PhD Thesis Defense

Functional End-to-End Testing of IoT Systems

Presented by

Jean Baptiste Minani

(ID:40166177)

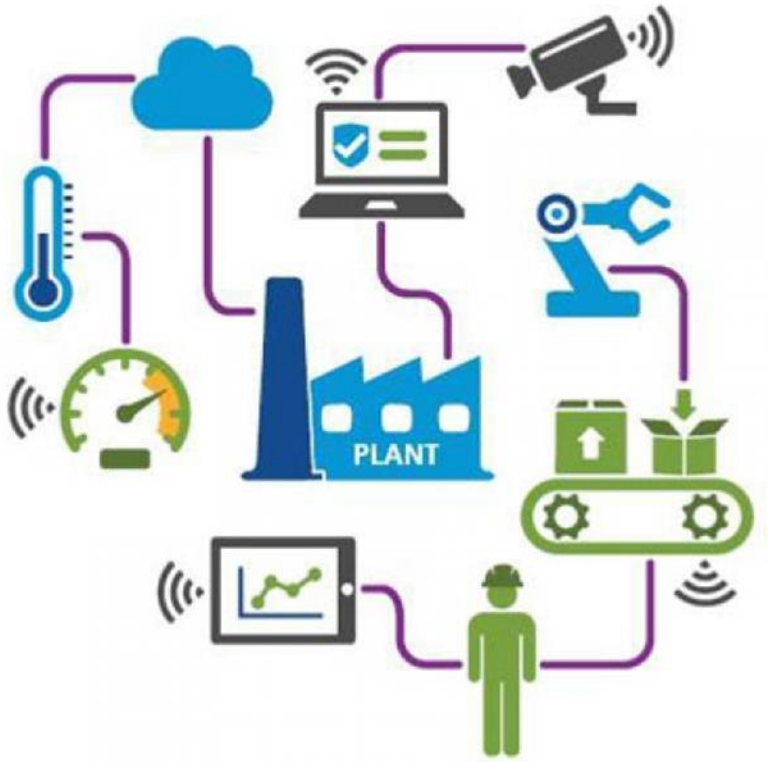
Supervised by

Dr. Yann-Gaël Guéhéneuc, Dr. Naouel Moha, and Dr. Fatima Sabir

Montréal, Québec, Canada

16 May 2025

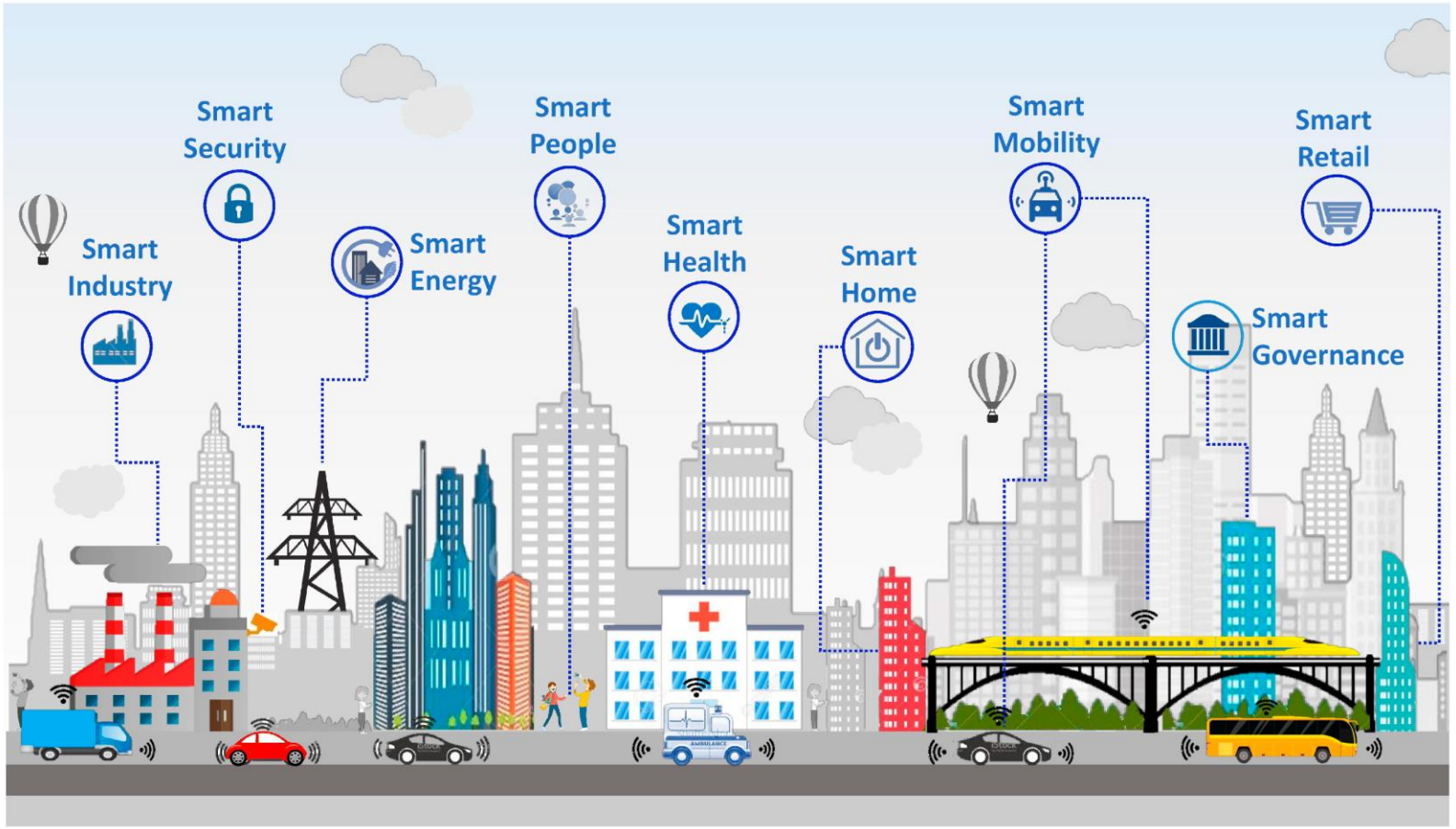
IoT Systems



“ An IoT system consists of **networked devices** whose purpose is to **collect** and **transmit** the collected data to provide **services** to end-users”

Xabier et al. ; Software engineering for the Internet of Things ; IEEE Software, 2017

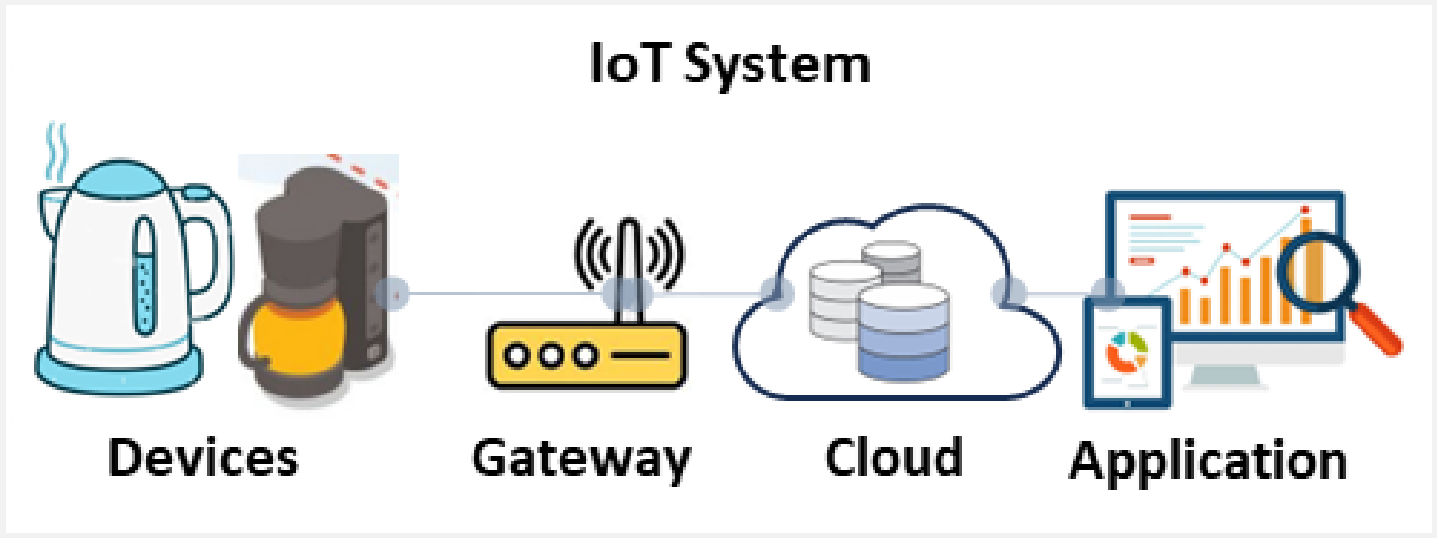
IoT System



Source:https://www.mdpi.com/1424-8220/18/9/2994

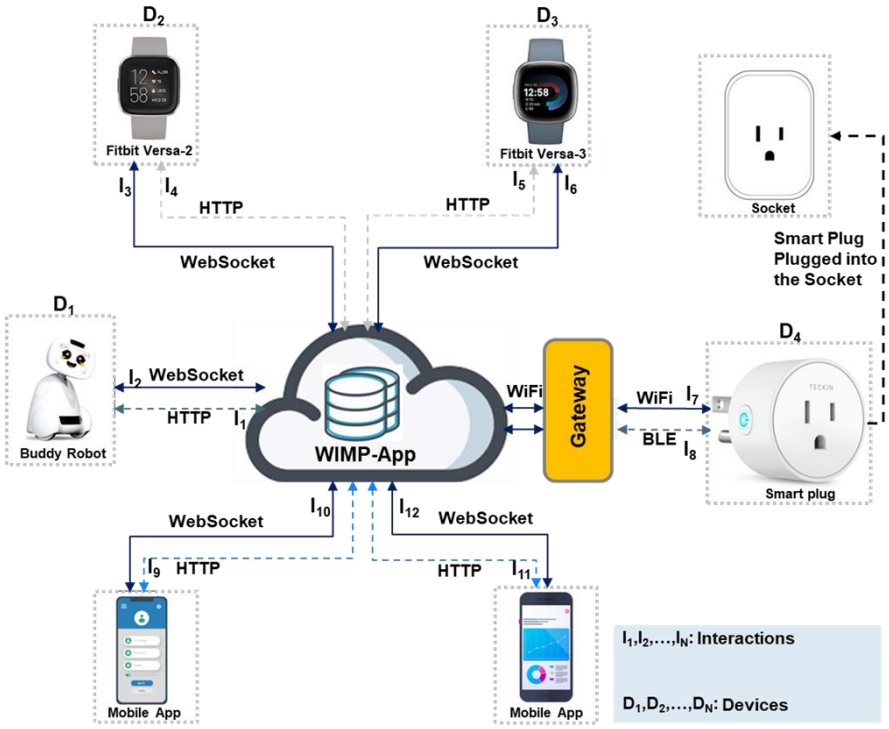
IoT System

- IoT system **commonly** consists of **4 layers**



Dorsemaine et al. ; A new approach to investigate IoT threats based on a four layer model; NOTERE, 2016

IoT System – Complexity

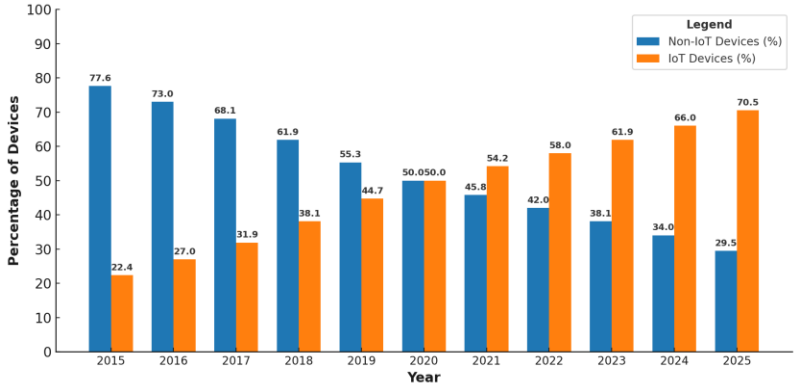


- Device diversity
  - Different OS, hardware, etc.
- Dynamic behavior
  - Changes in real-time
- Multiple protocols
  - HTTP, MQTT, CoAP, WebSocket, etc.
- Limited resources
  - Power, memory, etc.
- Interactions between components

Why IoT System

- **≈18.8B** connected IoT devices
- **Over 40B** connected IoT devices by **2030**
- **75%** of active devices are IoT

- **≈Over $1.1T** IoT market value
- **Over $1.56T** IoT market value by **2029**



Source: https://explodingtopics.com/blog/iot-stats

What is Testing

- A process of **executing** a program/system to find **errors** or **bugs**
- Bugs in any layer can cause **system failure**, leading to **financial loss** or **even loss of life**

Testing Traditional vs. IoT Systems

Traditional Systems

- Functional Testing
  - Unit testing, acceptance testing, etc.
- Non-functional Testing
  - Performance, security, etc.

IoT Systems

- Functional Testing
  - Unit testing, acceptance testing, etc.
- Non-functional Testing
  - Performance, security, etc.
- IoT Specific Testing
  - End-to-End (E2E)
    - Tests may pass despite hidden bugs in devices or protocols
  - Interaction between components or layers

It is possible to automate **functional end-to-end test generation** and systematically **execute the generated tests** to **detect bugs** in IoT systems

Research Design

Problem Identification

SLR

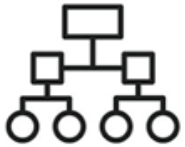
IoT Systems Testing (Academia)
- Objectives of Testing
- Tools and Approaches
- Challenges

Research Gap

Industry Study

IoT Systems Testing (Industry)
- Objectives of Testing
- Tools and Approaches
- Challenges

Taxonomy and Framework



Taxonomy

- Practical Guide for Testing IoT System
- Framework for Testing Technical Software Engineering Aspects

Technical SE Aspects

Approach and Experimentations

Functional E2E Testing

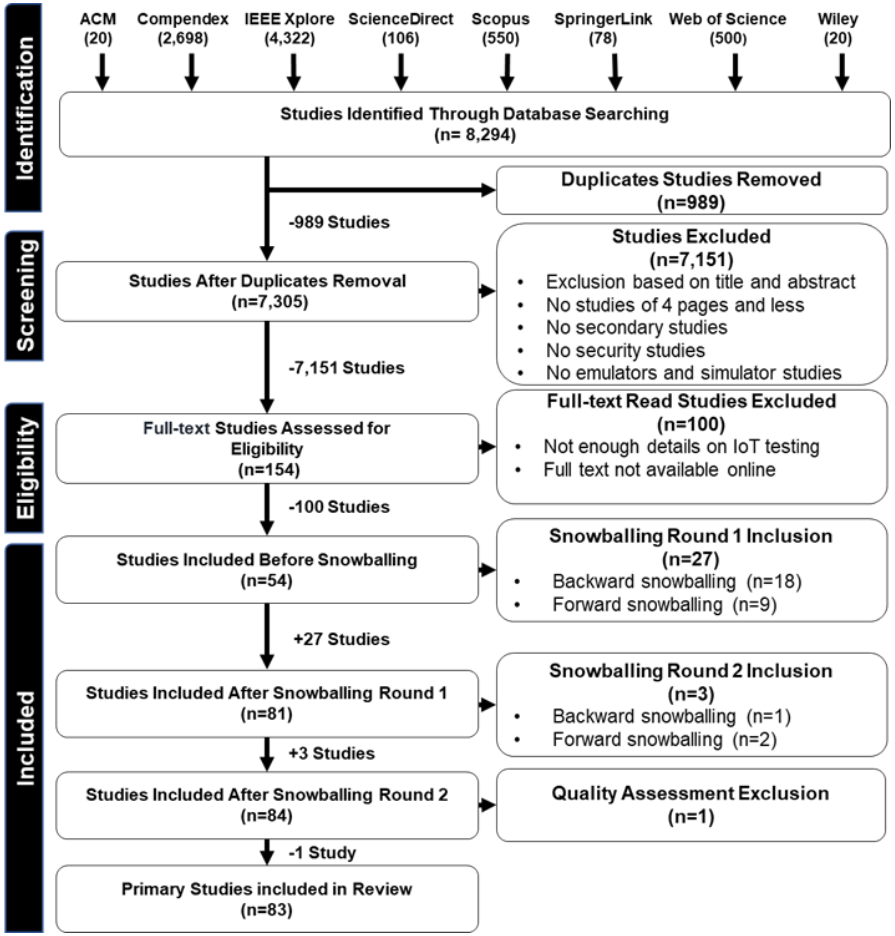
- Functional E2E Test Generation
- Metrics for Functional E2E Test Evaluation
- Experimentation

Systematic Literature Review (SLR)

- **83** IoT systems testing studies
- **Focus**
  - Testing objectives
  - Approaches
  - Tools
  - Challenges

SLR – Research Method

- **8,294** studies from **8** databases following PRISMA
- Inclusion criteria
  - **n=54**
- Snowballing
  - R1 (**+27**)
  - R2 (**+3**)
- Quality assessment
  - **-1**



PRISMA Flow for PSs Selection

SLR – Key Findings

- IoT systems testing in research
  - 47 testing objectives
    - Connectivity, device lifespan, etc.
  - 29 approaches
    - None fully automated
  - 19 tools
    - Selenium, PatrIoT, Héctor, etc.

