uniform

Integration Maturity Model

A business user's guide to understanding the different types of integration & how they affect usability & productivity

If a friend tells you that the food at a local restaurant is “too spicy” or that the forecast for this weekend is going to be “cold,” unless you’re familiar with the restaurant or your friend’s preferences for food and weather, it can be difficult if not impossible to know exactly what she means. Granted, the cost of this uncertainty can be relatively minor. You may wind up with an entrée that’s too bland or that burns your mouth, or you may spend the weekend shivering or sweating a little more than you would’ve liked. But when it comes to assembling a tech stack, this sort of ambiguity can be costly. It can lead to poor technology decisions that leave a bad aftertaste for years and have a chilling effect on your ability to deliver quality solutions on time and on budget.

Integration is essential, inescapable, and increasingly important to any tech stack. But not all products integrate equally well, and the definition of “integration” can be a little fuzzy. Uniform’s Integration Maturity Model provides a clear and simple vocabulary for differentiating levels of integration that helps you to better evaluate products you’re interested in and to assemble a well-orchestrated platform.

Welcome to a multi-tool world

Let’s face it: The days of the one-size-fits-all application or suite are long gone. Virtually no one spends the entire workday within a single software environment. Marketers are used to working with multiple tools, while developers have long since grown accustomed to operating across multiple systems. When these tools and systems work together well, both marketers and developers can be more productive. When they don’t, the results are inefficient, unproductive, and frustrating. Instead of focusing on delivering the best experience for their customers, they find themselves creating data silos that limit their capabilities and kill their productivity. The conclusion is inescapable: for the sake of a customer’s experience, a company’s business, and the sanity of employees, the tools we use require tighter integration.

Some vendors are rising to the occasion

For most companies, integration is no longer just nice to have; it’s mandatory. More and more products and services are being designed with integration, not as an afterthought, but as a "first-class citizen.”

Customers' increased comfort with the cloud and with the SaaS (software as a service) products that depend on it is paving the way to composability, making it more practical and easier to adopt. SaaS assumes integration, provides robust APIs that make integration possible, and eliminates the need, cost, inconvenience, and development time that was associated with installing upgrades and patches. Once pricey and painstaking changes now happen seamlessly.

Finally, the rise of decoupled architectures like Jamstack and MACH has led to a kind of integration "perfect storm." As an increasing number of vendors build products based on these two architectures, more and more systems integrators are developing solutions that rely on them.

Others are slow to come around

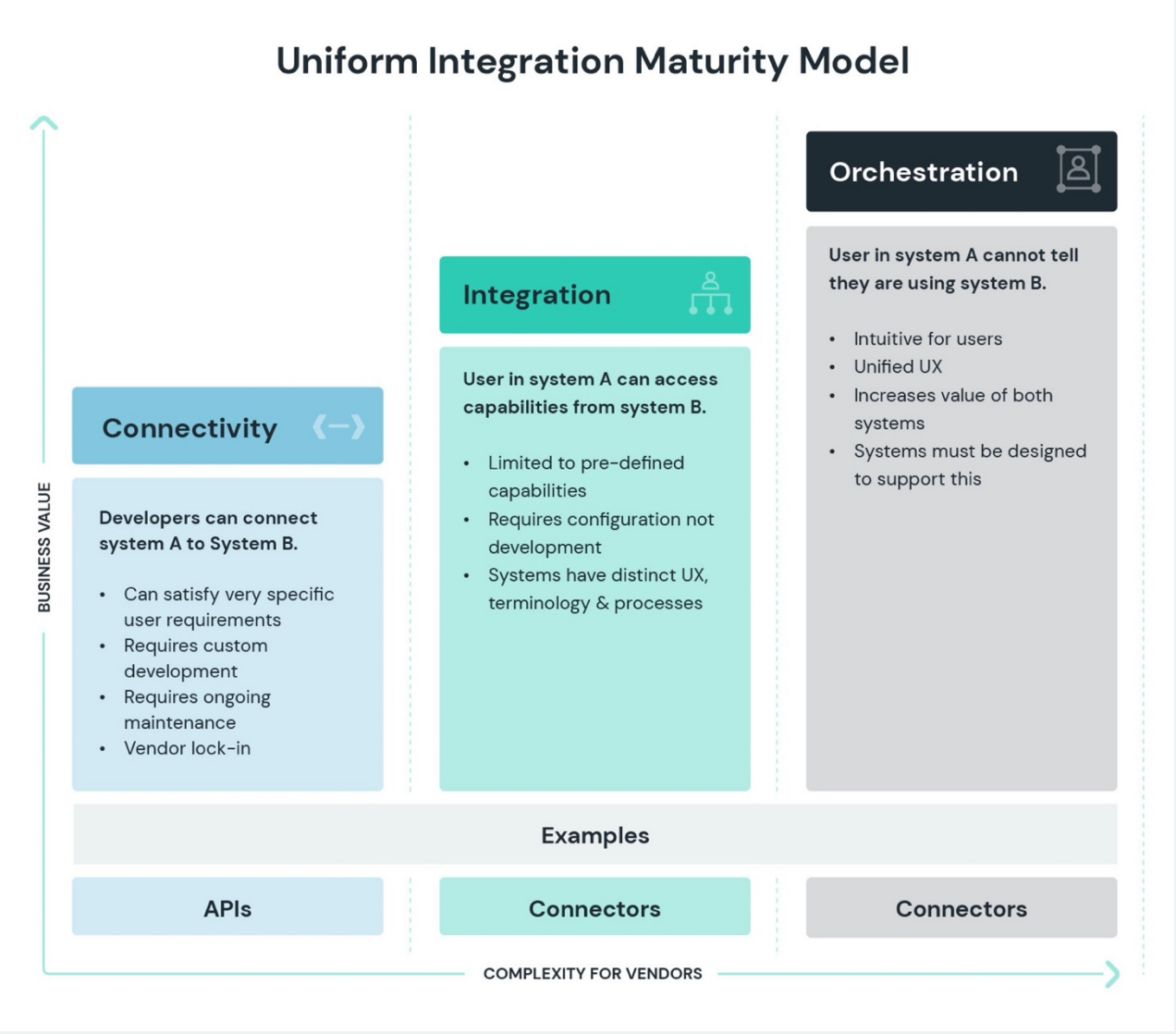
Given the growing necessity for integration, why is some integration so lackluster? One key reason is because many systems weren't designed for integration. Many vendors assumed they'd built the better mousetrap and that they had provided all the functionality their customers would ever need in a single, unified package. Unfortunately, the reality is that their systems aren't the be-all-and-end-all they originally envisioned or as "future proof" as they claimed.

