Things You Should Know

25 Lessons I've Learned About Selecting Content Technology and Services

Table of Contents

About the Author

Deane Barker is a veteran of the content management industry with decades of experience evaluating, selecting, implementing, and managing content solutions.

When this book began, Deane was the Chief Strategy Officer and consulting analyst at Blend Interactive, a content management consultancy.

As this book continued to develop, Deane joined Episerver, a content management and digital experience software vendor, as their Senior Director of Content Management Strategy.

Deane lives in Sioux Falls, South Dakota with his wife, Annie. They have three children, a dog, and a cat.

Other books by Deane Barker:

Web Content Management: Systems, Features, and Best Practices

Real World Content Modeling: A Field Guide to CMS Features and Architecture

Introduction

I've been working with content management systems (CMS) for almost 25 years. I found the Internet as a college student in the mid-90s, and started web programming not long after. This path spared me from three years of law school, and it spared the world from yet another uninspired lawyer.

What we now call "content management" used to be just a bunch of Perl scripts and flat files. I've seen the industry grow from the crudest of solutions to today's technical advancement.

I spent some time in corporate IT, working as an intranet and website manager, mostly in finance and banking. During this time, I managed enterprise purchasing projects, so I was on the receiving end of sales pitches.

In 2005, I co-founded a content management consultancy called Blend Interactive. I spent the next 15 years in professional services doing consulting on content management problems. In this capacity, I participated in CMS selections both as a service provider (I was selling services to a client) and as a selection consultant (I was helping a client navigate that same process).

In late 2019, I went to work for Episerver, a software vendor, as their Senior Director of Content Management Strategy. Blend Interactive

was their first North American partner (11 years at the time, and counting), so I was deeply familiar with the product and the company. My job at Episerver is to plan the long-term future of the CMS product, and work with the sales and marketing teams to increase the visibility and sales around CMS.

Over the years, I've seen the CMS buying process from almost every angle. During this time, I've learned a few things about the applicability of a particular CMS to a particular situation, the human dynamics of these projects, and how software gets sold by vendors and integrators.

In the middle of a CMS selection process in Detroit in 2019, I started writing down some observations. I would see the same patterns repeatedly, so I decided to keep track of situations that could be helpful to explain to potential customers. This process continued for about 18 months.

In 2018, I finally got around to fleshing them out a bit. What started as a simple blog post became a 20,000-word polemic on the state of enterprise software purchasing.

I eventually set the writing project aside. It had become too long to be a blog post, and I honestly wasn't sure what to do with it.

After joining Episerver, I thought perhaps we could use this as a contact point with organizations who were embarking on a buying process and wondering how it was going to play out. For the uninitiated, these projects can be intimidating, and there are things many of us wish we knew as a buyer and that prospects should know before they embark on their process.

While I work for a software vendor now, know that the bulk of this book was written long before I switched sides. This book was solidly written from the customer's perspective with the goal of helping them understand the larger process, the implementation project that's bearing down on them, and the experiences and motivations of the other parties.[1]

The dynamics of these processes is tough to capture in a business book. Addressing a business audience usually means writing with a tone of detached sterility and the generic positivity of a self-help guru. Additionally, every book has to make assumptions about your history, your capabilities, and the culture of your organization.

Unfortunately, most business books make the wrong assumptions. They assume your organization is rational, your project is well thought out, and everyone in your company gets along and works well together. The result is alienating for anyone who doesn't fit into those assumptions...which is most people.

Technology projects are complicated, even under the best circumstances. Some problems are simple and clear – if you're building a system to manage bank accounts, for example, at least you have clearly defined tasks and rules for success. And since the project is solidly in the "line of business,"[2] you have a strong mandate and strong support.

But marketing and digital content are different disciplines.

First, they're inherently subjective. The success metrics aren't clear cut, and there are innumerable paths your organization might take. Internal debates can rage, not only about the journey but also the destination – different people and groups might not even agree on the goal everyone should be working toward.

---

1. Note that Episerver had no input into the content. They never tried to censor or remove anything in these pages – even things that might not reflect well on vendors. What you're reading is the document I wrote, without any vendor filter.

2. "Line of business" is an activity critical to the mission of the organization. If you're a bank, it's the systems that manage customer accounts. If you're a manufacturer, it's the systems that support the creation and shipping of products. In most businesses, the website is a marketing concern and isn't required for the generation of revenue.

Second, the main purpose of an organization is not to market itself. Given that CMS and marketing functions are not a line of business concern, many organizations consider these projects to be outside the core mission and they're left to fight for resources and attention.

Therefore, the assumptions for this book are something like this:

- You might not have a clear mandate from the organization. Instead, you're being asked to fix a problem which is vague and ill-defined.
- Not everyone in your organization might be in agreement about the goals, methods, or even the necessity of your project.
- You might not have strong budget support, or even any clear idea of the budget ballpark in which you're playing. You might have an unsettling feeling you're going to have to fight for funds at some point.
- You might be highly dependent on other parts of the organization for services, and you might have no control over these groups. You might feel like you don't have all the human resources you need to get the job done.
- Your organization might suffer from some level of dysfunction, ranging from simple poor communication to more sinister scenarios like outright hostility and turf wars.[3]
- You might have little experience with technology selection or professional services vendors.
- You might simply feel like you're in *way* over your head.

3. Your organization might fit somewhere on the scale of the Capability Immaturity Model (https://en.wikipedia.org/wiki/Capability_Immaturity_Model), which ranks dysfunction by levels ranging from Negligent, Obstructive, Contemptuous all the way to Undermining.

That list feels cynical. Know that I'm not trying to black pill[4] or savvy[5] you, but I'd rather get that out in the open now, in hopes you understand I'm not making rose-tinted assumptions that your situation is a case study in efficiency and competence.

The real world isn't a breezy, self-promotional update that someone writes on LinkedIn. The real world is often messy and imperfect.

The lessons contained in this book are deeply varied in scope – some are overarching principles, while some seem like minutiae. Some are practical application, and others border on organizational psychology. Some positive and productive, some feel like small glimpses into human nature, and some feel like I'm peeling back layers on a flawed system with some bad actors.

Also know that when I use the word "lesson," I don't mean it in the form of "this is a lesson I need to teach you." Rather, the context is, *this is a lesson that I have learned*. Some of this, you may know. But this is primarily a collection of things I've discovered after 20 years working in this industry that might be helpful to you as you embark on your own project.

Unquestionably, this book is a collection of arguable opinions. I'm intentional about using the "I" pronoun all throughout, not out of narcissism but because I want to emphasize that these are the opinions of a specific person speaking from a specific perspective and history. Others might argue with what's written here, and that's fine. I can only speak to my own experience.

