**­­­Tribhuvan University**

**Institute of Science and Technology**

**Central Department of Computer Science and Information Technology**

**Kirtipur, Kathmandu**

**2024**

**Seminar Report on**

**“Improving Test Coverage with Boundary Value Analysis (BVA)”**

**In partial fulfillment of the requirement for master’s degree in computer science and information technology (M.Sc. CSIT), 1st Semester**

**Submitted to:**

Central Department of Computer Science and Information Technology, Tribhuvan University, Kirtipur, Kathmandu, Nepal

**Submitted By:**

Bishnu Maya Dhakal (8015008)

****

**Tribhuvan University**

**Institute of Science and Technology**

**Supervisor Recommendation**

This is to certify that Miss. Bishnu Maya Dhakal has submitted the seminar report on the topic "**Improving Test Coverage with Boundary Value Analysis (BVA)”** for the partial fulfilment of Master of Science in Computer Science and Information Technology, first semester. I hereby, declare that this seminar report has been approved.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Supervisor

Asst. Prof. Mr. Jagdish Bhatta

Central Department of Computer Science and Information Technology

**Certificate of Approval**

This is to certify that the seminar report prepared by Miss. Bishnu Maya Dhakal" **Improving Test Coverage with Boundary Value Analysis (BVA)”** in partial fulfilment of the requirements for the degree of Master of Science in Computer Science and Information Technology has been well studied. In our opinion, it is satisfactory in the scope and quality as a project for the required degree.

Evaluation Committee

…………….………………………….. ………..…………………………………

Asst. Prof. Sarbin Sayami Asst. Prof. Jagdish Bhatta

(H.O.D) (Supervisor)

Central Department of Computer Science Central Department of Computer Science

and Information Technology and Information Technology

……………………………

(External)

# **ACKNOWLEDGMENT**

The success and outcome of this report required a lot of guidance and assistance from many people, and I am very fortunate to have got this all through the completion. I am very glad to express my deepest sense of gratitude and sincere thanks to my highly respected and esteemed supervisor **Assoc. Prof. Jagdish Bhatta,** Central Department of Computer Science and Information Technology for his valuable supervision, guidance, encouragement, and support for completing this paper.

I am also thankful to **Asst. Prof. Sarbin Sayami**, HOD of the Central Department of Computer Science and Information Technology for his constant support throughout the period. Furthermore, with immense pleasure, I sincerely thank the Central Department of Computer Science and Information Technology, Tribhuvan University, and all the faculty members of CDCSIT for providing the platform to explore the knowledge of interest. At the end I would like to express my sincere thanks to all my friends and family who helped me directly or indirectly.

Thank you,

Bishnu Maya Dhakal

(8015008)

# **Abstract**

Boundary Value Analysis (BVA) is recognized as an integral software testing technique that enhances test coverage by focusing on input boundaries, where defects are most likely to occur. In this study, BVA's foundational principles and practical applications are explored, demonstrating its effectiveness in improving software reliability and robustness. As a critical aspect of software testing, BVA ensures that applications function correctly under various conditions. Errors at input extremes, such as buffer overflows and off-by-one mistakes, are targeted through single and multiple-variable boundary testing. A temperature conversion function case study illustrates how boundary condition defects that might be overlooked by conventional testing methods are detected. The example output shows: -40°C is converted to -40°F (valid boundary), -41°C to -41.8°F (invalid boundary), -39°C to -38.2°F (valid boundary), 100°C to 212°F (valid boundary), 99°C to 210.2°F (valid boundary), and 101°C to 213.8°F (invalid boundary). Automated testing of edge cases is implemented using JavaScript, TypeScript, and Selenium WebDriver. It is found that BVA significantly improves test coverage and reliability, uncovering boundary-related defects that conventional methods might miss. This study underscores BVA’s critical role in developing robust software systems, reducing maintenance costs, and enhancing user satisfaction through comprehensive evaluation of boundary conditions.

