Journal of Theoretical and Applied Information Technology
31st January 2014. Vol. 59 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

# A FIVE-FACTOR SOFTWARE ARCHITECTURE ANALYSIS BASED ON FAR FOR ATM BANKING SYSTEM

**[1]T.K.S. RATHISH BABU, [2]DR.N.SANKARRAM**

[1.]Assistant Professor, Department of Computer Science and Engineering, S.K.R. Engineering College, Chennai, India

[2.]Professor, Computer Science and Engineering, R.M.K college of Engineering and technology, puduvoyal, Chennai India

**ABSTRACT**

Software architecture represents the high level structures of a software system. It can be defined as the set of structures required to explain about the software system which comprise the software elements, the relations between them, and the properties of both the elements and relations. A major challenge in the software architecture design process is the accurate prediction and improvement of the software performance characteristics like outage frequency and duration. This paper proposes hybrid software architecture for an ATM banking system to overcome the difficulties of an existing architecture. The proposed system is based on the Fuzzy Association Rules (FAR). During the extraction of the FAR, a confidence index and theAprioriGen algorithm is utilized to compute the inverse fuzzy transform. Also, this paper presents a review of some of the Software Architecture Analysis Methods (SAAM). The performance of the proposed methodology is analyzed based on the metrics like reliability, flexibility, adaptability and security. The proposed software architecture is compared with the various existing software architectures. The implementation results obviously proves that the proposed methodology performs better than all the other existing software architectures.

**Keywords:** *Adaptability, Apriori Algorithm, Flexibility, Fuzzy Association Rule (FAR), Software Architecture Analysis Methods (SAAM), and Security.*

## 1. INTRODUCTION

Thesoftware architecture is defined as the "backbone" of a system at the maximum level of abstraction. It is a high-level representation that characterizes the major structure and communications of the components of a system and the system's communication with the external environments. It describes the components of the system and their related interconnections. Software architecture contributes to the development phase of the software and it has a direct effect on the quality and cost of the software. The software architects should develop the software architecture that can be altered without any risk of degradation. The software architecture analysis verifies the quality requirements to be addressed in the software design and detects the potential risks. The analysis of software architecture aims to pre-estimate the quality of a system.

A software architecture for an ATM banking system is proposed based on Fuzzy Association Rules (FAR). The recent various software architecture analysis methods (SAAM) and implementation of FAR are reviewed. A detailed performance analysis of the software architecture is performed in terms of reliability, flexibility, adaptability, and security. The performance analysis achieves various goals, depending on the applicationdevelopment phase [49]. The most dependent parameters are reliability and performance. The software architecture reliability involves subsystem interactions [19], fuzzy logic model [46], prediction models [39], and quality requirements [22]. The performance analysis of the software architecture involves distributed systems model [38], component-based analysis [45], multi-core platforms [41], Kieker Tool [37], and service-oriented architecture [2].

The proposed software architecture is analyzed, classified using FAR, and it observed that it is better compared to various existing software architectures in terms of reliability, flexibility, adaptability, and security.

## 2. RELATED WORK

### 2.1. Software Architecture Analysis for Enterprise Information Systems

Many software architecture proposals are accessible to industrial engineers in the development of enterprise information systems, but the systematic solutions for the assessment of software architecture are scarce [5]. Enterprise

Journal of Theoretical and Applied Information Technology
31st January 2014. Vol. 59 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Information Systems (EIS) are the main IT assets for industrial organizations to plan, control and schedule their business process. EISs have become major enablers for the advanced
enterprises to achieve effectiveness and streamline processes. An important component of the EIS is *software architecture*. The software architecture of EIS comprises a group of system components and their topological relations. The architecture analysis depends on the early decisions about the high level design of software systems.

1) *Classification of NFRs:*

During the design and implementation of an EIS, the software architecture must support the principle business factor known as quality attributes or non-functional requirements (NFRs). During the selection of software architecture for an EIS, one needs to consider many and often conflict NFRs. For example, the real time performance and a system's flexibility are conflicting with each other and must be balanced in software development. In a user-oriented technique, the key NFRs are estimated and the quality attribute scenarios are furnished to compute the degree to which the choices of the software architecture have influenced the satisfaction of the NFRs.

During the development of software architecture, the functional requirements define what the system can do and the non-functional requirements (NFRs) elaborate the adaptability of the system to satisfy the required functions. Some of the key NFRs for EISs are enlisted in TABLE 1., along with their related concepts and topics.

*Table 1: Key Nfrs For Eiss*

| NFR | Related Concepts | Topics |
|---|---|---|
| Customer - Oriented | Intelligence Customization Flexibility | Aligning an organization's business with customer's needs |
| Performance | Efficiency Schedulability Real-time Memory Usage | Optimization of system performance under many conditions |
| Agility | Adaptability Autonomy Flexibility | Rapid response to variations and uncertainties |
| Reliability | Robustness Fault Handling Accountancy | Control a system to resist or product failures |
| Security | Information Protection Safety | Free from malicious threats |

The NFRs such as reliability and security are software-driven, whereas NFRS such as flexibility, performance and adaptability are business-driven.

2) *Classification of Software Architecture:*

An important aspect in the software architecture development is the patterns codification which is used as the blueprint of constraints, components, and their relations. Patterns develop the general the solutions that can be used again to fasten the software development. Some techniques employ the operational patterns for the EIS design as follows.

**2.1.1. General Purpose Software Packages:**

This technique encapsulates the algorithms and data structures to implement a general and customizable solution of the business requirements. Some of the packages used are database-centered data sharing, event driven message invocation and pipeline-based data processing.

**2.1.2. Domain-specific software architecture:**

It focusses on a specific domain and has special components. Some of the examples are Enterprise Java Beans (EJB), business component factory and Microsoft's Component Object Model (COM+).

**2.1.3. Distributed Computing** involves many elements interacting and coordinating to achieve a goal. Some of the architectural options are client-server, peer-to-peer and n-tier architecture.

***2.1.4.* Agent and Multi-agent Systems (MAS):**

An agent is an independent element in the environment; whereas MAS consists of a set of agents. The agents inside the MAS can coordinate with each other to attain the goals at the system level.

**2.1.5. Service-oriented Architecture (SoA):**

It is the recent type of software architecture. It combines heterogeneous platforms and allows an EIS to enlarge its capabilities by employing reusable software modules.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

### 2.2. A Threat-model Based Security Testing for Software Architecture

Some of the issues in the cyberspace community are denial-of-service attacks, corruption of data and disclosure of confidential information [7]. Some of the issues affecting the software security are cross-site scripting (XSS) or SQL injection. Threat modeling is useful in both software testing phase and software design and development phases.

Threat modeling is a systematic way to detect threats that might bypass the security. The test case generation from the threat models has not been studied much, so a threat model-based security testing is proposed, which automatically produces test sequences from threat trees and converts them into executable tests.

The threat model examines the application in the view of a potential attacker and aids them estimate the potential security risks. The main functions of threat model are detection of the application's assets, determination of the threats to an application, ranking the threats and palliation of threats.

Threat modeling involves the identification, specification, evaluation and counter-measurements against potential security attacks. The threat modeling can be performed at different levels of abstraction and granularity. Some of the notations used in threat modeling are threat trees, threat nets, and misuse cases. Threat nets are built upon Petri nets, a mathematically based principle for modeling and checking distributed systems. Misuse case modeling defines misuse cases as threats to use cases and prefers mitigation use cases.

Mutation analysis is a technique which produces mutants by injecting threats deliberately and estimates how many of the injected threats can be revealed by security tests. There are three types of mutation analysis; they are implementation-level security mutation, mutation of access control policies and specification-level security mutation. In implementation-level security mutation the vulnerabilities are injected into the implementation and the number of vulnerabilities that can be revealed by the security tests is evaluated. Specification-level security mutation introduces errors in the description of the security-related behaviors so as to produce threat scenarios. In mutation analysis of access control policies, there are two important systems to create access and control policies. The first system uses role-based access control (RBAC) or organization-based access control. The second system uses eXtensibleAccess Control Markup Language (XACML) to produce the original implementation of a security scheme. During the transformation of the test sequences to the executable test codes, the feasible input data must be considered carefully.

### 2.3. Detection of Software Security Patterns

The use of security patterns improves the software security in the early software development stages [51]. The security patterns in the code are detected by a reverse engineering tool-suite known as *Bauhaus*. This approach detects the *Single Access Point* security pattern. The recent security tools support the developers during the implementation phase depending on the static analysis.

The software has to be altered frequently due to the varying constraints, bugs and security defects. Also, the reconstruction of patterns in developed systems is tedious. Security patterns should be acknowledged during the maintenance process to ensure security objectives. A Resource Flow Graph (RFG) representation is used by the Bauhaus tool-suite. This method reconstructs the software architecture by a hypothetical architecture to the actual software architecture obtained from the source code.

### 2.4. Software Architecture Adaptability Based on QoS Self-Adaption

The software systems should be able to adapt themselves to the environment dynamically to meet both the functional and non-functional requirements [42]. The functional requirements focus on the overall implementation logic and the non-functional requirements focus on the QoS levels to be ensured.

Adaptability has an effect over other software qualities such as performance, maintainability or reliability. Also in the worst case, the improvement of adaptability could lessen other qualities of the system. So, an efficient trade-off must be maintained between the system adaptability and other quality of the system. These adaptability metrics support a higher degree of quantification from the basic metric. The metrics are also quantified besides the tracking.

### 2.5. Construction and Exploitation of Flexibility in Software Architecture

Flexibility can be attained by aspect-orientation. It is a mechanism which focuses on the design-time separation of the crosscutting system features which

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

are embedded into the system [18]. This orientation permits an additional dimension of decomposition and strongly localized changes. Software Product Lines are a mechanism for systematic usage of commonalities among software products. This is attained by systematically reusing the common divisions of the systems. The commonalities and variations among the products are analyzed.

