International Journal of Innovative Research in Computer Science & Technology (IJIRCST)
ISSN: 2347-5552, Volume-9, Issue-1, January 2021
https://doi.org/10.21276/ijircst.2021.9.1.3
Article ID D10963, Pages 15-19
www.ijircst.org

A Framework for Modeling Non-Functional Requirements for Business-Critical Systems

Sameer S Paradkar

Enterprise Architect, ATOS, Architecture Group, Business & Platform Solutions, Mumbai, India

Correspondence should be addressed to Sameer S Paradkar; sameer.paradkar@atos.net

ABSTRACT- Proper definition and implementation of NFRs is critical. In case they are Over-specify, then the solution may be too costly to be viable; in case they are underspecified or underachieve them, the system will be inadequate for its intended use. An adaptive and incremental approach to exploring, defining, and implementing NFRs is vital for the successful delivering of NFRs. NFRs are not product backlog items. The constraints on development that limit degree of design freedom while building system. These constraints are articulated in the acceptance criteria for multiple product backlog items. For e.g., SAML-based Single Sign-on - SSO is a requirement for the product. SSO is a functional requirement, while SAML is the constraint. In that sense, any backlog item building sign-on functionality would reference the SAML constraint in its acceptance criteria. The sections that follow describe the phases of NFR journey from discovery to deployment for a large complex business critical systems including the NFR modelling framework.

KEYWORDS- NFR-Non-Functional Requirements, NFR Framework, NFR Modelling, NFR Methodology

I. INTRODUCTION

In terms of the lifecycle of NFRs, this paper outlines an NFR framework that describes all activities starting from elicitation, discovery, requirement definition, architecture and design, implementation, monitoring, QA, sizing and trade-off analysis of NFRs. In particular, business constraints identified during business analysis are used as input and refined further to define nonfunctional requirements during requirements definition in subsequent phases. NFRs are one of the key aspects to derive a comparison among competing software systems. NFRs such as performance, reliability, maintainability, security, accuracy, etc. and this has to be handled at the early stage of software development along with the functional requirements. Eliciting NFRs is considered one of the challenging aspects in requirement analysis. Although there are well-developed techniques for eliciting functional requirements, there is a lack of elicitation methodology for NFR and there is limited consensus pertaining to the techniques for NFR. In the software development life cycle, requirement elicitation is one of the most knowledge-intensive activities. Therefore, the elicitation technique has to be designed in manner that it will interact closely with the stakeholders. A major challenge of NFR is the trade-off analysis and sizing of NFRs. This paper proposes a framework approach that can be leveraged for all critical phases of NFRs starting from discovery to deployment.

II. RELATED WORK

[1] This Paper focuses on Non-Functional Requirements (NFR) for IT and IT-enabled business services and proposes the creation of enterprise architecture artifacts specifically addressing NFR.

[2] This in-progress paper, we have elicited NFRs with a systematic approach of a system which is Point of Sale system.

[3] This paper proposes an approach to model non-functional requirements in telecommunication systems and the identification of issues to be considered in modelling those requirements.

[4] This paper provides a model-based approach for enabling automatic software management in dynamic systems. By modeling nonfunctional requirements and capabilities we can determine valid configurations and provide a basis for reconfiguration and optimization of configurations.

[5] In this paper, suggests an extension to BPMN technique allowing the business constraints and NFRs to be modeled during the early requirements engineering phase.

[6] This paper presents the modelling and analysis of response time performance aspects leveraging the formal design analysis framework – FDAF.

[7] In this paper presents a novel approach for specifying non-functional requirements as constraint systems over the space of models. The approach, is based on structured programming and allows requirements to be specified independently from each other and elicitated together.

[8] The research aims to promote the use of existing NFR models and integrate them into the early phases of the software life cycle in a systematic way to overcome the stated limitations of the existing research.

[9] This paper presents the NoFun language for stating system quality in the framework of the ISO-IEC quality standards.