Finally, to be clear, this is not a start-to-finish guide to a selection process. It's not a comprehensive plan for anything. This book is a

---

4. *black pill* (noun or verb; slang): A catastrophic prophecy or spiritless prophesying for the future that is not necessarily grounded in reality.

5. *savvy* (verb; slang): A blanket dismissal of concerns or beliefs accompanied by the feigned nonchalance of someone with vastly more experience in a discipline.

series of anecdotes and asides describing things you should know before you get started, hence the title.

You can probably extract and distill an implied process from what I've written here, but if you're looking for a full script, I'd recommend one of these books:

- *The Right Way To Select Technology*, by Tony Byrne and Jarrod Gingras (Rosenfeld Media)
- *Off-The-Shelf IT Solutions: A Practitioner's Guide to Selection and Procurement*, by Martin Tate (BCS, The Chartered Institute for IT)

Those two titles outline a more formal – and less cynical – process to selecting a software platform.

That said, here are – in vague-but-by-no-means-ideal order – the lessons I've learned about buying and selling both CMS software and the services surrounding it.

I hope this helps you.

Deane Barker
April 2020
6 7

---

6. With sincere apologies to Bill Simmons and David Foster Wallace, I tend to write a lot of footnotes. I do this because my mind wanders off onto tangents, and trying to fit that all into the main narrative would be like forcing you to do algebra in a room filled with toddlers all hopped up on Pixy Stix.

7. That was what we'll call a "meta footnote," meaning it was a footnote about footnotes. It's meant to be ironically self-aware and therefore hilarious.

Applicability to Software Genres

My professional experience has mainly been with content management systems. However, as the subtitle indicates, this book is directed toward "content technology" in general, which includes:

- Content management
- Digital asset management
- Marketing automation
- Content recommendation
- Email marketing
- Content analytics
- Commerce and product marketing

These systems have a common link: they help organizations create and manage a human connection. As such, they tend to be more subjective, and they share common reasons for adoption and common challenges to usage and evaluation.

The lessons in the book should apply equally to this wide category of software.

Lesson #1
There is usually a familiar roster of players

Before you start planning, researching, or evaluating, let's identify the groups that you'll see most often in a technology and services selection process. I'll talk about these players throughout the rest of this book, so this is your cheat sheet for later on.

Most often, there are three main groups that are in it for the long-haul. These are the organizations at the heart of the selection process:

- **Customer:** The organization doing the buying. There might be a lot of different parties within the customer organization, but we'll group them all under one banner. This is probably you – the reader of this book.
- **Vendor:** The organization selling the software or, in the case of open-source, the software itself (more on this later). In a competitive selection, there will be more than one vendor involved.
- **Integrator:** The organization selling services to implement the software. This is often interchangeable with the term "partner," which means an integrator who works with a particular CMS (they're a "partner" of that software vendor).

This trio of core players is quite common.

However, sometimes the vendor and the integrator are the same organization, for various reasons:

- The vendor itself offers professional services. There are software companies that integrate their own software. Sometimes, in fact, this is where they make most of their money, and the software is just a handy way for them to sell services.
- The integrator might be building a custom CMS or integrating a CMS they built and use only for their customers. This is just a slight variation from the above, and it's rarely a good idea, unless the requirements are deeply specific.
- The software might be open-source. The integrator could be proposing software that has no selling organization behind it. So, while the integrator didn't write the CMS software (they are not the "vendor," strictly speaking), they are representing it and promoting it for this project.

In addition to the core players, there are sometimes other groups around the table.

- **Selection Consultant:** A company or person the customer has hired specifically to help them pick a vendor and/or an integrator. This is sometimes provided to the customer through a subscription with an analyst firm.
- **Hosting Provider:** Sometimes the customer self-hosts on their own on-premise or cloud infrastructure, but often the hosting is done by the vendor (for "cloud" solutions[1]) or the integrator.

1. "Cloud" is a vague term, but for our purposes, it means software that is hosted and managed by the same company that produced it. You don't "purchase" this software, rather you rent it as a service. This is becoming the predominant model of enterprise software acquisition.

Rarely, you might have a separate hosting provider, sometimes because they specialize in the customer's market (there are companies that specialize in health care hosting, for example).

- **Design Agency:** A company contracted solely to do the initial market research and design. This company's role is usually finished when the design is approved and handed off.
- **Marketing Agency:** A company engaging after launch to run marketing campaigns using the completed website.

Those last two are often part of the integrator's service offering – some integrators are so-called "full-service agencies" that do everything involved in a project, both for launch and beyond. Other integrators just do initial CMS builds and ongoing development, and additional providers come in to provide marketing support after launch.

These other service providers are outside the core because they're often not present during the purchasing process. If a customer is using an outside design or marketing firm, those firms only occasionally have significant input into a software selection.

Also, when engaging with a vendor, understand that there will likely be multiple people who have different roles:

- You'll likely encounter a salesperson (a "seller"), and they'll be paired with a more technical resource on the opportunity. This person will usually be a developer with the title of "Solution Architect" or "Sales Engineer." They'll run the demo, and mock up scenarios that you'd like to see. The technical capability of sellers varies greatly – some of them actually know relatively little about the underpinnings of the product, and were hired for their ability to build relationships and communicate, not their knowledge of how the system works.
- Once you purchase the software, you'll likely be handed off to a different seller at the vendor. At most organizations, sellers are segregated by "new business" and "existing/relationship busi-

ness." New business gets you in the door, and then an "account manager" works with you over time, with the goal to sell you more product. In many cases, they also do double-duty as customer service. They're often your first point of contact when you have an non-technical issue that you can't resolve.

There can be a lot of people to keep track of, and it speaks to another dynamic: when you embark on a project of the scale you might be considering, you are initiating a *lot* of relationships. A key role of the project or product manager's job is managing those relationships. You need to be explicit about roles, responsibilities, and lines of communication. Make sure everyone knows who is the main point-of-contact at every stage of the project.

Lesson #2

There is no "soulmate" for your project, and all that glitters will eventually lose its shine

Finding the right CMS is kind of like finding the right person to marry – we have fantasies about our one true love and our soulmate, but the reality is less romantic: there are often a lot of options that will work, and you might be equally happy with any one of a number of different partners.

The *New York Times* was even more cynical about marriage back in 2016 when they said:

> [We must abandon the idea] that a perfect being exists who can meet all our needs and satisfy our every yearning.[1]

In any genre of software, systems are give and take. One might be very good at X, acceptable at Y, and terrible at Z. Another one is

1. *Why You Will Marry the Wrong Person (https://www.nytimes.com/2016/05/29/opinion/sunday/why-you-will-marry-the-wrong-person.html)*, New York Times, May 28, 2016

moderately good at X and Y, and stellar at Z. Another one may be amazing at X, Y, *and* Z, but it's more money than you can afford.

A CMS is a complex adaptive system. As such, it might not even be possible to get everything you want in a single system.

