Software Driven Everything and What It Will Take to Get There

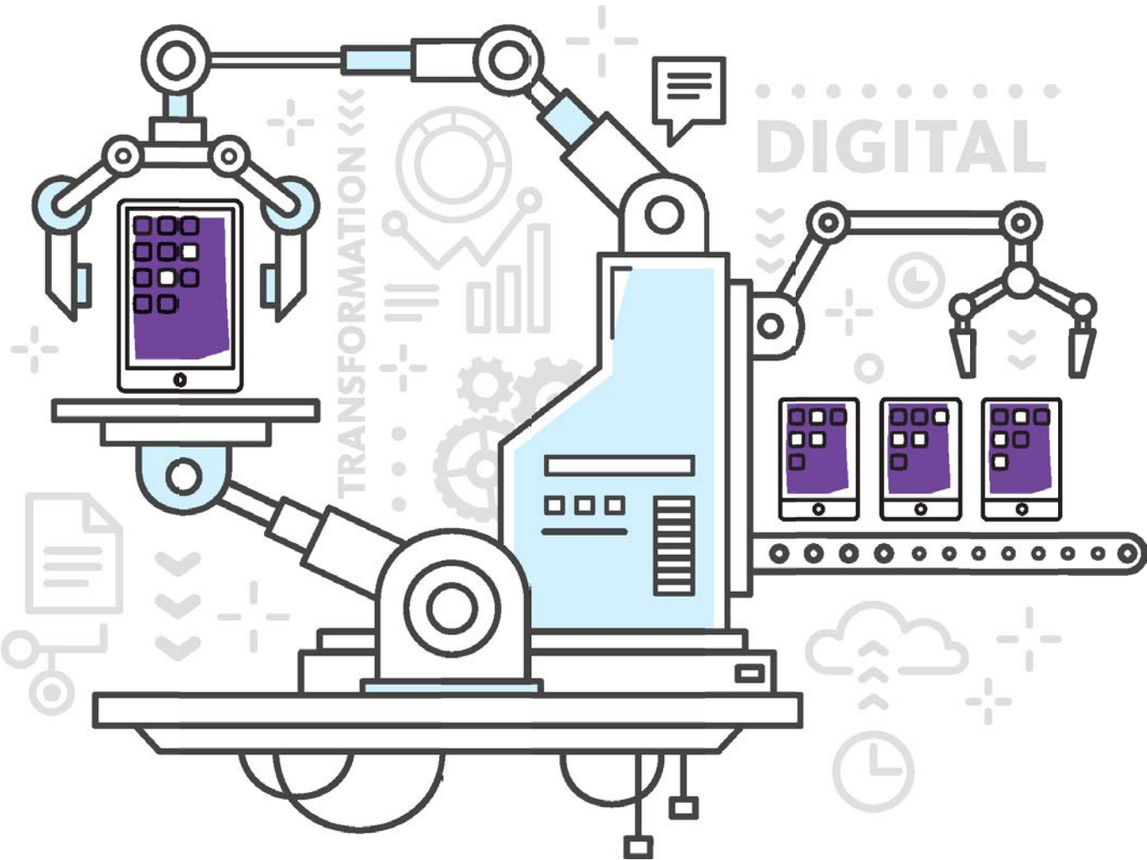
AGILEPOINT

Table of Contents

THE REALITY OF TOTAL DIGITAL TRANSFORMATION

One of the latest battle cries among IT thought leaders is “software-driven everything,” a state of being at the zenith of digital transformation that sparks the imagination, but which may not be realistic anytime soon. And while the topic of software-driven everything makes for lively debate, what almost everyone will agree on is that, for most—if not all—organizations, getting from where they are to a state where software drives every process from the top to the bottom of an enterprise will be an enormous and expensive challenge.

Not only will it require a methodology for lightning-fast creation of new, custom business applications that define workflow, codify best practices, and consolidate entire eco-systems, but this avalanche of new business apps will need added dimensionality—the ability to responsively adapt to changing technical and business requirements without having to be taken offline, updated, recompiled, linked, and executed every time something changes, which, in today’s business environment, would be constantly.



