International Journal of Web Applications

DLINE JOURNALS

Print ISSN: 0974-7710
Online ISSN: 0974-7729
IJWA 2026: 18 (4)
https://doi.org/10.6025/ijwa/2026/18/4/240-272

A Bipartite Network-Based Framework for IoT Requirement Traceability Test Coverage Assessment, and Critical Test-Case Identification

Juan Jose Prieto-Gutierrez
Complutense University of Madrid
Spain
jujpriet@ucm.es

ABSTRACT

The rapid expansion of Internet of Things (IoT) systems introduces significant complexity in software verification, necessitating robust mechanisms for requirement traceability and test management. Conventional manual approaches are prone to inconsistencies and fail to scale, while existing automated methods often treat traceability, coverage assessment, and test prioritization as isolated activities. This paper proposes a unified, bipartite network-based framework for IoT requirement traceability, test coverage assessment, and critical test-case identification. By modeling software requirements and test cases as distinct node sets connected by explicit traceability edges, the framework enables systematic graph theoretic analysis. Empirical evaluation on an IoT dataset comprising 82 requirements and 83 test cases reveals a 100% structural coverage but highlights a sparse network topology (1.60% density) fragmented into 63 connected components. The analysis successfully identifies structurally critical nodes, such as requirement DAR11 and test case TC-RMM3a, while simultaneously exposing critical data-quality gaps, including missing interface metadata. Ultimately, this framework provides a foundational architecture for transitioning from manual traceability maintenance to intelligent, requirement-aware test management and optimization in complex IoT environments.

Keywords: Internet of Things (IoT), Requirement Traceability, Bipartite Network, Test Coverage Assessment, Critical Test-Case Identification, Graph-Based Modeling

Received: 18 June 2026, Revised: 21 September 2026, Accepted: 30 September 2026

Copyright: **DLINE**

1. Introduction

The rapid development of sensor technologies and network-based communication has accelerated the wide

spread adoption of the Internet of Things (IoT), enabling physical entities to connect to digital networks and exchange information through interconnected sensing devices. By integrating sensors, communication technologies, and intelligent processing capabilities, IoT systems facilitate continuous monitoring, data collection, and interaction between physical and virtual environments. These capabilities have supported the implementation of IoT applications across several domains, including smart cities, smart homes, wearable medical devices, and environmental perception systems [1-4]. As IoT technology continues to expand, the complexity of interconnected devices, communication interfaces, and software components also increases, creating additional challenges for system development, verification, testing, and maintenance.

Unlike conventional computing environments, IoT systems frequently operate under resource constrained conditions. Many IoT devices function as interconnected network nodes with limited computational power, storage capacity, and processing resources to satisfy the operational requirements of network edge environments [5]. These constraints introduce additional considerations during software engineering and testing because system functionality must be verified while accounting for device limitations, communication dependencies, heterogeneous interfaces, and varying operational conditions. Consequently, ensuring the reliability and maintainability of IoT applications requires systematic approaches that connect software requirements with their corresponding implementation and testing activities.

One of the fundamental challenges in software testing is establishing and maintaining accurate traceability relationships between software requirements and test cases. Requirement traceability provides a systematic mechanism for determining whether individual requirements are adequately addressed by the corresponding test cases. These relationships are essential for evaluating test coverage, identifying untested requirements, supporting defect detection, and ensuring that software functionality meets its intended specifications. However, conventional manual traceability processes require considerable human effort to establish, verify, and update relationships between requirements and test cases. Such processes are susceptible to inconsistencies and human errors, particularly in large scale software projects involving numerous requirements and test artefacts. Furthermore, requirements frequently evolve throughout the software development lifecycle, making it increasingly difficult to maintain accurate traceability relationships. These challenges increase maintenance costs and complicate regression testing and overall test management [6].

The problem becomes more significant in large-scale IoT environments, where software functionality is closely associated with network communication, device interactions, and service-dependent operational characteristics. Generating high-fidelity packet trace data, including packet size and packet time intervals, is important for developing accurate network analysis tools and improving system monitoring capabilities. Such data supports network performance assessment, threat detection, and the development of resource allocation strategies that improve service quality and user experience. Nevertheless, generating detailed packet traces for multiple IoT services remains challenging because of variations in service characteristics and the complex temporal dependencies between network packets [7]. These characteristics demonstrate that IoT testing cannot be considered exclusively from a software functionality perspective; it must also account for the relationships among system requirements, network behavior, device constraints, and operational dependencies.

Against this background, effective requirement traceability and test management require approaches capable of representing complex relationships among heterogeneous system elements. In particular, a systematic connection between requirements and test cases can provide the foundation for evaluating requirement coverage, identifying missing test relationships, and determining which test cases are most important for system verification. Such an approach becomes particularly relevant when IoT applications contain numerous interconnected components and when changes to one requirement or interface may influence several related testing activities.

2. Related Literature and Research Background

2.1 Requirement Traceability in IoT Software Engineering

Requirement trace ability is an important activity in software engineering because it establishes explicit relationships between system requirements and the development artifacts associated with their implementation and verification. Requirements Engineering Technology provides mechanisms for facilitating traceability among IoT requirements, thereby supporting the systematic management of requirements throughout the development process [8]. In this context, traceability is not limited to documenting requirement relationships; it also provides a foundation for assessing whether requirements have been implemented and adequately tested.

Requirement to Code traceability extends this concept by establishing explicit links between individual requirements and the code elements responsible for implementing them [9]. These relationships support several essential software engineering activities, including requirement validation, change impact analysis, regression testing, and compliance checking [10]. When combined with requirement to test traceability, these relationships can provide a more comprehensive view of the software development and verification lifecycle. For example, identifying the code elements associated with a modified requirement can help determine which test cases should be reconsidered or executed during regression testing.

An additional consideration is the distinction between functional requirements (FRs) and non-functional requirements (NFRs). Functional requirements describe the services and operations that a system must perform, whereas non functional requirements address quality attributes and operational constraints. In IoT environments, non functional requirements are particularly important because system performance, security, resource consumption, and communication reliability can influence overall system behavior. Wang examined the characteristics of NFR traceability across multiple categories and compared them with FR traceability, including a focused analysis of security related non functional requirements [11]. This distinction indicates that comprehensive traceability mechanisms should consider both functional behavior and the quality-related constraints that influence IoT system operation.

Despite the importance of these relationships, maintaining traceability across complex software projects remains challenging. The increasing number of requirements, implementation elements, interfaces, and test cases makes manual relationship management difficult to scale. Moreover, modifications to requirements or system architecture may introduce inconsistencies in existing traceability links. These challenges establish the need for systematic relationship representations that can support requirement coverage analysis and facilitate the identification of test cases associated with critical system requirements.

2.2 Traceability and Testing in IoT and Industrial IoT Systems

The development of Industrial Internet of Things (IIoT) systems introduces additional complexity because software components interact with industrial devices, communication networks, and operational processes. Testing in such environments requires consideration of not only functional correctness but also execution constraints, fault detection capabilities, requirement criticality, and interface complexity. A conventional testing strategy based exclusively on test case execution order may therefore be insufficient for prioritizing the most important verification activities.

An IIoT framework integrating requirement traceability analysis, multi-factor prioritization scoring, and optimization-driven test selection has been proposed to dynamically prioritize regression test cases. The framework considers fault detection capability, execution time, requirement criticality, and interface complexity to establish a more informed test execution order [12]. This approach demonstrates the importance of combining traceability information with test prioritization and optimization mechanisms. In particular, traceability relationships can provide information about the requirements addressed by individual test cases, while prioritization criteria can help determine which tests should receive attention first under limited testing resources.

However, requirement traceability, test coverage assessment, and test-case prioritization represent related but distinct testing activities. Traceability establishes the relationships between requirements and test cases; coverage assessment evaluates whether requirements are adequately represented by the available tests; and

prioritization determines the relative importance or execution order of test cases. Integrating these activities into a unified analytical framework can provide a more systematic basis for identifying untested requirements, evaluating testing completeness, and recognizing test cases that contribute significantly to system verification.

2.3 Graph-Based Modeling and Its Relevance to NG-IoT Systems

The increasing interconnectivity of next-generation Internet of Things (NG-IoT) environments has created opportunities for applying graph-based computational approaches to model complex relationships among system entities. Graph Neural Networks (GNNs) have emerged as an important framework for representing and learning from interconnected structures. Their ability to model relationships among multiple entities makes them particularly relevant to complex NG-IoT networks, where devices, communication links, and other system components exhibit interconnected behavior.

Despite growing research interest in applying GNNs to next generation IoT systems, existing studies remain fragmented, and comprehensive guidance on their systematic application to NG-IoT environments is still limited [13]. This observation highlights the broader need for structured representations capable of capturing relationships within complex IoT environments.

