

Bimodal IT: Innovating at the Pace of Change with a Low-Code Platform

Bimodal IT: Innovating at the Pace of Change with a Low-Code Platform

PHASE 1. Introduction

PHASE 2. Why Traditional Development Paradigms Won't Work for Mode 2 Development

PHASE 3. Why Low-Code Platforms Are Well Suited for Mode 2 Development

- 01. Pre-Built Services
- 02. Declarative Frameworks
- 03. Metadata Abstraction
- 04. Model-Driven Architectures
- 05. Extensive Extensibility
- 06. Responsive Assets
- 07. Mid-Execution Asset Management

PHASE 4. Is There a Downside to Low-Code?




For the last few years, Gartner has been preaching the doctrine of bimodal IT, contending that today's digital enterprises need to sustain two types of software development efforts.

The first (Mode 1) is **traditional** development of applications and integrations using tried and proven development techniques. Mode 1 is hierarchical, safe, and predictable, but it is also, expensive, time consuming, and rigid.


The other (Mode 2) is non-hierarchical, highly adaptive to context, innovative, and potentially disruptive. Mode 2 emphasizes **speed, agility**, and short development cycles.

For Gartner, the two modes represent a sort of Yin and Yang of IT—almost antithetical and, yet, both essential and mutually complimentary, if implemented correctly. Given that Mode 1 development is what IT departments have been engaged in for decades, it's safe to say that Mode 2 development is of greater interest today, as IT departments look to innovate at the pace of change—not easy, to put it mildly.



Mode 1

- Traditional
- Hierarchical
- Expensive
- Predictable
- Safe
- Rigid
- Time Consuming



Mode 2

- Shortened Development Cycles
- Potentially Disruptive
- Innovative
- Highly Adaptive
- Fast
- Agile
- Non-Hierarchical

PHASE 2. Why Traditional Development Paradigms Won't Work for Mode 2 Development

As was already stated, the emphasis with Mode 2 initiatives is speed and agility—the primary vehicles for innovation. With Mode 2, over-planning is the enemy, failing has to be okay, and learning happens along the way.

Given the objectives of Mode 2, a traditional software development paradigm can't easily be applied. Traditional development projects are expensive and time and resource intensive, a reality that gives way to extensive planning. *A measure-twice-cut-once* mentality must be the standard in a world of hand-coded apps written against system APIs. But this old-school approach doesn't lend itself well to the more liquid requirements of experimental application development.



In contrast to the traditional development paradigm, low-code platforms (code-optional platforms) have been designed from the ground up for Mode 2 scenarios. Of course, not all low-code platforms are created equally. But assuming you choose a true, enterprise-class platform, here are several reasons why it will catalyze your Mode 2 agenda.

01. Pre-Built Services

Imagine that you need to build a new business application but that time, money, and human resources are at a premium. If your plan was to hand code such an app, you would first have to write a bunch of supporting code—for example, the application services layer and a database layer—as well as all the basic plumbing and generic functionality that business applications typically need.

In contrast, with a low-code platform, you'd have, out of the box, a web-based, easy-to-configure, application-development framework that would allow you to design and build a sophisticated data model with complex workflow, reporting, and a bunch of other available, prebuilt services.

This low-code platform would also automatically render for your users a modern, customizable, browser-based user interface.

Again, assuming you've got an enterprise-class platform, it will be JavaScript, flash, and Ajax compliant, having lists, forms, dialogues, and all the navigation elements typically required by business apps. It will provide supporting layers for SOAP and RESTful web services, email and notification services, and resulting applications will be responsive to devices and browsers. Furthermore, your low-code apps will be supported by a flexible, configurable, robust security model with everything easily and rapidly deployable into a highly scalable online environment for any number of

users—all of this with no effort to build out a supporting hardware infrastructure, middle-tier-logic, database layers, backup services, workflow engines, web service APIs, and so on.

In short, the pre-built services available with low-code platforms dramatically speed the application development cycle, in turn, lowering cost and other resource requirements, and paving the way for experimental app development.



PHASE 3. Why Low-Code Platforms Are Well Suited for Mode 2 Development

02. Declarative Frameworks

With a low-code platform, a citizen developer (business expert/non programmer) can build a powerful application by dragging forms, activities (actions, for the Microsoft crowd), and various types of controls onto a canvas, and then configuring each according to app specifications. The result is a graphical model that depicts all application componentry, as well as flow and inter-relationships of components.

