THE PATH TO RELEASING CONFIDENTLY IN DEVOPS

Perfecto

IT and development leaders face competitive pressure to accelerate, but it is often unclear how to do so without compromising the quality of the user experience. This report details specific processes that high-performing developers use to reclaim time and focus efforts on web and mobile user experiences.

Key Findings

1. High-performance teams define and test the user experience early

Developers who deliver value to production users also tend to embed quality into all stages of the software delivery lifecycle. High-performance teams are more than twice as likely to bake non-functional criteria into user stories during early planning phases. They also cover more features with unit and UI testing, spending twice as much time testing than lower performers.

2. High velocity software delivery is a game of inches

2-4 hours per developer per week can quickly translate into more work on new features, reducing technical debt, and process improvement. High-performance teams spend 5-10% less time on overhead tasks (admin, waste, other), but 5% more time writing tests. Overhead must be carefully managed; inefficiencies add up quickly. Developers have little time for activities that do not contribute to team velocity. The efficiency of high-performing developers can be glimpsed in how they reclaim and spend a few hours each week.
3. Delivering frequently requires confidence in app quality
Early and frequent feedback on how code changes affect user experience enables developers to feel confident and move quickly, resulting in a more reliable release process. High-performance teams release at least every few days, producing same or better quality apps than those who release once every quarter. The speed of continuous delivery requires mature teams to embed fast feedback loops across the development lifecycle, including automated testing triggered after builds.

Perfecto

00: Executive Summary

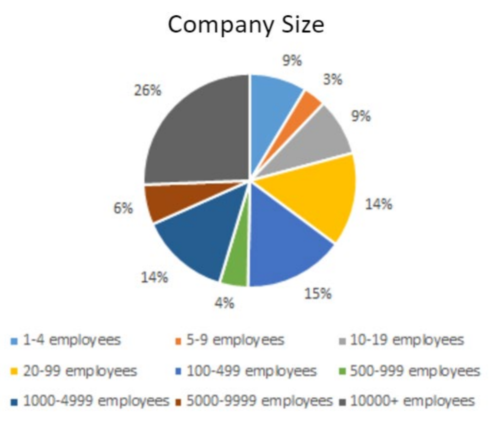
The following report highlights key practices that enable enterprise web and mobile app teams to achieve high velocity software development efficiently.

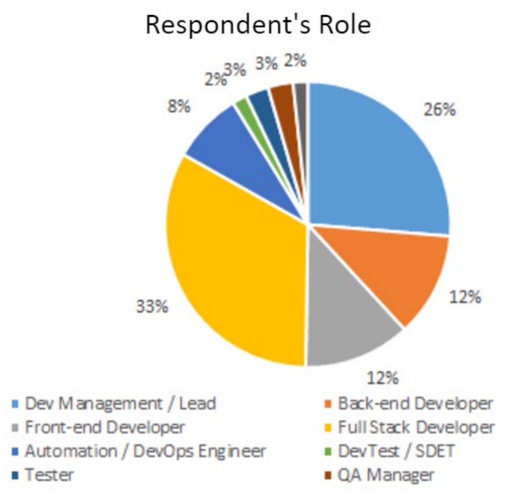
Developer efficiency is critical to releasing apps confidently and reliably. The focus of this research was to correlate practices which impact on release frequency and to identify factors of success in high-performing app development teams. We asked developers a variety of questions about how they focus their resources and approach their testing strategies.

