UDC 004.421.2

FORMALIZED METHODOLOGY FOR COMPATIBILITY AND ADAPTATION OF REQUIREMENTS IN INTELLIGENT DIAGNOSTIC SYSTEMS

Komleva N. O. – PhD, Associate Professor, Head of Software Engineering Department, Odesa Polytechnic National University, Odesa, Ukraine.

Liubchenko V. V. – Dr. Sc., Professor of Software Engineering Department, Odesa Polytechnic National University, Odesa, Ukraine.

ABSTRACT

Context. Ensuring the consistency and adaptability of requirements in systems operating under dynamic conditions and limited resources is a pressing issue in modern requirements engineering, especially in intelligent diagnostic and decision-making environments. These systems must process conflicting, outdated, or ambiguous requirements while operating in environments characterized by high uncertainty and dynamic conditions.

Objective. This work introduces a formalized methodology for analyzing and managing the compatibility of system requirements. The proposed approach integrates logical consistency, functional interaction, resource feasibility, and priority alignment to support system stability and responsiveness.

Method. The methodology is implemented as a multi-level framework that incorporates formal representations of functional, non-functional, and data-related requirements. It employs scenario-based modeling, a set of compatibility assessment models, and a dynamic algorithm for integrating new requirements. The integration process includes compatibility checks, adaptive refinement, expert-based weighting, and real-time feedback. The methodology's applicability is demonstrated through a hypothetical intelligent medical diagnostic system.

Results. The proposed methodology enables systematic identification and resolution of requirement conflicts, ensuring consistent execution and effective prioritization under resource constraints. Scenario-driven modeling and the formalization of core requirements establish a foundation for adaptive system behavior and real-time decision-making.

Conclusions. The developed methodology, which includes models and algorithms, enhances the reliability of intelligent systems operating in critical contexts. Future work will focus on extending the framework by incorporating fuzzy logic, machine learning techniques, and developing software tools for automated compatibility analysis and adaptive requirements management.

KEYWORDS: methodology, software engineering, requirements engineering, requirements compatibility, scenario-based modeling, priority-based harmonization.

ABBREVIATIONS

AI is an artificial intelligence;

AHP is an Analytic Hierarchy Process;

BP is a blood pressure;

HR is a heart rate;

Hx is a patient's electronic medical history;

NLP is a natural language processing;

SpO_2 is a blood oxygen saturation;

T is a body temperature;

TOPSIS is a Technique for Order Preference by Similarity to Ideal Solution;

UML is a Unified Modeling Language.

NOMENCLATURE

Acc is a degree to which the data matches the actual values of the quality attributes;

attemptCounter is a current counter of adaptation iterations for a requirement;

C is a set of contexts;

c_i is a context for i-th scenario;

Com is an indicator of the presence of values for all required attributes;

Con is an indicator of the absence of contradictions among attribute values;

Cred is an evaluator of the authenticity and truthfulness of the data;

D is a set of data elements;

d is a particular data element;

L is a total available resource capacity;

MaxAttempts is a maximum number of allowed adaptation attempts;

P is a set of system parameters;

Q is a set of data quality characteristics;

q_i is an i-th data quality characteristic;

R is a set of system requirements;

R_{base} is a set of core (baseline) requirements;

R_{current} is an active set of consistent requirements;

R_f is a set of functional requirements;

R_{nf} is a set of non-functional requirements;

R_{new} is a set of new or updated requirements;

r_i is a particular requirement;

Rec is an evaluator of the freshness and relevance of the data for use;

S is a set of system scenarios;

s_i is an i-th scenario;

t is a system time;

$w(r_i)$ is a priority weight for requirement r_i;

w_{limit} is a minimum acceptable weight threshold for attempting to integrate requirements;

$w_{\min}$ is a minimum weight threshold;

$\delta(s_i)$ is a criticality function for the i-th scenario;

$\rho(r_i)$ is a resource consumption function;

ρ_{total} is a total resource load;

Φ is a set of preconditions for the system requirements;

φ_i is a precondition for the i-th requirement;

DOI 10.15588/1607-3274-2025-4-9

Χ is a set of constraint satisfaction evaluators;

χ_i is a predicate function for the evaluation of the i-th constraint satisfaction;

Ψ is a set of system actions;

ψ_i is an i-th action of the system.

INTRODUCTION

As intelligent systems become increasingly complex, particularly in areas such as diagnostics and decision support, managing their requirements becomes more challenging. These systems operate in dynamic environments with limited computational resources and high degrees of uncertainty. In such environments, conventional methods of handling requirements often fall short. The effective and reliable functioning of such systems depends on their ability to manage evolving, sometimes conflicting, or ambiguous requirements in real-time.

Although many studies focus on classifying and ranking requirements, fewer explore how to resolve conflicts, manage resources, or adapt them over time. Despite substantial research, many methods still overlook how to resolve conflicts or adapt requirements under constraints [1]. In safety-critical domains such as medical diagnostics, transportation, and industrial automation, poorly coordinated or inconsistent requirements can lead to severe operational risks and failures [2].

The object of study is the process of managing system requirements under dynamic and uncertain conditions.

The subject of study is a formalized methodology designed to ensure the consistency and adaptability of requirements in intelligent diagnostic systems.

This work addresses critical limitations in existing methodologies by proposing a unified approach to functional, non-functional, and data-related requirements. The proposed solution supports real-time conflict detection, adaptive prioritization, and integration of new requirements through scenario-based modeling.

The purpose of the paper is to develop a formalized methodology for analyzing and managing the compatibility of system requirements in intelligent diagnostic systems. To achieve this goal, the following tasks are formulated:

– develop a multi-level framework for structuring requirements and operational scenarios;

– formalize logical, functional, and resource-based compatibility models;

– propose an adaptive algorithm for the integration of new requirements;

– validate the methodology through a practical example in the medical diagnostics domain.

The approach supports consistent, traceable, and context-aware requirements management across the system lifecycle. The methodology addresses key challenges faced by intelligent systems operating in dynamic and resource-constrained environments, such as resolving conflicts, adapting to new operational contexts, and maintaining stable system behavior. The proposed solution fills existing research gaps by integrating formal compatibility analysis, scenario-based modeling, and real-time adaptability mechanisms into a unified and coherent framework. This enables adaptation to changing requirements without compromising key functions.

1 PROBLEM STATEMENT

Intelligent diagnostic systems must ensure consistent execution of requirements even under conditions of dynamic environments, limited resources, and the continuous inflow of new data. In this context, a formalized procedure is needed to verify, update, and reconcile system requirements, ensuring stable and predictable system behavior.

The system operates based on a set of requirements that includes both functional actions and quality attributes. During operation, the system may receive new or modified requirements that must be checked for compatibility with those already implemented. It is crucial to consider the limitations of available computational resources, as well as the criticality of specific usage scenarios.

As input data, we consider an existing set of system requirements, R, to which new or updated requirements R_{new} may be added. Each requirement $r_i \in R \cup R_{\text{new}}$ is characterized by a priority weight $w(r_i)$, which reflects its relative importance, and a resource estimate $\rho(r_i)$, representing the cost of implementing it. The total available resource capacity in the current operating context is denoted by a scalar value L.

The objective of the proposed approach is to construct a consistent set of requirements that the system can implement without internal conflicts and without exceeding the available resource limit. At the same time, the set should preserve high-priority requirements as much as possible and ensure the inclusion of functions marked as critical, based on expert assessments or scenario definitions.

A solution is considered valid if the selected set contains no logically inconsistent or mutually exclusive requirements, and the total estimated resource usage does not exceed L. To ensure operational feasibility, all included requirements must satisfy compatibility constraints across logical, functional, and resource-related dimensions. In scenarios where conflicts arise or the available resources are insufficient, the system may discard new or lower-priority requirements.

The defined problem setting serves as a basis for constructing a formal model for compatibility checking, which supports the incremental integration of new requirements and enables adaptability under dynamic operational conditions. This model is particularly relevant for critical domains, such as medical diagnostics, where inconsistencies or outdated requirements can lead to severe errors in decision-making. The proposed approach can be further extended using rule-based algorithms or formal reasoning techniques to enable automated conflict detection and intelligent requirement filtering.

