

Fitting More Quality Into Each Build



“Quality is more important than quantity, and in the end, it's a better financial decision anyway. One home run is much better than two doubles.”

— Steve Jobs, BusinessWeek, 2005



Software teams aspire to work fast, adapt to their market's needs, and have a sense of pride in their work. Defects can crush those aspirations, so they need to be addressed as early as possible. Testing as part of every build improves quality and velocity by reducing re-work through fast feedback on code changes, but there's far more to it than automated testing.

Adaptability through Early Validation



Everyone wants to accelerate, but not everyone stops to ask why. Software teams that move fast provide their organizations with adaptability to changes in their market and agility in business decisions. This doesn't just apply to small and mid-size software teams; even enterprises such as Microsoft, Google, and Amazon have adopted this philosophy¹. Teams that move fast beat the competition more often than those who are slow to change.

Defects get in the way of forward progress. They force us to focus on fixing and spend less time on new, innovative work. Development teams measure their velocity in terms of story points; the less of these points they deliver, the slower they become. Fixing defects, a common type of re-work, is often the result of a lack of quality from a previous iteration.

The more of these defects get perpetuated to future work, the less time there is to do work that benefits users – a downward spiral that no one aspires to.

A solution for this downward spiral of defects comes by example from top performing teams, those that release often and experience far less lead time than teams who leave defects to late-cycle testing phases². Their secret weapon: early validation. These teams carefully manage batch size and work-in-progress limits, adopting a culture of quality through early validation and addressing release pain up front.

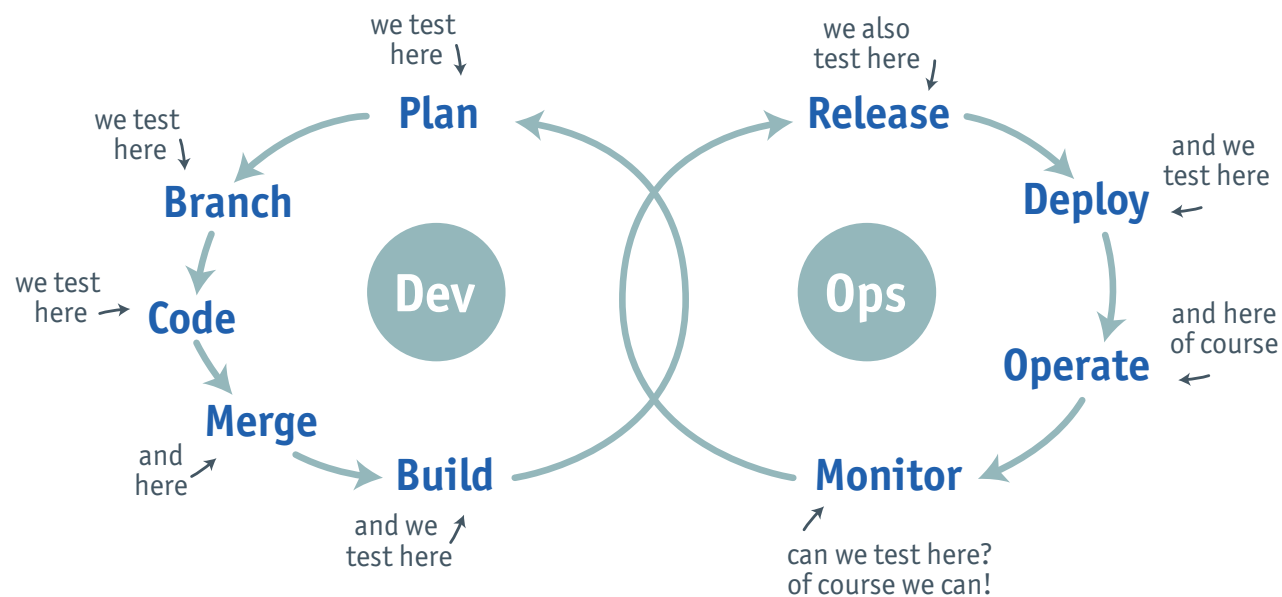
The State of DevOps 2016 report by Puppet Labs highlights comprehensive, fast and reliable test automation as one of the leading factors to less re-work.

¹ <https://arstechnica.com/information-technology/2014/08/how-microsoft-dragged-its-development-practices-into-the-21st-century/>

² <https://puppet.com/resources/whitepaper/2016-state-of-devops-report>

62% development teams cite “accelerate product delivery” as the number one reason for adopting agile practices – VersionOne State of Agile 2016

Depiction of testing at every phase of the software delivery lifecycle,
Dan Ashby, 2016³



In continuous delivery, release cycles start at the moment where code is checked in. Any parts of this process that are not automated slow the whole process down. If we want early validation at each of these steps, we need to automate testing across the board.

