**SOFTWARE IMPLEMENTATION REPORT**

a) COVER PAGE

CONTENTS:

1. **COVER PAGE**
2. **IMPLEMENTATION SUMMARY**
3. **CHANGES TO THE ALGORITHM AND USER INTERFACE DESIGN**
4. **TESTING**
5. **PLANNING AND MONITORING PROGRESS**

# b) IMPLEMENTATION SUMMARY

FUNCTIONAL REQUIREMENTS

1. The system must allow users to scan a QR code containing shape information.
2. The system must parse the QR code to extract the shape type and dimensions.
3. The system must validate the input to ensure:
   1. Shape type is either a square (S), triangle (T), or circle (C).
   2. Side lengths are between 15 cm and 85 cm.
   3. Triangle inputs follow the Triangle Inequality Theorem.
4. The system must display an error message if the input is invalid and prompt the user to re-enter a valid QR code.
5. The system must calculate the movement time required for SwiftBot to draw the shape based on predefined speed calibration data.
6. The system must instruct the SwiftBot to move and turn correctly to draw the specified shape.
7. The system must provide real-time feedback on progress (e.g., percentage completion bar).
8. The system must indicate completion by blinking SwiftBot’s green lights.
9. The system must log the drawn shape details, including:
   1. Shape type and dimensions
   2. Time taken to complete the drawing
   3. Largest shape drawn (by area)
   4. Most frequently drawn shape
   5. Average drawing time
10. The system must prompt the user to either scan a new QR code or exit the program.

NON-FUNCTIONAL REQUIREMENTS

1. Usability: The command-line interface must be intuitive and provide clear instructions to the user.
2. Performance: The system must process the QR code and begin drawing within 3 seconds of input validation.
3. Reliability: The system must correctly interpret input at least 99% of the time.
4. Portability: The program must run on different operating systems (Windows, macOS, Linux) that support Java.
5. Availability: The program must provide a continuous operation mode until the user chooses to exit.
6. Maintainability: The system codebase must be modular, making it easy to add support for additional shapes in the future.
7. Scalability: The system must support multiple shapes per QR code input (up to 5 shapes in sequence).
8. Security: The system must prevent invalid or malicious QR code inputs from crashing the application. ADDITIONAL FUNCTIONALITIES
9. The system allows scanning and drawing multiple shapes sequentially using '&' as a delimiter.
10. The system logs and calculates the angles of triangles for further analysis.
11. The system adjusts turn times dynamically to minimize alignment errors.
12. The system provides a blinking light effect upon successful drawing completion.
13. The system implements error handling for unexpected QR scan failures.
14. The system includes support for drawing circles (C), which was added as an additional functionality.

# c) CHANGES TO THE ALGORITHM AND USER INTERFACE DESIGN

ALGORITHM DESIGN

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Algorithm Design** | **Program** | **Alignment** |
| **QR Code Scanning** | Reads a QR code for shape input. | Uses  swiftBot.getQRImage() and  swiftBot.decodeQRImag e(QR). | Matches |
| **Error Handling** | If the QR code format is invalid, request new input. | Checks input format and prompts for new input if invalid. | Matches |
| **Side Length Validation** | Checks if lengths are between **15 cm - 85 cm**. | Uses  validateSideLength() method. | Matches |
| **Triangle Validation** | Ensures sides satisfy the  Triangle Inequality Theorem. | Implements  validateTriangle() function. | Matches |
| **Square Drawing** | Move forward → turn left 90° (repeat 4 times). | Implements a loop to move forward and turn  90° in the Square class. | Matches |
| **Triangle Drawing** | Move forward for each side, turn by calculated angles. | Uses calculateAngle() and moves for each side in Triangle class. | Matches |
| **Circle Drawing** | (Not originally in flowcharts) | Moves forward and turns  incrementally to approximate a circle. | Added later |
| **Progress Updates** | Displays real-time progress. | Uses percentage-based completion messages. | Matches |
| **Completion Indicator** | Blinks green lights when the shape is finished. | Implements blinkUnderlights() method for green light blinking. | Matches |
| **Shape Logging** | Stores details: type, dimensions, time taken, largest shape. | Uses saveLog() to record shape data. | Matches |
| **Restart or Exit** | Prompts the user to either scan a new QR code or exit. | Provides an option to restart or exit in the main() method. | Matches |

