Received May 18, 2021, accepted June 6, 2021, date of publication June 22, 2021, date of current version July 1, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3091617

Fault-Detection Tactics for Optimized Embedded Systems Efficiency

SALEH H. ALDAAJEH[1], (Member, IEEE), SAAD HAROUS[2], (Senior Member, IEEE), AND SAED ALRABAEE[1], (Senior Member, IEEE)

[1]Department of Information Systems and Security, College of Information Technology, United Arab Emirates University, Al Ain, United Arab Emirates
[2]Department of Computer Science and Software Engineering, College of Information Technology, United Arab Emirates University, Al Ain, United Arab Emirates

Corresponding authors: Saleh H. Aldaajeh (201990215@uaeu.ac.ae), Saad Harous (harous@uaeu.ac.ae), and Saed Alrabaee (salrabaee@uaeu.ac.ae)

This work was supported by the United Arab Emirates University Start-up Grant G00003261.

ABSTRACT Embedded systems operational environment poses tightened and usually conflicted design requirements. Software architects aim at introducing effective tradeoff methods to select the most appropriate design solutions to comply with the software specifications of an embedded system. When defining the software architecture for critical embedded systems it is mandatory to balance often conflicting goals to meet the different requirements in terms of resource consumption, schedulability, dependability, and security, among others. This is an engineering problem that can be addressed by employing Multiple-Criteria Decision-Making (MCDM) methods from the operational research domain. This paper combines two of these methods, Analytical Hierarchy Process (AHP) and Technique for Order Preferences by Similarity to Ideal Solution (TOPSIS), to determine the most efficient fault detection design decisions according to relevant metrics. The paper employs two algorithm efficiency metrics: run-time complexity and memory-space complexity. Furthermore, the fault-detection strategy design decisions, Ping/Echo and Heartbeat, were the subjects of this study.

INDEX TERMS Availability, AHP, design decisions, embedded systems' efficiency, tactics, TOPSIS, tradeoff.

I. INTRODUCTION

Embedded systems are widely used in modern society and can be found in almost every application of daily lives, including medical instrumentation, transportation, and national critical infrastructures, such as energy and government facilities [1], [2]. Operational environments in which embedded system are deployed are generally associated with the requirement to minimize resource consumption [3]. Accordingly, embedded systems need to operate under strict constraints associated with limited resource availability in real-time usage and necessity to operate during long-term periods, exhibiting high resilience to failures [3], [4]. The robustness and accessibility of embedded systems to authorized users has become of great importance. At present, embedded systems become more interactive as they are interconnected by networks and Internet-of-Things (IoT) applications, such as smart homes, smart farms, and fully automated manufacturers [4]. This new trend has induced new challenges in terms of the design of software used in embedded systems while providing high efficiency concerning resource consumption.

In the development of embedded systems software, quality is a key characteristic. Establishing high quality does not correspond specifically to a certain or distinct stage of the system development life cycle. On the contrary, it is considered as a continuous effort that begins by capturing quality requirements at the requirement engineering stage and continues throughout the system development life cycle [5]. The study conducted by [6] outlined that up to 45% of software quality could be achieved at the software architecture stage. Furthermore, software architecture society actively reported how the software architecture influences quality achievement in software systems [7]–[9]. Software architecture society identified several architectural design strategies and decisions (tactics) to consolidate availability in various software systems [7], [10], [11]. Architectural design decisions influenced and regulated the development

The associate editor coordinating the review of this manuscript and approving it for publication was Taehong Kim.

of quality attributes [7], [11]. Experience-based reusable architectural blocks were used to implement quality attributes in software systems [12].

To engineer important quality attributes such as dependability quality attributes to embedded systems at the software architecture development stage, it is essential to evaluate the software architecture design [5], [13]. The underlying objective of the evaluation process is to determine whether the proposed design conforms to system requirements. There are several methods to evaluate the software architecture conformity to its specifications, such as the architecture tradeoff analysis method (ATAM), architecture-level maintainability analysis (ALMA), and the software architecture analysis method (SAAM) [7]. According to the study conducted by R. Kazman *et al.*, there are two general categories that need to be considered during the evaluation of the architectural design of a software system: qualitative analysis and quantitative measurement [14]. Usually, questionnaires, checklists, and scenarios are used as means to conduct qualitative analysis, and quantitative measurements including simulations and metrics, for example, mathematical modeling and simulation tests. Moreover, quantitative measurement is focused on estimating the conformity of quality attributes to system requirements in terms of probabilities [14].

In the present study, we employ multi-criteria decision making techniques to analyze a tradeoff model concerning various architectural design decisions. The proposed tradeoff methodology integrates two multi-criteria decision analysis approaches: the analytical hierarchy process (AHP) and the technique for order preferences by similarity to ideal solution (TOPSIS). The proposed model is developed to enhance architectural design decisions tradeoff and evaluation process, considering fault-detection design decisions based on their influence on embedded systems efficiency. Furthermore, it employs algorithm-efficiency metrics (run-time complexity and memory-space complexity) to facilitate decision making in terms of optimizing resource consumption in embedded systems under real-time requirements. The fault-detection strategies ''Ping/Echo'' and ''Heartbeat'' are the subjects of this study.

A. ACRONYMS AND SYMBOLS

Table 1 provides the list of mathematical symbols used in this paper and their corresponding description.

TABLE 1. Acronyms and symbols description.

Symbol	Description
α	System Availability
MCDM	Multi-Criteria Decision Making
$MTBF$	Mean Time Between Failure
$MTTR$	Mean Time To Repair
$V(G)$	Algorithm Complexity
E	Graph Edge
N	Graph Node
A	Pair-wise Comparison Matrix
B	Normalized-Comparison Matrix
w	Weight
C	Normalized-weighted Matrix
λ_{max}	Eigen Value
$\mathbb{CR}$	Consistency Ratio
CI	Consistency Index
RI	Random Consistency Index
$v+$	Positive Ideal Solution (PIS)
$v-$	Negative Ideal Solution (NIS)
$Si+$	Distance to the Ideal Solution
$Si-$	Distance from the Ideal Solution
$Pi+$	Ideal Alternative Solution
DP	Dependability
RT	Real Time
RC	Recourse Consumption
LL	Long Operational Life
M-S	Memory-Space Complexity
R-T	Runtime Complexity
MAUT	Multi-Attribute Utility Theory
CBR	Case-based Reasoning
DEA	Data Envelopment Analysis
SMART	Simple Multi-Attribute Rating Technique
GP	Goal Programming
ELECTRE	Elimination and Choice Translating REality
PROMETHEE	Preference Raking Organization METHod for Enrichment of Evaluations
SAW	Simple Additive Weighting

II. BACKGROUND AND MOTIVATION

The design and development of embedded systems operating in IoT environments requires thorough analysis and evaluation. Numerous research studies have been conducted in the areas of software architecture, availability, and tradeoff design decisions concerning embedded systems. The following sections provide a comprehensive literature review of the previous work reported in the aforementioned areas.

A. EMBEDDED SYSTEMS SOFTWARE ARCHITECTURE

Designing the architecture of an embedded system software is a challenging task due to conflicted operational environment requirements, in particular, such as availability, resource and memory consumption, and the necessity to operate in real time. To consolidate the implementation of embedded system software specifications, software architects need to select the best candidate architectural design solution.

