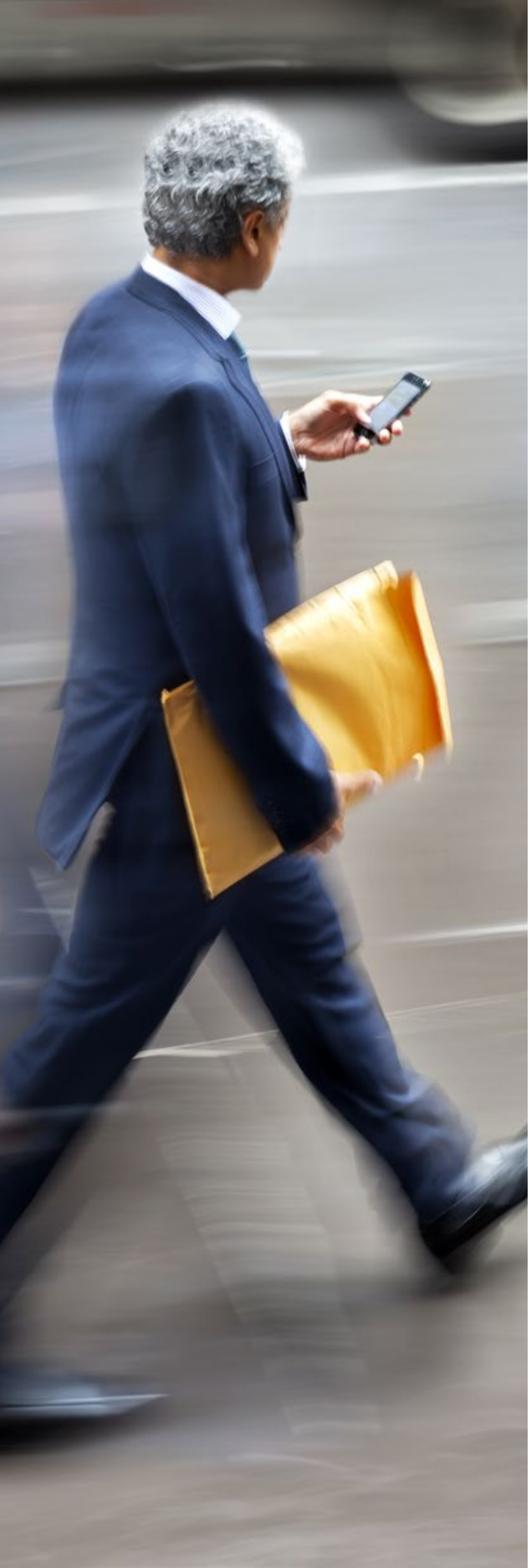
Why Agile Parallel Development Is Critical to Your Digital Transformation Strategy



The Need for Speed, Efficiency and Quality in the Application Economy

In today's application economy, software is rewriting the rules of business.

With competitive pressure to rapidly innovate and iterate applications, organizations like yours are adopting digital transformation strategies, in which digital components are added to existing products and services (e.g., mobile applications, universal device support, Internet of Things (IoT), etc.).

As companies attempt to build an ecosystem of value around these "connected" products, digital transformation is essentially spawning a new generation of software—and the application programming interfaces (APIs) they require to flourish in this collaborative environment.

At the same time, customer expectations for original, high-quality services are at an all-time high, forcing IT organizations to rethink how they develop, test and deploy software in order to maximize speed, efficiency and reliability—and ultimately meet market demand.

By 2016, more than half of B2B collaboration will take place through web APIs.[1]

[1] Gartner Hype Cycle for Application Development, 2014, Thomas E. Murphy, et al, July 29, 2014.

Digital Transformation Creates New Challenges in the SDLC

As organizations expand their application delivery goals in support of digital transformation initiatives, IT groups must contend with new challenges in and around the software delivery lifecycle (SDLC), including:

INCREASED COMPLEXITY

While the face of a modern application may be a simple collection of buttons and fields, its architecture comprises a complicated tangle of composite services and integrations—each of which must be accounted for during development and testing. In addition, most development involves connecting new user interface (UI) front ends (i.e., systems of engagement) to legacy systems of record on the back end, which creates new coding and integration challenges.

REQUIRED INTEGRATION

Because modern applications must integrate with complementary and third-party services—in addition to internal systems of record—APIs must be regularly built and tested, but dev and test teams do not always have access to these critical resources when they need them. And as trends like IoT continue to gain steam, IT teams are having to integrate a whole new set of objects with unique attributes and requirements.

RESOURCE CONSTRAINTS

IT teams are being asked to build and deploy more applications faster than ever before, but there are only so many resources on which to code and test. When constraints happen, teams move onto other things rather than just sit and wait for needed resources to become available—which leads to idle time for the project that pushes out the overall delivery timeline.