Software Architecture Analysis Method (SAAM) and Architecture Trade-off Analysis Method (ATAM) generally focus on quality attributes, and Architecture-Level Modifiability Analysis (ALMA) focus on maintainability. ArchE design assistant permits the monitoring of modifiability during the time of designing, but with particular support on selection of architectural techniques.

### 2.6. Software Lifecycle

There are some common stages that can be seen in every software process, where each stage consists of a set of well-defined functions [50]. These stages identify different abstractions of the software under development.

*Requirement Specification* concentrates on the functionalities of the system and on their operational limitations. *Software design and implementation* discusses about the creation of the software system according to its requirements. *Software verification and validation* is a level which proves that the software system conforms to the limitations in the specification stage. *Software evolution* is a period when the changes in the software architecture occur. The *waterfall* process model organizes and explains the Lifecycle stages as shown in Fig. 1.

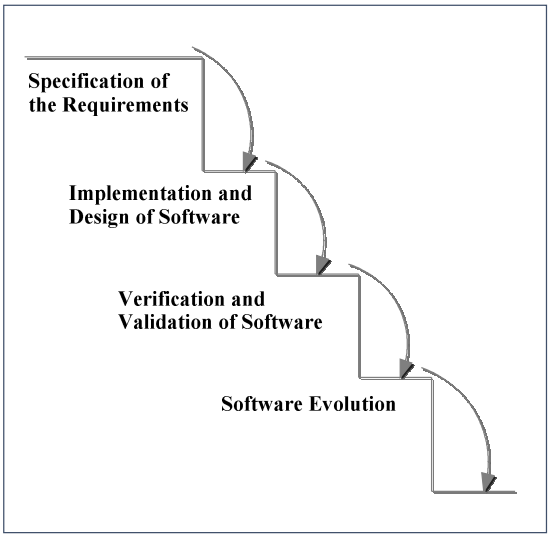



*Fig. 1. Waterfall Process Model.*

Another software process model is the *iterative* process model, which deals with thespecification, application and validation functions concurrently in order to rapidly create an initial edition of the software system, that can be later refined through iterations as shown in Fig. 2.

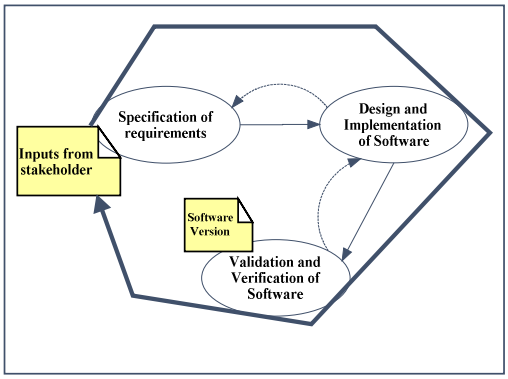



*Fig. 2. Iterative Process Model.*

Journal of Theoretical and Applied Information Technology
31st January 2014. Vol. 59 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

## 3. SOFTWARE ARCHITECTURE ANALYSIS

The proposed ATM banking software architecture and some of the methods for the software architecture analysis with respect to evaluation perspectives are discussed in this section.

### 3.1 Proposed ATM Banking Software Architecture Analysis

The software architecture for an ATM banking system is proposed based on Fuzzy Association Rules (FAR) and Erlang distribution. The flow of the proposed model is given in Fig. 3.

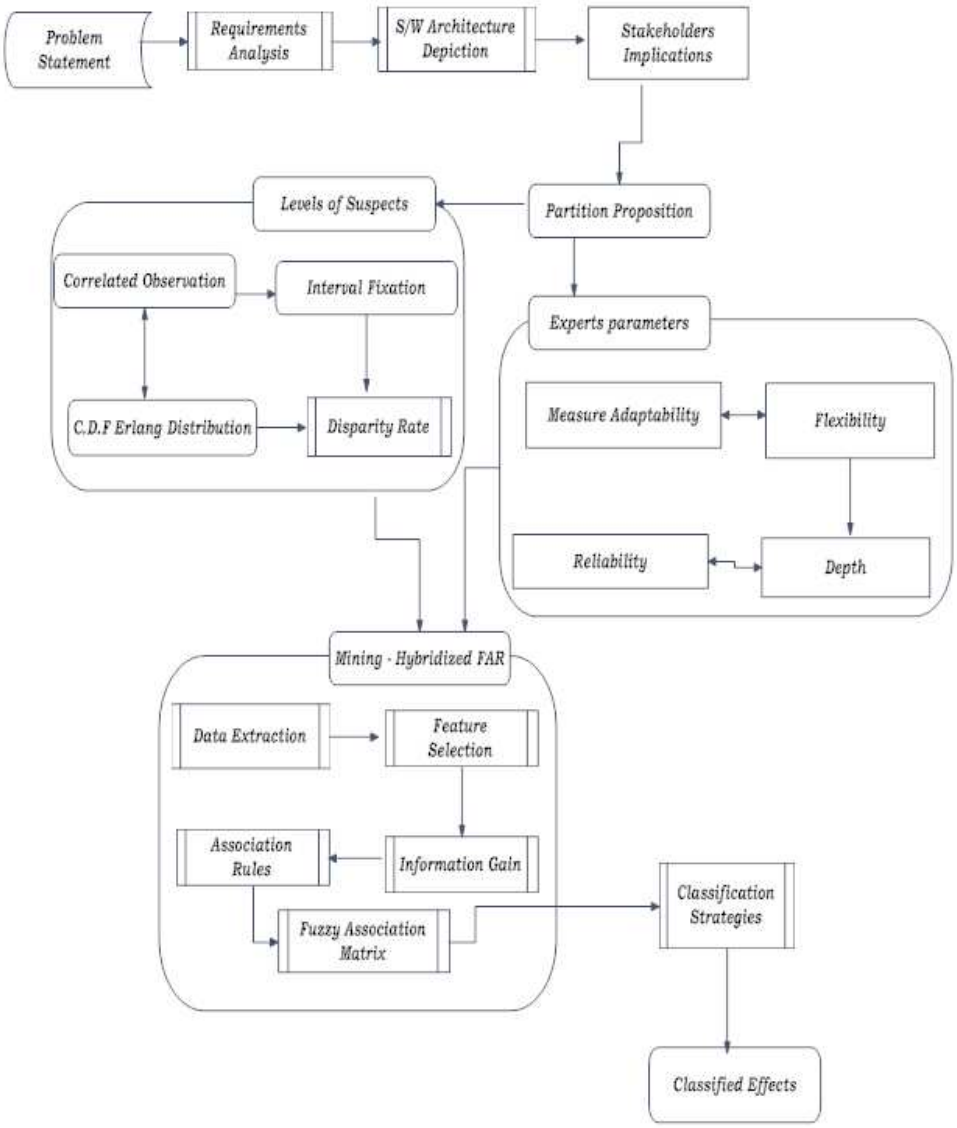



*Fig. 3. Software Architectural Design Process.*

The process model involves an input of schema file for the construction of the software architecture. Some of the operations in the ATM banking software architecture are *GetBalance*, *UpdateBalance*,*Deposit*, and *Withdraw*. Some of the messages in the software system are *BalanceMessage*, *BalanceChange*, and *CustomerMessage*. Based on the requirements of the ATM banking system, the software architecture is depicted.

The various parameters in the software architecture analysis are computed based on disparity rate and CDF (Cumulative Distributive Function) of the Erlang distribution. The computation of the parameters for software architecture analysis are given in TABLE 2.

Journal of Theoretical and Applied Information Technology
31st January 2014. Vol. 59 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

*Table 2: Computation Of Parameters For Software Architecture Analysis*

| Parameter | CDF of Erlang Distribution F (x, y) | *x* | y |
|---|---|---|---|
| Performance | x + y/2 | Memory utilization | Execution time |
| Flexibility | 100 - (x + y) | Dependency of software | Error probability |
| Reliability | (x + y)/2 | Closed loops | Execution time |
| Adaptability | x + y/2 | System requirements | Easy to use |
| Security | x - y | Data privacy | Dependency of network |

The classification results of the software architecture analysis are obtained by the use of Fuzzy Association Rules (FAR) and threshold value for each parameter.

### 3.2. Decision-making in Software Architecture Design

The software architecture of an intensive system can be defined as the group of relevant design decisions that has an impact on the qualities of the system [48]. A general of the software architecture design process is given in Fig. 4.

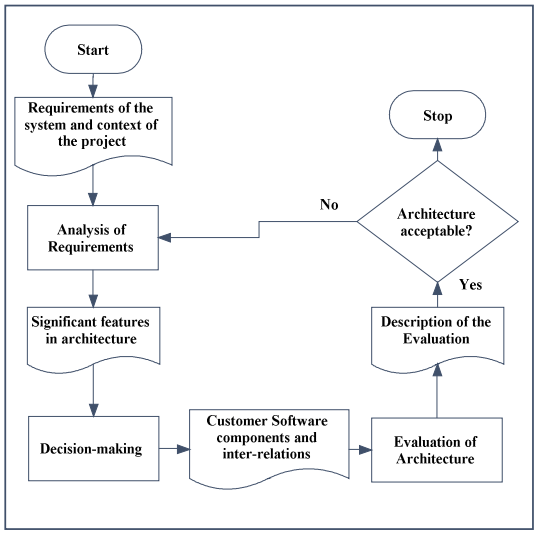



*Fig. 4. Software Architectural Design Process.*

### 3.3. Reliable Software Architecture

Reliability in Software Architecture has impact both on the cost of field operation and the experience of the users [36]. An integrative software reliability structure is framed to fulfill the emptiness between predictions, expectations, and the real behavior of systems. An integrative algorithm implementing the design for reliability (DfR) in software architectures is given in Fig. 5. And the general scheme for the software reliability prediction is given in Fig. 6.

### 3.4. Measurement of Software Architecture Reliability by a Fuzzy Model

Several models have been proposed for predicting the quality attributes of the software architecture [44]. This technique involves a Fuzzy model for the prediction of software reliability. Some of the parameters considered for the prediction analysis are Availability, Recoverability and Failure Probability. The Fuzzy Model aids to evolve the intermediate levels between the reliable and unreliable states of the software architecture.

