frequent item set from the database.
which is related to the frequent requirement set in this chapter. Apriori algorithm is architected such that it operates on the databases which contain transactions. A transaction refers to the set of requirements given by a particular stakeholder. Based on the threshold value (Kakar 2015). The threshold value is known as Minimum support count in this algorithm. The threshold value can be mentioned in two different ways. Firstly, it can be a value directly, which means the support value is the given value. Secondly, if the threshold percentage is given, then the minimum support count is calculated using the formula,

\[
\text{Minimum Support} = \frac{(\text{Given threshold value in percentage})}{100} \times \text{Number of transactions}
\] (5.1)

Round the value to the nearest integer value if any decimal value found.

The Apriori algorithm identifies the itemsets which are subsets of transactions in the database. The vocabulary which was used is,

- R1 means requirement one.
- The itemsets are denoted by using the parenthesis \( \{ , \} \).
- CR means combined requirement set.
- FR means frequent requirement set.

The flowchart is a diagram as shown in Fig 5.2 which gives the flow for the actions of people or related things to an action. So, lets have an overlook of our work in a flowchart, which can give a better idea of the proposal. The flowchart is as follow:
5.5 Algorithm

This algorithm follows a two-step process

(i) Join action

(ii) Prune action

- Join step: A set of items are generated by joining within itself.
- Prune step: Scan database for finding the items that satisfy the minimum support value and ignore all that does not satisfy minimum support value.

Minimum support value refers to the number of stakeholders who support a particular requirement.

This way frequent requirements are found by the continuous join of one requirement with one another until the process terminates. This is how process termination is explained in the next session. After it undergoes the Apriori process, the top requirements and strong associations are identified. Requirements at the top are taken to sprint backlog for implementation. This materializes in the backlog module. Later, it produces the end product which has high-quality requirements and thus it leads to the customer satisfaction.
Algorithm 3 Apriori process

Input: DT (Transactions in the database) RK (Requirements where k=1, 2, n) (Minimum support value)
Output: Rf (Frequently asked requirements)
BEGIN
STEP 1:
Scan DT and generate CR1 (Count for each requirement)
STEP 2:
Compare each CR1 requirements count with minimum support value IN CR1
for K=1 to n
if (count(RK) ≥ α)
then generate FR1 (list of Rk that satisfy min sup count) and omit others
END if
END for
STEP 3:
Generate CR2 (a set of requirements generated by joining with itself) CR2=FR1 ⊲◁ FR1 (FR1 join FR1)
STEP 4:
Repeat Step 1, step 2, step 3 until Rf is obtained
OBTAIN Rf
END

5.6 Experimental Analysis and Results Discussion

Consider stakeholders and let R1, R2, R3, R4, R5 be the requirements they wanted and D be the database where we store these details as shown in Fig 5.3.
R1 - User Interface
R2 - Security
R3 - Portability
R4 - Performance
R5 - Memory Efficient

Let minimum support count required is 2. This implies 2 out of 9 stakeholders support a particular requirement (i.e. minimum offered support = 2/9 = 22 %)
We have to primary find the frequently used requirements using Apriori algorithm as shown in Fig 5.4.

- Scan D and generate CR1 (Count for each requirement).

Compare each CR1 requirements count with minimum support value. Omit requirements that do not satisfy the minimum support value and generate FR1 (Frequent requirements).
### Fig. 5.3 Requirements Gathered

<table>
<thead>
<tr>
<th>Stakeholders</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stakeholder 1</td>
<td>R1, R2, R5</td>
</tr>
<tr>
<td>Stakeholder 2</td>
<td>R2, R4</td>
</tr>
<tr>
<td>Stakeholder 3</td>
<td>R2, R3</td>
</tr>
<tr>
<td>Stakeholder 4</td>
<td>R1, R2, R4</td>
</tr>
<tr>
<td>Stakeholder 5</td>
<td>R1, R3</td>
</tr>
<tr>
<td>Stakeholder 6</td>
<td>R2, R3</td>
</tr>
<tr>
<td>Stakeholder 7</td>
<td>R1, R3</td>
</tr>
<tr>
<td>Functional</td>
<td>R1, R2, R3, R5</td>
</tr>
<tr>
<td>Non-functional</td>
<td>R1, R2, R3</td>
</tr>
</tbody>
</table>