DOI 10.15588/1607-3274-2025-4-9

2 REVIEW OF THE LITERATURE

Recent studies on requirements engineering have increasingly emphasized the integration of artificial intelligence (AI), optimization techniques, and natural language processing (NLP) to enhance the prioritization, coordination, and adaptation of requirements in dynamic software systems. Habiba et al. [3] conducted a systematic mapping study on AI-based systems, identifying persistent challenges in aligning functional and non-functional requirements. Wong, Wagner, and Treude [4] provided a classification of approaches to self-adaptive systems, underscoring the need for flexible requirement management under changing conditions.

Several researchers have explored prioritization techniques that incorporate inter-requirement dependencies, usability metrics, and AI-based enhancements. Tauqeer and Jamil [5] introduced a method that combines dependency modeling with usability assessment to improve ranking precision. Tufail et al. [6] compared prioritization strategies and demonstrated that method selection can significantly affect software quality. Abbas et al. [7] proposed a test-case prioritization approach based on requirement dependencies, which proved especially useful during later stages of development. Ahuja, Sujata, and Batra [8] applied genetic algorithms to optimize the selection of critical requirements, enhancing prioritization outcomes through evolutionary computation.

To strengthen formalization in prioritization, Leshob, Hadaya, and Renard [9] developed a goal-oriented requirements language. Hudaib et al. [10] evaluated various techniques, highlighting the importance of selecting methods tailored to project-specific factors. Tasneem et al. [11] emphasized the value of adaptive re-prioritization in Agile frameworks. Maulana et al. [12] analyzed optimization-based algorithms for large-scale projects, confirming the effectiveness of heuristic approaches.

Tariq et al. [13] developed a UML profile to support the early identification of critical requirements, improving lifecycle planning. Ashraf et al. [14] examined factors influencing conflict within development teams, stressing the importance of clearly defined and achievable expectations in requirement negotiation. Vijayakumar and Nethravathi [15] reviewed NLP-based tools for analyzing requirement specifications, highlighting their potential for automation. Similarly, Anwar and Bashir [16] surveyed AI-driven prioritization techniques and outlined the potential of hybrid models that combine multiple analytical strategies.

Recent work by Hujainah et al. [17] introduced SRPTackle, a semi-automated technique that merges user input with optimization methods to effectively handle large requirement sets. Ali et al. [18] explored prioritization methods in highly configurable systems and found a positive impact on overall software quality. Deshpande et al. [19] employed active learning and ontology-based modeling to extract requirement dependencies, enabling more informed decision-making. Saeeda et al. [20] proposed a framework for enhancing requirements elicitation in Scrum environments, grounded in empirical insights from industrial projects.

Samer, Stettinger, and Felfernig [21] investigated the use of group recommender systems to facilitate collaborative prioritization, aiming to increase stakeholder consensus.

In summary, the reviewed literature reveals a clear trend toward automation, adaptability, and the use of intelligent techniques in requirements engineering. There is a growing emphasis on resolving inter-requirement conflicts, supporting decision-making under uncertainty, and optimizing requirement selection using AI, heuristics, and scenario-driven methods.

3 MATERIALS AND METHODS

This section outlines a formalized approach to representing system requirements, developing operational scenarios, and creating compatibility testing models and algorithms for the adaptive integration of new requirements within the dynamic environment of an intelligent diagnostic system.

System requirements are categorized into three main groups: functional, non-functional, and data requirements.

Functional requirements specify the expected behavioral logic of an intelligent system in response to defined input conditions. These requirements can be formalized using conditional-logical constructions of the form:

$$r_i \in R_f = (\varphi_i \Rightarrow \psi_i), \tag{1}$$

where $\varphi_i \in \Phi$ denotes a precondition expressed through observable features or parameter combinations, and $\psi_i \in \Psi$ represents the corresponding system action (e.g., hypothesis activation, state transition, or initiation of data processing).

Non-functional requirements impose constraints or define desired attributes of system operation, such as accuracy, response time, robustness to noise, adaptability, interpretability, and resource efficiency. These can be represented as predicates of the form:

$$r_i \in R_{nf} = \chi_i(p_k),\ p_k \in P, \tag{2}$$

where P is the set of system parameters, and $\chi_i \in \mathrm{X}$ is a predicate function that evaluates constraint satisfaction.

Data Requirements. Data constitutes a critical resource in intelligent systems, forming the informational basis for decision-making, model training, and behavioral adaptation. Data requirements define the type, structure, and quality of information the system must acquire, store, process, and validate. These requirements have a significant impact on system performance and reliability.

Each data element used by the system is characterized by a set of attributes, or named features, that capture the properties of objects or events. Quality requirements apply directly to the values of these attributes. Let Q denote the set of data quality characteristics, defined as follows:

© Komleva N. O., Liubchenko V. V., 2025
DOI 10.15588/1607-3274-2025-4-9

$$Q = \{Acc, Com, Con, Cred, Rec\}, \quad (3)$$

where *Acc* refers to the degree to which the data matches the actual values of the attributes, *Com* indicates whether values are present for all required attributes, *Con* represents the absence of contradictions among attribute values, *Cred* reflects the authenticity and truthfulness of the data, and *Rec* denotes the freshness and relevance of the data for its intended use. These characteristics are usually measured on a relative scale ranging from 0 to 1.

To enable formal verification of data quality, each characteristic $q_i \in Q$ is represented as a Boolean predicate applied to individual data elements. Let D be the set of data elements. For each $d \in D$, a predicate $q_i : D \to \{0,1\}$is defined to verify compliance with a specific quality characteristic:

$$q_i : D \to \{0,1\} \quad q_i(d) = \begin{cases} 1, \text{ if } d \text{ satisfies requirement } q_i \\ 0, \text{ otherwise.} \end{cases} \quad (4)$$

The integral predicate describes the overall compliance with quality requirements:

$$Q(d) = \begin{cases} 1, \text{ if } \forall q_i \in Q, \ q_i(d) = 1, \\ 0, \text{ otherwise.} \end{cases} \quad (5)$$

While predicates allow binary verification of compliance, practical applications often require more flexible, quantitative assessments. In cases of partial compliance or when data adaptation is required, it is beneficial to measure the degree of satisfaction with the requirement. For this purpose, each quality characteristic can be associated with a metric function that takes a value in the interval [0, 1], reflecting the level of compliance with the respective requirements.

To ensure the consistency of requirements, it is necessary to formally structure the system's operation scenarios, which represent its modes of functioning under various contextual conditions.

The formation of a consistent set of requirements necessitates the explicit structuring of operation scenarios, which represent distinct modes of functioning for intelligent systems. Scenarios facilitate the specification of the sequence of system actions under defined conditions, reveal dependencies among requirements, and enable the evaluation of contextual influences on the relevance of requirements.

A system operation scenario is defined as a formalized representation of the sequence of actions, triggering conditions, and corresponding system responses associated with a specific operational context or goal-oriented mode. Each scenario encompasses a localized operational environment, along with its corresponding subset of relevant requirements.

Let C denotes the set of possible contexts (e.g., input conditions, resource availability, external events), and S represents the set of system operation scenarios. A requirement $r_i \in R$ can be represented as an ordered structure that links a scenario, its activation context c_i, and the set of associated system actions $\psi_i \in \Psi$:

$$r_i = \langle s_i, c_i, \psi_i \rangle, \quad (6)$$

where $s_i \in S$ denotes the system scenario, $c_i \subseteq C$ defines the context under which the scenario is activated, $\psi_i \subseteq \Psi$ is the set of system actions to be performed within this scenario.

Among all scenarios $S = \{s_1, s_2, \ldots, s_n\}$, a subset is identified as critical scenarios, those in which requirement violations are unacceptable due to their direct impact on system safety, stability, or goal achievement. Failures in such scenarios may lead to catastrophic consequences, including functional degradation, loss of control, or violation of essential operational constraints.

