# Quote Loader Component

A junior developer was asked to create a reusable stock quote loader component, which should import quotes from a specified tab-separated file into the internal application database. Access to the database was provided by the *IQuoteRepository* interface located in the *Quotes* assembly. As a result, the junior developer produced *QuoteLoader* and *QuoteLoader.Tests* projects attached to this task.

During regular code review, a team lead pointed to a low quality of *QuoteLoader* code. One his concern was about the CSV reader/writer component, which is not generic enough to handle **various possible sources** of stock quotes. Another his comment was about **poor unit test coverage**.

## Task

Your task will be to focus on issues, which team leader considered the most critical for now, such as the CSV reader/writer component.

1. Review the existing CSV reader/writer implementation and **create a complete list of problems** you see in the code (i.e. for junior developer education purposes).
2. **Refactor** the CSV reader/writer component into a clean, performant and elegant code, without over-engineering. Make sure you follow **best practices for design** and implementation. Annotate your code with comments and explain any trade-offs you make.
3. Assume the CSV reader/writer already used in production, and **backward compatibility** must be maintained. Therefore, make sure your changes do not break any contracts. If **you decide to change** public interface in any way, please deprecate existing methods.
4. Make sure you write enough unit tests to achieve near 100% code coverage for CSV reader/writer component, also testing various exceptional cases. Feel free to introduce any mocking frameworks available in NuGet if needed.
5. public bool Read(out string date, out string ticker, out string value)
6. Use string.IsNullOrEmpty(line) instead line == null || line == string.Empty

1. var data = line.Split('\t');//separator
2. \_writer = new StreamWriter(fileName);// is exist
3. \_writer = new StreamWriter(fileName); //using
4. Read(…) // RealLine and ReadLines
5. Open Close to Dispose
6. Write Read as two classes
7. Check if file exists
8. StreamReader as dependency
9. Use IoC framework
10. Many not required ‘using’ directives
11. Order ‘using’ derectives. User directives write after standart libs directives
12. Exceptions and errors
13. Inject new CsvReaderWriter()
14. Use data structs
15. Use interfaces for injections
16. Use decimal type for money
17. Using in QuoteImporter for CsvReaderWriter
18. Error Logger
19. Parse as TryParse
20. Parse helpers
21. Use culture to parse date, money
22. BaseImporter and QuoteImporter for different types of Quote, etc
23. Tests for errors, exceptions and max-min boundary cases
24. Not dublicate code of FakeQuoteRepository
25. Write class to own file
26. Generate test data not in FakeQuoteRepository
27. Not enough test data for all code cases
28. Use Moq framework
29. When you design an program architecture you need think like “Is it possible to add a new data class or data source and re-use existed code?” For example: Quote2 with Excel file.