SLR – Key Findings

- Challenges researchers identified
  - 14 key challenges
    - Lack of testing guide, lack of APIs, device diversity, data formats, etc.
    - **No study** addressed functional **E2E testing** in IoT

Industry Study

- Survey
  - **49** practitioners
- Interview
  - **9** practitioners
- Analysis
  - **4** Eclipse IoT survey datasets

Industry Study – Key Findings

- IoT testing in industry
  - 14 objectives
    - Performance
    - Connectivity
    - Interoperability
  - 13 testing approaches
    - Manual or semi-automated
  - 18 tools
    - Traditional
    - IoT-specific
- Top challenges
  - Resource constraints
  - Test case generation
    - Approaches
    - Tools

Industry Study – Key Findings

- Testers' Decisions
  - No fixed approach
    - Navigation 44%
    - Simulation 33%
    - Model-based 30%
  - Metrics vary by project
  - Bug detection
    - Logs 56%
    - Simulation 22%
    - Fault injection 11%
- Trends in IoT testing
  - Increasing concerns
    - Connectivity
    - Security
    - Diversity
    - Data format

Discussion

- Studies addressed **14 of 47** objectives
- Some academic tools **aren't used** in industry and vice versa
- Limitation of existing **tools** and **approaches**
- Studies focuses **most on devices**
- **No standard metrics** for testing

Threats to Validity

- Missing studies
  - **Security** and **non-English** studies
- **Manual extraction**
- **Low generalizability**
  - Only 49 practitioners
- Participant bias
  - **Incomplete** or **ambiguous** responses

Challenges

- C1 - **Lack of testing guide** for IoT systems
- C2 - **Automation** gap in IoT system testing
- C3 - **Limited focus** on functional E2E testing in IoT systems
- C4 – Lack of **test case generation tools** and **approaches** for IoT systems

We now examine **studies** that address **functional testing** in IoT systems relevant to our research focus

Related Work (1/4)

- Olianas et al. & Leotta et al.

Focus on **functional testing** of the **application layer**, but **are limited** to **functionalities** with **visible GUI feedback**

Olianas et al.; "MATTER: A tool for generating end-to-end IoT test scripts"; Software Quality Journal, 2022
Leotta et al.; "An acceptance testing approach for Internet of Things systems"; IET Software, 2018

Related Work (2/4)

- Almeida et al.

Focuses on functional **test case generation** for **application layer**, but requires **significant manual effort** to create and manage the feature models

Almeida et al.; "A Pattern-Based Test Platform for Families of Smart Health Products"; International Conference on Software Quality, Reliability and Security (QRS), 2022

Related Work (3/4)

- Wang et al.

An approach to **generate executable tests** focusing on acceptance testing of **embedded systems**

Wang et al.; Automatic Generation of Acceptance Test Cases From Use Case Specifications: An NLP-Based Approach; TSE, 2020

Related Work (4/4)

- Hu et al.

Approach for combinatorial **testing path selection** for IoT systems focusing on **identifying which devices** should be tested **to detect more bugs**, but **does not generate tests** for the entire IoT system

Hu et al.; "CT-IoT: A Combinatorial Testing-based Path Selection Framework for effective IoT Testing"; EMSE, 2022

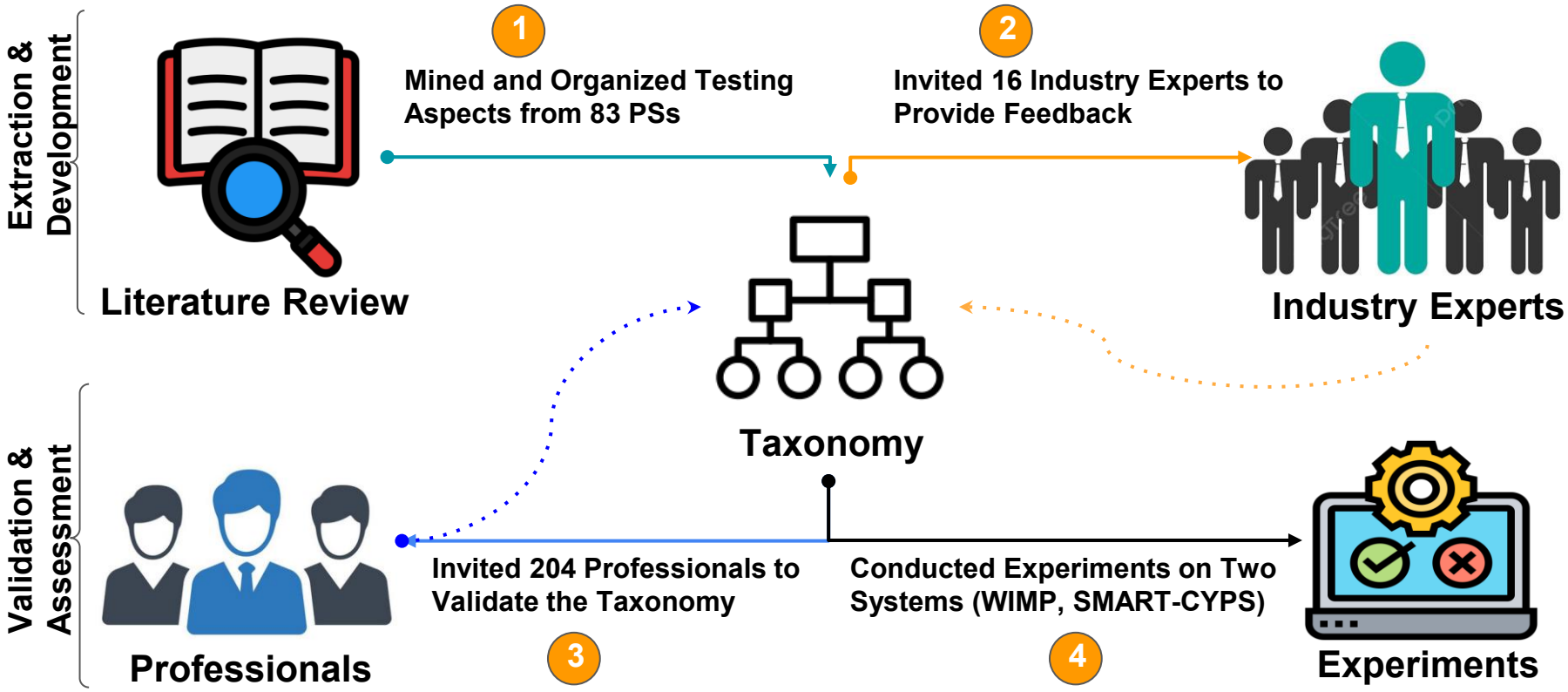
Existing approaches are **few** and **limited** to specific layers of IoT system, require a **lot of manual effort**, and rely on a detailed **behavioral model** or **source code** of the system under test (SUT)

We now address the **lack of a testing guide** (C1) in IoT systems testing

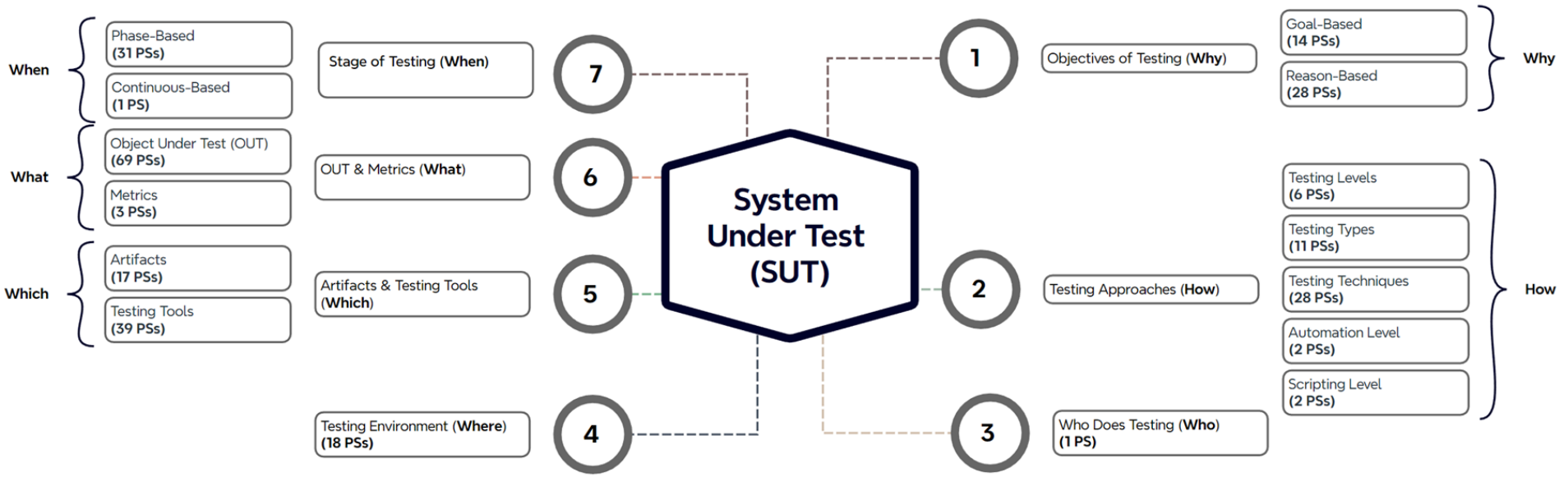
Taxonomy – Introduction

- **Practitioners** are not aware of various **testing aspects**
- Taxonomy **organizes testing aspects** of IoT systems **to guide the practitioners**

Taxonomy – Research Method

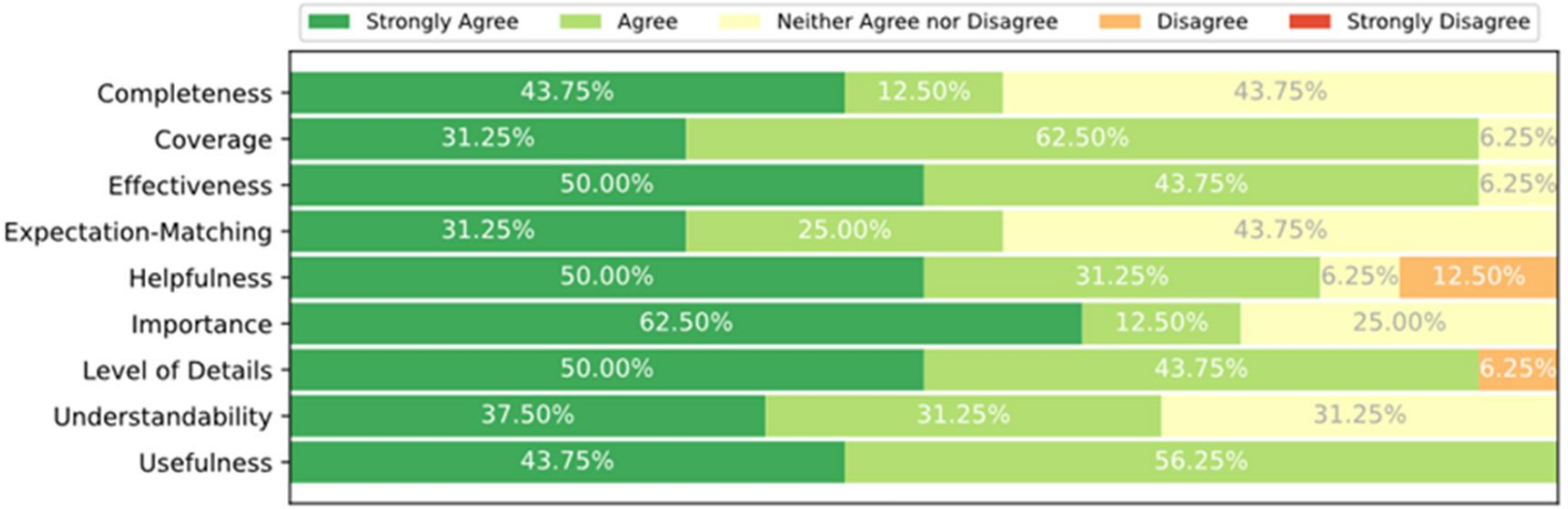


Taxonomy – Description



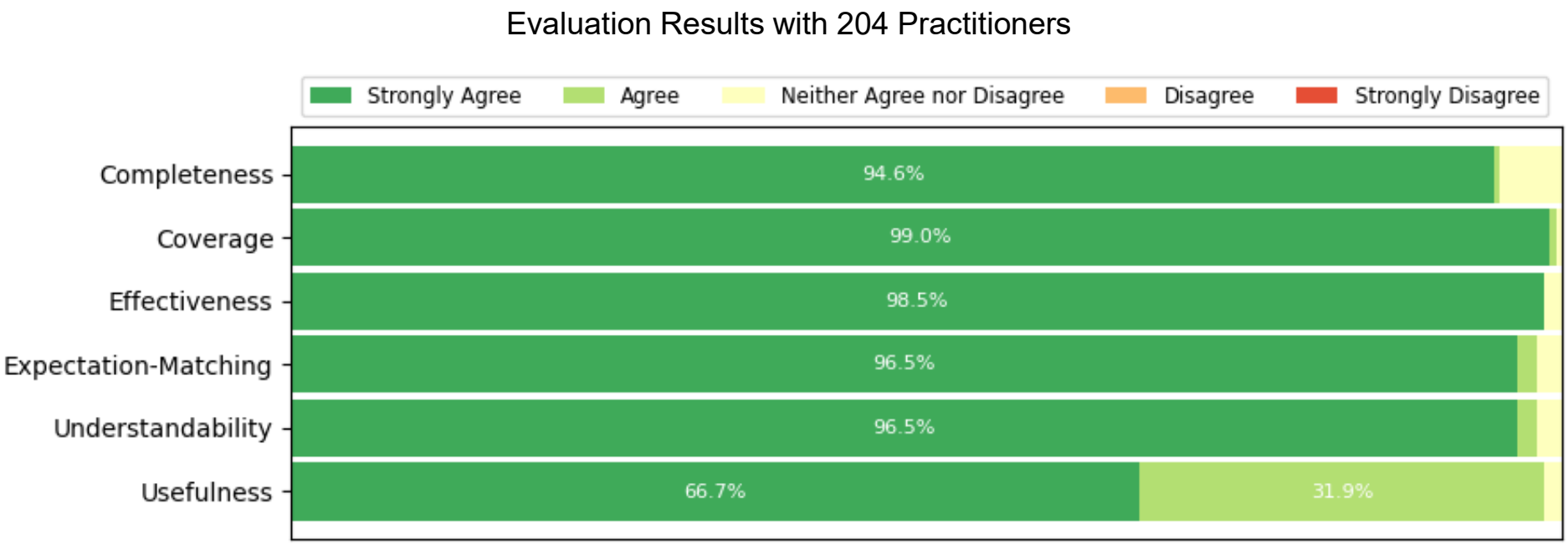
Taxonomy – Validation

Evaluation Results with 16 Practitioners



12.5% found it less helpful **6.25%** found it not detailed enough

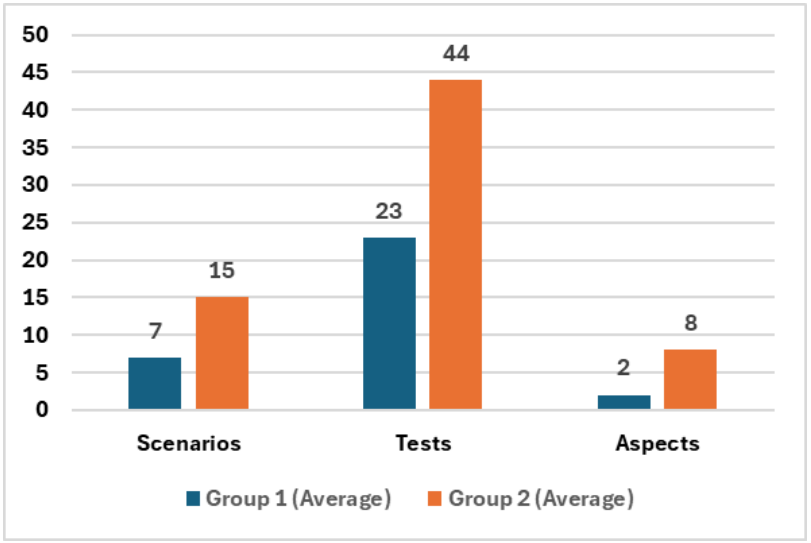
Taxonomy – Validation



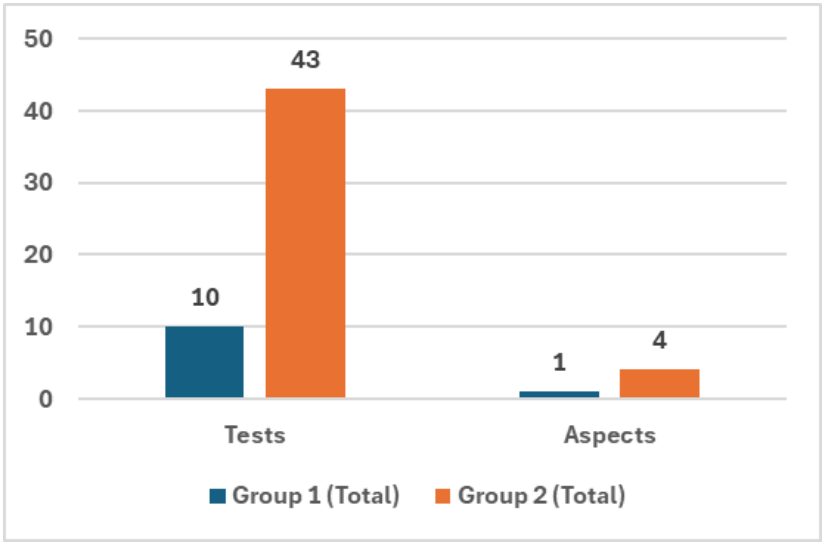
No disagreement on evaluation criteria **31.9%** found it **less useful**

