Litcoder
Powered by Litwork

Code Testability

Case Reference: React

Litcoder Parameter

Code Readability refers to the ease with which the source code can be comprehended and understood by humans. It emphasizes writing clear, concise, and well-structured code, using meaningful variable names, following consistent coding conventions, and incorporating appropriate comments and documentation. Readable code enhances maintainability, reduces the likelihood of errors, and facilitates collaboration among team members, ultimately leading to higher productivity and code quality.

Code Testability is a measure of how easy it is to test the given code. The more paths the code has, the tougher it is to test the same. Also the more branches and checks it has, the more the code becomes brittle. The industry uses a measure called cyclomatic complexity that measures how complex the code is from a testability perspective.

About React

React is an open-source JavaScript library created by Facebook for building interactive user interfaces and web applications. Since its release in 2013, React has grown exponentially in popularity and usage. React code bases tend to be more readable, maintainable, and scalable compared to other libraries. Over 100,000 websites and web apps use React including Facebook, Instagram, Netflix, Airbnb, and more. The test-driven development methodology of React

Challenges in React's Digital Transformation Journey

As the usage of React exploded, several growing pains and technical debt challenges emerged. The frequent release of new React versions made it hard for teams to stay updated and leverage the latest features, while complex dependency trees led to integration issues between React and other libraries. The lack of standardized state management confused choosing Flux, Redux, or MobX. As React codebases scaled, potential vulnerabilities arose, including testing debt due to inadequate test coverage, readability debt caused by poor modularity leading to dense coupled code, maintainability debt slowing velocity over time, and reliability debt resulting in undetected bugs due to lack of tests. React teams acknowledged significant technical debt such as outdated test suits, lack of mocking for dependency, tight coupling between components, and unmanaged state across components which slowed feature development and increased defects leakage.

As React codebases expanded in complexity over time, developers encountered escalating hurdles concerning both quality and speed: Testing and integrating new code changes demanded more time, while the burden of testing debt impeded developers' pace. Inadequate test suites heightened the risk of merging defects, with

prevalent issues such as memory leaks, race conditions, and edge case bugs. Moreover, code readability suffered without modularity, resulting in bloated components, tight coupling, and a lack of reusability.

Solutions that were Adopted

To mitigate technical debt and maintain optimal velocity, React teams prioritized code quality measures. They set clear, quantifiable quality goals early on, establishing thresholds for test coverage, maximum complexity per component, and minimum documentation coverage. Safeguard policies were implemented, necessitating peer code reviews, static analysis in CI pipelines, and re-testing of impacted areas with each code change. Focused training sessions were conducted to instill modular coding best practices throughout the team. Leveraging design patterns facilitated loose coupling between components, better encapsulation, and higher cohesion within components. Ongoing refactoring sprints were planned to simplify complex areas of the codebase. By enforcing these quality standards, the teams ensured continuity of velocity and innovation, even amidst scaling codebases and changes in team composition.

References: Click Here