**Keywords:** *Automated Testing, Boundary Value Analysis (BVA), Software Testing, Test Coverage, Temperature Conversion*

# **Table of Content**

[**ACKNOWLEDGMENT** iii](#_Toc173070628)

[**Abstract** iv](#_Toc173070629)

[**Table of Content** v](#_Toc173070630)

[**List of Figures** vii](#_Toc173070631)

[**List of Tables** viii](#_Toc173070632)

[**List of Listing** ix](#_Toc173070633)

[**List of Abbreviations** x](#_Toc173070634)

[**Chapter 1: Introduction** 1](#_Toc173070635)

[**1.1** **Introduction** 1](#_Toc173070636)

[**1.2** **Problem Statement** 2](#_Toc173070637)

[**1.3** **Objective** 2](#_Toc173070638)

[**Chapter 2: Background Study and Literature Review** 3](#_Toc173070639)

[**2.1 Background Study** 3](#_Toc173070640)

[**2.1.1 Boundary Value Analysis (BVA)** 3](#_Toc173070641)

[**2.2 Literature Review** 3](#_Toc173070642)

[**Chapter 3: Methodology** 5](#_Toc173070643)

[**3.1 Conceptual Framework** 5](#_Toc173070644)

[**3.2 Boundary Value Analysis Algorithm** 6](#_Toc173070645)

[**3.3 Illustration** 6](#_Toc173070646)

[**Chapter 4: Implementation and Result Analysis** 8](#_Toc173070647)

[**4.1 Implementation** 8](#_Toc173070648)

[**4.1.1 Implementation Tools** 8](#_Toc173070649)

[**4.2.2 Implementation Details** 8](#_Toc173070650)

[**4.2 Finding** 10](#_Toc173070651)

[**4.2.1 Explanation** 10](#_Toc173070652)

[**4.3 Result Analysis** 12](#_Toc173070653)

[**Chapter 5: Conclusion and Future Recommendations** 13](#_Toc173070654)

[**5.1 Conclusion** 13](#_Toc173070655)

[**5.2 Future Recommendations** 13](#_Toc173070656)

[**References** 14](#_Toc173070657)

# **List of Figures**

[**Figure 1: Boundary value** 2](#_Toc172916332)

[**Figure 2 Boundary values** 5](#_Toc172916333)

[**Figure 3: Flowchart of Boundary Value Analysis** 5](#_Toc172916334)

# **List of Tables**

[**Table 1: All Test Cases** 7](#_Toc173070568)

[**Table 2: Output of Test Cases** 12](#_Toc173070569)

# **List of Listing**

[**Listing 1: Function definition** 8](#_Toc173070557)

[**Listing 2:All Test Cases** 9](#_Toc173070558)

[**Listing 3 :Function Definition** 9](#_Toc173070559)

[**Listing 4: Test Lower Boundary (Valid)** 9](#_Toc173070560)

[**Listing 5: Test Lower Boundary (Invalid)** 9](#_Toc173070561)

[**Listing 6: Test Upper Boundary (Valid)** 10](#_Toc173070562)

[**Listing 7: Test Upper Boundary (Invalid)** 10](#_Toc173070563)

[**Listing 8: Test Nominal Case (Within Range)** 10](#_Toc173070564)

[**Listing 9 :Output of Lower Boundary** 11](#_Toc173070565)

[**Listing 10: Output of Upper Boundary** 11](#_Toc173070566)

[**Listing 11 :Output for Nominal case** 11](#_Toc173070567)

# **List of Abbreviations**

**BVA**  Boundary Value Analysis

**BCD**  Boundary Condition Definition  
**EQP** Equivalence Partitioning   
**ST**  Software Testing   
**TC** Test Case