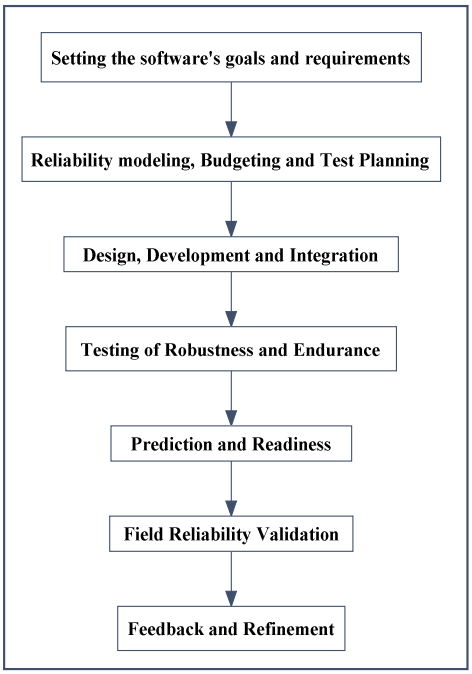



*Fig. 5. Integrative Design for Reliability.*

### 3.5. Prediction of Reliability in Fault-Tolerant Software Architectures

The software fault tolerance schemes improve the reliability of the software architectures [52]. A new approach is proposed for the analysis of the effect of software tolerance scheme for multiple and differing software architectures. This approach models the various alternatives of the software architecture and configurations of the product line from a shared core property. This mechanism provides flexibility to various fault tolerant

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

schemes. The models are transformed into Markov chains by a tool and the system (software and hardware) reliabilities are computed. This provides a validation process for the software architects during the early development stages.

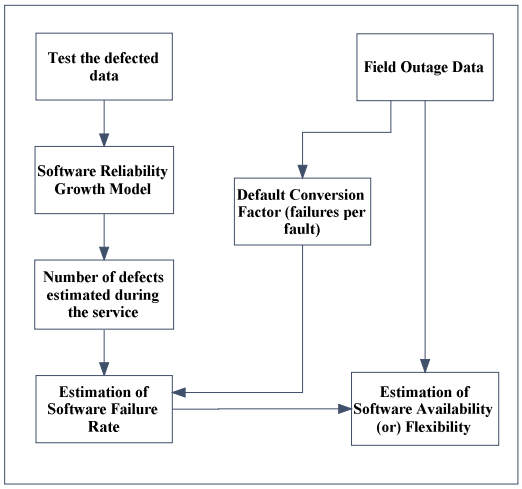



*Fig. 6. Overview of Software Reliability Prediction.*

### 3.6. Performance Analysis of an Adaptive Software Architecture for Multi-Core Platforms

This analysis method focuses on the performance parameters for an adaptive software architecture, which has many levels of fault detection, recovery and masking through reconfiguration of the architecture [41]. The architecture is initialized with a formal requirement model denoting the multiple levels of information assurance and functional capability. The architecture encompasses a multi-layer design to apply the constraints using N-variant techniques. The architecture also integrates a reconfiguration scheme that utilizes the lower layer to assess the higher layers. When a fault is detected the reconfiguration scheme reconfigures the system to maintain the required services.

A general reliability model is provided with cross-surveillance for reconfiguration, based on the generalized stochastic Petri nets. Then, a probabilistic automation-based behavioral model is defined. This model is appropriate for modeling the problems in securing by value faults. Petri net permits the reliability modeling and reconfiguration and the performance analysis is given by the probabilistic model checking. The goal of this analysis is to capture the computational power of the under-utilized cores for enhancing the survivability and adaptability of critical applications. The layered architecture provides on-the-fly reconfiguration.

### 3.7. Security Analysis of Software Architecture Based on Analytical Hierarchy Process (AHP)

The evaluation of the software security at an early stage is important, as it guarantees the stakeholder's security objectives [53]. A robust security evaluation method to estimate the security support of software architecture in the design phase is developed and is known as Security Evaluation Framework (SEF). The SEF also has some limitations such as the securityevaluation considers only the architectural level and not the scenarios on the structural components. A hybrid software security risk evaluation model based on Analytical Hierarchy Process (AHP) is proposed. The SEF consists of six steps [53] as described in Fig. 7.

The initial constituent of the process is the evaluation of a scenario-based architecture. The evaluation of the software architecture is a competent way of guaranteeing design quality. The architecture's ability to render a system that satisfies the stakeholders' quality requirements to detect the potential risks. The involvement of the security patterns increases the quality of the security constituents in the architecture.

### 3.8. Software Architecture Adaptability Based on QoS Self-Adaption

The design of the systems in Software Engineering must be able to adapt to quick alterations of their requirements and evolve over time [42]. The quantification and evaluation of metrics such as software adaptability are defined. The quality of service (QoS) must be ensured by the values of these metrics. These metrics are used by the software architect to assist the system adaptation to satisfy the overall quality requirements.

Journal of Theoretical and Applied Information Technology
31st January 2014. Vol. 59 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

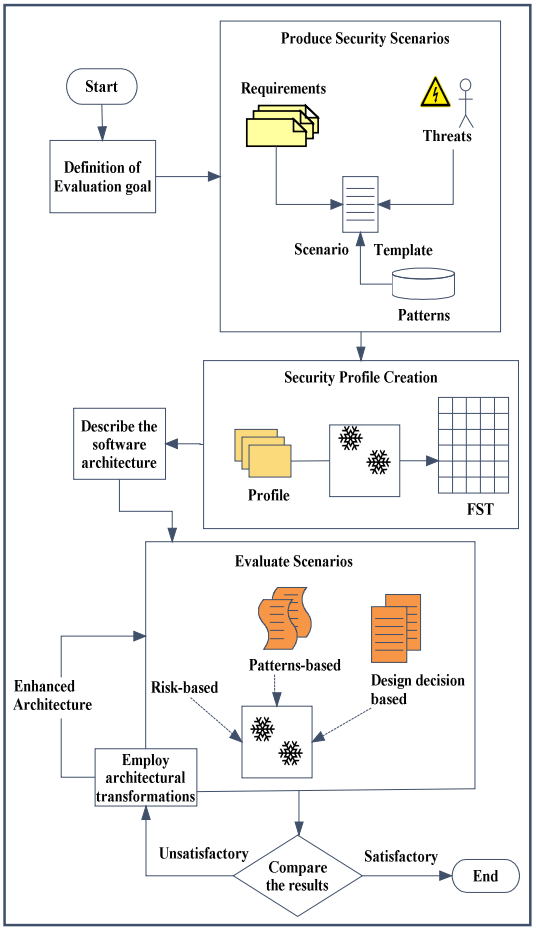



*Fig. 7. Security Evaluation Framework (SEF).*

The Analytical Hierarchy Process (AHP) is a decision making technique used in a multivariable decision problem. The construction of the hierarchy is performed so that the elements at similar level are of the same order of magnitude and must be associated with some or all elements in the nextupper level. After the construction of hierarchies, the prioritization process determines the relative importance of theconstituents in each level of the hierarchy. The AHP employs a pairwise comparison matrix to compute the relative objective weight of factors. The components in each level are pairwise compared relative to their importance in deciding under consideration. A pairwise comparison matrix is constructed to compute the relative importance between factors of the decision elements.

### 3.8.1. Quantification of the Software Architecture's Adaptability:

The adaptability metrics of the software architecture are based on the system's architectural knowledge; hence, adaptability can be computed before the software designing process.

A system of $n$ different functionalities, $f_i \mid i = \{1 \ldots n\}$, and existence of $n$ sets, $SU_i$ are assumed. Each $SU_i$consists of software units offering the functionality. For each $f_i$, an architect can select a subset ($E_i$) of the existences, where $Ei$is the software units considered. Thus, the system adapts its behavior in $|E_i|$ ways for each $f_i$. Some of the adaptability metrics used for the software architecture is given as follows:

*3.8.1.1. Absolute Functionality Adaptability Index (AFAI)* defines the number of software units it uses for a given functionality.

*3.8.1.2. Relative Functionality Adaptability Index (RFAI)* defines the number of software units it uses with respect to the units actually offering needed functionality. This adaptability index ($A_i$) communicates with the architect about the adaptability of the functionality. When the RFAI vector values are near unity means that the system is using most of the adaptability which can be offered.

*3.8.1.3. Absolute Software Units Index (ASUI)*defines the total number of software units used by the system. This adaptability index provides a global view of the system size and an overview on the attempt needed to manage the whole software architecture.

*3.8.1.4. Mean Functionality Adaptability Index (MFAI)*defines the mean number of software units utilized per functionality. This metric gives an overview of the mean size and the steps to manage the functionalities.

*3.8.1.5. Mean of Relative Functionality Adaptability Index (MRSAI)* defines the mean of the number of software units utilized for the functionalities relative to the number of units that could be used. It also explains how the functionalities concentrate their adaptability choices in average. This index communicates with the architects about the mean extent of utilization of the potential units for the functionalities. A value close to unity denotes that the present architecture uses almost all the software units available. And a value close to zero denotes that the system can be more adaptable and different architecture alternatives of the same adaptability metric could be created.

*3.8.1.6. Absolute System Adaptability Index (ASAI)* defines the number of software units utilized by the system relative to the number of available software units for the construction of the system. A value close to unity denotes that there are only a few

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

choices available to increase the system adaptability. With respect to MRSAI, this adaptability metric encompasses a global view of the system size relative to its maximum attainable size, but does not forecast the amount of different architectural options the system could attain.

*3.8.1.7. Adaptability and Quality Requirement:*The requirements are for the software architecture are focused and classified as presented in TABLE 3.