To formalize this, a Boolean criticality function is introduced:

$$\delta(s_i) \in \{0,1\}, \quad (7)$$

where $\delta(s_i) = 1$ indicates that scenario s_i is critical, $\delta(s_i) = 0$ denotes a non-critical or auxiliary scenario that supports extended functionality.

Scenario criticality is typically assessed using the following criteria:

– user expectations: the scenario delivers a primary outcome for which the system is designed;

– functional necessity: the scenario is indispensable for the system's core operation;

– safety or regulatory compliance: the scenario is mandated by standards or constraints;

– risk assessment: failure to execute the scenario may result in unacceptable operational risks;

– execution context: the criticality of a scenario may vary based on the system's current state or environment.

Thus, operation scenarios serve as a structural backbone for requirements modeling, defining the functional context within which system behavior unfolds. They establish formal connections between contextual conditions, expected system actions, and target operational states. Scenarios enable:

– the formation of a dynamic set of core (baseline) requirements;

– the verification of functional consistency within specific operating modes;

– the resolution of requirement conflicts and prioritization under resource constraints.

Furthermore, operation scenarios form the basis for constructing behavioral models of the system, which describe adaptive responses to dynamic external conditions. By incorporating scenario criticality, it becomes possible to identify and maintain a minimal subset of essential requirements that ensure continuous execution of key system functions in the most operationally significant situations.

To identify the minimally sufficient conditions for system stability and functionality, it is essential to formal-

DOI 10.15588/1607-3274-2025-4-9

ize a set of core requirements that reflect both universal needs and critical operational scenarios.

Core requirements represent the minimally sufficient set of conditions required to ensure the execution of a system's critically essential functions. These requirements are derived by combining two distinct subsets:

– universal requirements, which are common across all operational scenarios, and

– requirements that appear in at least one critical scenario.

Formally, the set of core (baseline) requirements R_{base} is defined as:

$$R_{base} = (\bigcap_{i=1}^{n} r_i) \cup (\bigcup_{\delta(s_i)=1} r_i), \quad (8)$$

where

– $\bigcap_{i=1}^{n} r_i$ denotes the intersection of all scenario-specific requirement sets, i.e., the set of universal requirements that are included in every system operation scenario, independent of criticality;

– $\bigcup_{\delta(s_i)=1} r_i$ represents the union of requirements associated with all critical scenarios, regardless of whether they are also present in other scenarios.

By treating universal requirements as a separate component, their independent monitoring becomes more manageable in practical scenarios. For example, when operational scenarios are modified or extended, this separation enables validation of whether the universality property of specific requirements remains intact.

This formulation ensures that both the system's global behavioral characteristics and its performance under critical conditions are taken into account. The core requirements thus serve as a foundation for:

– initial requirement alignment,

– subsequent compatibility verification of new or changing requirements, and

– decision-making in adaptive systems, particularly under resource-constrained conditions where trade-offs between requirements must be managed.

To ensure reliable system behavior, it is necessary to establish clear criteria for determining the compatibility of requirements across multiple dimensions, including logic, resources, functionality, and priorities.

Compatibility refers to the ability of two or more requirements to be executed concurrently under given contextual conditions, resource constraints, and the system's logical architecture. Identifying incompatible requirements is essential to prevent conflicts during system modeling, testing, and operation. The evaluation of requirement compatibility is conducted along the following key dimensions.

Logical consistency. Requirements must not contradict each other in their formal representations. For instance, if one requirement initiates an action under a condition φ, while another prohibits the same action under the same condition, a logical conflict exists. Such inconsistencies can be detected through formal logical analysis or the construction of requirement dependency graphs.

Resource feasibility. Logically consistent requirements may nonetheless be incompatible due to resource limitations, such as insufficient computational time, processing power, or data access. Compatibility in this context is assessed by comparing the aggregate resource demands of a requirement set to the available resource budget.

Functional interaction. Within a given scenario, requirements should be either functionally complementary or neutral. Conflicts arise when the fulfillment of one requirement modifies the system state in a way that renders another requirement infeasible or irrelevant. Functional compatibility is verified through system state transition analysis or impact modeling of requirements.

Priority consistency. In cases where requirements are mutually exclusive, a resolution must be guided by their relative priority. A priority function (e.g., weight coefficient or rank ordering) is applied to determine which requirement prevails under conflict. Compatibility is ensured if the system can resolve conflicts following this priority structure.

Importantly, incompatibility is often multidimensional, meaning that requirements may pass compatibility checks in one dimension but fail in another. Therefore, multi-stage analysis is recommended, with each stage evaluating compatibility in a distinct domain (logical, resource, functional, or priority-based) and considering the specific application context.

A crucial aspect of requirements management is maintaining consistency within the core set while accounting for its potential variability over time due to dynamic operational conditions and evolving system contexts.

Let R_{base} denote the set of core requirements, as defined in (8). We now formulate the consistency condition and a theorem on its temporal variability.

The following *statement* defines the condition under which the set R_{base} can be considered consistent.

The set R_{base} is said to be consistent if all pairs of requirements r_i, $r_j \in R_{\text{base}}$ satisfy the criteria of logical consistency, resource feasibility, functional interaction, and priority alignment.

If any of these criteria are violated, the structure of R_{base} must undergo preliminary alignment or adaptive modification.

The *corollary* to the above statement demonstrates the dynamic nature of R_{base}.

Since the criticality status of operational scenarios may evolve due to changes in environmental conditions, system configurations, or user priorities, the composition of R_{base} is also subject to change. Because the set changes over time, it needs to be revisited when the situation changes noticeably.

The following theorem formalizes the temporal variability of the core requirement set R_{base} under changing system conditions.

The set of core requirements R_{base} is temporally variable, i.e.,

© Komleva N. O., Liubchenko V. V., 2025
DOI 10.15588/1607-3274-2025-4-9

$$R_{\text{base}} = R_{\text{base}}(t), \tag{9}$$

where t denotes system time. This implies that the composition of R_{base} may evolve due to changes in the set of scenarios S, the criticality function $\delta(s_i)$, and the structure or definition of the associated requirements.

Proof of the theorem.

Assume the contrary, i.e., R_{base} remains constant over time. Suppose the system operates in a dynamic environment, where external changes such as fluctuations in resource availability, threat levels, or technical parameters may occur. Moreover, user goals and priorities may shift over time. These factors naturally lead to the modification of scenarios S, with some scenarios being deprecated and others gaining critical status.

Since each scenario s_i is associated with a specific set of requirements r_i, any change in scenario composition or criticality status directly affects the union of requirements included in R_{base}. According to Equation (8), such changes invalidate the assumption that R_{base} is time-invariant.

Therefore, the assumption of constancy is false.

Various models can be employed to assess the compatibility of requirements, including those based on logical consistency, resource feasibility, functional interaction, and priority harmonization.

Logical analysis of condition-consequence rules is employed to detect structural incompatibilities among functional requirements specified in the form of conditional statements. A typical representation of a functional requirement $r_i \in R_f$ is as follows:

$$r_i = (\varphi_i \Rightarrow \psi_i), \tag{10}$$

where φ_i denotes the activation condition, ψ_i denotes the expected system response or action.

Under this representation, logical incompatibility may arise in several typical cases:

1. Identical Conditions with Conflicting Consequences. When $\varphi_i = \varphi_j$ but ψ_i and ψ_j are contradictory. For example:

– if temperature > 80°C $\Rightarrow$ activate_cooling

– if temperature > 80°C $\Rightarrow$ shut_down_system

2. Overlapping Conditions with Divergent Effects. If $\varphi_i \cap \varphi_j \neq \varnothing$, both conditions may be simultaneously satisfied and trigger requirements leading to different, possibly incompatible consequences. For example:

– if temperature > 70°C $\Rightarrow$ activate_alarm

– if temperature > 80°C $\Rightarrow$ stop_engine

3. Mutually Exclusive Consequences. Even when conditions are distinct, the resulting actions ψ_i and ψ_j may conflict. For example:

– ψ_1= activate_standby_mode

– ψ_2= activate_operational_mode

If these represent mutually exclusive system states, simultaneous activation leads to inconsistency.