### Fig. 5.4 Generation of 1 frequent requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Support count</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>6</td>
</tr>
<tr>
<td>R2</td>
<td>7</td>
</tr>
<tr>
<td>R3</td>
<td>6</td>
</tr>
<tr>
<td>R4</td>
<td>2</td>
</tr>
<tr>
<td>R5</td>
<td>2</td>
</tr>
</tbody>
</table>

**Compare requirements support count with minimum support count**

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Support count</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>6</td>
</tr>
<tr>
<td>R2</td>
<td>7</td>
</tr>
<tr>
<td>R3</td>
<td>6</td>
</tr>
<tr>
<td>R4</td>
<td>2</td>
</tr>
<tr>
<td>R5</td>
<td>2</td>
</tr>
</tbody>
</table>

**FR1**

### Fig. 5.5 Generation of 2 frequent requirements

<table>
<thead>
<tr>
<th>Req. Set</th>
<th>Support Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1, R2</td>
<td>4</td>
</tr>
<tr>
<td>R1, R3</td>
<td>4</td>
</tr>
<tr>
<td>R1, R4</td>
<td>1</td>
</tr>
<tr>
<td>R1, R5</td>
<td>2</td>
</tr>
<tr>
<td>R2, R3</td>
<td>4</td>
</tr>
<tr>
<td>R2, R4</td>
<td>2</td>
</tr>
<tr>
<td>R2, R5</td>
<td>2</td>
</tr>
<tr>
<td>R3, R4</td>
<td>0</td>
</tr>
<tr>
<td>R3, R5</td>
<td>1</td>
</tr>
<tr>
<td>R4, R5</td>
<td>0</td>
</tr>
</tbody>
</table>

**CR2**

<table>
<thead>
<tr>
<th>Req. Set</th>
<th>Support Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1, R2</td>
<td>4</td>
</tr>
<tr>
<td>R1, R3</td>
<td>4</td>
</tr>
<tr>
<td>R1, R5</td>
<td>2</td>
</tr>
<tr>
<td>R2, R3</td>
<td>4</td>
</tr>
<tr>
<td>R2, R4</td>
<td>2</td>
</tr>
<tr>
<td>R2, R5</td>
<td>2</td>
</tr>
<tr>
<td>R3, R4</td>
<td>0</td>
</tr>
<tr>
<td>R3, R5</td>
<td>1</td>
</tr>
<tr>
<td>R4, R5</td>
<td>0</td>
</tr>
</tbody>
</table>

**FR2**

92
All the requirements satisfy the minimum support value. Now, we join one requirement with other and repeat the same process as shown in Fig 5.5.

- Generate CR2 (a set of requirements generated by joining with itself and count the occurrence)
- CR2=FR1 $\bowtie$ FR1 (FR1 join FR1)
- Repeat step 1, step 2 and generate FR2, FR3...so on until the process terminates.
- Generate CR3, CR3=FR2 $\bowtie$ FR2 (FR2 join FR2)
- Repeat step 1, step 2 and generate FR3

Eliminate requirements that do not satisfy minimum support value. Again, join the requirements with one another and then repeat the process.

Frequent requirement set is obtained i.e., R1, R2, and R3 and R1, R2, R5.

The algorithm uses FR3 Join FR3 to generate a frequent set of 4-item sets, CR4. Although the join results in R1, R2, R3, R5, this item set is pruned since its subset R2, R3, R5 is not frequent as shown in Fig 5.6.

Thus, CR4 = NULL, and algorithm terminate.

We can first develop a product using either R1, R2, R3 or R1, R2, R5.

Let us consider the requirements R1, R2, and R5.

Now, generate association rules. Association rules are used to find out the relation between item sets in large databases (Chun-Sheng and Yan 2014). It uses some measures of interestingness (confidence) to identify strong rules. Confidence is a sign of how frequently the rule is valid. These association rules help us to find how the requirements should be related to each other so as to get appreciable output.
Let the minimum confidence threshold be at least 70%.

Here, the confidence threshold value means the percentage of which the main objective of the product is satisfied.

Association rule that generates confidence value above 70% are taken into consideration and others are rejected.

The confidence of a rule is defined as:

\[ \text{Confidence}(AB) = \frac{\text{Support}(AB)}{\text{Support}(A)} \quad (5.2) \]

where, \( \text{support (A \cup B)} \) = number of transactions having both A and B

\( \text{Support (A)} \) = number of transactions having A

Generation of Association Rules from Frequent Requirements

**AR1**: \( R_1 \wedge R_2 \rightarrow R_5 \)

\[ \text{Confidence} = \frac{\text{Support}(R_1, R_2, R_5)}{\text{Support}(R_1, R_2)} \]

\[ = \frac{2}{4} \]

\[ = 50\%, \]

which is less than 70% and thus AR1 is Rejected.