The application components are pre-built by the platform vendor and will enable the construction of cross-functional apps that will incorporate activities from major line of business systems (SharePoint, SAP, Oracle, NetSuite, Salesforce, Marketo, etc.) as well as storage utilities (Box, OneDrive, Google Drive, OneDrive, etc.), eSignature systems (DocuSign, Sertifi, etc.) and other core technologies (MySQL, SQL Server, .NET, etc.)

Perhaps the best part is this: Because of this declarative framework, some or all of these low-code apps can be built by staff members outside the IT department, a fact which will likely expand your pool of available resources and enable business users to participate in development cycles.



03. Metadata Abstraction

With the types of model-driven applications produced by low-code platforms, a developer determines application characteristics by defining metadata, rather than by writing low-level computer code, toggling switches on or off, or using tables or configuration files. The application model is an abstraction that allows anyone (IT staff or citizen developers) to more easily manipulate any or all facets of an application, such as an information model, a process model, or a user interface model, by simply entering information into fields on forms. And this information (metadata) defines the characteristics of each activity, form, or other control.

It's this characteristic—low-code, model-based applications are metadata-driven—that makes low-code apps quick-to-build, quick-to-change, quick-to-deploy, and, just as importantly for Mode 2, easy to jettison, if they don't work out.



PHASE 3. Why Low-Code Platforms Are Well Suited for Mode 2 Development

04. Model-Driven Architectures

The term “technical debt” is bandied around quite a lot these days, and, while it can be applied in a number of different ways, one of the most common has to do with hand-coded apps written against system APIs. While APIs make cross-system integration readily available, such hand-coded apps and integrations become legacy debt the minute they are deployed. The debt, of course, takes the form of ongoing maintenance, which must be performed by software engineers, and can never be repaid—maintenance must be performed throughout the code’s useful life. Consequently, over the course of time, an organization’s technical debt begins to resemble, well, the national debt—unimaginably large and growing out-of-control.

The good news is low-code application models are the solution to a significant percentage of the technical-debt problem. Rather than an ever growing mountain of spaghetti code, new app development, as well as integrations across systems, take the form of easy-to-understand models. Beyond the fact that these model-based apps are dramatically easier to build than traditional, hand-coded apps, the models actually serve as run-time interfaces for system administrators, who can monitor with a glance the stage at which a process instance has progressed.

Because these models are easy to understand,

- **they can function as departmental APIs—visual guides of operational procedures to departmental and non-departmental staff.**
- **they can be maintained by anyone with process knowledge and platform credentials.**

It all comes down to a technical balance sheet. Where hand-coded apps written against system APIs become ongoing costs, clearly on the Debt side, low-code application

models pull their own weight. They require maintenance, of course, but, relatively speaking, very little, and once they’re built, deliver value beyond the software functionality they provide on an ongoing basis. Put another way, low-code models are technical assets.

05. Extensive Extensibility

So what’s the difference between “no-code” and “low-code”? Well, assuming there really is such a thing as a no-code platform, it would be based on the notion that every conceivable activity, form control, etc. imaginable for every system and web service in use at any given time would be pre-built by the vendor. The other alternative, of course, would be an acknowledgement by the vendor that its no-code platform simply couldn’t do everything any particular customer might need, which would tend to dampen long-term prospects.

In contrast, vendors of low-code platforms assume going in that powerful, sophisticated business applications may require something beyond what their respective platforms currently provide and make extensibility available at every level. Visually composed eForms that, under the hood, consist of HTML5 and JavaScript, can be modified however a developer wants. Existing coded objects in whatever language can be integrated into a low-code application, and programmers can develop their own activities and controls as needed.

The point with this level of extensibility is that a true, enterprise-class, low-code platform will be equipped to build just about any kind of business application you could come up with. And knowing that there really is no glass ceiling with low-code platforms is a gigantic security blanket—you’ll never have to abandon a promising, experimental application because your platform won’t go where you need it to. (Again, make sure you choose a platform that can play at this level.)