The second stage in the software system development life cycle is to design the software architecture of a system. According to L. Bass *et al.*, software architecture can be described as a structure or structures of a system including software components, the externally visible properties of those components, and the relationships among them [6]. The implementation of quality attributes in software architecture is performed at two levels: requirements and solutions. Several frameworks can be used to identify, specify, and categorize quality requirements [6], [11], [14]–[16]. These frameworks are applied to define quality attributes in a form of general and concrete scenarios, which can be considered in evaluating the conformity of software architecture to the specified functional and non-functional requirements [6]. General scenarios provide a template to formulate concrete scenarios to identify quality attributes. Concrete scenarios are utilized as a means to develop a detailed description of the requirements associated with quality attributes, thereby enabling software architects to investigate whether the

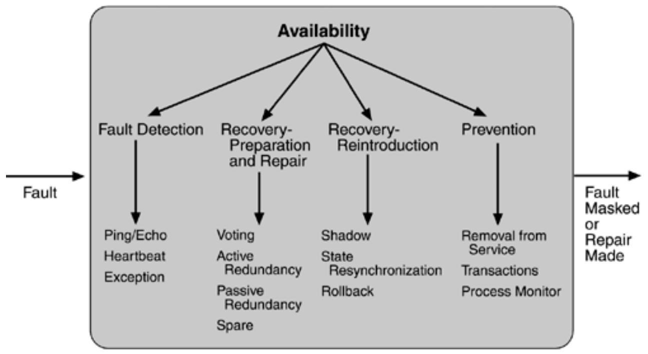

FIGURE 1. Availability architecture design strategies and decisions [7].

selected architectural strategies and design decisions satisfy and comply with the desired functional and non-functional requirements [17].

Software architecture society has been active investigating various available architectural strategies and design decisions that can be used to implement different quality attributes in a software system [6]–[8], [10]–[12], [18], [19]. Generally, these architectural design decisions are based on experienced-based solutions [19]–[21]. Figure 1 represents availability architecture strategies and their corresponding design decisions [6].

The semantic specification of architecture design decisions can be realized by adopting the role-based meta-modeling language (RBML) [18], [22]. The major benefits of utilizing RBML can be summarized as follows: (1) RBML facilitates the use of architectural tactics at a model-view level; (2) RBML is capable of capturing the generic structure of a tactic; (3) RBML enables instantiating a tactic in various structures through realizing the multiplicity of roles, thereby facilitating the reuse of architectural tactics [22], [23].

B. EMBEDDED SYSTEMS AVAILABILITY

Embedded systems software operating in a safety-critical environment are expected to provide high availability for long operational periods. Most importantly, embedded systems are designed to be efficient and therefore they are designed with certain limitations on resources consumption. Due to the considerable importance of this aspect in the development of software systems, research society has been actively investigating this question. According to A. Avižienis *et al.*, availability can be defined as a runtime quality attribute that describes the readiness of a system to provide services for authorized users [17]. In the same study, dependability is determined as a subset of quality attributes including availability. Information security society considers system availability as one of the three major security attributes (availability, integrity, and confidentiality). From this viewpoint, availability is defined as the system capability of granting authorized users access to the system itself and required resources [16]. The International Organization of Standards (ISO) ISO/IEC 25010 model considers availability as a sub-quality factor contributing to the reliability of systems. Specifically, in this model, availability is described as a degree to which the system is operational and accessible when required for use [20]. Most importantly, the achievement of dependability quality attributes, including availability, in the software development process is strongly related to the software architecture [24].

From practical perspective, steady-state availability serves as a measurement of a system up-time during a sufficiently long operational period (90 days, one year, entire project, etc.) [18]. A well-known expression used to derive a steady-state availability of a system is formulated as follows:

$$\alpha = \frac{\mathbf{MTBF}}{\mathbf{MTBF} + \mathbf{MTTR}}$$

As shown in Figure 1, availability can be realized by implementing various design strategies, such as fault detection, recovery-and-prevention, recovery reintroduction, and prevention.

The proposed model was applied to the selection of a fault-detection architecture strategy as a case study. Fault detection capability in embedded systems software is crucial to maintain availability as it underlies overall system reliability. By detecting faulty systems or system components, the likelihood of achieving a higher degree of steady-state availability in a system can be substantially increased [18]. The fault detection architecture strategy relies on three design decisions: Ping/Echo, Heartbeat, and Exception, as represented in Figure 1. These design decisions can be summarized as follows [18], [22]:

- Ping/Echo: An asynchronous request/respond message pair is exchanged between nodes/components to determine reachability and a round-trip delay corresponding to the paths associated with a component in question. Therefore, this tactic allows checking the availability of a component by sending ping messages.
- Heartbeat: A periodic message is exchanged between the system monitor and the process. It provides a system with indications on when a fault is incurred in the process. Accordingly, this tactic allows checking the availability of a component by listening to heartbeat messages sent from components.
- Exception: This tactic is aimed at detecting system conditions that do not comply with the normal flow of execution. Then, the system raises an exception as soon as it detects a fault. In this way, such a design decision is used to recognize and handle faults. Usually, it is implemented in combination with other fault detection design decisions, such as Ping/Echo or Heartbeat.

Figures 2 and 3 illustrate the RBML specifications for the fault-detection architecture strategy design decisions Ping/Echo and Heartbeat, respectively. It should be noted that both aforementioned design decisions are used in integration with the one called Exception to implement fault detection and can be utilized in various embedded control and monitoring systems.

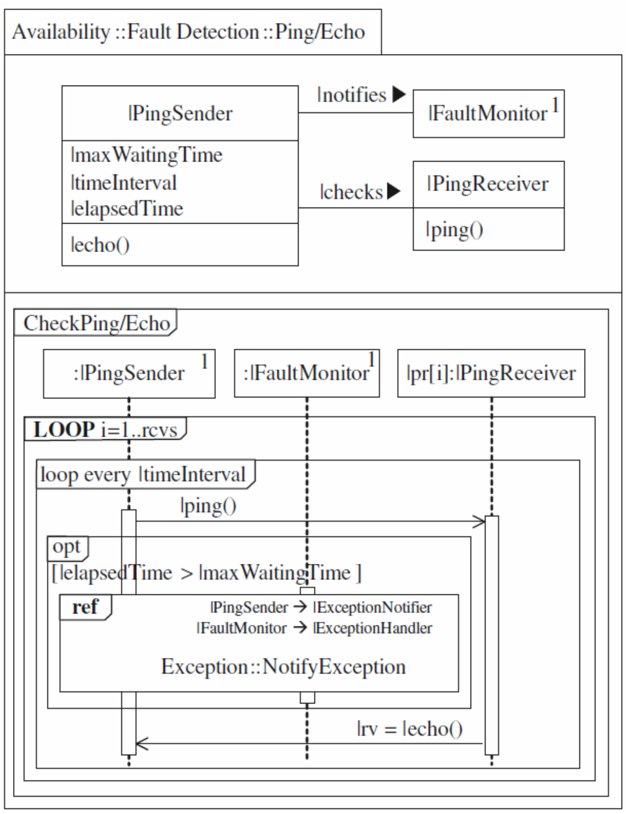

FIGURE 2. RBML specification for the Ping/Echo design decision [18].

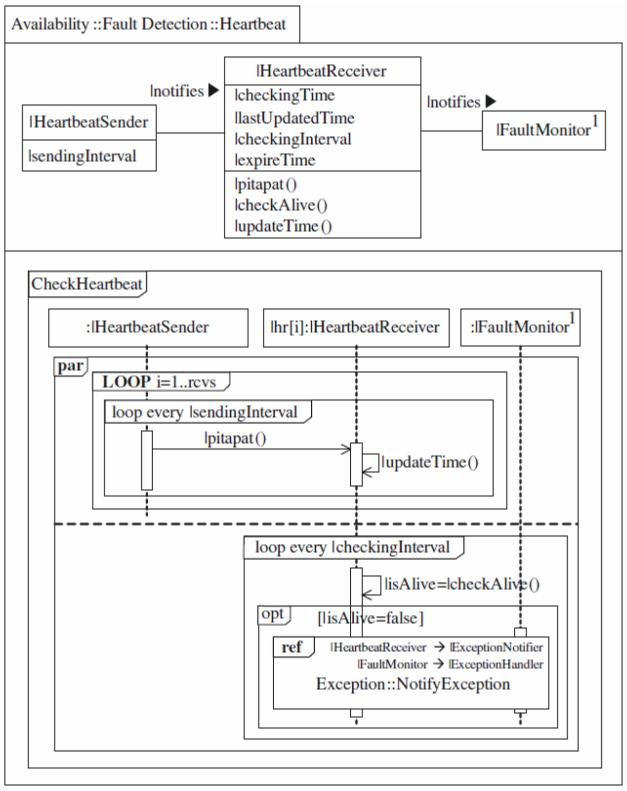

FIGURE 3. RBML specification for the heartbeat design decision [18].

