avanade

# Go Native

Making Optimal Use of Native System Functionality

LAWRENCE MATUSEK

Do what matters

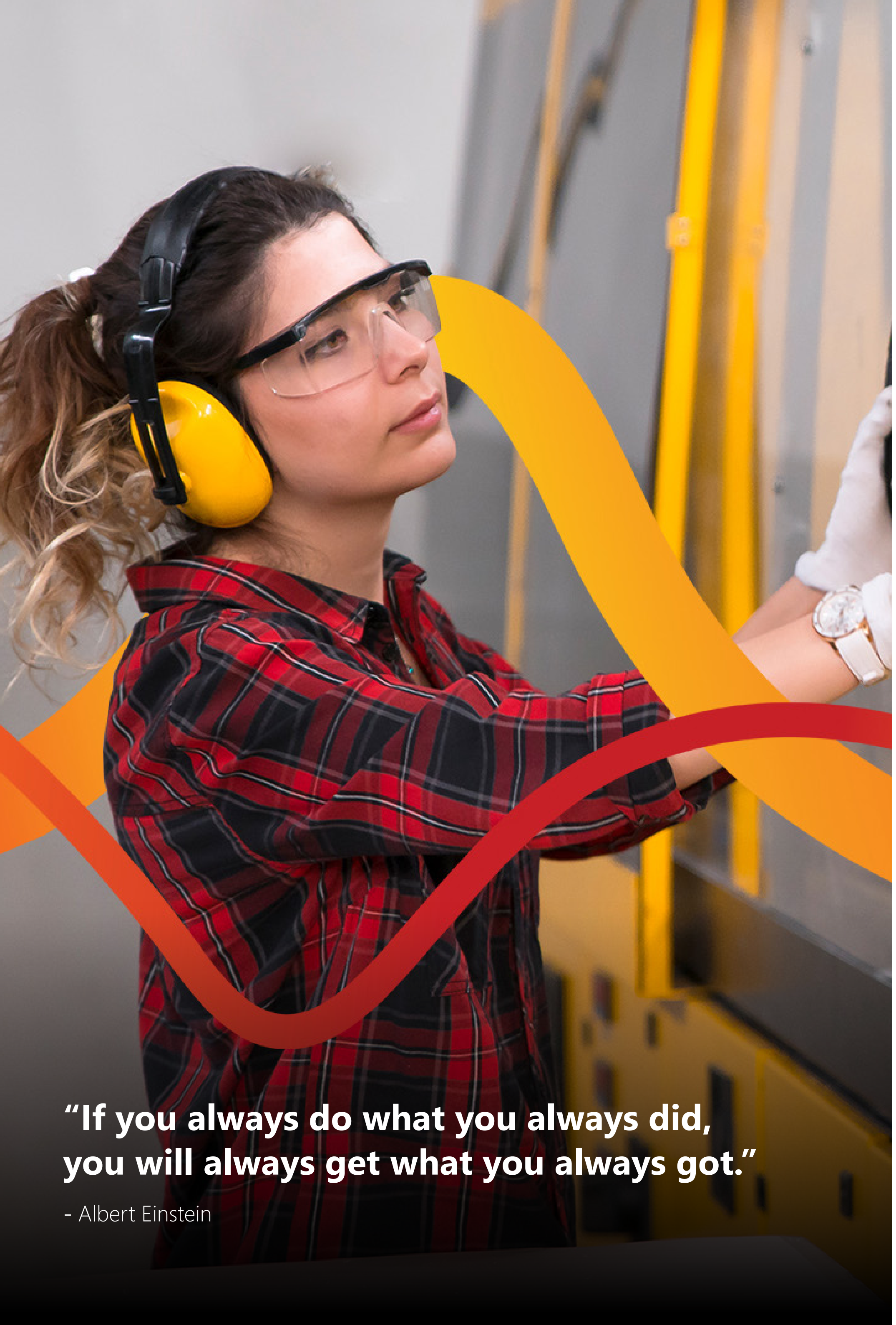
# Introduction to going native

“I have been implementing packaged business systems (ERP, CRM, PLM, etc.) for more than 20 years. Over that time, I have seen some companies implement very successfully, and many others who struggle mightily. Why the difference?