*Table 2: Adaptability's Behavior Relative To A Quality Requirement*

| As adaptability increases | Requirement formulated as | |
|---|---|---|
| | *Higher than* | *Lower than* |
| Increase of quality value | Helps | Hurts |
| Decrease of quality value | Hurts | Helps |
| Quality value not affected | No effect | |

The first dimension of TABLE 3, distinguishes the software qualities into three categories:

- Qualities that increase their value when adaptability is increased.
- Qualities that decrease their value when adaptability is increased.
- Qualities that do not rely on adaptability variations.

The second dimension in TABLE III defines how the requirement is computed in terms of the variations in intensity as *higher* or *lower*. The labels *Helps* and *Hurts* are given in TABLE III to obtain the effect of adaptability on quality requirements.

Fig. 8and Fig. 9, gives a general graph for quality requirements that belongs the first *Helps*, i.e. the quality value which increases when there is an increase in adaptability and are computed as *higher than*. X-axis denotes increasing values of the applicable adaptability metric ($A_i$). Y-axis denotes the values for the target quality; $Adapt^{Max}$ is the maximum adaptability value which can be considered by the architect. When the adaptability metric is not explicitly given, then a combination of all the adaptability metrics is considered. The upper bound ($V_{AiU}$) and the lower bound ($V_{AiL}$) on the quality the architecture with adaptability $A_i$ can be reached is estimated.

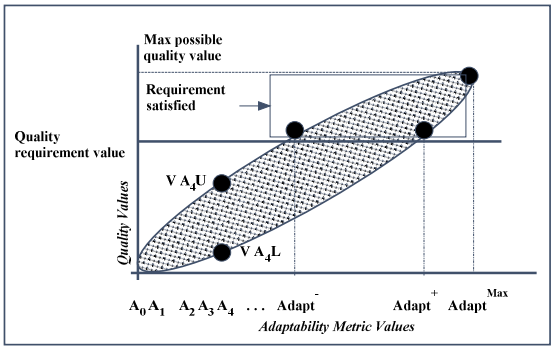



*Fig. 8. Quality in Helps wherethe requirement is higher than.*

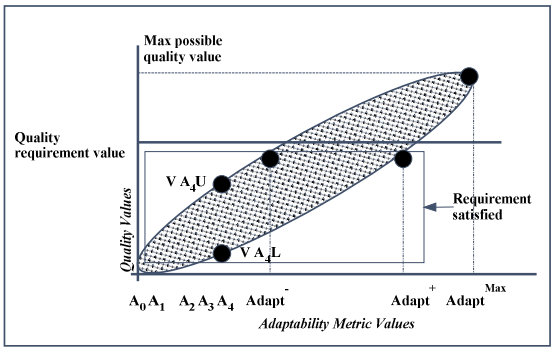



*Fig. 9. Quality in Hurts where the requirement is lower than.*

In Fig. 7, the requirement to satisfy is called *Quality Requirement value*. $Adapt^-$ is the lowest $A_i$ which can fulfill the requirement, and $Adapt^+$ denotes the lowest $A^i$ whose lower bound also fulfills the requirement, which indeed is fulfilled until $Adapt^{Max}$. These values satisfy the requirements: the system should have at least adaptability $Adapt^-$, and, any selected architecture with at least $Adapt^+$ adaptability value will also satisfy it. For intermediate adaptabilities, there will be architectural choices that will fulfill the requirement and others that will not fulfill the requirements.

Fig. 8 shows the graph for requirements belonging to *Hurts* whose quality value increases with adaptability and are computed as lower than. $Adapt^-$ refers to the maximum $A_i$ for whatever architectural option, with such value, for sure fulfills the constraint, and $Adapt^+$ refers to the maximum Ai for which options that fulfill the requirement.

The evaluation of a given constraint or, more generally, the (adaptability vs. property) trade-off study implies to produce the graph of the system under observation. Starting with $A_i = A_0$, $V_{A0U}$, $V_{A0L}$ are computed. Next, the adaptability value is

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

increased and considered the next lowest value $A_i = A_1$ and its $V_{A1U}$ and $V_{A1L}$ are computed. The process goes until $A_i$ reaches $Adapt^{Max}$. These results produce the actual upper and lower bound curves. Depending on where the target quality belongs, the variable $Adapt^-$ is considered as the lowest $A_i$ that can fulfill the requirement or the highest $A_i$ where both $V_{AiU}$ and $V_{AiL}$ fulfill the requirement. Similarly, to compute $Adapt^+$, quality is checked whether it belongs to *Helps* the lowest $A_i$ where both $V_{AiU}$ and $V_{AiL}$ bounds fulfill the constraints, or belonging to *Hurts* the highest $A_i$ that can fulfill the requirement.

The practical calculation of the bounds for a given $A_i$ (i.e., $V_{AiU}$ and $V_{AiL}$), avoid taking into consideration the non-suitable architectural options. This refers that a software unit offering functionality $f_j$ should not be considered in architecture unless another software unit requiring $f_j$ had already been considered.

## 3.9. Construction and Exploitation of Flexibility in Software Architecture

Flexibility in the software architecture is considered during system evolution [18]. An automated flexibility analysis is designed with real-time feedback. A flexibility exploitation analysis is implemented for the purpose of software evolution. This constructs the design time analysis and results in effective utilization of the given flexibility by forming flexibility-aware function-plans. Flexibility is the main attribute that defines the easiness, cost and speed in which the required changes to the software systems could be conducted. A modeling and analysis approach is proposed for the construction and applicationof flexibility throughout the life-cycle in order to increase systematic support for flexibility.

Flexibility is about the response to the future variations of the software. Flexibility is a group of expected potential variations, similar to modifiability. Flexibility scenarios are conditions which define the constraints about the potential changes to the system. The realization of the flexibility scenarios prepares the system for the changes at a later point in time with less effort.

The software architecture estimates with its key design conclusions how many changes are required at the implementation level. The architect has the responsibility to lead design decisions, thereby providing flexibility. The system is modularized in a manner that a specific modification depicted in a flexibility scenario can be performed with minimal local impact alone.

One way to build flexibility is by introduction of *metadata*, which means that specific alterations can be performed in configuration files. Thus, the recompilation in the source code is not required and modifications do not involve programming knowledge.

### 3.9.1. Overview of Flexibility Enhancement:

During the construction of a software system, flexibility is built into the system. There can be variations in the system during evolution, which increases the present flexibility. Fig. 10 shows the construction and exploitation of flexibility. For the construction of flexibility, the scenarios act as the input to the design of architecture. While designing the architecture, an automated feedback is provided about the present degree of flexibility in an architecture modeling tool.

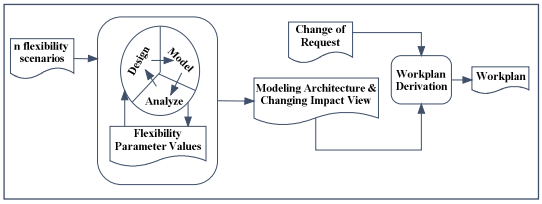



*Fig. 10. Construction and Exploitation of Flexibility.*

A tool-based support for derivation of work plans and analysis of change impacts is provided. Exploiting the flexibility is a development process, which is activated by a strong change request. The exploitation process can be used for rapid identification of the areas of a system which are affected by the saving effort for flexibility exploitation and change request.

### 3.9.2. Construction of Flexibility:

Generally there are two approaches for the optimization of flexibility. First, the architect can be guided to choose more appropriate decisions. Second, the architect can be supported in recognition of sufficient or insufficient architectural decisions, by the measurement of the attained level of characteristics. Practically, architecture design deals with large-scale systems, which makes the impact change analysis manually, a tedious process. Also, additional complexity is established for the architecture model.

The automated analysis of a modeled software architecture relative flexibility and its scenario is not easy. So, a new architectural view known as *Change Impact View* is introduced. Here, the flexibility scenarios are the initial-class modeling entities. Also, an explicit *Impacts*-relationship is introduced between flexibility scenarios and

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

architectural entities. The architect models the impact of each scenario on every impacted entity by adding these details as numbers. During the initial phase of the system design, the architects can come up only with rough estimates of element sizes. A link of the architecture model with the size metrics of the implementation is possible when an implementation already exists.

For the automated flexibility analysis, the metric should be comparable and computable. The impact analysis for every change on an element is estimated. Also, the architecture element's code size is estimated roughly. The flexibility metric is defined according to the function shown in Fig. 11, where 'LoC' refers to the Lines of Code executed. The flexibility of software architecture is defined in the time period [0, 1], where '1' refers for high flexibility and '0' refers low flexibility. The flexibility is unity when the change due to a scenario impacts less than 10 lines of codes. This limit refers a minimal change impact to the present system. Similarly, the flexibility is zero when the change impacts more than 10 lines of code. This limit refers a considerable division of the system which has to be modified to account for the change requirements. The intermediate values of the function are linear. The boundary limits are estimated from the practical experiences and can also be altered by the architect.

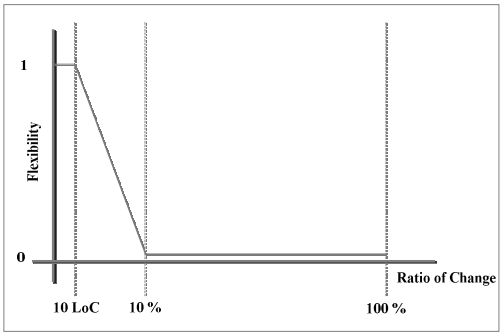



*Fig. 11. Flexibility Metric.*

The automated analysis algorithm can execute the architecture model in near real-time and provide direct feedback to the architect, based on the flexibility-specific enhancements of the architecture model. This flexibility metric can aggregate the flexibility values for multiple scenarios and system parts. The computation of the aggregated flexibility scenarios can be performed by weighing the scenarios so that it does not strongly impact the flexibility results. Accurate flexibility analysis results can be obtained by regarding the persistent and explicit considerations in the architecture model. When a change request was received similar to an already regarded flexibility scenario, the change impact view is utilized to identify how the variation is accounted and it can yield a rough computation of the effort to be spent.