Taxonomy – Empirical Evaluation

Test Coverage Results for WIMP



(a) All Results

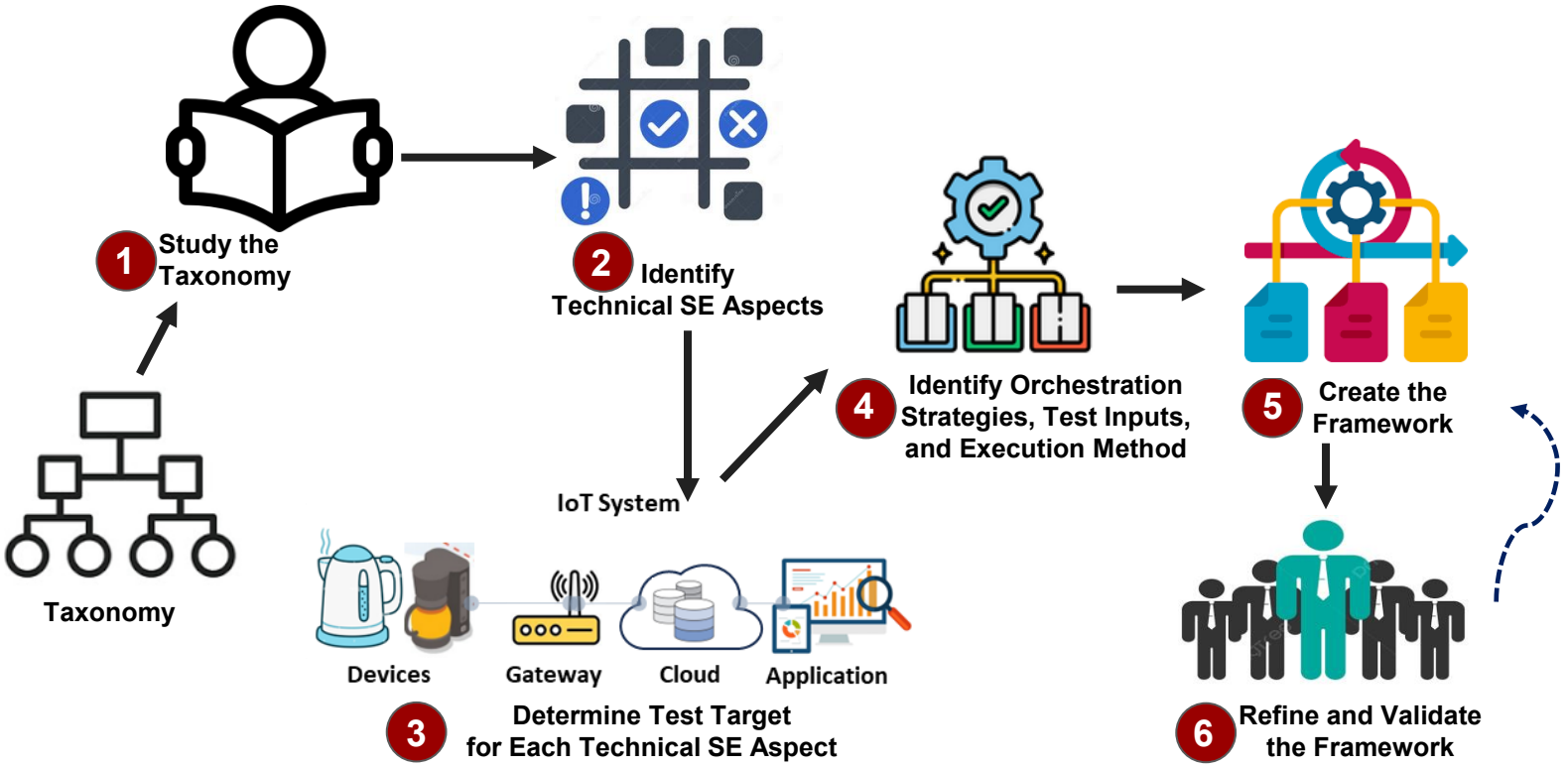


(b) Only IoT specific Results

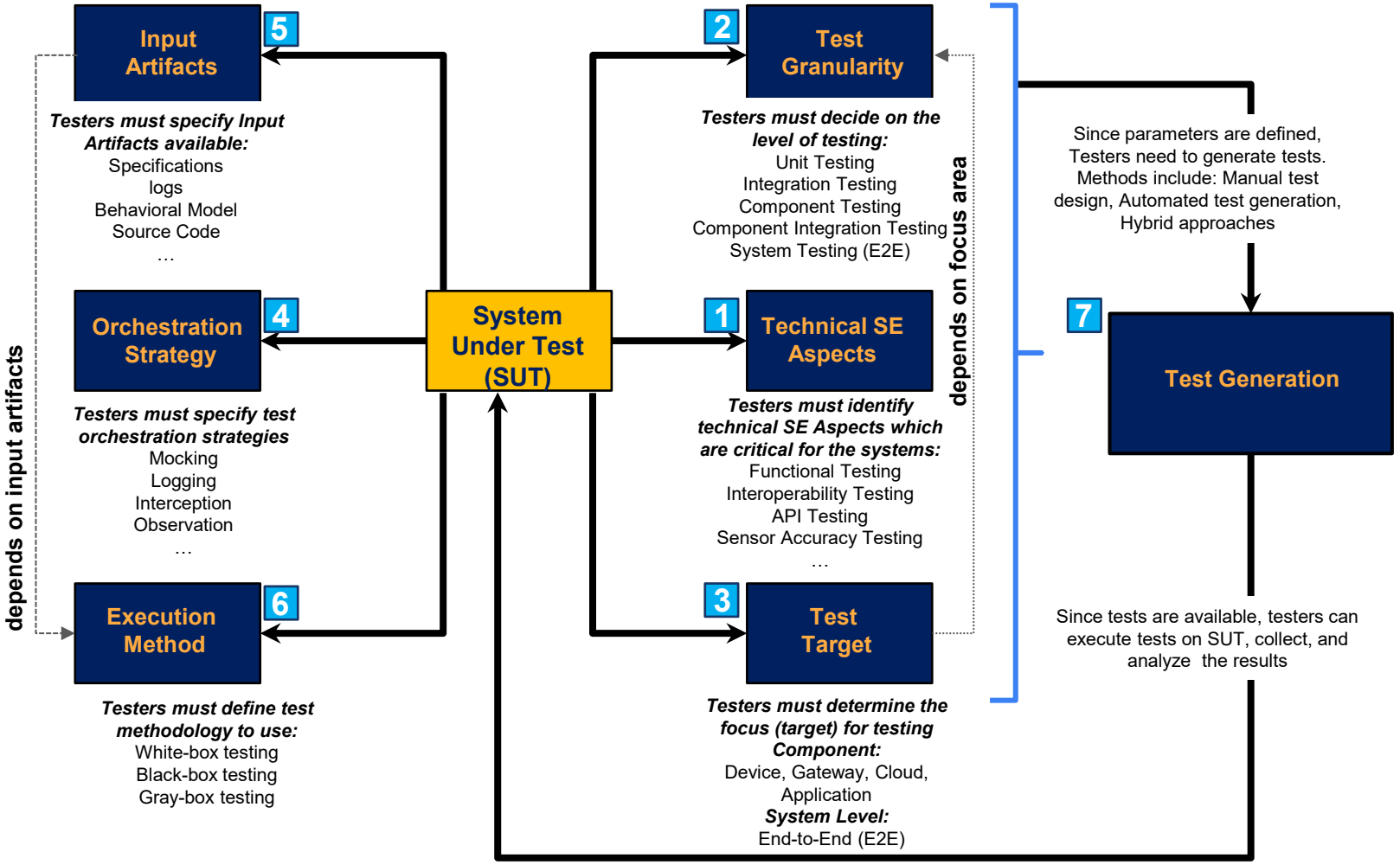
Participants **guided by the taxonomy performed** better

The taxonomy covers **aspects** beyond software engineers' scope, including **security**, **infrastructure**, and **communication**, and leads to a **framework** that guides the testing of **SE aspects** in IoT systems

Framework – Research Method

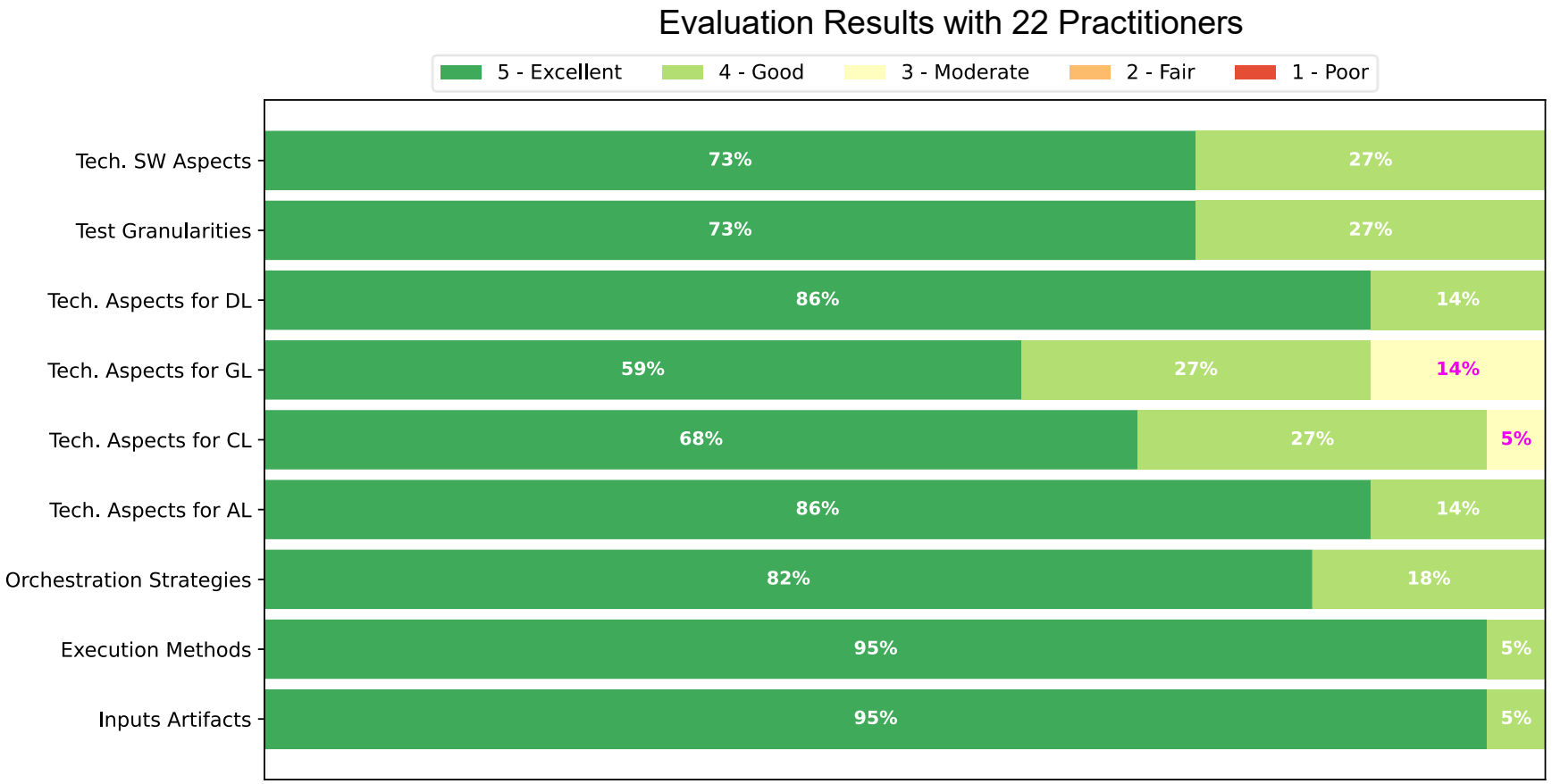


Framework – Description



Minani et al.; "TISSEA: A Framework for Testing IoT Systems Based on Technical Software Engineering Aspects"; IEEE IoTJ (Under review); 2025

Framework – Validation



Results show **strong agreement** from practitioners

Discussion

- **Some practitioners** prefer **automated tools** and **approaches** over taxonomy
- Taxonomy **available online** for access **and regular updates**

Taxonomy

Threats to Validity

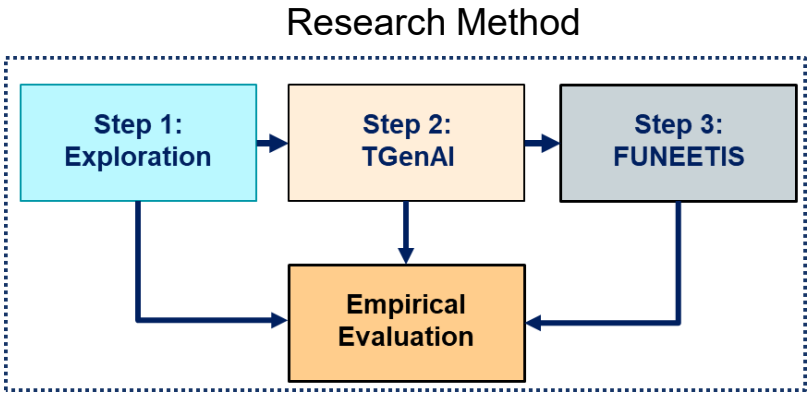
- **Limited evaluation**
  - **Small-scale** systems and **few use** cases
- Participant & survey bias
  - Survey **responses** may be **subjective**
  - Tester **experience** and **commitment varied**
- **Limited value** in taxonomy alone
  - **Tools** and **approaches** preferred

We now apply TISSEA to prioritize **functional E2E testing** of IoT systems

E2E Testing – Introduction

- Focused on **test generation** and **execution** for **functional E2E testing** of IoT systems
- **Functional testing** verifies that a system **behaves** as specified
- **E2E functional testing** ensures that data **flows correctly** across components and the **expected outcome** is achieved