When I think about the most critical success factor, it is clearly, **making optimal use of native system functionality to address the widest range of business needs and opportunities.**

They then make limited enhancements to gain or secure competitive advantage. In a nutshell, we say to **GO NATIVE!**

This eBook serves as an introductory guide to leveraging native functionality in packaged business systems. I will explore challenges and benefits of implementing these systems the way they were designed to be implemented.”



“If you always do what you always did,
you will always get what you always got.”

- Albert Einstein

# Avoid the Legacy Trap.

## “At some point, every business system reaches the end of its useful life cycle.”

The end can be precipitated by internal influences… like a company outgrowing its system’s capability or perhaps the merger of two companies needing to consolidate operations into one system. **With time, the end will inevitably be precipitated by continual architectural advances in hardware and software or ever increasing expectations around user experience and system features or capabilities.**

Eventually, your company will decide to replace each business system with a new one. “How should the new system be configured to meet your company’s current business requirements, just like our legacy system?” you say, “After all, we know that it works. We have been through all the requirements that led to the processes that led to the system implemented and refined over the years. **Our legacy system is the embodiment of how our business runs.”**

I have been in a discussion like that many times. First of all, a legacy system is the embodiment of how a business ***ran***. Over many years of usage, the business’ requirements, processes, and system can become incorrectly viewed as one and the same. In fact, when the legacy system was implemented, your processes were shaped in part by that system’s capabilities and limitations. When replacing it, you must take a fresh look at your business requirements and determine what processes are now possible with the latest generation of business system functionality.

**This is where you must be on the lookout for the “legacy intransigents”.**

## Your new system should enable you to take your business to the next level.

Some people become quite emotionally attached to the way that their legacy system works. They find comfort in the familiar, the tried and true. They know where everything is and how it works. New systems are unfamiliar and rather intimidating to them. They view their legacy system through rose colored glasses. Long gone are their memories of when the legacy system ***was*** the new system, how its implementation was just as challenging, and when it wasn't quite the panacea that it is today.

**Be mindful not to fall into an all too familiar trap of trying to use a patch-work of workarounds and "fixes' that only give you a perceived "new and improved" version of the same legacy system.** This outcome is actually the worst of both worlds. You end up with a highly customized new system that isn't quite as quick or easy to use as legacy was, and it is more difficult and costly to maintain. Worse still, most of the new capabilities and features that prompted the purchase and implementation of the new system were likely not used because they did not exist in the legacy system. **In my experience, the most egregious examples of "falling into the legacy trap" occur in product configurator implementations**

For example, I have been a solution architect or consultant in over 100 SAP ERP variant configuration projects. Granted most of these were small projects and my role was mainly design, but nevertheless, I have seen too many instances of customers trying to force fit legacy configurator constructs and logic into SAP's configurator.