³ <https://www.linkedin.com/pulse/continuous-testing-devops-dan-ashby>

Prerequisites to Achieving High Quality and Velocity in Every Build

Stable Tests

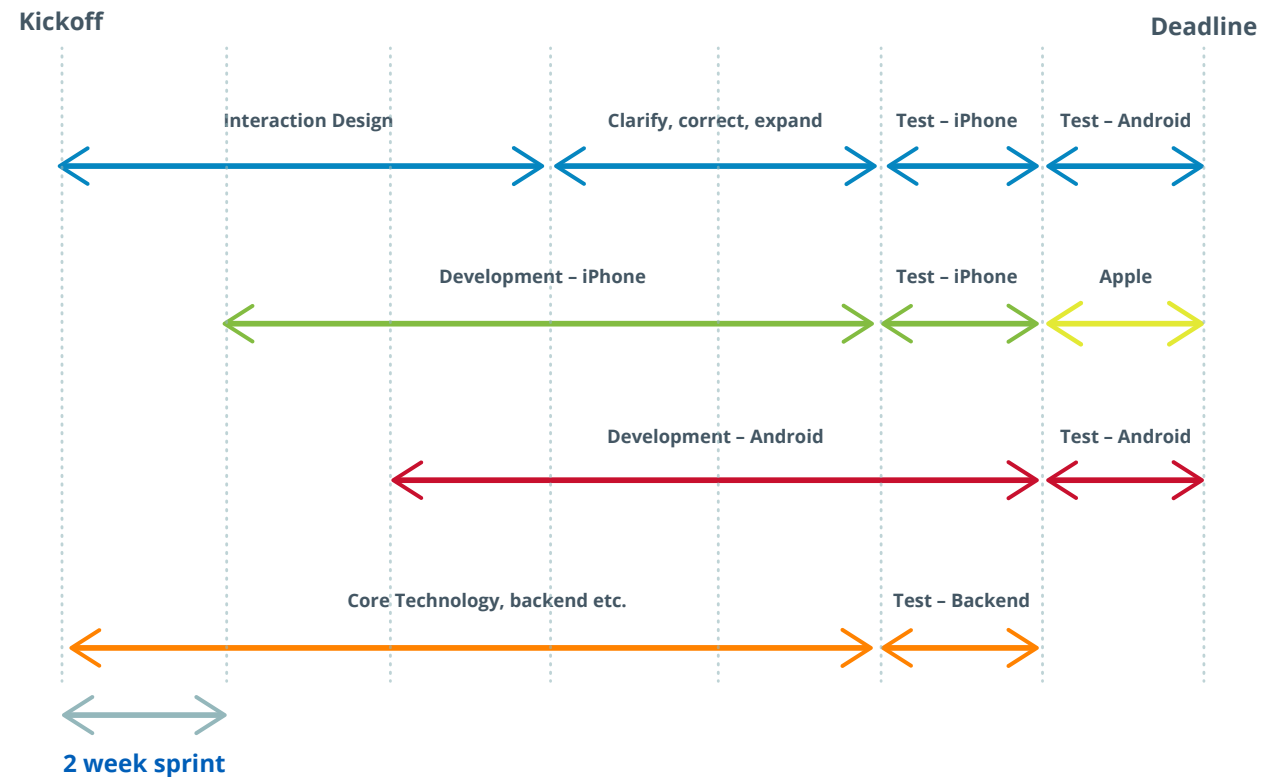
Early validation is useless if it is not correct and consistent. Flaky tests waste our time and undermine confidence in automated processes. A stable test is worth its weight in gold; only stable tests actually deliver fast feedback and improve team velocity. For more information on this topic, see “Eliminating Flaky UI Tests to Stabilize Continuous Delivery” by Perfecto⁴.

Reliable Infrastructure

Once stable tests and correct data are prepared, execution of tests requires resources to run on, typically in a lab or simulated environment. Running tests across various devices, platforms, and endpoints typically requires an orchestration layer to maintain these processes and results.


An example of this is how Facebook engineering teams developed a rack based modular device lab design⁵. Though it took many years, hundreds of iterations, and millions of dollars to develop, the particular needs of the Facebook app teams demanded that level of internal engineering

Time Planning



⁴ <http://blog.perfectomobile.com/mobile-application-testing/eliminating-flaky-ui-tests-to-stabilize-continuous-delivery/>

⁵ <https://code.facebook.com/posts/300815046928882/the-mobile-device-lab-at-the-prineville-data-center/>



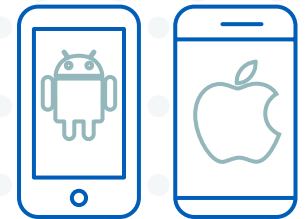
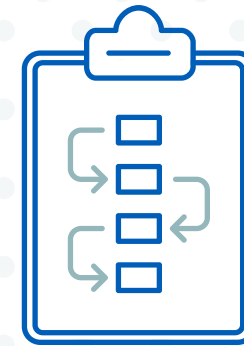
Many organizations don't have this luxury, and instead use a mixture of virtualized and physical hardware to match their needs for timely, scalable, and affordable test execution. In many cases, management over these lab resources is beyond the scope of software teams who chose to focus their efforts on app development

Testing aligned to development cadence

Once these prerequisite conditions are met, teams can begin to align their activities to achieve early validation throughout the software delivery lifecycle. However, any activity to improve the development process needs to meet the pace of the development team.