Not only that, but adding integration after the fact can be difficult and expensive. Some vendors have considered the cost and the effort involved and concluded that it simply isn't worth it. As long as a suite is "good enough" to impress potential purchasers during a sales demo, they're content to let their customers pick up the cost and the complication of hiring systems integrators to add any extra functions, confident that the custom code will further increase vendor lock-in. In short, from a certain perspective, neglecting to add integration isn't a bug; it's a feature. It can actually be considered a viable business strategy for retaining customers. In addition, some suite vendors have found that they can add a flashy new user interface that creates the impression of integration, when if you look under the hood, you'll find that any actual capabilities of integration have been tacked on as an awkward afterthought.

Taking a more granular approach to defining integration

Part of the problem lies in the reckless use of the term integrated itself. Unfortunately, calling a tool "integrated" can be as meaningless as labeling a food "natural." Just as many grocery shoppers wrongly assume that "natural" means that a food is organic and doesn't contain artificial ingredients (the former isn't true, and the latter isn't necessarily true), companies that purchase an "integrated" component may assume that they can plug it into their current platform right out of the box.

The issue shouldn't be whether a tool is "integrated" or not but rather whether it is "well integrated." The time has come to add more precision to the use of the term integration. That's the goal of Uniform's Integration Maturity Model. Rather than lumping everything under the single label of "integration" the Integration Maturity Model lays out three distinct levels of integration, each one successively more advanced.



Level 1. Connectivity

The base level is **connectivity**. This simply means that a connection can be made from one system to another. Typically, this depends on APIs the vendor has provided that enable developers to build connections to third-party tools. Unfortunately, this method has some conspicuous drawbacks. Because the vendor defines the API and supplies no user interface, adding functionality or a UI requires costly and time-consuming custom development and increases the risk of introducing potential bugs. It not only adds to vendor lock-in, but also to *version* lock-in. Upgrades, even ones for bug fixes or security patches, may not be possible unless the custom code is updated as well.

Level 2. Integration

Next comes **integration**. At the integration level, users can access one system while working within another. This is accomplished with pre-built components called *connectors*. Unlike APIs, which offer only the *potential* for functionality, connectors provide *actual* functionality. Of course, connectors only work with the systems they support. Even if you're lucky enough to be working with a supported system because the connectors are pre-built, this means you are limited to certain predefined capabilities.

Difference isn't superiority

It is important to note that while the Integration Maturity Model categorizes the different kinds of integrations that exist, this does not mean that one type of integration is inherently "better" or "worse" than another. Each type of integration has a place in a system, and modern systems couldn't function without a mix of all of them.

What is important is that you understand that these different categories exist so that when you are making a technology buying decision you know what you are getting.

Level 3. Orchestration

Finally, the most mature of the model's three levels of integration is **orchestration**. The essence of orchestration is extensions. An extension does just what it sounds like. It extends a system by tapping into features already present that allow developers to add functionality. Although a third-party vendor may have built the extension, it operates almost as though it were part of the original component. As a result, the user feels as though only one system is being used. That said, the goal isn't to hide the fact that there are multiple systems. It is to shift the focus away from the connection between systems and on to the task at hand instead.

The ambiguity of describing an entrée as too spicy can be resolved by the Scoville scale, which measures the pungency of peppers. As for determining what someone means by “cold,” we have the Fahrenheit and Celsius scales to help clear up any confusion. Now we have the Integration Maturity Model to help us to better understand what “integration” truly means.

This more granular approach offers numerous benefits. By distinguishing between types of integration, it helps you to better understand exactly what you are buying, how much custom development will be required, and how "future proof" your architecture will be. With this model in your toolkit, you can choose the tools you need with increased confidence and avoid decisions that put a bad taste in your mouth or leave you feeling that you’ve been left out in the cold.