What's the ETA?

So how far off is this futuristic world of software driven everything? An answer is hard to give but not because of a lack of hardware, bandwidth, or big ideas. Rather, the chasm to be crossed is inadequacies in the current technologies and methodologies used to develop and maintain software.

The Great Software Divide

It's been years now since Agile overtook the old Waterfall method of software development, perhaps the biggest catalyst for this change being shorter development cycles. And while there are lots of different schools of Agile development, each, nonetheless, shares some sizable shortcomings in regard to full-on digital transformation:

1. Agile development requires actual software engineers to write code, which, of course, is time and resource intensive. Put another way, while Agile is faster than Waterfall, it's still nowhere near fast enough to automate every little process in a company, at least not in most of our lifetimes. Furthermore, given the sheer number of apps that will need to be written, the gap between available and necessary software engineers capable of this level of development is, at present, huge but will get ever wider for the foreseeable future.

2. Hand-coded software has to be compiled, linked, and executed. And it's this reality—baked-in feature sets—that may be the biggest short-coming of current software-development methodology when juxtaposed with the sheer size and scope of full-on digital transformation.

The bottom line is hand-coded apps simply can't be produced fast enough to keep up with status-quo requirements, let alone the exponentially larger requirements of digital transformation. And these traditional, hand-coded, baked-hard-as-rock apps lack the responsive characteristics necessary to function in today's more liquid business environment without constant upkeep. In fact, upkeep is arguably the real rub, being a pernicious form of technical debt that would quickly eclipse the financial upside of digital transformation.

AN EMERGING SOFTWARE SOLUTION

Low-Code Platforms—a Step in the Right Direction

Over the past several years, many pure-play Business Process Management Suites (BPMS) have morphed into low-code process platforms, which can still be used to automate processes and improve operational efficiency but which can also be used to build composite apps offering broader value to entire business ecosystems. This new breed of low-code platform is lighter and more nimble than its BPM predecessors and requires much less upfront investment and long-term commitment from customers.

Low-code platforms across the board utilize a point-and-click development environment that allows citizen developers (power users but not necessarily software engineers) to compose applications by dragging activities, forms, and other types of controls onto a canvas and then configuring each to application specifications. The result is an application model, a visual construct that incorporates any number of on-premises-based systems as well as cloud-native services and which depicts flow as well as inter-relationships between application components.

Part Way There

Today's low-code platforms address one of the problems listed above: low-code apps can be built much faster than hand-coded apps, and they can be built by people with less technical skill sets than actual software engineers, a fact which dramatically increases available human resources. Shorter development cycles and more hands on deck is definitely a step in the direction of software driven everything.

However, the other, perhaps bigger problem mentioned above (actual digital transformation will require software that can self adapt to continual changes) will demand a technology far in advance of the current low-code standard, which is *code generation*.

Baked-In = Non-Responsive

Virtually all low-code platforms use a point-and-click approach that produces application models. Most of today's low-code platforms must then transform the visual model into an actual software application through a process known as code generation, which, as the name implies, converts the model into low-level computer code. This code must then be compiled, linked, and executed. In other words, once a low-code app is deployed, it is no different than any other hand-coded app, having baked-in features and functionality.

Exacerbating the problems of “baked-in,” such apps, in order to run, must be loaded in their entirety into a process (workflow) engine, where they will live throughout execution. This permanent residence in the engine accounts for much of the shortfall of today’s software in regard to digital transformation, and here’s why:

When business or technical requirements change the state of each running instance of the app must be preserved while the app is taken out of memory. Modifications are then made to the model, at which point, the model must once again go through the code-gen process. The resulting modified code must then be recompiled, loaded back into memory, and reconciled with the preserved state of each running instance of the app.

The sheer weight of the machinery involved in this endlessly recurring cycle brings the digital-transformation problem into specific relief: An app with baked-in features that must live, in its entirety, in a process engine throughout execution is dysfunctional in an environment where application characteristics must continually change, primarily because baked apps must be manually updated. Again, in the world of software driven everything, apps must be self adaptive, eliminating the resources and accompanying expense of manual maintenance.

