End2End Testing

DevOps Advanced

Agenda

- E2E Testing
- Tooling
- Playwright

E2E Testing

Integration Testing

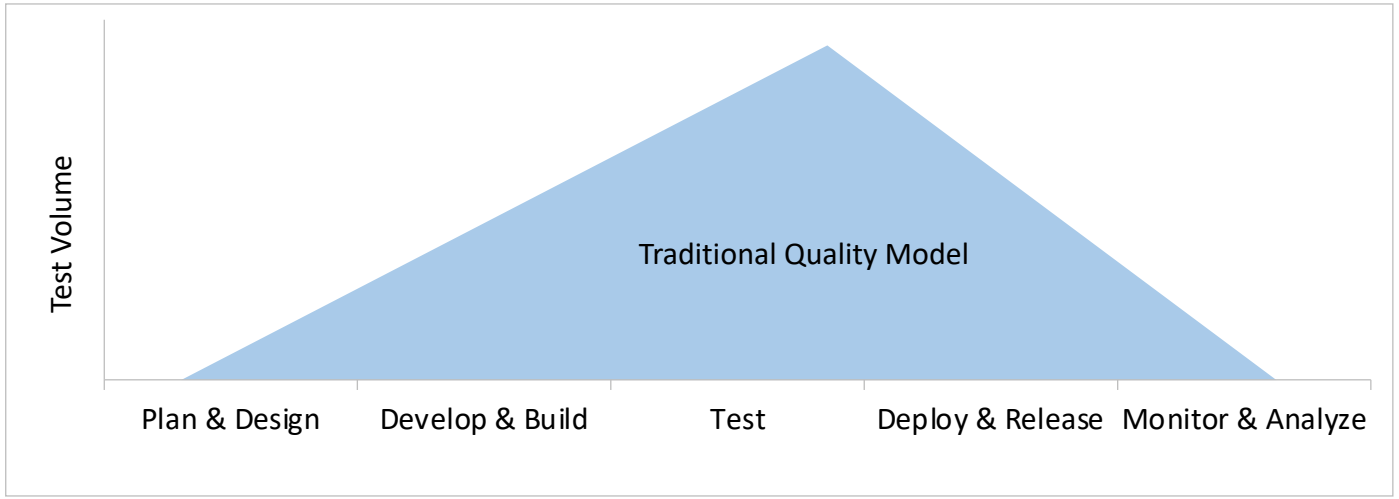
Shift-Left Testing

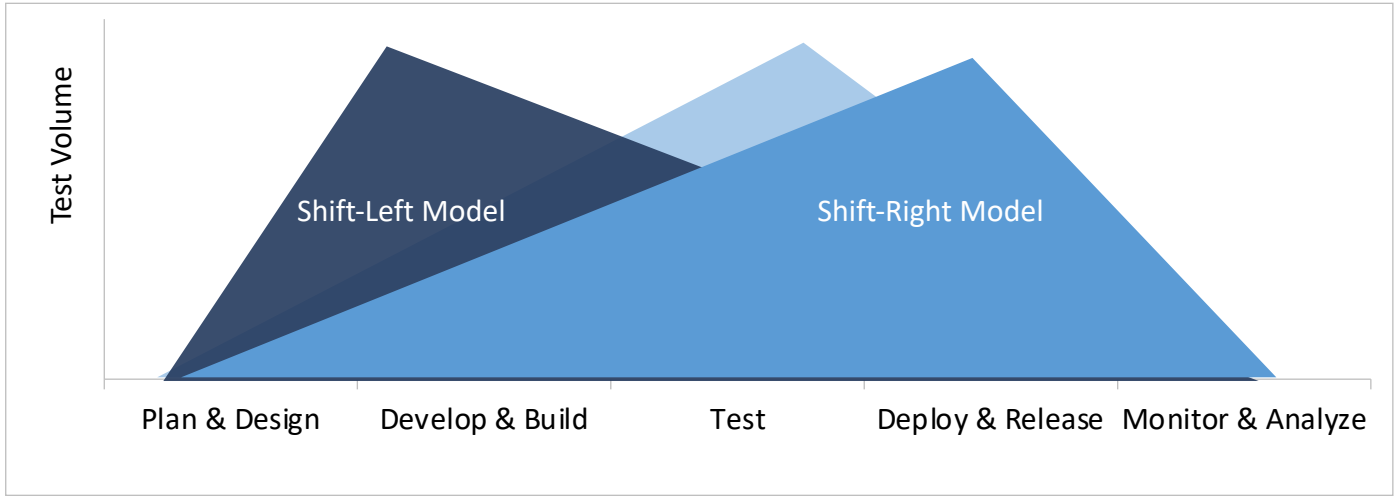
- Shifting left is a reference to moving testing to the left on a timeline
- Approach used to speed software testing and facilitate development by moving the testing process to an earlier point in the development cycle
- Shift left testing is designed to be an improved model for shift left (fast lane) development because traditional testing models that wait until later in the development cycle can bottleneck development
- With this approach you get quality on code sooner on the development cycle
- Practices: TDD & BDD, Automated Unit Test, CI/CD Integration, Telemetry