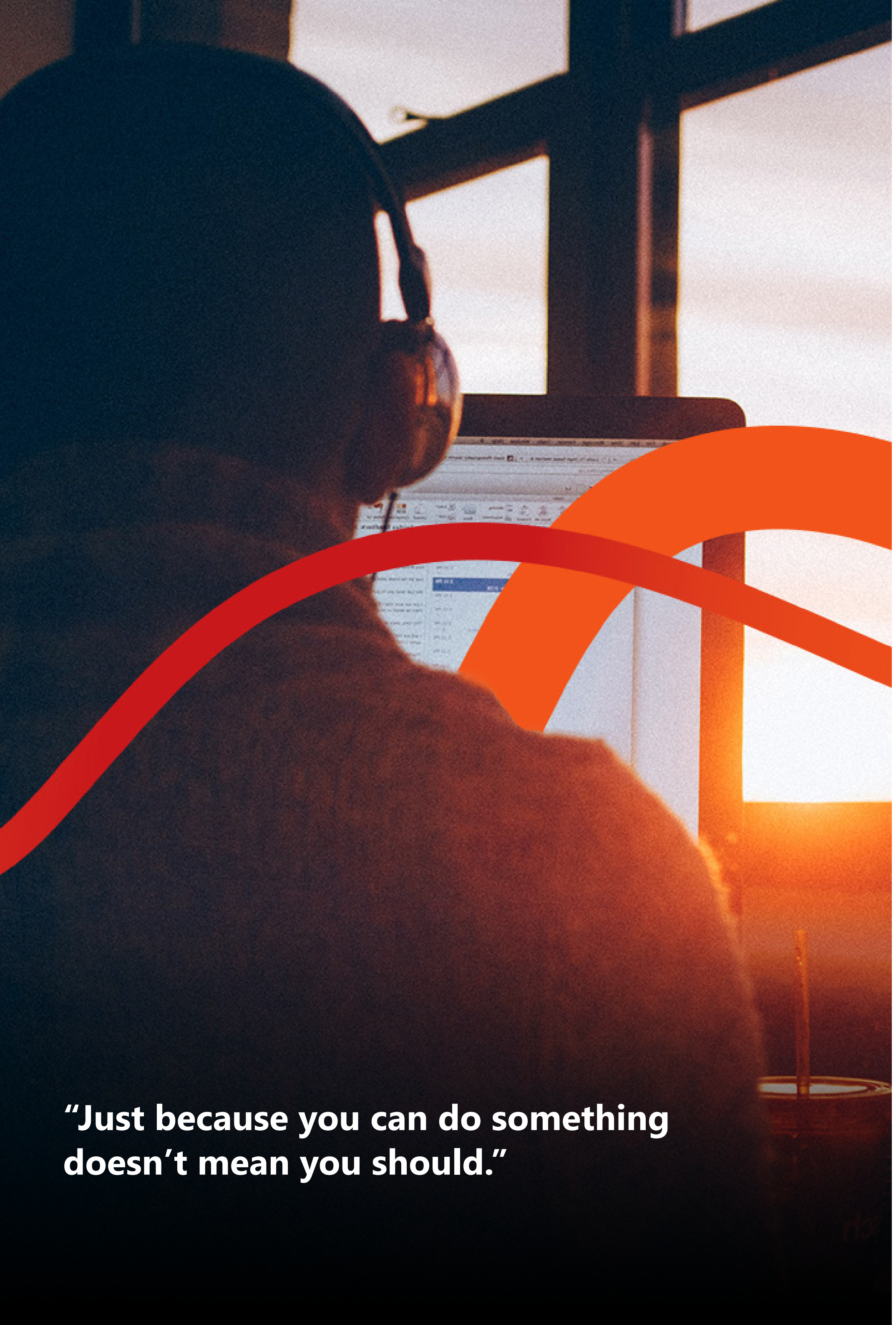
**The resulting models typically have so many cryptic rules and design obscurities that it becomes quite difficult for anyone not deeply involved in the implementation to gain a comprehensive understanding of the overall solution.**

## So if legacy isn't the path to the future, then what should you do?

**First and foremost, learn the architecture, capabilities, and native functionality of your new system** so that you can make informed implementation decisions. Identify your best technical and functional people and give them ample time to explore, experiment, and learn. Give them access to experienced consultants with deep domain and/or system experience. Keep your team small while ambiguity is high.

**Start with a proof of concept or pilot project** where you can afford to go through the necessary cycles of learning. This isn't "throw away" work; the time you will save and missteps you will avoid down the road are truly invaluable.

**In the next chapter, I will talk about the dangers of needlessly enhancing a new system instead of leveraging its advanced native functionality simply because the necessary learning described above did not happen. This "do-it yourself" approach occurs all too often and leads to an array of problems. In fact, a significant portion of our business has come from rescuing such failing or failed implementations. All other things being equal, I would gladly give up this "rescue" business in favor of helping companies do it right the first time around.**


“Just because you can do something doesn’t mean you should.”

## Resist the Temptation to Optimize

### Most everyone is familiar with the “80/20 rule”.

In the context of business system implementation, it suggests that at least 80% of your business needs should be addressed by the native functionality of your business system while the other 20% or so must be addressed through customizations and/or manual processing.

**Hopefully your business system has obvious native solutions for most of your needs, but what should you do when a business need arises without an obvious native solution?**

Whatever you do, you must resist the temptation to develop a custom solution before fully investigating native alternatives. All too often I have seen customers develop custom reports, transactions, etc. when native functionality would have provided a better and more extensible solution.