Economist Thomas Sowell is famous for saying, "There are no solutions, only trade-offs." Getting better at one thing might mean getting worse at another. Improvements in content modeling, for example, might reduce simplicity and clarity which raises implementation and training time.

When you evaluate a CMS, don't ask yourself "is this the 'right' CMS," as if there's a simple answer to that question, because oftentimes, there's not. Just ask yourself which CMS has the best chance of working for your project.

Alternately, ask what CMS you can see your team being successful with. You might have more than one answer here, which means your decision can fall back to secondary factors like pricing, contractual compatibility, and services scheduling.

Though vendors will try hard to convince you otherwise, many systems are simply interchangeable. There are common patterns to any genre of software that vendors tend to gravitate toward, and rarely is any one vendor's product genuinely revolutionary compared to others. Differences tend to be incremental, not paradigm-shifting. Additionally, differences are often spread throughout the system as small advantages and drawbacks in dozens of areas.

When large differences do appear, they're often compartmentalized to specific aspects of functionality that a vendor happens to excel at.

For example, a particular vendor might be good at handling rich media – images, video, audio, etc. – especially if their product was generalized out of a more specifically targeted system. Perhaps their product started life as a digital asset management (DAM) system, and was then modified into a more general CMS. In this case, it will likely excel at rich media but suffer correspondingly in other areas.

This is because vendors move in packs. One vendor will come out with New Feature X, which they'll evangelize to the market. However, a vendor can only maintain a competitive advantage in any particular area for so long before other vendors take notice that this has become The Next Big Thing™. Then they'll all start to replicate it (if they weren't already).[2]

I remember a conversation with an Episerver architect back in 2008 about an idea to modify content based on the real-time behavior of the visitor. That idea became Episerver's Visitor Groups, which turned into a foundational feature of the product.

Within a couple of years, everyone was doing this – all of Episerver's competitors had jumped on the personalization bandwagon, and capturing visitor behavior and context was *the* thing every vendor wanted to show in a demo. This is just the way the market was moving back then, and all the vendors stumbled out that gate at roughly the same time.[3]

Anonymous personalization was unheard of, then it was leading edge, then it was the only thing vendors wanted to show, then it became expected, and, finally, it was boring. And we all moved onto the next thing.

The larger point is that any competitive advantage is fleeting, and it has to be re-established by moving onto The *Next* Next Big Thing™.

---

2. This is not just in software. In the book *The Omnivore's Dilemma*, an executive from food company General Mills says, "You can't patent a new [breakfast] cereal. All you can hope for is to have the market to yourself for a few months to establish your brand before a competitor knocks off the product."

3. "Multiple discovery" is a well-known phenomenon. Science is littered with people who independently discovered or invented the same thing at about the same time. Any "new" thing is built on a stack of existing innovations and is a response to the state of the market at the time. Consequently, the same ideas tend to simultaneously occur in multiple places and minds. See List of multiple discoveries (https://en.wikipedia.org/wiki/List_of_multiple_discoveries)

At any given time, vendors are chasing each other in multiple areas of functionality, winning at some and losing at others.

For years, I've said this about two leading systems: "You're going to end up with the same website either way. It might work a little differently, but it's basically going to do the same things."

As much as it horrifies the marketing departments of their respective companies, I stand by this statement. I've seen high-end integrations of both systems. One might be better in Scenario A and the other in Scenario B, but they're roughly the same on balance. They compete directly and frequently, winning a roughly even split of opportunities.

You'll see this when you watch a series of vendor demos, especially from documented scenarios. You'll watch different vendors show you basically the same thing, over and over. They all may do it a little differently, but they all provide the same result in the end, and it can be hard to definitively say which one of the options might be "better" than the others.

Earlier, I said that innovations and improvements were incremental and evolutionary, rather than dramatic and revolutionary. The same might be said of failure. Technology projects don't spontaneously collapse over a single feature lapse. Rather, they tend to break down over time due to accumulated baggage. They die by a thousand cuts.

Back to the marriage analogy: while some marriages spontaneously implode due to a singular event, many crumble over time because one partner is tired of dozens of small issues. The fact that the other person didn't make the bed didn't seem like a big deal when they were dating. But a few decades down the road, when the rush of them being a such great dancer has worn off, the other side gets tired of the little annoyances that accumulate.

Additionally, our fed-up partner might one day realize that things *they* do annoy the other person just as much. They begin to understand they should have taken a longer, more critical look at themselves before walking down the aisle.

The same is true of technology. That amazing feature you think will change everything for your business turns out to have a limited impact, and it's overwhelmed by all the small fundamentals that the system is missing. You were so blinded by a flashy feature that you forgot to be sure this was a system you wanted to live with day after day.

Worse, you never stopped to consider how well *you* were ready for it. You didn't consider the technology in the context of your current people and processes and figure out how they needed to change to really make this work.

I have undoubtedly seen projects stall or disappoint because of spectacularly bad technology. Far more often, I've seen projects fail because of more mundane things like small incompatibilities or staffing and process issues, which will have a much larger impact on success or failure.

The vast majority of project failures transcend specific features of the CMS, so thinking that any single CMS will sink or save a project is hopelessly myopic.

Lesson #3

Sometimes you just can't estimate ROI on your project

In a perfect world, every organization would know exactly what it would get in return for every dollar it invested in a project. Appropriately, this called "return on investment," or ROI.

You probably already know that though, because people have been asking you for it. Your organization might want this number before anyone will agree to the financial outlay.

Sometimes, you can estimate this. Sometimes, you can't.

And "Estimate" is the only correct word, because no one can know for sure how your project is going to work out, or what effect it's going to have on the business. It might succeed wildly, or it might fall flat. Anyone who tells you that you're going to get $X back for putting $Y in is making a *lot* of assumptions.

The key is measurement. How do you measure the actual uplift in revenue from a website change?

You need three things:

1. Prior measurement, so you have a baseline

2. An identifiable point of conversion[1]
3. A method of valuing that conversion

Let's consider the easiest possible scenario: e-commerce.

If you're selling something online, you probably have all of these things:

1. You have some prior measurement in your accounting software. You've certainly kept track of every purchase at some level.
2. Your customers perform clear identifiable actions (i.e. they add a product to their cart), culminating when they checkout and actually pay for their purchase.
3. Every checkout is quantitatively measurable. You know how much they spent, what they bought, and how much profit you made.

Under these circumstances, measuring ROI is straightforward. I'm oversimplifying a bit, but you can make a change, then compare the numbers both before and after.

It might be a little more complex if you have multiple changes happening at once, but it's hard to deny that this is the best-case scenario for determining ROI. The conversion is binary – they either checked out or they didn't – and you know exactly how much that checkout is worth.

Outside of e-commerce, it's not that simple.