An example of this comes from Trifork, as presented by Janne Jul Jensen at Goto Conference in 2014⁶. Her depiction of how user experience design must fit into an agile release train highlights the importance of process alignment when integrating improvements into development cycles.

Unlike the example above, responsive web app teams have the luxury of releasing without requiring user interaction to accept the update. While mobile apps can achieve continuous delivery, web apps controlled completely on the server side can implement full continuous deployment intrinsically. However, this model requires even closer attention to detail in automated testing since the 'always green' state infers immediate release to production.



⁶ <https://www.youtube.com/watch?v=NdqtTlCi4cE&feature=youtu.be&t=19m39s>

Transaction Configuration

Environment: UATE Type: IM Subtype: IM85

Transaction Type: C Transaction Desc: ACH deposit Transaction Code: 1007

Description: Tester Effective Date: 3/2/2015 Indiv ID: A

Company Name: capone Check Number: 0 Amount: 10.00

Account Number: 7000133 Bank Region: 81 Count: 25

Transaction Data

Row_ID	Environ.	Account	AccountNumber	Bank	Transactio.	Transacti.	Amount	EffectiveDate	SubT.	Check Number	De
0	UATE	IM	7000133	81	1007	C	10.00	3/2/2015 3:28:23 PM	IM85	0	Te
1	UATE	IM	7000133	81	1007	C	10.00	3/2/2015 3:28:23 PM	IM85	0	Te
2	UATE	IM	7000133	81	1007	C	10.00	3/2/2015 3:28:23 PM	IM85	0	Te
3	UATE	IM	7000133	81	1007	C	10.00	3/2/2015 3:28:23 PM	IM85	0	Te
4	UATE	IM	7000133	81	1007	C	10.00	3/2/2015 3:28:23 PM	IM85	0	Te
5	UATE	IM	7000133	81	1007	C	10.00	3/2/2015 3:28:23 PM	IM85	0	Te
6	UATE	IM	7000133	81	1007	C	10.00	3/2/2015 3:28:23 PM	IM85	0	Te
7	UATE	IM	7000133	81	1007	C	10.00	3/2/2015 3:28:23 PM	IM85	0	Te
8	UATE	IM	7000133	81	1007	C	10.00	3/2/2015 3:28:23 PM	IM85	0	Te
9	UATE	IM	7000133	81	1007	C	10.00	3/2/2015 3:28:23 PM	IM85	0	Te

OneSource, an Experimental Platform for Self-service Test Data Management at CapitalOne⁷

⁷ <http://www.slideshare.net/AgileCampSV/adam-auerbach-presentation/28>

⁸ <https://www.youtube.com/watch?v=3NsZc1DWTdA&t=4m30s>

Test Data Availability

Additionally, many tests require data to run properly. Test data is often constructed to execute code and UI under various permutations, representing boundary and negative testing conditions where defects are often overlooked. One of the biggest time consumers for software teams is the acquisition of proper test data, closely followed by maintaining the data as systems change.

Forward-thinking teams at Capital One use a mix of existing solutions and homegrown tools to overcome the challenge of delivering test data for automated testing in continuous delivery⁸.

Results Retention Policy

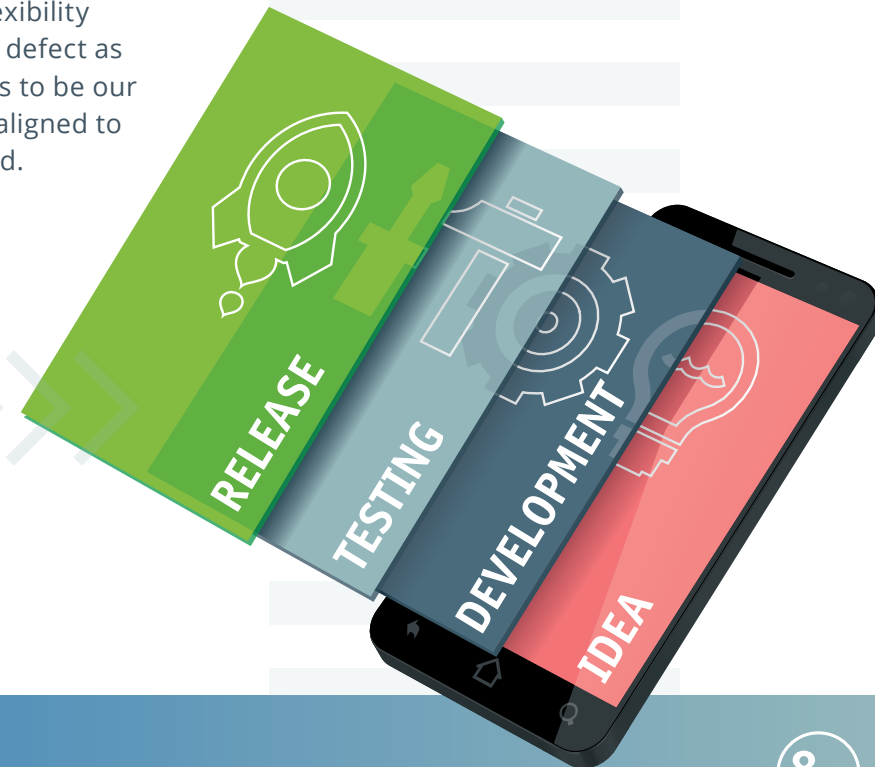
The combined footprint of tests, data, and execution requires organizations to plan for additional storage and processing capacity. Depending on an organization's test result retention policies, this can amount to sizable maintenance and financial cost. Many organizations struggle to manage this cost internally though there are reasonable alternatives in the market today.