Shift-Right Testing

- There's only one environment really similar with Production… and it's Production!
- A common DevOps practice is to not only perform testing early in the development cycle and more often application behavior is also tested in production ("shift right" testing).
- Application stability and resiliency can be improved by performing this "shift right" testing, combined with using new capabilities to monitor applications when under stress and/or unexpected (new) elements are introduced.
- Shift-right testing increases customer feedback, drives Hypothesis-Driven Testing and helps to achieve High Test Coverage.
- Practices: Release Rings, Feature Flags, Hypothesis-Driven Testing, Fault Injection, Insights gained from Telemetry

Shift-Left & Shift-Right Testing





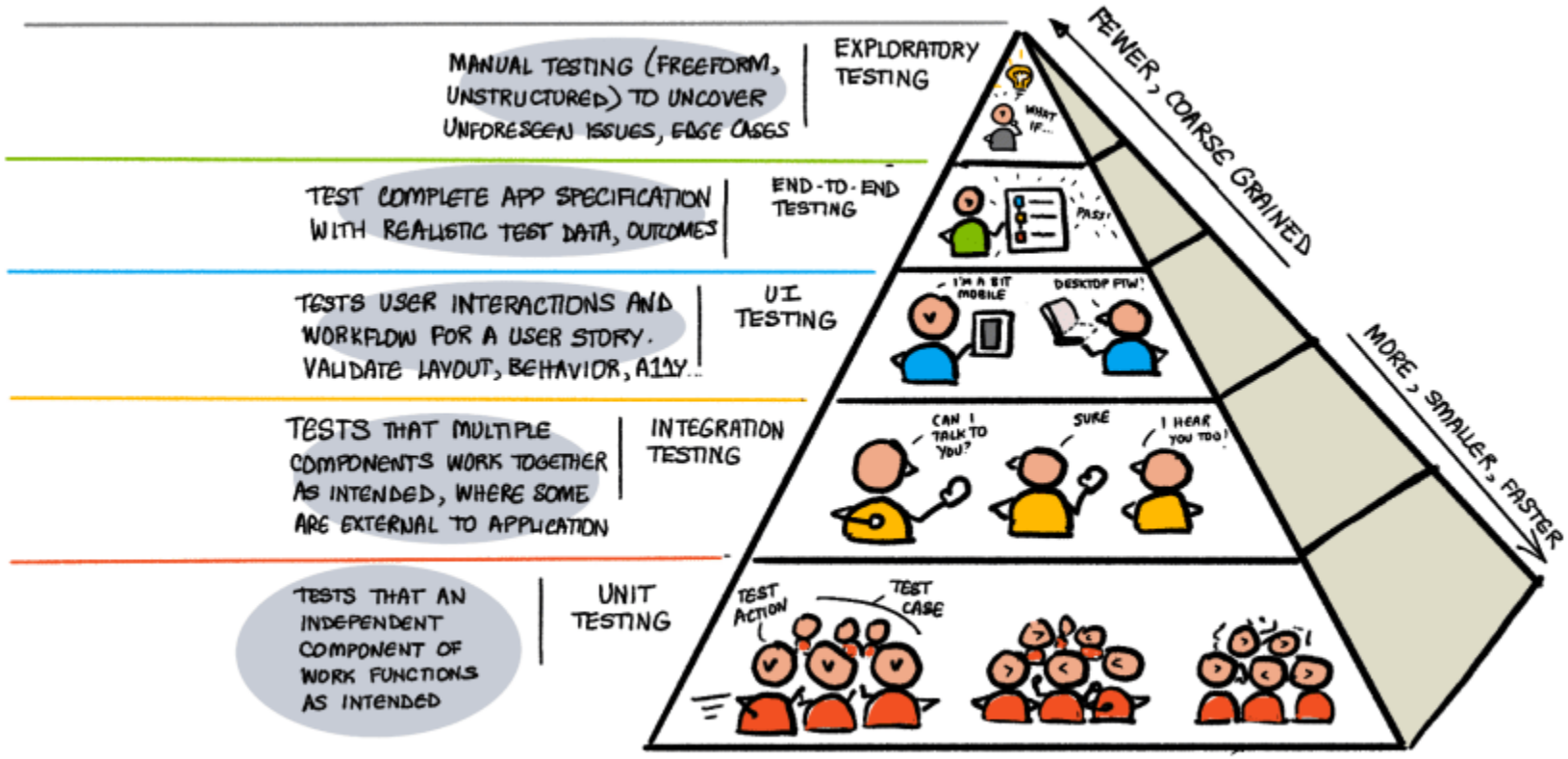
Test Automation

- Test automation is a core activity in DevOps
- Key aspect in driving the goal of faster delivery of value to customers with higher quality
- Automation should consider both Functional and Non-functional testing covering as many layers as possible of the application/system to be tested
- Different tests provide validation of different aspects of the system and contribute to provide a complete quality assessment of what is being delivered
- Examples: Unit, Load, Integration, Regression, Functional/UI, Smoke, …

Automated vs. Manual Testing

Automated Testing	Test procedures that can be run in a consistent and repeated manner Increased ROI, test can be run at anytime Add new tests sustainably in a growing regression suite Self-documented test steps
Manual Testing	Humans are inherently bad at running the same set of steps reliably repeatedly Humans are less efficient, more error prone and may produce inconsistent outcome Humans can provide much more value in less procedural tasks like exploratory testing and bring innovation to quality assurance phase

Test Pyramid



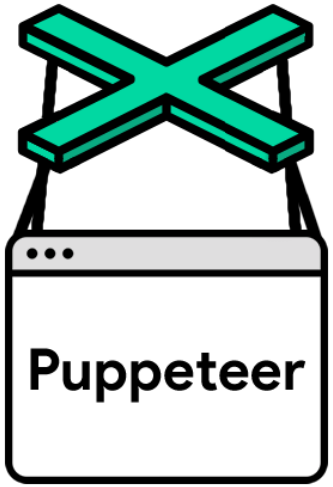
Benefits from E2E Testing Automation

- Reduce risk, providing more comprehensive testing coverage
- Enable faster and more reliable execution
- Facilitate greater test coverage, supporting the execution of test scripts across all popular browsers and operating systems