C. TRADEOFF APPROACHES

Generally, tradeoffs are made subconsciously on daily basis in all fields of human activity. Tradeoffs are usually made based on three main approaches [15]:

- Experience-based approach. This approach relies on the experience of an individual in conducting a tradeoff process and is considered to be risky as the produced output is subjective. Moreover, it is difficult to apply to complex situations. This approach is widely utilized in the qualitative analysis methodology to evaluate software architecture.
- Model-based approach. This approach implies constructing a model (for example, a geographical one) to illustrate influential relationships among tradeoff subjects/ elements. The underlying reason to utilize such models is to simplify and evaluate interrelationships between various elements and how the interrelationship is affected among two elements or more.
- Mathematical reasoning approach. This approach relies on mathematical formulas and metrics to construct and represent tradeoff processes.

The complexity of a tradeoff process depends on a number of conflicting factors [10]. Multi-Criteria Decision-Making (MCDM) addresses the need for a numerate structure and provides a foundation for selecting, sorting, and prioritizing alternatives based on their appropriateness [25]. Due to its pervasive application in the decision making process in various domains, the MCDM has induced several methods such as Multi-Attribute Utility Theory (MAUT), AHP, Case-based Reasoning (CBR), Data Envelopment Analysis (DEA), Fuzzy set Theory, Simple Multi-Attribute Rating Technique (SMART), Goal Programming (GP), Elimination and Choice Translating REality (ELECTRE), Preference Ranking Organization METHod for Enrichment of Evaluations (PROMETHEE), Simple Additive Weighting (SAW), and TOPSIS [26].

According to [26], AHP is easy to use and scalable. Its hierarchy structure can be easily adjusted to fit many sized problems and it is not data intensive. AHP drawback appears when interdependence exists between criteria and alternatives; where it may cause inconsistencies between judgment and ranking criteria. However, AHP areas of application is not limited to performance-type problems but also can be found in other operational research domains such as resources management and planning. TOPSIS implementation process has standard steps. Therefore, TOPSIS method is considered to be a straight-forward process and easy to use and program. By using the Euclidean Distance as a measure of most suitable solution, it neglects the correlation of attributes making it difficult to weight and maintain consistency of judgment. Nevertheless, It has a wide range of applications in various domains such as engineering, manufacturing and resources management.

Each MCDM method has its own shortcomings. Hence, MCDM methods are combined simultaneously to overcome these shortcomings and to identify the most suitable solution/decision. Several studies have combined or integrated MCDM methods such as [27]–[35]. However, MCDM methods are compared in terms of their areas of application, advantages, and disadvantages in the study conducted by [26].

Due to the multi-criteria nature for selecting most suitable design decisions/ tactics, MCDM is an effective approach to resolve this type of selection challenges. In particular, the analytical model in combination with the integration between AHP and TOPSIS support determining the most suitable design decision/ tactic based on specific goals and objectives [26]. This study employs integrated AHP-TOPSIS techniques taking into consideration both qualitative and quantitative factors. In this regard, AHP can be very useful in implicating several decision making possibilities with multiple conflicting criteria to arrive at a consensus in the decision making process. On the other hand, TOPSIS technique is used to calculate the alternatives ratings based on their suitability. Moreover, both AHP and TOPSIS are integrated to create a tradeoff framework and identify most suitable fault-detection design decision given their implications on embedded systems' efficiency.

III. TRADEOFF MODEL

Software architects need to investigate and to analyze the implications and the influence of each design decision. For instance, fault detection design decisions, such as Ping/Echo and Heartbeat, may have the same influence in terms of contributing to the implementation of availability of an embedded system software installed in a safety-critical environment. At the same time, they may have different implications on resource consumption.

Software architecture evaluation methods are heavily dependent on expert's judgment. The proposed tradeoff model is adopted from operational research domain and uses two (MCDM) methods. Namely, these methods are AHP and TOPSIS. The AHP method is applied to score and define the share of importance based on the set of decision criteria with relative weights [36]. The TOPSIS method is used to rank an ideal solution from the list of alternatives [37].

We estimate the resources in terms of processing efforts and memory space that are required to implement design decisions using a specifically developed algorithm. This algorithm has been formulated based on architectural design using RBML, as reported in [22]. Moreover, we evaluate the generated algorithm for defining design decisions in terms of consistency and testability by considering McCabe cyclomatic complexity [38]. The generated algorithm is further analyzed in terms of run-time complexity and memory-space complexity [38], [39]. Figure 4 depicts the implementation framework for the multi-criteria tradeoff model.

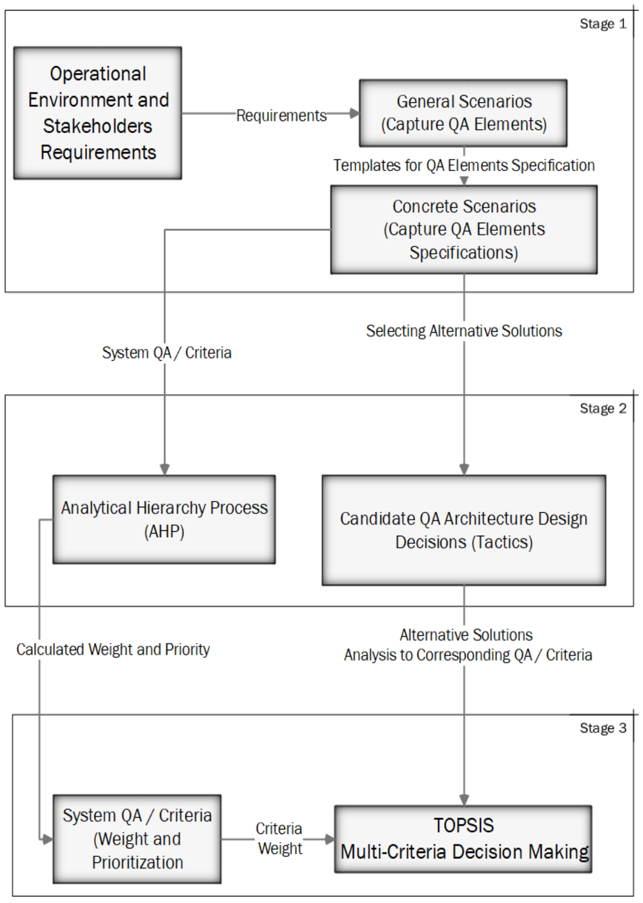
Stage 1
Operational Environment and Stakeholders Requirements
Requirements
General Scenarios (Capture QA Elements)
Templates for QA Elements Specification
Concrete Scenarios (Capture QA Elements Specifications)
System QA / Criteria
Selecting Alternative Solutions
Stage 2
Analytical Hierarchy Process (AHP)
Candidate QA Architecture Design Decisions (Tactics)
Calculated Weight and Priority
Alternative Solutions Analysis to Corresponding QA / Criteria
Stage 3
System QA / Criteria (Weight and Prioritization)
Criteria Weight
TOPSIS Multi-Criteria Decision Making

FIGURE 4. Implementation framework.

A. DEVELOPING AND VALIDATING DESIGN DECISIONS

We create a set of general and concrete scenarios to identify the quality attributes associated with availability. General scenarios provide the means and templates to develop concrete scenarios corresponding to specific quality attributes. The development of such scenarios can be facilitated using technical information or relying on the requirements of a system [7], [11]. Moreover, technical information provided by a supervisory control and data acquisition (SCADA) system implemented in a smart-grid is used for this purpose.

- General Scenario: *A SCADA system deployed in a smart grid (environment) is designed to manage power distribution using engineering planning and budgeting functions (stimulus) by providing access to engineers (the source of stimulus). Therefore, the SCADA system must provide access to authorized users with the response rate as close as possible to 100%, while providing planned or unplanned maintenance (stimulus) throughout its operation time.*
- Concrete Scenario: *A SCADA system in a smart gird (environment) is designed to control power distribution and to provide access to engineers in a work station Alpha (the source of stimulus) in terms of power distribution and data management (stimulus).*

TABLE 2. Runtime complexity and memory-space complexity calculation for (Ping/Echo).