Fitting More Testing in Every Build

Change is a constant. The software and hardware ecosystems we build change all the time, and we have to respond quickly or risk downtime. When we make changes to code, the result is often breakage. Tests stop working, errors begin to pop up in strange places, and the time it takes to diagnose these issues steals our time from new work. So while the effect of our changes may be breakage, the cost is team velocity.

An exception to this dynamic is when testing and development share the same language, the same artifacts, and the same alignment to goals. Examples of this in effect are the Espresso and XCTest frameworks, both designed to dramatically simplify native mobile testing on Android and iOS respectively. Coupling development and testing activities in this way improves our visibility into the effects of our changes at the code level, but also simplifies and stabilizes our efforts⁹.

On a good day, we want to feel confident about our changes. We want to be in a state of “always green”, builds that provably work as expected under the most likely conditions that users will face in the real world. We want to provide our business decision makers the flexibility to release software that is as free of defect as possible at any point in time. If this is to be our ready state, comprehensive testing aligned to business goals must be in every build.



⁹ <https://testing.googleblog.com/2016/05/flaky-tests-at-google-and-how-we.html>

What does “more” mean?

Unless you have very low code coverage, simply adding more tests will not always improve the quality of your work. The number of tests and permutations should correlate to the complexity of your app and your goals over code coverage. “Don’t be a slave to your tests,” says Chris Williams, Principal Engineer at Audioium UK. “Have your core engineering team lead test automation to ensure you make educated, value-based decisions about your test suite¹⁰”.

Beyond considerations over test suite size, it is also important to understand how your users actually use your apps. Tests that replicate key workflows can be effectively used as smoke and sanity checks in build verification testing to provide early feedback about unanticipated breakage from day-to-day code changes.

More also means *more platforms*. Our apps now run on a vast landscape of different devices, browsers, and under conditions we can’t possibly fathom. The one or two devices you have plugged in to your laptop gives a sense for how your code will run, but doesn’t come close to addressing the true reality of our connected world. Once we realize this, it seems far more natural to want to bring more of this reality into build verification testing on every build in continuous integration.

Early warning signals that a recent code change broke a key workflow in the app on one out of

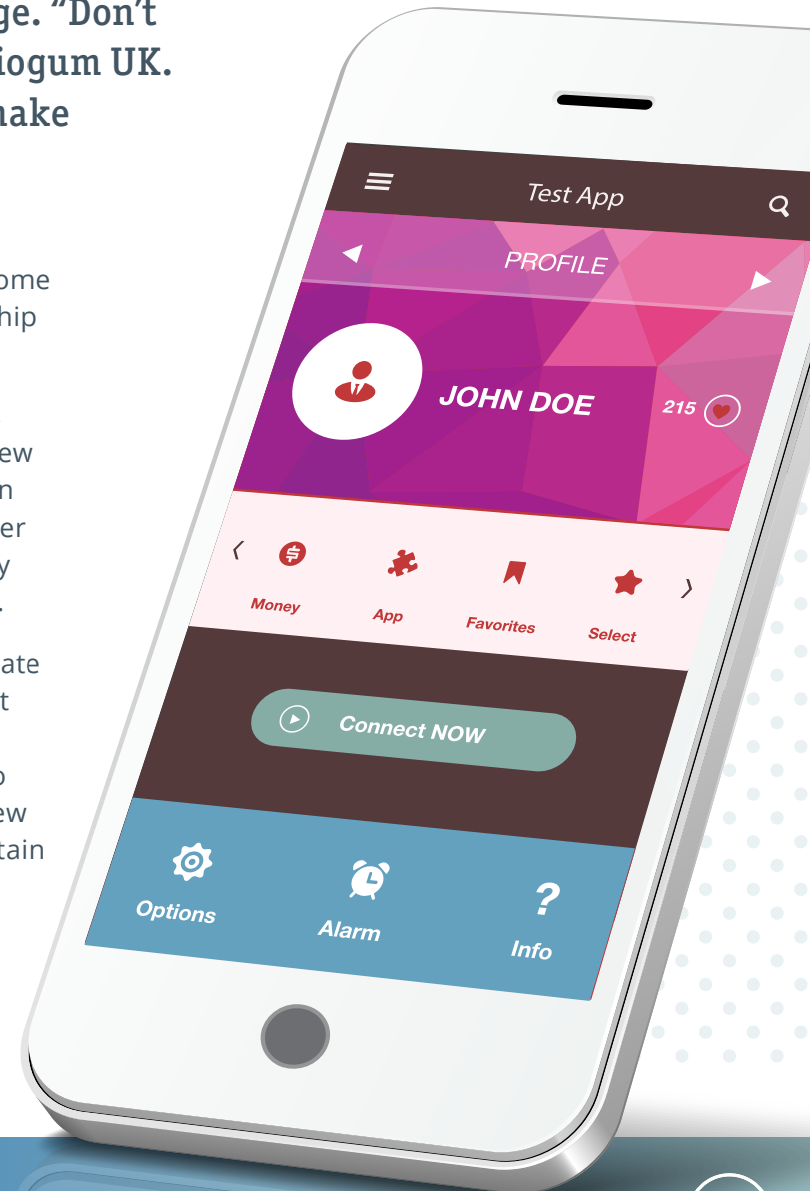
8 major platforms is priceless information. Left unchecked, those kinds of defects become re-work later on and damage our relationship with users even after we fix them.