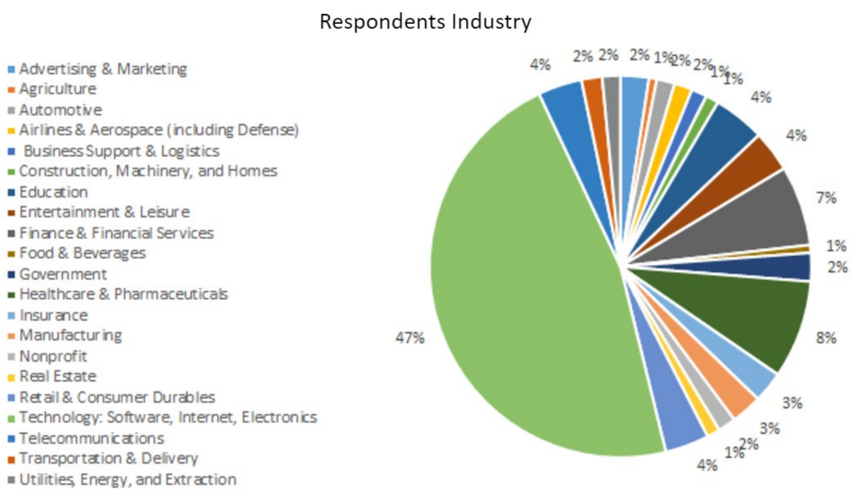
Process is the new competitive advantage. The capability to efficiently deliver new and tailored experiences before your competition will enable you to win and maintain more customers. Unlike technological breakthroughs, process improvements can be much more incremental, with earlier improvements underpinning new ones, leading to compounded results. Optimizing processes is often a game of inches. Inches turn into yards - so improvement should be a constant focus. For this reason, Smith's Point Analytics and Perfecto Mobile have collaborated to examine practices, attitudes and behaviors demonstrated by top developers.

01: Methodology & Audience

The study, consisting of 478 respondents, was conducted over a six-week period from February 6th to March 20th 2017. Respondents were recruited from Perfecto Mobile's community of development professionals and Smith's Point Analytics' developer panel. Respondents' job function, chosen industry, and company size varied widely, as shown in the graphs below.







02: Defining High-Performance Teams

Releasing software is the mechanism by which development teams deliver value to users. Release frequency is a key indicator of the performance of development teams because it allows for more opportunity to hypothesize, act, and learn how to better satisfy users. The more chances and flexibility development teams provide their organization to exercise this learning loop, the faster they can fit solutions to the needs of the market. Release frequency differs across development teams. The study revealed that the frequency of software delivery clustered around three means or groups.

These three groups can be defined as rapid-delivery teams, middle-tier-delivery teams and slow-delivery teams. Below we define and characterize these groups.

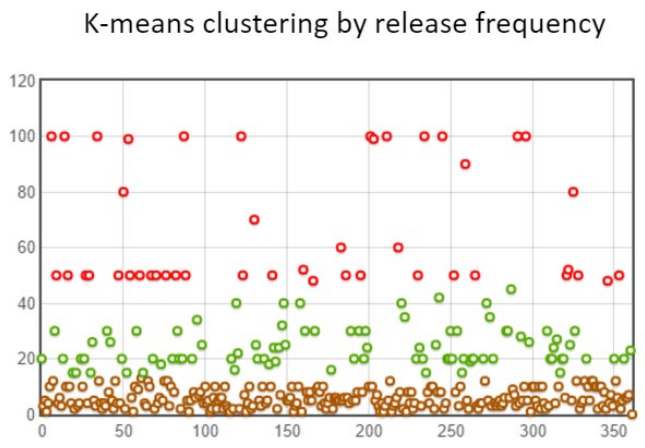


Figure 1 – Defining performance cohorts

	Rapid Delivery (High Performance)	Middle-Tier Delivery Pace	Slow Delivery (Low Performance)
Minimum times software was delivered to production in 2016	40	12	<=1
Average times software was delivered to production in 2016	**64.6**	21.3	**5.2**
Average number of times code is checked in, per team per day	**6.9**	4.7	**3.7**
Number of respondents	53	97	215
Average team size (Includes developers, testers, ops, QA and team leaders	15.6	15.2	10.7