(\( R_1 \) and \( R_2 \) are dependent on \( R_5 \) produces a poor confidence value).

**AR2**: \( R_1 \wedge R_5 \rightarrow R_2 \)

\[ \text{Confidence} = \frac{\text{Support}(R_1, R_2, R_5)}{\text{Support}(R_1, R_5)} \]

\[ = \frac{2}{2} \]

\[ = 100\%, \]

which is greater than 70% and thus AR2 is Selected.

(\( R_1 \) and \( R_5 \) are dependent on \( R_2 \) produces a good confidence value).

**AR3**: \( R_2 \wedge R_5 \rightarrow R_1 \)

\[ \text{Confidence} = \frac{\text{Support}(R_1, R_2, R_5)}{\text{Support}(R_2, R_5)} \]

\[ = \frac{2}{2} \]

\[ = 100\%, \]

which is greater than 70% and thus AR3 is Selected.

(\( R_2 \) and \( R_5 \) are dependent on \( R_1 \) produces a good confidence value).

**AR4**: \( R_1 \rightarrow R_2 \wedge R_5 \)

\[ \text{Confidence} = \frac{\text{Support}(R_1, R_2, R_5)}{\text{Support}(R_1)} \]

\[ = \frac{2}{6} \]

\[ = 33\%, \]

which is less than 70% and thus AR4 is Rejected.

(\( R_1 \) is dependent on \( R_2 \) and \( R_5 \) produces a poor confidence value).
AR5 : $R_2$
\[ \rightarrow R_1 \land R_5 \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \text{Confidence} \]
\[ = \frac{\text{Support}(R_1, R_2, R_5)}{\text{Support}(R_2)} \]
\[ = \frac{2}{7} \]
\[ = 29\%, \]
which is less than 70% and thus AR5 is Rejected
($R_2$ is dependent on $R_1$ and $R_5$ produces a poor confidence value).

AR6 : $R_5$
\[ \rightarrow R_1 \land R_2 \rightarrow \rightarrow \rightarrow \rightarrow \text{Confidence} \]
\[ = \frac{\text{Support}(R_1, R_2, R_5)}{\text{Support}(R_5)} \]
\[ = \frac{2}{2} \]
\[ = 100\%, \]
which is greater than 70% and thus AR6 is Selected
($R_5$ is dependent on $R_1$ and $R_2$ produces a good confidence value).

In this way, three strong association rules (AR2, AR3, and AR6) are found. Using Apriori algorithm, we find the most frequently sought requirements and implement them first for the first delivery of the product. Later, we add requirements based on the stakeholder request and then release the product. Hence, we have seen how Apriori algorithm helps in prioritizing the requirements. But, it consumes a lot of time. When there are thousands of requirements it consumes hours of time. So, in order to overcome this, there is a technique called Transaction reduction technique. In this transaction reduction method, we omit all the requirements which do not satisfy the minimum support value. This helps in decreasing the execution time. Then, we apply Apriori algorithm to the remaining requirements and find the requirements that frequently occur.

5.7 Performance Evaluation

Let us consider $R_1$, $R_2$, $R_3$...$R_n$ to be the requirements, where $n$ = number of requirements gathered. There are many existing prioritization methods such as Moscow, Business value etc., which are used for prioritization as shown in Table 5.1. But, in these methods the accuracy (A) decreases with the increase in requirements and Time (T) also increases.

\[ R \propto \frac{T}{A} \]  \hspace{1cm} (5.9)

The number of requirements is directly proportional to the time consumed and indirectly proportional to Accuracy. As per the general relation between the efficiency and time is given by

\[ \text{Efficiency} \propto \frac{1}{T} \]  \hspace{1cm} (5.10)
The efficiency is indirectly proportional to the time consumed. As the number of requirements increases, it results in an increase in time to prioritize them. While calculating efficiency in Apriori, the number of requirements decreases through step by step due to pruning. So it shows that Apriori is more efficient than existing methods as shown in Fig 5.7.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n=100</td>
</tr>
<tr>
<td>Apriori</td>
<td>84</td>
</tr>
<tr>
<td>MoSCoW</td>
<td>90</td>
</tr>
</tbody>
</table>