# **Chapter 1: Introduction**

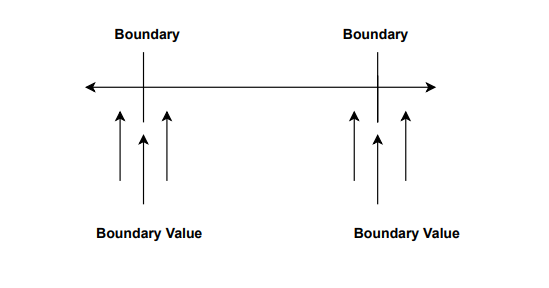
## **Introduction**

Software testing is a process in the software development lifecycle, aimed at identifying and resolving defects to ensure the quality and reliability of software applications. This process involves executing a program or system with the intent of finding errors, verifying that the software meets specified requirements, and ensuring it performs as expected in various scenarios. Software testing helps identify problems early, reduces the likelihood of production failures, and increases user happiness by carefully evaluating different program elements, such as usability, performance, security, and functionality. Techniques like Boundary Value Analysis (BVA), which place a strong emphasis on edge circumstances and rigorously process input values at the boundaries of their permissible ranges, are vital to this process. [1]

Boundary Value Analysis (BVA) is recognized as a prominent and systematic technique used in software testing to enhance test coverage and identify defects that occur at the edges of input domains. This approach centers on the selection of test cases at the boundaries between equivalency divisions, where input data handling quirks and limits increase the likelihood of errors occurring. [2] By focusing on the boundary values, problems that might be overlooked by other testing techniques are identified by BVA, offering a more comprehensive and reliable assessment of the software's behavior in edge situations.

The main objective of BVA is to evaluate input range boundaries to ensure the software responds appropriately and consistently to these extreme scenarios. Examples of these boundary conditions include the lowest and highest values within an input range, the values slightly inside or outside of these boundaries, and the precise midpoint value. This method is especially effective for locating buffer overflows, off-by-one mistakes, and other boundary-related problems that might cause serious consequences if ignored. [2]

In this study, the fundamentals and practical applications of boundary value analysis are examined as a method to enhance test coverage. The significance of BVA in guaranteeing the dependability and efficiency of software is explored, and real-world instances are provided to demonstrate the approach. The benefits and drawbacks of BVA are also examined, contrasting it with alternative methods of test case design and emphasizing how it may be incorporated into a comprehensive testing approach to improve the general robustness and quality of software systems. [3] Through this analysis, the importance of boundary value analysis in achieving high test coverage and finding bugs early in the software development lifecycle is demonstrated.



**Figure 1: Boundary value**

## **Problem Statement**

When complete test coverage is lacking in software development, major issues including undetected flaws, unstable program performance, higher maintenance costs, and a deteriorated user experience arise. Conventional testing techniques frequently fall short in providing sufficient coverage for important edge cases, which leads to defects being overlooked that may result in software malfunctions. Particularly vulnerable to errors if not thoroughly tested are boundary conditions, where software frequently encounters extreme values or transitions. A methodical technique for locating and evaluating these crucial edge situations is provided by boundary value analysis (BVA), which guarantees in-depth investigation and increased test coverage. By using BVA to address these problems, software quality may be greatly increased, defect risk can be decreased, and overall dependability can be improved.

## **Objective**

The main objective includes:

* To investigate the effectiveness of Boundary Value Analysis (BVA) for improving test coverage.
* To enhance the reliability and performance of software by ensuring thorough testing

of boundary conditions.

# **Chapter 2: Background Study and Literature Review**

## **2.1 Background Study**

### **2.1.1 Boundary Value Analysis (BVA)**

A key tool in software testing is boundary value analysis (BVA), which works especially well to increase test coverage. It is predicated on the idea that mistakes frequently happen outside of input domains rather than at their center. BVA looks at input variable boundary values to find edge scenarios that are most likely to show flaws. [3]

BVA can be categorized into two main types:

**1. Single Variable Boundary Testing:** Single Variable Boundary Testing focuses on the boundary values of individual input variables. It involves selecting test cases at the minimum, just above the minimum, nominal, just below the maximum, and maximum values of each variable. This approach ensures that each variable is tested at its critical boundaries, identifying potential issues that may arise from extreme input values.

**2. Multiple Variable Boundary Testing:** Multiple Variable Boundary Testing extends the concept to scenarios involving multiple input variables. It tests combinations of boundary values from different variables, considering the interactions between them. This comprehensive approach helps in uncovering defects that may not be apparent when variables are tested in isolation. [4]By examining the boundaries of multiple variables simultaneously, this method provides a more robust assessment of the software's behavior under extreme conditions.