DIFFERENCES AND ADJUSTMENTS

|  |  |
| --- | --- |
| **Change** | **Explanation** |
| **Circle Shape Added** | The algorithm design didn't originally include circles, but the implementation added them as an **additional feature**. |
| **Real-Time Progress Tracking** | The flowcharts don't explicitly show real-time feedback, but the code provides **progress messages at each step** (e.g., "50% Complete"). |
| **Dynamic Turn Adjustments** | The code adjusts turns dynamically to **improve accuracy**, which wasn't specified in the algorithm design. |
| **Enhanced Logging** | The logging mechanism in the code includes details like the **largest shape drawn**, which wasn't explicitly shown in the algorithm. |

USER INTERFACE DESIGN

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **UI Design** | **Program** | **Alignment** |
| **Welcome Screen** | Displays a title, program instructions, and example QR input format. | Prints an introduction, usage instructions, and valid input examples. | Matches |
| **QR Code Input** | User enters a QR code manually. | Reads QR code using swiftBot.getQRImage() but allows manual input if needed. | Matches |
| **Input Validation** | Displays errors for incorrect format, side length out of range, or invalid triangle sides. | Implements validation with error messages and prompts for re-entry. | Matches |
| **Real-time Drawing Progress** | Displays a percentage progress bar (e.g., 50%, 80%) while drawing. | Uses system output to print progress updates at different stages. | Matches |
| **Completion Indicator** | Displays a success message and blinks  SwiftBot's green lights. | Implements blinkUnderlights() to indicate completion. | Matches |
| **Drawing Log** | Logs details such as shape type, size, and time taken. | Saves and prints log data, including largest and most frequent shape. | Matches |
| **Restart or Exit** | Asks if the user wants to scan a new QR code or exit. | Prompts the user for a new scan or to exit in main(). | Matches |

DIFFERENCES AND ADJUSTMENTS

|  |  |
| --- | --- |
| **Change** | **Explanation** |
| Circle Shape Not in UI Prototype | The prototype only mentions squares and triangles, but the final implementation added **circle drawing** as an enhancement. |
| More Detailed Progress Updates | The implementation prints incremental progress (e.g., **25%, 50%, 80%**), which adds **better user feedback** compared to the prototype. |
| QR Code Handling | The UI prototype assumes manual entry, while the code first tries to scan a QR code before allowing manual input. |