---

1. In marketing, a "conversion" is technically when someone "converts" into a paying customer. More generally, it means when someone takes an action that provides some value to your organization, such as completing a form, making an appointment, purchasing something, etc. What constitutes a conversion is different for every digital property.

Prior metrics are rare. When organizations want to make a change, they often have to concede that they haven't been keeping track of much. That lack of reporting and control could be one reason why they need a change in the first place.

Even if your organization *does* have metrics, the scale of change you're undergoing might disrupt user patterns to the point where you're not measuring a "change" so much as you're creating something entirely new. You're not moving along a known scale; rather, you're implementing an entirely *new* scale. How do you measure the change in a conversion action that didn't exist before?

Defining what constitutes a conversion can be difficult. If you have a general corporate website meant to promote your company, do you even have a specific point when a visitor takes a proactive action that provides value?

Best case, you might have a "Make an Appointment" or "Contact Me" webform. When someone submits it, that's a conversion. They may not have become a paying customer, but they've converted from anonymous visitor to sales lead. You can use this to measure the effectiveness of changes, and you can structure your website to maximize this number. It gives you an anchor with which to determine if you're making things better or worse.

Assuming you have this, you can certainly measure your conversion rate, but true ROI requires you put some dollar amount on it. This can be difficult.

1. You could track every specific visitor all the way from initial conversion to sale. You would then know when and if they spent any money with you sometime in the future.
2. You could track aggregates and extrapolate an individual value. Let's say for every 100 people who make an appointment, seven of them buy something, and your average sale is $439. Math reveals that 100 people completing that form equals $3,073 in revenue (7 x $439), meaning each completion is "worth" $30.73,

more or less.

It's a far cry from the clarity of e-commerce, but it's something.

However, this requires comprehensive tracking. You need to make sure you can identify customers who came in through a conversion, and track them all the way to a sale, then somehow compile those statistics automatically.

When confronted by this, a customer once told me that this would require them to integrate *six* different systems from *three* different departments, and there was *zero* chance it was going to happen.

Additionally, it gets complicated when you have a multi-channel marketing strategy. When you're actively and passively contacting customers through multiple methods, then who's to say that the website was the specific thing that converted them?

Maybe they saw a display ad while walking through an airport. For all you know, the time they spent filling out the contact form was the very first time they had ever seen the website. And maybe they hated the experience and the website was a drawback that almost *prevented* a conversion, not an advantage that enabled one.

Things get worse when you don't have an identifiable conversion. Maybe you don't have a form that someone can just "fill out" or even if you do, you don't have a dollar amount associated to that conversion. Maybe your website is just filled with content about your organization or cause, and you're just trying to plant some idea in a visitor's head.

Let's say you own a construction company that bids on massive government highway projects. No one goes to a website, puts 100 miles of interstate in their shopping cart, then checks out. No one even fills out a contact form. The process of being selected for these projects happens largely offline, mainly through relationships and formal procurement processes.

At best, your website is background marketing. Someone on the contractor selection team might go there just to make sure your company is legitimate. They'll browse around for a while, review some case studies, and hopefully come away with an impression that your company is one that could get the job done. Months later, while they're staring down bids and being asked to vote, you just hope that something they saw on your website – or perhaps the overall impression they got – surfaces and influences their decision.[2]

In this situation, all you're really "selling" is a memory. There's no online conversion, and the actual conversion takes place sometime in the future, offline. The best you can hope for is that the memory your digital property leaves behind is strong enough to contribute to a future point of value.

And this is the fundamental problem of marketing attribution, verbalized by the now-classic lament: "I know that half of my marketing budget is wasted, I just don't know which half."

Sometimes, figuring out ROI is just an unsolvable problem. You need good metrics, identifiable conversions, and a way to value those conversions in financial terms. It's a minority of projects that have all those pieces fall into place.

And at the risk of complete cynicism, understand that some customers have a website only because it would seem weird *not* to have one. Having a competent website is one price of admission to appear as a legitimate company that your customers want to do business with.

By all means, try to figure out an ROI model and use it to make projections and post-justify an expense, but also be prepared to concede that sometimes, this is not possible. I've seen websites contorted far

2. I concede that you might use this website for other purposes, like recruiting employees. But if someone is staring down a budget request, they usually want to see revenue upside.

beyond their original intention because someone was demanding an ROI number from a scenario that wasn't likely to yield it. The resulting mess might have been measurable, but was it was likely less effective to the ultimate goal.

Sometimes, your gut feeling on ROI is the best you can do, and you shouldn't automatically let any misgivings derail your project.

Lesson #4

Software usually has to fit into larger technology landscapes

Every decision is influenced by restrictions. The art of good decision-making is knowing what the restrictions are and how to identify the optimal solution that can thread in between them.

You don't want your technology selection to fail at the last minute. It's much better to figure out where rigid restrictions are and steer around them than have your process implode after you've fallen in love with a system.

In a perfect world, your CMS floats in a technology vacuum, unaffected by anything else. For example, it might be purchased by Marketing, developed by a third-party integrator, and hosted in the vendor's cloud. Consequently, it will never touch your IT infrastructure nor be touched by your IT staff.

This is actually becoming more common. Marketing departments are making end runs around their own IT by placing entire projects outside the walls of the organization.

And many IT managers welcome this. Lots of them are overwhelmed just dealing with infrastructure and line-of-business technology – if

the network is down and no one can log on at their desktop, it's hard to get worked up over the wrong font size on the website.

These situations are great for vendors because they don't run into any sales objections related to existing technology infrastructure or staffing. In many cases, you're only talking to Marketing, and IT isn't even at the table.

On the other end of scale, there are times when your IT infrastructure and staffing imposes some rigid requirements.

- If you want to develop in-house, do you have developers with the right programming skills?
- If you want to host on-premise, do you support the technology stack software requires?
- If you're externally regulated or audited, does the software support the right protocols?
- If you must connect to some other system, does the software support that integration?

These questions can be unfortunately binary and brutally final – either a system works or it doesn't, and if it doesn't, then there's often no point continuing down that line of selection. I've seen some great software be incompatible in one infuriatingly specific way, which killed a potential deal.

For example, a client had multiple web applications on multiple different programming stacks, and they wanted to manage all the content that was displayed in these applications from one system.

They were evaluating a very capable CMS built on Microsoft's .NET framework which everyone was excited about...until IT noted that the CMS used a "coupled" architecture, which meant that the CMS managed all inbound requests to the web server. The CMS really had to "own" the websites it managed from a processing standpoint.

This meant that the web server the CMS ran on had to be Microsoft Windows-based server. And, by extension, this meant these web applications would all have to be re-written in the .NET Framework in order to co-exist with the CMS. The CMS was going to drop on top of their environment like blanket, covering everything it was expected to manage and forcing it all to run under an entirely new architecture.