A recent Facebook app update¹¹ exemplifies the difficulties faced when testing misses a few key platforms. Though there may have been tests that expose this kind of failure on other platforms, a gap in platform coverage easily results in bad app reviews and upset users.

“In addition to the latest Facebook app update causing crashes to iPhones, another recent software problem that has been flagged is the update to iOS 10.1. While the update to Apple’s mobile operating system added new features and improvements, there are certain users who have experienced drops in the performance of their iPhone or iPad upon moving from iOS 10 to iOS 10.1.”

¹⁰ <https://crswlls.wordpress.com/2016/08/22/5-common-pitfalls-of-ui-test-automation/>

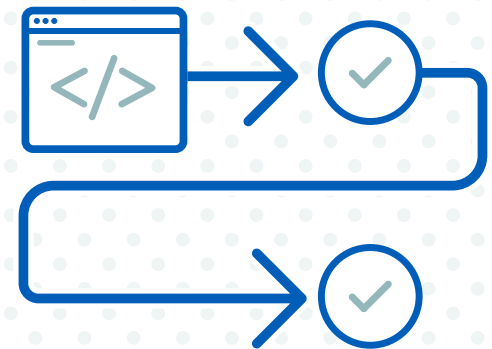
¹¹ <http://www.techtimes.com/articles/184256/20161030/iphone-users-beware-latest-facebook-app-update-causing-crashes-and-heres-how-to-fix-it.htm>



More also means *more often*. If the goal is actionable feedback as early as possible, then automated testing must adapt to an execution pattern that satisfies the cadence of development activities. An example of scheduling various levels and types of automated testing to provide development teams early feedback throughout their daily cycles is provided below:

Job	Total CI time budget	Test Time (avg.)	Scope
Commit (VCS)	15–30 mins	<100ms	Unit, API, Integration
Hourly	20–40 mins	Varies	New Test, Key Smoke (BVT)
Multiple/day	30–60 mins	<2s	All Smoke, Key Functional, Some Data
Nightly	2–7 hrs	<120s	All Functional, All Data
Weekend	10–48 hrs	all	All Tests incl. Performance

To facilitate this adaptation, the structure of test suites may need to change or be annotated to identify their relevancy and execution limits. This enables tests to be automatically executed after events such as after check-in to version control (i.e. build verification), scheduled moments throughout the day such as lunch and nightly regression, and as a result of other release engineering tasks like pipeline orchestration jobs.

