PROD PERFECT

WHITE PAPER

Breaking Iron Triangle Thinking

A New Approach to Agile vs. Continuous Testing

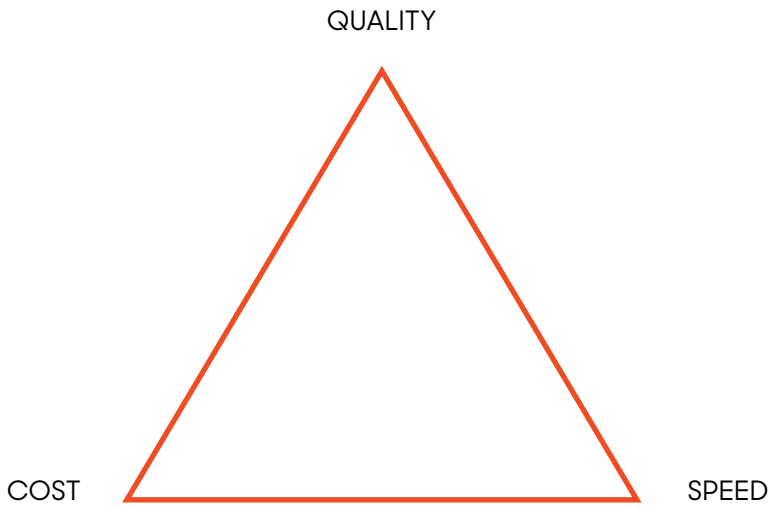
Erik Fogg, CoFounder & Chief Revenue Officer

TABLE OF CONTENTS

THE IRON TRIANGLE OF PROJECT MANAGEMENT

When I used to work as a project management consultant, I would hear brilliant leaders around me say: "Speed, quality, cost. Pick two." In other words: if you want a higher quality product, you have to sacrifice speed, cost, or both. And everyone would nod along: this was true, and it was obvious. In this frustratingly pervasive "iron triangle" decision-making philosophy, true improvement is not possible—only trade-offs are possible. We are doomed to do only as well as we are doing now; our only options are to choose which priorities will be sacrificed to others.

Because the leaders I met believed this model, they missed opportunities to actually improve their project—for example, to find cost improvements without negatively impacting quality or timeline. They never put in the effort to find these opportunities because they implicitly believed these opportunities didn't exist. I have learned over my career that our implicit, unchallenged beliefs about the world have a powerful impact on the choices we make, for good or ill.



IRON TRIANGLE THINKING IN SOFTWARE DEVELOPMENT

In software development, people may not talk about the project management triangle, but this same trade-off-only mindset—an implicit belief in this Iron Triangle—still exists. Iron Triangle thinking means that engineers often feel stuck between prioritizing quality of code, speed (time to market), and cost. Of course we will always have choices that we can make between speed, quality, and cost. But the fallacy of Iron Triangle thinking is believing that the parameters of these choices are fixed. When we believe that improving speed means necessarily sacrificing quality, cost, or both, we limit our potential for innovation.

The pervasiveness of Iron Triangle thinking means that many engineering leaders see only a hard trade-off between the cost and speed benefits of continuous development, and the quality benefits of deploying slower and testing more. They believe these competing demands are irreconcilable. As a result, many development teams who want to improve cost and speed are hesitant to make the move toward continuous development because they're afraid of impacting code quality. They feel forced to choose between developing in two-week sprints—with a massive test suite that is expensive and slow, but assures quality at the end of the sprint—and moving to a Continuous Integration/Continuous Delivery (CI/CD) mode that is faster and cheaper, but sacrifices quality with less testing.

"Iron Triangle thinking means that engineers often feel stuck between prioritizing quality of code, speed (time to market), and cost."

QUALITY: THE "AGILE" APPROACH (BIG AND EXPANSIVE)

In traditional end-to-end (E2E) testing processes, there is no pressure not to add tests because you want a minimal chance that any bugs will make it to production. Typically, in the sprint-based ("agile") development approach, quality is optimized at the price of cost and speed.

Most leaders are accustomed to running E2E testing at the end of a sprint, so the test suite can run as long as it needs to. Once E2E test suites grow to a certain size, you have to run them less frequently, as they can take several hours and simply can't be run every time developers check in code. This means that there's a feedback lag: developers discover bugs in their code days or weeks after they're written. At that point, it's impossible to pinpoint which deployment even caused the bug, so you can't toss the buggy code back to the developer who wrote it. Often, leaders assign these bugs to more junior developers who are forced to mine the code to understand the intent of the original developer. In all, these bugs require a bunch of time and resources to fix.