You could feel the room instantly deflate. Everyone around the table knew this was never going to happen. Getting a CMS implemented *at all* was going to be a challenge, and they certainly couldn't rewrite all their line-of-business applications to accommodate it. What they needed was a decoupled CMS which could "push" content into those environments in a neutral form.[1]

The deal died at that second. The CMS was fantastic, but it just wasn't going to fit into their technical or organizational infrastructure. The single technical incompatibility was both massive and fatal, leaving the editorial and marketing teams stranded at the altar.

Software can't just be great – it needs to be great in your exact environment, and this is idiosyncratic. You might find yourself trying to thread your way through a minefield of specific and rigid requirements. Human processes, budget and schedule can bend and flex, but technology limits are much more rigid.

You need to know these limitations up-front, even before you write an RFI. This means pointed questions with whomever manages your technology environment:

1. Who will implement the CMS?
2. Where will the website be hosted?

---

1. An architecture known as "decoupled," where the management of content is separated from the delivery. A decoupled CMS generates static artifacts like HTML files or database records and transmits them to another system for delivery.

3. What language, technology stack, or protocol restrictions are inviolable in your organization?
4. What other applications have to work with the CMS?

You might not want to open a can of worms by discussing this, but understand that you're going to be ultimately limited by these restrictions, so you may as well get them out on the table early. These restrictions are the first filter on what goes in the top of the selection funnel.

There's a tendency to try to fly under the radar during your evaluation phase. You don't want to "release the hounds" of IT on your project until you've found something you like. But in some situations, you'll be writing checks that you can't cash, because you're not in control of the environment and systems with which your CMS needs to integrate.

Identify these restrictions early to save yourself disappointment later on. You might argue them in some cases, but if they're inviolable, it's better to know early.

Lesson #5
Internal IT groups can be territorial for a variety of reasons

Ten years ago, vendors and integrators used to sell CMS to IT. Developers and system administrators usually initiated the sale, we sold directly to them, and they often implemented it themselves. They were part of the process.

One of the big shifts of the last decade is that we now sell to Marketing. IT is often just advised that this is happening. Sometimes, they're not even at the table.

Often, they're fine with this, because they're busy just managing line-of-business systems and can't be bothered with the website. Some IT departments consider the corporate website a trivial marketing exercise anyway.

Sometimes, though, IT gets territorial, especially if they're the ones who have been managing and developing the website so far. Occasionally, this manifests as them coming in and trying to hobble the selection process.

This sets up the seemingly perennial conflict of these projects: Marketing versus IT.

This happens for a number of sub-reasons under the general theme of IT protecting their territory.

- Some people think a change is a judgment on what they've done so far. Since Marketing is looking for a new system, they are, in some senses, saying, "What you've done so far is not good enough, and we want to talk to real experts." This can hurt.
- Lots of companies have a system they built internally, and some developers just really like working on their in-house CMS. I totally get this. CMS is fun, and I'd love to work on CMS development all day. When you enjoy what you're doing and feel engaged with it, your emotional investment makes it tough to have someone tell you that you're going to lose it.
- Some people are trying to keep dead bodies buried. There isn't a developer in the world who doesn't have something they built that they *really* don't want to see the light of day. They can just imagine a third-party integrator looking at what they've done, rolling their eyes, and saying, "No wonder they called us in. This is amateur hour."
- Some IT groups are concerned about decreasing relevance. At the extreme, some developers worry about losing their jobs. More likely, an IT administrator is concerned about his group losing organizational influence or budget dollars.
- Some developers just know how hard this stuff is, and they're concerned that Marketing is being unrealistic about what they want and getting their hopes too high. A developer who has been deep in the weeds with their organization's content problems has a lot of institutional knowledge, and they might be watching a selection play out and thinking, "These guys have no idea how complicated this is. None of these vendors is actually going to solve the problems we have."

I have some personal experience with that last one. I once sold a project to a Marketing team, over the objections of IT. We had been as-

sured by Marketing that they knew exactly what they needed, and we had to move quickly or they would lose the budget opportunity.

Sadly, it turned out that Marketing had *no idea* of how the internals of their own website worked. The functionality they worked with from day-to-day was just the tip of the iceberg.

As the project started and we peeled back the layers, it began to dawn on us how wildly under-scoped the project was. My company completed it, but lost a shocking amount of money on the project, while the unheeded warnings of IT echoed through my head every day.

We'll talk a bit later about the novelty of change, and how your current CMS represents everything you don't like, while your new CMS represents a bright new future where everything works out. The same can be true of staffing. Your IT group might represent everything you dislike about your current situation. You might simply be laying all your current problems on their doorstep, sometimes unfairly.

These issues can be tough to work through, because we're not therapists, and some of this stuff can get personal. Feelings get hurt, jobs are on the line, and professional competence gets questioned. More than once, I've had to mediate passionate disagreements between Marketing and IT where the past gets dredged up and blame gets thrown around.

The best advice I can give is to be sensitive to how your search for a new solution might be perceived. Not everyone who disagrees with it is a bad actor, they're often just human.

Lesson #6

The relationship dynamics between the players are different

With three core players, network math tells us we have three unique couples.[1] Assuming a deal is made, there are some dynamics that are helpful to understand. Different players have different motivations and relationships, and the strength and intensity of those relationships will vary over time.

Your project can be roughly divided into time periods.

- **Evaluation:** The period during which you're actively reviewing software and service providers, before making a final selection.
- **Development:** The period during which you're customizing ("integrating") the software to fulfill your specific needs.
- **Post-Launch:** The period after which the bulk of the integration is complete, and the project has publicly launched to its intended audience. The only end to this period is when the website is removed from the Internet.

1. `n*(n-1)/2` in case you were wondering.

Here are how the various relationships exist and evolve over time.

- **Customer-Vendor:** The closeness of this relationship depends on whether or not an integrator is in the middle. The vendor will often be involved during the evaluation period, but the nature of their involvement might change once a purchase decision is made. If the customer is integrating the software themselves, they'll usually stay close with the vendor post-launch, especially their support team during development. If an integrator is working for the customer, the integrator will often sit in the middle of this relationship during development and post-launch. The vendor might have an account manager who occasionally contacts the customer directly, but the integrator usually ends up becoming the vendor's proxy for most issues.
- **Customer-Integrator:** In most cases, this is the relationship that's the longest and most intimate. The integrator has to stick around and make the vendor's software fulfill all the promises. Customers who plan to use an integrator will usually evaluate them alongside the vendor. Clearly, the customer will work heavily with the integrator during development. Post-launch, the integrator will often continue working with the customer for years afterward. The integrator will usually be the customer's first phone call in the event of a problem.
- **Integrator-Vendor:** There's usually always a wider relationship here, beyond any particular customer or project. Integrators will work with the same vendor over and over again, and sometimes *only* work with that vendor. They may have been doing this for years, and their entire company might revolve around this partic-

ular vendor.[2] They often have helpful contacts and backchannels with the vendor.