[10] This paper proposes a Control Case approach to record and model nonfunctional requirements.

III. NFR MODELLING – PROBLEM CONTEXT

Traditional IT systems largely deal with transaction loads originating from within the organization, accommodating demands generated by hundreds or perhaps thousands of users. The evolutionary expansion due to the internet has placed a heavy demand on non-functional requirements for business critical solutions. While the expected response times, reliability, and availability may have remained largely constant, the magnitude of users has increased significantly; from several thousand into millions of users. In view of the growing transactional demands on business critical solutions, the nonfunctional aspects of an architecture are increasingly important, motivating the need to manage these requirements throughout the life cycle of systems development. The rate of project failure increases due to insufficient NFR gathering at the proper stage. NFRs are to be treated as the constraints of the system which are needed to satisfy the customers. In many cases customers' expectations are un-fulfilled due of the inadequacy of the business system properties. The time and cost for software development can be brought down by giving critical importance to NFRs. Typically, customers may not know the constraints of system in the early stage of the development process. In a complex system, NFRs are vital and critical. The system can be at risk if NFRs are neglected during the system development. As the complexity of software is ever increasing and customers are focusing critically on the quality of software, NFR is no longer a secondary option in the requirements management process.

IV. NFR MODELLING - METHODOLOGY

NFRs are the constraints on the system and these constraints are for development and deployment processes. The quality requirements are also known as NFRs. These typically include availability, performance, reliability, usability, modifiability, performance, security, flexibility, etc. While functional requirements are gathered at an early stage, ignorance of NFRs leads to project failure. A common problem is that often stakeholders are unaware of the system NFR requirements. Although there are standard definitions of functional requirements, there is a lack of well-formed definition of NFR. To formally specify and characterize the NFRs are very much harder, because NRFs vary in different circumstances. Sometimes both functional and NFRs are mixed up and ambiguity arises differentiating between them. Since NFRs are linked to functional requirements, they create conflicts among stakeholders, but the later will increase the cost of the system which is associated with NFRs. For the lack of domain knowledge, one will not get adequate NFRs, besides it is not even certain which NFR will be taken into consideration. NFR is not equally considered as functional requirement in software development [1,2,3,6,8]. The next paragraph describes the different stages of NFRs Life Cycle. Interms of the additional deep dive details refer to table NFR framework table that follows this section.

A. Discovery (Elicitation)

As a first step, it is important to define the hierarchy of the NFRs - Non-Functional Requirements and arrive at a consolidated catalogue of the NFR, and **FURPS+ methodology** is one of the main frameworks that is widely leveraged in the software industry for the NFR definition: **FURPS+ stands for Functionality, Usability, Reliability, Performance, Supportability and the Constraints (for the "+").** There are different methodologies for elucidating nonfunctional and the NFRs KPIs are derived from **business users' goals, industry trends, competitive analysis, and legacy systems constrains** [2,7]. E.g., Availability 99.99%, Throughput 2 milli-seconds. On the other hand, it is also equally important to identify the major risks that may undermine those business goals. The risks stem from the business constraints, which manifest in various operating conditions in the business process has the context.

B. Architecture & Design

Architecture for NFRs involves the refinement of NRFs and the development of architecture specifications. NFR related use-case scenarios and use-cases are the key outputs of this phase. Architecture for NFRs covers the evaluation and selection of a set of building blocks and design patterns that will implement the set of non-functional requirements. Architecting for non-functional requirements includes the process of refining non-functional requirements and mapping these nonfunctional requirements to specific architecture building blocks [6,7,8]. This also involves fixing the baseline without ambiguity and reviewing of **Architecture Building Block – ABBS** that are influenced by NFR. The next steps are to document the rationale of the architectural decisions and trade-offs **Set-Based Design - SBD** is a practice that keeps requirements and design options flexible for as long as possible during the development process. Instead of choosing a single point solution, SBD methodology identifies and simultaneously explores multiple options, eliminating weaking choices over time. This increases the flexibility in there of the design process by committing to technical solutions after validating the assumptions, and hence produces better outcomes. Applying SBD can keep options open by initially specifying NFRs as a range, e.g., 99.98%, 99.999%. Teams can explore the solution space and gain additional knowledge that leads to an optimum decision.