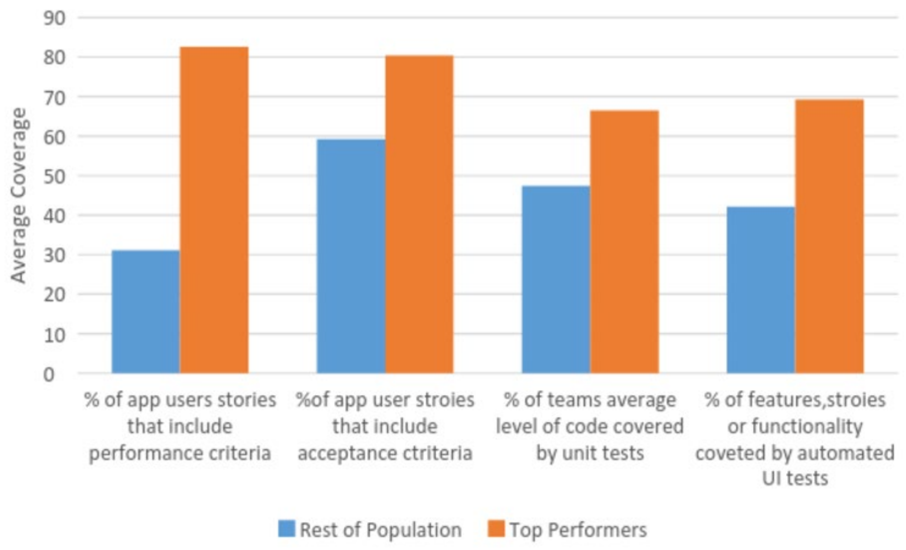
The difference in release frequency between fast teams and slow teams is extreme. High-performance teams release software to production 2-3 times per week. Low performers tend to release in large batches, closer to once per quarter.

Many factors combine to create a culture of quick release. Fast teams utilize high levels of automation to deliver high quality code to production. Testing and review cycles must be short and cannot be left to lengthy, large batches of work; embedding these practices into a continuous process requires automation to minimize redundant work.

03: Embedding Quality with Early Coverage

When we examine team performance as not only a function of release frequency but also the quality of the code produced, differences in both criteria coverage and testing coverage emerge.
In figure 2 we categorized top-performing app teams as ones that not only deliver code more often than ¾ of the rest of the population but also ship hot fixes, rarely, very rarely or never. This group included 17% of the survey respondents and delivered software at least 20 times a year.

Figure 2 - Criteria and Code Coverage, High Performance Teams vs. General Population



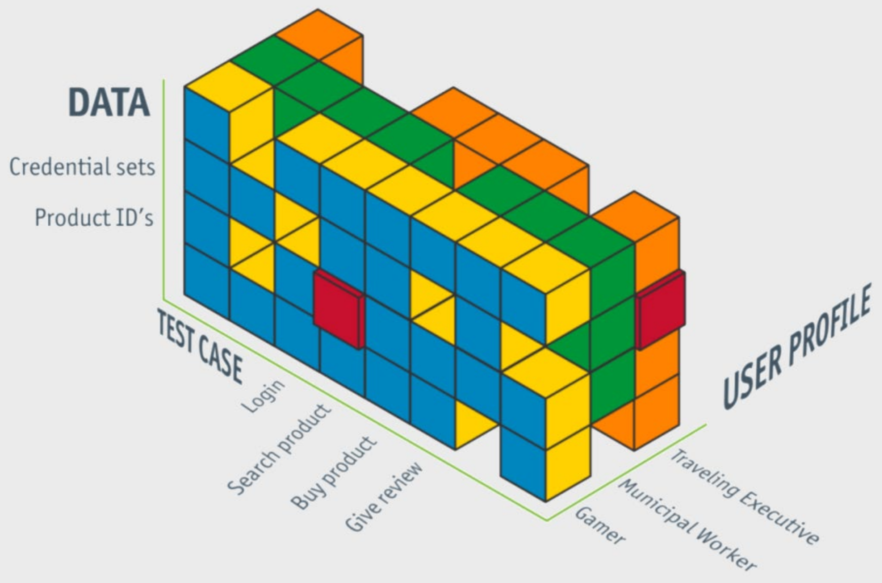
The chart shows that for top performers, on average, 82% of users' stories included performance criteria vs. just 31% of the rest of the survey respondents. Understanding your user's requirements leads to fewer hot fixes to improve performance after apps are launched.

There are also stark differences in the amount of code that high-performance teams cover with their UI test and unit tests. Despite the fact that challenges around testing performance and UI testing, particularly with mobile devices, can be especially daunting, these teams invest in higher test coverage.

What does "optimal" test coverage mean for web and mobile apps?

