Enhancing iOS Development with Automated Testing and Tools

This blog post was organized with the assistance of ChatGPT-4o.

The Importance of Unit Testing

At LeanCloud, we implemented unit testing early in our projects, which proved invaluable. Each pull request (PR) triggers unit tests on Jenkins, and we aim for a coverage rate of around 80%. There are two primary scenarios for writing tests: to validate new interfaces and to reproduce and fix bugs. The more tests we accumulate, the stronger our codebase becomes. Automated testing allows us to confidently release and refactor code without manual verification.

Test Flows and Practical Applications

Here are some practical examples of how unit tests help us:

Test Flow 1: A user reported an error when saving an object with a description key. I wrote a test to reproduce the issue, found and fixed the bug, and retained the test for future validation.

Test Flow 2: When developing new interfaces, I write tests after implementing the code to ensure it works correctly.

Test Flow 3: After modifying the `AVObject.m` code, I run `AVObjectTest.m` to check if the changes cause any tests to fail.

Test Flow 4: Submitting a PR triggers automated tests on Jenkins.

Benefits of Writing Unit Tests

- **Reduced Manual Verification:** Unit tests save time by eliminating the need for manual checks.
- **Bug Detection:** Early detection of issues caused by code changes prevents bugs from affecting other parts of the project.
- **Collaborative Projects:** In multi-developer projects, unit tests ensure consistency and reliability, even when handed over to others.
- **High-Quality Open Source Projects:** Popular open-source projects often have extensive unit tests, which contribute to their reliability and popularity.

How to Write Effective Unit Tests

- **Modular Code:** Separate data and UI layers to make testing easier.
- **Maximize Coverage:** Write minimal test code to achieve maximum coverage.

- **Asynchronous Handling:** Ensure tests can handle asynchronous operations.
- **Framework Selection:** Choose the right testing framework for your needs.
- **Coverage Reports:** Use coverage reports to understand which parts of your code are tested.

Evaluating Testing Frameworks

We evaluated several frameworks: - **Expecta:** `expect(error).not.beNil()` - **Specta:** `describe("") it("")` - **Kiwi:** `describe("") it("")` - TDD and BDD frameworks have limitations, such as poor integration with Xcode, lack of test buttons, and incomplete unit test listings in the sidebar.

Handling Asynchronous Tests

Asynchronous testing is crucial for operations that do not complete immediately. Ensure your framework supports async tests effectively. For example, using expectations in XCTest to wait for asynchronous operations to complete before proceeding with assertions.

Coverage Reports

Xcode 7 introduced built-in coverage reports. To enable them: 1. Turn on `Gather Coverage Data` in your scheme settings. 2. Test against the App Target, not the Test Target.

This feature allows developers to see exactly which lines of code are being tested, helping to identify untested parts of the codebase. For more details, visit Big Nerd Ranch’s blog.

Remote Automated Testing with Jenkins

Setting up Jenkins for automated testing involves several steps: 1. **Jenkins Installation:** Set up Jenkins on a local machine or a server in a data center. 2. **GitHub Integration:** Use the GitHub PR build plugin to trigger tests when a pull request is submitted. - Configure webhooks to send events to Jenkins. - Ensure Jenkins can access the latest code from the pull request. 3. **Test Scripts:** Set up test scripts in Jenkins to automate the testing process. - Ensure Jenkins can notify GitHub about test results. - Configure notifications to Slack or email for test failures.

Remote automated testing with Jenkins offers the full benefits of automated tests, surpassing local testing by running tests in a clean, controlled environment.

Remote Packaging and Deployment

While remote packaging may not be necessary for all projects, it can streamline the deployment process for SDKs and other reusable components. This involves: - Configuring Jenkins to read your code. - Reading the release version. - Unlocking the keychain in the command line to access signing certificates.

Additional Tools and Tips

- **Reveal:** Analyzes any app’ s UI on a jailbroken iPhone.
- **Flex:** Analyzes network requests, UI, local files, NSUserDefaults, and logs on a jailbroken iPhone.
- **Pod Management:** Local Pod usage, advanced Podfile configurations, and publishing Pods.
- **Framework Creation:** Differences between dynamic and static libraries, and how to package frameworks for both simulators and real devices.
- **Xcode Tips:** Useful shortcuts like Shift + Command + J for revealing files in the navigator, and Shift + Command + O for quickly opening files.

Conclusion

Automated testing and the right tools significantly enhance the development process. By incorporating unit tests early, leveraging asynchronous handling, and utilizing coverage reports, we can build more reliable and maintainable applications. Tools like Jenkins for CI/CD and Xcode for development, combined with a robust testing strategy, ensure high-quality software delivery.

Acknowledgements

Special thanks to the LeanCloud team and everyone who contributed to our testing processes.