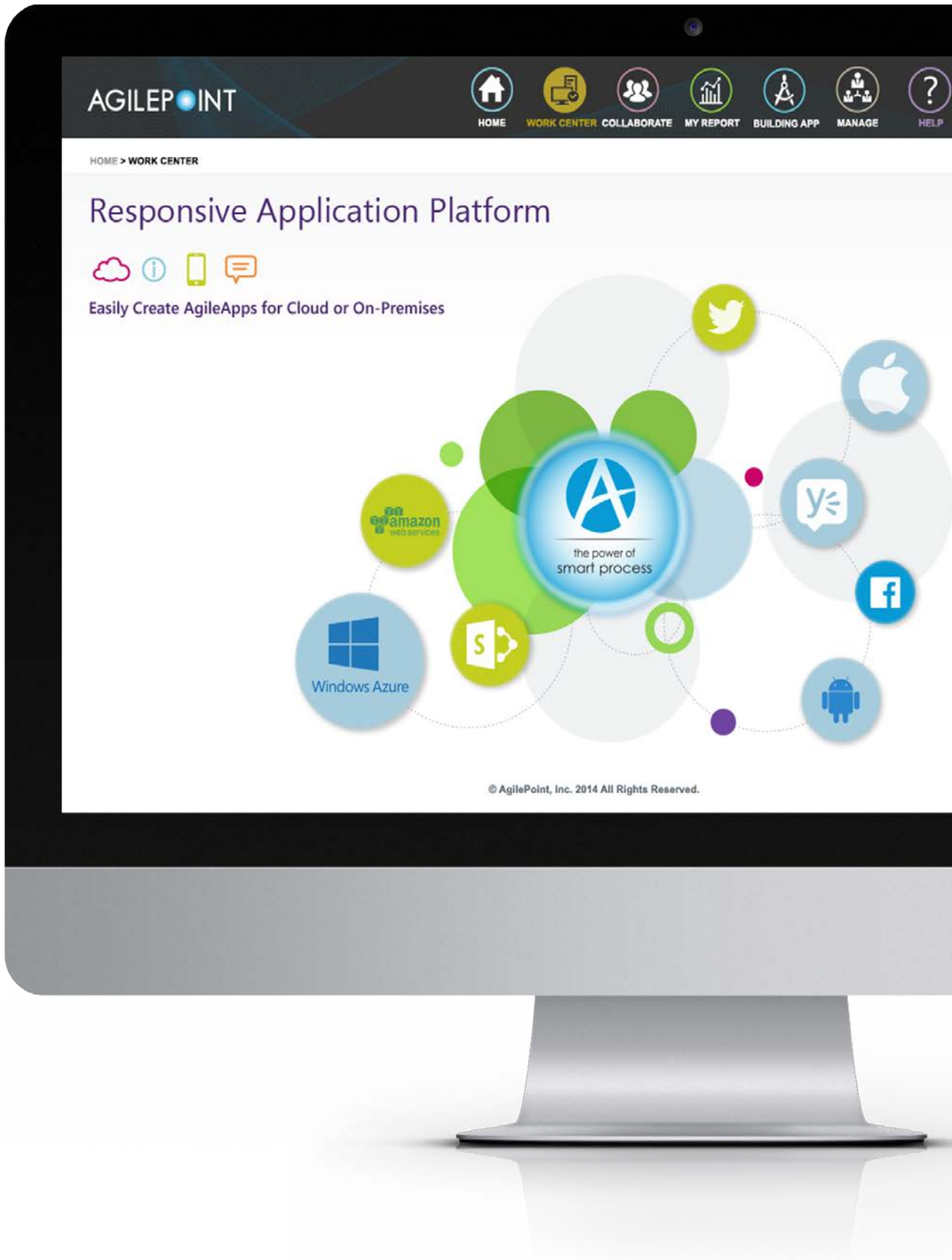
Responsive Low-Code Platforms

What's needed, then, to catalyze software driven everything is a new software development paradigm, one that satisfies the quantity problem—lots of apps delivered fast—as well as the responsive problem—software that can dynamically respond to the constantly changing conditions of digital business.

