**Suggestions of improvements in order to make the product easier to be tested by automation.**

1. A **shift-left testing approach** reflects this idea and suggests conducting testing activities from the very beginning of the development process instead of making it a final step as traditional methodologies typically suggest.
2. **Implement a test-friendly architecture**: Create a test-friendly architecture that allows automation tools to interact with the product easily. Use consistent naming conventions, add unique identifiers (e.g., IDs, classes) to elements, and provide well-defined APIs or hooks for test automation to access and manipulate the product's functionalities.
3. **Understand the product -** A graphical model, like a mind map, can represent the product in a clear, understandable manner.
4. **Build a comprehensive test suite:** Develop a robust and comprehensive set of test cases that cover the product's functionality. Include both positive and negative test scenarios, edge cases, and stress tests. This ensures that the product is thoroughly tested, and any potential issues are identified early**.**
5. **Beginning with a pilot experiment -** engineers might want to shorten the regression test runtime from two weeks to 11 days. Running an initial small-scale project before adopting a full-scale automation testing approach might also give you a better idea of whether it is viable in the long term.
6. **Create versatile tests -** tests remain stable and unaffected by UI changes to make your automated tests run smoothly even without frequent adjustments.
7. **A formal technical review (FTR) -** developed software meets the predefined standards and requirements and Automation scripts can be improved or not.

**Testing Strategies**

1. **Test Automation:** Utilize automated testing frameworks to cover a wide range of tests, including unit tests, integration tests, and end-to-end tests. This approach enables fast and reliable feedback on code changes, ensuring that defects are caught early in the development process.
2. **Continuous Integration**: Implement a continuous integration (CI) process that integrates code changes frequently. With each commit, the CI system should automatically build the application, run all relevant tests, and provide immediate feedback on the code quality and test results. This helps identify and resolve issues quickly, reducing the risk of introducing bugs into the codebase.
3. **Continuous Deployment:** Combine continuous integration with continuous deployment (CD) to automate the release process. Once code changes pass all tests successfully, the CD system should automatically deploy the application to the staging or production environment. This approach ensures that the latest changes are promptly delivered to users, enhancing speed and responsiveness.
4. **Test Coverage and Monitoring**: Maintain a comprehensive test suite that covers critical functionality and edge cases. Regularly review and update the test suite to reflect new features and changes in the application. Additionally, implement monitoring and alerting mechanisms to proactively identify and address issues in the production environment. (Cover functional & non-functional testing)
5. **Performance and Load Testing**: Include performance and load testing as part of the testing strategy to evaluate the system's behavior under different levels of stress. By simulating real-world scenarios, you can identify performance bottlenecks, optimize the application, and ensure it can handle the expected workload.
6. **Feedback Loop and Iterative Improvement**: Establish a feedback loop where developers, testers, and stakeholders collaborate to continuously improve the testing process. Regularly review test results, collect feedback, and make necessary adjustments to enhance the pipeline's speed, confidence, and quality.