LACK OF AUTOMATION

As applications have become more complex and layered, organizations often adopt different testing tools that require manual coding or translation across the different layers and environments. And if they use Agile methods across distributed teams to expedite dev/test activities, the application sub-components cannot be tested fully until integration testing, when all the parts come together for the first time.

Collectively, these challenges prevent IT teams from achieving parallel development and test practices.

The Failure of the Mock and Stub Workaround

Many IT organizations attempt to work around constraints in development and testing practices by manually coding custom mocks and stubs. For example, when a developer needs access to an unavailable system or API, he might create a bit of code that simulates the behavior of that constraint and gives him the response he needs to move onto the next step. The problem with mocks and stubs is that, while they can address specific constraints in a pinch, they create larger issues that can jeopardize digital transformation goals.

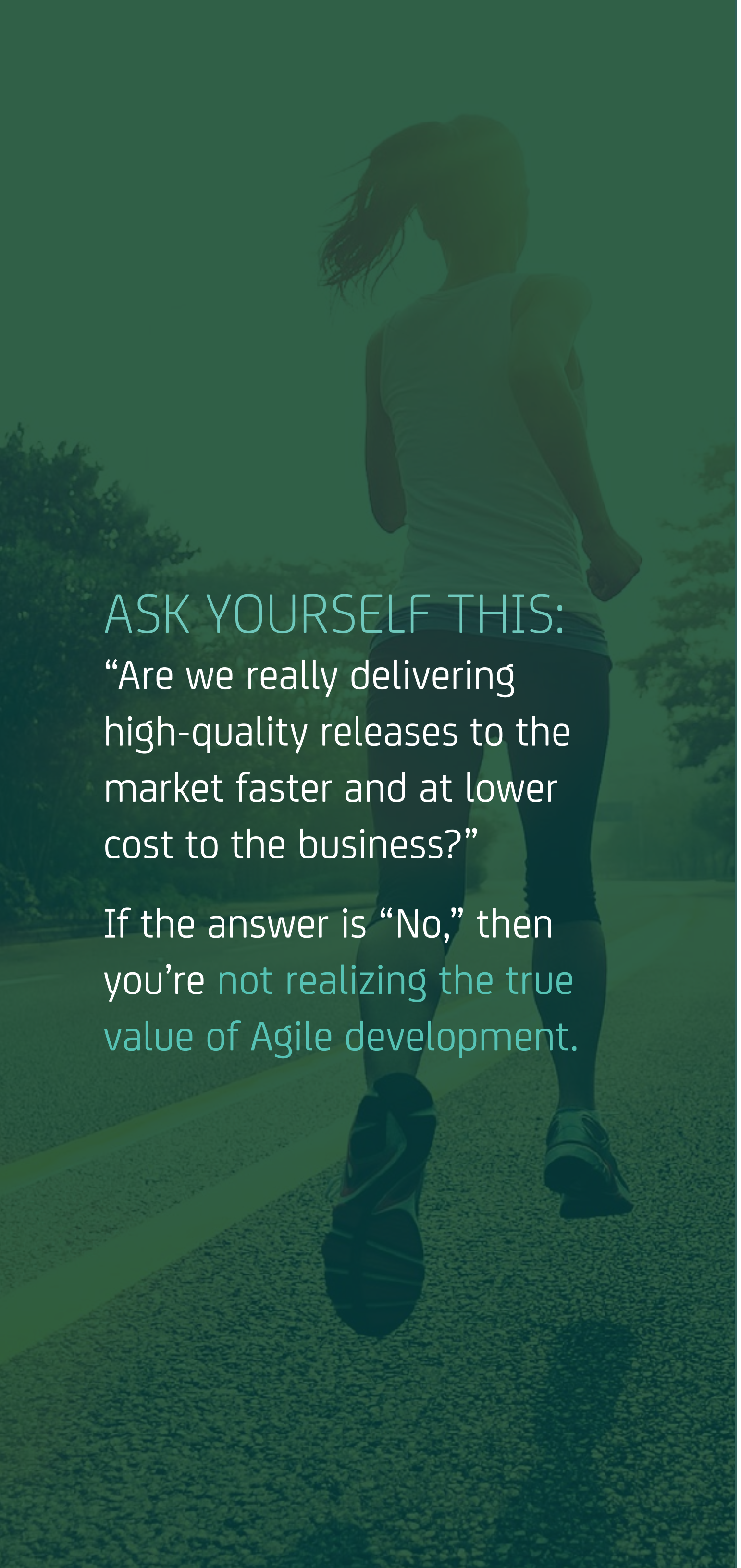
For one, developers should be maximizing their time coding the application itself, rather than wasting it on mocks and stubs that have their own maintenance, consistency and data limitations. While teams create these assets for specific tests, they ultimately have to throw them away, wasting valuable time and effort.

There's also a problem of scale. As today's composite applications get more intricate and complex on the back end, they require increasing numbers of custom mocks and stubs to simulate component parts. On top of that, every new mock or stub represents an opportunity for human error to enter the equation, which can slow an application's movement through the SDLC and hurt its quality overall.