A requirement dependency graph provides a structured means to model logical relationships among conditional requirements. Each node in the graph represents a requirement r_i, while directed edges $r_i \rightarrow r_j$ indicate a dependency, implying that:

– the consequence ψ_i of requirement r_i alters the condition φ_j of requirement r_j, or

– the actions ψ_i and ψ_j interact or contradict each other.

With the help of the dependency graph, four types of problematic situations can be identified.

1. Detection of Cyclic Dependencies. Cycles of the form $r_1 \rightarrow r_2 \rightarrow \ldots \rightarrow r_n \rightarrow r_1$ indicate logical loops. These loops suggest that the execution of a requirement depends on a condition influenced by its own consequence. Such constructs can lead to:

– undefined or infinite execution behavior,

– situations where no requirement can be executed without violating preconditions.

2. Identification of Conflict Zones. A conflict zone is a subgraph where identical or overlapping conditions activate multiple requirements yet produce incompatible outcomes. These are especially critical in systems with parallel execution, where:

– conflicting requirements may compete for resources,

– simultaneous execution may produce inconsistent states.

3. Determination of Execution Order (Topological Sorting). If the dependency graph is acyclic, a topological sort can be applied to determine a valid sequence of requirement execution. This ensures:

– all prerequisites of a requirement are satisfied before its activation,

– consistency in execution logic.

For example: Check data availability → Perform computation → Output result.

4. Localization of Critical or Unstable Nodes. These are the nodes with:

– many incoming edges are condition-sensitive and may become unstable under minor context variations,

– many outgoing edges have high propagation impact and must be monitored for change management or testing prioritization.

By incorporating logical analysis and dependency graph modeling, system engineers can preemptively detect structural incompatibilities, ensure correct execution ordering, and manage the complexity of interdependent requirements in a dynamic system environment.

The resource feasibility model is employed to assess whether the cumulative consumption of system resources by active requirements remains within predefined limits. This assessment enables the detection of potential resource conflicts or system overloads before implementation or deployment.

Let $R = \{r_1, r_2, \ldots, r_n\}$ denote the set of active requirements. The set R is said to be resource-feasible if the following inequality is satisfied:

$$\sum_{r_i \in R} \rho(r_i) \leq L, \tag{11}$$

where $\rho(r_i)$ is a resource consumption function that quantifies the amount of a specific system resource (e.g., CPU time, memory, bandwidth) required to fulfill requirement

DOI 10.15588/1607-3274-2025-4-9

r_i, L represents the resource constraint, i.e., the maximum available quantity of the resource under consideration.

This inequality ensures that the aggregate resource demand of all concurrently active requirements does not exceed the system's resource capacity. If the condition is violated, it implies that the system cannot safely support the simultaneous execution of all requirements in R. In such cases, one or more of the following strategies must be employed:

– Requirement prioritization and pruning. Eliminate or defer requirements with the lowest priority or least criticality.

– Requirement adaptation. Modify individual requirements to reduce their resource consumption, potentially through simplification, partial execution, or degradation of non-critical features.

– Resource reallocation or scaling. If feasible, increase the available resource capacity (e.g., dynamic scaling in cloud environments).

The model thus provides a quantitative basis for ensuring safe and efficient system operation under constrained conditions, especially in real-time, embedded, or resource-sensitive domains.

Functional analysis is employed to verify the consistency of requirements that operate within the scope of a single system scenario. This analysis is based on the formal representation of scenarios as triplets defined in expression (6). It involves examining whether the set of requirements within a given scenario exhibits contradictions in terms of activation conditions, expected effects, or execution order.

The following categories of conflicts are commonly identified through functional analysis:

– State conflict. A requirement alters the system state in a way that renders another requirement irrelevant or inapplicable. For example, the requirement "switch the system to standby mode" conflicts with "start diagnostics", as standby mode may disable diagnostic operations.

– Conflict of execution conditions. Two requirements are activated under partially overlapping conditions but prescribe contradictory system actions. For example, one requirement triggers a shutdown when the system temperature exceeds 80°C, while another activates emergency cooling under the same or similar conditions.

– Effect inconsistency. Improper sequencing of requirement execution leads to erroneous system behavior. For example, attempting to read data from a sensor after the sensor has been powered down results in system failure or invalid outputs.

Scenarios identified as critical, as formalized in expression (7), require particular attention. Functional inconsistencies within such scenarios may compromise system safety, stability, or goal attainment. Therefore, ensuring the functional consistency of requirements in critical scenarios is a key step in the validation and reliability assurance process.

In scenarios where the simultaneous execution of all system requirements is infeasible due to logical conflicts or resource constraints, the system must selectively prioritize the most critical requirements. This is accomplished through priority-based harmonization, which utilizes weight coefficients to quantify the relative importance of each requirement.

Each requirement $r_i \in R$ is associated with a priority weight $w(r_i) \in [0,1]$, representing its contextual importance. These weights can be derived through expert judgment, analytical techniques, or data-driven methods, including historical analysis or dependency-based inference.

When a conflict between requirements arises, the system retains the requirement r_i for which $w(r_i) > w(r_j)$, ensuring that higher-priority requirements are preserved.

To resolve conflicts or address execution constraints, the system selects a subset of requirements based on their priorities. The following algorithmic strategies are commonly applied:

– Acceptability Threshold Strategy. A predefined minimum threshold, $w_{\min}$, is established. Only requirements satisfying $w(r_i) \geq w_{\min}$ are eligible for execution. This approach filters out low-priority requirements without requiring additional compatibility checks. The threshold may be static or dynamically adjusted based on the operational context.

– Greedy Sorting Strategy. All requirements are sorted in descending order of weight. Each requirement is sequentially evaluated for compatibility with the current active set. If compatible, it is included; otherwise, it is either discarded or deferred. This strategy ensures the maximum inclusion of high-priority requirements within constraints.

– Compromise Harmonization Strategy. Requirements with lower weights may be accepted, provided they do not conflict with those already selected. This approach enhances system flexibility by enabling the execution of non-critical yet beneficial requirements, particularly when strict prioritization is not mandatory.

In addition to basic strategies, more advanced or context-sensitive approaches may be applied:

– Multi-Criteria Ranking. Requirements are evaluated across multiple dimensions, such as benefit, risk, cost, and aggregated via AHP, TOPSIS, or similar methods.

– Heuristic and Evolutionary Algorithms. Techniques such as genetic algorithms or ant colony optimization are employed to identify near-optimal subsets of requirements in the presence of complex interdependencies and constraints.

– Contextual Harmonization Based on Scenarios. The priority weight $w(r_i)$ is interpreted relative to the specific scenario in which the requirement is applied. A requirement may have low priority in routine operation but become critical in emergency or failure scenarios. This adaptive approach ensures that system behavior remains aligned with environmental demands.

– Group Expert Harmonization. The importance of requirements is evaluated through a collective expert assessment, using aggregation methods such as the median or weighted voting to form a consensus-based priority structure.

– Stochastic Methods. Weights are modeled as random variables or probability distributions, allowing harmonization strategies to function under uncertainty and incomplete knowledge.

Priority-based harmonization provides a systematic framework for managing competing requirements under limitations. By combining weighted importance with compatibility analysis, the system can adaptively optimize its functionality, achieving an effective balance between goal satisfaction, resource efficiency, and operational safety.

The integration of new requirements into a software system is a critical task that demands rigorous checks for compatibility, relevance, and priority. To preserve system consistency and prevent operational conflicts, this process must be governed by structured algorithmic mechanisms that support informed decision-making.

To support structured integration of new requirements, a baseline algorithm is proposed that ensures compatibility validation, prioritization, and bounded adaptation.

At system initialization, the active set of consistent requirements is defined as:

$$R_{\text{current}} := R_{\text{base}},$$

where R_{base} represents the baseline functionality, guaranteed to be conflict-free and executable.