**3.9.3. Realization of Changes**:

The flexibility is established by incorporation of flexibility mechanisms and consideration of flexibility scenarios. During the software evolution, major change requests are needed. Now the provided flexibility has to be exploited. The original modifications of the system relative to the change request have to be organized and planned in a manner that the present flexibility schemes are employed during the implementation.

Flexibility Exploitation Analysis is a semi-automated analysis based on the results in work plans which address the present flexibility explicitly during the architecture construction. The architects *apply* the change requests to the model of the software architecture and *derive* the work functions from the modifications of the model, i.e. the architecture model is altered to denote the target system state for a change request, differences in the earlier software architecture are computed and it is translated into work activities.

The design time approach involves the modeling of flexibility scenario at least for the explicit flexibility schemes in a change impact view. When a change request similar to a modeled flexibility scenario occurs in the change impact view, the architects can alter the software architecture model as suggested by the change impact view. The derived work functions can be denoted in terms of flexibility scheme roles that are affected. The implementation of the changes often leads to work functions from various fields of functions. So, besides the implementation functions other fields of functions during the work planning are also considered.

*3.9.1.1. Preparation of Input Software Architecture Model*:

The software architecture model acts as the primary input for the analysis. For this input, the architects can reuse the architecture model from the design phase. The model of the architecture is enriched in following steps. First, the architects should guarantee that all known flexibility schemes are documented explicitly. This is done by annotating the components with role names of flexibility patterns or styles. Second, the architects utilize the following model annotations to show other fields of activities.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

- All the components are specified by the physical representation of the configuration (meta-data) files and source files.
- All the components are annotated with the number of test cases.
- All the components are specified with the number of deployment nodes and the multiplicities in the deployment model.

This information is furnished only once and then it can be applied to any number of change request analysis.

3.9.3.2. Extension of Change Requests:

A copy is created for every change request. The architects apply the changes to the architecture model so that it contemplates the desired system state after the implementation of the change request. Changes which do not alter the structure of the component structure but only the internal components are highlighted in the architecture model with internal alteration annotations.

The manner in which the change request should be applied in the architecture should be decided. The creative decisions involve decisions which cannot be entirely automated by the tool-support.

3.9.3.3. Computation of Difference and Derivation of Work Plans:

The analysis tool calculates the variations between the base edition and target editions of the software architecture model and converts them into workplan activities.

Some of the alterations mapped to the activities with respect to specific features are:

- Component repository: *Add, Remove, Modify Component; Add, Remove, Modify Interface Port*
- System structure: *Add, Remove, Update Component Instance*
- Component deployment: *Increase, Decrease the count of Deployment Nodes for all Component Instances*
- Flexibility Roles: *Add, Delete a Role to or from Component*

The following work activities are derived from the additional model information:

- When a component is added or altered and there is an annotation of the source code representation, results in *coding activity.*
- The alterations to the parts with a meta-data representation will result in *configuration activity.*
- *Coding activity* will end up in accumulative *build activity.*
- *Coding* and *Configuration activities* will result in accumulative *deployment activities.*
- Test case annotations to various parts will lead to accumulative *test execution activities.*

## 4. FUZZY ASSOCIATION RULE

Various methods to implement the fuzzy association rule have been proposed. FAR is usually applied in databases and statistics. Some of the applications of FAR are in a pattern deduction on Geo-referenced crime [13], EC (Electronic Commerce) environments [33], and stock markets [3]. The FAR is implemented in various mechanisms involving, Multilevel FAR based on Cumulative Distributive Function [25], a quantitative algorithm for generic extraction of FAR [14], multi-level FAR for membership functions [26], and multi-class fuzzy classifiers [32].

### 4.1. Fuzzy Transforms To Detect TheFuzzy Association Rules

Fuzzy transforms are used for the detection of coarse-grained fuzzy association rules (FAR) in the datasets [31]. The FAR are denoted in the means of linguistic expressions. A pre-processing phase evaluates the optimal fuzzy division of the domains of the quantitative features. During the extraction of the FAR, a confidence index (con) and the AprioriGen algorithm is utilized to compute the inverse fuzzy transform. This technique is also used in the data mining process, where a detailed analysis of the FAR is not necessary.

The extraction process of the fuzzy association rules is shown in Fig. 12. The extraction process involves two sub-processes. The AprioriGen algorithm is used for the selection of user FAR and for selecting the predecessors with maximum dimension and support greater than or equal to the threshold $sup_{\varepsilon}$. The AprioriGen algorithm involves two steps: Join step and Pruning step. The join step involves the production and formation of an itemset by 'k' attributes. This is performed by combining two (k-1) attribute having similar first (k-2) attributes. The pruning step involves the removal of all the entities that do not contain all the first (k-1) subsets. The second sub-process involves the extraction of the FAR if the grade of confidence (con) is greater than or equal to the threshold $con_{\varepsilon}$.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

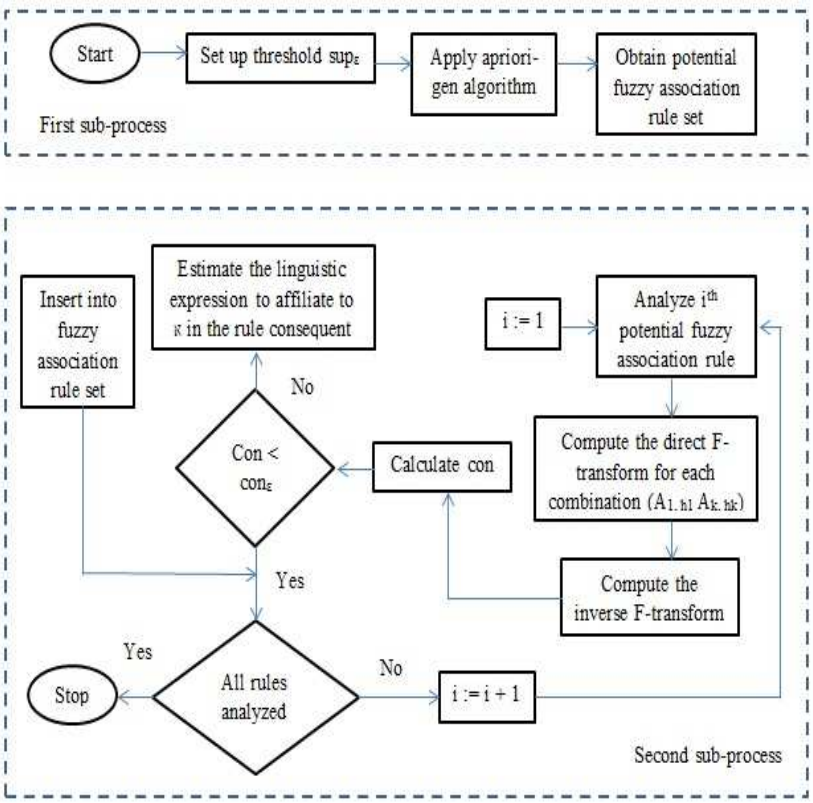



*Fig. 12. Extraction of Fuzzy Association Rules.*

In Fig. 12, $X_1 \ldots X_k$ is the group of data sets collected. $X_1$ is denoted as $A_{1, h1}$ and $X_k$ is denoted as $A_{k, hk}$. $X_z = H (X_1 \ldots X_k)$, where H is a function estimated through a suitable fuzzy partition of the individual attribute domains. The average of $X_z$ is denoted as א.

### 4.2. Enhanced Fuzzy Association Rule

A fuzzy Associative Rule Mining (ARM) algorithm known as FFI_Stream is proposed to deal with statistical values in data streams [15]. Real datasets and synthetic datasets are used for the performance analysis of the system. When compared to discrete methods, this technique establishes a trade-off between the numbers of fuzzy association rules and the efficiency of the system. A time based sliding window model is used for the estimation of the fuzzy association rules in the given input and the clustering technique is used to compute the fuzzy sets.

### 4.3. Fuzzy Association Rule in Financial Data Association

The investment profitability is enhanced by the fuzzy association rules and a decision support system [27]. The investors could cognize the association among the different parameters. After the extraction of the associations, the investors can implement the rules in their decision support systems.

### 4.4. Temporal Fuzzy Association Rule With 2-tuple Linguistic Expression

Fuzzy association rules that have temporal patterns are dealt with the 2-tuple linguistic expression [20]. This method identifies the FAR in a temporal manner while conserving the interpretability of linguistic variables. Iterative Rule Learning (IRL) combined with a Genetic Algorithm (GA) implements the rules and shapes the membership functions. The rules found are differentiated with those from a classical method of finding the FAR.

### 4.5. Unsigned Fuzzy Association Rules with Minimum Supports

Classical algorithms for mining the association rules are constructed on the binary factors of the databases. This has three demerits. First, the algorithm can focus on quantitative attributes. Second, only positive FAR are found in the data mining. Third, the algorithm treats every item with

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

similar frequency although various items may have various frequencies [10]. A discovery algorithm is proposed for mining unsigned fuzzy association rules.

### 4.6. Fuzzy Association Rules Using Genetic Algorithm and 2-tuple Linguistic Expressions

A multi-objective genetic algorithm is proposed for detecting the FAR without any specifications of the minimum confidence and support [28]. This technique involves the extraction of both membership functions and FAR in a single step. Iterative Rule Learning (IRL) is used to cover the uncovered instances.

### 4.7. Fuzzy Association Rules in Low-Quality Data

A data-mining algorithm is proposed to obtain the useful information from the databases with imprecise data [17]. This technique combines the fuzzy apriori algorithm and imprecise data concepts to deduce the useful fuzzy association rules in the given databases.

### 4.8. Axiomatic Fuzzy Set Association Rules

Fuzzy Set Association Rules are proposed for classification issues such as, AFS (Axiomatic Fuzzy Set) theory and AFSRC (AFS association rules for classification) [12]. It is a simple and efficient mechanism which retains the significant rules for classes which are imbalanced. This is performed by fuzzifying class support of a rule. A new model creates the membership functions automatically by executing the available data.