E2E Testing – Introduction

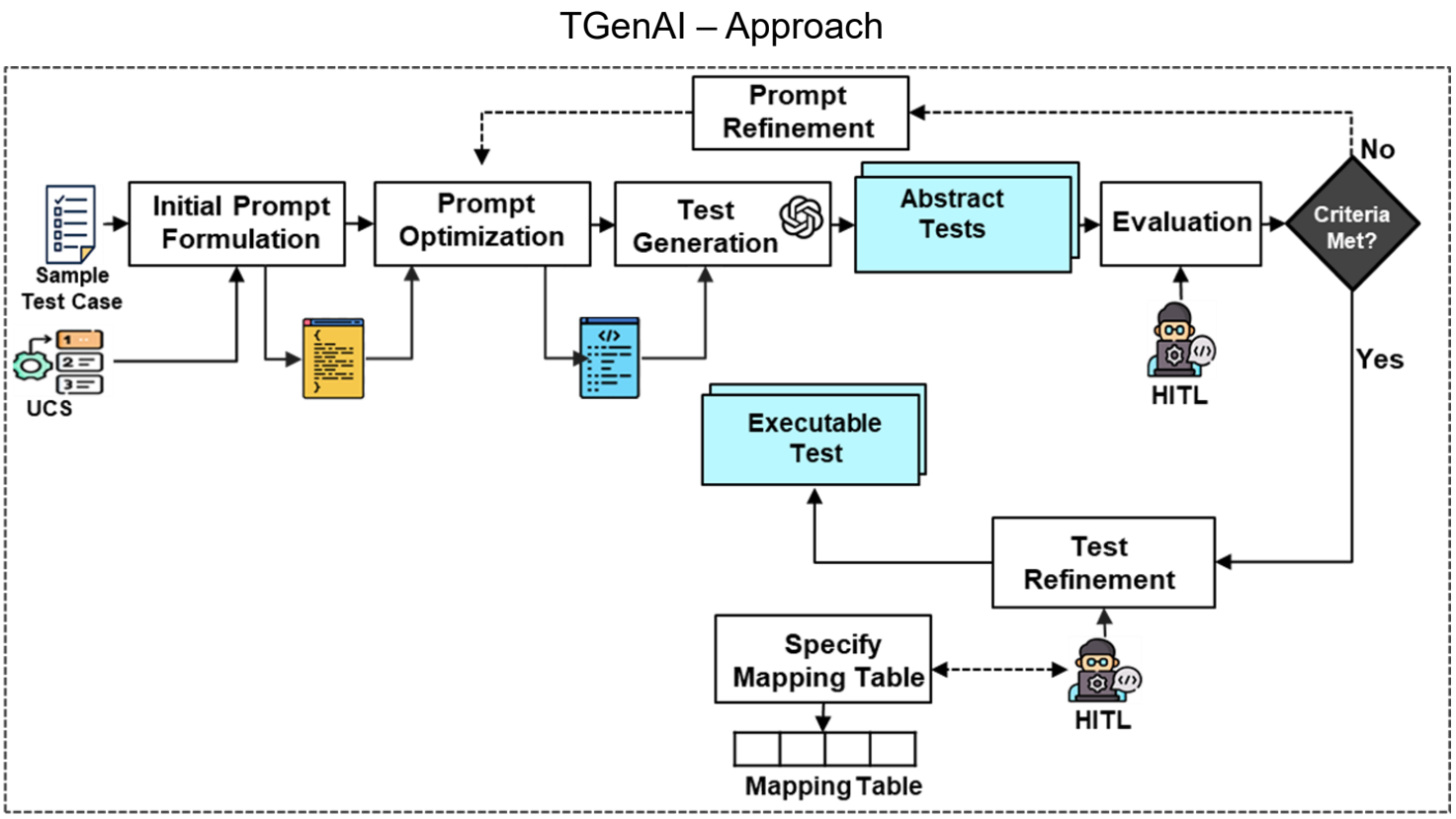


- Exploration
  - Custom (**CA**)
  - Single-Stage LLM (**SSLA**)
  - Multi-Stage LLM (**MSLA**)
  - Hybrid (**HA**)
- TGenAI
  - SSLA
- FUNEETIS
  - CA

Zacharie, **Minani** et al.; "Test Generation from Use Case Specifications for IoT Systems: Custom, LLM-Based, and Hybrid Approaches"; IEEE International Conference on Software Testing, Verification and Validation (ICST); 2025

We now focus on TGenAI, LLM-based approach for functional E2E test generation

TGenAI – Approach



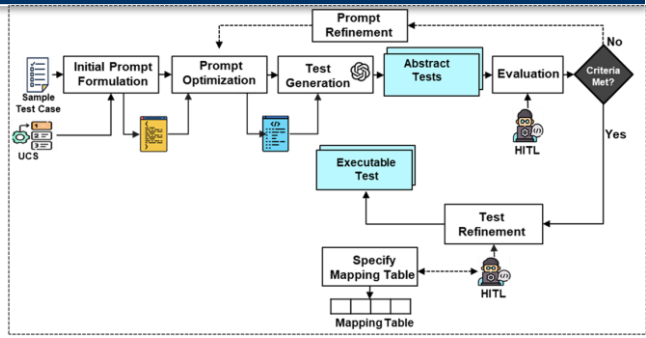
TGenAI – Approach

```
"steps": [
  {
    "entity": "fitbit",
    "operation": "sendHeartData",
    "inputs": {"heartrate": "80 bpm"},
    "target": {"protocol": "http","name": "smartphone"},
    "expectations": {"execution_status": "OK"}
  },
  {
    "entity": "smartphone",
    "operation": "sendHeartData",
    "inputs": {"heartrate": "80 bpm"},
    "target": { "protocol": "websocket","name": "cloudapp"},
    "expectations": {"execution_status": "OK" }
  },
  {
    "entity": "cloudApp",
    "operation": "sendMoveCommand",
    "inputs": {"distance": "1","speed": "1"},
    "target": {"protocol": "websocket","name": "buddy"},
    "expectations": {"execution_status": "OK" }
  },
  {
    "entity": "buddy",
    "operation": "sendMoveStatus",
    "inputs": { "execution_status": "wheel_move_finished" },
    "target": {"protocol": "websocket","name": "cloudapp"},
    "expectations": {"execution_status": "WHEEL_MOVE_FINISHED"}
  }
]
```

Sample E2E Test

- Each step has
  - Entity (i.e., sending node)
  - Operation (i.e., action)
  - Inputs (e.g., distance)
  - Target (i.e., receiving node)
  - Expectation (i.e., outcome)

TGenAI – Approach



- Prompt in three parts
  - **Task description**
  - **Input** (i.e., UCS)
  - **Example** of tests
- Prompt optimization
  - **SAMMO**
- Test generation
  - **Tool** using **LLM API**
- Evaluation
  - **Coverage**
  - **Correctness**
- Prompt refinement
  - **Refines**
  - **Regenerate**
- Test refinement
  - **Adds missing** details for execution

UCS: Use Case Specification

TGenAI – Approach

Initial Prompt

Generate all possible tests from the following Use Case Specification:
<Insert_UCS_Here>

ensuring all nodes, operations, protocols, and expected outcomes are captured.

Below is the example of UCS and corresponding tests:
<Insert_UCS_And_Tests_Here>

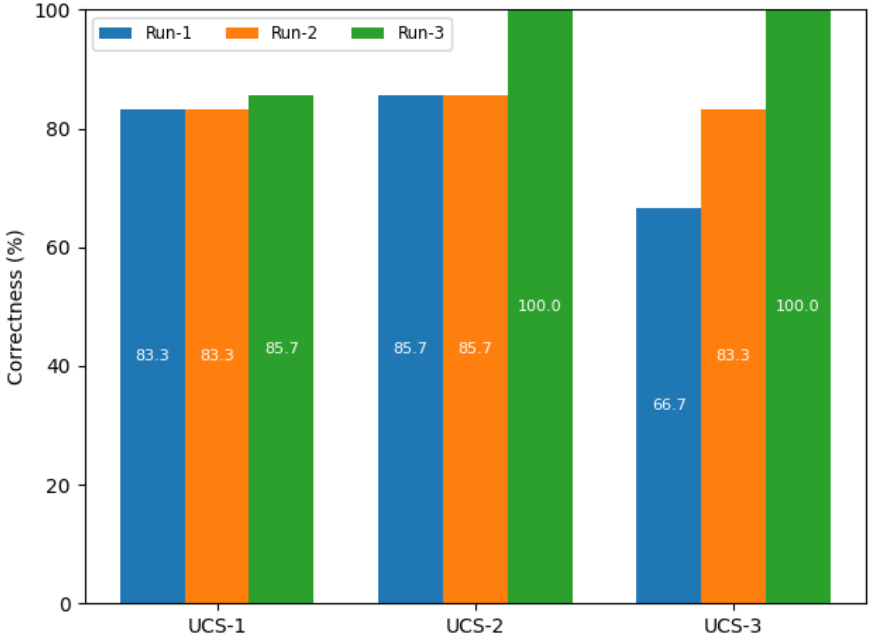
Optimized Prompt

****Instruction****
Generate Test Cases from Use Case Specification
****Input: Use Case Specification****
<INSERT_UCS_HERE>
****Examples of tests to generate****
<Example_of_UCS_And_Generated_Tests>
****Note****
Ensure that the tests cover all nodes, operations, protocols, and expected outcomes for basic and alternative flows.

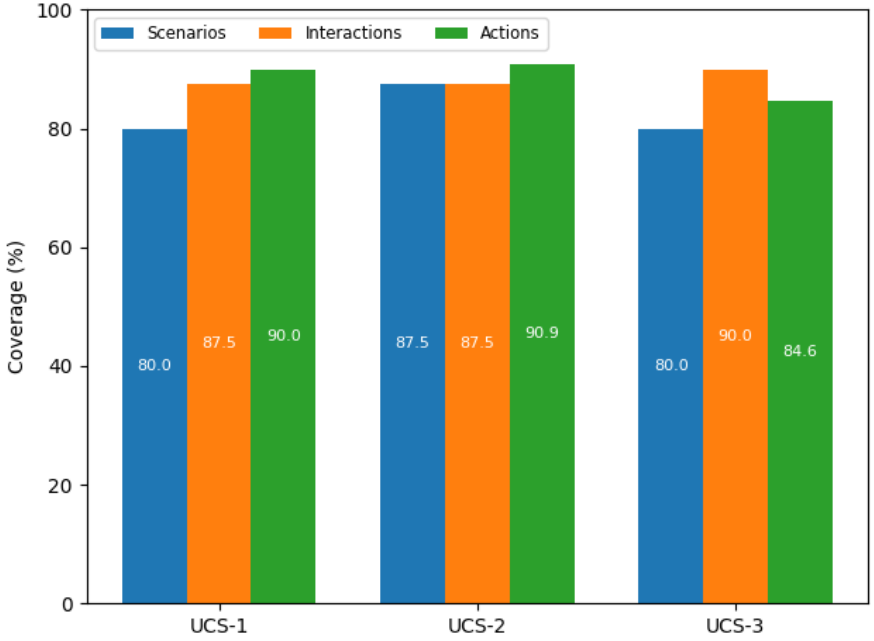
Prompt Template

TGenAI – Results

Correctness and Coverage



Correctness of Generated Tests



Metric Coverage in Generated Tests

TGenAI – Results

TGenAI vs. Manual Generation

Test Generation Time (in seconds)

	TGenAI			Engineer		
Model	**UCS1**	**UCS2**	**UCS3**	**UCS1**	**UCS2**	**UCS3**
GPT-4o-mini	17	17	20	3921	2374	1580
GPT-4o	55	69	68	3921	2374	1580
GPT-4 Turbo	113	109	102	3921	2374	1580
GPT-3.5 Turbo	142	132	138	3921	2374	1580

Tests Generated

	TGenAI			Engineer		
Model	**UCS1**	**UCS2**	**UCS3**	**UCS1**	**UCS2**	**UCS3**
GPT-4o-mini	5	8	6	6	5	5
GPT-4o	6	7	6	6	5	5
GPT-4 Turbo	5	6	5	6	5	5
GPT-3.5 Turbo	10	9	19	6	5	5

TGenAI reduces **test generation time** significantly

TGenAI sometimes generates **more tests** than manual efforts

TGenAI – Threats to Validity

- Limited validation
  - **One IoT system** evaluated
- LLM selection
  - Results based on **few LLMs**
- LLM parameters
  - Used **default parameters**
- Prompt optimization tool
  - Used SAMMO

TGenAI – Conclusion

- TGenAI showed **potential** for E2E test generation
- Needed **significant HITL** intervention
- TGenAI
  - **Inconsistencies**
    - **Missing** some scenarios
    - Generating **incorrect tests**
  - Led to the development of **FUNEETIS**

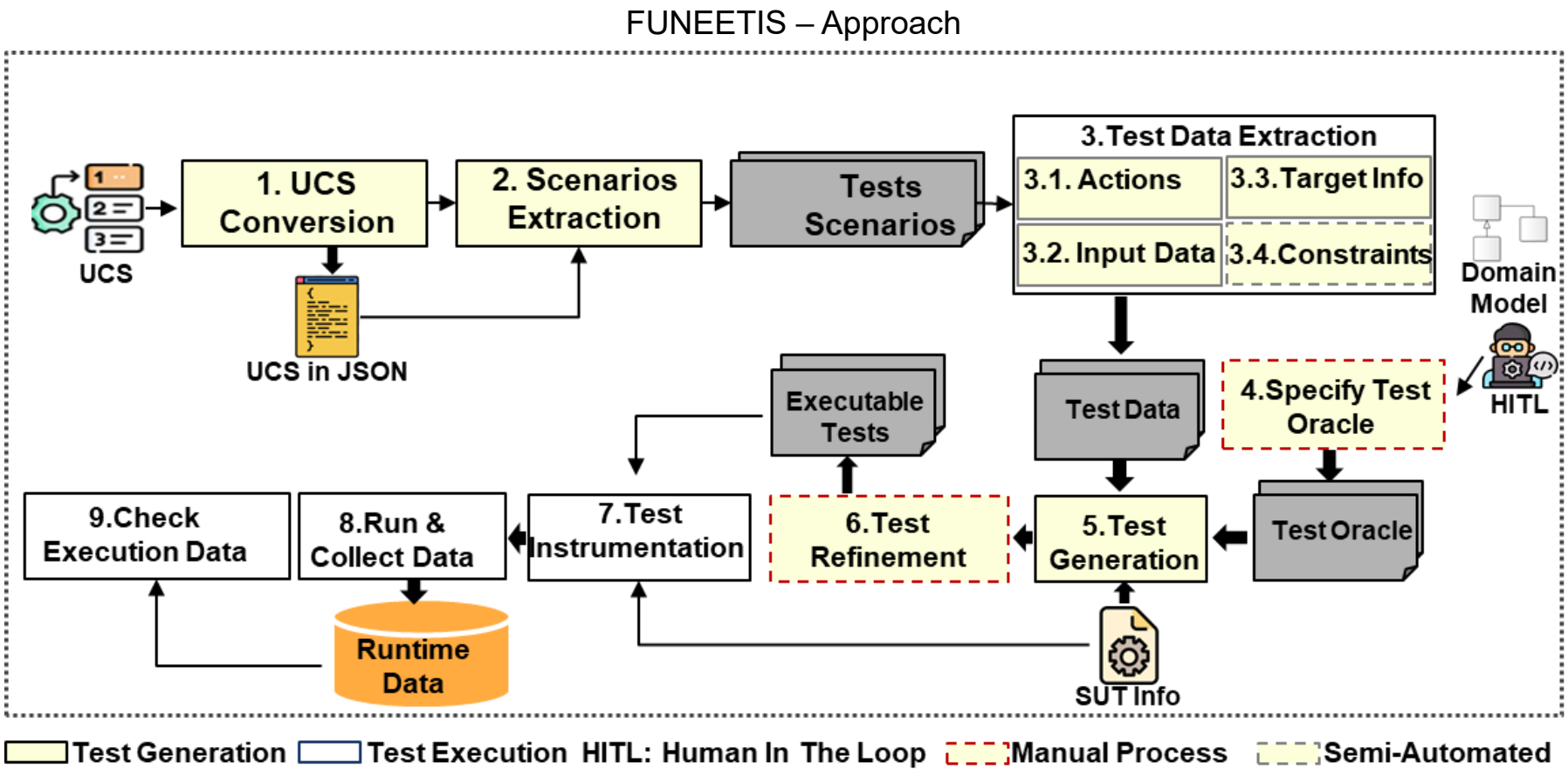
HITL: Human-In-The-Loop

We proposed FUNEETIS, a functional E2E test generation and execution approach aiming to minimize inconsistencies and manual effort observed in TGenAI