Ping			
Line	Code	Run-time Complexity	Memory-Space Complexity
2	***for*** *i=1 to Rcvrs* ***do***	(n)	n
3	***if*** *Time=TimeInterval* ***then***	1	1
4	***Ping(Rcvrs)***	1	1
5	***elapsedTime=Reset)***	1	1
6-14	***do*** ... ***while (Condition)***	$n * (n)$	1
15-17	***Else*** *Time, i* $\rightarrow$ *Increment*	1	1
Ping analysis		$\theta(n^2)$	$\theta(n)$
Echo/Reply sender			
18	***if*** *Ping (Received) == True* ***then***	1	1
19	***Echo (Sender)***	1	1
Echo analysis		$\theta(1)$	$\theta(1)$
Ping/Echo algorithm analysis		$\theta(n^2)$	$\theta(n)$

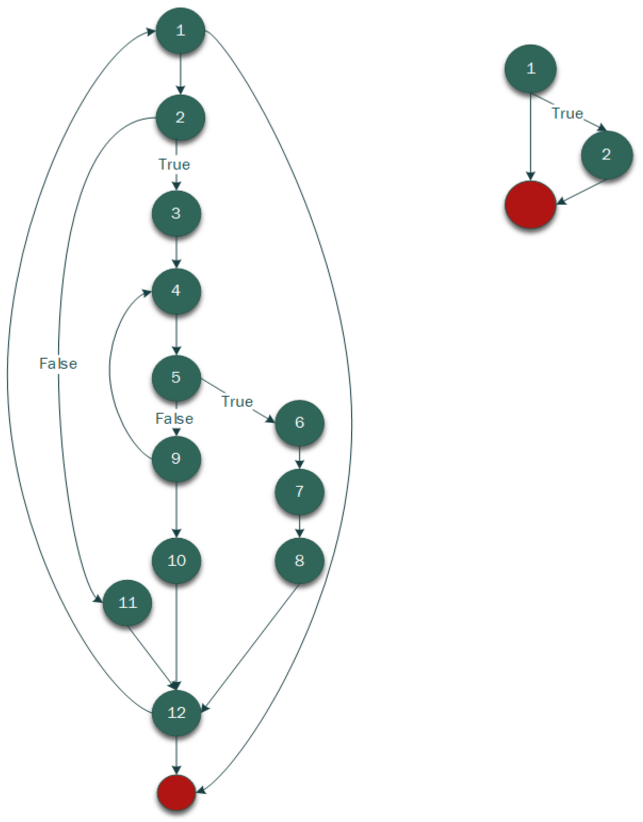

FIGURE 5. Flow-graph for the design decision (Ping/Echo).

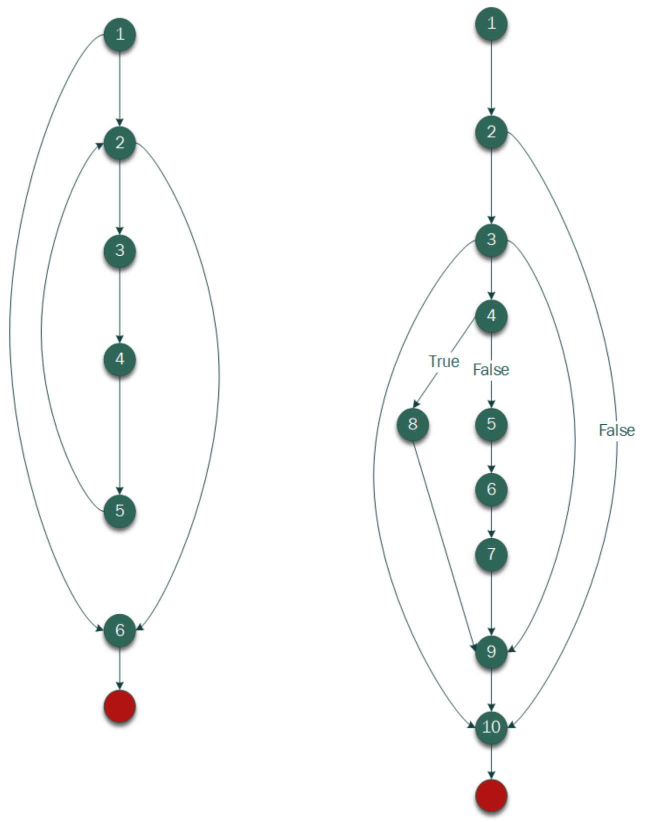

FIGURE 6. Flow-graph for the design decision (Heartbeat).

A power-distribution system and its components are inspected every three minutes, and faults are constantly reported to the engineers controlling the workstation Alpha (response measure).

As shown in Figure 1, software architecture society proposes three design decisions to implement a fault detection strategy in embedded system software. Both design decisions Ping/Echo and Heartbeat are used integrated with the design decision Exception.

To the best of authors' knowledge, there exists no study that is focused on investigating an algorithm for fault detection design decisions. The related algorithms are based on the design semantic specification introduced in the study conducted by D. Kim [22], as shown in Algorithm 1 and Algorithm 2 for Ping/Echo and Heartbeat, respectively. Accordingly, we provide the results of run-time complexity and memory-space complexity calculations in Tables 2 and 3 for Ping/Echo and Heartbeat, respectively.

TABLE 3. Run-time complexity and memory-space complexity calculation for (Heartbeat).

Heatbeat (Sending)			
Line	Code	Run-time Complexity	Memory-Space Complexity
2	***if** SendingInterval≠ expireTime **then***	(1)	1
3-6	***for** i =1 to Rcvrs **do***	n	n
7-8	***else** i = increment*	1	1
Heartbeat sending analysis		$\theta(n)$	$\theta(n)$
Heartbeat (Listening)			
9	***if** CheckingInterval= CheckingTime **then***	(1)	1
10	***for** i =1 to Senders **do***	n	n
11	***isAlive() =ChecAlive(Sender)***	1	1
12-16	***if** isAlive= True **then***	n	1
17-18	***Else (i) ← increment***	1	1
19-20	***Else (CheckingInterval) ← Decrements***	1	1
Heartbeat (Listening) analysis		$\theta(n)$	$\theta(n)$
Heartbeat algorithm analysis		$\theta(n)$	$\theta(n)$

TABLE 4. Cyclomatic complexity measurement scale and description.

$V(G)$	Description	Cost
1-10	Well-structured algorithm with high testability	Requires less costs and efforts
10-20	Complex algorithm with medium testability	Costs and efforts are medium
20-40	Very complex algorithm with low testability	Costs and efforts are high
>40	Not testable	Unfeasible - Very high costs and efforts

Both design decisions are further analyzed to evaluate their complexity considering McCabe cyclomatic complexity [38], [39]. Figures 5 and 6 represent independent paths and flow graphs corresponding to both considered algorithms. Cyclomatic complexity can be calculated using the following equation:

$$V(G) = E - N + 2 \tag{1}$$

The underlying reason to apply the cyclomatic complexity metric to the proposed algorithms is to assure that they are consistent and testable. Table 4 provides the cyclomatic complexity $V(G)$ measurement intervals/ranges and their meaning. Cyclomatic complexity for both algorithms is calculated as follows:

- "Ping/Echo": $V(G) = E - N + 2 = 20 - 16 + 2 = 6$
- "Heartbeat": $V(G) = E - N + 2 = 22 - 18 + 2 = 6$

B. DESIGN DECISIONS TRADEOFF ANALYSIS

The second step in implementing the proposed tradeoff model implies applying AHP to obtain the weights for the predefined tradeoff criteria. The criteria of embedded system software correspond to the common requirements proposed by I. Crnkovic: dependability, real time execution, resource consumption, and long operational life [3]. The generated criteria weights are then used in TOPSIS. The steps of applying AHP can be summarized as follow:

Algorithm 1: Ping/Echo Design Decision

```
1  Faults detected in an unresponsive component are
       reported Data: MaxWaitingTime = value,
       TimeInterval = value, elapsedTime = 0
   /* Ping Receivers (n)                          */
2  for $i = 1$ to $Rcvrs$ do
3      if $Time == TimeInterval$ then
4          Ping (Rcvr)
5          elapsedTime → $Reset$
6          do
7              if $elapsedTime > MaxWaitingTime$ then
8                  PingSender → $ExceptionNotifier$
9                  FaultMonitor → $ExceptionHandler$
10                 Exception (NotifyException)
11                 Break
12             else
13                 elapsedTime ← $increament$
14         while $Echo(Rcvr) = False$
15     else
16         $Time \rightarrow Increment$
17     $i \rightarrow Increment$
   /* Reply Sender                                */
18 if $Ping(received)$ is $True$ then
19     $Echo(ReplySender)$
```

1) Modeling a hierarchical structure that illustrates the considered problem (a tradeoff case) to perform design decision selection, with the goal at the top

TABLE 5. Random consistency index.