Let R_{new} denote the set of new requirements to be considered for integration; $w(r_i)\in[0,1]$ be the priority weight of requirement r_i; w_{limit} be the minimum acceptable weight threshold for attempting integration; *MaxAttempts* be the maximum number of adaptation attempts allowed; *attemptCounter* be the current counter of adaptation iterations for a requirement.

The integration process proceeds as follows:

1. Sorting. Sort all requirements in R_{new} in descending order by their weights $w(r_i)$.

2. Sequential Processing. For each requirement $r_i\in R_{\text{new}}$:

– Perform a compatibility check with the current set R_{current};

– If compatible, add r_i directly to R_{current};

– If incompatible, continue to Step 3.

3. Priority Evaluation

– If $w(r_i)<w_{\text{limit}}$, reject r_i without adaptation;

– If $w(r_i)\geq w_{\text{limit}}$, proceed to adaptation.

4. Adaptation Loop

– Initialize *attemptCounter* := 0;

– While r_i remains incompatible and *attemptCounter* < *MaxAttempts*:

- o Attempt to refine or modify r_i;
- o Re-evaluate compatibility with $R_{current}$;
- o Increment *attemptCounter*.

5. Post-Adaptation Decision

– If the modified r_i becomes compatible, add it to R_{current};

– Otherwise, permanently reject r_i.

Figure 1 illustrates the integration flow, depicting the logical transitions between evaluation stages, compatibility checks, and adaptation outcomes.

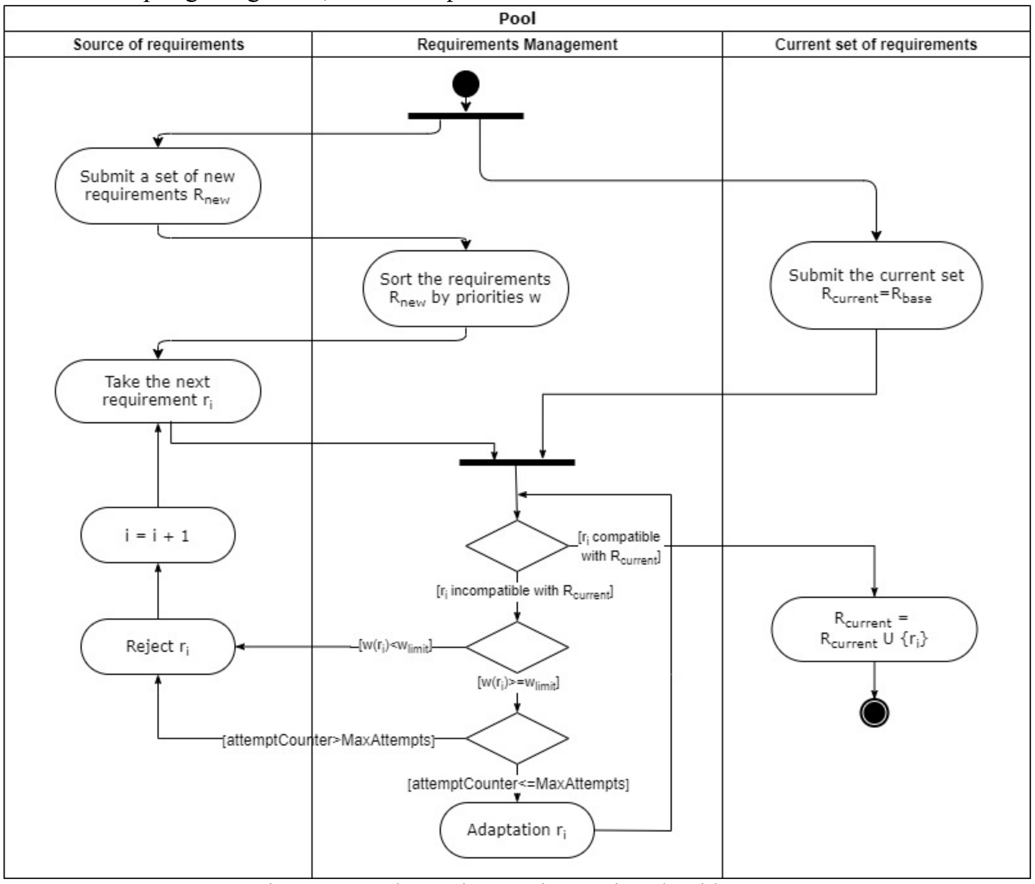
Pool
Source of requirements
Requirements Management
Current set of requirements
Submit a set of new requirements R_{new}
Sort the requirements R_{new} by priorities w
Submit the current set $R_{current}=R_{base}$
Take the next requirement r_i
i = i + 1
Reject r_i
[r_i compatible with $R_{current}$]
[r_i incompatible with $R_{current}$]
[$w(r_i)<w_{limit}$]
[$w(r_i)>=w_{limit}$]
[attemptCounter>MaxAttempts]
[attemptCounter<=MaxAttempts]
Adaptation r_i
$R_{current} = R_{current} \cup \{r_i\}$

Figure 1 – Basic requirement integration algorithm

DOI 10.15588/1607-3274-2025-4-9

We introduce a structured algorithm to align new requirements with the current system configuration. It enables incremental evolution of the requirement set while ensuring that consistency is maintained through compatibility validation and bounded adaptation attempts. Unlike expert- or context-driven methods, this model assumes a static decision logic, which is suitable for baseline harmonization.

During system operation, runtime feedback can be collected regarding the failure patterns of specific requirements, e.g., those that consistently trigger conflicts, violate constraints, or fail due to environmental conditions. This feedback may be used to adjust the priority weights $w(r_i)$ dynamically.

Such adjustments support the development of an adaptive prioritization mechanism, where:

– frequently violated or inapplicable requirements may have their weight reduced;

– persistently successful or mission-critical requirements may have their weight increased;

– requirements with declining utility can be excluded from critical scenarios.

By leveraging operational feedback, the system gradually evolves into a context-aware and conflict-resilient behavior, thereby improving its ability to reconcile incoming requirements while maintaining stability and resource efficiency.

4 EXPERIMENTS

To demonstrate the applicability of the proposed requirement alignment and adaptation model, we present a hypothetical use case of an intelligent diagnostic system designed for real-time medical monitoring in an intensive care unit. The system continuously evaluates patient physiological parameters and generates diagnostic actions to enable timely intervention in critical clinical situations.

The system acquires real-time data from a network of physiological sensors, which measure the following parameters:

– body temperature (T);
– heart rate (HR);
– blood oxygen saturation (SpO2);
– blood pressure (BP).

Additionally, the system utilizes information from the patient's electronic medical history (Hx) for contextual analysis.

The system supports the execution of three essential diagnostic scenarios:

s_1: Detection of hyperthermia,
s_2: Recognition of hypoxia,
s_3: Shock prediction and warning.

Each scenario is operationalized through a corresponding set of functional requirements, structured according to the representation model introduced in (6).

Table 1 outlines six functional requirements associated with scenarios. Each requirement includes the activation context, the prescribed system action, and the relevant scenario.

Table 1 – System Requirements and Scenario Association

Requirement	Context	System actions	Scenario
r_1	T > 38°C	Activate cooling	s_1
r_2	SpO_2 < 90%	Trigger alarm	s_2
r_3	BP < 90/60	Check heart rate	s_3
r_4	HR > 110	Raise suspicion of tachycardia	s_3
r_5	Hx contains "stroke"	Increase the priority of SpO_2 check	s_2
r_6	T > 38°C ∧ SpO_2 < 85%	Activate complex response	$s_1 \cap s_2$

Figure 2 presents a dependency graph illustrating the logical interdependencies and potential conflicts among the system requirements.