Some of the above dynamics stem from the fact that a digital property of some kind (website, app, whatever) isn't a thing that gets built and then is never touched again. These projects never completely end – launch day is the *starting* line, not the finish line. A project eventually becomes a product and continues to develop.

So the integrator that builds your website will likely stick around. If they build it and then immediately disappear, that's not normal – usually indicative of a broken relationship at some level.[3]

Two other relationships are notable:

- **Selection Consultant-Vendor** and **Selection Consultant-Integrator:** A selection consultant will have vendors and integrators who they know are competent and who they invite into multiple processes. Very often, a specific project won't be the first time the vendor or integrator has worked with a particular consultant. Selection consultants don't want to be embarrassed by poorly prepared vendors or integrators, so they'll re-use ones they trust to perform well – they'll evaluate the customer's needs, and reach out to the "usual suspects" for that type of project.

---

2. The downside of a relationship this tight is that "when all you have is a hammer, then everything looks like a nail." An integrator might try to bend *every* situation to their chosen vendor, even when it's not the ideal choice. This is also known as "The Law of Instrument."

3. Occasionally, this is the plan. In some cases, a customer will have an integrator do the initial build, then *intend* to take the project in-house. However, this actually happens far less often than planned. If a customer reached out to an integrator in the first place, it's usually because they don't have the in-house skills or capacity. The best of intentions tend to break down in the face of workload, and I've seen "planned handoffs" get repeatedly delayed until it became obvious that the project was never coming in-house.

- **Hosting Provider-Integrator:** Once the site has launched, the hosting provider and integrator will need to work together to keep it running. This relationship can sometimes be contentious when something goes wrong and the two parties start pointing fingers at each other.

That last relationship is not common, just because it's rare to have a completely dedicated hosting provider. Hosting is usually provided by the vendor themselves, or the integrator who does the services work.

This is simply because the hosting environment for a project becomes deeply integrated into the functionality of the project as a whole. The stability of a boat is a combination of (1) the size and configuration of the boat itself, and (2) the state of the water the boat is currently floating in. The same is true with hosting – the stability of the app depends on both the app itself and the environment in which it runs, so it gets hard to separate one from the other.

This chapter simply reinforces the earlier point about relationships: a big part of any project is managing all the different people and organizations who come to the table and understanding how they perceive and interact with each other.

Lesson #7

There can be a blurry line between software and services

When looking for a new CMS to be implemented by an external integrator, you need to decide where to start:

- Do you find the vendor first, then look for an integrator?
- Do you find the integrator, and have them recommend the vendor?

Software then services? Or services then software?[1]

These two options speak to the two sides of your project: (1) the CMS itself, and (2) the work it takes to get it installed, integrated, hosted, and supported over time.

An implementation is like dancing: there are two partners who both have to be skilled individually and together. Your vendor needs to have a competent product, your integrator needs to know how to

1. Annoyingly, I'm not going to answer this question definitively. It depends on the project and its long-term future.

build a website, and *your integrator needs to know that particular vendor's software*. Even skilled partners don't dance well together the first time out.[2]

Understand too that if you start by selecting your integrator, they're going to guide you to a vendor with which they have experience or a relationship. Many times, they're quite open about this – some integrators only work with a single vendor. They promote this and you know this going in (indeed, that may be why you picked them in the first place: "We love Drupal. We just need someone to integrate it.").

Blend Interactive, the services company I co-founded, is a good example. That company became an Episerver partner early in its history, and basically evolved around that software. Blend was Episerver's first North American reselling partner. Blend became very skilled with Episerver, evangelized it to customers, other partners, and to the technical community in general. While Blend worked with a few other systems, they never hid their bias toward Episerver.

As is common in the services business, Episerver became our "leading system," meaning it was the system we tended to default to. When we evaluated a new project, the first lens we looked through was "will this work with Episerver?" If we found factors that precluded Episerver – budget, technical stack, etc. – only then did we turn to something else.

This also speaks to another common pattern: an integrator often has two leading systems, based on budget capacity. They'll have the commercial system they prefer, but will also have an open-source or lower-cost system on standby for situations where the customer doesn't have the budget for the alternative.[3]

---

2. I might waffle a bit here, because some platforms are simple enough that a smart integrator can figure it out as they go along. But that's not common, and you should never count on it.

You might think this is limited or biased – and it is – but it's also natural. This is just how the human brain works. When you evaluate any situation, you're comparing it to past situations and looking for clues or markers that make that situation similar or different. You adjust your actions based on this.[4]

Take the most neutral, unbiased consultant in the world, show them your project, and I promise they'll mentally categorize it against a known set of past projects. They might not tell you this, or even be aware they did it, but I guarantee it happened.

Sometimes, an integrator will offer to set aside their bias and help you select a CMS from a wide-open field of options...but not really. I maintain that it's nigh impossible to ignore past experience enough to be truly neutral. Beyond financial incentives and relationships, an integrator usually works with software because they like it and believe in it, so they're going to tend toward what they know. And even if they ignore software with which they have direct experience, they will likely tend toward software that exhibits similar architecture and patterns.

This isn't always a bad thing – no one can be an expert in everything, and an integrator usually has a genuine passion that comes from a history of successful work with a particular vendor. If you're asking for someone's opinion, then you have to accept that they can't forget work they've done in the past.

---

3. Although, I do know of companies that intentionally refuse to offer a lower-cost option, and use the cost of their preferred vendor as a qualifying device: "If they can't afford [insert vendor here], then they can't afford us."

4. Gary A. Klein is a cognitive psychologist who has studied this phenomenon extensively. His book *Sources of Power: How People Make Decisions* is one of the definitive works in this space. His model is entitled "Recognition-Primed Decisions," which speaks to the theory that we base decisions on how we recognize them in relation to past experience.

Just know that both sides of the vendor-integrator pairing matter, as does the intersection between them. A lot of great software has been destroyed by a poor implementation. Conversely, sometimes mediocre software can be saved by a veteran integrator who knows where all the bodies are buried and has experience in working around limitations.[5]

Is there a "right" answer to which side you should select first: integrator or vendor? Not necessarily. I've seen organizations have successful projects using both methods. However, many organizations already have an established tendency on one side – they like a specific CMS, or they have a relationship with a specific integrator. The other side tends to flow naturally from that.

It's usually not possible to completely isolate one side from the other. Every CMS is going to influence the selection of integrator, and vice-versa. There's no perfect firewall between the two sides.

The main point here is not to get "software myopia." No matter how great the vendor seems, someone has to work with their system, and that's the real trick. Giving a Ferrari to a teenage driver is a good way to end up with a car wreck.