For requirement traceability and test coverage analysis, a graph based representation offers a suitable conceptual foundation because requirements and test cases can be treated as distinct categories of entities connected through traceability relationships. A bipartite network is particularly relevant to this problem because its two sets of nodes represent different entity types, while the connections between them express the relationships between requirements and test cases. This representation provides a structured way to examine which requirements are associated with particular tests, which requirements lack corresponding test cases, and which test cases cover multiple requirements.

Importantly, the application of a bipartite network to requirement traceability differs from using a graph merely to represent communication among IoT devices. Its primary purpose in this context is to model the relationships between software requirements and testing artifacts. Such a representation can provide an analytical basis for evaluating traceability completeness, examining coverage distribution, and supporting the identification of critical test cases.

2.4 Existing Traceability and Fault-Analysis Approaches

Trace ability has also been investigated in several application domains beyond conventional software requirement management. These studies provide useful insights into the different ways in which relationships among data, processes, system components, and faults can be established and analyzed. However, their objectives and underlying mechanisms vary considerably, indicating that the effectiveness of a traceability approach depends on the nature of the entities being traced and the analytical requirements of the application.

Research on the traceability of abnormal data points has demonstrated its practical value for operational monitoring and the activities of operation and monitoring centers. Such traceability can assist in investigating abnormal observations and identifying information relevant to system operation. Caiyun et al. proposed a distributed traceability model based on digital watermarking and blockchain technology in edge computing environments [14]. The model enhanced the security of conventional traceability mechanisms by separating internal and external regions and selecting master nodes. Although the approach improved security, its principal emphasis was on protecting traceability information, and its ability to analyze anomalies remained limited.

Fu et al. [15] introduced a traceability and interaction mechanism for spatial image data to support the tracking of information throughout data-processing operations [15]. The mechanism employed grid-code partitioning and indexing techniques to preprocess spatial image data, thereby improving data retrieval efficiency. Its core algorithm was a fast, practical Byzantine fault-tolerant mechanism implemented through an alliance chain. Although this approach provided an effective mechanism for spatial data traceability and retrieval, its primary focus was on data-processing traceability rather than the systematic analysis of software requirements and test coverage.

In another application domain, Li et al. [16] developed a continuous commutation fault-tracing method for multi infeed direct current systems [16]. Their approach quantitatively evaluated the influence of contact impedance on the probability of continuous commutation failure and analyzed potential fault causes by comparing weighting factors. Nevertheless, the approach primarily traced fault sources from the perspective of physical implementation and therefore had limited universality when applied to different types of systems.

Wang et al. proposed a fault-traceability model that combined historical production cases with process data information to identify potential defect sources [17]. The model initially determined defect-cause tags using text-similarity calculations. It subsequently applied L1 regularization for feature selection and employed XGBoost to develop a correlation model between process data and defects. The contribution of individual features to the data source was then evaluated, and Dempster Shafer (D-S) evidence theory was used to establish different rules for determining the potential fault source. Although this approach integrated historical information, feature selection, machine learning, and evidence-based reasoning, the absence of an effective feature-extraction model limited its traceability performance.

Similarly, Li et al. proposed a composite welding-quality traceability model incorporating a quality early-warning mechanism based on a long short-term memory (LSTM) network and an optimization algorithm for quality-data backtracking queries [18]. The model combined early-warning prediction with query optimization to support quality traceability. The LSTM component was used to distinguish between good-quality and poor-quality outcomes, while the optimization component was intended to improve the retrieval of relevant quality information. However, the query optimization algorithm lacked an effective analytical strategy, resulting in unsatisfactory traceability performance.

Collectively, these studies demonstrate that traceability mechanisms have been developed using a variety of technologies, including blockchain, digital watermarking, indexing, machine learning, feature selection, evidence theory, and deep learning. Their applications range from abnormal data investigation and spatial-data processing to industrial fault diagnosis and manufacturing quality management. Nevertheless, the approaches differ in their objectives, analytical capabilities, and applicability to heterogeneous systems. In particular, some emphasize information security, others focus on data retrieval or physical fault identification, and several depend on domain-specific feature extraction and analytical procedures. [19].

3. Research Gap and Motivation

The reviewed literature establishes the importance of traceability in software engineering, IoT systems, industrial applications, and data driven fault analysis environments. Existing research has demonstrated the usefulness of requirement engineering technologies, requirement to code relationships, regression test prioritization, graph based learning, block chain enabled traceability, machine learning based fault identification, and deep learning based quality monitoring. These contributions provide important foundations for improving system verification, information tracking, and fault investigation.

However, several limitations remain relevant to the problem of requirement traceability and test coverage assessment in IoT environments. First, requirement traceability, test coverage evaluation, and test-case prioritization are frequently addressed as separate activities. This separation makes it difficult to obtain an integrated understanding of how requirements are represented in the available test cases and how individual test cases contribute to overall verification completeness. Second, conventional manual traceability processes are difficult to maintain in large scale IoT applications because requirements, interfaces, and testing artifacts may change throughout the development lifecycle. Third, existing traceability approaches developed for blockchain-based data protection, spatial data retrieval, physical fault identification, and manufacturing-quality analysis are generally designed around domain specific objectives. Consequently, their analytical mechanisms do not directly address the combined problem of requirement to test relationship modeling, coverage assessment, and critical test case identification.

Furthermore, although graph based approaches and GNNs have attracted increasing attention in NG-IoT research, their potential for systematically representing the relationships between software requirements

and test cases remains an important area for investigation. In particular, a structured representation is needed to distinguish the two categories of entities while preserving the relationships that connect them. A bipartite network provides a relevant modeling foundation for this purpose because requirements and test cases can be represented as two distinct node sets, with edges indicating their traceability relationships.

The central research gap, therefore, concerns the lack of an integrated analytical framework that uses a structured requirement test relationship representation to support traceability analysis, test coverage assessment, and the identification of critical test cases in IoT-oriented software systems. Addressing this gap requires a unified approach in which the relationships between requirements and test cases can be examined systematically rather than treated as isolated documentation or testing activities.

3.1 Proposed Research Direction

To address the identified research gap, this study focuses on a bipartite network based framework for IoT requirement traceability, test coverage assessment, and critical test case identification. The proposed research direction considers software requirements and test cases as two distinct categories of entities connected through explicit traceability relationships. Representing these relationships as a bipartite network provides a structured basis for analyzing the extent to which individual requirements are associated with available test cases and determining how test cases contribute to requirement coverage.

The framework is intended to bring three closely related activities into a unified analytical process. The first activity concerns requirement traceability, which establishes and examines the relationships between requirements and their corresponding test cases. The second activity concerns test coverage assessment, which evaluates the representation of requirements within the testing structure and helps identify requirements without corresponding test coverage. The third activity concerns critical test-case identification, which focuses on recognizing test cases that have particular importance for verifying system requirements.

This integrated perspective is particularly relevant to IoT systems because their heterogeneous devices, resource limitations, communication dependencies, and evolving software requirements create challenges for maintaining comprehensive testing relationships. By organizing requirement test relationships within a bipartite network, the study aims to provide a more systematic foundation for understanding traceability structure, evaluating testing completeness, and supporting informed test case selection.

The proposed direction also complements existing research on multi-factor regression test prioritization [12] by emphasizing the structural relationships between requirements and test cases. While the previous prioritization approach considers fault detection capability, execution time, requirement criticality, and interface complexity, the present study focuses on the bipartite representation of requirement test relationships and its relevance to coverage assessment and critical test case identification. These perspectives are complementary because traceability information can provide the structural basis for understanding testing coverage, while prioritization criteria can support subsequent decisions regarding test execution.

The increasing deployment of IoT systems has intensified the need for reliable software verification and systematic testing approaches. The resource limitations of IoT devices, the complexity of network interactions, the diversity of software requirements, and the continuous evolution of system components make conventional manual traceability and testing processes increasingly difficult to maintain. Although previous studies have contributed various techniques for requirement engineering, traceability management, graph-based modeling, fault analysis, and regression test prioritization, important challenges remain in integrating requirement traceability, test coverage assessment, and critical test-case identification.

The proposed bipartite network-based research direction addresses this challenge by providing a structured representation of the relationships between software requirements and test cases. Through this representation, the study establishes a conceptual foundation for examining traceability relationships, assessing requirement coverage, identifying potential coverage gaps, and determining the importance of individual test cases. The intended contribution is an integrated approach that connects requirement management with systematic test analysis, with particular relevance to the complexity and evolving verification requirements of IoT-based software systems.

3.2 Research Methodology

3.21 Research Design and Analytical Framework

This study adopts a quantitative, graph based analytical approach to investigate requirement traceability, test coverage, and critical test case identification in Internet of Things (IoT) software systems. The proposed methodology transforms the requirement to test case traceability matrix into a bipartite network, enabling the systematic examination of relationships between software requirements and their corresponding test cases.

