Playbook Methods Repository

# **Test Automation**

Verify existing functionality and behaviour in a consistent, repeatable fashion. Test Automation moves the responsibility of regression testing to code, enabling faster release cycles and a better distribution of testing tasks.

### Remote Agility: **•** High

### Linked Tactic(s): Quality Assurance

## Why we do it:

Test Automation (“Automation”) helps eliminate the expensive, time-consuming manual regression test cycles that are often performed immediately before a new application version is released. Instead, these cycles are performed programmatically and continuously throughout the software development lifecycle, ideally with each code change pushed by developers. Typical Automation tools simulate interactions with the product’s user interface (mouse clicks, finger taps, gestures, etc.), and make assertions on the resulting changes to application state. As such, end-to-end application flows are the best candidates for automation, as they map to how the product is actually used, and allow for multiple aspects of the application to be verified in a single test.

## 

## When to apply it:

* In web or mobile applications: The ecosystem of Automation tools for web and mobile applications is very mature, with many offerings across the different platforms. Unless the application being developed runs on custom hardware or uses a proprietary operating system as its foundation, it should not be difficult to find a suitable tool.
* When supporting infrastructure is available, or being developed: Although automated test suites can be executed on team members’ local machines, this method does not unlock their full value. In order to extract the full value of the suites by executing them as part of CI/CD pipelines, corresponding test environments are required. These are usually cloud-hosted Virtual Machines or Containers, and are now included as part of many CI/CD services, minimizing the amount of setup required.
* As soon as possible! Implementing test automation as soon as application UI is available ensures that the UI is implemented in the most automation-friendly manner possible, and allows for automated regression tests to be created and executed as soon as the flows are complete. Conversely, it’s never too late to implement an automated test suite if a manual regression test is currently being executed repeatedly. The automation suite will provide immediate value in reducing the manual testing effort, and allowing for the verification to be performed at multiple points in the development cycle, not just before release.

## Best Practices & Considerations:

* Be prepared for initial test development to take much longer than anticipated, especially if main flows in the application have already been completed. In these situations the application may have been developed in a way that is not conducive to automation, requiring either development rework, or workarounds in the automation code.
* Ensure that automated tests are setup for success by providing a sufficiently-resourced test environment. Unexpected application timeouts and increased latency are some of the main causes of false failures in automated tests, and are only exacerbated by an underpowered environment with degraded application performance.
* In order to maximize the value of an automated test suite, it should be executed against each change pushed to the source control repository. This ensures that regressions are caught immediately, which reduces the amount of context-switching for developers, and the administrative overhead in fixing the issue(s).
* There are sometimes concerns from teams on the impact of automated testing on CI/CD pipeline execution times. As automated tests have much longer execution times than Unit or Integration tests, team members may fear that the feedback loop from the pipeline will grow exponentially as the test suite grows. Fortunately, there are several strategies for getting ahead of this issue:
  + If the team is greatly concerned about pipeline execution times, a discussion should be had to determine the acceptable threshold for build times. From here, mitigation techniques can be put into place if build times begin exceeding the threshold.
  + Some examples of mitigation techniques include:
    - Partitioning the suite into different categories (e.g., Smoke, Sanity, and Regression), and running a subset of categories on each repository push, and the rest on a fixed schedule. For example, the Smoke category is run on each push, Smoke and Sanity when a pull request is merged into the Main branch, and Smoke, Sanity, and Regression twice a day at noon and midnight, respectively.
    - Parallelize execution of the test suite. Some tools, such as Cypress, offer built-in support for this (for an additional cost), while others require additional configuration or supporting infrastructure.
    - Re-examining build time thresholds, to see if the value added by the test suite justifies an increase in the threshold.

## Responsible roles:

* Software Engineer: Write automation-friendly application code that facilitates the development of automated tests. Support QA team members with the tasks listed in the QA Engineer role, or perform those tasks themselves if the project does not have QA resources.
* Product Manager: Collaborate with QA Engineers to identify the highest-priority application flows and scenarios, which will be used as an input for creating the backlog of tests to be automated.
* QA Engineer: Select and configure the test automation tool(s) for the project. Develop the test frameworks and scenarios, and inform the team of blockers to automation present in the application. Integrate the automated test suite into the CI/CD pipeline, and configure any supplementary tooling, e.g., Test Dashboards. Monitor test results from the pipeline and investigate any failures.

## Tools:

### Online tools/platforms/services

* + Cypress: <https://www.cypress.io>. A modern automated testing tool for web applications that aims to avoid pitfalls and limitations of previous tools. Also offers a paid service that includes features such as integrated test suite parallelization.
  + Selenium Webdriver: <https://www.selenium.dev>. The most widely-used automated testing tool for web applications, with a mature ecosystem of addons and online material.
  + Appium: <https://appium.io>. An automated testing framework for both Android and iOS applications, based on the Webdriver protocol also used by Selenium.
  + XCUITest: <https://developer.apple.com/documentation/xcode/testing-your-apps-in-xcode>. Apple’s own automated testing framework for iOS applications, which is a subset of the overall XCTest framework.
  + Espresso: <https://developer.android.com/training/testing/espresso>. Google’s automated testing framework for Android applications.