---

5. My company once made a good living working with a terrible CMS just because we knew how to make it...less bad. We hated the CMS, but we got called in over and over on what we called "salvage operations": one of this vendor's customers was upset about their resulting implementation, and we came in to manipulate the system in unique ways to make it better. We didn't love the work, but it sure paid well.

Lesson #8

Open-source software often has no representation

As noted earlier, you can go find a software vendor who will bring in an integrator, or you can find an integrator who will bring in a vendor. Unfortunately, under that dichotomy, open-source becomes an orphan case, especially when you want to integrate it yourself. Who represents it?

For any software, the only people who are going to advocate for it, demonstrate it, and try to sell you on it are organizations that stand to make money from it. Either there's a vendor who sells licenses, or an integrator who sells services.

No one will formally evangelize an open-source CMS for no possible return, no matter how much they like it or believe in it. To find someone willing to sell you on an open-source platform, you need to find:

- An integrator who specializes in that CMS and who can generate professional services revenue by integrating it for you
- A hosting vendor who specializes in that CMS and can charge you hosting fees

- An organization that provides some value-added product on top of that CMS (though, in that case, what they're selling is their product, not the CMS itself)

Consider a situation where none of those are true. Let's say you've heard about Drupal, you think you have the talent to integrate it in-house, and you want to host in your own data center or your own cloud infrastructure.

If you still need to sell this to your organization, who advocates for it? Who answers questions about it? Who demos it?

Well...*you* do. You might theoretically pay an integrator or expert specifically to give you a demo with the understanding that this is all they're going to do, but I've never heard of this actually happening. You *might* make this happen with the promise of a follow-on consulting engagement as you get up to speed, but then we're back to the first option above, and you've found someone who stands to make money from your selection.

What can be problematic is when your organization requires technical answers about the system for regulatory or compliance reasons. Selling open-source software to larger enterprises or government organizations can be difficult because the organization's purchasing process may be designed around the expectation of a selling entity. In many circles, open-source software still has a perception problem.

For example, organizations often have long lists of questions that need technical answers. Some examples:

1. How does your software avoid SQL injection attacks?
2. What is your process for background checks on your development team?
3. How many locked doors are between the outside and your development servers?[1]

These questions aren't totally without merit, but they pre-suppose a traditional vendor, or some other organization or service provider. How can you get answers for questions like these for an open-source product? Some of the questions would require you to find and interview a single developer in the software's ecosystem. Other questions simply don't apply.

In the mind of the organization asking these questions, there simply *has* to be some governing authority standing behind every software platform. Their entire procurement process might be based on this premise, and they would likely penalize software not exhibiting the structure they expect.

Just remember that if you have no party who can generate revenue by you deciding to use a particular CMS, then no one will put effort into selling it. With commercial CMS, you have a vendor who's the obvious beneficiary. With open-source that someone integrates for you, you have an integrator who is going to sell services.

With open-source software that you integrate and host yourself, there's no advocate, and that can be a lonely place to be. In that situation, all you have is the ecosystem of the CMS – the online community, documentation, and existing evangelism about it. You can still do an evaluation, but you'll need to have a technical resource in-house to act as the system's "vendor proxy."

---

1. I'm not making this one up. I actually saw this in an RFP for a government agency once.

Lesson #9

There's sometimes tension between the vendor and their partner integrator

At the largest scope, a vendor-integrator relationship is usually always benevolent. If it wasn't, it wouldn't exist – both parties would find someone else to work with. However, in specific instances, the relationship can get trickier.

If you're talking with both a vendor and an integrator, you've put them in the position of selling into the same opportunity. They have to work together, as they'll succeed or fail as a pair. Sometimes, they don't totally agree on the specific path to that goal.

A sacrosanct rule to acknowledge is this: Vendors always want to sell as much software as possible.

This is how sales works, universally. Selling $1 worth of product is great, but selling $5 is always better.

Every vendor has a variety of products. They have a core CMS, certainly, but they have all sorts of other add-ons ("modules," "extensions," whatever), services, and licensing options. No vendor has a

single flat fee. Every quoted price is a sliding scale of size based on what particular combination and number of products are in it.

Every vendor wants to load up every deal with as many products as they can jam into it to pump up the total amount of the deal. This is how vendors stay in business. Their sales staff is heavily incentivized to sell everything they possibly can, regardless of actual need.

This might sound harsh, but this is true of every industry. Have you ever been sold the "weather-proofing package" when you bought a new car? Or the extended warranty when you bought electronics? Or even got a better deal if you "super size" your food order? Selling software is no different than selling anything else, and it's your job to police your own needs against what you're being offered.

Still, for an integrator, the vendor's natural tendency to increase the size of a sale is sometimes problematic. The integrator is usually fine with the vendor's core product, since it's the basis for the entire relationship. However, sometimes integrators can take issue with additional products. These are "add-ons" or services that vendors want to sell in addition to the core CMS.

Issues vary:

- The integrator might not think that the customer needs the add-on
- The integrator might not have much experience with the add-on
- The integrator might not think the add-on is very good
- The integrator might think the add-on is overpriced, and they think they can do the same thing for less money through alternative means

Occasionally, an integrator has to shield the customer from a hailstorm of options and add-ons. In years past, I've been involved in some tense conversations about how a vendor was pricing their package, and whether or not that package was benefiting the customer.

Why does the integrator even care? If a vendor wants to sell more and more product, how does this negatively affect the integrator?

The noble answer is that a good integrator advocates for their customer and defends them against unnecessary expense. This is certainly true in many cases, but there are a couple more cynical, practical reasons.

- **The vendor and the integrator are competing for the same dollars.** Every customer has a budget limit, and a dollar that a customer spends on software or hosting is a dollar they can't spend on services and vice-versa. Integrators sometimes resent vendors that suck up an inordinate share of the budget with products that just aren't necessary, and vendors get annoyed with integrators that over-price their services leaving no budget for product (or price themselves out of an opportunity entirely).
- **The integrator and the vendor rely on the other's competence.** It's easy for a vendor to make big promises about a subpar product, then walk away and let the integrator try to figure out how to deliver. In some senses, a vendor is gleefully writing checks that the integrator has to cash, and when a vendor oversells a bad product, the integrator is the one who looks bad when it doesn't work out. Alternately, a vendor might get stuck in a deal with a substandard integrator who doesn't have the skill necessary to make their product look good during the sales process, or that botches the implementation, leaving an unhappy customer on the vendor's doorstep.

The dynamics of this relationship depend heavily on where a deal originated. The customer usually controls where a deal is born. Sometimes customers pick a vendor who then brings in an integrator, and sometimes it's the other way around. That origination of the relationship will influence who controls or "steers" the deal.

If the customer contacts the vendor first, and the vendor picks a particular integrator to assist on the deal, then there's an implied oblig-

ation there. The integrator knows it's the vendor's deal, and they're usually going to go along with whatever the vendor wants to sell.