Elements (n)	1	2	3	4	5	6	7	8	9	10
Ratio Index (RI)	0	0	0.52	0.89	1.11	1.25	1.35	1.4	1.45	1.49

TABLE 6. Relative-importance saaty scale.

Importance	Definition	Description
1	Equal importance	Both elements have equal contribution to objective
3	Moderate importance	Moderate advantage of one element compared to other
5	Strong importance	Strong favoring of one element compared to other
7	Very Strong importance	One element is strongly favored and has domination in practice compared to the other element.
9	Extreme importance	One element is favored in comparison with the other, based on strongly proved evidences and facts.
2,4,6,8	Intermediate Values	
1/3, 1/5	Inverse Comparison Values	
1/7, 1/9	Inverse Comparison Values	

Algorithm 2: *Heartbeat Design Decision*

```
1  Faults detected in n components are reported
       Data: CheckingTime = value,
       LastUpdatedTime = 0, CheckingInterval =
       value, expireTime = value
   /* Heartbeat sender */
2  if $SendingInterval \neq expireTime$ then
3      for $i = 1$ to $Rcvrs$ do
4          $pitapat\ (Rcvr)$
5          $UpdateTime(lastUpdatedTime)$
6          $i \leftarrow increment$
7  else
8      $SendingInterval \leftarrow increment$
   /* Heartbeat receiver */
9  if $Checking\ Interval = CheckingTime$ then
10     for $i = 1$ to $Senders$ do
11         $isAlive = CheckAlive(Sender)$
12         if $isAlive$ is $false$ then
13             $Heartbeat \rightarrow ExceptionNotifier$
14             $FaultMonitor \rightarrow ExceptionHandler$
15             $Exception\ (NotifyException)$
16             $i \leftarrow increments$
17         else
18             $i \leftarrow increments$
19 else
20     $Checking\ Interval \rightarrow CheckingTime$
21
```

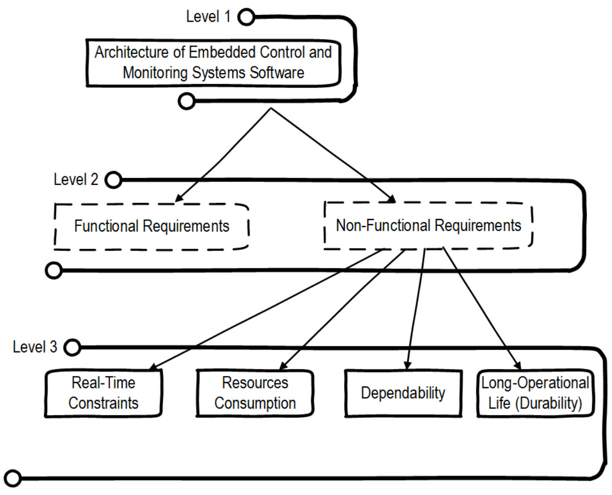
Level 1
Architecture of Embedded Control and Monitoring Systems Software
Level 2
Functional Requirements
Non-Functional Requirements
Level 3
Real-Time Constraints
Resources Consumption
Dependability
Long-Operational Life (Durability)

FIGURE 7. Three-level hierarchical structure of AHP.

of the hierarchical structure, the attributes/criteria of embedded system software at the second level, and alternative candidate solutions (design decisions) at the lowest level, as depicted in Figure 7.

2) Determining a relative importance of different attributes /criteria with respect to the stated goal. This, however, can be realized by formulating a pair-wise comparison matrix and measuring relative importance on a pre-defined *Saaty* scale [40]. Table 6 provides the relative importance scale measurements.

The importance scale for a concrete scenario in a smart-grid SCADA system can be defined as follows:

- Resource consumption **(RC)** is considered to have the highest importance compared to the other decision criteria. It has a very strong importance, strong importance, and extreme importance with respect to dependability **(DP)**, real-time execution **(RT)**, and long operational life **(LL)**, respectively.
- Dependability is considered to have moderate importance and very strong importance compared to **R-T** and **LL**, respectively.
- Real-time execution is considered to have strong importance compared to **(LL)** long operational life.

Thereafter, the results of relative importance assessment are inserted into a pair-wise matrix and are read row-wise, as shown in Table 7.

TABLE 7. Comparing the relative importance of decision criteria.

	RC	DP	RT	LL
Resource consumption **(RC)**	1	7	5	9
Dependability **(DP)**	0.14	1	1	5
Real-time execution **(RT)**	0.2	1	1	5
Long operational life **(LL)**	0.11	0.2	0.2	1

3) Conducting sensitivity analysis. The AHP method requires complying with four axioms formulated as follows [41], [42]:
   - Reciprocity: In case when element x is n times more important than y, then element y is of an $(1/n)$ importance of element x.
   - Homogeneity: Only equally comparable elements can be compared, in contrast to heterogeneous elements.
   - Dependence: Correlating a group of elements from one level with the other elements from a higher level.
   - Expectations: Changes in the hierarchy model should be reflected in the data presented in the matrix.

C. AHP ANALYSIS

AHP is applied to obtain the weights for the predefined tradeoff criteria for optimized embedded systems efficiency. A pair-wise comparison matrix is shown in equation 2. Its dimension depends on the number of decision criteria $(n = 4)$.

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix} \tag{2}$$

The values of the pair-wise comparison matrix are expressed in equation 3. Considering the reciprocity axiom, they are calculated according to equation 3, the third case, where (v) is considered based on the Saat relative importance scale, as shown in table 6.

$$A_{ij} = vi < j1\forall \quad i = j\frac{1}{A_{ji}}i > j \tag{3}$$

A normalized matrix (B) is computed using equation 4.

$$B_{ij} = \frac{A_{ij}}{\sum_{i=1}^{n} A_{ij}} \tag{4}$$

Weights are calculated using an eigen vector from the normalized matrix B by computing the arithmetic mean for each row of the matrix using equation 5.

$$w_i = \frac{\sum_{j=1}^{n} b_{ij}}{n} \tag{5}$$

To examine the extent of consistency in matrix A, a quantifiable measure is considered. The resulting normalized weight matrix is represented in matrix equation 6, and the consistency ratio is obtained according to equation 7.

$$C = \begin{bmatrix} \frac{w_1}{w_1} & \frac{w_1}{w_2} & \cdots & \frac{w_1}{n} \\ \frac{w_2}{w_1} & \frac{w_2}{w_2} & \cdots & \frac{w_2}{w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{w_n}{w_1} & \frac{w_n}{w_2} & \cdots & \frac{w_n}{w_n} \end{bmatrix} \tag{6}$$

$$\begin{bmatrix} \frac{w_1}{w_1} & \frac{w_1}{w_2} & \cdots & \frac{w_1}{n} \\ \frac{w_2}{w_1} & \frac{w_2}{w_2} & \cdots & \frac{w_2}{w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{w_n}{w_1} & \frac{w_n}{w_2} & \cdots & \frac{w_n}{w_n} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = n \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \tag{7}$$

In the case when Matrix A is inconsistent, the relative weight w_i is approximated utilizing the average of n elements in row i corresponding to the normalized matrix C. If the greatest eigenvalue is equal to the dimension of the matrix, this indicates that the matrix is consistent. Therefore, the distance between λ_{max} and n can be used as a measure of inconsistency (namely, the closer λ_{max} to n, the more consistent is the pair-wise comparison. Assuming that $\bar{w}$ is the computed average vector, the consistency relation can be formulation as $C\bar{w} = \lambda_{max}\bar{w}, \lambda_{max} \geq n$. However, the consistency ratio is computed as per equation 8. In the case when the calculated $(\mathbb{CR} \leqslant 1.0)$, the level of consistency is considered to be at an acceptable rate. Otherwise, a decision maker is required to re-evaluate the elements of A_{ij} in matrix A.

$$\mathbb{CR} = \frac{CI}{RI} \tag{8}$$

CI is the consistency index of the matrix and is calculated according to equation 9.

$$CI = \frac{\lambda_{max} - n}{n - 1} \tag{9}$$

RI is obtained based on the random consistency index [43], [44]. Table 5 provides the values of this index.