C. Engineering (Implementation)

In terms of implementation approaches, many NFRs prescribe additional work that needs to be done either now or in the future to satisfy them. At times the NFR must be implemented all at once, or at times the teams will have to take a more incremental approach. Various parameters are considered for the trade-offs and these impact the NFR implementation approach. There also facilitate prioritizing the NFRs. Five considerations that affect NFRs trade-off decisions:

- **Development expense** – the cost of labour and materials required to implement a capability
- **Lead time** – time needed to implement the capability
- **Product cost** – the manufacturing cost or deployment and operational costs
- **Value** – the business-worth of the capability to the business and customer
- **Risk** – the un-certainty of the solution's technical or business success

Implementing design patterns allows early optimizing the code and implementing best practices to address performance, availability, and security which are the main QA parameters, but this could be extended to other NFRs [6]. NFR Implementation should be planned in a way that allows several iterations to ensure the right level of NFR. The different alternatives that are leveraged for implement the NFRs are:

- **All at Once** – Few NFRs will require immediate implementation. For example, any regulatory requirements require customer to respond within the specified time constraints or risk being in violation.
- **Incremental story-by-story**: The teams have options. For e.g., the need for improved performance can be addressed over time, one story at a time, as the figure below illustrates.

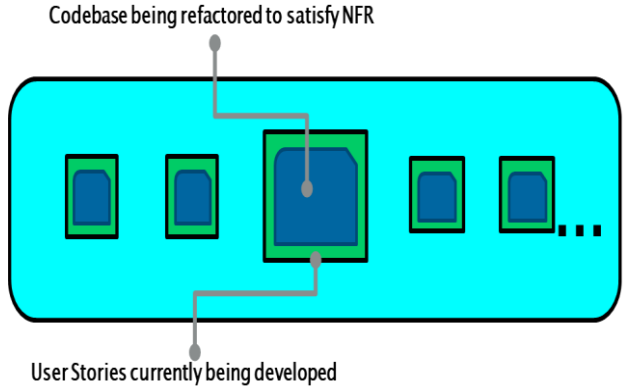

Fig 1: Implementing NFRs

D. Monitoring & Instrumentation

Monitoring tools can help perform NFR monitoring and reporting. Measurable and reportable metrics that map to the NFRs are key for NFR monitoring. Non-functional requirements themselves can be performance targets for certain key metrics. For example, **Mean Time to Restore Service (MTTRS)** is a key metric for availability and recoverability and the number and percent of times the MTTRS is met or exceeded for e.g. email as a service becomes a measurable and reportable nonfunctional requirement metric for e-mail. Measurable and reportable metrics that map to the non-functional requirements are key for non-functional requirements monitoring. Instrumenting the code to measure the NFR aspects, e.g., metric collection, when it is tested and real-time monitoring and alerting when the system is deployed in production. **Performance aspects** log response time of the key components in all layers and sub-layers. **Availability aspects** of the Health control probe, are leveraged to check the availability of the critical components/subsystems.

E. Quality Assurance

As part of the quality assurance phase, there are various nonfunctional tests that are carried out on the solution. These testing processes are fully integrated in the release and deployment processes and will be fully automated. Different tools (open source or commercial) are leveraged during the nonfunctional testing phase: **Load Testing**: To generate progressively and to trace the performance of a system as different levels and layers:

- **Stress Testing**: Conducted to push the application beyond its capabilities to observe how it reacts and responds
- **Vulnerability Testing**: The scanners are used to discover the weaknesses in a given system.
- **Penetration Testing**: A penetration test is an authorized simulated cyber-attack performed to evaluate the security of the application.
- **Reliability**: Ensures the redundancy mechanism works when the system encounters heavy load or unexpected failure

F. Sizing of NFRs

Software Non-functional Assessment Process - SNAP framework provides the basis for sizing nonfunctional requirements.