Just as code coverage considers permutations, lines, and branches, a complete picture of the user experience must also take into account data, usage scenarios, and environmental challenges of the world beyond just code. Optimally, UI testing should verify that an app meets expectations from the perspective of the user and verify the real user experience.

Unless all aspects of the user experience 'cube' is tested, how do you know where the true gaps in your coverage strategy lie?



Source: User-driven Testing is the New Norm, Perfecto Blog

Owning the Problem: Align with the Real User Experience

Greater coverage reduces the likelihood of defects escaping development cycles. With clear well defined criteria established early in the design phase, developers don't have to fret over whether their code is meeting requirement and can focus on delivering code.

The increased focus on non-functional requirements for high-performing teams puts developers closer to the design process and the end users.

High-performing dev teams do not spend any more or less time coding than their peers; rather, they are focused on delivering against changing users requirements. An iterative approach enables business and developers to quickly learn what works in the real world and what needs to be changed. This process will be the foundation of real digital advantages into the future.

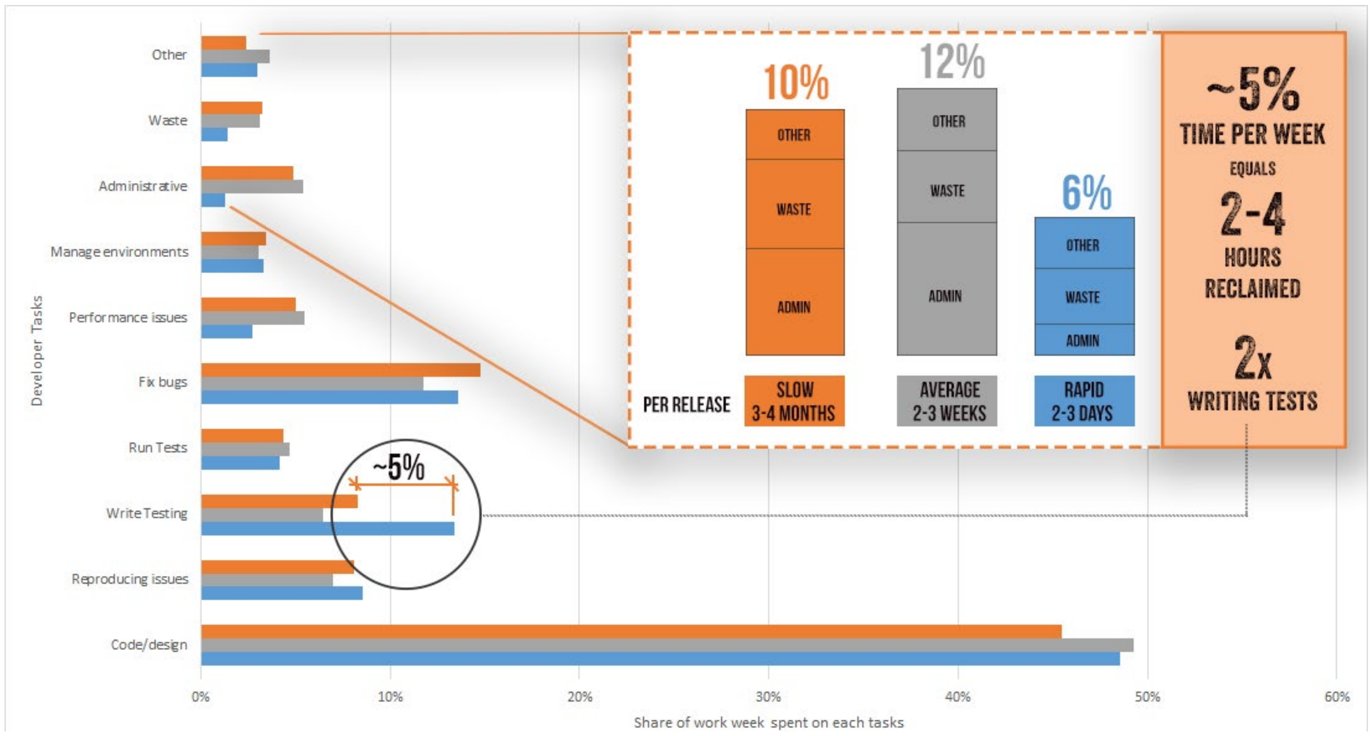
04: Game of Inches - Use Moments Wisely

To better understand what behaviors a top performer exhibits, the survey also focused on how front-end UI developers spend their day. What percentage of their time is spent on various activities?