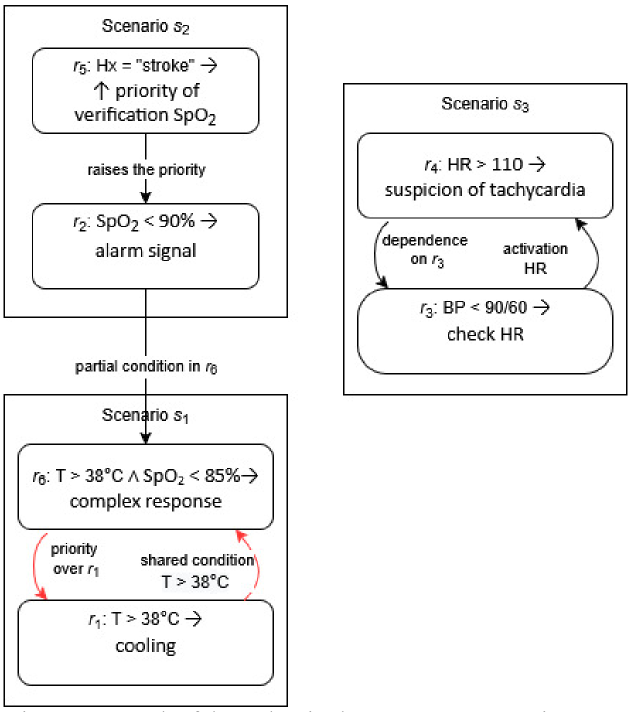
Scenario s_2
r_5: Hx = "stroke" → ↑ priority of verification SpO_2
raises the priority
r_2: SpO_2 < 90% → alarm signal
partial condition in r_6
Scenario s_1
r_6: T > 38°C ∧ SpO_2 < 85% → complex response
priority over r_1
shared condition T > 38°C
r_1: T > 38°C → cooling
Scenario s_3
r_4: HR > 110 → suspicion of tachycardia
dependence on r_3
activation HR
r_3: BP < 90/60 → check HR

Figure 2 – Graph of dependencies between system requirements with conflict indications

In scenario s_3, a cyclic interdependence exists between r_3 and r_4: a drop in blood pressure (r_3) triggers HR monitoring, which, if elevated (r_4), indicates a risk of tachycardia, potentially reinforcing the check.

In scenario s_2, requirement r_5 increases the priority of r_2 when the patient's history includes a diagnosis of stroke. Additionally, r_2 is logically a subset of r_6, which activates a complex response in the presence of combined hyperthermia and severe hypoxia.

A notable conflict arises between r_1 and r_6. Both are activated by the same temperature threshold (T>38°C), but prescribe different actions: cooling (r_1) and a complex response (r_6), which may be mutually incompatible. This conflict, indicated in red in the graph, necessitates priority-based harmonization.

DOI 10.15588/1607-3274-2025-4-9

5 RESULTS

To assess system feasibility under operational constraints, Table 2 provides the estimated resource consumption, $\rho(r_i)$, and priority weights, $w(r_i)$, for each requirement. Let the maximum allowable resource load be limited to 100 units.

Table 2 – Requirement Resource Usage and Priority

Requirement	Action description	$\rho(r_i)$	$w(r_i)$
r_1	Activate cooling	20	0.7
r_2	Trigger alarm	10	0.9
r_3	Check heart rate	15	0.6
r_4	Raise suspicion of tachycardia	20	0.5
r_5	Increase the priority of SpO_2 check	10	0.8
r_6	Activate complex response	50	1.0

As shown in the table, the highest priority is assigned to response r_6, while the lowest belongs to r_4. Resource usage varies from 10 to 50 units, enabling the evaluation of acceptable requirement combinations under the 100-unit constraint.

Feasibility analysis indicates that the execution of r_6 in combination with any two additional requirements remains within the acceptable load. However, a configuration involving r_1, r_6, r_4, and r_5 reaches the upper resource limit. If all six requirements are simultaneously triggered, the total resource load reaches 125 units, exceeding the system's capacity.

The system must harmonize requirements based on their relative priorities to resolve conflicts under constraints. In the case of a conflict between r_1 and r_6, the higher priority of r_6 ($w = 1.0$) prevails over r_1 ($w = 0.7$), and r_6 is selected.

Assume that the system detects the following conditions concurrently:

– $T > 38°C$,
– SpO2 < 90%,
– BP < 90/60,
– HR > 110.

This scenario activates the requirements: r_1, r_2, r_3, r_4, and r_6. The corresponding total resource load is:

$$\rho_{total} = 20(r_1)+10(r_2)+15(r_3)+20(r_4)+50(r_6) = 115 \text{ units.}$$

Since this exceeds the 100-unit limit, a greedy algorithm is applied. Requirements are sorted by descending priority: $r_6(1.0)$, $r_2(0.9)$, $r_1(0.7)$, $r_3(0.6)$, $r_4(0.5)$.

The system sequentially includes requirements until the cumulative load reaches the limit:

– include $r_6 \to \rho_{\text{total}} = 50$;
– include $r_2 \to \rho_{\text{total}} = 60$;
– include $r_1 \to \rho_{\text{total}} = 80$;
– include $r_3 \to \rho_{\text{total}} = 95$;
– r_4 is excluded (would exceed the limit).

Thus, the system executes r_6, r_2, r_1, and r_3, while r_4 is dropped due to insufficient resources.

These results are obtained via simulation-based evaluation of requirement interaction models within the developed formal framework.

Thus, the model enables a quantitative assessment of requirement compatibility and supports decision-making regarding inclusion/exclusion based on priorities and available resources.

6 DISCUSSIONS

The proposed method demonstrates that adaptive harmonization, grounded in compatibility, prioritization, and resource constraints, supports reliable, context-aware system behavior. Even under conflicting conditions and limited capacity, the system prioritizes mission-critical functionality, ensuring continuity of care and system consistency.

These findings demonstrate that the proposed methodology significantly enhances the reliability of intelligent diagnostic systems operating in real-time, resource-constrained environments. By applying formalized compatibility checks and dynamic priority-based filtering, the system effectively balances operational feasibility with functional adequacy.

Unlike conventional approaches that rely on static prioritization or fixed thresholds, the proposed model introduces structured multi-factor evaluation. For instance, while rule-based engines are commonly used in medical expert systems, they often lack built-in mechanisms to resolve requirement conflicts under resource constraints. Our method enhances these capabilities by integrating logical consistency, functional dependencies, and resource feasibility into the modeling process.

Unlike approaches that mainly focus on requirement classification and selection, our framework directly addresses internal inconsistencies and contextual variability. Moreover, the integration of scenario-based reasoning enables the adaptive inclusion of critical functionality, even in rapidly changing conditions, the factor that has been underrepresented in prior models.

A key advantage of the proposed model is its applicability across a wide range of mission-critical domains. While the use case involves medical diagnostics, the methodology is equally applicable to other mission-critical domains, such as industrial monitoring, autonomous systems, and cyber-physical infrastructures.

Nonetheless, the current implementation has several limitations that warrant attention. The current implementation assumes that input parameters, such as priority weights and resource estimates, are externally defined, which may introduce subjectivity or inconsistency in real-world deployment. Additionally, the conflict resolution algorithm used in the simulation is greedy and may not always produce globally optimal configurations.

Further research is required to explore the integration of machine learning techniques for the automated estimation of requirement parameters, as well as advanced optimization heuristics to manage trade-offs among competing requirements better.

CONCLUSIONS

This paper presents a systematic methodology for ensuring the compatibility and adaptability of requirements in intelligent diagnostic systems operating under dynamic environments and limited resources. The proposed approach addresses critical shortcomings of conventional techniques, which often emphasize prioritization but insufficiently consider logical consistency, resource constraints, and context-awareness in requirement harmonization.

The key contributions of this work are as follows:

– formal structure that distinguishes functional, non-functional, and data-related requirements, treating data-specific constraints as a distinct and formally defined category;

– scenario-driven framework for determining the criticality of requirements and constructing a dynamic baseline set of core requirements;

– multi-dimensional compatibility analysis model that integrates logical consistency, resource feasibility, functional interaction, and priority-based reasoning;

– adaptive integration algorithm that incorporates compatibility checks, expert-informed prioritization, and feedback-driven weight adjustments.

A practical evaluation using a hypothetical intelligent medical diagnostic system demonstrated the effectiveness of the proposed models in detecting requirement conflicts, improving system consistency, and supporting stable behavior under operational constraints.