The research is organized into five interconnected analytical stages: (i) dataset collection, validation, and preprocessing; (ii) requirement and test case traceability matrix construction; (iii) bipartite network modeling; (iv) requirement coverage and network centrality analysis; and (v) critical test case identification and testing-gap assessment.

The methodology considers requirements and test cases as two distinct categories of network entities. Relationships between these entities are represented through binary connections, where an edge indicates that a particular test case is associated with a specific requirement. This representation enables the identification of requirements with extensive testing relationships, test cases that verify multiple requirements, and structural dependencies that may influence regression testing.

The analysis further integrates interface level traceability information to examine how testing relationships are distributed across system interfaces. Network density, connected components, degree centrality, and betweenness centrality are used to characterize the resulting network structure. These measures provide complementary information about traceability completeness, relationship distribution, and the structural importance of individual requirements and test cases.

3.22 Dataset Collection and Data Preparation

The experimental analysis uses the IoT requirement traceability matrix as the primary data source, supported by the corresponding requirements register and test case attribute information. These data sources provide information about requirement identifiers, test case identifiers, requirement categories, interface associations, and the relationships established between requirements and test cases.

The traceability matrix contains 82 valid requirement identifiers, 83 distinct test cases, and 109 requirement–test-case relationships. During initial inspection, 19 placeholder rows containing zero values were identified. These entries were excluded because they do not represent valid requirement identifiers or testing relationships. Data preprocessing was performed to ensure consistency among the different data sources. Requirement identifiers were normalized to establish a consistent representation across the requirements register and traceability matrix. Particular attention was given to combined requirement identifiers, where multiple requirements were represented within a single spreadsheet cell. Such entries require separation into individual identifiers before comparison and relationship validation.

The preprocessing stage also examined missing interface identifiers, duplicate relationships, and inconsistencies between the requirements register and traceability matrix. Three requirements, namely NFM8, NFM9, and NFM10, were present in the traceability matrix but were not separately listed in the requirements register.

These requirements also lacked interface identifiers. They were retained in the network analysis because they represented valid requirement test case relationships, while their metadata inconsistencies were documented as data quality issues.

The resulting normalized traceability dataset was used as the foundation for subsequent network construction and coverage analysis.

3.3 Construction of the Requirement–Test-Case Traceability Matrix

The normalized dataset was represented using a binary requirement test case adjacency matrix. Let the set of

software requirements be represented as

$$R = \{ r_1, r_2, \ldots\ldots, r_m \}$$

and the set of test cases as

$$T = \{ t_1, t_2, \ldots\ldots, t_m \}$$

The relationship between requirements and test cases is represented by an adjacency matrix A:

$$A = [a_{ij}]_{mxn}$$

where

$$a_{ij} = \{ 1, \text{ if test case } t_j \text{ is linked to requirement } r_i, \text{ otherwise } 0, $$

In the present dataset, m=82 and n=83, resulting in an adjacency matrix of dimensions 82x83.

Each non-zero matrix element represents an explicitly recorded traceability relationship. Consequently, the matrix contains 109 valid relationships. This binary representation provides a consistent basis for examining the distribution of testing relationships across requirements and test cases.

The matrix also supports the identification of requirements associated with multiple test cases and test cases associated with multiple requirements. These relationships are important for understanding testing redundancy, shared verification activities, and potential dependencies during regression testing.

3.4 Bipartite Network Modeling

The adjacency matrix was transformed into an undirected bipartite graph:

$G = (V, E)$ where the vertex set is divided into two disjoint subsets: $V = R \cup T$, $R \cap T = \varphi$ Here, R represents requirement nodes, while T represents test-case nodes. The edge set E contains the recorded requirement–test-case relationships. An edge is established between requirement r_i and test case t_j whenever $a_{ij} = 1$. No edges are introduced between two requirement nodes or between two test-case nodes. This restriction preserves the bipartite structure of the network.

The resulting network consists of 165 nodes, comprising 82 requirement nodes and 83 test-case nodes, connected through 109 edges. Each edge represents an individual traceability relationship.

The bipartite representation enables the investigation of both local and global network characteristics. At the local level, the number of connections associated with individual requirements and test cases can be examined. At the global level, network density and connected-component analysis provide information about the overall distribution and fragmentation of traceability relationships.

3.5 Requirement Coverage Assessment

Requirement coverage analysis was performed to determine whether every valid requirement recorded in the traceability matrix was associated with at least one test case.

For each requirement, the number of linked test cases was calculated as:

$C(r_i) = \sum_{j=1}^{n} a_{ij}$ where $C(r_i)$ represents the number of test cases associated with requirement r_i. A requirement was considered covered if at least one corresponding test case existed: $I(r_i) = \begin{cases} 1, & C(r_i) > 0 \\ 0, & C(r_i) = 0 \end{cases}$ overall requirement coverage was calculated as: where represents the total number of valid requirements. The analysis identified

82 requirements with corresponding test cases and no uncovered requirements within the traceability matrix. Therefore, the recorded requirement coverage was: $RC = \frac{82}{82} \times 100 = 100\%$

In addition to overall coverage, the number of test cases associated with each requirement was examined to identify differences in testing support. This analysis distinguishes requirements supported by multiple test cases from those dependent on a single test case.

An important distinction is maintained between recorded coverage and verification adequacy. Although a requirement may have at least one associated test case, this does not necessarily establish that the test is sufficiently comprehensive, independent, or effective in detecting faults.

3.6 Test-Case Relationship and Multi-Requirement Coverage Analysis

The contribution of individual test cases was examined by calculating the number of requirements associated with each test case.

The test-case connectivity measure is defined as

$$C(t_j) = \sum_{i=1}^{m} a_{ij},$$

where C (*tj*) denotes the number of requirements verified by test case t_j.

Based on this measure, test cases were categorized into three groups: test cases associated with one requirement, test cases associated with multiple requirements, and test cases without any recorded requirement association.

The analysis identified 67 test cases linked to a single requirement and 16 test cases linked to multiple requirements. No test cases were found without a recorded requirement relationship.

Particular attention was given to test cases with higher connectivity because they may contribute to the verification of several requirements within a single execution. For example, TC-RMM3a was associated with four requirements: RMM3, RMM4, RMM5, and RMM7.

Such test cases are potentially important when evaluating test-suite efficiency and planning regression testing. However, high connectivity alone does not demonstrate that a test case is more effective at detecting faults. Its actual effectiveness must be evaluated using execution and fault detection evidence.

3.7 Interface-Level Traceability Analysis

Interface-level analysis was performed to investigate how requirement test case relationships were distributed across the system interfaces.

The interface identifier associated with each requirement was used to group the corresponding traceability relationships. For each interface, three measures were examined:

1. Number of distinct requirements associated with the interface.

2. Number of distinct test cases associated with the interface.

3. Total number of recorded requirement test case relationships.

This analysis provides an additional perspective on the organization of testing activities within the IoT system. Interfaces with a larger number of traceability relationships indicate more extensive recorded testing associations, while interfaces with concentrated relationships may represent areas requiring closer examination

during change impact analysis.

The dataset contains 12 distinct valid interface identifiers. Three requirements, NFM8, NFM9, and NFM10, do not have associated interface identifiers and were therefore excluded from interface specific grouping.

The interface analysis was used to identify areas with extensive testing relationships and to highlight incomplete interface metadata. Relationship counts were interpreted as measures of traceability distribution rather than direct indicators of interface risk.

3.8 Bipartite Network Density Analysis

Network density was calculated to evaluate the proportion of possible requirement test case connections that were present in the observed traceability network.

For a bipartite graph, network density is defined as

$$D = \frac{|E|}{|R| \times |T|},$$

where

- |E |is the total number of recorded edges,
- | R | is the number of requirement nodes,
- | T |is the number of test-case nodes.

Using the dataset values,

$$D = \frac{109}{82 \times 83} = 0.0160$$

Therefore, the bipartite network density is approximately 1.60%.

This measure indicates that a relatively small proportion of all theoretically possible requirement test case relationships are recorded. The resulting structure is sparse, reflecting selective associations between requirements and test cases. Such sparsity is not necessarily undesirable because individual test cases are generally designed to validate specific subsets of system requirements rather than every requirement.

3.9 Connected-Component Analysis

Connected-component analysis was conducted to examine the fragmentation and structural organization of the bipartite network.

A connected component represents a group of requirement and test case nodes in which every node is reachable from every other node through a sequence of edges.

The number of connected components was calculated by identifying independent groups within the network. The analysis identified 63 connected components.

The component-size distribution was examined to determine whether the traceability network was dominated by isolated requirement test case pairs or larger interconnected groups. The results showed that 51 components consisted of two nodes, while the remaining components contained larger combinations of requirements and test cases.

The largest connected component contained 12 nodes. The absence of isolated nodes indicates that all valid requirements and test cases in the constructed network had at least one recorded relationship.