The layman might assume that developers spend all their time writing code,but the reality is quite different. We found that the average developer spent less than 50% of their time writing code.

What differentiates high-performance teams from lower performers is the time investment on testing, specifically writing tests. High-performance teams are delivering code twice as often as mid-tier teams and are driving higher expectations in quality through greater non-functional criteria coverage. This translates into spending twice as much time writing tests to meet these evolved expectations and results in a better user experience.

Where efficiency drops precipitously is when comparing low performers and mid-tier performers. Both groups are spending simi-lar amounts of time testing but mid-tier teams can deliver code to production more quickly.



The development lifecycle of high-performance teams is more efficient as they spend less time on overhead activities. They spend 3% of their time on admin and waste, mid-tier performers spend 7% and low performers spend 8%. Lower levels of overhead provide the opportunity to focus 4-5% more time on productive activities. Coincidentally, high performance teams spend 2 to 2.5 more hours writing tests than low performers.

Avoiding Wasteful Practices

Agile gurus Mary and Top Poppendieck define the seven primary practices that lead to waste in agile software development as:

- Partially done work
- Extra Features
- Relearning
- Handoffs
- Delays
- Task Switching
- Defects

High-performance teams minimize these events. They deliver quickly on their user requirements and they give quicker feedback. Greater visibility into the software development life cycle also reduces time spent documenting and reporting status, reducing admin time. Less confidence in processes requires greater oversight and corrective action, but justified confidence minimizes wasteful re-work and improves velocity.

"Developers need crisp, complete requirements to work efficiently, but also need to be part of defining them and not just passively receiving them. The more that individual contributors are empowered to own the goal of their work, the more likely they are to find their own efficiencies that help them achieve the goal quickly. This aspect of autonomy plays a key role in enabling developers to meet both time and quality goals, and is kept in check by a culture of quality through encouragement over enforcement."

- Kunal Jain, Product Manager, Circle CI

"Lots of meetings are a time-drain for developers, but consolidating meetings to a core time each day leaves plenty of heads-down time to cut code. This reduces context switching, minimizes distractions, and encourages more focused work. Peer code reviews triggered by pull requests help to ensure the quality of the code before release. Retrospectives also provide the whole team an opportunity to identify gaps and learn along the way."

- Sean Williams, Principal Software Engineer, Prism Tech Studios

"In agile nothing is 100% perfect, but establishing success criteria cutoffs instead of deadlines helps to limit work with diminishing returns. Development teams are efficient when they work closely with stakeholders, receiving real-time feedback, validating features with concrete use cases. Cross-talks between data analysis, development, product, and stakeholders held regularly cultivates a culture of collaboration increases flow and ultimately gets everyone to move more efficiently towards business goals."

- Hari Elduri, Sr. Program Manager, Landmark IT Solutions

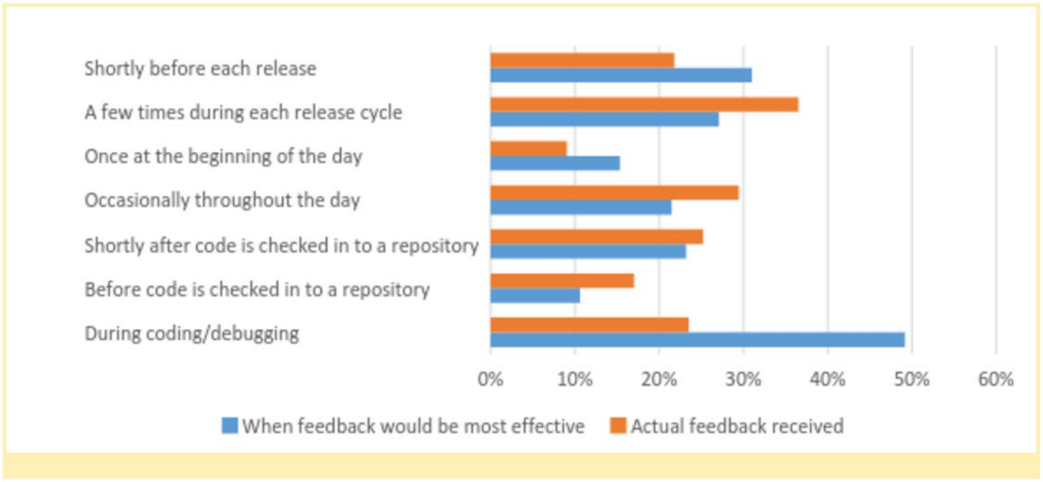
05: Increasing Developer Efficiency with Fast Feedback