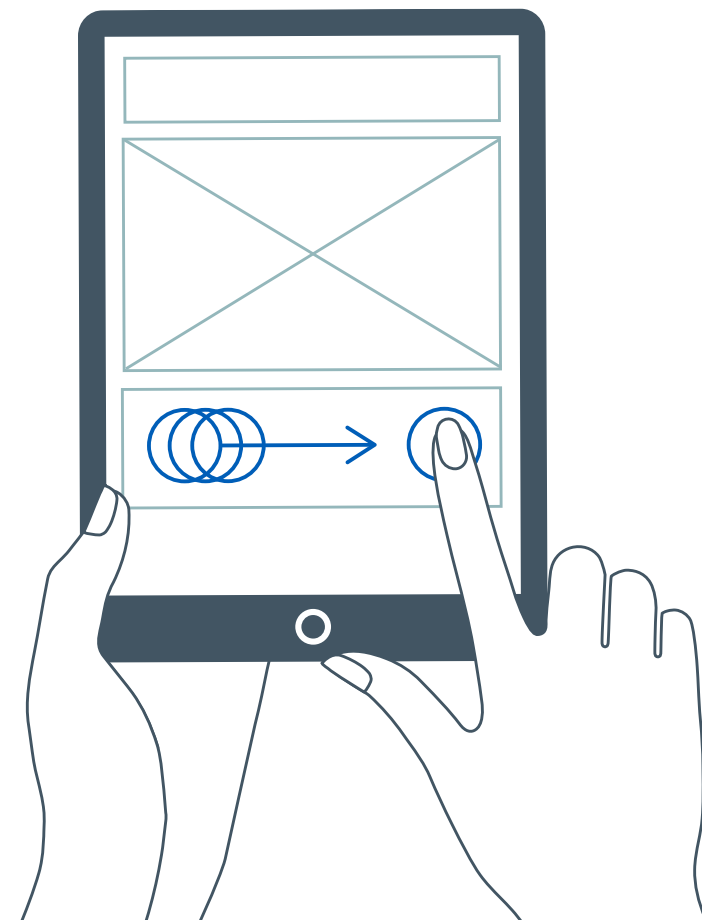


Where do your tests come from?

In many cases we see that developers write unit tests, but participate less in UI testing. The most common reason for this is that developers are not properly incentivized to do so, even though they are the ultimate recipients of the responsibility to resolve defects that escape due to a lack of adequate UI testing.

In his keynote at Velocity 2016, David Hayes of PagerDuty states, “Production is where you make money. What makes you money is new features which look a lot like code.”¹² This statement highlights the problem: developers want to build, not verify. But in order to maintain or improve velocity, teams need to constantly verify that defects are kept to a minimum. Therefore, testing activities must be counted against work-in-progress limits for whoever performs them, otherwise flow will quickly be lost¹³.

The important thing to understand is that it matters less who writes the tests and more that the quality of the delivered product can be verified as part of every build. Some teams choose to have separate test engineers or developers who focus on writing tests; other teams decide to make exploratory testing an activity developers regularly participate in. What matters is that new code is covered by new tests to the satisfaction of the team, with a goal of delighting the user with as defect-free an experience as possible.