A large number of small components suggests that many testing relationships are requirement specific. Larger components, in contrast, represent interconnected testing structures in which multiple requirements and test cases are related. These larger structures may be particularly relevant for regression testing and change impact analysis because modifications to one element may influence several connected verification activities.

3.10 Degree Centrality Analysis

Degree centrality was used to quantify the number of direct connections associated with each requirement and test-case node.

For requirement nodes, degree centrality is calculated as: $C_D(r_i) = \frac{\sum_{j=1}^{n} a_{ij}}{n}$ For test-case nodes, degree centrality is calculated as: $C_D(t_j) = \frac{\sum_{i=1}^{m} a_{ij}}{m}$ where m and n represent the total number of requirements and test cases, respectively.

Requirement degree centrality identifies the relative connectivity of individual requirements within the testing structure. Requirements associated with multiple test cases receive higher values, indicating broader recorded testing support.

Similarly, test case degree centrality identifies the relative number of requirements associated with each test case. Test cases with higher degree centrality have broader requirement coverage within the traceability network.

The analysis identified DAR11 as the most connected requirement, with nine associated test cases and a normalized degree centrality of 0.1084. TC-RMM3a was identified as the most connected test case, with four associated requirements and a normalized degree centrality of 0.0488.

These measures provide an initial structural basis for identifying requirements with extensive testing relationships and test cases that contribute to multiple requirements.

3.11 Betweenness Centrality Analysis

Betweenness centrality was employed to identify nodes that occupy intermediary positions within the network structure. Unlike degree centrality, which considers direct connections, betweenness centrality measures the extent to which a node participates in shortest paths connecting other nodes.

• The normalized betweenness centrality of a node v is calculated as: $C_B(v) = \frac{2}{(N-1)(N-2)} \sum_{\substack{s<t \\ s\neq v,\, t\neq v}} \frac{\sigma_{st}(v)}{\sigma_{st}}$ where: • N represents the total number of nodes in the network. • σ_{st} represents the number of shortest paths between nodes s and t. • $\sigma_{st}(v)$ represents the number of shortest paths between and that pass through node v.

The summation considers all unordered pairs of distinct nodes, excluding the node \(v\) itself.

The normalization factor is appropriate for an undirected graph and expresses the measure on a normalized scale.

Nodes with relatively higher betweenness centrality may occupy important intermediary positions within interconnected traceability groups. Such nodes are potentially useful for identifying dependencies and supporting change impact analysis.

The results identified DAR11 and TC-RMM3a as the leading nodes in betweenness centrality, followed by RMM7 and RSM7. However, because the network contains numerous disconnected components, the resulting betweenness values are relatively small. Consequently, these values were interpreted comparatively rather than as indicators of strong global connectivity.

3.12 Requirement Coverage Centrality and Test Case Importance

To provide application specific indicators for the IoT testing context, normalized degree centrality was interpreted in terms of requirement coverage centrality and test case importance centrality.

Requirement coverage centrality is defined as:

Requirement coverage centrality is defined as:

Test-case importance centrality is defined as:

where

• is a test case,

• is the number of requirements covered by ,

• is the total number of requirements.

These are simply the degrees of the respective nodes divided by the size of the opposite partition, yielding values in

The first measure represents the proportion of all available test cases associated with a particular requirement. The second represents the proportion of all requirements associated with a particular test case.

These measures provide an intuitive interpretation of the structural connectivity of individual nodes. DAR11 achieved the highest requirement coverage centrality, while TC-RMM3a achieved the highest test case importance centrality.

The measures are useful for identifying highly connected requirements and test cases. Nevertheless, they quantify structural relationships rather than actual test quality, execution reliability, or fault detection effectiveness.

3.13. Critical Test-Case Identification and Testing-Gap Assessment

Critical test case identification was performed by examining the combined evidence obtained from requirement coverage, test case connectivity, degree centrality, betweenness centrality, and interface level traceability.

The analysis considered three important characteristics.

First, test cases associated with multiple requirements were examined because they provide shared verification across several system requirements. Such test cases may be particularly relevant when constructing compact regression test suites.

Second, test cases with higher degree centrality were identified because they demonstrate broader direct connections within the requirement test case network.

Third, test cases and requirements with relatively high betweenness centrality were examined because their intermediary positions may indicate structural dependencies within connected components.

In addition to these network characteristics, testing gaps were identified through the examination of uncovered requirements, requirements with only one associated test case, and missing interface metadata.

The analysis identified 70 requirements with exactly one associated test case. Although these requirements were covered according to the recorded traceability matrix, their verification depends on a single documented test case. Therefore, requirements with high operational importance, security relevance, or potential system impact should be examined to determine whether additional independent testing is necessary.

An important distinction is that the current analysis identifies structurally important test cases rather than experimentally verified fault-detection performance. A comprehensive optimization strategy would require additional information about test execution time, historical fault detection, requirement criticality, and interface complexity.

3.14 Visualization and Analytical Interpretation

Network visualization was used to provide a graphical representation of the bipartite traceability structure. Requirement nodes and test-case nodes were displayed as separate node classes, while edges represented their recorded traceability relationships.

Node sizes were associated with connectivity to make highly connected requirements and test cases visually distinguishable. The resulting visualization supports the interpretation of the network's sparse and fragmented structure, including the identification of small independent components and larger interconnected groups.

Additional visualizations were used to examine test-case frequency, interface level traceability, and degree distribution. These representations complement the numerical network measures by providing a visual understanding of how testing relationships are distributed across the IoT system.

The visualization results were interpreted together with the quantitative indicators rather than as independent evidence of testing effectiveness.

3.15 Methodological Limitations

The methodology has several limitations that should be considered when interpreting the findings.

First, the analysis depends on the accuracy and completeness of the recorded traceability matrix. Missing or incorrectly documented relationships may affect the estimated network structure and coverage measures.

Second, 100% requirement coverage indicates that every valid requirement in the analyzed matrix has at least one associated test case. It does not establish that all requirements have been adequately or independently verified.

Third, centrality measures quantify structural importance and cannot independently establish actual fault-detection effectiveness. Execution results and historical defect information are required to evaluate this dimension.

Fourth, the missing interface identifiers for NFM8, NFM9, and NFM10 limit the completeness of interface-level analysis.

Finally, the current methodology does not implement a fully optimized test execution algorithm. Rather, it establishes the structural and analytical foundation for developing a subsequent requirement aware test case prioritization strategy.

4. Proposed System Architecture and Design

4.1 Architecture Overview

The proposed architecture is designed as a modular, data-driven framework that integrates IoT requirement management, test-case traceability, bipartite network modeling, coverage assessment, and critical test-case identification.

The architecture follows a sequential analytical pipeline in which information from the requirements register, test-case repository, and traceability matrix is processed and transformed into a bipartite network. The resulting network is analyzed using graph-theoretic measures to determine requirement connectivity, test-case importance, traceability completeness, and potential testing gaps.

The architecture consists of seven major layers:

1. Data acquisition layer.

2. Data preprocessing and validation layer.

3. Traceability matrix construction layer.

4. Bipartite network modeling layer.

5. Network analysis and coverage assessment layer.

6. Critical test-case identification layer.

7. Visualization and decision-support layer.

Each layer performs a specific analytical function while exchanging structured information with the subsequent layer. The modular design allows the framework to be extended with additional testing criteria and optimization algorithms without modifying the underlying traceability representation.

4.1 Proposed Architecture Diagram

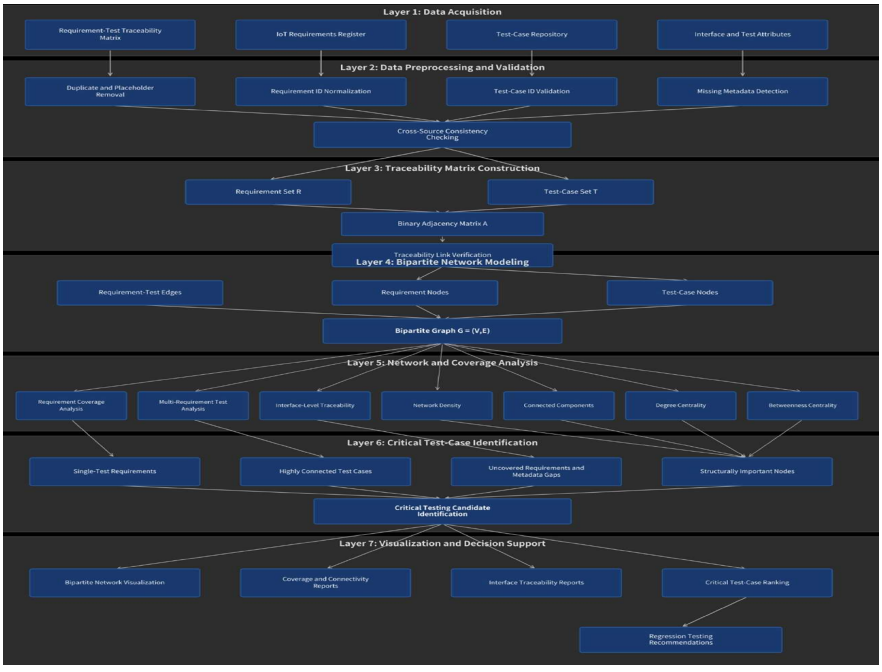

Figure 1. Proposed architecture of the bipartite network-based IoT requirement traceability, test coverage assessment, and critical test-case identification framework

4.2 Functional Description of Architecture Components

Layer 1: Data Acquisition

The data acquisition layer collects the information required for traceability analysis. It integrates the requirements register, test-case repository, requirement–test-case traceability matrix, and interface-related metadata. The requirements register provides requirement identifiers and categories, while the test-case repository provides test-case identifiers and associated testing attributes. The traceability matrix supplies the relationships between the two entity types.