So rather than solve challenges, mocks and stubs actually create extra work (and re-work) for developers and testers. In addition, they take time away from more productive coding activities and open the door for additional defects and bugs to make their way into the application.

94% of line-of-business executives feel **pressure to release new applications faster**, citing "customer demand" and "competitive actions" as their top two drivers.[2]

[2] CA and Vanson Bourne, Application Economy Research, 2014.



ASK YOURSELF THIS:

"Are we really delivering high-quality releases to the market faster and at lower cost to the business?"

If the answer is "No," then you're not realizing the true value of Agile development.

The Differences Between "Doing Agile" and "Being Agile"

Another way some organizations are tackling the challenges of digital transformation is with Agile development methodologies.

The original goal of Agile was to break the slow and monolithic development practices of the past into smaller chunks, so developers, testers and business development people could work concurrently and speed new features to market. Despite the best of intentions, however, many IT groups fail to get applications into production with the speed and quality they would like, and there are some common reasons why.

In some cases, they follow Agile practices in development, but then save all testing for the end, which essentially shifts bottlenecks to later in the SDLC and leaves bugs and glitches in a stage where they are tougher to identify and more expensive to fix.

In others, IT teams are constrained by the limited amount of dev and test resources available to them. This forces them to stagger their efforts, rather than work in parallel, which cuts into the speed and efficiency their Agile methods were meant to enhance—and pushes out the application's deployment into production.

The Benefits of Agile Parallel Development

With Agile Parallel Development, you can:

REDUCE COMPLEXITY by simulating a variety of development and testing environments, services and behaviors—whenever and wherever needed

STREAMLINE INTEGRATION and collaboration with third parties by accelerating API prototyping and management

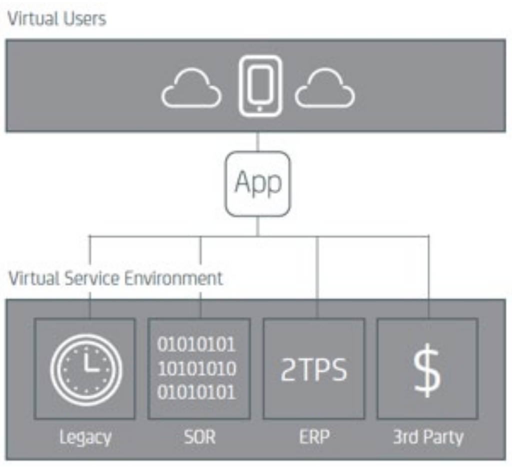
ELIMINATE CONSTRAINTS that prevent parallel development and testing, so you can speed time-to-market and raise the quality of your applications overall

AUTOMATE TESTING earlier in the SDLC, reducing the need for time-consuming manual testing and improving overall application quality

Agile Parallel Development in Three Steps

STEP 1: **Remove Constraints**

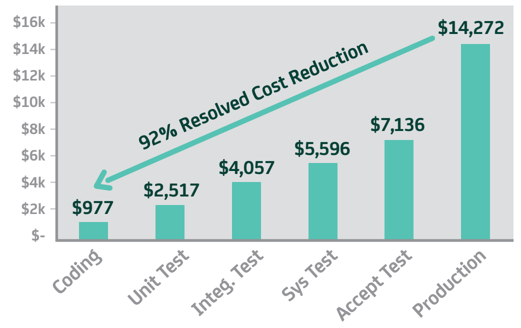
Remove time, data, availability and cost constraints by simulating dependent systems and customer behaviors as virtual services—whenever and wherever they are needed.



STEP 2: **Shift Testing Left**

Leverage virtual services and test assets to automate testing at earlier stages of the SDLC— where defects can be fixed more quickly and at lower cost—and improve the quality and stability of your production applications.

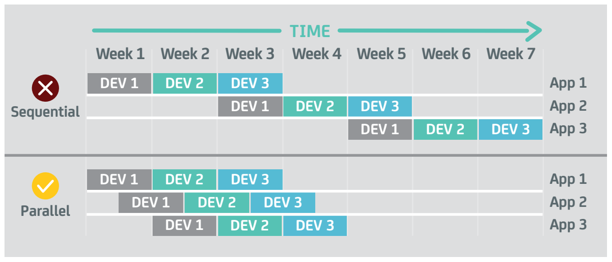
Defect Correction Costs



Source: Lyon, Dan, "Systems Engineering: Required for Cost-Effective Development of Secure Products," The SANS Institute, 2012.

STEP 3: **Develop in Parallel**

By simulating needed systems, services and test assets as needed and automating testing, you can dramatically increase developer productivity and empower teams to work in parallel.



What's Next?

In chapter 2 of our ebook series, we'll take an in-depth look at the various types of constraints throughout the SDLC, how they impact the business and what development and test teams can do to eliminate them.

For more information, visit ca.com/agiledevelopment.

CS200-127705-1