G. Analysis of NFRs

Architecture trade-off analysis method - ATAM methodology from the Software Engineering Institute - SEI is a Method to analyze NFR for Architectural Quality Attributes. Performance, reliability are the most leveraged, but others can be considered, e.g., security. ATAM reveals how well an architecture satisfies quality goals, and also enables the insight into how those quality goals interact with each other and how they trade-off against each other.

V. NFR FRAMEWORK

The below diagram depicts the NFR framework proposed in the paper. This is based on the various domains we have already in the earlier sections of this paper.

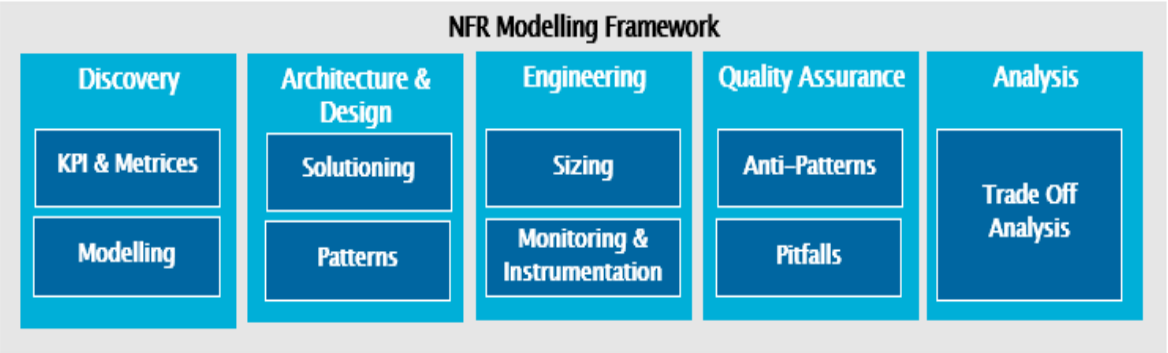

Fig. 2: NFR Modelling Framework

The table (see Annexure I on page no. 6) outlines the framework that provides the deep dive details for all lifecycle phases of NFRs from discovery to Deployment.

VI. CONCLUSION AND FUTURE WORK

Software architecture is an area of software engineering directed at developing large, complex applications in a manner that reduces development costs, increases quality and facilitates evolution. A central and critical problem software architect face is how to efficiently design and analyse IT architecture to meet NFRs. The paper proposed a framework-centric approach and an NFR Framework to address this problem, where NFRs are defined as reusable aspects to design and analysis. The software architecture is then able to be refined iteratively with analysis results until it is fit for the purpose to be accepted by the customer, thus reducing the development cost and time while enhancing the final system's completeness and consistency. Acceptance of software depends on the end user satisfaction, which largely depends on maximizing NFR elicitation and incorporation in the business applications. In this paper has proposed NFRs method and framework to address the entire lifecycle from discovery to deployment. We have also illustrated in the framework to model the NFRs that facilitates the analysis and sizing of NFRs. There are a number of interesting directions for the future work of NFR frameworks. One direction is to investigate the modelling and analysis of additional aspects of the NFRs that are not covered in this paper, e.g., Extensibility, usability, recovery, auditability. The NFR Framework can be leveraged to systematically analyse the synergistic and conflicting relationships among different NFRs.

CONFLICTS OF INTEREST

The authors declare that they have no conflicts of interest.