Layer 2: Data Preprocessing and Validation

The preprocessing layer ensures that the collected information is suitable for network construction. Requirement identifiers are normalized, combined identifiers are separated into individual records, and invalid placeholder entries are removed. Duplicate records are examined, and missing interface identifiers are identified. Cross-source consistency checking is performed to determine whether the requirements recorded in the traceability matrix are also represented in the requirements register.

This layer also generates a data-quality report so that metadata inconsistencies are not confused with actual coverage deficiencies.

Layer 3: Traceability Matrix Construction

The normalized requirements and test cases are converted into a binary adjacency matrix. Rows represent requirements, columns represent test cases, and matrix entries indicate whether a traceability relationship exists.

This matrix provides the primary data structure for the subsequent graph-based analysis. It also allows the framework to calculate the number of test cases associated with each requirement and the number of requirements associated with each test case.

Layer 4: Bipartite Network Modeling

The adjacency matrix is transformed into a bipartite graph containing requirement nodes and test-case nodes. Edges represent the recorded relationships between requirements and test cases.

The network model preserves the distinction between the two node categories and provides a common representation for network-level and node-level analysis. This layer enables the identification of connected components, interconnected testing groups, and requirements or test cases with high connectivity.

Layer 5: Network and Coverage Analysis

This layer performs the primary quantitative analysis. Requirement coverage is calculated to identify uncovered requirements. Multi-requirement test analysis identifies test cases that verify multiple requirements. Interface-level analysis examines the distribution of testing relationships across interfaces.

Graph-theoretic measures, including network density, connected components, degree centrality, and betweenness centrality, are then calculated to characterize the network structure and identify structurally important nodes.

Layer 6: Critical Test-Case Identification

The critical test-case identification layer integrates the findings from coverage analysis and network centrality measures. It identifies highly connected test cases, test cases occupying important intermediary positions, and requirements with limited testing support.

The layer distinguishes between structurally important test cases and experimentally validated effective test cases. In the current implementation, the output represents structural criticality candidates. Actual fault-detection performance can be incorporated when execution history and defect data become available.

Layer 7: Visualization and Decision Support
The final layer converts the analytical results into interpretable reports and visual representations. It produces bipartite network diagrams, requirement coverage summaries, interface traceability visualizations, and test-case connectivity rankings.

These outputs support software testers and IoT system developers in understanding traceability relationships, identifying potential testing weaknesses, and planning regression testing activities.

4.3 Operational Workflow of the Proposed Framework

The operational workflow begins by importing the requirements register, test-case repository, and traceability matrix. The collected records are normalized and validated to remove invalid entries and identify metadata inconsistencies.

The validated records are then transformed into a binary adjacency matrix, which is subsequently converted into a bipartite graph. Once the network is constructed, the framework performs requirement coverage analysis, multi-requirement test-case analysis, and interface-level relationship analysis.

The graph is then evaluated using density, connected components, degree centrality, and betweenness centrality. These measures are integrated to identify structurally important requirements and test cases. Finally, the analytical results are presented through network visualizations, coverage reports, and critical test-case candidate rankings.

The workflow also supports iterative maintenance. When a requirement or test case changes, the corresponding records and traceability relationships can be updated, after which the affected matrix and network analyses can be recalculated.

4.4 Proposed Algorithm

The overall framework can be summarized using the following algorithm.

Input:

R_register = Requirements register

T_repository = Test-case repository

M = Requirement-test traceability matrix

I = Interface metadata

Output:

G = Bipartite traceability network

Coverage = Requirement coverage results

Centrality = Network centrality results

Critical_TC = Critical test-case candidates

Reports = Analytical reports and visualizations

Step 1: Data preprocessing

Normalize requirement identifiers

Normalize test-case identifiers

Separate combined requirement identifiers

Remove invalid placeholder records

Check duplicate traceability relationships

Detect missing interface identifiers

Reconcile requirements across data sources

Step 2: Construct traceability matrix

Identify valid requirement set R

Identify valid test-case set T

Initialize binary adjacency matrix A

For each requirement r in R:

For each test case t in T:

If a valid traceability link exists:

A[r,t] = 1

Else:

A[r,t] = 0

Step 3: Construct bipartite network

Create requirement nodes

Create test-case nodes

Create edges for all non-zero entries in A

Construct graph G = (R union T, E)

Step 4: Assess requirement coverage

Calculate test-case degree for each requirement

Identify uncovered requirements

Calculate overall requirement coverage

Step 5: Analyze test-case relationships

Calculate requirement degree

Calculate test-case degree

Identify multi-requirement test cases

Identify single-test requirements

Step 6: Analyze network structure

Calculate bipartite network density

Identify connected components

Calculate degree centrality

Calculate betweenness centrality

Step 7: Perform interface-level analysis

Group traceability links by interface

Calculate requirement and test-case counts

Identify missing interface metadata

Step 8: Identify critical test-case candidates

Examine test-case connectivity

Examine degree centrality

Examine betweenness centrality

Examine associated requirement coverage

Identify structurally important test cases

Step 9: Generate outputs

Generate network visualization

Generate coverage reports

Generate interface-level reports

Generate centrality rankings

Generate critical test-case candidate report

Return G, Coverage, Centrality, Critical_TC, Reports

Algorithm 1. Bipartite IoT Requirement Traceability and

Critical Test-Case Identification

4.5 Proposed Extension: Multi-Factor Test-Case Prioritization

An important future extension of the architecture is the integration of a multi-factor test-case prioritization mechanism. The current network analysis identifies structurally important test cases, but structural connectivity alone cannot determine which test case should be executed first.

The proposed extension can incorporate the following factors:

Factor	**Purpose**
Degree centrality	Measures the number of requirements covered by a test case
Betweenness centrality	Identifies intermediary positions in the traceability network
Requirement criticality	Reflects the importance of the associated system requirements
Fault-detection capability	Measures the ability of a test case to detect defects
Execution time	Represents the computational and operational cost
Interface complexity	Reflects the complexity of the associated interface relationships

A future optimization module could normalize these criteria and combine them into a weighted prioritization score:

$P(t_j) = \sum_{k=1}^{K} w_k f_k(t_j)$ where $P(t_j)$is the prioritisation score of test case

t_j, $f_k(t_j)$ represents the normalized value of criterion k, and $w_k$represents its corresponding weight.

For criteria such as fault-detection capability and requirement criticality, higher values may indicate greater priority. For execution time, a lower value may be preferred, depending on the optimization objective.

The weights should be determined and validated using an appropriate weighting or optimization procedure rather than assigned arbitrarily. This extension would enable the framework to progress from structural test-case identification toward resource aware regression test optimization.

Recommended research contribution

The resulting framework would provide three complementary capabilities:

1. Traceability intelligence: Establishes and analyzes requirement test case relationships.

2. Testing intelligence: Identifies coverage gaps, connectivity patterns, and structurally important test cases.

3. Optimization intelligence: Extends structural analysis with execution cost, requirement criticality, and empirical fault-detection information.

5. Requirement traceability and bipartite network analysis

The analysis below uses the uploaded Excel workbooks, particularly the IoT-system-traceability-matrix.xlsx, and cross-checks the requirements register and test-case attribute file.

5.1 Traceability matrix: initial findings

The traceability matrix contains:

- 82 valid requirement identifiers.
- 83 distinct test cases.
- 109 requirement–test-case relationships.
- 19 placeholder rows containing 0, which must be excluded.
- 3 valid requirement rows without an interface identifier.

The 109 relationships form the basis of the bipartite network analysis.

An important distinction is that the requirement register contains combined requirement identifiers (for example, several requirement IDs represented in a single cell). These need to be normalised before comparing the requirement register with the traceability matrix.