### 4.9. Follow-upMamdani Fuzzy Modeling

Fuzzy logic has exploited for the interpretation of human derived control rules [29]. The linguistic fuzzy rules are used to represent the data in classification and association rule mining. Fuzzy Rules Based Systems (FRBS) are applied in data mining for classification of datasets which are imbalanced.

### 4.10. Fuzzy Association Rules for Discovering Threats to Privacy

This technique focuses on the preservation of privacy in databases through fuzzy rules. The fuzzy rules are based on a decision tree [40]. The predecessor of each created by these systems comprise the details about the released variables (quasi-identifier), whereas the accompanying comprises the data only about the protected variable.

### 4.11. Multilevel Association Rules for Quantitative Data

Data-mining involves the translation of data into useful and organized information [23]. A multi-level fuzzy association rule model is proposed for the extraction of general information stored as quantitative values during transfers. This method uses the various support value at each level and various membership functions for every item. The derivation of large itemsets involves a top-down progressively deepening scheme. Also fuzzy boundaries are utilized instead of the conventional sharp boundary intervals.

### 4.12. Fuzzy Association Rules Based on Equivalence Redundancy of Items

The computational time of the mining process is improved and the extracted repetitive rules are pruned in this method [11]. The equivalence redundancy of the fuzzy itemsets and related theorems are defined. A basic algorithm based on Apriori algorithm is proposed for the extraction of the rules using the equivalence redundancy of fuzzy itemsets.

### 4.13.Fuzzy Association Rule Based Extraction of Frequent Patterns

Frequent itemset is usually located in large transactional databases [16]. A new method for the estimation of the frequent patterns is proposed for the uncertain data. The fuzzy concept is utilized the detection of the recurring patterns. This method can produce large frequent itemsets and then estimate the FAR from the uncertain dataset. The advantage of this technique is that the dataset scanning is performed only once.

## 5. BACKGROUND WORK

### 5.1. Software Architecture for Online Services

The web-services are increasing drastically, with their functions and memory consumption. An efficient software architecture is proposed for web-services like social networking, search and geo-location services [4]. Social networks like Twitter have risen as an efficient open social network among all communities. A spatial and semantic analysis of the social network data is analyzed. A functional diversity approach is proposed to enhance the fault tolerance for the data collection, where its performance is evaluated. This software architecture enables the spatial patterns inside the geo-locations and can give the user with resourceful unpredictable elements.

The software system is able to collect a huge amount of geo-located Twitter data online and

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

provide different search functionalities on the database. The architecture is constructed on the assumption that the real time condition is reduced which allows more advanced semantic analysis. The software architecture is built on Python-based web framework Django. The software system is combined with Apache Lucene / Solr system. The architecture comprises a highly efficient inverted index representation. The semantic aspect of the search needs to be considered carefully by the following strategy. First, an adaptable Slang database is combined into the original system which permits the system to replace correct English phrases to the detected slangs in tweets. Second, a semantic similarity based short path and a wordnet lexical database is used to expand the actual query to synonyms, hypernyms and hyponyms of every query word. Finally, by employing advanced Lucene's query handling, phrase and word based queries with different logical conditions are enabled efficiently. A fault tolerant system is implemented by using individual machines simultaneously for collection and then merging the databases.

### 5.2. Software Architecture Evolution

Software evolvability is the software system's capability for change to future events [35].This results in better economic value of the software. For long-term systems, evolvability is required explicitly during the whole software lifecycle to have a good productive lifetime of software systems.

The software lifecycle studies can be based on five sub-categories:

#### 5.2.1. Quality Considerations during Design:

The approaches for the assessment of the software lifecycle can be based on conditions like quality attribute focused on the requirement, influencing factors and scenario.

#### 5.2.2. Evaluation of Architectural Quality:

The approaches for the evaluation of the architectural quality can be experience-based, metric-based and scenario-based.

#### 5.2.3. Economic Valuation:

These approaches enlarge the information on the architectural conclusions' business consequences, and aid development teams in selecting among architectural options.

#### 5.2.4. Architecture Knowledge Management:

These approaches enhance the architectural integrity by improving architecture documentation by the extraction of architectural knowledge from different information sources.

#### 5.2.5. Models for Software Evolution:

These techniques function as modeling software artifacts providing traceability and visualization of the impact of software architecture artifacts' evolution.

### 5.3. Software Architecture Analysis for Large-Scale Distributed Systems

Large-scale distributed systems involve substantial investment and high risk [1]. Some of the former architectural decisions describe how the system is organized in terms of permanent data communication, coarse-grained modularization, data management, data I/O and allocation. Such an organization's mainframe is known as System Organization Pattern. Analysis of the software architecture early in the development cycle identifies the significant technical risks and avoids them at minimal cost. But, architecture analysis methods such as the Architecture Trade-off Analysis Method (ATAM) cannot be applied very early in conceptually designed architecture, as the influence of System Organization Pattern on the minute details of the final system cannot be accurately defined. So, the Early Architecture Evaluation Method (EAEM) is developed to estimate the System Organization Pattern before an ATAM-based estimation would be feasible. The architecture evaluation works upon the Goal-Question-Metric scheme. It recognizes the substantial risks faced by the architectural decisions involving the System Organization Pattern. The EAEM depends on the existing quality scenario-directed architecture analysis methods.

### 5.4. Concern-based Software Architecture for Groupware Systems

This software architecture analysis involves collaboration and interaction analysis, which permits the study and characterization of the collaborative functions by the groupware system users [30]. The automation of the collaboration and interaction analysis endows the assessment of the users' work and improvement of groupware system behavior and support. A concern-based architecture is proposed for the groupware development as a model for the integration of analysis subsystems into groupware systems. The whole groupware system can be represented by a group of subsystems (Meta-information, Analysis, Identification, Application and Awareness). The subsystems communicate with each other to substantiate the basic functions and analyze the collaborative work of the users. The basic functions are user management and support for collaborative functions within the shared workspaces. This architecture is designed for the COLLECE groupware system,

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

which sustains the collaborative programming principles.

### 5.5. Combining Software Architecture with Team Structure in Open Source Software Development

The integration of the developer team structure and open source software (OSS) architecture monitors the socio-technical interactions in a system development [6]. A high level of structural interdependency combined with larger teams result in better project performance.

### 5.6. Session Reliability Analysis of Web Systems Under Heavy-tailed Workloads

A framework for modeling the session reliability has been proposed by integration of user view and system view [8]. The user view is defined by the session layer, whereas the system view is defined by the service layer. The session layer can handle heavy tailed workloads which exist in real Web systems. The service layer concentrates on the observed request reliability from the service provider's view. Multi-web server architecture and the manner in which the components interact to service the requests are considered. The reliability of the requests is estimated by cognizing the individual component reliabilities. The manner in which the components interact with each other is defined by the software architecture.

### 5.7. Enhancing Software Adaptability through Reflection Programming

The development of software is analyzed from the programming aspect and it is deemed that it is the period of adaptive programming at present [9]. Reflex technology for enhancing the software adaptability. The self-reflection and meditation a principle for encoding the execution state as code known as *reification*. The reflection programming enables the automatic execution of the software through self-reflection.

### 5.8. Variability Analysis in Software Architecture

Variability is an important parameter in the context of software architecture [47]. Besides the idea of product lines, the issue of variability in the software domain is examined. In this survey, a study is conducted among various subjects in terms of variability. It is observed that there is no common apprehension of "variability" in the view of software architecture. It is also noted that some challenges in the variability of the software architecture also exist in the product line domain. Some of the challenges with variability in the software architecture are complexity, formality and management.

### 5.9. Improving the Deployment Architecture of Software in Distributed Systems

The allocation of the software components of the respective hardware nodes has a significant impact on the QoS of distributed software systems (DSS) [21]. For a specified system, the deployment architectureprovides the same functionalities but different QoS levels. The parameters which influence the deployment architecture's quality are often not cognized before the system's initial deployment and may alter at runtime. This necessitates the redeployment of the software to enhance the QoS of the system. A framework is evaluated in determining the most suitable deployment architecture for a DSS relative to multiple confusing QoS dimensions. The framework bolsters a formal modeling and a group of adaptive algorithms for enhancing the system's deployment.

A Deployment Improvement Framework (DIF) is proposed to enhance the software-intensive system's QoS by determining the appropriate deployment of the system's software parts onto its hardware hosts. DIF permits quick, quantitative exploration of a system's deployment space. The framework model's design and the algorithms permit the random specification of new QoS dimensions and their enhancements. The data about the system parameters are either obtained at design-time or at run-time and an enhanced deployment architecture is computed and applied.

### 5.10. Reliability Oriented Software Evolution Based on Contribution Degree of Component

The reliability of the software architecture is enhanced by the analysis contribution degree of component [24]. The various components of the system serve various roles in the reliability-oriented software architecture evolution. The reliability-oriented evolution technique upon the contribution degree of component is employed in an ATM system.

### 5.11.Fuzzy Rules for Time-Series Data

Time series analysis is used frequently in various applications [34]. Several data mining techniques have concentrated only on binary-valued data, but time series data are quantitative values. A fuzzy mining scheme is proposed to estimate the linguistic association rules. This method uses a sliding window to produce continuous subsequences from a specified time series. The

Journal of Theoretical and Applied Information Technology
31st January 2014. Vol. 59 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

fuzzy itemsets are analyzed from the subsequences and relevant post-processing is performed to exclude the repetitive patterns.

## 6. DISCUSSION AND RESULTS

### 6.1. Software Architecture Analysis for an ATM Banking System Based on Fuzzy Association Rules

The software architecture for an ATM banking system is implemented in NetBeans IDE, working upon the principle of Fuzzy Association Rules (FAR). The software architecture is analyzed in terms of reliability, performance, security, flexibility, and adaptability. The analysis result is shown in Fig. 13.