REFERENCES

[1] Rajesh Radhakrishnan, "Non-Functional Requirements (NFR) Framework", Open Group, 2009
[2] Md. Mijanur Rahman, Shamim Ripon, "Elicitation and Modeling Non-Functional Requirements – A POS Case Study", International Journal of Future Computer and Communication, Vol.2, Issue.5, 2013
[3] Mehrdad Saadatmand, Antonio Cicchetti, Mikael Sj¨odin,"UML-Based Modeling of Non-Functional Requirements in Telecommunication Systems", The Sixth International Conference on Software Engineering Advances, 2011
[4] Michael Dinkel, Uwe Baumgarten, "Modeling Nonfunctional Requirements: A Basis for dynamic Systems Management", ACM SIGSOFT Software Engineering Notes. Vol.30. pp.1-8, 2005.
[5] Christopher J. Pavlovski, Joe Zou, "Non-Functional Requirements in Business Process Modeling" , Fifth Asia-Pacific Conference on Conceptual Modelling, 2008
[6] Lirong Dai, Kendra Cooper, "Modeling and Analysis of Non-functional Requirements as Aspects in a UML Based Architecture Design", International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2005
[7] Ethan K. Jackson, Dirk Seifert and Markus Dahlweid, Thomas Santen, Nikolaj Bjrner, Wolfram Schulte, "Specifying and Composing Non-Functional Requirements in Model-based Development", International Conference, Software Composition, 2009
[8] Pavan Kumar Nanduru, "Non-Functional Requirement Modeling in the Early-Phase Software Product Life Cycle", Thesis for: Master's Degree in Software Engineering, 2017
[9] Pere Botella, Xavier Burgués, Xavier Franch, Mario Huerta, Guadalupe Salazar, "Modeling Non-Functional Requirements", 2001
[10] Christopher J. Pavlovski, Joe Zou, "Modeling Architectural Non Functional Requirements: From Use Case to Control Case", IEEE International Conference on e-Business Engineering (ICEBE'06), 2006

ABOUT THE AUTHOR

Sameer S. Paradkar is an enterprise architect with more than 20 years of extensive experience which spans System Integration, Product Development, and advisory Organizations. Sameer works as an SME on architecture modernization and transformation initiatives. He has worked on multiple digital transformations, engagements and large complex deals in North America, Europe, Middle East, and ANZ regions that presented a phased roadmap to the transformation maximizing business value while minimizing costs and risks. Sameer is certified and competent in different methodologies and frameworks including: TOGAF, NGOSS (e-TOM & SID), ITIL, COBIT, Agile, Scrum, DevOps, Scaled Agile Framework – SAFe and Business Capability Modeling. Sameer is part of the Architecture Group in AtoS. Prior to AtoS, he has worked in top tier SI and consulting organizations.