Future research will focus on developing a prototype tool for automated compatibility analysis and adaptive requirements management. Additionally, the framework will be expanded to incorporate fuzzy logic, machine learning techniques, and scenario-based multi-criteria optimization, thereby supporting advanced real-time decision-making.

REFERENCES

1. Radwan A. M., Abdel-Fattah M. A., Mohamed W. AI-Driven Prioritization Techniques of Requirements in Agile Methodologies: A Systematic Literature Review, *International Journal of Advanced Computer Science and Applications*, 2024, Vol. 15, No. 9, pp. 812–823. DOI: 10.14569/IJACSA.2024.0150983.
2. Heyn H.-M., Knauss E., Muhammad A. P. et al. Requirement Engineering Challenges for AI-intense Systems Development, *arXiv*, 2021. DOI: 10.48550/arXiv.2103.10270.
3. Habiba U., Haug M., Bogner J. et al. How mature is requirements engineering for AI-based systems? A systematic mapping study on practices, challenges, and future research directions, *Requirements Engineering*, 2024, Vol. 29, pp. 567–600. DOI: 10.1007/s00766-024-00432-3.
4. Wong T., Wagner M., Treude C. Self-adaptive systems: A systematic literature review across categories and domains, *Information and Software Technology*, 2022, Vol. 148, 106934. DOI: 10.1016/j.infsof.2022.106934.
5. Tauqeer M., Jamil H. Requirements Prioritization – Modeling Through Dependency and Usability with Fusion of Artificial Intelligence Technique, *International Journal of Innovative Science and Technology*, 2025, Vol. 7, No. 1, pp. 146–158.
6. Tufail H., Qasim I., Masoo M. F. et al. Towards the Selection of Optimum Requirements Prioritization Technique: A Comparative Analysis, *IEEE 5th International Conference on Information Management : proceedings*. Cambridge, UK, 2019, pp. 227–231. DOI: 10.1109/INFOMAN.2019.8714709.
7. Abbas M., Inayat I., Saadatmand M. et al. Requirements Dependencies-Based Test Case Prioritization for Extra-Functional Properties, *IEEE 12th International Conference on Software Testing, Verification and Validation : proceedings*. Xi'an, China, 2019, pp. 159–163. DOI: 10.1109/ICSTW.2019.00045.
8. Ahuja H., Sujata, Batra U. Performance Enhancement in Requirement Prioritization by Using Least-Squares-Based Random Genetic Algorithm, *Innovations in Computational Intelligence*, 2018, Vol. 713, pp. 251–263. DOI: 10.1007/978-981-10-4555-4_17.
9. Leshob A., Hadaya P., Renard L. Software Requirements Prioritization with the Goal-Oriented Requirement Language, *Lecture Notes in Data Engineering and Communications Technologies*, 2020, Vol. 41, pp. 187–198. DOI: 10.1007/978-3-030-34986-8_13.
10. Hudaib A., Masadeh R., Qasem M. H. et al. Requirements Prioritization Techniques Comparison, *Modern Applied Science*, 2018, Vol. 12, No. 2, pp. 62–80. DOI: 10.5539/mas.v12n2p62.
11. Tasneem N., Zulzalil H. B., Hassan S. Enhancing Agile Software Development: A Systematic Literature Review of Requirement Prioritization and Reprioritization Techniques, *IEEE Access*, 2025, Vol. 13, pp. 32993–33034. DOI: 10.1109/access.2025.3539357.
12. Maulana M. Z. N., Siahaan D., Saikhu A. et al. Optimization Algorithm for Prioritizing Software Requirements: A Comparative Study, *7th International Seminar on Research of Information Technology and Intelligent Systems : proceedings*. Yogyakarta, 2024, pp. 284–289. DOI: 10.1109/ISRITI64779.2024.10963649.
13. Tariq A., Azam F., Anwar M. W. et al. A UML Profile for Prediction of Significant Software Requirements, *IEEE 10th Annual Conference on Information Technology, Electronics and Mobile Communication : proceedings*. Vancouver, 2019, pp. 979–984. DOI: 10.1109/IEMCON.2019.8936227.
14. Ashraf M., Tubaishat A., Al-Obeidat F. et al. Managerial Conflict Among the Software Development Team, *Lecture Notes in Networks and Systems*, 2022, Vol. 350, pp. 331–341. DOI: 10.1007/978-981-16-7618-5_29.
15. Vijayakumar S., Nethravathi P. S. Use of Natural Language Processing in Software Requirements Prioritization – A Systematic Literature Review, *International Journal of Applied Engineering and Management Letters*, 2021, Vol. 5, No. 2, pp. 152–174. DOI: 10.47992/IJAEML.2581.7000.0110.
16. Anwar R., Bashir M. B. A Systematic Literature Review of AI-Based Software Requirements Prioritization Techniques, *IEEE Access*, 2023, Vol. 11, pp. 143815–143860. DOI: 10.1109/ACCESS.2023.3343252.
17. Hujainah F., Abu Bakar R. B., Nasser A. B. et al. SRPTackle: A semi-automated requirements prioritisation technique for scalable requirements of software system projects, *Information and Software Technology*, 2021, Vol. 131, 106501. DOI: 10.1016/j.infsof.2020.106501.
18. Ali A., Hafeez Y., Hussain S. et al. Role of Requirement Prioritization Technique to Improve the Quality of Highly-Configurable Systems, *IEEE Access*, 2020, Vol. 8, pp. 27549–27573. DOI: 10.1109/ACCESS.2020.2971382.
19. Deshpande G., Motger Q., Palomares C. et al. Requirements Dependency Extraction by Integrating Active Learning with Ontology-Based Retrieval, *IEEE 28th International Confer-*

ence on Requirements Engineering : proceedings. Zurich, 2020, pp. 78–89. DOI: 10.1109/RE48521.2020.00020.
20. Saeeda H., Dong J., Wang Y. et al. A Proposed Framework for Improved Software Requirements Elicitation Process in SCRUM: Implementation by a Real-Life Norway-Based IT Project, *Journal of Software: Evolution and Process*, 2020, Vol. 32, Issue 7, e2247. DOI: 10.1002/smr.2247.
21. Samer R., Stettinger M., Felfernig A. Group Recommender User Interfaces for Improving Requirements Prioritization, *28th ACM Conference on User Modeling, Adaptation and Personalization : proceedings*. New York, 2020, pp. 221–229. DOI: 10.1145/3340631.3394851.

Received 07.07.2025.
Accepted 16.10.2025.

УДК004.421.2

ФОРМАЛІЗОВАНА МЕТОДОЛОГІЯ УЗГОДЖЕННЯ ТА АДАПТАЦІЇ ВИМОГ В ІНТЕЛЕКТУАЛЬНИХ ДІАГНОСТИЧНИХ СИСТЕМАХ

Комлева Н. О. – канд. техн. наук, доцент, завідувач кафедри програмної інженерії Національного університету «Одеська політехніка», Одеса, Україна.

Любченко В. В. – д-р техн. наук, професор кафедри програмної інженерії Національного університету «Одеська політехніка», Одеса, Україна.

АНОТАЦІЯ

Актуальність. Забезпечення узгодженості та адаптивності вимог у системах, що функціонують в умовах динамічного середовища та обмежених ресурсів, є актуальною проблемою сучасної інженерії вимог, особливо в інтелектуальних діагностичних та експертно-рішальних системах. Такі системи мають обробляти суперечливі, застарілі або неоднозначні вимоги під час роботи в складних умовах.

Мета роботи – розробка формалізованої методології аналізу та керування сумісністю вимог в інтелектуальних діагностичних системах. Запропонований підхід інтегрує логічну узгодженість, функціональну взаємодію, ресурсну здійсненність та узгодження пріоритетів для забезпечення стабільності та адаптивності системи.

Метод. Методологія реалізована як багаторівнева структура, що включає формальні представлення функціональних, нефункціональних та вимог, пов’язаних з обробкою даних. Вона використовує моделювання на основі сценаріїв, набір моделей оцінки сумісності та динамічний алгоритм інтеграції нових вимог. Процес інтеграції включає перевірку сумісності, адаптивне уточнення, експертне зважування та зворотний зв’язок у реальному часі. Застосовність методології демонструється на гіпотетичному прикладі інтелектуальної медичної діагностичної системи.