![Fig. 5.7 Comparison of Efficiency over Moscow and Apriori](image)

In previous methods such as Moscow, Validated learning, Walking Skeleton etc., as the requirements grow the accuracy to prioritize the requirements declines. The way of prioritizing differs based on the peoples perspective and finally leads to inaccurate results and stakeholder conflict. Taking each requirement and conducting Moscow makes it a complex task as requirements grow and mapping sometimes may go wrong Table 5.2.

\[ A = \frac{1}{n} \sum_{i=1}^{n} x_i \]  

(5.11)

where \( X_i \) = Accuracy at different points of time
\( n \) = Number of different points of time
Average accuracy = \(\frac{90 + 85 + 82 + 79 + 73 + 70}{6}\)
\[= 79.83 \text{ (Approximately 80\%)}\]  

Table 5.2 Comparison of Time and Accuracy over Moscow and Apriori

<table>
<thead>
<tr>
<th>Methods</th>
<th>Time and Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n=100</td>
</tr>
<tr>
<td>Apriori</td>
<td>89</td>
</tr>
<tr>
<td>MoSCoW</td>
<td>90</td>
</tr>
</tbody>
</table>

![Comparison of Time and Accuracy over Moscow and Apriori](image1)

(a) Comparison of Time and Accuracy over Moscow and Apriori

**Fig. 5.8** Comparison of Time and Accuracy over Moscow and Apriori

The above graphs shows how the time increases and accuracy decline with the increase in the number of requirements as shown in Fig 5.8. This is what happens in prioritizing requirements using Moscow process. As requirements increase, it is very difficult to tag each requirement with Must have, Should have, Could have, wont have. To overcome this problem, the apriori algorithm is used to prioritize requirements effectively with good accuracy. In this chapter, we prioritized requirements using Apriori algorithm. Applying Apriori algorithm always produces the correct results irrespective of the number of requirements. When requirements are prioritized using Apriori algorithm, the accuracy does not get affected with the increase in requirements. So, when compared with the existing methods, prioritizing using Apriori produces better accuracy.

Average accuracy = \(\frac{89 + 89 + 89 + 89 + 89 + 89}{6}\)
\[= 89\%\]  

The graph reveals how accuracy remains constant and consistent irrespective of the increase in a number of requirements. The accuracy remains constant when requirements
are prioritized using the apriori algorithm. The time increases with increase in require-
ments. But, there is no effect on accuracy.
When compared, the accuracy of requirements prioritized using Apriori is higher than
that of the existing method Moscow (89% > 80%). Prioritizing using Apriori produces
better accuracy irrespective of the increase in a number of requirements.
Here we are adding a measure of performance for the algorithm to show its complexity.
Suppose, N be the number of transactions, T be the value of threshold and U be the
number of unique elements. The complexity for generating a set of size j is O (Uj)
and the time for finding support for every set can be done in O (n). Therefore, time
complexity would be

\[
\text{Complexity} : O[(U + N) + (U \land 2 + N) + (U \land 3 + N) \ldots ]
\]
\[
= O[TN + (U \land 1 + U \land 2 + \ldots U \land T)]
\]
\[
= O(TN + (1 - U \land T)/(1 - U))
\]

5.8 Summary

Agile methods deliver the product iteratively in the form of several iterations. The fea-
tures which have been released are prioritized according to the customer requirements
and changing proposals. Here, the major deal is prioritizing the user requirements which
yield good results as well as customer satisfaction. Existing methods to prioritize re-
quirements such as Moscow, Business value, validated learning, and Walking skeleton
are not able to address the problem of stakeholder conflict. These existing methods are
used for prioritizing but when there is an increase in the number of requirements, they
provide less accuracy and time taken will be large which results in the late release of
the agile product which may not be acceptable for the customer.
Apriori algorithm is used to find out the frequent requirements that are wanted by most
of the users and thus reduces stakeholder conflict. This method is followed by finding
the association between the frequent requirements which results in a greater advantage
for the software developers, as they can understand which set of requirements can af-
fect other requirements. The main advantages of using this method are using large data
properly, easily implementable, easily parallelized. By these advantages, this method
had highlighted its presence among the existing methods. In Apriori, the accuracy re-
mains almost constant irrespective of a number of requirements as at every prune step
fewer priority requirements are detached. The result produced by applying Apriori al-
gorithm is observed to be more accurate than the existing methods. This also proves
that its efficiency dominates the efficiency of existing methods.