# d) TESTING

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case Description** | **Requirem ent No.** | **Test Data** | **Expected Output** | **Observed Output** | **Pass/Fail** |
| **TC-01** | Welcome message and instructions display correctly. | FR-1 | N/A (Start the program) | Welcome message with shape input instructions is displayed. | Message appears as expected. | **Pass** |
| **TC-02** | The system allows scanning and manual input of QR code. | FR-2 | S-30 (manual input) | The system accepts input and starts processing. | Input accepted, shape processing begins. | **Pass** |
| **TC-03** | Validation of square input with correct range. | FR-3 | S-40 | System accepts the square and starts drawing. | Square drawn successfully. | **Pass** |
| **TC-04** | Validation of triangle input with correct range. | FR-3 | T-30-40-50 | System accepts the triangle and starts drawing. | Triangle drawn successfully. | **Pass** |
| **TC-05** | Validation of circle input with correct range. | Additional Feature | C-50 | System accepts the circle and starts drawing. | Circle drawn successfully. | **Pass** |
| **TC-06** | Rejects square input outside valid range. | FR-3 | S-100 | Error message: “Side length must be between 15 cm -  85 cm.” | Error message displayed correctly. | **Pass** |
| **TC-07** | Rejects triangle input outside valid range. | FR-3 | T-10-20-30 | Error message: “Side length must be between 15 cm -  85 cm.” | Error message displayed correctly. | **Pass** |
| **TC-08** | Rejects invalid triangle (fails  Triangle  Inequality). | FR-3 | T-30-10-5 | Error message: “Invalid triangle sides.” | Error message displayed correctly. | **Pass** |
| **TC-09** | Rejects invalid input format. | FR-3 | X-30 | Error message: “Invalid shape type.” | Error message displayed correctly. | **Pass** |
| **TC-10** | Real-time progress updates during drawing. | FR-7 | S-30 | Displays 25%, 50%, 75%, and 100% progress updates. | Progress updates appear as expected. | **Pass** |
| **TC-11** | Blinks green lights upon shape completion. | FR-8 | S-30 | SwiftBot blinks green upon completion. | Green blinking confirmed. | **Pass** |
| **TC-12** | Logs shape details correctly. | FR-9 | S-30 & T-  20-30-40 | Log should show shape type, dimensions, time taken, and most frequent shape. | Log correctly updates and displays statistics. | **Pass** |
| **TC-13** | Detects the largest shape drawn. | FR-9 | S-50 & S-70 | System logs the largest shape as S70. | Correctly identifies S-70 | **Pass** |
|  |  |  |  |  | as the largest shape. |  |
| **TC-14** | Allows scanning multiple shapes in one QR code. | FR-10 | S-30 & T20-30-40 &  C-50 | Processes and draws all shapes sequentially. | All shapes drawn in order. | **Pass** |
| **TC-15** | Restart prompt after drawing completion. | FR-10 | N/A (After drawing a shape) | Asks the user: "Would you like to scan a new QR code?" | Correctly prompts the user. | **Pass** |
| **TC-16** | System exits correctly upon user selection. | FR-10 | 2 (Exit  command) | Program exits gracefully. | Exits without errors. | **Pass** |

# e) PLANNING AND MONITORING PROGRESS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Week** | **Planned Tasks** | **Completed**  **Tasks** | **Challenges**  **Faced** | **Solutions Implemented** | **Adjustments to Plan** |
| Week 1 | * Understand the project requirements. * Set up the development environment. * Research SwiftBot   API documentation. | - Reviewed requirements and user interface prototype. - Installed necessary software (Java, SwiftBot API). | Initial difficulty understanding SwiftBot API's movement functions. | Explored SwiftBot documentation  and experimented with basic  movement commands. | No changes; proceeded as planned. |
| Week 2 | - Implement QR code scanning and input validation. - Implement error handling for incorrect inputs. | - Developed QR scanning functionality. - Implemented validation for squares and triangles. - Added error messages for invalid inputs. | Triangle validation (Triangle  Inequality  Theorem) was complex to implement. | Used test cases to debug triangle validation logic. | No major adjustments, but required additional debugging time. |
| Week 3 | - Implement shape drawing functions  (square, triangle). - Implement progress tracking messages. | * Developed square and triangle drawing logic. * Implemented real-time progress updates. | SwiftBot’s turn angles were inconsistent, causing misalignment. | Adjusted turn time values and tested multiple configurations to improve accuracy. | Added **dynamic turn adjustments** for better accuracy. |
| Week 4 | - Test drawing functions and implement logging system. - Refine shape statistics tracking (largest and most frequent shape). | - Fully tested square and  triangle drawing. - Implemented shape logging system. | Log system  initially did not correctly track the most frequent shape. | Adjusted the counting logic and verified outputs with multiple test cases. | No changes, but required additional debugging time. |
| Week 5 | - Add additional functionalities (circle drawing). - Conduct systemwide testing with different inputs. | - Implemented circle drawing as an enhancement. - Conducted full system testing. | Circle  movement required smooth turning adjustments to form a proper shape. | Used **incremental turns** for smooth circular motion. | Added **circle drawing** to  final  implementati on (not in original plan). |
| Week 6 | - Finalize  documentation (test plan, UI comparison, algorithm validation). - Prepare submission. | - Completed all documentation sections. - Reviewed all requirements to ensure full implementation. | Ensuring that all test cases covered different edge cases. | Double-checked requirements and UI alignment with implementation. | No major adjustments; prepared for submission. |