With service virtualization, your DevOps teams can use virtual services instead of production services, enabling frequent and comprehensive testing even when key components are missing from your system architecture. By emulating the behavior of essential components that will be present in a final production environment, service virtualization allows complex applications to undergo integration testing much earlier in the development process, removing key bottlenecks that would otherwise delay production and time-to-market for an application under test (AUT). For the development of most enterprise applications, which rely on a mixed array of system components working together in harmony, service virtualization “fills in the gaps” of missing system components by simulating their responses to show how the various components interact. It is especially useful in the development of complex cloud-, API-, and SOA-based systems, as well as at any point in a production cycle where important hardware and software components aren’t readily available for testing purposes.

More and more companies are using service virtualization to improve productivity, reduce testing costs, and [deploy higher-quality software](http://smartbear.com/product/ready-api/servicev/features/share-virtual-services) in a shorter timeframe. In addition to emulating major software applications, third-party services, and even whole backend systems, the virtual assets can also be reliably shared and used by your entire production team, facilitating more efficient parallel development practices. By quickly and easily removing dependency constraints across your organization through virtualization, you can gain a competitive advantage over other companies still waiting in the linear-development limbo.

Why Use Service Virtualization?

Traditionally, testing teams have had to wait for nearly completed applications to be deployed before proper functional, integration, and performance testing could begin. Distinct project teams might produce different components of a system or application, one piece at a time, and then assemble them into a single working product before allowing the testers to have their way with it. It’s logical, it’s linear, and it’s slow.

Focusing on testing a nearly finished software application—with all of its components nicely integrated behind a functional user interface—will obviously remain an essential step in any development cycle, but in these days of rapid (and continuous) development cycles, it often isn’t practical to wait that long before testing to see how various software components communicate with each other. Testing needs to happen alongside development from the start, and this is especially true in the production of heterogeneous systems involving multiple layers of interdependent components, [third-party apps, and APIs](http://smartbear.com/all-resources/articles/what-is-api-virtualization/).

API layer, where major problems down the line are often first introduced into system interfaces, rather than having to wait to test a more complete production-ready app. Using service virtualization, developers can validate integrations earlier than would otherwise be possible. And considering that the user experience of an application is wholly a function of the sum of its parts, it makes sense to ensure that those parts are working well as they’re created, rather than waiting for a finished application. Service virtualization can be useful at any point in the development of an application, from assisting with small manual unit tests to enabling automated performance tests of an integrated system.

Unless your company has sufficient resources to provide your developers and testers with every actual component required by a production system, using virtual services throughout the development cycle can save a lot of time and money. Any software development team can benefit from service virtualization, especially when it isn’t practical to repeatedly test against dependent third-party components, such as Salesforce, Oracle, or PayPal. By virtualizing the behavior of a CRM, ERP, or payment gateway in your system architecture—with simulated data and software responses—your development efforts will be able to proceed freely and you’ll be able to conduct tests as often as desired, paving the way to smooth user acceptance tests down the line once the actual third-party components are deployed.