FUNEETIS – Introduction

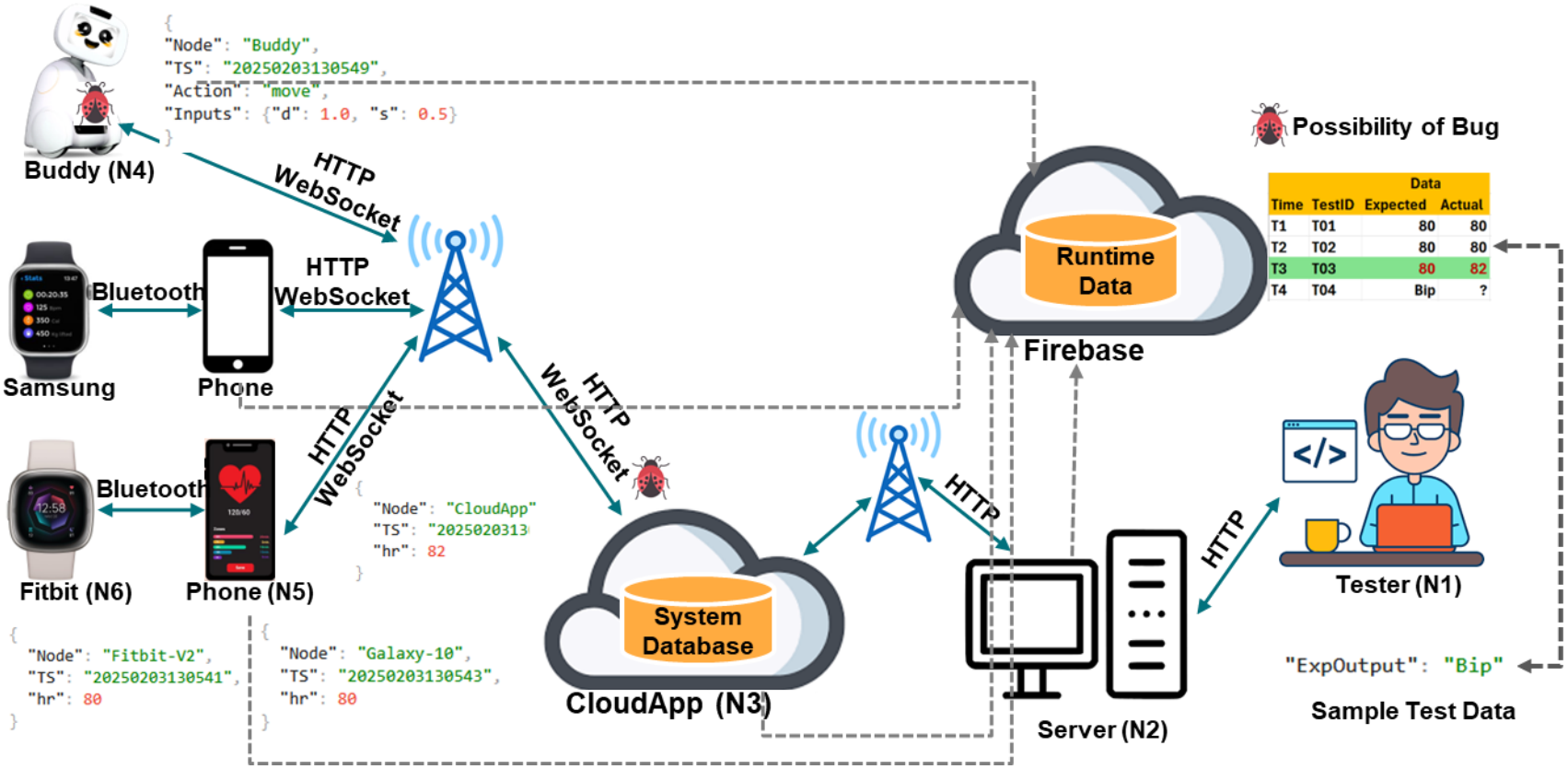
- FUNEETIS
  - Depends on **UCSs** written following **RUCM** with **controlled keywords**, to enable data extraction
  - Captures **runtime execution data** and validates tests against **real execution behavior**

FUNEETIS – Approach



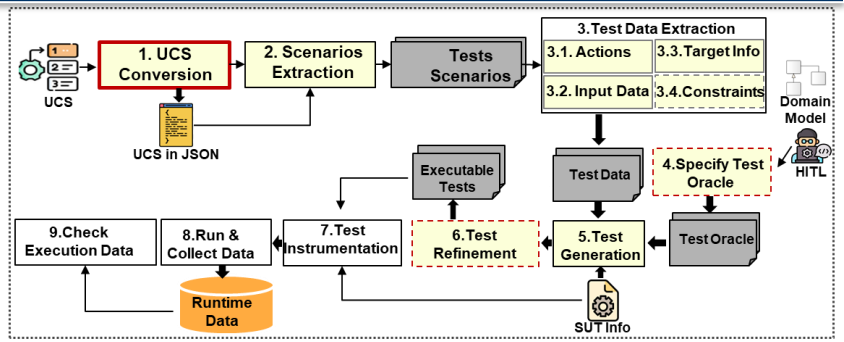
Minani et al.; "FUNEETIS: Approach for Functional End-to-End Testing of IoT Systems"; ACM Transactions on Software Engineering and Methodology (Under revision); 2025

Approach (1/9)



Mismatch between **actual** and **expected** data indicates a **bug**

Approach (2/9)



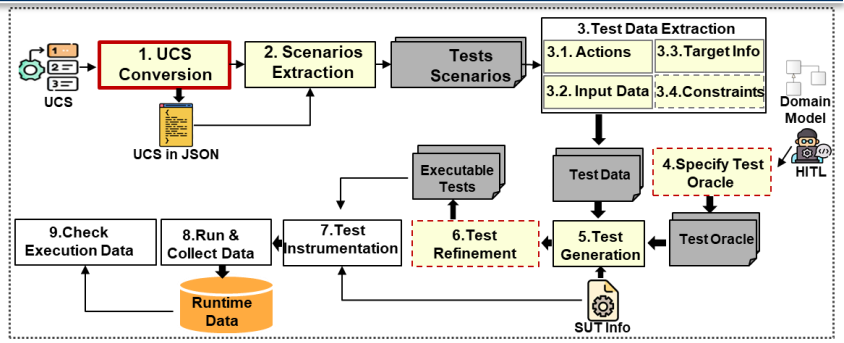
- UCS statements (S) follow **four patterns**
- New keywords used
  - IS, SENDS, RECEIVES, WITH, TO, FROM, and OVER

Patterns for IoT Use Case Specifications

#	Pattern
S1	{NODE1} SENDS {COMMAND/FEEDBACK/DATA} [WITH {INPUTS}] TO {NODE2} OVER {PROTOCOL}
S2	{NODE1} RECEIVES {COMMAND/FEEDBACK/DATA} [WITH {INPUTS}] FROM {NODE2} OVER {PROTOCOL}
S3	{NODE} {ACTION}{ATTRIBUTE} [WITH {INPUTS}][IS {CONSTRAINT}]
S4	{ATTRIBUTE1} IS {VALUE1},[{ATTRIBUTE2} IS {VALUE2}, …]

S1: SENDING step; **S2**: RECEIVING step; **S3**:PROCESSING step; **S4**:POSTCONDITION.

Approach (3/9)



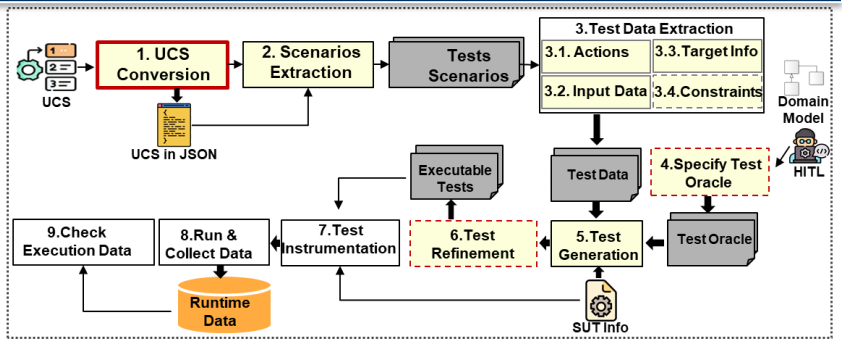
Use Case: Fitbit Sends Data and Buddy Moves
1.1 Precondition
EnableWheels is 0
Mood is 1
1.2 Basic Flow
1. Fitbit **SENDS** heart data to SmartPhone with heartrate=80 bpm Over HTTP
2. SmartPhone **RECEIVES** the heartrate data From Fitbit
3. SmartPhone **SENDS** heart data with heartrate=80 bpm to CloudApp over WebSocket
4. CloudApp **RECEIVES** heartrate data FROM fitbit Over HTTP
5. CloudApp **CHECKS THAT** heartrate is between 60 and 100 bpm
6. CloudApp **SENDS** MOVE command to Buddy with distance=1 and speed=1 Over WebSocket
7. Buddy **RECEIVES** MOVE command from CloudApp
8. Buddy **SETS** EnableWheels with leftWheel=1 and rightWheel=1
9. Buddy **MOVES** at specified distance and speed.
10. Buddy **SENDS** MOVE status to CloudApp with execution_status=WHEEL_MOVE_FINISHED Over WebSocket
11. CloudApp **RECEIVES** MOVE execution status from Buddy Over WebSocket

Postcondition
Execution_status is WHEEL_MOVE_FINISHED
1.3 Bounded Alternative Flow
RFS 1-11
1. **IF** CloudApp **RECEIVES** no heartrate data FROM SmartPhone **THEN**
2. CloudApp **SENDS** ROTATE command To Buddy with angle=50 and speed=30 Over WebSocket
3. Buddy **SETS** EnableWheels with leftWheel=1 and rightWheel=1
4. Buddy **ROTATES** at specified angle and speed.
5. Buddy **SENDS** ROTATE status to CloudApp with execution_status=WHEEL_MOVE_FINISHED Over WebSocket
6. CloudApp **RECEIVES** ROTATE execution status from Buddy Over WebSocket
7. **ENDIF**

Postcondition
Execution_status is WHEEL_MOVE_FINISHED

Excerpt from UCS-1

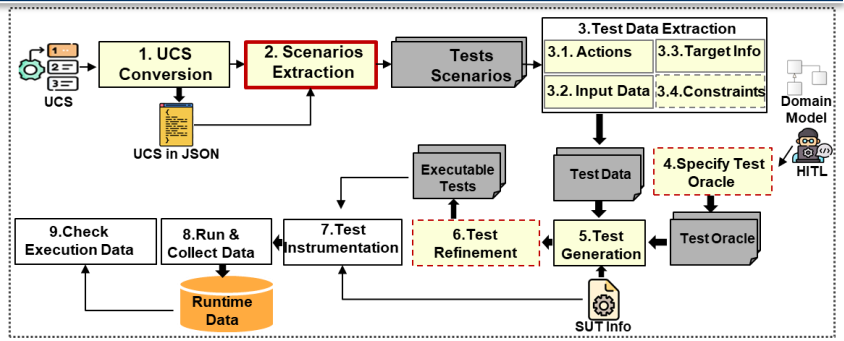
Approach (4/9)




```
{
  "preconditions": [
    { "EnableWheels": "0" },
    { "Mood": "1" }
  ],
  "name": "fitbit sends data and buddy moves",
  "basic_Flow": {
    "steps": [
      { "1": "Fitbit SENDS heart data to SmartPhone with heartrate=80 bpm Over HTTP" },
      { "2": "SmartPhone RECEIVES the heartrate data From Fitbit" },
      { "3": "SmartPhone SENDS heart data with heartrate=80 bpm to CloudApp over WebSocket" },
      { "4": "CloudApp RECEIVES heartrate data FROM fitbit Over HTTP" },
      { "5": "CloudApp CHECKS THAT heartrate is between 60 and 100 bpm" },
      { "6": "CloudApp SENDS MOVE command to Buddy with distance=1 and speed=1 Over WebSocket" },
      { "7": "Buddy RECEIVES MOVE command from CloudApp" },
      { "8": "Buddy SETS EnableWheels with leftWheel=1 and rightWheel=1" },
      { "9": "Buddy MOVES at specified distance and speed" },
      { "10": "Buddy SENDS MOVE status to CloudApp with execution_status=WHEEL_MOVE_FINISHED Over WebSocket"},
      { "11": "CloudApp RECEIVES MOVE execution status from Buddy Over WebSocket" }
    ],
    "postconditions": [ { "Execution_status": "WHEEL_MOVE_FINISHED" } ]
  }
}
```

Excerpt of Converted UCS-1

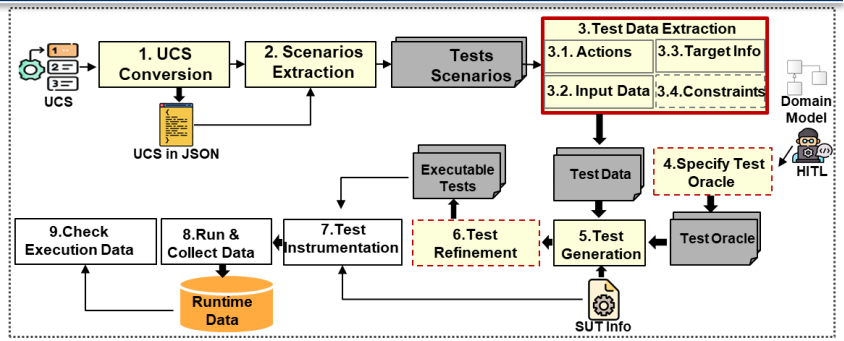
Approach (5/9)




```
{
    "1": {
        "postconditions": [{"Execution_status": "MOVE_FINISHED"}],
        "steps": [
            {"1": "Fitbit SENDS heart data to SmartPhone with heartrate=80 bpm Over HTTP"},
            {"2": "SmartPhone RECEIVES the heartrate data From Fitbit"},
            {"3": "SmartPhone SENDS heart data with heartrate=80 bpm to CloudApp over WebSocket"},
            {"4": "CloudApp RECEIVES heartrate data FROM fitbit Over HTTP"},
            {"5": "CloudApp CHECKS THAT heartrate is between 60 and 100 bpm"},
            {"6": "CloudApp SENDS MOVE command to Buddy with distance=1 and speed=1 Over WebSocket"},
            {"7": "Buddy RECEIVES MOVE command from CloudApp"},
            {"8": "Buddy SETS EnableWheels with leftWheel=1 and rightWheel=1"},
            {"9": "Buddy MOVES at specified distance and speed"},
            {"10": "Buddy SENDS MOVE status to CloudApp with execution_status=MOVE_FINISHED Over WebSocket"},
            {"11": "CloudApp RECEIVES MOVE execution status from Buddy Over WebSocket"}
        ]
    },
    "2": {...},
    "3": {...}
}
```


Part of extracted scenarios from UCS-1

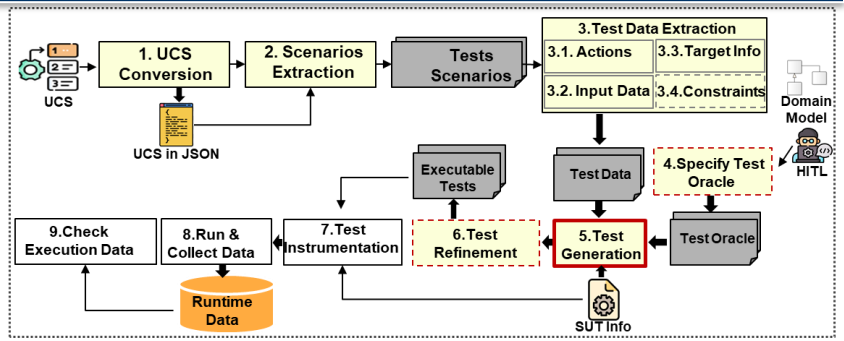
Approach (6/9)




```
"steps": [
  {
    "entity": "fitbit",
    "operation": "sendHeartData",
    "inputs": {"heartrate": "80 bpm"},
    "target": {"protocol": "http","name": "smartphone"},
    "expectations": {"execution_status": "OK"}
  },
  {
    "entity": "smartphone",
    "operation": "sendHeartData",
    "inputs": {"heartrate": "80 bpm"},
    "target": { "protocol": "websocket","name": "cloudapp"},
    "expectations": {"execution_status": "OK" }
  },
  {
    "entity": "cloudApp",
    "operation": "sendMoveCommand",
    "inputs": {"distance": "1","speed": "1"},
    "target": {"protocol": "websocket","name": "buddy"},
    "expectations": {"execution_status": "OK" }
  },
  {
    "entity": "buddy",
    "operation": "sendMoveStatus",
    "inputs": { "execution_status": "wheel_move_finished" },
    "target": {"protocol": "websocket","name": "cloudapp"},
    "expectations": {"execution_status": "WHEEL_MOVE_FINISHED"}
  }
]
```