## I have seen two general reasons why customers have developed a custom solution instead of leveraging a native one....

### 1 Lack of detailed knowledge about the native functionality (Or how to use it)

The first reason is typically the more difficult to overcome. It can be avoided with leadership from solution architects who have a deep understanding of the business system architecture and its design philosophies PLUS the skill to match business needs with native system capabilities (i.e. a fit/gap analysis). These individuals possess several key characteristics and abilities like the following:

**Natural Curiosity:**

e.g. what does a given feature do and in what situations would one want to use it?

**Translation:**

i.e. connecting business needs to functionality even though terminology may be different

**Pattern Recognition:**

e.g. this need is similar to one that was solved using a certain native functionality

**Resourcefulness:**

e.g. finding ways to use leverage native functionality in creative ways or combinations

**Gap Identification:**

i.e. determining when a need cannot or should not be met with native functionality

### 2 Perception that a given custom solution is easier to implement or use than the native one

The second reason is usually more of a corporate culture or training challenge. Many people have an inclination to "reinvent the wheel" or else that phrase would not be so commonly used. This mentality is rather self-defeating when implementing a business system. It can result in a lot of unnecessary effort and expense. I have seen people offer an array of excuses for justifying such customizations – see some examples below along with my typical rebuttal.

**"The system doesn't do it."**

Where is your analysis and proof? Or is it more accurate to say "I don't know how to do it in the system"?

**"I have to  see it like this."**

Have you tried to view it the native way? Can you quantify the additional effort or risk for using the native transactions instead? By the way, you will need that to justify a customization.

**"We must have this report."**

Legacy systems often need custom reports because they lack native reports and transactions to see data that most modern systems naively provide. Another key metric is how often a given custom report is actually used. You might be surprised to learn how infrequently some "must have" reports are generated.

**"We have always done it this way."**

This is perhaps the quintessential excuse. Unless your business did not exist before the computer age, it is probably more accurate to say "I have always done it this way". Or are you saying "a customization.

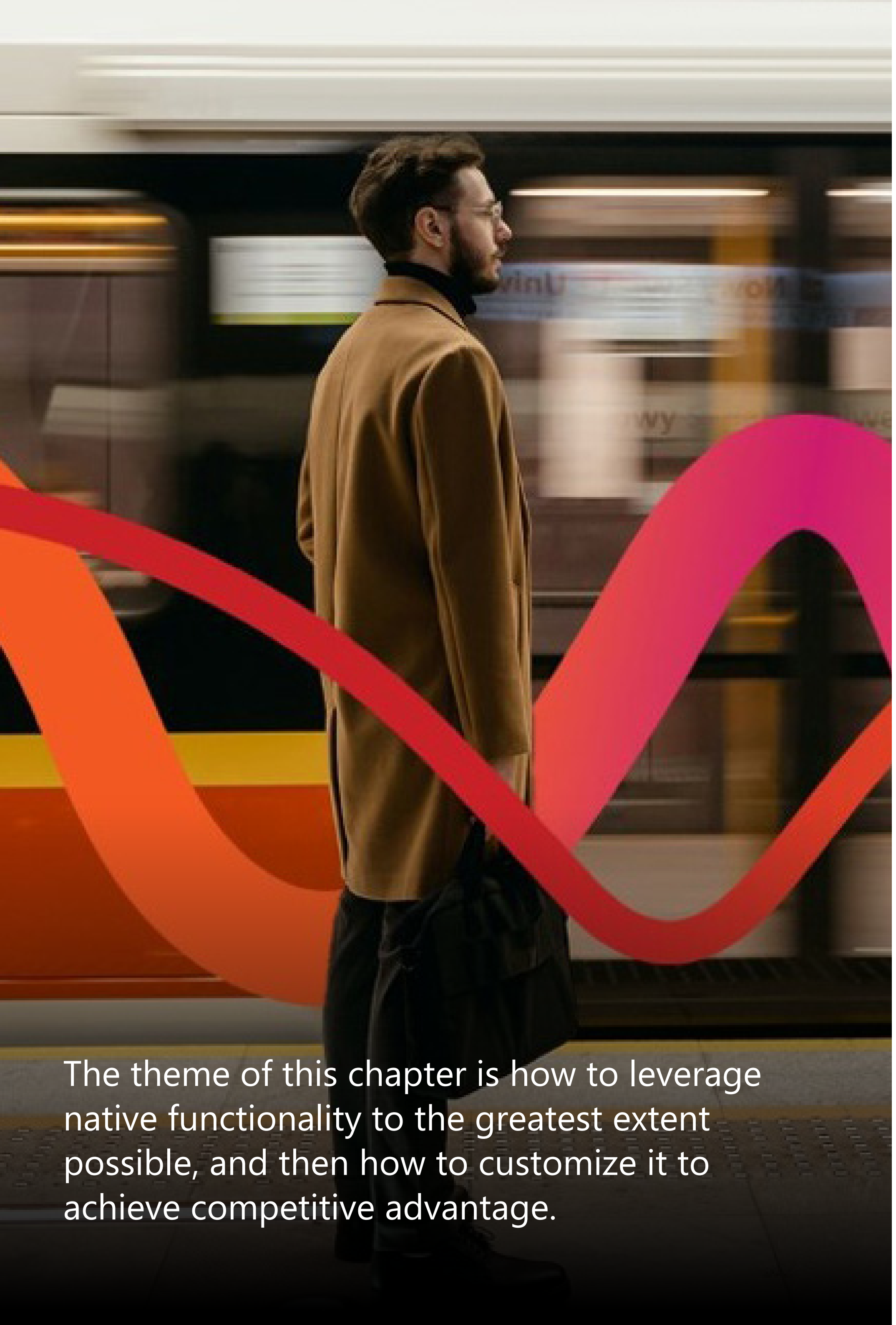
Despite the question of why you would want to (re)build functionality that you have already licensed AND pay a vendor to maintain, there is another point you should consider.