"Typically, in the sprint-based ("agile") development approach, quality is optimized at the price of cost and speed."

SPEED AND COST: THE CONTINUOUS APPROACH (LEAN AND MEAN)

In a CI/CD environment, a team supports continuous development with continuous testing: every deployment is tested individually before it is staged or shipped to production. With fewer tests, the test suite runs in a matter of minutes, feedback lag is minimal, and buggy code returns directly to the engineer who wrote it. With this approach, developers are more fresh on what they just wrote so that when bugs do get caught, they're able to pinpoint exactly where the problem is and resolve it in minutes, rather than hours or days. Developers kick out code and move on, producing more code in the same amount of time. Thus, with the continuous approach, the

KPIs that the engineering leader maximize are developer velocity (faster deployments) and cost (fewer tests to maintain).

The drawback to having fewer tests, of course, is that you're ensuring less quality before your code goes out the door. Fewer tests simply means fewer opportunities to catch bugs before they hit production. A purely continuous testing suite sacrifices quality in favor of speed.

A NEW APPROACH: AGILE AND CONTINUOUS

As engineering leaders, as long as we continue to subscribe to Iron Triangle thinking, we will never be provoked to ask ourselves if we can actually improve a process without making a trade-off. When we break Iron Triangle thinking, we can find a third way: an option that is not some form of compromise between competing priorities. We can find paths that give us the best of everything. In software development, choosing a third way means believing that it is simply not the case that every improvement in either speed, cost, or quality must necessarily come at the expense of the other two. Breaking Iron Triangle thinking requires that we do the harder thing and choose a smarter, more strategic approach to deployment design. One such way to break the Iron Triangle is to simply choose the best parts of testing from both agile development and continuous deployment.

"One such way to break the Iron Triangle is to simply choose the best parts of testing from both agile development and continuous deployment."

What gets us both the speed of the continuous environment and the quality of the agile environment?

One possible design change to your deployment structure is to implement continuous testing early in your SDLC with the quality benefits of the end-of-sprint testing that occurs in agile development practice.

In this structure, you preserve the benefits of continuous development: your developers can still contribute small chunks of code and run tests continuously for rapid feedback. In addition, you can run a larger test suite in a staging environment against a number of builds to catch bugs that the continuous test suite missed. This approach means many of your bugs—and if your continuous test suite is well-targeted, your priority 1 bugs—are

caught sooner and fixed faster. Anything missed here is caught by the larger test suite later. This way, your developers get the speed advantage of continuously developing and testing, and your application gets the quality benefit of running a larger test suite—all without costs beyond maintaining the larger suite that would've been written for an agile team anyway.

MOVING FORWARD: A PRACTICAL APPLICATION

How do we manage both agile and continuous test suites?

At ProdPerfect, our team uses data to maintain a continuous test suite focused on our customers' most commonly-used features, while they manage their own larger test suite. Whatever your prioritization mechanism is, you can start the process by building your total regression test suite and tagging your top priority tests. Then, you can configure your CI to continuously run only the tests that are tagged as priority, and run the whole kit daily (or at whatever frequency works for you) on your staging environment. Compared to the "agile" testing approach we explored, this new approach to deployment design improves speed and cost with no negative impact on quality. Compared to pure continuous testing, this approach improves quality with zero to minimal impact on speed and a positive impact on cost (you're catching more bugs before production). Instead of choosing one approach and sacrificing the other, you are implementing both intelligently and improving multiple parts of your project triangle without negatively impacting the others. You've broken the Iron Triangle and get the best of all worlds.

As engineering leaders and teams, you can break the Iron Triangle of project management by structuring your improvement cycle not around trade-offs, but on continuous improvement. A great engineering leader will use the momentum and mindset shift gained from their first Triangle-breaking win to push their team to challenge the assumptions that have held them back from improving all parts of their engineering practice.

"As engineering leaders and teams, you can break the Iron Triangle of project management by structuring your improvement cycle not around trade-offs, but on continuous improvement."

ABOUT PRODPERFECT

Unleashing the power of machine learning to solve the hardest, most important, and previously unsolved problems in end-to-end (E2E) QA testing, ProdPerfect is the only autonomous E2E regression testing solution on the market that continuously identifies, creates, maintains, and evolves E2E test suites via data-driven, machine-led analysis of anonymous live user traffic. It is a fully-managed testing solution that addresses insufficient test coverage which causes critical and costly bugs in production; removes the burden that consumes massive engineering resources; and eliminates long test suite runtimes that slow deployments and decrease developer velocity.

SCHEDULE A PRODUCT INTRODUCTION TODAY

PROD PERFECT