- Statements covered
  - Fitbit sends data to phone (HTTP)
  - Phone sends data to cloud (WebSocket)
  - CloudApp sends move command to Buddy (WebSocket)
  - Buddy sends execution status to the CloudApp (WebSocket)

Approach (7/9)



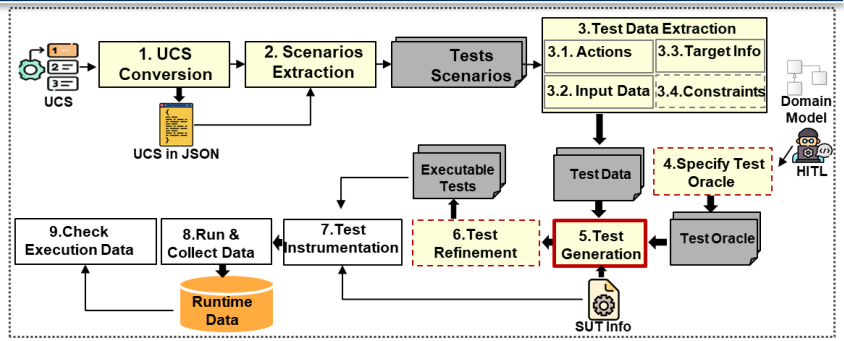
Require: testData file is provided and valid ▷ Payload
Require: sutDescription file is provided ▷ SUT description
Ensure: $T \leftarrow$ Test Suite ▷ A file containing test cases

```
Let S_sut ← Entities from sutDescription
Let S_td ← ∅                                   ▷ initialized entities (i.e., Node)
testData ← Read test data from the file
testClassContent ← Initialize content with required imports and class definition
Begin setup method in testClassContent
for each td ∈ testData do
    for each step ∈ td.steps do
        entity ← step.entity
        if entity ∈ S_sut then                 ▷ Only process if entity is in SUT
            if entity ∉ S_td then
                S_td ← S_td ∪ {entity}         ▷ Add entity to set
                testClassContent ← testClassContent ∪ {initialization for entity}
            end if
        end if
    end for
end for
End setup method in testClassContent
for each td ∈ testData do
    Append new test method for td to testClassContent
    for each step ∈ td.steps do
        Extract operation, inputs, and expectations from step
        Generate assertion code for operation using inputs and expectations
        testClassContent ← testClassContent ∪ {assertion code}
    end for
    End test method in testClassContent
end for
T ← testClassContent
Write T to a file                              ▷ Save T to the file
```

Test Generation Algorithm

- Parts of the algorithm
  - Initialize the **test class**
  - Identify and add **each method** to test class
  - Generate **test content** for **each method**

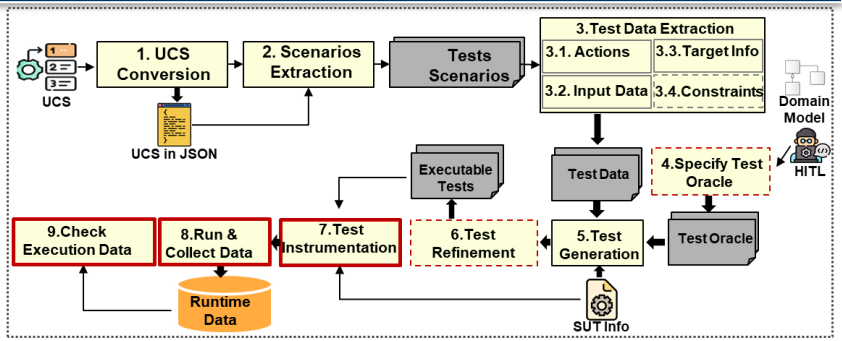
Approach (8/9)



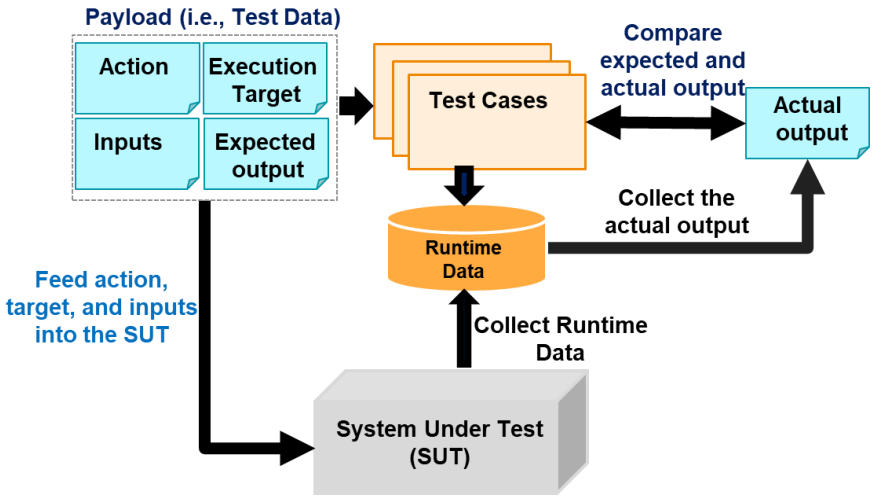

```
@Before
public void setUp() {
    fitbitFetcher = new fitbitDataFetcher("TBP");
    gatewayFetcher = new gatewayDataFetcher("TBP");
    cloudFetcher = new cloudDataFetcher("TBP");
    buddyFetcher = new buddyDataFetcher("TBP");
}

@Test
public void testCase1() {
    assertEquals(80, fitbitFetcher.getdHeartData(2));
    assertEquals(80, gatewayFetcher.getHeartData(2));
    assertEquals(80, cloudFetcher.getHeartData(2));
    assertEquals(1.0, buddyFetcher.getMoveDistance(2));
    assertEquals("MOVE_FINISHED", buddyFetcher.getExecutionStatus(2));
}
```

Approach (9/9)



Execution Flow



Collects runtime data and store them in **centralized database**

Instrumentation Excerpt

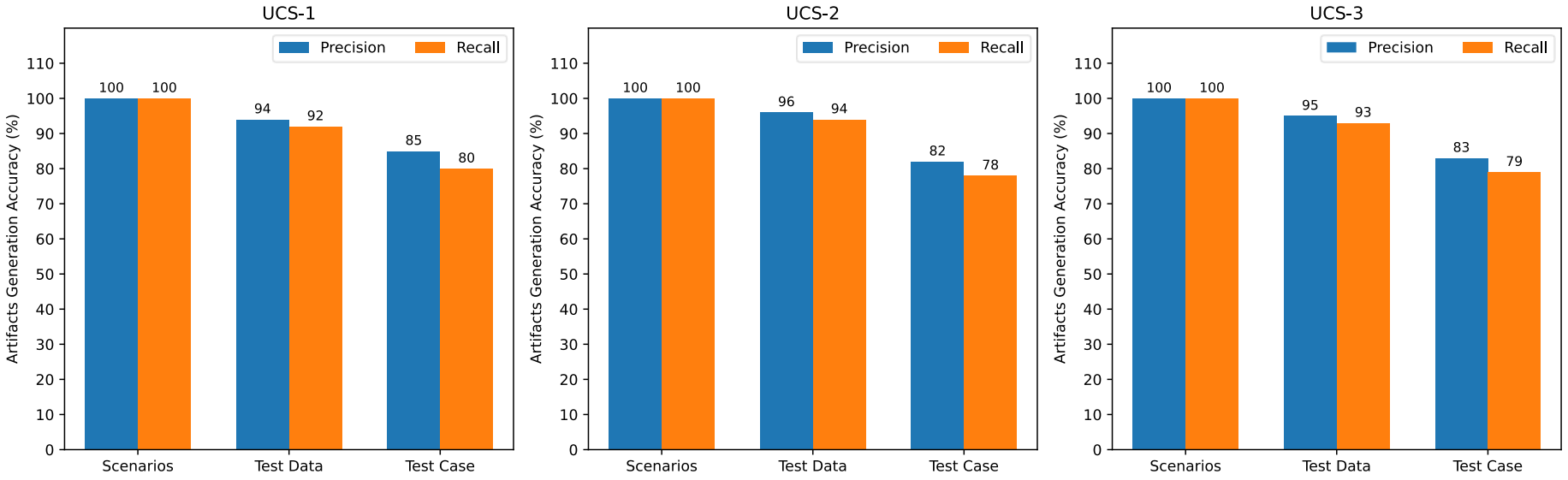
```
public class FirebaseDataSender {
 private DatabaseReference mDatabase;
 public FirebaseDataSender() {
     // Set up the database reference with the specific URL
     mDatabase = FirebaseDatabase.getInstance("URL").getReference();
 }
 public void sendDataToFirebase(int test_ID, int hr, double d,double s,
 String action, String ts, String eR, String aR) {
     Map<String, Object> dataMap = new HashMap<>();
     dataMap.put("timestamp", ts);
     dataMap.put("operation", action);
     dataMap.put("heartRate", hr);
     dataMap.put("actualResults", aR);
     dataMap.put("expectations", eR);
     dataMap.put("taskID", test_ID);

     Map<String, Object> inputs = new HashMap<>();
     inputs.put("distance", d);
     inputs.put("speed", s);
     dataMap.put("inputs", inputs);
     // Send the data to Firebase under the node 'buddy.json'
     mDatabase.child("buddy.json").push().setValue(dataMap);
 }
}
```

Captures the data and **sends them** to the **central database**

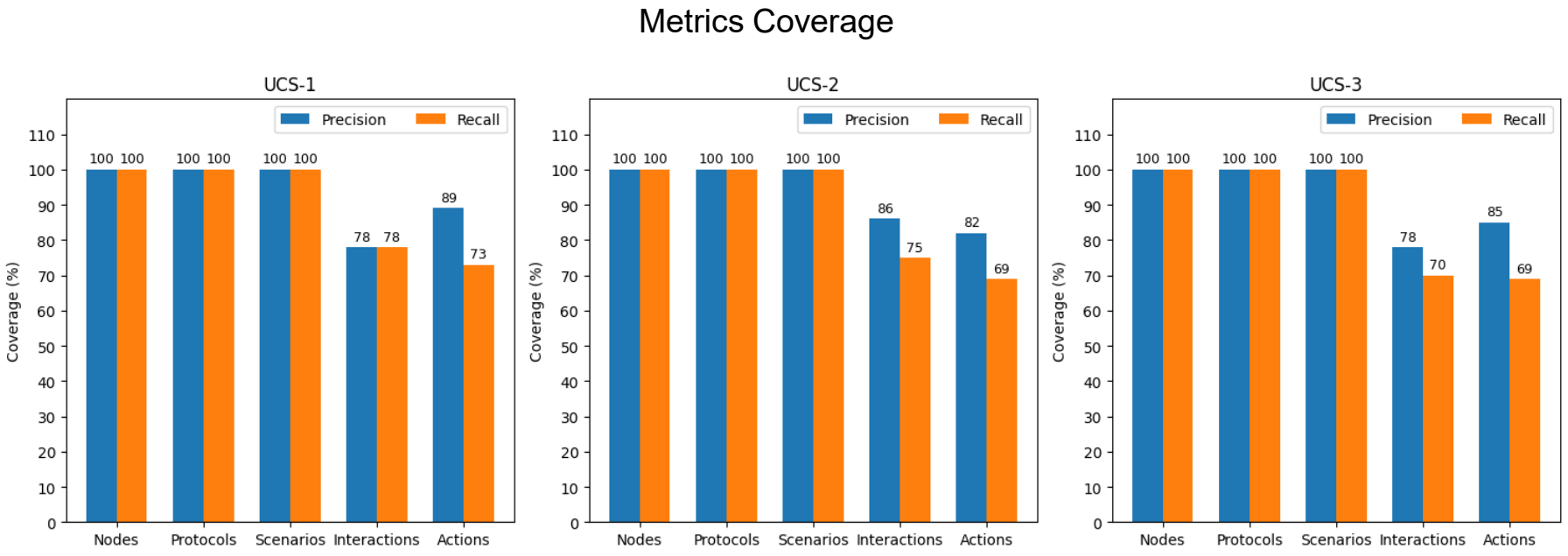
FUNEETIS – Results

Results Analysis



Good accuracy in **scenario** and **test data** extraction

FUNEETIS – Results



High coverage overall, but low for **interactions** and **actions**

FUNEETIS – Results

FUNEETIS vs. **TGenAI**

	Time (secs)		
Approach	**UCS-1**	**UCS-2**	**UCS-3**
Manual	3922	2375	1580
TGENAI	55	69	68
FUNEETIS	14	15	12

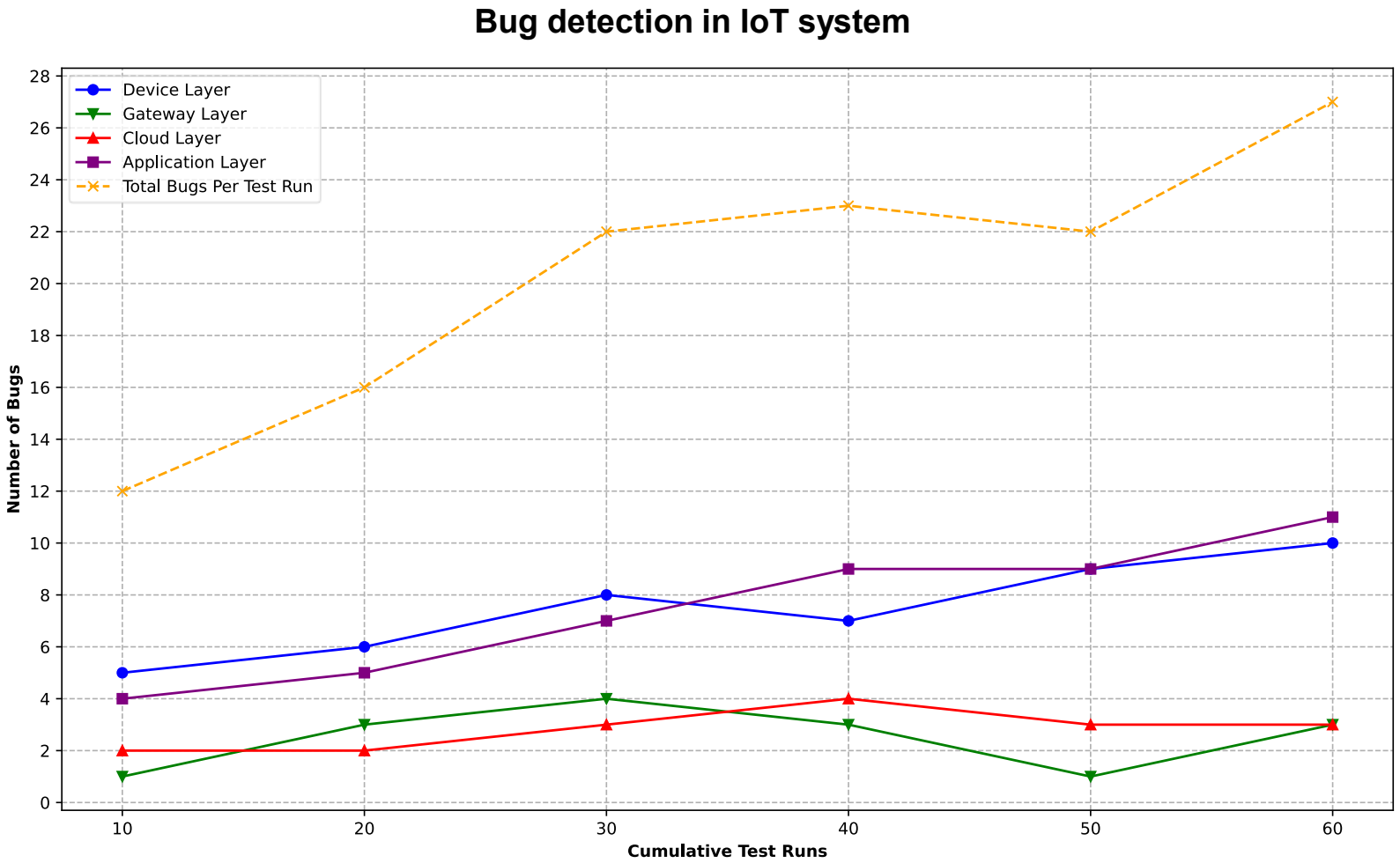
Time Required for Test Data Generation

	Test Scenario Coverage		
Approach	**UCS-1**	**UCS-2**	**UCS-3**
Manual	6	7	5
TGENAI	6	7	6
FUNEETIS	6	7	6