Spending less time on overhead activities can directly translate into writing better criteria and tests, but it's also important to consider what developers are not getting that leads to inefficiencies. It's easy to jump to doing things that improve downstream work, but what happens when that work itself isn't optimized?

When developers get feedback on their UX is also important. Respondent data clearly shows that developers want more feedback than they are getting earlier in the development process. If they get it, they perform better. Higher release frequency requires greater visibility; high-velocity developers need to feel confident that they are delivering code that will not slow down the process.

Figure 4 shows front-end developers' responses when asked, "When would feedback about bugs related to your app's user experience be most effective in resolving them?". Almost 50% said feedback during the coding and debugging process would be most effective. When asked when they receive actionable feedback about bugs, only 24% of developers indicated they were getting it during the coding and debugging process.

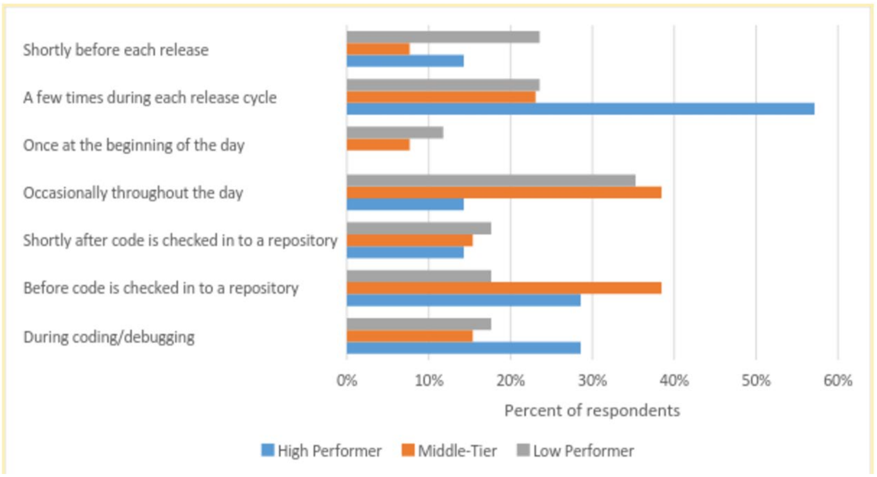
Figure 4 – Quantifying the Gap: Actual feedback vs. desired feedback



Twenty nine percent of high-performing teams were receiving feedback during coding/debugging compared to 18% and 15% for the middle tier and bottom tier performers.

Developers that deliver code at a higher release frequency also are more likely to receive feedback before code check-in, 38% of the mid-tier and 29% of the high performers.

Figure 5 – When feedback on the user experience is received by performance tiers



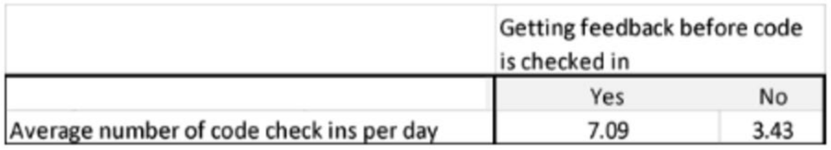
The Impact of Fast Feedback on Developer Collaboration

High-performance developers who release every few days say that they need feedback a few times each release, which for them correlates to at least once or twice per day. Similarly, mid-tier performers releasing every few weeks report a need for feedback occasionally throughout the day too.
Drilling further into t3he data, it can be seen that UI developers who received feedback before checking in code checked in code twice as often as developers who did not receive this feedback.