Результати. Запропонована методологія дозволяє систематично виявляти та усувати конфлікти між вимогами, забезпечуючи узгоджене виконання та ефективну пріоритизацію в умовах обмежених ресурсів. Сценарне моделювання та формалізація базових вимог створюють основу для адаптивної поведінки системи та прийняття рішень у реальному часі.

Висновки. Розроблена методологія, яка включає моделі та алгоритми, підвищує надійність інтелектуальних систем, що працюють у критичних умовах. У майбутніх дослідженнях передбачається розширення структури шляхом впровадження нечіткої логіки, методів машинного навчання та розробки програмних засобів для автоматизованого аналізу сумісності та адаптивного керування вимогами.

КЛЮЧОВІ СЛОВА: методологія, програмна інженерія, інженерія вимог, сумісність вимог, моделювання на основі сценаріїв, узгодження на основі пріоритетів.

ЛІТЕРАТУРА

1. Radwan A. M. AI-Driven Prioritization Techniques of Requirements in Agile Methodologies: A Systematic Literature Review / A. M. Radwan, M. A. Abdel-Fattah, W. Mohamed // International Journal of Advanced Computer Science and Applications. – 2024. – Vol. 15, No. 9. – P. 812–823. DOI: 10.14569/IJACSA.2024.0150983.
2. Requirement Engineering Challenges for AI-intense Systems Development / [H.-M. Heyn, E. Knauss, A. P. Muhammad et al.] – Access mode: https://arxiv.org/abs/2103.10270. DOI: 10.48550/arXiv.2103.10270.
3. How mature is requirements engineering for AI-based systems? A systematic mapping study on practices, challenges, and future research directions / [U. Habiba, M. Haug, J. Bogner et al.] // Requirements Engineering. – 2024. – Vol. 29. – P. 567–600. DOI: 10.1007/s00766-024-00432-3.
4. Wong T. Self-adaptive systems: A systematic literature review across categories and domains / T. Wong, M. Wagner, C. Treude // Information and Software Technology. – 2022. – Vol. 148. – 106934. DOI: 10.1016/j.infsof.2022.106934.
5. Tauqeer M. Requirements Prioritization – Modeling Through Dependency and Usability with Fusion of Artificial Intelligence Technique / M. Tauqeer, H. Jamil // International Journal of Innovative Science and Technology. – 2025. – Vol. 7, No. 1. – P. 146–158.
6. Towards the Selection of Optimum Requirements Prioritization Technique: A Comparative Analysis / [H. Tufail, I. Qasim, M. F. Masoo et al.] // Information Management : 5th International Conference, Cambridge, UK, 24–27 March 2019 : proceedings. – Cambridge: IEEE, 2019. – P. 227–231. DOI: 10.1109/INFOMAN.2019.8714709.
7. Requirements Dependencies-Based Test Case Prioritization for Extra-Functional Properties / [M. Abbas, I. Inayat, M. Saadatmand et al.] // Software Testing, Verification and Validation Workshops : IEEE 12th International Conference, Xi’an, China, 22–23 April 2019 : proceedings. – Los Alamitos: IEEE, 2019. – P. 159–163. DOI: 10.1109/ICSTW.2019.00045.
8. Ahuja H. Performance Enhancement in Requirement Prioritization by Using Least-Squares-Based Random Genetic Algorithm / H. Ahuja, Sujata, U. Batra // Innovations in Computational Intelligence. – 2018. – Vol. 713. – P. 251–263. DOI: 10.1007/978-981-10-4555-4_17.
9. Leshob A. Software Requirements Prioritization with the Goal-Oriented Requirement Language / A. Leshob,

P. Hadaya, L. Renard // Lecture Notes in Data Engineering and Communications Technologies. – 2020. – Vol. 41. – P. 187–198. DOI: 10.1007/978-3-030-34986-8_13.

10. Requirements Prioritization Techniques Comparison / [A. Hudaib, R. Masadeh, M. H. Qasem et al.] // Modern Applied Science. – 2018. – Vol. 12, No. 2. – P. 62–80. DOI: 10.5539/mas.v12n2p62.
11. Tasneem N. Enhancing Agile Software Development: A Systematic Literature Review of Requirement Prioritization and Reprioritization Techniques / N. Tasneem, H. B. Zulzalil, S. Hassan // IEEE Access. – 2025. – Vol. 13. – P. 32993–33034. DOI: 10.1109/access.2025.3539357.
12. Optimization Algorithm for Prioritizing Software Requirements: A Comparative Study / [M. Z. N. Maulana, D. Siahaan, A. Saikhu et al.] // Research of Information Technology and Intelligent Systems : 7th International Seminar, Yogyakarta, Indonesia, 11 December 2024 : proceedings. – Yogyakarta: IEEE, 2024. – P. 284–289. DOI: 10.1109/ISRITI64779.2024.10963649.
13. A UML Profile for Prediction of Significant Software Requirements / [A. Tariq, F. Azam, M. W. Anwar et al.] // Information Technology, Electronics and Mobile Communication : IEEE 10th Annual Conference, Vancouver, British Columbia, Canada, 17–19 October 2019 : proceedings. – Vancouver: IEEE, 2019. – P. 979–984. DOI: 10.1109/IEMCON.2019.8936227.
14. Managerial Conflict Among the Software Development Team / [M. Ashraf, A. Tubaishat, F. Al-Obeidat et al.] // Lecture Notes in Networks and Systems. – 2022. – Vol. 350. – P. 331–341. DOI: 10.1007/978-981-16-7618-5_29.
15. Vijayakumar S. Use of Natural Language Processing in Software Requirements Prioritization – A Systematic Literature Review / S. Vijayakumar, P. S. Nethravathi // International Journal of Applied Engineering and Management Letters. – 2021. – Vol. 5, No. 2. – P. 152–174. DOI: 10.47992/IJAEML.2581.7000.0110.
16. Anwar R. A Systematic Literature Review of AI-Based Software Requirements Prioritization Techniques / R. Anwar, M. B. Bashir // IEEE Access. – 2023. – Vol. 11. – P. 143815–143860. DOI: 10.1109/ACCESS.2023.3343252.
17. SRPTackle: A semi-automated requirements prioritisation technique for scalable requirements of software system projects / [F. Hujainah, R. B. Abu Bakar, A. B. Nasser et al.] // Information and Software Technology. – 2021. – Vol. 131. – 106501. DOI: 10.1016/j.infsof.2020.106501.
18. Role of Requirement Prioritization Technique to Improve the Quality of Highly-Configurable Systems / [A. Ali, Y. Hafeez, S. Hussain et al.] // IEEE Access. – 2020. – Vol. 8. – P. 27549–27573. DOI: 10.1109/ACCESS.2020.2971382.
19. Requirements Dependency Extraction by Integrating Active Learning with Ontology-Based Retrieval / [G. Deshpande, Q. Motger, C. Palomares et al.] // Requirements Engineering : IEEE 28th International Conference, Zurich, Switzerland, 31 August – 4 September 2020 : proceedings. – Los Alamitos: IEEE, 2020. – P. 78–89. DOI: 10.1109/RE48521.2020.00020.
20. A Proposed Framework for Improved Software Requirements Elicitation Process in SCRUM: Implementation by a Real-Life Norway-Based IT Project / [H. Saeeda, J. Dong, Y. Wang et al.] // Journal of Software: Evolution and Process. – 2020. – Vol. 32, Issue 7. – e2247. DOI: 10.1002/smr.2247.
21. Samer R. Group Recommender User Interfaces for Improving Requirements Prioritization / R. Samer, M. Stettinger, A. Felfernig // User Modeling, Adaptation and Personalization : 28th ACM Conference, Genoa, Italy, 14–17 July 2020 : proceedings. – New York : ACM, 2020. – P. 221–229. DOI: 10.1145/3340631.3394851.