## **2.2 Literature Review**

The foundational work in software testing approaches is where Boundary Value Analysis (BVA) got its start. One of the first to present BVA as an expansion of equivalency partitioning underlined that as flaws frequently arise at the limits of input ranges, boundary testing is a crucial procedure for identifying mistakes that other testing techniques could overlook.

Further studies have concentrated on improving the foundational ideas of BVA. Improvements like robustness testing and multiple boundary value analysis were incorporated to find off-by-one mistakes and boundary condition violations by considering both legitimate boundary values and values that were slightly outside of these bounds. [1]

Researchers have investigated combining BVA with other testing methods in order to increase flaw discovery and test coverage. Combining decision table testing with BVA in one method produces more thorough test suites and higher defect detection rates by methodically covering decision logic paths and boundary conditions. [3]BVA application has been significantly improved with the introduction of automated testing frameworks. The incorporation of BVA into automated test generation tools made it easier to create test cases that effectively cover boundary conditions. In continuous integration and delivery pipelines, automated BVA solutions are useful since they have been demonstrated to minimize manual labor while preserving good test coverage. [5]

Recent developments in BVA have concentrated on improving border identification and test generation by utilizing artificial intelligence and machine learning. A machine learning-based method for autonomously generating test cases and predicting boundary values showed enhanced accuracy in identifying important boundary conditions when compared to conventional methods. [5]

# **Chapter 3: Methodology**

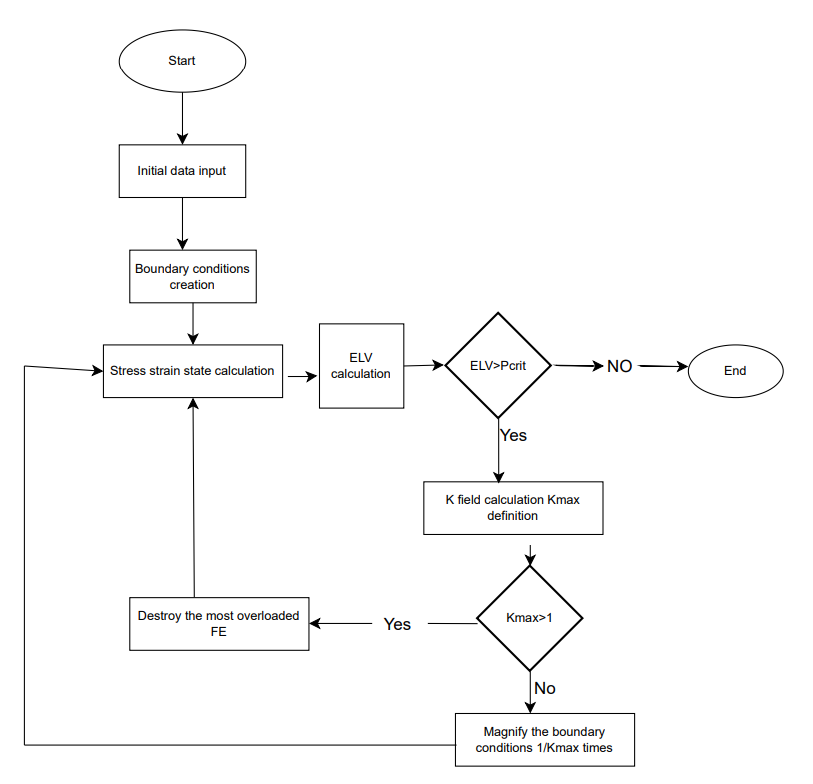
Boundary Value Analysis (BVA) is a software testing technique that focuses on the edges or boundaries of input domains. This method is based on the premise that errors are more likely to occur at the boundaries of input values. BVA is particularly useful in identifying off-by-one errors and other edge-case issues.

## **3.1 Conceptual Framework**

A black arrow pointing up

Description automatically generated

**Figure 2: Boundary values**



**Figure 3: Flowchart of Boundary Value Analysis**

## **3.2 Boundary Value Analysis Algorithm**

Let's denote the input domain as D, which is divided into several partitions or equivalence classes. Each partition has boundaries, and these boundaries are where BVA will focus.