One of the characteristic features of AHP is that it allows comparing the considered criteria between each other, resulting in the formulation of a pair-wise comparison matrix. The AHP method progresses by determining global and local preferences (*weights*) based on a pair-wise comparison matrix and calculates compliance factor (*CI*). The final step is to create the final ranking of accepted alternatives [37]. Tables 7 and 9 provide the pair-wise comparison matrix and the normalized matrix, respectively, obtained using the AHP method.

D. TOPSIS ANALYSIS

TOPSIS is one of the methods used to identify an optimal solution by solving multiple criteria decisions or

TABLE 8. Design decisions relative importance for various criteria.

	RC	DP	RT	LL
Ping/Echo R-T Complexity θn^2	1	1	1	1
Ping/Echo M-S Complexity θn	1	1	1	1
Heartbeat R-T Complexity θn	5	1	7	1
Heartbeat M-S Complexity θn	1	1	1	1

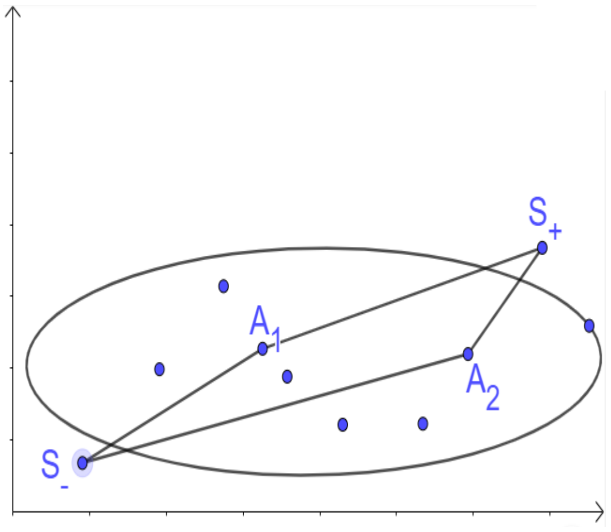

FIGURE 8. TOPSIS illustrating PIS and NIS.

problems [23], [27], [37], [45], [46]. This method is applied to assess and evaluate the appropriateness of alternative candidate solutions by simultaneously measuring their proximity to the positive ideal solution (PIS) and to the negative ideal solution (NIS). PIS is an alternative solution that is mostly preferred by a decision maker, while NIS corresponds to the least preferred solution [37], [42]. The preference order is then formulated according to the relative closeness of the alternative solutions to PIS and is considered as a scalar criterion that combines these two distance measures. TOPSIS requires the sufficient understanding of the complexity of a decision or a problem. Furthermore, the evaluation criteria, criteria weights, alternatives, and their resolution levels are analyzed and are precisely defined in the form of a decision matrix. TOPSIS relies on two hypotheses:

1) Each attribute in a decision matrix takes either monotonically increasing or monotonically decreasing utility.
2) A set of weights is determined for attributes.

Figure 8 illustrates the scheme underlying the TOPSIS method. Below, we describe the steps executed while applying TOPSIS to the proposed tradeoff model concerning the fault-detection design decisions.

- First step: Constructing a normalized decision matrix using equation 10.

$$n_{ij} = \frac{r_{i,j}}{\sqrt{\sum_{i=1}^{m} r_{ij}^2}} \tag{10}$$

- Second step: Creating a weighted dimensionless matrix with w vector obtained from AHP. N_D is a matrix in which the rates of indices are dimensionless and comparable, and $W_{n\times n}$ is a diagonal matrix in which only elements of its original diameter have non-zero values.
  The second step is executed by applying equation 11.

$$\mathbf{V} = N_D \times W_{n\times n} = \begin{bmatrix} v_{11} & \cdots & v_{1j} & v_{1n} \\ v_{21} & \cdots & v_{2j} & v_{2n} \\ \vdots & \ddots & \vdots & \vdots \\ v_{m1} & \cdots & v_{mj} & v_{mn} \end{bmatrix} \tag{11}$$

- Third step: Calculating PIS and NIS $v+$ and $v-$ by finding the maximum beneficial value for the best ideal solution, and the minimum value for the worst one. Equations 12 and 13 are used to define PIS and NIS.

$$\begin{aligned} v^+ &= \{(maxV_{ij} | j \in J' | i = 1, 2, 3 \cdots m\} \\ &= \{V_1^+, V_2^+, \cdots V_j^+, \ldots V_n^+\} \end{aligned} \tag{12}$$

$$\begin{aligned} v^- &= \{(minV_{ij} | j \in J' | i = 1, 2, 3 \cdots m\} \\ &= \{V_1^-, V_2^-, \cdots V_j^-, \ldots V_n^-\} \end{aligned} \tag{13}$$

- Fourth step: Calculating the distance between an alternative solution and the ideal one using the Euclidean method, as shown in equation 14.

$$\begin{aligned} S_i^+ &= \sqrt{\sum_{j=1}^{n} (v_{i,j} - v_j^+)^2}; \quad i = 1, 2, 3 \cdots m \\ S_i^- &= \sqrt{\sum_{j=1}^{n} (v_{i,j} - v_j^-)^2}; \quad i = 1, 2, 3 \cdots m \end{aligned} \tag{14}$$

- Fifth step: Calculating the relative closeness to the ideal solution utilizing equation 15. Ranking alternatives are based on the calculated value of P_i^+. The maximum value is considered to be the best alternative.

$$P_i^+ = \frac{S_i^-}{S_i^+ + S_i^-}; 0 \leqslant P_i^+ \leqslant 1; \quad i = 1, 2, 3 \cdots m \tag{15}$$

IV. TRADEOFF ANALYSIS

The generated algorithms corresponding to the design decisions Ping/Echo and Heartbeat indicated that the former provided worse run-time complexity $\theta(n^2)$, while the run-time complexity of the latter was equal to $\theta(n)$. Both design decisions exhibited the same space complexity equal to $\theta(n)$.

According to the cyclomatic complexity metric interval/rages provided in Table 4, the Ping/Echo design decision was considered to be well-structured with the high testability of $V(G) = 6$. In turn, the Heartbeat design decision was also observed to be well-structured with the high testability of $V(G) = 6$. The consistency of a normalized matrix was calculated using eigenvalue λ_{max} that was compared to the dimension of the normalized matrix (n). The distance between (λ_{max}) and ((n)) could be utilized to determine an

TABLE 9. AHP application on multiple-decision criteria for embedded systems software.

	RC	DP	RT	LL	w_{ij}	RC	DP	RT	LL	$\sum w_{ij}$	$\frac{Weight}{n}$	$\frac{\lambda_{max}-n}{n-1}$	CI	$\mathbb{CR}$
RC	0.688	0.82	0.543	0.409	**0.615**	0.615	1.505	0.641	0.374	3.135	5.95			
DP	0.98	0.117	0.326	0.318	**0.215**	0.088	0.215	0.385	0.291	0.978	4.552			
RT	0.138	0.039	0.109	0.277	**0.128**	0.123	0.072	0.128	0.208	0.531	4.141	4.517	0.172	0.193
LL	0.076	0.023	0.022	0.045	**0.042**	0.068	0.043	0.026	0.042	0.178	4.278			
$\sum$	1.453	8.533	9.2	22.0										

TABLE 10. TOPSIS application on fault detection design decisions (Ping and Echo, and Heartbeat).