Number of Test Scenario Generated

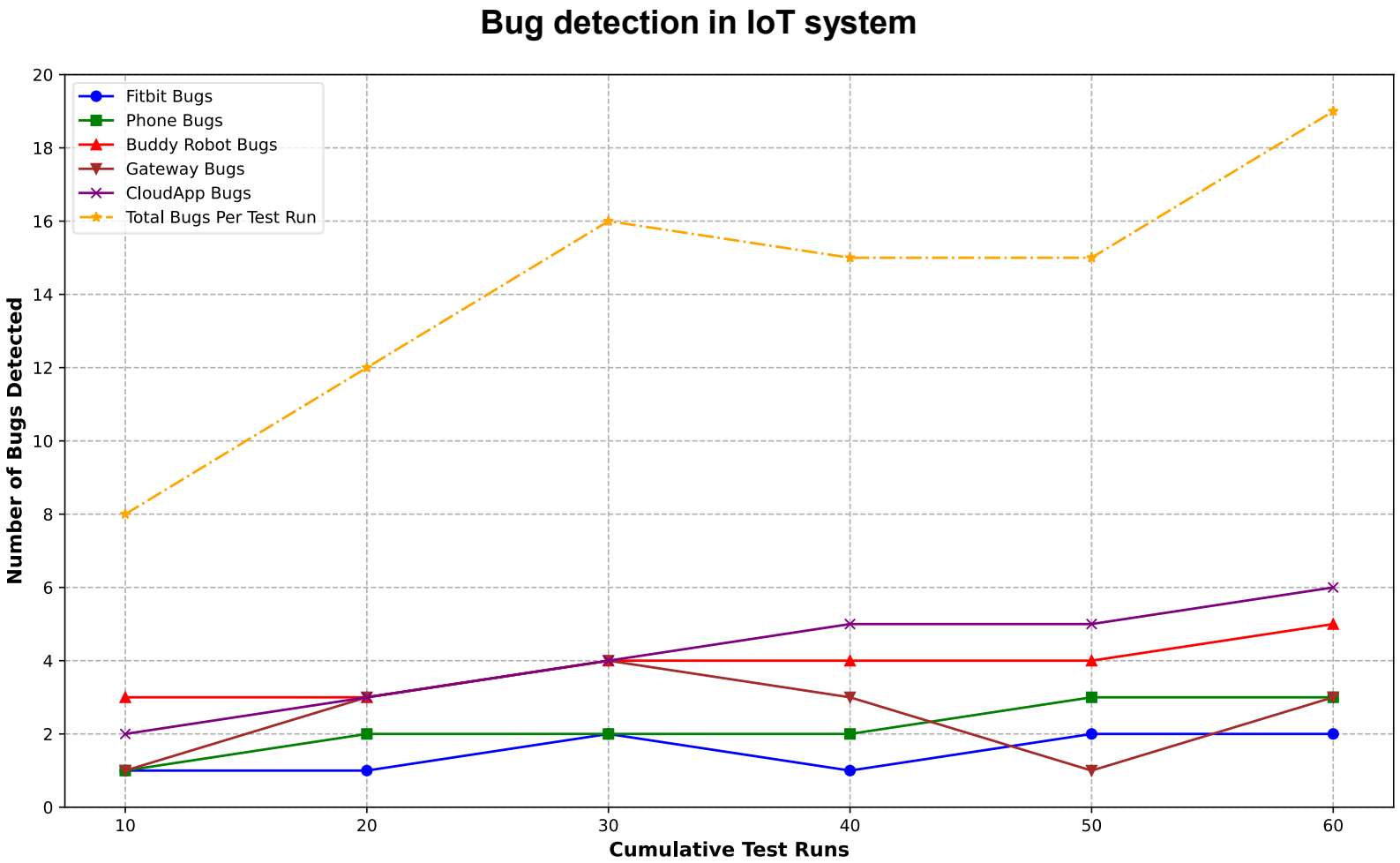
FUNEETIS significantly **reduces test data generation time** compared to TGenAI

FUNEETIS – Results



More bugs in Device and Application; **fewer** in Gateway and Cloud

FUNEETIS – Results



Devices with **more functionalities** tend to reveal **more bugs**

FUNEETIS – Threats to Validity

- Evaluation is based on **a single** IoT system with few scenarios
  - Limits generalizability due to **lack of open-source** IoT systems
- FUNEETIS requires **well-structured UCSs**
- **HITL Dependence** for manual intervention
  - Minimizing HITL is a goal for future improvements

FUNEETIS – Conclusion

- FUNEETIS **derives tests** from UCSs using **specific keywords**
- **Reduces** test generation **time** and
  - Achieves **good coverage** for scenarios, nodes, and protocols, but
  - Has limited coverage for interactions and actions
- Allows **bug detection** across **IoT layers**
- Full automation is **limited by HITL** reliance

Conclusion

- Conducted **SLR** and **industry study**
- Highlighted several challenges
  - C1 – Lack of testing guide
  - C2 – Manual/semi-automated testing
  - C3 – Limited focus on functional testing
  - C4 – Lack of approaches and tools
- Addressed
  - Taxonomy & Framework (**C1**)
  - Functional E2E testing approaches **(C2-C4)**

Presented the **first approach** for **functional E2E test generation** and **execution** in IoT systems, enabling bug **detection across all layers**

It is possible to automate **functional E2E test generation** and systematically **execute the generated tests** to **detect bugs** in IoT systems

Future Work

- **Short-Term**
  - Enhance FUNEETIS to **minimize HITL** effort
  - Make FUNEETIS an **industry-ready** solution
- **Mid-Term**
  - Testing of non-functional aspects in IoT systems
  - **Open-source IoT systems** for tools and approaches evaluation

Future Work

- **Long-Term**
  - **Digital twins** to simulate and validate IoT system behavior
  - **Testable specifications** through **requirements engineering** (RE) **in IoT systems**

PhD Thesis Defense

Functional End-to-End Testing of IoT Systems

Presented by

Jean Baptiste Minani

(ID:40166177)

Supervised by

Dr. Yann-Gaël Guéhéneuc, Dr. Naouel Moha, and Dr. Fatima Sabir

Montréal, Québec, Canada

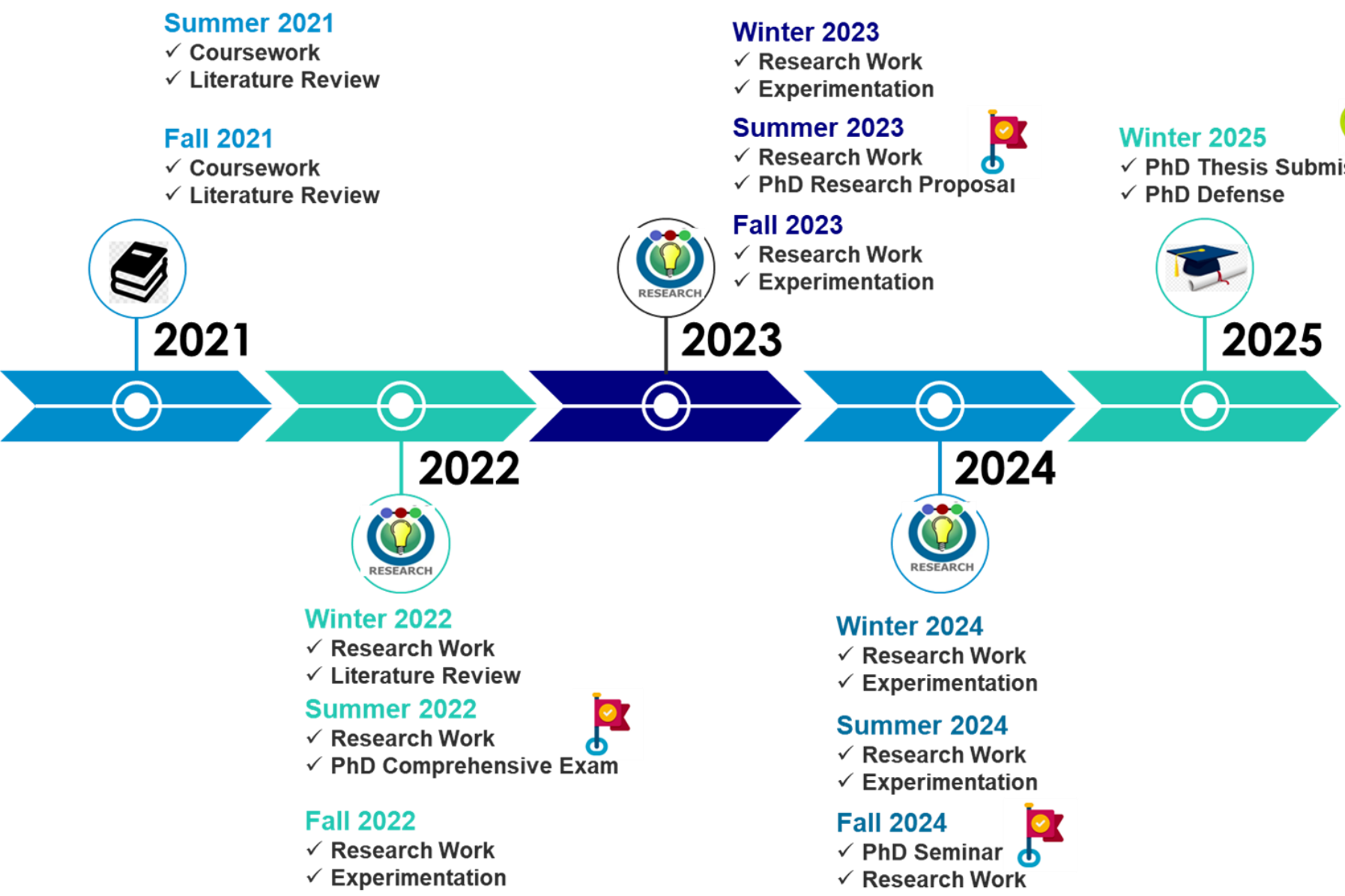
16 May 2025

Backup Slides

Publications

- [1] **Minani** et al.; "A Multimethod Study of Internet of Things Systems Testing in Industry"; **IEEE Internet of Things Journal**; 2023
- [2] **Minani** et al.; "A Systematic Review of IoT Systems Testing: Objectives, Approaches, Tools, and Challenges"; **IEEE Transactions on Software Engineering**; 2024
- [3] **Minani** et al.; "Practical Guidance for IoT Systems Testing: A Taxonomy"; **SERP4IoT '24: Proceedings of the ACM/IEEE 6th International Workshop on Software Engineering Research & Practices for the Internet of Things**; 2024
- [4] **Minani** et al.; "Towards an Automated Approach for Testing IoT Devices"; SERP4IoT '24: **Proceedings of the ACM/IEEE 6th International Workshop on Software Engineering Research & Practices for the Internet of Things**; 2024
- [5] **Minani** et al.; "An Exploratory Study on Code Quality, Testing, Data Accuracy, and Practical Use Cases of IoT Wearables"; **2024 7th Conference on Cloud and Internet of Things (CIoT)**; 2024
- [6] Khezemi N., **Minani J.B et al.;** "A Systematic Literature Review of IoT System Architectural Styles and Their Quality Requirements"; **IEEE Internet of Things Journal**; 2024
- [7] Zacharie C., **Minani JB** et al.; "Test Generation from Use Case Specifications for IoT Systems: Custom, LLM-Based, and Hybrid Approaches"; **IEEE International Conference on Software Testing, Verification and Validation (ICST);** 2025
- [8] **Minani** et al.; "IoT systems testing: Taxonomy, empirical findings, and recommendations**"; Journal of Systems and Software**; 2025
- [9] **Minani**; "TISSEA: A Framework for Testing IoT Systems Based on Technical Software Engineering Aspects"; **IEEE Internet of Thing Journal** (Under review); 2025
- [10] **Minani** et al.; "TGenAI: LLM-based Approach for Functional Test Cases Generation for IoT System"; **Information and Software Technology**; 2025
- [11] **Minani** et al.; "FUNEETIS: Approach for Functional End-to-End Testing of IoT Systems"; **ACM Transactions on Software Engineering and Methodology** (Under revision); 2025
- [12] Trabelsi I., Mahmoudi B., **Minani J.B**. et al; "A Systematic Literature Review of Machine Learning Approaches for Migrating Monolithic Systems to Microservices"; **IEEE Transactions on Software Engineering**; Submitted revision; 2025
- [13] Mahmood K., **Minani J.B.**, et al.; "Exploring the Impacts of Antipatterns on Object-Oriented, Service-Oriented, and Mobile-Oriented Systems"; **Software: Practice and Experience**; Submitted revision; 2025
- [14] El Fellah Y., **Minani J.B.**, et al.; "Analysis of Microservices-Based IoT Systems: Deployment Challenges, Industry Practices, and Performance Insights"; Under review, **IEEE Internet of Things Journal**; 2025

Timeline


Summer 2021
✓ Coursework
✓ Literature Review
Fall 2021
✓ Coursework
✓ Literature Review
2021
RESEARCH
2022
Winter 2022
✓ Research Work
✓ Literature Review
Summer 2022
✓ Research Work
✓ PhD Comprehensive Exam
Fall 2022
✓ Research Work
✓ Experimentation
RESEARCH
2023
Winter 2023
✓ Research Work
✓ Experimentation
Summer 2023
✓ Research Work
✓ PhD Research Proposal
Fall 2023
✓ Research Work
✓ Experimentation
RESEARCH
2024
Winter 2024
✓ Research Work
✓ Experimentation
Summer 2024
✓ Research Work
✓ Experimentation
Fall 2024
✓ PhD Seminar
✓ Research Work
2025
Winter 2025
✓ PhD Thesis Submission
✓ PhD Defense

Acknowledgments

- Special thanks to
  - **Supervisors** for your invaluable guidance and support
  - **Collaborators** for sharing your insights
  - **Family** & **friends** for your endless love and support
  - **Ptidej team** for your feedback

Outline

- Context
- State-of-the-Art
- Relevant Studies Overview
- Taxonomy & Framework
- Generating E2E Tests
- Conclusion & Future Work

Bugs in IoT Systems

- A bug or fault in a system is a defect that can lead to errors during execution, which, in turn, may result in system failures [3].
- Recent studies indicate that application layer accounts for many bugs in IoT systems. However, some bugs can be found at the network and device layers [4].

[3] Melo et al. ; The pathology of failures in IoT systems ; ICCSA 2021
[4] Anandayuvaraj et al. ; Reflecting on recurring failures in IoT development ; IEEE/ACM International Conference on Automated Software Engineering, 2022

Thesis Focus

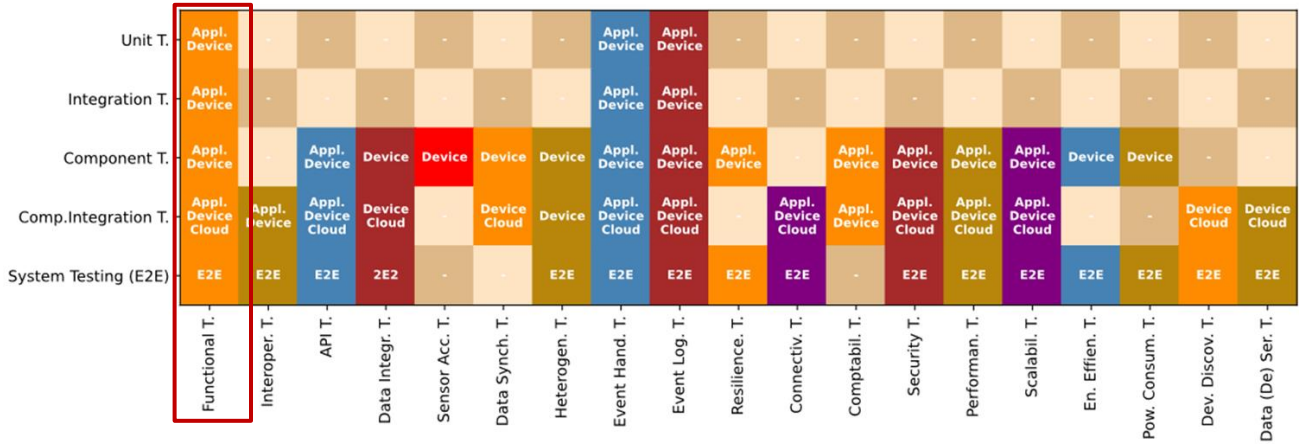
- This thesis specifically focuses on end-to-end (E2E) functional testing of IoT systems.
- E2E functional testing ensures that data flows correctly from devices to gateways, cloud, and applications, and that the expected outcome is achieved.