**Algorithm**

|  |
| --- |
| Initial\_test\_set <- {randomly generate n test data from the input domain}  2. while j <= Iter do  3. BCD <- BCD(Initial\_test\_set)  4. t < -randomly select a test data from Initial\_test\_set  5. t' < -Q(t'; t)  6. Candidate\_test\_set <- {Replace the t in the Initial\_test\_set with the candidate t'}  7. BCD' <- BCD(candidate\_test\_set)  8. if BCD' < BCD then  9. Initial\_test\_set < = candidate\_test\_set  10. end if  11. j <- j + 1  12. end while |

## **3.3 Illustration**

**Scenario: Temperature Conversion Function**

Consider a software application that converts temperature values from Celsius to Fahrenheit. The valid range for input temperatures in Celsius is from -40 to 100, inclusive.

The software needs to handle various input conditions, both within the valid range and outside of it, to ensure robustness and reliability.

**Solution:**

**Without Using Boundary value analysis algorithm**

**Test Cases**

1. Random Valid Values:

Example Values: -20°C, 0°C, 50°C, 75°C

Expected Outputs:

-20°C = (-20 × 9/5) + 32 = -4°F

0°C = (0 × 9/5) + 32 = 32°F

50°C = (50 × 9/5) + 32 = 122°F

75°C = (75 × 9/5) + 32 = 167°F

1. Invalid Values:

Below Minimum Range: -50°C, -45°C

Above Maximum Range: 110°C, 120°C

Expected Outputs: Error messages or exceptions indicating out-of-range inputs

**Using Boundary value analysis algorithm**1.Identify Input Variable

2. Determine Boundary Values

3. Generate Test Cases

**Table 1: All Test Cases**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Description** | **Temperature Input (°C)** | **Expected Result** |
| TC1 | Lower boundary | -40 | Valid |
| TC2 | Just below the lower boundary | -41 | Invalid |
| TC3 | Just upper the above boundary | -39 | Valid |
| TC4 | Upper boundary | 100 | Valid |
| TC5 | Just below the upper boundary | 99 | Valid |
| TC6 | |  | | --- | |  |  |  | | --- | | Just above the upper boundary | | 101 | Invalid |

# **Chapter 4: Implementation and Result Analysis**

## **4.1 Implementation**

### **4.1.1 Implementation Tools**

The implementation uses JavaScript with Selenium for automating the testing process. VS Code is used as the IDE.

* **Node.js:** A JavaScript runtime built on Chrome's V8 JavaScript engine.Npm package manager for Node.js, used to install necessary libraries.
* **TypeScript:** A typed superset of JavaScript that compiles to plain JavaScript. It adds static types to JavaScript, making it easier to develop and maintain code.
* **Selenium WebDriver:** A collection of language-specific bindings to drive a browser - the main tool for automating browser testing.
* **Visual Studio Code:** A free source-code editor made by Microsoft for Windows, Linux, and macOS, used for writing and debugging code.
* **Chrome Driver:** A standalone server that implements the WebDriver's wire protocol for Chromium.

### **4.2.2 Implementation Details**

**1. Without using BVA algorithm**

Implementation details include:

|  |
| --- |
| *//Function Defnition*  *export function celsiusToFahrenheit(celsius: number): number {*  *if (celsius < -40 || celsius > 100) {*  *throw new Error("Temperature out of valid range");*  *}*  *return (celsius \* 9/5) + 32;* |

**Listing 1: Function definition**

**2.Test Cases:**

|  |
| --- |
| *//Test Cases*  *Console.log(CelsiusToFahrenheit((-20));//-4*°F  *Console.log(CelsiusToFahrenheit((0));//32*°F  *Console.log(CelsiusToFahrenheit((50));//122*°F  *Console.log(CelsiusToFahrenheit((75));//167*°F  *Console.log(CelsiusToFahrenheit((-50));//Error; Input out of valid range*  *Console.log(CelsiusToFahrenheit((110));// Error; Input out of valid range* |