5.2 Requirement coverage analysis

100% -Requirement coverage

82 of 82 matrix requirements have linked test cases

109 - Traceability relationships

Distinct requirement–test-case links

16 - Multi-requirement test cases

Test cases linked to two or more requirements

63 - Connected components

Separate groups in the bipartite network

5. 2.1 Individual requirement coverage

“For each requirement, test-case coverage was calculated as:

$$TC\ Coverage_i = \frac{\text{Number of linked test case}}{83} \text{ x } 100$$

The highest-connected requirements are:

Requirement	Category	Test cases	TC coverage
DAR11	User devices	9	10.84%
RMM7	Data processing	4	4.82%
RMM3	Data processing	3	3.61%

RMM6	Data processing	3	3.61%
RPG5	Sensors & Actuators	3	33.61%
RSM7	Data processing	3	3.61%
RTC1	Data processing	3	3.61%
RTC9	Sensors & Actuators	3	33.61%
DAR21	User devices	2	2.41%
DAR22	User devices	2	2.41%
DAR5	User devices	2	2.41%
RMM4	Data processing	2	2.41%

Table 1. Connected Requirements

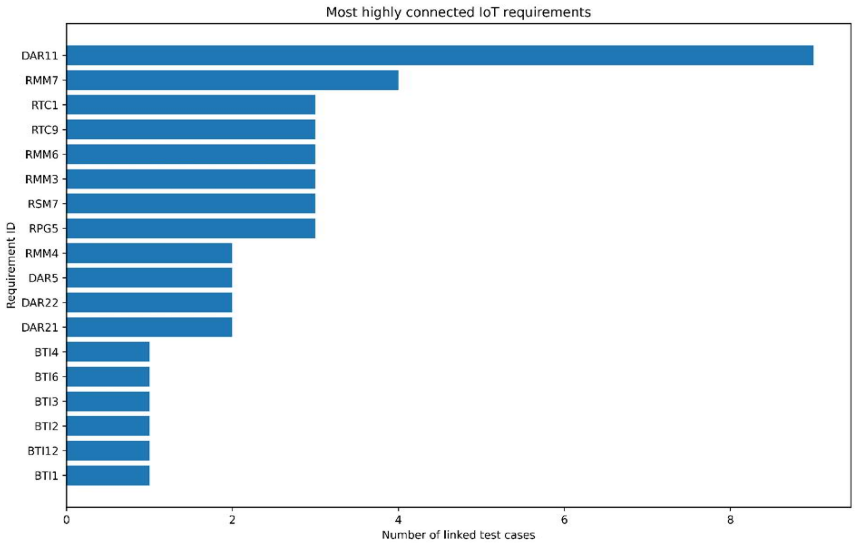

Figure 1. Requirement connectivity and test case coverage. The displayed illustration is a visualization reference; the actual dataset-derived figure is available below.

DAR11 has the highest test-case connectivity, with nine associated test cases. This indicates that the requirement has relatively extensive verification support.

RMM7 is the second most connected requirement, with four test cases. Several requirements in the RMM and RSM groups also have multiple associated test cases.

However, high connectivity does not necessarily imply higher testing quality. It indicates broader traceability and potential redundancy, not necessarily independent verification or better fault detection.

5.3 Requirements with no associated test cases

No uncovered requirements in the traceability matrix

All 82 valid requirement identifiers in the traceability matrix have at least one associated test case.

Indicator	Result
Total requirements in matrix	82
Requirements with test cases	82
Requirements without test cases	0
Requirement coverage	100%

This is a positive result. However, the requirement register and traceability matrix have a metadata inconsistency that needs correction, discussed under critical gaps.

5.4 Test cases associated with multiple requirements

The analysis identified:

Test-case relationship	Count
Linked to one requirement	67
Linked to multiple requirements	16
Linked to no requirements	0
Total	83

The most connected test cases are:

Test case	Associated requirements
TC-RMM3a	4
TC-RMM6a	3
TC-RMM6b	3
TC-RMM6c	3
TC-RPG5b	3
TC-RSM1	3

TC-RSM3	3
TC-RSM5	3
TC-RTC4	3
TC-DAR8	2
TC-NFM2	2
TC-NFM7	2
TC-RMM3b	2
TC-RPG1	2
TC-RPG5a	2
TC-RPG5c	2

TC-RMM3a is particularly important because it is associated with four requirements: RMM3, RMM4, RMM5 and RMM7.

Such test cases may provide efficient verification of multiple requirements. They should be considered carefully when designing optimized test suites.

5.5 Interface-level traceability analysis

The traceability matrix contains 12 distinct interface identifiers with valid associated requirements. Three additional requirement rows have no interface identifier.

The following table summarizes the interface-level relationships.

Interface	Requirements	Test cases	Relationships
F-7	19	29	30
39150...-5	8	8	12
F-13	3	4	10
F-9	10	8	10
F-11	8	5	8
39150...-7	8	8	8
39150...-10	8	6	8
F-6	4	4	7

F-8	6	6	6
F-72	2	4	4
F-10	2	2	2
F-12	1	1	1

Table 2. Interface Level Relationships

Interface IDs are abbreviated in this table for readability; complete identifiers are provided in the downloadable results.

Important outcome includes the following

- F-7 has the highest number of relationships, with 30.
- F-7 connects 19 requirements and 29 test cases.
- F-13 has 10 relationships involving only three requirements and four test cases.
- F-6 has seven relationships involving four requirements and four test cases.
- F-12 has only one requirement test case relationship.

The highest relationship count does not necessarily indicate the highest interface risk. It reflects the extent of traceability recorded for that interface.

Interface F-13 has a particularly concentrated relationship structure and deserves attention when examining testing dependencies.

5.6 Critical testing gap identification

Although the matrix indicates 100% requirement coverage, the analysis identifies three important data-quality and traceability issues.

Issue 1. Requirement-register inconsistency

High priority

Three requirement identifiers appear in the traceability matrix but are not separately listed in the requirements register:

- NFM8
- NFM9
- NFM10

These identifiers should be reconciled against the original requirement specifications.

Issue 2. Missing interface identifiers

Three requirement rows lack interface metadata. These correspond to NFM8, NFM9 and NFM10.

This prevents complete interface-level traceability for these requirements.

Issue 3. Limited test redundancy

Seventy requirements have exactly one associated test case.

Although these requirements are technically covered, their verification depends on a single test case. Additional independent test cases may be warranted for safety-critical, security-sensitive or high-impact requirements.

There are also 19 placeholder rows containing zero values in the matrix. These must not be interpreted as actual requirements.

Critical interpretation: The dataset demonstrates complete recorded requirement coverage, but complete coverage should not be equated with complete verification adequacy. Test-case quality, independence, fault-detection effectiveness and requirement criticality need to be considered separately.

5.7 Advanced bipartite network analysis

The traceability matrix was converted into an undirected bipartite graph.

Network structure - 82

Requirement nodes - Set R

109 edges

83 Test-case nodes

Set T

165 total nodes and 109 requirement–test-case relationships.

Each edge represents a recorded relationship between one requirement and one test case.

5.71 Network density

For a bipartite graph, density is calculated as:

$D = \frac{E}{|R| \times |T|}$ Where :

- E = 109 relationships.
- |R|=82 requirements.
- |T|= 83 test cases. $D = \frac{109}{82 \times 83} = 0.0160$

1.60% network density

The relatively low density indicates that only a small proportion of all theoretically possible requirement–test-case relationships are present.

This is not automatically a weakness. In software testing, a test case is generally expected to validate a subset of requirements rather than every requirement.

The result indicates a sparse traceability structure with relatively selective requirement test case associations.

5.72 Connected components

63 - Connected components identified

Component size	Number of components
12 nodes	1
10 nodes	2
6 nodes	1
4 nodes	1
3 nodes	7
2 nodes	51
Total	**63**

There are no isolated requirement or test case nodes. The largest component contains 12 nodes, while 51 components consist of only two nodes each.

The network is highly fragmented. Many requirements are connected to only one test case, forming independent requirement test case pairs. This suggests that the test architecture is largely requirement specific, with relatively few test cases providing cross-requirement verification. The larger components represent areas where multiple requirements and test cases are interconnected. This is important for regression testing because changes to a highly connected requirement or test case may affect multiple verification activities.

5.73 Degree centrality

Degree centrality identifies the number of connections associated with each node.

For requirements: $C_D(R_i) = \frac{Degree(R_i)}{83}$

For test cases: $C_D(T_j) = \frac{Degree(T_j)}{82}$

Requirement	Degree	Centrality
DAR11	9	0.1084
RMM7	4	0.0482
RMM3	3	0.0361
RMM6	3	0.0361
RPG5	3	0.0361

RSM7	3	0.0361
RTC1	3	0.0361
RTC9	3	0.0361

DAR11 is the most highly connected requirement.