Twenty nine percent of high-performing teams were receiving feedback during coding/debugging compared to 18% and 15% for the middle tier and bottom tier performers.

Developers that deliver code at a higher release frequency also are more likely to receive feedback before code check-in, 38% of the mid-tier and 29% of the high performers.

Figure 6 – Average code check-ins by those who get UI feedback vs. those who do not



	Getting feedback before code is checked in	
	Yes	No
Average number of code check ins per day	7.09	3.43

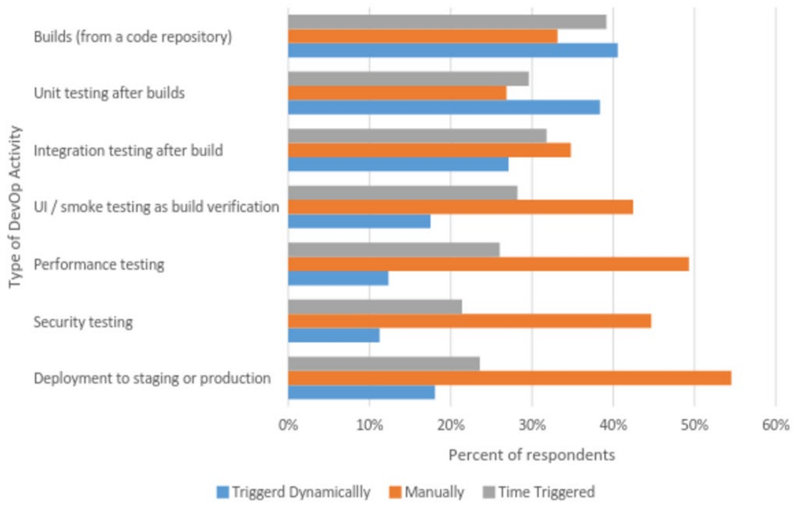
Similarity in these responses indicates that there is a baseline demand by app developers for early feedback about the impact of code changes on the user experience. This correlates to responses about feedback before code check-in with respect to desired vs. actual level of feedback.

Though many developers are aware that better feedback on the user experience is necessary before check-in, only high-performance developers have successfully translated this awareness to operational activities that improve the situation.

06: Accelerating Delivery with Confidence

Moving from increased criteria coverage during planning to operational execution across an automated code pipeline, we asked respondents when specific activities occur across their deployment pipeline to understand how tasks are being executed (i.e. manually, scheduled/timed, triggered).

Figure 7 – DevOps Pipeline Events by Execution Model



Bring Pain in the Process Forward

As we follow the code through this process, we see that the percentage of manual testing jumps with UI testing. Only 27% of unit tests are done manually compared to 42% of UI testing. UI testing can be particularly tricky especially when targeting a variety of platforms such as web browsers and fragmented mobile operating system.

While previously we've established that high-performance teams place a greater importance on non-functional criteria, we also find that they have operationalized quality considerations using automated testing. Figure 8 – UI testing, high performance teams
vs. general population

Both efficiencies and inefficiencies compound as code moves through the pipeline, elevating the importance of input quality at each stage.

As we saw with high-performing teams that carefully manage small repetitive tasks, we see this same trend in their test execution strategies: details matter. The better the criteria, the better the code, the better the test results, and ultimately the better the reliability of the release.