ANNEXURE I

Table 1: NFR Modelling Framework

#	NFRs	Availability	Scalability	Performance	Reliability	Maintainability	Interoperability	Security
1	**Category**	RunTime	RunTime	RunTime	RunTime	Design Time	Design Time	RunTime & Design Time
2	**KPI**	Availability, Planned Uptime, Planned Downtime, RTP - Recovery Time Objective, RPO - Recovery Point Objective	Concurrent Users, Growth Projections, Cost Per Transaction, Resource Projections	Throughput, Response Time, Storage capacity	MTBF : Mean time failures MTTR : Mean time to resume operation MTTF: Mean time to Failure	Fan-In, Fan-Out, Coupling & Cohesion metrics, Number of antipatterns, Cyclomatic complexity, and mean time to fix a defect, mean time to add new functionality	Compatibility with different OS, Platforms and Applications	Resistance to known attacks, time/effort/resources needed to find a key, probability/ time/ resources to detect an attack, Percentage of useful services still available during an attack, Percentage of successful attacks
3	**KPI-Metrics**	Availability: 99.99% Planned Downtime: 30 minutes RPO: 2 Hours RTO: 20 hours	Concurrent Users: 10000 Growth Projections: 10,0000 (5 years)	Transactions per second: 1000 Response Time: 2 ms	MTBF: 11 hours MTTR: 2 hours MTTF: 22 hours	Fan-In: 4 Fan-Out: 2 Cyclomatic Complexity: 10 Lines of Code - LOC: 10,000	Findability, Accessibility, Reusability, Discoverability, Openness, Transparency	Level of preparedness, Intrusion attempts, Mean Time to Detect (MTTD), Mean Time to Contain (MTTC), Mean Time to Resolve (MTTR), Days to patch, Number of cybersecurity incidents, Security ratings, Virus infection monitoring, Phishing attack success
4	**Modelling**	FURPS+, SAFe, TOGAF NFR Framework	FURPS+, SAFe, TOGAF NFR Framework	FURPS+, SAFe, TOGAF NFR Framework	FURPS+, SAFe, TOGAF NFR Framework	FURPS+, SAFe, TOGAF NFR Framework	FURPS+, SAFe, TOGAF NFR Framework	FURPS+, SAFe, TOGAF NFR Framework
5	**Strategy/Solutioning**	Clustering, Load Balancing, Fail Solutions, Geographic Redundancy, Stateless Model, Data Backups, Recovery and Replication, HA Configuration for Database Tier, Caching	Horizontal vs. Vertical Scaling, Database Read Replicas, Database Caching, Database Partitioning & Sharding, Distributed Architecture, Connection Pooling, Caching, Loosely Coupled Systems, Stateless Model, Lightweight Components, Avoid Chatty API	Caching, Distributed Architecture, Lightweight Components, AJAX APIs, Loose Coupling, Resource Pooling, Load Balancing, Lower Traffic on Wire, Coarse Grained Interfaces	Error & Exception Handling, Instrumentation, Store & Forward Mechanism, Queuing, Redundancy at all levels, Data Integrity – Full-Commit or Full Roll-back	Logical separation between components, leveraging Patterns, Object orientation, Program to interface not implementation, Design Pluggable Architecture, Leverage in-built platform and container APIs, Architect High Cohesion and Low Coupling	Leverage Canonical Model, Leverage Open Standards, Publish Semantics, Architect Leverage High Cohesion and Low Coupling, Program to Interfaces not Implementation, Exposing Well Defined Interfaces	Security Controls, Monitoring & Instrumentation, Encryption, Secure Transport Channel, DMZ, LDAP/AD, Auditing, Security Policies
6	**Patterns**	Queue Based Load Levelling, Throttling, Health End Point Monitoring, Failure Detection, Fast Recovery, Alternate Routes	Stateless Components, Loose Coupling, Lazy Loading, Caching, Parallelism, Partitioning, Routing	Cash-Aside, Choreography, CQRS, Event Sourcing, Fast Path, First Things First, Materialized View, Priority Queue, Sharding, Throttling, Alternate Route	Bulkhead, Circuit Breaker, Compensating Transaction, Leader Election, Scheduler Agent Supervisor	Ambassador, Anti-Corruption Layer, Backends for Frontends, CQRS, Gateway Routing, Leader Election, Sidecar, Strangler	Cross Platform Access, Cross Application Domain Access, Platform Independence, Platform-Scale Independence, Higher-level Service Facades	Federated Identity, Gatekeeper, Valet Key, Authentication, Authorization, Encryption, Data Confidentiality, SSO Delegator, Audit Interceptor, IAM, SIEM
7	**Sizing**	Software Non-functional Assessment Process - SNAP	Software Non-functional Assessment Process – SNAP	Software Non-functional Assessment Process – SNAP	Software Non-functional Assessment Process - SNAP	Software Non-functional Assessment Process - SNAP	Software Non-functional Assessment Process – SNAP	Software Non-functional Assessment Process - SNAP