Test-case degree centrality

Test case	Degree	Centrality
TC-RMM3a	4	0.0488
TC-RMM6a	3	0.0366
TC-RMM6b	3	0.0366
TC-RMM6c	3	0.0366
TC-RPG5b	3	0.0366
TC-RSM1	3	0.0366
TC-RSM3	3	0.0366
TC-RSM5	3	0.0366
TC-RTC4	3	0.0366

TC-RMM3a is the most highly connected test case.

5.74 Betweenness centrality

Betweenness centrality identifies nodes that lie on shortest paths between other nodes.

The standard normalized betweenness centrality for an undirected graph is:

$$C_B(v) = \frac{2}{(N-1)(N-2)} \sum_{\substack{s,t \\ s \neq v \neq t}} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Where:

- N=165 (total number of nodes in the graph)
- σ_{st}= number of shortest paths from node s to node t
- $\sigma_{st}(v)$= number of those shortest paths that pass through node v
- The summation is taken over all pairs of distinct nodes and that are different from

Equivalent expanded form (same value):

$$C_B(v) = \frac{2}{(N-1)(N-2)} \sum_{s \neq v} \sum_{\substack{t \neq v \\ t \neq s}} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

This normalization maps the betweenness score into the interval [0, 1].

Use the same notation as my original formulaExplain the summation limitsWhat is the difference between this and the original formula

The leading results are:

Node	Betweenness centrality
DAR11	0.002693
TC-RMM3a	0.002619
RMM7	0.002319
RSM7	0.002020
TC-RSM1	0.001122
TC-RSM3	0.001122
TC-RSM5	0.001122
RMM3	0.001047

DAR11 and TC-RMM3a demonstrate the strongest intermediary positions in their respective node groups.

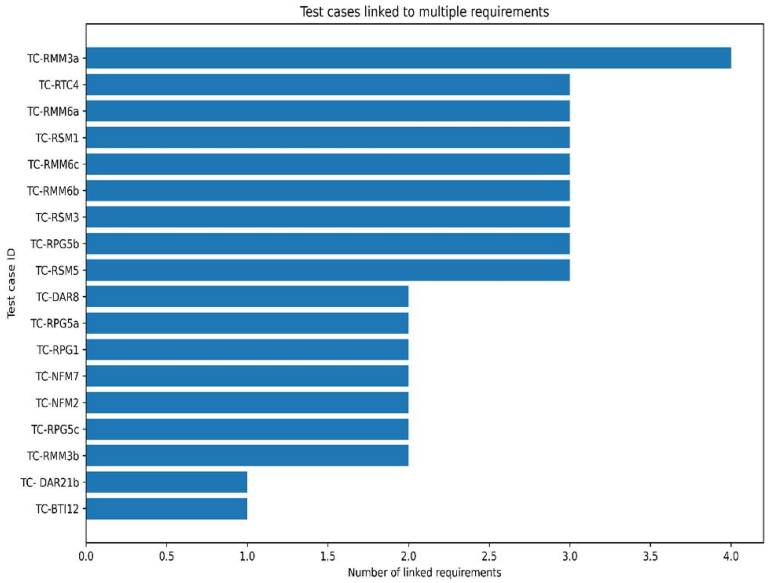

Figure 2. Test Case Frequency

RMM7 and RSM7 also exhibit relatively high betweenness centrality.

These nodes are useful candidates for impact analysis and regression test planning because their connections link multiple parts of the traceability structure.

However, betweenness values are small because the network contains many disconnected components. They should be interpreted comparatively rather than as evidence of strong global connectivity.

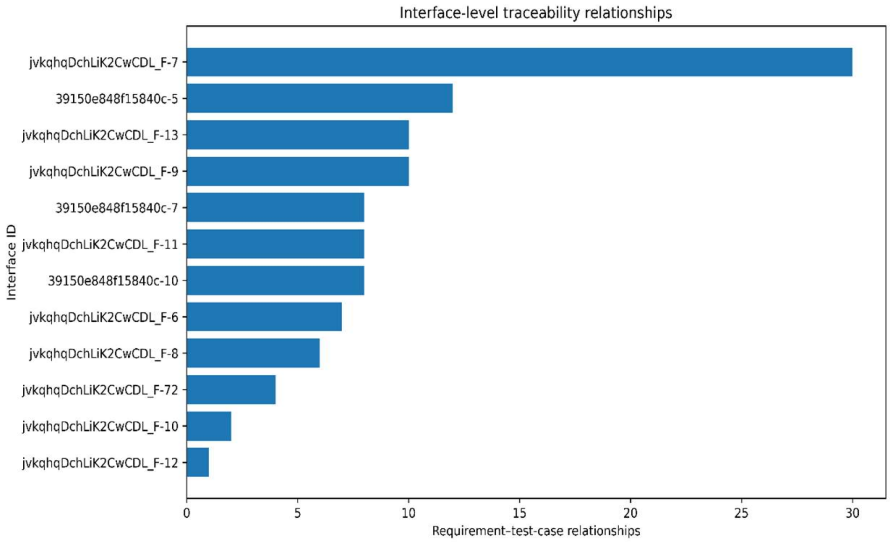

Figure 3. Interface Tracebility

5.8 Requirement coverage centrality and test-case importance centrality

For this analysis, two application-specific measures were defined using normalized degree centrality.

Measure	Definition	Interpretation
Requirement coverage centrality	Requirement degree / 83	Proportion of test cases associated with a requirement
Test-case importance centrality	Test-case degree / 82	Proportion of requirements associated with a test case

The highest requirement coverage centrality belongs to DAR11 (0.1084).

The highest test-case importance centrality belongs to TC-RMM3a (0.0488).

These measures identify highly connected nodes and provide a useful foundation for test-case prioritization.

They do not independently measure test quality or fault-detection effectiveness.

5.9 Dataset-derived network visualization



Figure 4. Bipartite Traceability Network

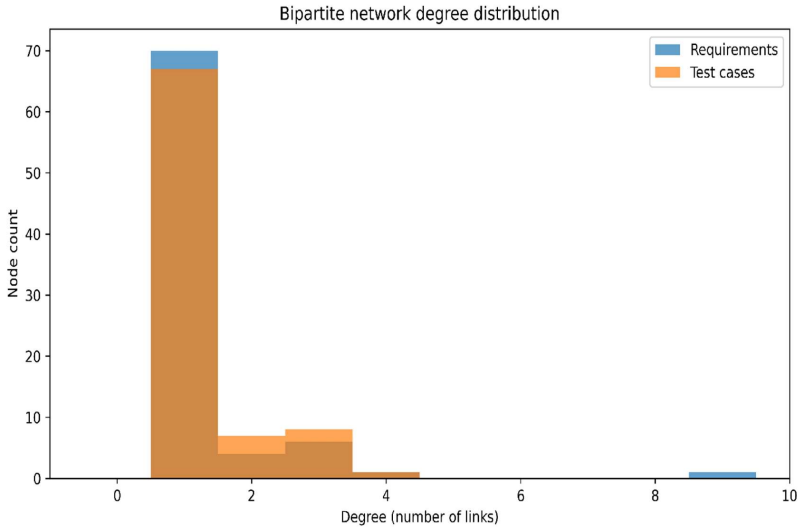

Figure 5. Degree Distribution

Conceptual representation of the two node classes. The actual network visualization is available as a high-resolution file below.

The generated network contains 82 requirement nodes and 83 test-case nodes. Node sizes represent connectivity, and the two node groups are visually distinguished.

The actual figure reveals several small connected groups and a few larger interconnected structures.

This supports the quantitative finding that the traceability network is sparse and fragmented rather than forming one integrated network.

5.9 Dataset-derived network visualization

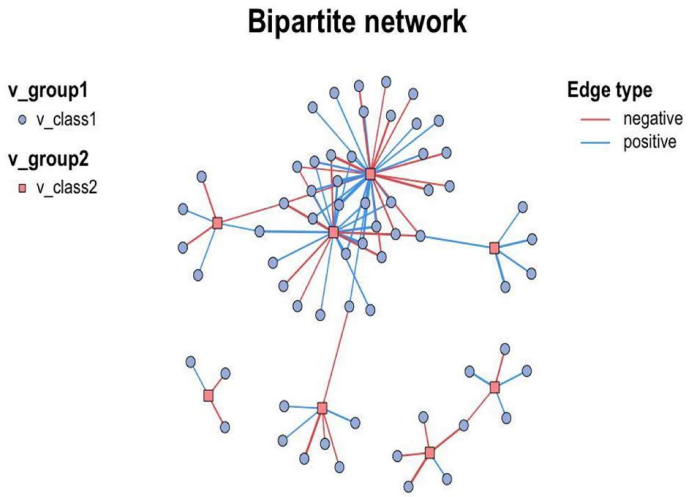

Figure 6. Bipartite network

Conceptual representation of the two node classes. The actual network visualization is available as a high-resolution file below.