	Normalized matrix				Weighted normalized matrix				Distance		$Pi+$
	RC	DP	RT	LL	RC	DP	RT	LL	$Si+$	$Si-$	
	0.615	0.215	0.128	0.042							
Ping/Echo R-T Complexity	0.189	0.5	0.139	0.5	0.116	0.108	0.178	0.021	0.477	0	0
Heartbeat R-T Complexity	0.945	0.5	0.971	0.5	0.581	0.108	0.124	0.021	0	0.477	1
Ping/Echo M-S Complexity	0.189	0.5	0.189	0.5	0.116	0.108	0.178	0.021	0.477	0	0
Heartbeat M-S Complexity	0.116	0.108	0.178	0.021	0.116	0.108	0.178	0.021	0.477	0	0
Negative Ideal Solution (NIS) $(v-)$					0.116	0.108	0.0178	0.021			
Positive Ideal Solution (PIS) $(v+)$					0.581	0.108	0.1243	0.021			

inconsistency (namely, the closer was (λ_{max}) to (n), the better was the consistency of the matrix). The consistency index corresponding to the normalized pair-wise comparison matrix was found to be **(0.172)**. The consistency ratio was computed for normalized matrix scoring **(0.0193)** and was considered to be acceptable as ($\mathbb{CR}$:0.0193 $\leqslant$ 1.0).

The criterion weight was obtained based on the normalized matrix w_i. It was then utilized at the evaluation step while implementing the TOPSIS method. It was assumed that the Ping/Echo and Heartbeat design decisions exhibited a neutral contribution with respect to the criterion "Dependability". Both design decisions enabled the realization of a fault-detection architectural strategy and consolidated the realization of the dependability quality attribute "Availability". Using the proposed algorithm, it was found that both design decisions, Ping/Echo and Heartbeat, had a neutral contribution with respect to the criterion "Long operational life". Both design decisions were assigned the value of (1), Table 8. Table 10 represents the application of the TOPSIS method and provides the normalized matrix and the calculated distance between the alternative candidate solutions (the Ping/Echo and Heartbeat design decisions) and a hypothetical ideal solution obtained using the Euclidean method (Si^+&Si^-). (Pi^+) in the ranking process. It was found that the Heartbeat design decision was a better option with the value of run-time complexity equal to ($Pi^+ = \mathbf{1}$) in comparison with that of ($Pi^+ = \mathbf{0}$) in the case of the Ping/Echo design decision. It was also found that both design decisions exhibited the same value of memory-space complexity equal to Pi^+ value.

V. CONCLUSION

In the present study, we have employed multiple-criteria decision making techniques (AHP and TOPSIS) to prioritize fault-detection design decisions influence on embedded systems efficiency. The application AHP was focused on estimating relative contribution of each criteria included in the tradeoff process towards the goal (weights). While TOPSIS was applied to obtain the actual ranking (prioritization) using the provided weights obtained from AHP. Hence, calculating a quantifiable evidence to prioritize the most "ideal" fault-detection design decision to optimize embedded systems' efficiency in terms of resources consumption.

Tradeoff process was applied to a case study considering multiple criteria imposed on embedded systems. The obtained results indicated that the proposed model could be successfully employed as a practical tool to identify the most appropriate architectural design decision for various types of software systems.

Previously, software architects had relied on objective-reasoning approaches that depended solely on their experience in analyzing tradeoff at the software architecture development stage. The proposed model could be effectively used by software architects as an evidence-based multi-criteria tradeoff decision making compelling tool. Embedded systems operational environment is evolving. Emerging factors influencing the efficiency of embedded systems may rise as criteria and new efficiency metrics might be applied. Hence, this research area will continue to evolve towards more optimized embedded systems' efficiency.

ACKNOWLEDGMENT

The authors are grateful to the anonymous IEEE Access reviewers for their comments and suggestions.

REFERENCES

[1] S. Sridhar, A. Hahn, and M. Govindarasu, "Cyber–physical system security for the electric power grid," *Proc. IEEE*, vol. 100, no. 1, pp. 210–224, Oct. 2011.

[2] L. Cheng, K. Tian, and D. Yao, "Orpheus: Enforcing cyber-physical execution semantics to defend against data-oriented attacks," in *Proc. 33rd Annu. Comput. Secur. Appl. Conf.*, Dec. 2017, pp. 315–326.

[3] I. Crnkovic, "Component-based approach for embedded systems," in *Proc. 9th Int. Workshop Compon.-Oriented Program.*, 2004, pp. 1–6.

[4] S. Hammoudi, Z. Aliouat, and S. Harous, "Challenges and research directions for Internet of Things," *Telecommun. Syst.*, vol. 67, no. 2, pp. 367–385, Feb. 2018.

[5] S. H. Al-Daajeh, R. E. Al-Qutaish, and F. Al-Qirem, "Engineering dependability to embedded systems software via tactics," *Int. J. Softw. Eng. Appl.*, vol. 5, pp. 45–53, 2011.

[6] C. Ebert and D. Reiner, *Best Practices in Software Measurement: Establish-Extract-Evaluate-Execute*. Heidelberg, Germany: Springer-Verlag, 2007.

[7] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Reading, MA, USA: Addison-Wesley, 2015.

[8] S. Mikael and W. Claes, "A comparative study of quantitative and qualitative views of software architectures," in *Proc. 7th Int. Conf. Empirical Assessment Softw. Eng.*, 2003, pp. 1–8.

[9] W. Wu and T. Kelly, "Safety tactics for software architecture design," in *Proc. 28th Annu. Int. Comput. Softw. Appl. Conf. (COMPSAC)*, 2004, pp. 368–375.

[10] L. Zhu, M. A. Babar, and R. Jeffery, "Mining patterns to support software architecture evaluation," in *Proc. 4th Work. IEEE/IFIP Conf. Softw. Archit. (WICSA)*, Jun. 2004, pp. 25–34.

[11] G. M. L. Bass and H. M. Klein, "Applicability of general scenarios to the architecture tradeoff analysis method," Softw. Eng. Inst., Carnegie-Mellon Univ., Pittsburg, PA, USA, Tech. Rep. CMU/SEI-2001-TR-014, 2001.

[12] J. Scott and R. Kazman, "Realizing and refining architectural tactics: Availability," Softw. Eng. Inst., Carnegie-Mellon Univ., Pittsburg, PA, USA, Tech. Rep. CMU/SEI-2009-TR-006, 2009.

[13] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[14] R. Kazman, S. J. Carrière, and S. G. Woods, "Toward a discipline of scenario-based architectural engineering," *Ann. Softw. Eng.*, vol. 9, nos. 1–2, pp. 5–33, 2000.

[15] T. J. McCabe, "A complexity measure," *IEEE Trans. Softw. Eng.*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976.

[16] N. Lassing, D. Rijsenbrij, and H. van Vliet, "On software architecture analysis of flexibility, complexity of changes: Size isn't everything," in *Proc. 2nd Nordic Softw. Archit. Workshop (NOSA)* Ronneby, Sweden, 1999, pp. 1103–1581.

[17] J. Heit, "Impact of methods and mechanisms for improving software dependability on non-functional requirements," Ph.D. dissertation, Univ. Stuttgart, Stuttgart, Germany, 2007.

[18] R. B. France, D.-K. Kim, S. Ghosh, and E. Song, "A UML-based pattern specification technique," *IEEE Trans. Softw. Eng.*, vol. 30, no. 3, pp. 193–206, Mar. 2004.

[19] J. Bogner, S. Wagner, and A. Zimmermann, "Using architectural modifiability tactics to examine evolution qualities of service- and microservice-based systems," *SICS Softw.-Intensive Cyber-Phys. Syst.*, vol. 34, nos. 2–3, pp. 141–149, Jun. 2019.

[20] R. Wojcik, F. Bachmann, L. Bass, P. Clements, P. Merson, R. Nord, and W. Wood, "Attribute-driven design (add), version 2.0," Softw. Eng. Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-2006-TR-023, 2006. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=8147

[21] F. Bachmann, L. Bass, and R. Nord, "Modifiability tactics," Softw. Eng. Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-2007-TR-002, 2007. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=8299

[22] D.-K. Kim, "The role-based metamodeling language for specifying design patterns," in *Design Pattern Formalization Techniques*. Hershey, PA, USA: IGI Global, 2007, pp. 183–205.