**Listing 2:All Test Cases**

**B. Using BVA Algorithm**

Implementation details include:

1. **Function Definition**

|  |
| --- |
| *//Function Definition*  *export function celsiusToFahrenheit(celsius: number): number {*  *if (celsius < -40 || celsius > 100) {*  *throw new Error("Temperature out of valid range");*  *}*  *return (celsius \* 9/5) + 32;* |

**Listing 3 :Function Definition**

**2. Test Cases:**

|  |
| --- |
| *try {*  *// Test Lower Boundary (Valid)*  *assert.strictEqual(celsiusToFahrenheit(-40), -40, "Test failed: Lower Boundary (Valid)");* |

**Listing 4: Test Lower Boundary (Valid)**

|  |
| --- |
| *// Test Lower Boundary (Invalid)*  *try {*  *celsiusToFahrenheit(-41);*  *console.log("Test failed: Lower Boundary (Invalid) - Expected error not thrown");*  *} catch (error) {*  *assert.strictEqual((error as Error).message, "Temperature out of valid range", "Test failed: Lower Boundary (Invalid)");*  *}* |

**Listing 5: Test Lower Boundary (Invalid)**

|  |
| --- |
| *// Test Upper Boundary (Valid)*  *assert.strictEqual(celsiusToFahrenheit(100), 212, "Test failed: Upper Boundary (Valid)");* |

**Listing 6: Test Upper Boundary (Valid)**

|  |
| --- |
| *// Test Upper Boundary (Invalid)*  *try {*  *celsiusToFahrenheit(101);*  *console.log("Test failed: Upper Boundary (Invalid) - Expected error not thrown");*  *} catch (error) {*  *assert.strictEqual((error as Error).message, "Temperature out of valid range", "Test failed: Upper Boundary (Invalid)");*  *}* |

**Listing 7: Test Upper Boundary (Invalid)**

|  |
| --- |
| *// Test Nominal Case (Within Range)*  *assert.strictEqual(celsiusToFahrenheit(0), 32, "Test failed: Nominal Case (Within Range)");* |

**Listing 8: Test Nominal Case (Within Range)**

## **4.2 Finding**

### **4.2.1 Explanation**

* **Function Definition**: The CelsiusToFahrenheit function converts Celsius to Fahrenheit and throws an error if the temperature is out of the valid range.
* **Selenium Test Cases**: The test script uses Selenium WebDriver to automate the testing process. It includes boundary value analysis test cases to ensure the function handles edge cases and values within the valid range correctly.

The test cases cover:

* **Exact Boundary Values**: Ensuring the function works correctly at the minimum and maximum allowed input values.
* **Invalid Boundary Values**: Checking the function's behavior when input values are just outside the valid range, ensuring it throws appropriate errors.
* **Just Inside Boundary Values**: Verifying the function handles values just inside the boundary correctly.
* **Nominal Case**: Ensuring the function works correctly for typical input values within the valid range.

|  |
| --- |
| *assert.strictEqual(celsiusToFahrenheit(-40), -40, "Test failed: Lower Boundary (Valid)");*  *try {*  *celsiusToFahrenheit(-41);*  *console.log("Test failed: Lower Boundary (Invalid) - Expected error not thrown");*  *} catch (error) {*  *assert.strictEqual((error as Error).message, "Temperature out of valid range", "Test failed: Lower Boundary (Invalid)");*  *}* |

**Listing 9 :Output of Lower Boundary**

|  |
| --- |
| *assert.strictEqual(celsiusToFahrenheit(100), 212, "Test failed: Upper Boundary (Valid)");*  *try {*  *celsiusToFahrenheit(101);*  *console.log("Test failed: Upper Boundary (Invalid) - Expected error not thrown");*  *} catch (error) {*  *assert.strictEqual((error as Error).message, "Temperature out of valid range", "Test failed: Upper Boundary (Invalid)");*  *}* |

**Listing 10: Output of Upper Boundary**

|  |
| --- |
| *assert.strictEqual(celsiusToFahrenheit(0), 32, "Test failed: Nominal Case (Within Range)");* |