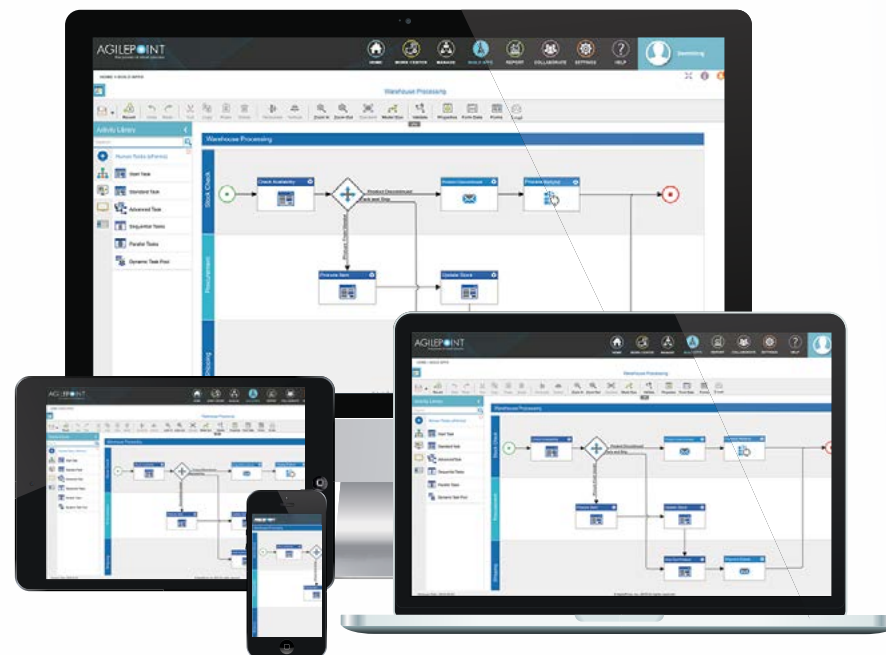


PHASE 3. Why Low-Code Platforms Are Well Suited for Mode 2 Development

06. Responsive Assets

So, you've heard of responsive websites, which will reconfigure on-the-fly to fit different devices. Well, elite low-code platforms will not only dynamically reconfigure forms—adding, resizing, or eliminating fields and other controls—to fit devices, but low code apps can be easily configured at the component level (forms/activities/controls) to fit the needs of disparate business units; can be designed, in some use cases, to self adapt to changing business conditions; and can be modified at runtime, all without coding. Put another way, a single, low-code app can be repurposed any number of different times and ways, and all instances can run in parallel. Good luck pulling that off with Java/Python/etc. on an experimental schedule and budget.

NOTE: To learn more about this level of functionality, drop us a line. We'll be happy to show you how an elite BPM-enabled Application Platform as a Service can do all of this and more.



07. Mid-Execution Asset Management

In a June 2014 publication, Forrester divided the low-code space into three sectors: general app platforms, web content platforms, and business process platforms (BPMS). The last one—BPMS—is especially applicable for most IT departments, these days, because of the need to build workflow-centric apps that are triggered by events.

And there are two key characteristics of process apps that must be noted: First, business requirements for these types of composite process apps tend to change frequently, which changes must be reflected in the apps; and second, organizational processes could take months—even years—to complete.

Furthermore, depending on your organization, you could have dozens, hundreds, or thousands of processes in mid-execution at any given time, which brings up an important question: What happens if you've got thousands of instances of long-running processes mid-execution and one or more important business requirements or conditions change?

In such a scenario, taking a process app offline to fix it is almost unthinkable—all sub processes would be orphaned, and your organization would have to start all running processes over. And yet that's exactly what you'd need to do if you had hand coded such process apps. But take fair warning—choosing a low-code platform that doesn't allow for mid-flight updates will leave you with the similar sort of Sophie's choice scenario.

The good news is that elite **BPMS-enabled** low-code platforms are built to handle exactly this sort of inevitable scenario, a fact which helps validate their Mode 2 credentials.



PHASE 4. Is There a Downside to Low-Code?

What I'm describing is the state of the art for low-code platforms, especially the ones that emphasize process-centric capabilities. But that doesn't mean there won't be challenges. Power and sophistication breed complexity. Consequently, enterprise-class, low-code platforms are the domain of power users—those who understand computing at a deep level and have the experience and skills necessary to learn and utilize the extensive features of such platforms. In other words, not just anyone can do it.

Obviously, not all platforms have pre-built stencils (collections of activities) for every software system and business application. So choose your low-code platform carefully, and beware of the fact that you may need to call on your IT department to extend platform functionality in some situations.



Bimodal IT: Innovating at the Pace of Change with a Low-Code Platform

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

AGILEPOINT

Learn More

Request Trial

Request Demo