Aspects	
Compliance Testing	
Functional Testing	
Accessibility Testing	
Compatibility Testing	Interoperability Testing
	Co-Existence Testing
Performance Testing	Load Testing
	Stress Testing
	Volume Testing
	Endurance Testing
	Spike Testing
	Capacity Testing
	Disaster Recovery Testing
Maintainability Testing	Modularity Testing
	Reusability Testing
	Analysability Testing
	Modifiability Testing
Portability Testing	Installability Testing
	Replaceability Testing
Reliability Testing	Availability Testing
	Fault-Tolerance Testing
	Recoverability Testing
Security Testing	Vulnerability Scanning
	Penetration Testing
	Security Auditing
	Configuration Scanning
	Data Privacy Testing
Usability Testing	Accessibility Testing
	UI Testing

Aspects	
	UI Testing
	Error Protection Testing
	Learnability Testing
	UX Testing
Scalability Testing	
Connectivity Testing	
Distributivity Testing	
Deployability Testing	
Interoperability Testing	
Energy Efficiency Testing	
Sensor Accuracy Testing	
Power Consumption Testing	
Firmware Update Testing	
Event Handling Testing	
Even Logging Testing	
Device provisioning Testing	
Geolocation Accuracy Testing	
Data Synchronization Testing	
Communication Protocol Testing	
OTA Update Testing	
Device Provisioning and onboarding testing	
Mobility Testing	
API Testing	
Resilience Testing	
User Attitude Testing	
Heterogeneity Testing	
Data Integrity Testing	
Authentication & Authorization	
Device Discovery Testing	
Data (De) Serialization Testing	

Framework – Technical SE

- **19 technical SE aspects** (e.g., functional testing, API Testing, etc.)
- **5 test** granularities
  - Unit, Integration, Component, Component Integration, System



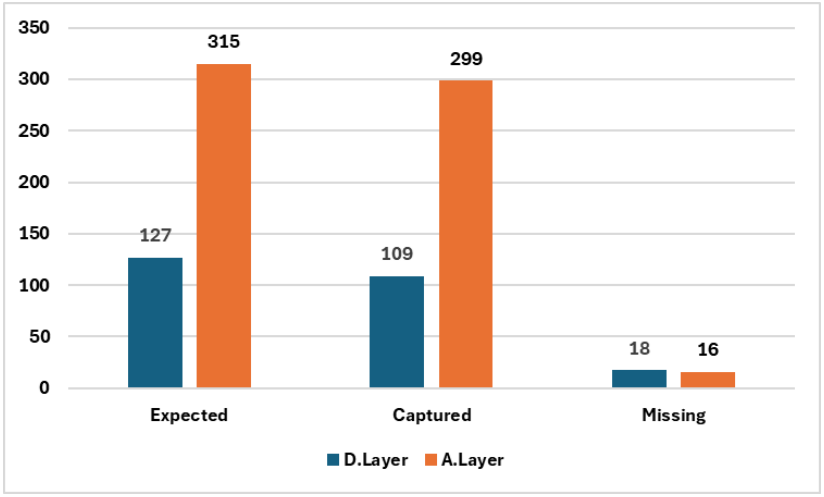
Framework-Validation (Survey)

Framework Evaluation with Survey

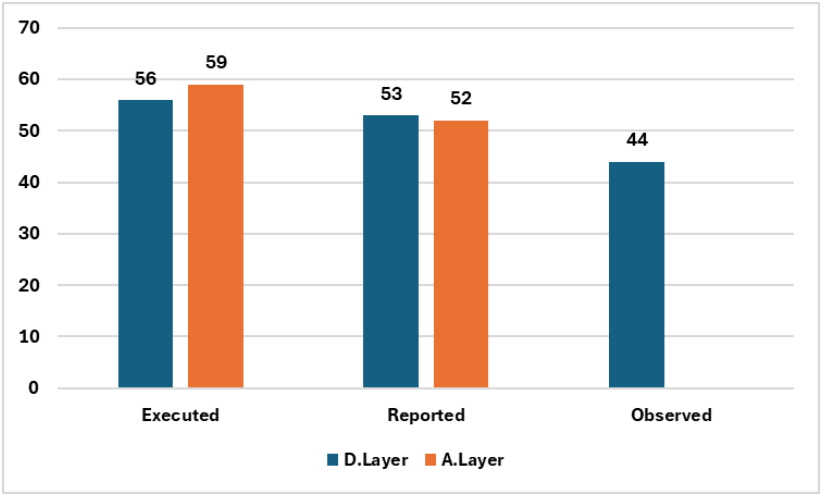
Category	Mean	Std	Var	t-Statistic	p-Value
1.Tech. SE Aspects	4.727	0.456	0.198	-2.806	0.011
2.Test Granularities	4.727	0.456	0.198	-2.806	0.011
3.Tech.SE Asp. for Device layer	4.864	0.351	0.118	-1.821	0.083
4.Tech.SE Asp. for Gateway Layer	4.455	0.739	0.521	-3.464	0.002
5.Tech.SE Asp. for Cloud Layer	4.636	0.581	0.322	-2.935	0.008
6.Tech.SE Asp. for App. Layer	4.864	0.351	0.118	-1.821	0.083
7.Orchestration Strategies	4.818	0.395	0.149	-2.160	0.042
8.Execution Methods	4.955	0.213	0.043	-1.000	0.329
9.Input Artifacts	4.955	0.213	0.043	-1.000	0.329

Framework – Empirical Evaluation

Expected vs. Captured Log Entries



Reported vs Observed Events



Observed **missing logs** at the **device** and **application** layers, with **fewer events** executed

Conclusion

- Developed **IoT testing taxonomy** with input from **200+ practitioners**
- **Published** taxonomy as a **living tool**
- Proposed **TISSEA**, a **framework** to test technical **SE aspects** of IoT systems
- **Validated** TISSEA with **22 practitioners**

Step 1: Exploration - Results

- RQ12: How do test generation approaches compare for E2E testing of IoT Systems?

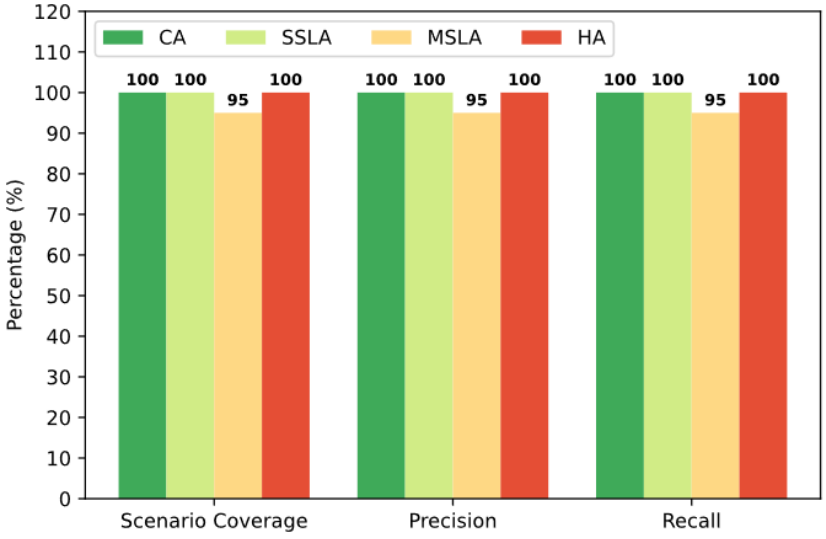
	Test Steps			Accuracy	
Approach	**Coverage**	**Valid**	**Invalid**	**Precision**	**Recall**
CA	100%	100%	0%	100%	100%
SSLA	94%	99%	1%	98%	97%
MSLA	100%	96%	4%	96%	95%
HA	100%	100%	0%	100%	100%

Exploration Insights

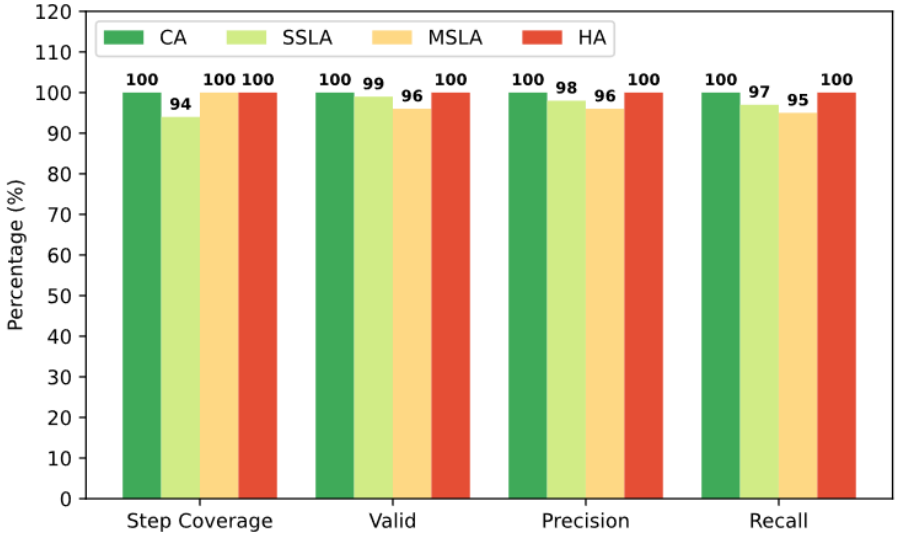
- Exploration was based **on simple UCSs**
- Findings are **promising** but require **validation** on more **complex UCSs**
- Defined two distinct approaches
  - LLM-based (**TGenAI**)
  - Structured non-LLM-based (**FUNEETIS**)

Exploration – Results

Comparing Test Generation Approaches



Scenario Coverage (Accuracy)



Test Correctness (Accuracy)

CA & **LLMs** show promise in test generation

Zacharie C., **Minani JB** et al.; "Test Generation from Use Case Specifications for IoT Systems: Custom, LLM-Based, and Hybrid Approaches"; IEEE International Conference on Software Testing, Verification and Validation (ICST); 2025

TGenAI – Approach

API Key and model

Calling LLM

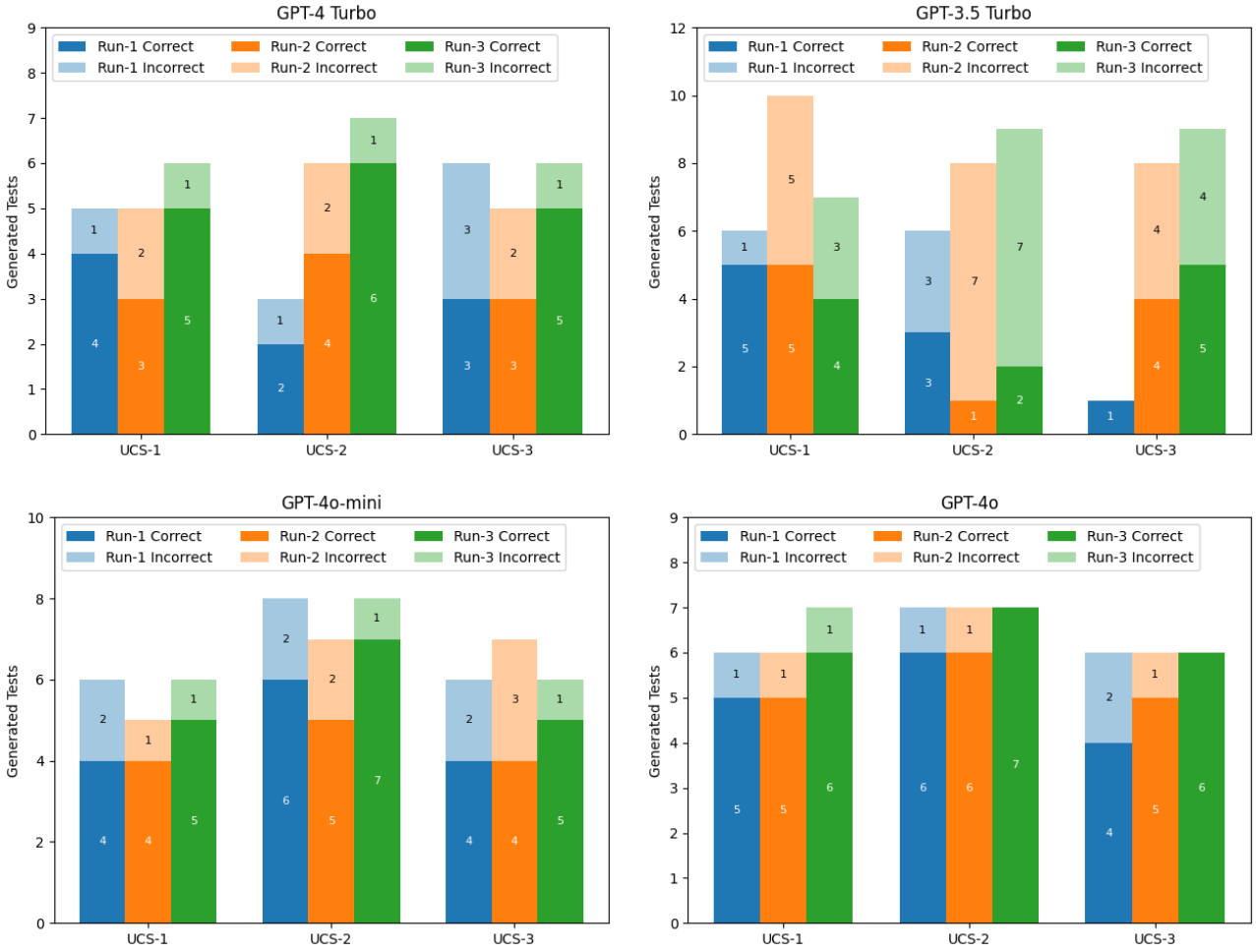
Extracting tests from response

```
public static String callLLM(String prompt) {
    String url = "https://api.openai.com/v1/chat/completions";
    String apiKey = "<VALID-KEY>";
    String model = "gpt-4o";
    try {
        URL obj = new URL(url);
        HttpURLConnection con = (HttpURLConnection) obj.openConnection();
        con.setRequestMethod("POST");
        con.setRequestProperty("Authorization", "Bearer " + apiKey);
        con.setRequestProperty("Content-Type", "application/json");
        String body = "{\"model\":\"" + model + "\", \"messages\":[{\"role\":\"user\",
        ↪  \"content\":\"" + prompt + "\"}]}";
        con.setDoOutput(true);
        OutputStreamWriter writer = new OutputStreamWriter(con.getOutputStream());
        writer.write(body);
        writer.flush();
        writer.close();
        // Check the response code
        int responseCode = con.getResponseCode();
        if (responseCode != HttpURLConnection.HTTP_OK) {
            <<do-something>>
        }
        // Get the response
        BufferedReader in = new BufferedReader(new
        ↪  InputStreamReader(con.getInputStream()));
        String inputLine;
        StringBuffer response = new StringBuffer();
        while ((inputLine = in.readLine()) != null) {
            response.append(inputLine);
        }
        in.close();
}
```

Segment of Code to Invoke LLM using API

TGenAI – Results

Evaluating Correctness of LLMs for Test Generation



GPT-4o is the most reliable among selected models

LLMs require significant **HITL** effort to validate and refine results

```
{
  "nodes": [
    {
      "name": "Fitbit-V2",
      "os": "FitbitOS",
      "app": "HeartRateApp",
      "pl": "nodeJS",
      "network": {},
      "dataFlow": [
        {
          "nodeId": "Glaxy-S10",
          "interactionType": "send",
          "protocol": "WebSocket",
          "data": "heartRate"
        }
      ],
      "codeConstruct": {
        "moduleOrClass": {
          "name": "HeartSensor",
          "methodOrFunction": [
            {
              "name": "getHrData",
              "description": "Retrieves the current heart rate data from the sensor.",
              "inputs": [
                {
                  "paramName": "requestedData",
                  "paramType": "Integer",
                  "description": "Requested Data by indicating 1 for heart rate data, 2 for geolocation data",
                  "testData": 1
                }
              ],
              "errorHandling": {
                "exceptions": [
                  {
```


Part of SUT Description

```
      "errorHandling": {
        "exceptions": [
          {
            "type": "Exception",
            "message": "Sensor is not responding."
          }
        ]
      },
      "returns": {
        "type": "String",
        "description": "Returns heart rate data as a formatted string."
      }
    },
    {
      "name": "sendHrData",
      "description": "Sends heart rate data to a connected smartphone.",
      "inputs": [
        {
          "paramName": "hrRateData",
          "paramType": "Integer",
          "description": "Heart rate data to be sent.",
          "testData": 85
        }
      ],
      "errorHandling": {
        "exceptions": [
          {
            "type": "Exception",
            "message": "Failed to send data due to network issues."
          }
        ]
      },
      "returns": {
        "type": "void",
        "description": "Void means no return value."
      }
    }

  ]
 }
}
},
```


Part of SUT Description