**Customizations are often designed and tested to work with only the SUBSET of functionality that your company has implemented. All native functionality is designed to be fully integrated and supported across the entire system with few limitations.**

Improperly implemented customizations can inadvertently limit your ability to use advanced native functionality or upgrades in the future.

A significant portion of our business comes from "retrofit projects" in which we implement or properly re-implement native functionality in situations where the expected ROI for a business system was not initially achieved. In such projects, existing customizations are the first thing that we evaluate to determine the following.

- Which customizations can be reduced or eliminated using native functionality? The more the better.
- Will newly implemented native functionality work properly with existing customizations? We routinely find poorly implemented customizations that prevent native functionality from working properly. They must be corrected or rewritten.
- Will existing customizations continue to work with newly implemented native functionality? This is usually the largest and most difficult effort to quantify because many existing customizations can be complex, improperly scoped, and/or understood by only their original authors.



The theme of this chapter is how to leverage native functionality to the greatest extent possible, and then how to customize it to achieve competitive advantage.

# Mind The Gap

**When customizing a business system, you should strive to keep your customizations as succinct as possible so that they work reliably through future system upgrades.**

For those who are not familiar with "the tube" (i.e. the subway) in London, you may be wondering about the title of this chapter. "Mind the gap" is a warning repeated to passengers to take caution while crossing the gap between the station platform and train car. Under normal conditions, the gap should be small and easy to cross but you must be careful to avoid twisting your ankle as you step in or out.

**For more details**

I chose this title because it is critically important to "mind the gap" when customizing business systems. No business system can or should attempt to address every single business nuance with native functionality. Well architected systems typically do a good job of providing a very robust functional framework that anticipates where customers will need the ability to provide custom logic in the form of enhancements.

Enhancements are used to close the small functional gaps that you encounter during your blueprinting or realization activities. They are not intended nor should they generally be used to fill gaping functional holes. If you find many of such holes during the fit/gap analysis in your software selection process, then you are likely evaluating a system or type of system that is a bad fit for your business needs.

**Mind the gap! But how?**

There is a lot of wisdom in this quotation. My corollary is that you can find 100 or more ways to solve any given problem – but probably only a few of those ways are good ways – and the simplest way is usually the best way.

## Paradoxically simple solutions can often be the most difficult to find.

One of my favorite anecdotes is about a tractor trailer that was driving under a bridge and became stuck because of insufficient clearance. The trailer was so tightly jammed that it could not be moved. Engineers and experts were called to the site to figure out how to free the truck. They devised all sorts of elaborate plans to raise the bridge, cut the trailer top off, dig under the truck, etc. A child walked up to the scene and asked "why don't you just let the air out of the truck's tires?"