[23] T. Kühn, C. Werner, H. Schön, Z. Zhenxi, and U. Aßmann, "Contextual and relational role-based modeling framework," in *Proc. 45th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Aug. 2019, pp. 442–449.

[24] C. Andersson, "Managing software quality through empirical analysis of fault detection," Ph.D. dissertation, Lund Univ., Lund, Sweden, 2006.

[25] A. Jahan, K. Edwards, and M. Bahraminasab, "Multi-criteria decision-making for materials selection," in *Multi-Criteria Decision Analysis for Supporting the Selection of Engineering Materials in Product Design*, 2nd ed. Oxford, U.K.: Butterworth-Heinemann, Feb. 2016, pp. 63–80.

[26] M. Velasquez and P. T. Hester, "An analysis of multi-criteria decision making methods," *Int. J. Oper. Res.*, vol. 10, no. 2, pp. 56–66, 2013.

[27] R. Kumar, A. I. Khan, Y. B. Abushark, M. M. Alam, A. Agrawal, and R. A. Khan, "An integrated approach of fuzzy logic, AHP and TOPSIS for estimating usable-security of Web applications," *IEEE Access*, vol. 8, pp. 50944–50957, 2020.

[28] F. A. Al-Zahrani, "Hesitant-fuzzy sets-based computational approach for evaluating the survivability impact of multi-fiber WDM networks: Kingdom of Saudi Arabia perspective," *IEEE Access*, vol. 8, pp. 212409–212422, 2020.

[29] M. Alenezi, A. Agrawal, R. Kumar, and R. A. Khan, "Evaluating performance of Web application security through a fuzzy based hybrid multi-criteria decision-making approach: Design tactics perspective," *IEEE Access*, vol. 8, pp. 25543–25556, 2020.

[30] A. Agrawal, A. K. Pandey, A. Baz, H. Alhakami, W. Alhakami, R. Kumar, and R. A. Khan, "Evaluating the security impact of healthcare Web applications through fuzzy based hybrid approach of multi-criteria decision-making analysis," *IEEE Access*, vol. 8, pp. 135770–135783, 2020.

[31] A. Agrawal, M. Alenezi, R. Kumar, and R. Ahmad Khan, "Measuring the sustainable-security of Web applications through a fuzzy-based integrated approach of AHP and TOPSIS," *IEEE Access*, vol. 7, pp. 153936–153951, 2019.

[32] R. Kumar, A. Baz, H. Alhakami, W. Alhakami, M. Baz, A. Agrawal, and R. A. Khan, "A hybrid model of hesitant fuzzy decision-making analysis for estimating usable-security of software," *IEEE Access*, vol. 8, pp. 72694–72712, 2020.

[33] K. Sahu, F. A. Alzahrani, R. Srivastava, and R. Kumar, "Evaluating the impact of prediction techniques: Software reliability perspective," *Comput. Mater. Continua*, vol. 67, no. 2, pp. 1471–1488, 2021.

[34] K. Sahu, F. A. Alzahrani, R. K. Srivastava, and R. Kumar, "Hesitant fuzzy sets based symmetrical model of decision-making for estimating the durability of Web application," *Symmetry*, vol. 12, no. 11, p. 1770, Oct. 2020.

[35] A. Alharbi, W. Alosaimi, H. Alyami, M. Nadeem, M. Faizan, A. Agrawal, R. Kumar, and R. A. Khan, "Managing software security risks through an integrated computational method," *Intell. Autom. Soft Comput.*, vol. 28, no. 1, pp. 179–194, 2021.

[36] L. Zhu, A. Aurum, I. Gorton, and R. Jeffery, "Tradeoff and sensitivity analysis in software architecture evaluation using analytic hierarchy process," *Softw. Qual. J.*, vol. 13, no. 4, pp. 357–375, Dec. 2005.

[37] K. Halicka, "Technology selection using the TOPSIS method," *Foresight STI Governance*, vol. 14, no. 1, pp. 85–96, Mar. 2020.

[38] K. S. Kumar and D. Malathi, "A novel method to find time complexity of an algorithm by using control flow graph," in *Proc. Int. Conf. Tech. Adv. Comput. Commun. (ICTACC)*, Apr. 2017, pp. 66–68.

[39] J. Graylin, J. E. Hale, R. K. Smith, H. David, N. A. Kraft, and C. Ward, "Cyclomatic complexity and lines of code: Empirical evidence of a stable linear relationship," *J. Softw. Eng. Appl.*, vol. 2, no. 3, p. 137, 2009.

[40] T. L. Saaty, "How to make a decision: The analytic hierarchy process," *Eur. J. Oper. Res.*, vol. 48, no. 1, pp. 9–26, Sep. 1990.

[41] A. Ishizaka and M. Lusti, "How to derive priorities in AHP: A comparative study," *Central Eur. J. Operations Res.*, vol. 14, no. 4, pp. 387–400, Nov. 2006.

[42] A. Ishizaka and P. Nemery, *Multi-Criteria Decision Analysis: Methods and Software*. Hoboken, NJ, USA: Wiley, 2013.

[43] T. L. Saaty, "An exposition of the AHP in reply to the paper 'remarks on the analytic hierarchy process,'" *Manage. Sci.*, vol. 36, no. 3, pp. 259–268, Mar. 1990.

[44] H. Zhang, X. Chen, Y. Dong, W. Xu, and S. Wang, "Analyzing Saaty's consistency test in pairwise comparison method: A perspective based on linguistic and numerical scale," *Soft Comput.*, vol. 22, no. 6, pp. 1933–1943, Mar. 2018.

[45] R. Kumar, A. I. Khan, Y. B. Abushark, M. M. Alam, A. Agrawal, and R. A. Khan, "A knowledge-based integrated system of hesitant fuzzy set, AHP and TOPSIS for evaluating security-durability of Web applications," *IEEE Access*, vol. 8, pp. 48870–48885, 2020.

[46] L. Wang, Y. Ali, S. Nazir, and M. Niazi, "ISA evaluation framework for security of Internet of health things system using AHP-TOPSIS methods," *IEEE Access*, vol. 8, pp. 152316–152332, 2020.

SALEH H. ALDAAJEH (Member, IEEE) received the B.Sc. degree in computer science from the University of Petra, Amman, Jordan, in 2007, and the M.Sc. degree in software engineering from the Blekinge Institute of Technology, Karlskrona, Sweden, in 2010. He is currently pursuing the Ph.D. degree in information security with the College of Information Technology, United Arab Emirates University, Al Ain, United Arab Emirates. His research interests include information security, reverse engineering, the Internet of Things, and dependability engineering in safety-critical systems.

SAAD HAROUS (Senior Member, IEEE) received the Ph.D. degree in computer science from Case Western Reserve University, Cleveland, OH, USA, in 1991. He is currently a Professor with the College of Information Technology, United Arab Emirates University. He has more than 30 years of experience in teaching and research in three different countries: USA, Oman, and United Arab Emirates. He has published more than 200 journal articles and conference papers. His teaching interests include programming, data structures, design and analysis of algorithms, operating systems, and networks. His research interests include parallel and distributed computing, P2P delivery architectures, wireless networks, VANET, and the use of computers in education and processing Arabic language.

SAED ALRABAEE (Senior Member, IEEE) received the Ph.D. degree in information system engineering from Concordia University, Montreal, QC, Canada, which was executed under the supervision of Prof. Mourad Debbabi and Prof. Lingyu Wang. He is currently an Assistant Professor with the Department of Information Systems and Security, United Arab Emirates University (UAEU). Prior to joining UAEU, he was a Visiting Assistant Professor with the Department of Electrical and Computer Engineering and Computer Science, University of New Haven (UNH), USA. He is also a Permanent Research Scientist with the Security Research Center, CIISE, Concordia University, Canada. His research and development activities and interests focus on the broad area of reverse engineering, including, binary authorship attribution and characterization, malware investigation, and function fingerprinting.