4.14 Implementation & Testing

Use this checklist to make sure your implementation and testing sections clearly show what you built, how you built it, whether it works, and how well it meets your objectives. The aim is not just to describe your program, but to provide evidence of a complete, robust and technically demanding solution.

1. Implementation: what to include

- Explain how the main parts of your solution were implemented, not just what they do.
- Show evidence of the most important code sections, especially where you used complex or interesting techniques.
- Link your implementation back to your objectives, design and success criteria.
- Include screenshots, code snippets or short explanations that prove features have actually been built.
- Comment on any changes from your original design and explain why the changes were needed.
- Make it clear how data is stored, processed, validated and output by the system.
- Highlight important algorithms, data structures, classes, functions, databases, files, APIs or user interface features.

2. Implementation evidence checklist

Evidence to include	Have I done this?
I have shown the key parts of my code and explained why they are important.	☐
I have explained how each major feature helps meet a user need or objective.	☐
I have included evidence of validation, error handling and robustness.	☐
I have explained any complex techniques, algorithms or data structures used.	☐
I have shown how my solution changed during development and why.	☐
I have avoided pasting pages of code without explanation.	☐

3. Testing: what to include

- Create a test plan that covers the main requirements of your system.
- Use normal, boundary and erroneous test data where appropriate.
- State the purpose of each test so it is clear what feature or requirement is being checked.
- Record the expected outcome before testing and the actual outcome after testing.
- Include evidence such as screenshots, output examples, database changes, logs or short video references.
- Comment on whether each test passed or failed, and what you changed if something did not work.
- Show that your solution is reliable and robust, not just that it works once with ideal data.

4. Suggested testing table

Test number	Feature tested	Test data	Expected result	Actual result	Evidence	Pass/fail and action taken
1						
2						
3						

5. Final student checks

- Can someone else understand how my program was built from my explanation?
- Have I used evidence rather than unsupported claims?
- Have I tested every important objective or requirement?
- Have I included failed tests and explained how I fixed the problems?
- Have I kept screenshots purposeful, labelled and easy to follow?
- Have I shown enough technical detail to demonstrate the complexity of my solution?