- Enable regression testing of ever-changing applications and environments
- Deliver higher test accuracy and find more defects earlier
- Provide formalized and documented processes
- Facilitate the re-use of tests

Tooling

Integration Testing

UI/E2E Testing Frameworks



Selenium still industry leader, but…

- Authoring tests is difficult
- Tests are “flaky” with unreliable results
- Tests are slow and frequent execution is expensive
- Failing tests starts to be not maintained and disables
- Lack of visibility to debug test failures
- Flaky test failures are difficult to reproduce

Playwright

Integration Testing

Playwright

- Any browser. Any platform. One API
- **Cross-browser**. Playwright supports all modern rendering engines including Chromium, WebKit, and Firefox
- **Cross-platform**. Test on Windows, Linux, and macOS, locally or on CI, headless or headed.
- **Cross-language**. Use the Playwright API in TypeScript, JavaScript, Python, .NET, Java.
- **Test Mobile Web**. Native mobile emulation of Google Chrome for Android and Mobile Safari. The same rendering engine works on your Desktop and in the Cloud.

Playwright: Resilience, no flaky tests

- Auto-wait. Playwright waits for elements to be actionable prior to performing actions. It also has a rich set of introspection events. The combination of the two eliminates the need for artificial timeouts - the primary cause of flaky tests.
- Web-first assertions. Playwright assertions are created specifically for the dynamic web. Checks are automatically retried until the necessary conditions are met.
- Tracing. Configure test retry strategy, capture execution trace, videos, screenshots to eliminate flakes.

Playwright: Powerful tooling

- Codegen. Generate tests by recording your actions. Save them into any language.
- Playwright inspector. Inspect page, generate selectors, step through the test execution, see click points, explore execution logs.
- Trace Viewer. Capture all the information to investigate the test failure. Playwright trace contains test execution screencast, live DOM snapshots, action explorer, test source, and many more.

Using Feature Flags

Demo