The generated network contains 82 requirement nodes and 83 test-case nodes. Node sizes represent connectivity, and the two node groups are visually distinguished.

The actual figure reveals several small connected groups and a few larger interconnected structures.

This supports the quantitative finding that the traceability network is sparse and fragmented rather than forming one integrated network.

These findings provide a strong foundation for developing an intelligent requirement aware test-case prioritization framework. Such a framework could combine degree centrality, betweenness centrality, fault-detection assessment, execution time and requirement criticality to prioritize test cases more effectively.

One important limitation should be retained in the paper: network connectivity measures indicate structural importance, not actual fault-detection performance. Those two dimensions should be integrated in the next optimization stage rather than treated as interchangeable

6. Discussion

The application of the proposed bipartite network framework yields several critical insights into the structural dynamics of IoT requirement traceability and test coverage. First, the achievement of 100% recorded requirement coverage indicates comprehensive initial mapping; however, the discovery that 70 out of 82 requirements are linked to exactly one test case underscores a significant vulnerability in verification adequacy. In resource constrained and safety critical IoT environments, reliance on single point testing necessitates further risk assessment to determine if independent, redundant testing is required for high impact requirements.

Second, the network topology analysis reveals a highly sparse and fragmented structure, characterized by a low network density (1.60%) and 63 distinct connected components. This fragmentation suggests that the current testing architecture is predominantly requirement specific, with limited cross requirement verification. While this is typical in modular software design, the larger interconnected components (e.g., the 12-node component) represent critical areas for change impact analysis, as modifications within these clusters could cascade across multiple requirements and test cases during regression testing.

Third, the identification of central nodes, such as requirement DAR11 (highest degree and betweenness centrality) and test case TC-RMM3a (verifying four distinct requirements) demonstrates the framework's capability to pinpoint structural bottlenecks and high leverage test cases. These entities are prime candidates for regression test prioritization, as their failure or modification would disproportionately impact the overall verification status.

Furthermore, the framework proved highly effective as a metadata auditing tool. The automated reconciliation process successfully flagged inconsistencies, such as the absence of interface identifiers for requirements NFM8, NFM9, and NFM10, which would otherwise remain hidden in traditional spreadsheet based traceability matrices.

A crucial limitation acknowledged in this study is the distinction between structural importance and empirical effectiveness. Network centrality measures quantify relational connectivity but do not inherently reflect a test case's actual fault detection capability, execution reliability, or historical defect yield. Therefore, while the bipartite model establishes the structural foundation for test management, it must be coupled with empirical execution data to fully optimize regression testing strategies.

7. Conclusion

This study addressed the critical challenge of maintaining accurate requirement traceability and assessing test coverage in complex, heterogeneous IoT environments. We proposed and validated a novel bipartite network-based framework that unifies requirement traceability, coverage assessment, and critical test-case identification into a single analytical pipeline. By transforming traditional traceability matrices into bipartite graphs, the framework leverages graph-theoretic metrics such as degree centrality, betweenness centrality, and connected component analysis to evaluate testing completeness and identify structural dependencies.

The empirical analysis demonstrated the framework's efficacy in mapping the IoT testing landscape, revealing both robust structural coverage and underlying vulnerabilities, such as fragmented testing clusters and metadata inconsistencies. The proposed seven-layer modular architecture provides a scalable foundation for integrating advanced analytics and supports future extensions toward multi-factor test case prioritization.

Future work will focus on extending the current structural analysis by incorporating empirical execution metrics, including historical fault detection rates, execution time, and requirement criticality. By integrating these multi-factor variables into a weighted optimization model, the framework will evolve from a structural traceability tool into a comprehensive, resource-aware regression test optimization engine, further enhancing the reliability and maintainability of next-generation IoT systems.

References

[1] Ming, F. X., Habeeb, R. A. A., Md Nasaruddin, F. H. B., Gani, A. B. (2019). Real-time carbon dioxide monitoring based on IoT cloud technologies. In *Proceedings of the 2019 8th International Conference on Software and Computer Applications* (p. 517–521). Cairo, Egypt. https://doi.org/10.1145/3316615.3316728.

[2] Moin, S., Karim, A., Safdar, Z., Safdar, K., Ahmed, E., Imran, M. (2019). Securing IoTs in distributed blockchain: Analysis, requirements and open issues. *Future Generation Computer Systems, 100*, 325–343. https://doi.org/10.1016/j.future.2019.05.023.

[3] Chu, F., Yuan, S., Peng, Z. (2006). Using machine learning techniques to identify botnet traffic. In *Encyclopedia of Structural Health Monitoring* (pp. 967–974). Wiley.

[4] Xu, Z., Liu, W., Huang, J., Yang, C., Lu, J., Tan, H. (2020). Artificial intelligence for securing IoT services in edge computing: A survey. *Security and Communication Networks, 2020*, Article 8872586. https://doi.org/10.1155/2020/8872586.

[5] Xie, T., Gai, K., Zhu, L., Guo, Y., Choo, K.-K. R. (2023). Cross-chain-based trustworthy node identity governance in Internet of Things. *IEEE Internet of Things Journal, 10*(24), 21580–21594. https://doi.org/10.1109/JIOT.2023.3308130.

[6] Wang, Z., Guo, X., Tsuchiya, T. (2025). Graph-centric approaches for coverage optimization in software requirement testing. In *2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC)* (pp. 1270–1280). Toronto, ON, Canada. https://doi.org/10.1109/COMPSAC65507.2025.00160.

[7] Zhang, S., Chai, H., Li, Y., Qiu, B., Yue, L., Pan, R. (2025). PacketDiff: A flow guided diffusion model for network packet trace generation. *IEEE Internet of Things Journal, 12*(18), 38804–38819. https://doi.org/10.1109/JIOT.2025.3587658.

[8] da Silva, D. V., de Souza, B. P., Gonçalves, T. G., Travassos, G. H. (2021). A requirements engineering technology for the IoT software systems. *arXiv preprint*. https://arxiv.org/abs/2103.14348.

[9] Cleland-Huang, J., Gotel, O. C., Huffman Hayes, J., Mäder, P., Zisman, A. (2014). Software traceability: Trends and future directions. In *Future of Software Engineering Proceedings* (pp. 55–69). https://doi.org/10.1145/2593882.2593891.

[10] Rempel, P., Mäder, P. (2016). Preventing defects: The impact of requirements traceability completeness on software quality. *IEEE Transactions on Software Engineering, 43*(8), 777–797. https://doi.org/10.1109/TSE.2016.2622264.

[11] Wang, Y., Keung, J., Ma, X., Zhang, S., Li, Y. (2026). NFR-to-code traceability in a blockchain-IoT system: An empirical study. *arXiv preprint*. https://arxiv.org/abs/2607.25325.

[12] Pichappan, P. (2026). Requirement traceability and intelligent test selection for industrial IoT systems. *International Journal of Web Applications, 18*(2), 66–93.

[13] Xuan Tung, N., et al. (2026). Graph neural networks for next-generation-IoT: Recent advances and open challenges. *IEEE Communications Surveys & Tutorials, 28*, 2226–2262. https://doi.org/10.1109/COMST.2025.3613845

[14] Caiyun, L., Liu, C., Chen, X., et al. (2020). A novel data traceability model based on blockchain and digital watermarking in edge computing. *Journal of Physics: Conference Series, 1682*(1), Article 012041. https://doi.org/10.1088/1742-6596/1682/1/012041.

[15] Fu, W., Shi, H., Huang, J., et al. (2019). Spatial image data traceability and interaction mechanism based on alliance chain. In *Proceedings of ICSESS*. https://doi.org/10.1109/ICSESS47205.2019.9040778.

[16] Li, J., Zhai, H., Zhang, L., Luo, A., Zhang, C., & Yang, X. (2020). A fault tracing method of successive commutation failure for multi-infeed DC system. In *Proceedings of APPEEC* (p. 1–5). https://doi.org/10.1109/APPEEC48164.2020.9220679.

[17] Wang, Z., Wang, J., Chen, S. (2020). Fault location of strip steel surface quality defects on hot-rolling production line based on information fusion of historical cases and process data. *IEEE Access, 8*, 171240–171251. https://doi.org/10.1109/ACCESS.2020.3024582.

[18] Li, J., Yin, W., Yang, B., et al. (2023). Research on welding quality traceability model of offshore platform block construction process. *Computer Modeling in Engineering & Sciences, 2023*, Article 1. https://doi.org/10.32604/cmes.2022.020811.

[19] He, P., Li, X., Shen, W., Deng, S., Xiao, L., Zhang, Y. F. (2024). Traceability and analysis method for measurement laboratory testing data based on intelligent Internet of Things and deep belief network. *Journal of Intelligent Systems, 33*(1), Article 20240076. https://doi.org/10.1515/jisys-2024-0076.