As in the example, I instinctively know when I have found the simplest and most elegant solution to a problem. It just feels right and makes perfect sense. So how could something be simpler than possible? That just means that you didn't entirely solve the problem but instead gave a simplistic solution that has limitations and didn't fully meet the requirements.

In most cases, this is a less than satisfactory outcome. You shouldn't "dumb down" a problem just because it is difficult to solve, however it is always prudent to ask whether an elaborate solution is really required and whether it will actually be used as expected. quantify because many existing customizations can be complex, improperly scoped, and/or understood by only their original authors.

**"Everything should be as simple as possible, but not simpler."**

- Albert Einstein

## When you determine that you should or must enhance your business system, how do you know if you have done so appropriately?

I have seen a lot of enhancements in my experience – some good, many bad. I use the following three simple rules of thumb to evaluate enhancements.

### Question — How many lines of code and how many lines of comments?

**Rule:** IMHO, the best customizations generally have a few lines of intuitive executable code. In addition, they have lots of comments that explain how and why the customization will work in the necessary context (and how it otherwise intentionally fails or is bypassed).

### Question — How many database queries vs. API calls?

**Rule:** The best customizations can organize native functions in a specialized manner or process. This approach effectively extends native functionality. On the other hand, it is sometimes necessary to directly query the system database when no API is provided for a required function. Be careful to fully understand the table keys and relationships in such queries.

### Question — Does it stay "between the lines"?

**Rule:** Business systems are designed to be customized within limited scopes and using specific techniques. The greatest danger in enhancements is that they can introduce unexpected or unpredictable behavior into normal system operation

**Be careful about designing and testing enhancements to work with only the subset of functionality that your company has implemented so far.**

While this may seem like a smart time saver at first, poorly implemented customizations can inadvertently limit your ability to use advanced native functionality or system upgrades in the future. Ideally your enhancements will work properly with the full range of native functionality, but if not then at least ensure that they gracefully trap situations beyond their original design and can be extended to support future needs

## How does one learn to implement good enhancements?

In my experience, quality training for enhancing business systems is hard to find. The training that is available is all too often focused on the mechanics of the functionality, i.e. definitions and explanations of features and functions and how they work. Think of the analogy of learning about the various tools in a toolbox – does that training make you a skilled carpenter? Not by a long shot. I can personally attest to that fact. Both of my grandfathers were skilled carpenters. They showed me how to use every tool, but let's just say that my carpentry would never be mistaken for expert craftsmanship. While learning the mechanics of a business system is an obvious place to start, that shouldn't be the end game. In practice, I have seen far too many customers send their team to one or more weeks of "mechanics" training and then have them return to design and implement enhancements for addressing complex business needs. You can probably guess how this typically works out.

Perhaps I can illustrate by continuing my carpentry analogy with this quotation:

**"If all you have is a hammer, everything looks like a nail."** - Bernard Baruc

**Perhaps the worst thing you can do is learn a few example enhancements and then start trying to apply those design patterns to every enhancement need that comes along.**

Perhaps the worst thing you can do is learn a few example enhancements and then start trying to apply those design patterns to every enhancement need that comes along. We call that the "blunt instrument surgery" or "bull in the china shop" approach. Implementing good enhancements requires a lot of knowledge and finesse. You should consider tapping your colleagues or user groups for ideas and experience in closing similar functional gaps. Chances are that your company is not the first or only to face a given customization challenge.

In my opinion, on the job training is usually the best and only way to really master the design of business system enhancements, but not all experience counts equally.

Make sure that someone claiming "ten years of experience" truly has ten years of broad and progressively more challenging design work (as opposed to repeating a year's worth of the same type and complexity of design work ten times).

**North America**
Seattle
Phone +1 206 239 5600
America@avanade.com

**South America**
Sao Paulo
AvanadeBrasil@avanade.com

**Asia-Pacific**
Australia
Phone +61 2 9005 5900
AsiaPac@avanade.com

**Europe**
London
Phone +44 0 20 7025 1000
Europe@avanade.com

avanade

**Do what matters**