8	**Monitoring & Instrumentation**	Application Performance Monitoring Tools	Application Performance Monitoring Tools, Load & Volume Testing	Application Performance Monitoring Tools, Load and Stress Testing	Static Code Analysers, Reliability Testing	Static Code Analysers	Static Code Analysers	Security Code Analysers, Vulnerability and Penetration testing
9	**Anti-Patterns**	Lack of Logging. Lack of Monitoring, Single Point of Failure, No Data Replication, Data Integrity, Lack of Redundancy	Rethrowing Exceptions, Logging to the Failed Resource, SQL Injection, Assumed Database Reliability, Configuration	Busy Database, Busy Front End, Chatty I/O, Extraneous Fetching, Improper Instantiation, Monolithic Persistence, No Caching, Synchronous I/O	Excess Flow of Notifications, Leveraging Advance Configuration Management Tools, Excess of systems frameworks & practices, Dependency Complexity impending interoperability or availability	Premature Optimization, Bike shedding, Analysis Paralysis, God Class, Fear of Adding Classes, Inner-platform Effect, Magic Numbers and Strings, Management by Numbers, Useless (Poltergeist) Classes	Stovepipes, Interface Migration, Islands of Implementation, Migration from Legacy Systems, Vendor Lock-In, Wolf Ticket, Reinvent the Wheel, Swiss Army Knife, Jumble, Autogenerated Interfaces	Injection, Broken Authentication, Sensitive Data Exposure, Broken Access Control, Security Misconfiguration, Cross Site Scripting, Insecure Deserialization, Leveraging Components with Vulnerabilities, Insufficient Logging & Monitoring
10	**Pitfalls**	Infrastructure failure, Infrastructure overload, Malicious activity, Data inconsistency, Many-to-one failover	Record locking, Thread synchronization, Database sequences, Opening connections, Swapping, I/O synchronization, Process spawning, Network contention, ORM, Synchronous Processes, Single Database	Database Connections, Network Latency and Connectivity Issues, Application Server Bottlenecks, Thread Deadlocks and Gridlocks, Improper Data Caching, Exceptions and Logs One Too Many, Infrastructure, Throttling	Cascading Failures, Failure at Scale, Redundancy of Code Base, Chaos Engineering, Traffic Routing Policies, Coupling, Failure Detection & Recovery	Excessive logging, Close Database Connections, underestimating Production load, Loading large result sets, Hard coding Configuration Parameters, Platform specific Code, Multiple versions JAR files	Platform & OS Integration, Data Integration, Cloud APIs & Interfaces, Standards, Portability	SQL Injection, Cross Site Scripting, Denial of Service, Man-in-the-Middle
11	**Trade-off Analysis**	Scenario-based Architecture Analysis Method - SAAM, Architecture Tradeoff Analysis Method - ATAM, Cost Benefit Analysis Method - CBAM, Scenario-Based Architecture Reengineering - SBAR, Performance Assessment of Software Architectures - PASA.	Scenario-based Architecture Analysis Method - SAAM, Architecture Tradeoff Analysis Method - ATAM, Cost Benefit Analysis Method - CBAM, Scenario-Based Architecture Reengineering - SBAR, Performance Assessment of Software Architectures - PASA.	Scenario-based Architecture Analysis Method - SAAM, Architecture Tradeoff Analysis Method - ATAM, Cost Benefit Analysis Method - CBAM, Scenario-Based Architecture Reengineering - SBAR, Performance Assessment of Software Architectures - PASA.	Scenario-based Architecture Analysis Method - SAAM, Architecture Tradeoff Analysis Method - ATAM, Cost Benefit Analysis Method - CBAM, Scenario-Based Architecture Reengineering - SBAR, Performance Assessment of Software Architectures - PASA.	Scenario-based Architecture Analysis Method - SAAM, Architecture Tradeoff Analysis Method - ATAM, Cost Benefit Analysis Method - CBAM, Scenario-Based Architecture Reengineering - SBAR, Performance Assessment of Software Architectures - PASA.	Scenario-based Architecture Analysis Method - SAAM, Architecture Tradeoff Analysis Method - ATAM, Cost Benefit Analysis Method - CBAM, Scenario-Based Architecture Reengineering - SBAR, Performance Assessment of Software Architectures - PASA.	Scenario-based Architecture Analysis Method - SAAM, Architecture Tradeoff Analysis Method - ATAM, Cost Benefit Analysis Method - CBAM, Scenario-Based Architecture Reengineering - SBAR, Performance Assessment of Software Architectures - PASA.