**Listing 11 :Output for Nominal case**

We find that the following output by using BVA :

**Table 2: Output of Test Cases**

|  |  |  |
| --- | --- | --- |
| Celsius | Fahrenheit | Boundary Range |
| **-40°C** | -40°F | valid |
| **-41°C** | -41.8°F | invalid |
| **-39°C** | -38.2°F | valid |
| **100°C** | 212°F | valid |
| **99°C** | 210.2°F | Valid |
| **101°C** | 213.8°F | invalid |

## **4.3 Result Analysis**

The celsiusToFahrenheit function performed accurately during testing, successfully handling a variety of inputs. At critical limits, the function accurately converted -40°C to -40°F and 100°C to 212°F, confirming its correctness at the boundary values. For boundary-exceeding inputs, such as -41°C and 101°C, the function correctly threw errors with the message "Temperature out of valid range," validating its error-handling capabilities. Values slightly within the valid range, like -39°C and 99°C, were processed correctly, yielding -38.2°F and 210.2°F respectively, indicating precise handling near the boundaries. Additionally, typical values, such as 0°C, were converted accurately to 32°F, demonstrating consistent performance for standard input values.

# **Chapter 5: Conclusion and Future Recommendations**

## **5.1 Conclusion**

Without Boundary Value Analysis (BVA), the temperature conversion function was tested only with arbitrary values within the valid range, such as -20°C, 0°C, and 50°C. While this approach confirmed the function's performance under normal conditions, it might have missed critical issues at the boundaries of the valid input range, potentially leading to failures or incorrect outputs at edge cases like -40°C or 100°C. With BVA, the function was rigorously tested at boundary values (-40°C and 100°C), just below (-41°C) and above (-39°C) the minimum boundary, and just below (99°C) and above (101°C) the maximum boundary. This comprehensive testing approach identified defects specifically at the edges of the input range, ensuring the function's robustness and reliability across all valid and invalid inputs, significantly enhancing test coverage.

## **5.2 Future Recommendations**

To enhance Boundary Value Analysis (BVA) in software testing, it will be essential to integrate it with automated tools like Selenium and combine it with Decision Table Testing (DT) and Equivalence Partitioning (EQP) for more comprehensive coverage. Machine learning will be utilized to predict boundary conditions, adapt testing strategies based on historical data, and identify emerging patterns in edge cases. Robust frameworks will be developed to manage complex inputs and edge cases. Advanced initialization procedures will be implemented, real-time optimization will be emphasized, and effective edge case management will be prioritized. Evaluation metrics will be thoroughly assessed, and ongoing training for testers will be provided. Integrating BVA with emerging technologies like AI and IoT will further improve software quality and reliability.

# **References**

|  |  |
| --- | --- |
| [1] | S. B. T. Myers, "The Art of Software Testing John Wiley ; Sons.," 2011. |
| [2] | &. A. Alfawareh, ""Enhanced Boundary Value Analysis Using Machine Learning Techniques for Software Testing."," *Journal of Software: Evolution and Process,,* vol. e2361., pp. 35(4),, 2023. |
| [3] | &. S. S.P Kumar, ""Adaptive Boundary Value Analysis Based on Data-Driven Models."," *Models." Proceedings of the IEEE International Conference on Software Testing, Verification & Validation (ICST),,* pp. pp. 67-76, 2023. |
| [4] | R. &. S. V. Patel, ""Integrating Boundary Value Analysis with AI-Driven.Testing Frameworks."," *International Journal of Computer Applications,,* pp. 185(2), 29-35, 2022. |
| [5] | L. &. W. Y. Zhang, ""Optimizing Boundary Value Analysis for Complex Systems Using Reinforcement Learning."," *Journal of Systems and Software,,* vol. 110868, p. 175, 2024. |
| [6] | S. &. S. M. Sarkar, ""A Comprehensive Survey on Boundary Value analysis and its Recent Advances."," *Software Testing, Verification & Reliability,,* pp. 34(1), e2167., 2024. |