CODE REVIEW GUIDELINES

FOR GUI-BASED TESTING

ANDREAS BAUER
Blekinge Institute of Technology
andreas.bauer@bth.se

DECEMBER 02, 2024
Version 0.2.0

This document contains actionable advice on improving the code review process of GUI-based tests. The recommendations are based on an interview study with testers from different companies. More information is provided in the study. This document contains two types of recommendations: (1) information that the tester should provide during a code review and (2) key aspects that the reviewer should consider during the code review.

Provide information during code review

Provide context information

Testers should provide information about the rationale behind proposed test cases, providing a clear explanation of why a particular change is necessary. This rationale should address the following elements:

- *Purpose of the test cases*: Clearly define the objectives of the test case and its intended contribution to the overall system. Specify the feature requirements it aims to validate and describe the overarching context in which the test case operates.
- *Big picture alignment*: Explain how the test case contributes to achieving the system's goals, ensuring reviewers can evaluate its alignment with the project's overall design and objectives.

Further, testers should also provide the following supplementary details:

- *Cross-file dependencies*: If the test case depends on files or components not included in the immediate code review, explicitly outline these relationships.
- *Impact assessment*: Highlight areas where the change might introduce vulnerabilities or performance implications.
- *Edge case coverage*: Identify specific edge cases the test case addresses. This ensures the test suite robustly handles unusual or extreme scenarios.
- *Design and architectural insights*: For complex changes, provide a summary of relevant design or architectural decisions. Consider providing a high-level overview to clarify how the test cases integrate with existing infrastructure.
- *Review prioritization*: When a change consists of many files, prioritize them based on their relevance and criticality to the change. This helps reviewers focus on the most

impactful components and streamlines the review process.

Provide performance and coverage metrics

- *Determine efficiency*: Provide metrics that indicate the execution time of test cases.
- *Determine effectiveness*: Provide metrics that indicate the test coverage.

These metrics should ideally be automated, allowing for consistent and reliable data that can be easily referenced during the review.

Provide information about used test data

- *Selected test data*: Provide information about the test data set (or subset) used for the test case under review.

This helps reviewers assess whether the selected data is appropriate for the test case's purpose and suitable for the specific test environment.

Provide logging information of tests

- *Log files*: If log files from test executions are available, provide access to them during the code review.

Logs provide valuable insights into the execution process and can help identify issues that may not be immediately apparent from the code alone. In general, enhancing logging practices during testing can facilitate deeper analysis and more effective code reviews.

Aspect to consider during the review

Conformity against requirements

Ensure the changes in test cases are aligned with requirements and test specifications so that changes are not implemented or tested the “wrong thing”.

Test Robustness

Reviewers should look for locators that could reduce the robustness of the tests, such as overly simplistic XPath locators that are prone to break with minor changes in the user interface. Encouraging the use of more stable and resilient locators, such as CSS selectors or custom attributes designed for testing, can help improve the longevity and reliability of the tests.

Test Maintenance

Reviewers should evaluate whether test cases are still relevant and aligned with the current requirements of the project due to changes to the system under test or overall testing strategy. Test cases that are not relevant should be removed from the code repository.

Adherence to design principles and patterns

Test cases, like production code, should adhere to recognized design principles and patterns to ensure clarity, maintainability, and reliability. Key considerations include:

- *Focus and simplicity*: Test cases should be concise and target a single functionality or behavior. Avoid bundling multiple responsibilities into a single test case to maintain focus and readability.
- *Separation of concerns*: Ensure test logic is clearly separated from setup, execution, and verification stages. This modularity facilitates reuse and reduces complexity.
- *Documentation*: Provide comprehensive yet succinct documentation within the test code. This includes comments and annotations that clarify the purpose, scope, and methodology of the test cases, aiding future reviewers and maintainers.

- *Testing techniques*: Ensure the appropriate techniques are used for testing and exception handling.

Managing Complexity

Excessive complexity in test cases and testing frameworks should be avoided, as it can hinder maintainability and scalability. Complex solutions increase cognitive load, making it more challenging for future developers/testers to understand, modify, or extend the tests. During the review process, consider the following:

- *Simplification opportunities*: Identify areas where complex logic or structures can be simplified without compromising functionality.
- *Readability*: Ensure test cases are written in a way that is easy to follow, minimizing ambiguity. This includes following coding styles and naming conventions.
- *Scalability*: Evaluate whether the proposed design can accommodate future changes with minimal disruption.

Levels of Abstraction

Reviewers should ensure that different levels of abstraction are not mixed within a single test case. A consistent level of abstraction helps maintain clarity and simplicity in test code. For example, high-level abstractions like page objects should not be combined with low-level details such as direct DOM manipulations within the same test case.

Consideration of Git Commit History

Understanding the development history of a test case can provide valuable insights into its evolution and purpose. Reviewers should consider the git history, including commits and their messages, to understand how the test case was developed and why certain decisions were made.

Disclaimer

License