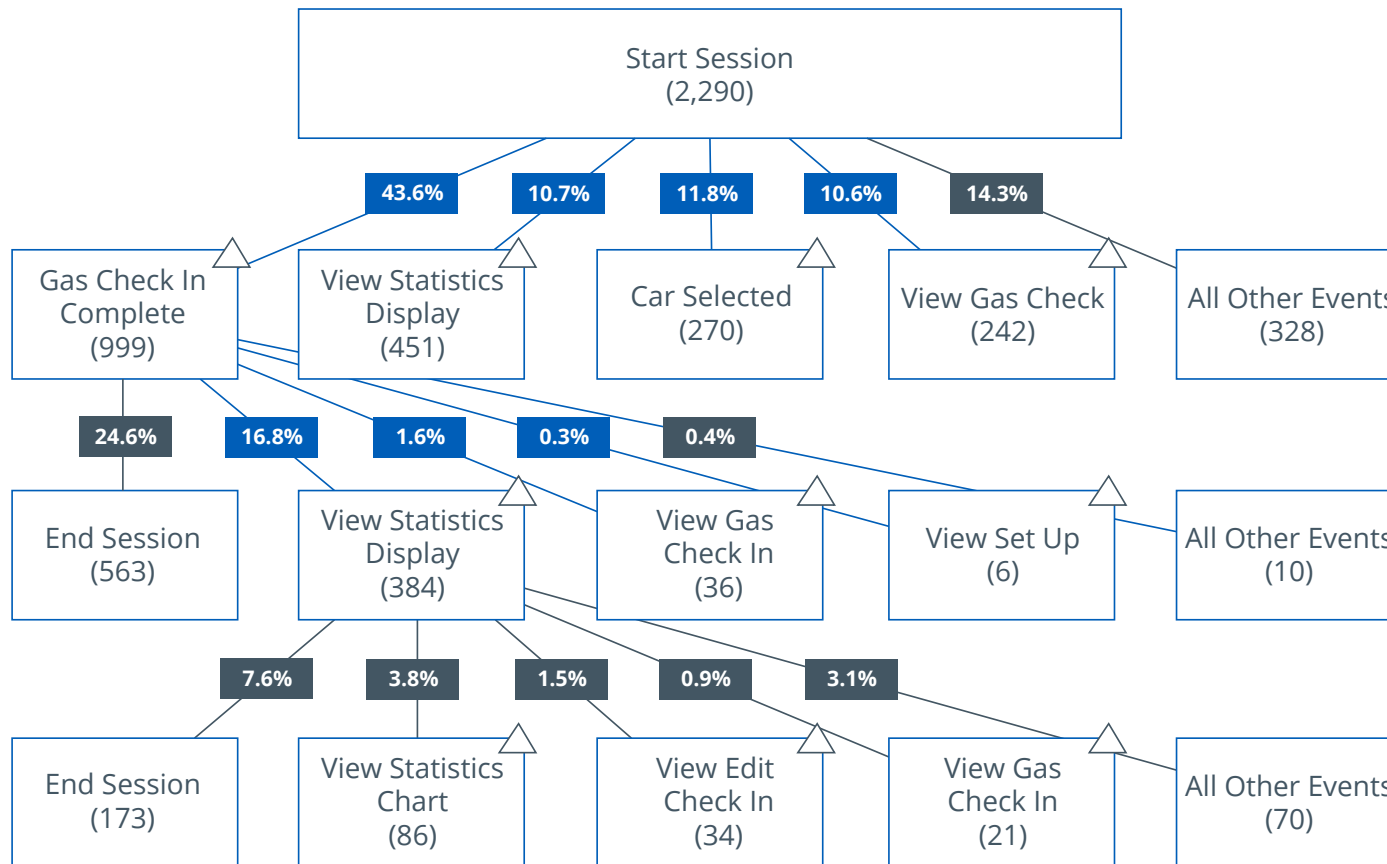


¹² <https://www.oreilly.com/ideas/why-devops-is-more-than-just-automation>

¹³ <https://leankit.com/blog/2014/03/limiting-work-in-progress/>

How should tests be prioritized?

Every test should have value, otherwise you should cut it from the heap. The most tangible form of value to the business is in terms of the customer experience. Just as teams incorporate user behavior into how they optimize flow in their apps, testing should also use analytics flow diagrams as input for which tests to create, change, and retire. An example flow diagram is below:



Source: Apptamin, App Analytics Tools Round-up

With the right set of analytics in your app, you can quickly identify workflows and platforms that are most used; a good indicator of where to focus your testing efforts. If you don't already know which platforms your app is running, your marketing team likely does. Have a conversation with them about which mobile devices and browsers most accurately represent your user base, then prioritize tests that cover those conditions.

Likewise, new features don't have analytical data yet, but need to be prioritized to ensure the success of that new work. Teams should consider the impact of new planned work on existing improvements to test coverage to reduce technical debt, then reprioritize accordingly with product management. As you iterate on features, make sure that changes to original expectations are perpetuated in tests created in the initial stages of feature development.

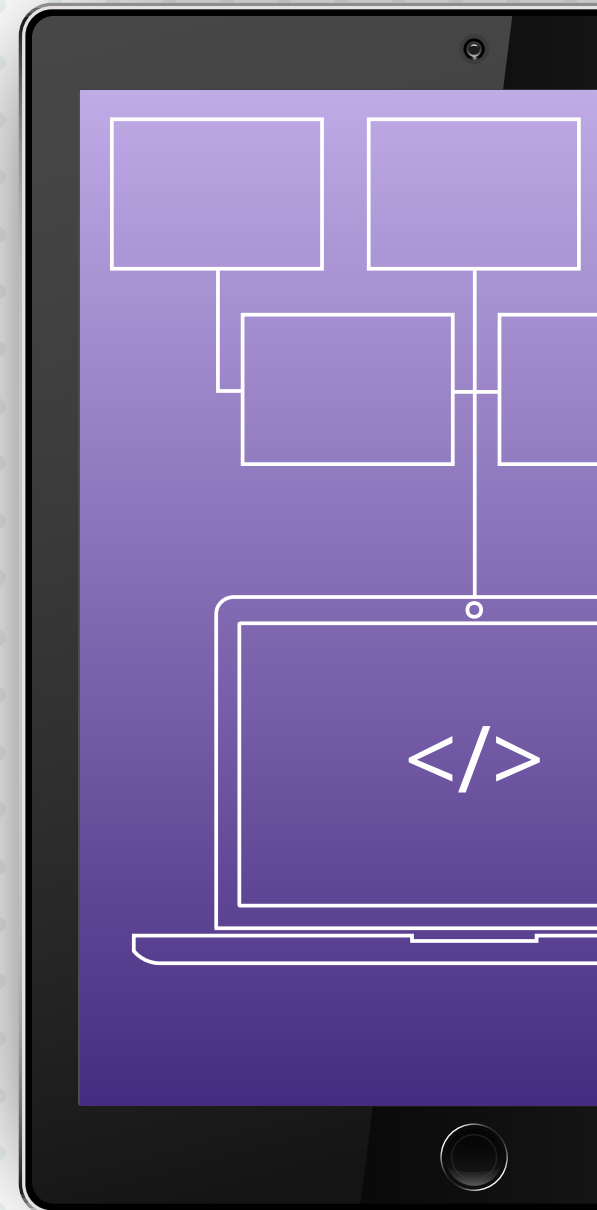
Using a strategy for executing tests based on their importance to business goals can help to quickly run the right tests after changes are made. Many teams have adopted the test management practice of annotating (or tagging) tests based on various affinities.

