Homework 4: Reflection

1 What could I have done better to reduce the time spent solving this homework?

I could have improved my time management by starting with a clear, high-level design sketch before diving into detailed specifications or implementation. Initially, I spent more time than necessary refining individual method specifications before having a clear overall picture of how the API would fit together. In retrospect, I should have:

1. Created a simple UML diagram or sketch of the main components and their relationships
2. Identified the core operations needed for the GraphWrapper interface first
3. Validated my overall design approach by creating minimal stubs for the required classes
4. Progressively refined the specifications and implementation

Additionally, I could have referred to more example ADTs from previous assignments earlier in the process to better understand the expected style and detail level for specifications.

2 What could the teaching staff have done better to improve my learning experience?

The assignment was generally well-structured, but a few improvements could enhance the learning experience:

1. Providing a small example of a complete specification (AF, RI, and several methods) specifically for a graph-like structure would help establish expectations for specification style.
2. Including more detailed guidance on how to effectively test graph structures beyond just the "0-1-2" heuristic would be beneficial, perhaps with examples of effective property-based testing for graphs.

3. Adding optional stretch goals for students who finish early, such as implementing additional traversal algorithms (DFS, BFS) or graph properties (connected components, cycles), would provide opportunities for deeper learning.

3 What do I know now that I did not know before beginning the homework?

Through this assignment, I gained several important insights:

1. A deeper understanding of the relationship between abstract specifications and concrete implementations. I now better comprehend how a well-designed ADT can hide implementation details while providing a useful interface.

2. Practical experience with designing mutable data structures that maintain invariants while allowing for modification. This is more complex than designing immutable data structures.

3. An appreciation for the importance of representation choice in graph algorithms. The efficiency tradeoffs between adjacency lists, adjacency matrices, and edge lists are now much clearer to me, particularly for sparse vs. dense graphs.

4. A better grasp of how to design tests that verify both correctness and adherence to performance requirements. Testing a graph ADT requires carefully considering edge cases like reflexive edges and multigraph capabilities.

5. Improved understanding of how to write clear, accurate specifications that are both precise and understandable to clients of an ADT.