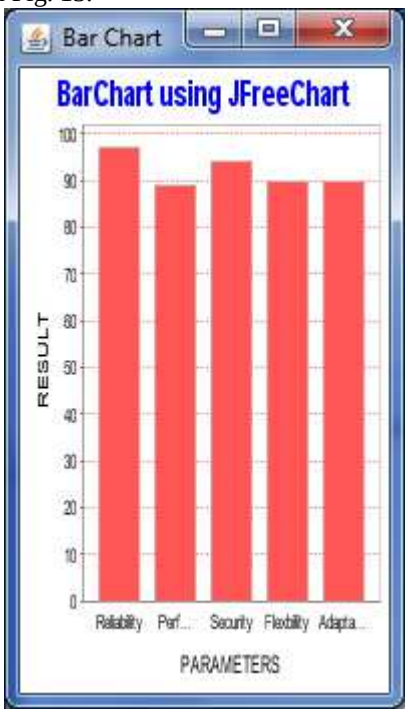



*Fig. 13. Software Architecture Analysis Results of Proposed ATM Banking System.*

The threshold values for the individual parameters are initiated using FAR as $Reliability_{TH}$ = 90%, $Flexibility_{TH}$ = 80%, $Performance_{TH}$ = 95%, $Security_{TH}$ = 95%, and $Adaptability_{TH}$= 98%. The architectural analysis with respect to the threshold values classifies the ATM banking system software architecture with the following results:

- High reliability
- High security
- Low adaptability
- High flexibility
- High performance

### 6.2. Security Analysis of Software Architecture Based on Analytical Hierarchy Process (AHP)

The security analysis is done for a flower shop system. Security vulnerabilities such as Cross Site Scripting (XSS) and SQL injection were deliberately implemented in the less secure version of the system. A MySql server and an Apache HTTP server were configured as a database server and web server respectively. The security analysis is performed by estimating the security risk. The risk values for various scenarios of different Eigenvectors and Eigen scores are given in Fig. 14.

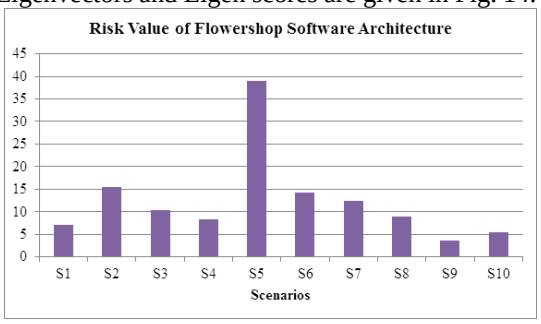


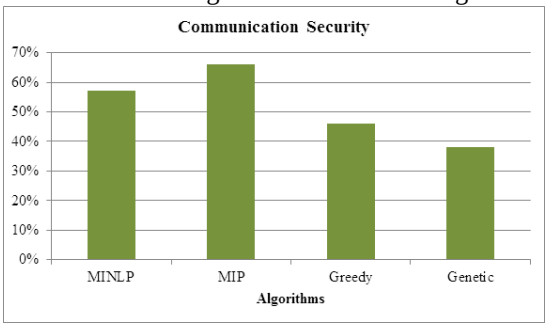

*Fig. 14. Comparison of Risk Values of Flower Shop Software Architecture.*

### 6.3. Construction and Exploitation of Flexibility in Software Architecture

An Airline Reservation System is taken as the example software architecture for the flexibility analysis [18]. Four scenarios are considered and the overall flexibility metric for these scenarios is 0.4.

### 6.4. Improving the Deployment Architecture of Software in Distributed Systems

Several algorithms like Mixed-Integer Nonlinear Programing (MINLP) algorithm, Mixed Integer linear Programming (MIP), Genetic algorithm and Greedy algorithm are considered in the security analysis in the distributed systems' software architecture with 12 components, 8 users, 8 services, and 5 hosts [21]. The comparison of the average security analysis among eight various services for these algorithms is shown in Fig. 15.



ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

*Fig. 15. Comparison of Communication Security for Various Algorithms in a Distributed System.*

### 6.5. Reliability Oriented Software Evolution Based on Contribution Degree of Component

The reliability analysis is performed for an ATM Banking System. Several scenarios such as Account Manager (R6), Helper (R7), Transactor (R9), Verifier (R10) and Messenger (R8) are used[24]. The reliability of two scenarios R6 and R7 is analyzed and shown in Fig.16. The improved software architecture model provided a reliability of 54.51%. After software evolution the reliability of the system increased to 85.12%, which is a 56% increase.

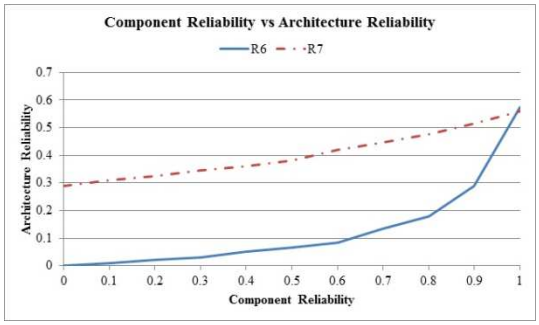



*Fig. 16. Comparison of Architecture Reliability for Two Scenarios Provided With Component Reliability.*

### 6.6. Reliability Evaluation of Software Architecture under Uncertainty

A Monte Carlo (MC) simulation process has been considered in the analysis of reliability in the software architecture under uncertainty [43]. The analysis is based on 3000 MC trails. The histogram of the reliability samples is computed and is shown in Fig. 17.

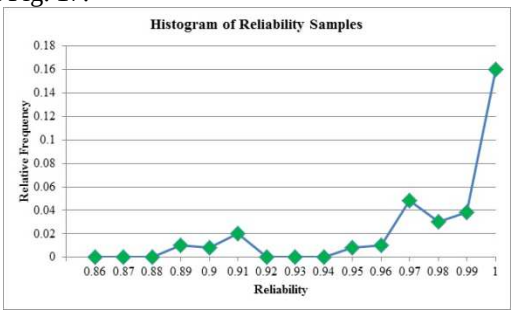



*Fig. 17. Histogram of Reliability Samples.*

### 6.7. Overall Comparison of Proposed Software Architecture With Other Software Architectures

The proposed ATM banking software system architecture is analyzed with various existing software architectures [18], [21], [24], [54], [55] in terms of reliability, flexibility, performance, security, and adaptability. The comparison result is given in Fig. 18.

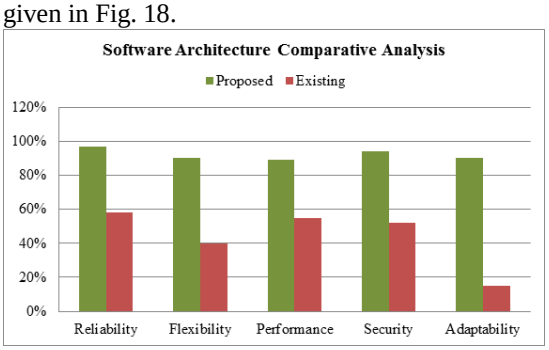



*Fig. 18.* Software Architecture Comparative Analysis.

## 7. 7. CONCLUSION

The performance characteristics of the software architecture have been analyzed in terms of reliability, flexibility, adaptability and security. A software architecture is proposed for an ATM banking system based on Fuzzy Association Rule (FAR). Various techniques concentrating on individual performance parameters can be integrated for the efficient analysis and design of the software architecture. The various techniques to implement the fuzzy association rule have been discussed. The results of the software architecture analysis with respect to security, component reliability, architecture reliability, adaptability and risk value has been presented.The proposed software architecture is efficient compared to the existing software architectures in terms of reliability, flexibility, performance, security, and adaptability. The future of software architecture analysis involveshybrid Architecture Trade-off Analysis Method (ATAM), hybrid Architecture-Level Modifiability Analysis (ALMA), and software evolution which is an advancement to SAAM.

## REFERENCES:

[1] A. Zalewski and S. Kijas, "Beyond ATAM: Early architecture evaluation method for large-scale distributed systems," *Journal of Systems and Software*, vol. 86, pp. 683-697, 2013.

[2] P. Potena, "Optimization of adaptation plans for a service-oriented architecture with cost, reliability, availability and performance tradeoff," *Journal of Systems and Software*, vol. 86, pp. 624-648, 2013.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[3] P. Paranjape-Voditel and U. Deshpande, "A stock market portfolio recommender system based on association rule mining," *Applied Soft Computing*, vol. 13, pp. 1055-1063, 2013.

[4] M. Oussalah, *et al.*, "A software architecture for Twitter collection, search and geolocation services," *Knowledge-Based Systems*, vol. 37, pp. 105-120, 2013.

[5] N. Niu, *et al.*, "Enterprise Information Systems Architecture - Analysis and Evaluation," *Industrial Informatics, IEEE Transactions on*, vol. PP, pp. 1-1, 2013.

[6] N. Nan and S. Kumar, "Joint Effect of Team Structure and Software Architecture in Open Source Software Development," *Engineering Management, IEEE Transactions on*, vol. PP, pp. 1-12, 2013.

[7] A. Marback, *et al.*, "A threat model-based approach to security testing," *Software: Practice and Experience*, vol. 43, pp. 241-258, 2013.

[8] N. Janevski and K. Goseva-Popstojanova, "Session Reliability of Web Systems Under Heavy-Tailed Workloads: An Approach based on Design and Analysis of Experiments," *Software Engineering, IEEE Transactions on*, vol. PP, pp. 1-1, 2013.

[9] W. Yudong and X. Xinjun, "The Analysis and Reflection of Software Adaptability and Its Supported Technical," in *Computer Science and Electronics Engineering (ICCSEE), 2012 International Conference on*, 2012, pp. 635-638.

[10] O. Weimin, "Mining positive and negative fuzzy association rules with multiple minimum supports," in *Systems and Informatics (ICSAI), 2012 International Conference on*, 2012, pp. 2242-2246.

[11] T. Watanabe and R. Fujioka, "Fuzzy association rules mining algorithm based on equivalence redundancy of items," in *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, 2012, pp. 1960-1965.

[12] X. Wang, *et al.*, "Mining axiomatic fuzzy set association rules for classification problems," *European Journal of Operational Research*, vol. 218, pp. 202-210, 2012.