Here are a few ways you might tag tests:

- Purpose in the pipeline: smoke, regression
- Execution speed: Small / Medium / Large (example: Espresso test sizes¹⁴)
- Nature of the test: positive, negative
- Conditions: interruption, latency, low-battery
- By module/screen: home, settings, etc.
- By workflow: login, logout, timeout, check-balance
- By problem areas: long-list, video-overlay, rotation

With additional metadata, tests can be grouped for execution based on the work being done. Test jobs in continuous integration can then be kicked off at will or based on specific code changes, and their execution lengths monitored for optimization over time.

By employing an automated testing strategy that: incorporates user behavior data, fits to the cadence of the development process, and includes the appropriate platform coverage, we can make tangible progress on improving both app quality and team velocity at the same time.



¹⁴ <https://testing.googleblog.com/2010/12/test-sizes.html>

What We Get With “More” Testing in Every Build

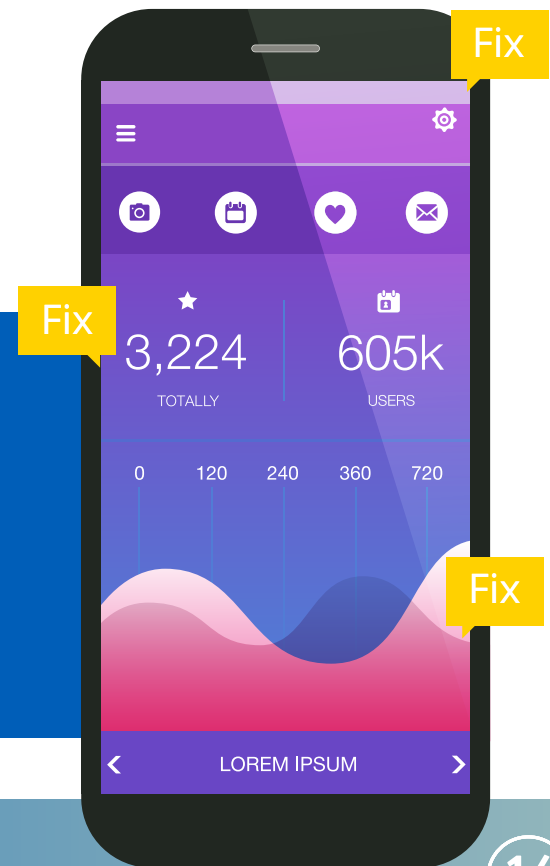
Revisiting the original goal of more innovation through adaptability, more testing helps us reduce defects and re-work. With more time for new work, we're able to improve our team's velocity and the quality of our user experience. The faster that development teams can respond to business needs and market changes, the faster we learn, adapt, and innovate.

More testing in every build doesn't just improve the quality of our apps; instilling fast feedback in development cycles makes sure that our work today doesn't negatively impact our work

tomorrow. Addressing defects as early as possible helps us maintain a healthy work-in-progress limit, improving our predictability and reducing the long-term cost of carrying defects between releases.

With the proper test coverage in every build, development teams can ship their products with greater confidence on time. It all starts with owning the results of our work, being willing to accept responsibility for how our apps will work for people in the real world, and continuously improving on our successes.

“Bugs can kill the speed of your delivery team. If you have a delivery schedule and you're fixing issues from a previous delivery, you will have a hard time hitting your date. One of the most important ways to reduce faulty work is to limit your work-in-progress. When you focus on getting things right, the surprise is that you can move faster.” — Matt Bernier, Product Manager, SendGrid





Paul Bruce

currently works with the Perfecto team as Developer Advocate, focusing on helping development groups improve their velocity and software quality throughout the delivery process. With more than 15 years of work as a full-stack developer over front-end apps and back-end solutions in non-profit, enterprise, startups, and small business IT, Paul now uses his experience to help other developers to save time, build cool and useful things, and encourages open source contribution wherever possible. Paul writes, speaks, listens, and teaches about software delivery practices in key industries worldwide and regularly publishes his research to blogs and engages online communities. You can read more on his blog at paulsbruce.io

About Perfecto

Perfecto enables exceptional digital experiences. We help you transform your business and strengthen every digital interaction with a quality-first approach to creating web and native apps, through a cloud-based test environment called the **Continuous Quality Lab™**. The CQ Lab is comprised of real devices and real end-user conditions, giving you the truest test environment available.

More than 1,500 customers, including 50% of the Fortune 500 across the banking, insurance, retail, telecommunications and media industries rely on Perfecto to deliver optimal mobile app functionality and end user experiences, ensuring their brand's reputation, establishing loyal customers, and continually attracting new users. For more information about Perfecto, visit www.perfectomobile.com, join our community follow us on Twitter at [@PerfectoMobile](https://twitter.com/PerfectoMobile).

Get content just like this delivered to your inbox!