Figure 8 – UI testing, high performance teams vs. general population

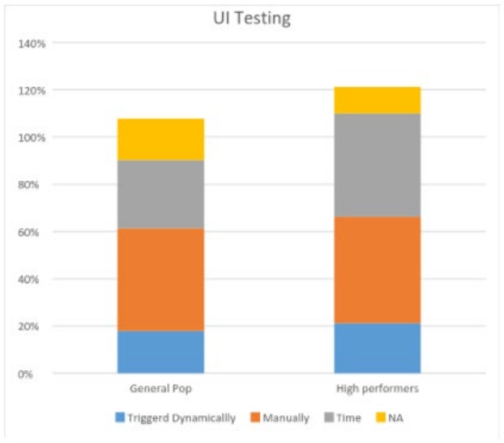
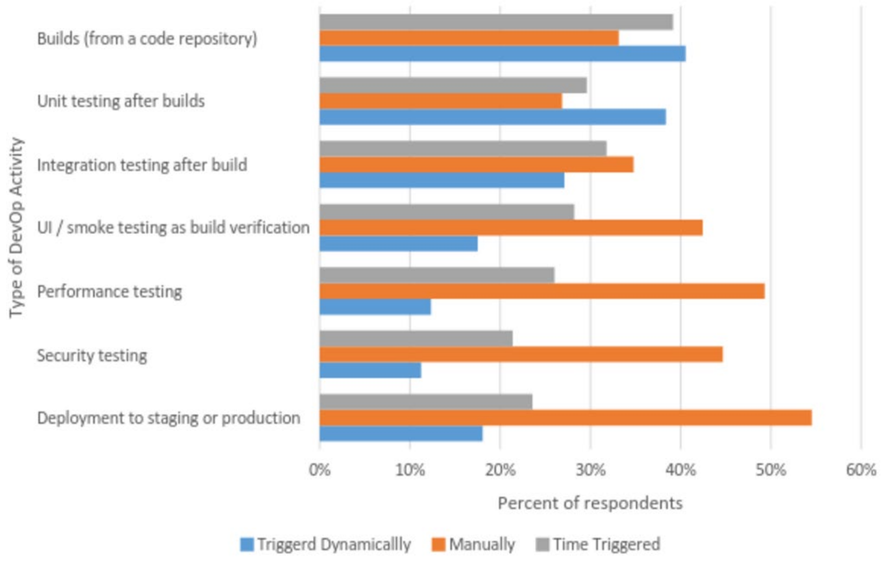


Figure 7 – DevOps Pipeline Events by Execution Model



Better Code Coverage Requires a Multi-modal Testing Strategy
Testing is an important factor in delivering differentiated experiences, but automated testing requires investments in time and tools. To better understand how developers get the most code coverage out of the time they spend testing, we also compared time spent on testing activities with effects on code coverage. What we found was that spending more time on testing will not always translate into greater coverage, but with dynamic automation the relationship is much more predictable.

For unit testing, increases in time spent testing was best correlated with increased coverage when tests were dynamically triggered. For UI testing, we found that testing that increases in time spend best correlated with increases in coverage when testing was triggered at a certain time of the day. With UI testing automation, we found that a mix of dynamic and timed test execution schedules optimizes coverage.

Conclusion

Development teams that frequently deliver high quality user experiences focus on including performance and acceptance criteria in early planning cycles. They follow through with higher test coverage to improve feedback loops along their automated pipelines.

High-performing teams also waste less time, spending half as much time on overhead tasks and twice as much time writing tests than others. Efficiently using minutes throughout the day translate into hours per week for each contributor to improve their app and process.

Despite only half of developers receive the feedback as early as they'd like, those who receive feedback on the user experience before code check-in are twice as likely to contribute code. Mature automated testing increases collaboration, confidence, and predictability in development.

About the Authors

Peter Crocker is the founder and principal analyst at Smith's Point Analytics, a technology research and consulting firm helping vendors and brands capture opportunities in the digital and mobile engagement market. Peter has been an entrepreneur, marketer and analyst in the mobile and wireless industry for 15 years.

Prior to founding Smith's Point Analytics, Peter was a senior analyst with VDC Research covering the enterprise mobility and mobile software markets. In addition to Peter's experience following the mobile and wireless market as an analyst, he has been instrumental in building businesses and guiding strategy at mobile software start-ups. Peter is also part of the IDG contributor network and has contributes to a variety of publications including, InformationWeek, Business Insider, Programmable Web and ReWire.

Paul Bruce currently works with the Perfecto team as Developer Advocate, focusing on helping development groups improve their velocity and quality throughout the software 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, start-ups, and small business IT, Paul now uses his experience to help other developers to save time 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 paulsbruce.io and other online communities.