[13] R. Sridhar, *et al.*, "Analysis and Pattern Deduction on Linguistic, Numeric Based Mean and Fuzzy Association Rule Algorithm on Any Geo-referenced Crime Point Data Integrated with Google Map," in *Proceedings of the International Conference on Soft Computing for Problem Solving (SocProS 2011) December 20-22, 2011*. vol. 131, K. Deep, *et al.*, Eds., ed: Springer India, 2012, pp. 15-27.

[14] I. B. A. Sougui, *et al.*, "A quantitative algorithm for extracting generic basis of fuzzy association rules," in *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*, 2012, pp. 23-27.

[15] L. Shen and S. Liu, "A New Fuzzy Association Rules Mining in Data Streams," in *Advanced Technology in Teaching - Proceedings of the 2009 3rd International Conference on Teaching and Computational Science (WTCS 2009)*. vol. 117, Y. Wu, Ed., ed: Springer Berlin Heidelberg, 2012, pp. 163-172.

[16] D. S. Rajput, *et al.*, "Fuzzy association rule mining based frequent pattern extraction from uncertain data," in *Information and Communication Technologies (WICT), 2012 World Congress on*, 2012, pp. 709-714.

[17] A. M. Palacios, *et al.*, "Mining fuzzy association rules from low-quality data," *Soft Computing*, vol. 16, pp. 883-901, 2012/05/01 2012.

[18] M. Naab and J. Stammel, "Architectural flexibility in a software-system's life-cycle: systematic construction and exploitation of flexibility," presented at the Proceedings of the 8th international ACM SIGSOFT conference on Quality of Software Architectures, Bertinoro, Italy, 2012.

[19] M. Meitner and F. Saglietti, "Software Reliability Testing Covering Subsystem Interactions," in *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*. vol. 7201, J. Schmitt, Ed., ed: Springer Berlin Heidelberg, 2012, pp. 46-60.

[20] S. G. Matthews, *et al.*, "Temporal fuzzy association rule mining with 2-tuple linguistic representation," in *Fuzzy Systems (FUZZ-IEEE), 2012 IEEE International Conference on*, 2012, pp. 1-8.

[21] S. Malek, *et al.*, "An Extensible Framework for Improving a Distributed Software System's Deployment Architecture," *Software Engineering, IEEE Transactions on*, vol. 38, pp. 73-100, 2012.

[22] A. Koziolek, "Research Preview: Prioritizing Quality Requirements Based on Software Architecture Evaluation Feedback," in

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

*Requirements Engineering: Foundation for Software Quality*. vol. 7195, B. Regnell and D. Damian, Eds., ed: Springer Berlin Heidelberg, 2012, pp. 52-58.

[23] A. M. N. Kousari, *et al.*, "Improvement of Mining Fuzzy Multiple-Level Association Rules from Quantitative Data," *Journal of Software Engineering and Applications*, vol. 5, pp. 190-199, 2012.

[24] W. Jun and C. WeiRu, "A Reliability-oriented Evolution Method of Software Architecture Based on Contribution Degree of Component," *Journal of Software (1796217X)*, vol. 7, pp. 1744-1750, 2012.

[25] C. Jr-Shian, *et al.*, "Enhance the Multi-level Fuzzy Association Rules Based on Cumulative Probability Distribution Approach," in *Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), 2012 13th ACIS International Conference on*, 2012, pp. 89-94.

[26] T.-P. Hong, *et al.*, "A multi-level ant-colony mining algorithm for membership functions," *Information Sciences*, vol. 182, pp. 3-14, 2012.

[27] G. T. S. Ho, *et al.*, "Using a fuzzy association rule mining approach to identify the financial data association," *Expert Systems with Applications*, vol. 39, pp. 9054-9063, 2012.

[28] H. L. Ghazi and M. S. Abadeh, "Mining fuzzy association rules with 2-tuple linguistic terms in stock market data by using genetic algorithm," in *Artificial Intelligence and Signal Processing (AISP), 2012 16th CSI International Symposium on*, 2012, pp. 354-359.

[29] A. Fernández and F. Herrera, "Linguistic Fuzzy Rules in Data Mining: Follow-Up Mamdani Fuzzy Modeling Principle," in *Combining Experimentation and Theory*. vol. 271, E. Trillas, *et al.*, Eds., ed: Springer Berlin Heidelberg, 2012, pp. 103-122.

[30] R. Duque, *et al.*, "Integration of collaboration and interaction analysis mechanisms in a concern-based architecture for groupware systems," *Science of Computer Programming*, vol. 77, pp. 29-45, 2012.

[31] F. Di Martino and S. Sessa, "Detection of Fuzzy Association Rules by Fuzzy Transforms," *Advances in Fuzzy Systems*, vol. 2012, p. 12, 2012.

[32] E. D'Andrea and B. Lazzerini, "A hierarchical approach to multi-class fuzzy classifiers," *Expert Systems with Applications*, 2012.

[33] H.-P. Chiu, *et al.*, "Applying cluster-based fuzzy association rules mining framework into EC environment," *Applied Soft Computing*, vol. 12, pp. 2114-2122, 2012.

[34] C.-H. Chen, *et al.*, "Fuzzy data mining for time-series data," *Applied Soft Computing*, vol. 12, pp. 536-542, 2012.

[35] H. P. Breivold, *et al.*, "A systematic review of software architecture evolution research," *Information and Software Technology*, vol. 54, pp. 16-40, 2012.

[36] A. Asthana and K. Okumoto, "Integrative Software Design for Reliability: Beyond Models and Defect Prediction," *Bell Labs Technical Journal*, vol. 17, pp. 37-59, 2012.

[37] Andr, *et al.*, "Kieker: a framework for application performance monitoring and dynamic software analysis," presented at the Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering, Boston, Massachusetts, USA, 2012.

[38] B. A. Akinnuwesi, *et al.*, "A framework for user-centric model for evaluating the performance of distributed software system architecture," *Expert Systems with Applications*, vol. 39, pp. 9323-9339, 2012.

[39] F. Zeshan and R. Mohamad, "Software architecture reliability prediction models: An overview," in *Software Engineering (MySEC), 2011 5th Malaysian Conference in*, 2011, pp. 119-123.

[40] L. Troiano, *et al.*, "Interpretability of fuzzy association rules as means of discovering threats to privacy," *International Journal of Computer Mathematics*, vol. 89, pp. 325-333, 2012/02/01 2011.

[41] L. Tan and A. Krings, "An Adaptive N-Variant Software Architecture for Multi-Core Platforms: Models and Performance Analysis," in *Computational Science and Its Applications - ICCSA 2011*. vol. 6783, B. Murgante, *et al.*, Eds., ed: Springer Berlin Heidelberg, 2011, pp. 490-505.

[42] D. Perez-Palacin, *et al.*, "Software architecture adaptability metrics for QoS-based self-adaptation," presented at the Proceedings of the joint ACM SIGSOFT conference -- QoSA and ACM SIGSOFT symposium -- ISARCS on Quality of software architectures -- QoSA and

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

architecting critical systems -- ISARCS, Boulder, Colorado, USA, 2011.

[43] I. Meedeniya, *et al.*, "Architecture-based reliability evaluation under uncertainty," presented at the Proceedings of the joint ACM SIGSOFT conference -- QoSA and ACM SIGSOFT symposium -- ISARCS on Quality of software architectures -- QoSA and architecting critical systems -- ISARCS, Boulder, Colorado, USA, 2011.

[44] R. Kumar, *et al.*, "Measuring software reliability: a fuzzy model," *SIGSOFT Softw. Eng. Notes*, vol. 36, pp. 1-6, 2011.

[45] S. S. Jalali, *et al.*, "A New Approach to Evaluate Performance of Component-Based Software Architecture," in *Computer Modeling and Simulation (EMS), 2011 Fifth UKSim European Symposium on*, 2011, pp. 451-456.

[46] O. Georgieva and A. Dimov, "Software reliability assessment via fuzzy logic model," presented at the Proceedings of the 12th International Conference on Computer Systems and Technologies, Vienna, Austria, 2011.

[47] M. Galster and P. Avgeriou, "The notion of variability in software architecture: results from a preliminary exploratory study," presented at the Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems, Namur, Belgium, 2011.

[48] D. Falessi, *et al.*, "Decision-making techniques for software architecture design: A comparative survey," *ACM Comput. Surv.*, vol. 43, pp. 1-28, 2011.

[49] V. Cortellessa, *et al.*, "What Is Software Performance?," in *Model-Based Software Performance Analysis*, ed: Springer Berlin Heidelberg, 2011, pp. 1-7.

[50] V. Cortellessa, *et al.*, "Software Lifecycle and Performance Analysis," in *Model-Based Software Performance Analysis*, ed: Springer Berlin Heidelberg, 2011, pp. 65-77.

[51] M. Bunke and K. Sohr, "An Architecture-Centric Approach to Detecting Security Patterns in Software," in *Engineering Secure Software and Systems*. vol. 6542, Ú. Erlingsson, *et al.*, Eds., ed: Springer Berlin Heidelberg, 2011, pp. 156-166.

[52] F. Brosch, *et al.*, "Reliability prediction for fault-tolerant software architectures," presented at the Proceedings of the joint ACM SIGSOFT conference -- QoSA and ACM SIGSOFT symposium -- ISARCS on Quality of software architectures -- QoSA and architecting critical systems -- ISARCS, Boulder, Colorado, USA, 2011.

[53] A. Alkussayer and W. H. Allen, "Security risk analysis of software architecture based on AHP," in *Networked Computing (INC), 2011 The 7th International Conference on*, 2011, pp. 60-67.

[54] N. Subramanian and L. Chung, "Metrics for software adaptability," *Proc. Software Quality Management (SQM 2001), April*, 2001.

[55] M. Frigo and S. G. Johnson, "FFTW: an adaptive software architecture for the FFT," in *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, 1998, pp. 1381-1384 vol.3.