Responsive is, of course, the hard part. Most low-code platforms evolved from BPM suites and simply don't have architectures that support responsive-application development. One low-code aPaaS that does is AgilePoint NX, which can be used to highlight requisite responsive platform characteristics.

• Model-Driven VS. Code Generation

In contrast to code-gen-based systems, AgilePoint NX incorporates a true, model-driven architecture. With this approach, each component of a visual model constitutes metadata (stored in an XML registry) that defines how an underlying chunk of code will work. For example, as a developer drags an activity into a model, the relative positioning of the activity to other model components constitutes metadata, and this metadata affects the underlying code. If a developer changes the relative positioning of a component to other components, the underlying code, in turn, gets modified to reflect the change. Likewise, any operational characteristics provided by the developer (configurations typed into dialog windows) along with flow lines between components constitute metadata, and all of it, taken in totality, is abstracted into application features and functionality.



• XML-Based Process Engine

As was mentioned above, with AgilePoint NX, the application model, itself, is XML. In execution, the model (XML registry file) is fed directly into the process engine. Note that, in contrast to traditional compiled apps, which must be fed into a process engine in their entirety, an AgilePoint XML-based model can be processed one component at a time. This piece-by-piece approach reduces storage and processing requirements, but, more importantly, enables components not currently in memory to undergo modifications. Under any circumstances, any part of a model is in memory only while it's in use (for a fraction of a second at a time) and then is removed. And once a component is no longer in memory, it can be modified.

• Runtime Updates

A model-driven architecture combined with an XML-based process engine results in apps that can be modified at runtime—as new business conditions are pushed into the model, the underlying code is modified, and the running application changes mid-flight. Depending on the application in question, updates to components could be made manually—a system admin or, perhaps, a line manager, entering data into component dialogs. But with digital-business apps, modifications are more likely to be made programmatically, via feedback loops of fresh data from any of millions of devices equipped with sensors and radio frequency tags.

Continuously Variable Applications (CVAs)

If you've ever pulled a trailer up a steep mountain grade, you may have experienced your vehicle repeatedly shifting back and forth between gears in an attempt to find a ratio that is appropriate for constantly changing requirements. One gear is too low, the other is too high. In contrast, continuously variable transmissions (CVTs) offer infinite gearing ratios, enabling a transmission, based on external data sources, to deliver the optimal ratio at all times during the climb. Needless to say, the CVT approach is way better.

In some ways, the difference between a five-speed automatic transmission and a CVT can be likened to the difference between baked apps and responsive (continuously variable) apps. Baked apps offer a predefined set of options to handle various conditions, but when actual conditions vary from expected scenarios, the necessary permutations are not available. The app can be taken offline and modified to account for new conditions (think adding a couple of extra ratios to a transmission) but the same problem still exists—the machinery of adaption is not well suited to a liquid business environment.

In contrast, responsive apps are architected in such a way that they can dynamically reconfigure on the fly to account for any number of changing business conditions and technical requirements, much the way a continuously variable transmission can generate any gearing ratio necessary to account for hundreds of external factors. Just as CVTs are way better than regular transmissions, responsive apps are way better than baked apps, especially when it comes to the requirements of a software-driven-everything organization.

APPLICATION LIQUIDITY

Regardless of the architecture that is used, software that is capable of catalyzing complete digital transformation must be able to adapt to changes without recompilation and the baggage that goes along with it. AgilePoint NX utilizes a mode-driven architecture, a metadata abstraction layer, and a stateless process engine to get the desired results. Other platform vendors may get a similar result utilizing a different architectural design.

AGILEPOINT

Software Driven Everything and What It Will Take to Get There

To learn more about **AgilePoint NX, the Responsive Application Platform,** visit www.agilepoint.com.

Learn More | Request Trial | Request Demo