As someone eloquently put it to me once, "You dance with the one who brung you."

If the roles are reversed – the deal started with the integrator, and they brought the vendor in – then the integrator might be more vocal about what products the vendor sells. Each side will act to protect a deal they consider "theirs."

And sometimes this is in dispute. In some situations, both sides think the deal is theirs to steer.

For example, when selling services with Blend Interactive, I had been contacted by a selection consultant and had recommended a vendor's product. The customer then reached out to both of us – vendor and integrator – simultaneously.

We came to terms with the customer on a purchase of the the vendor's core product.

The vendor then quickly proposed an additional service at $3,000/month because, "At that price, it's usually a slam dunk." This was a service that the customer didn't need at the moment, and wouldn't have time to learn amidst all the other changes that were about to happen.

"Slam dunk" or not, over a five-year horizon, this was $180,000 they would be paying the vendor, *that they would not be paying us*. I declined to offer this option, to the vendor's annoyance.

This led to a follow-on conversation about which party the deal actually belonged to. Since the customer reached out to both parties together, the vendor felt like this was a "marketing win" for them, while we felt like the vendor was only in the deal because we had recommended them – to us, this was a "partner-led opportunity." We had to argue the case that this was *our* deal to steer.

This situation resolved itself amicably, and the customer was never even aware of the dispute, but just know that these things come up.

Vendor and integrator want to present a united front, so they're not going to air each other's dirty laundry in front of the customer, but behind the scenes, disputes happen.

In a close, long-term vendor-integrator relationship, they become a lot like siblings – they love each other and want good things for each other overall, but they still squabble occasionally.

Lesson #10

The most thorough selection processes are a funnel of deepening analysis

A selection process is a definable project which should have a methodical process behind it. It's neither magical nor accidental. It should be managed with all the milestones and reporting that any project would have.

In any selection process, you need to decide how deep you're going to go. There's a balance between doing adequate due diligence and suffering from "paralysis by analysis." In many situations, customers have just asked someone they trust to recommend a system, followed that advice, and been very happy.

In other situations, the process goes much deeper. The more thorough processes work down through a "funnel" of analysis. You start with the widest possible view, then begin to eliminate vendors in response to their responses and your evaluation, narrowing your view, until you arrive at a single point of decision.

A typical funnel will look something like this:

- **Request for Information (RFI)** sent to a "long-list" of 7-8 vendors. This is set of questions designed to eliminate vendors on the largest of criteria. Some systems will be priced out of your range, some will run on the wrong architecture, some won't offer cloud, etc. You have a list of deal breakers, and this will filter them out. An RFI asks vendors about their platform – you're not necessarily revealing a lot about your project at this point.
- **Request for Proposal (RFP)** sent to a "short-list" of 3-4 vendors. This is where you explain your project, exactly what you're looking for, the goals you need to achieve, and provide a list of demo scenarios for the vendor to show you. This is also where you ask for more detailed, binding pricing. This is customarily followed by a demo session from the responding vendors.
- **Proof of Concept (POC)** for one vendor. This is where you have a final selection, and you want to make sure the system is as good in practice as it seems. You'll ask the vendor to come in and actually implement something small for you, or train your team and work alongside them while they implement.[1]

At each stage, you'll learn more and more, and you'll eliminate poor-fitting systems. Eventually, your POC should vet the final candidate. If that vendor fumbles the POC, then you go back to your RFP round, pick second place, and move them into a POC. If this happens, consider yourself lucky for finding out before you consummate the purchase.

Some consultants prefer more than one vendor in a competitive POC, while some consider the POC a final vetting for a single vendor. When consulting on a selection, I generally cut from three RFP vendors to a single vendor then verify that vendor with a POC, rather

1. Be prepared to pay for a POC, though perhaps not at full rate. Sometimes the POC is done at no cost, but with a "kill fee" or "consolation fee" if the vendor or integrator does not win the project.

than cutting a single vendor, then conducting a competitive, two-vendor POC.

One trick is figuring out what goes into the top of the funnel – the long list, to which you send the RFI.

There's no set answer here. Some ideas:

- Ask peers in your industry
- See who is sponsoring conferences in your industry
- Ask analysts in the industry
- Review analyst reports
- Hire a consultant who specializes in the field

That last option is ideal, if you have the budget for it and you can find someone to work on a limited engagement. A few emails to exchange documents and a single phone call might be enough to them to recommend a starting list for you.

Key to this process is engaging with as few non-viable vendors as possible. At the highest possible level, you can eliminate vendors on a few rigid criteria:

1. **Technology stack.** If you're a .NET shop and their system is Java, then it's not going to work. (Assuming, of course, you're restricted by tech stack.)
2. **Available budget.** Do you have $100/month, or $10,000/month? A lower budget will eliminate a lot of options.
3. **Market space.** If you want a general, marketing-capable CMS, then it's wide-open. But more specialized use cases – you need a learning management system for a university, for example – will narrow the pool sharply.

The goal is two-fold –

First, you don't want to waste vendors' time. They have to put forth considerable effort to respond to your requests. Don't create needless work for them.

Second, *don't create needless work for yourself*. Working through a formal RFP process is complicated. It requires extended analysis and divergent thinking along multiple axes of functionality and process. To do it well can be exhausting. You can only analyze so much, and every system you add to your search is one more that you have to work through.

In addition to the increased workload, every new system runs the risk of sending you off on a unnecessary tangent. A new vendor might storm in, guns blazing, promoting a massive paradigm shift and upending everything. More often than not, this just confuses everyone.

Be careful with your shortlist. I defy anyone to give fair, deep-dive analysis to more than three vendors.

The larger point is that you shouldn't get overwhelmed by attempting to do a "Big Bang" selection. Additionally, you shouldn't fixate on a single product to the exclusion of everything else.

You should start with a wide funnel, then narrow it down over time by consciously and deliberately eliminating vendors. Considering a wider array of products will allow you to learn what's in the market, and the elimination of products will force you to acknowledge why you didn't think they would work, which is critical to understanding your project.

Your final selection shouldn't be climactic or the result of a wild debate. If a disagreement is still raging late into the process, then it might be worth backing up and asking if members of the team are in complete agreement on the goals of the project, or if there is a foundational misalignment that would cause the group to be so far apart.

At the end of the process, you should have gone through a methodical level of analysis, narrowing over time, which has given you a measure of peace and consensus about your decision.

Lesson #11

A vendor's ecosystem should be evaluated as a core feature

Eleven years ago, when Episerver had first entered the North American market and Blend Interactive was one of the few integrators working with it, a certain consultant I knew would always respond to my evangelism the same way: "Let me know when they have more than a half-dozen partners."

This comment would annoy me, because in my head, *we* worked with the software, and that was enough.

But I've since come to understand that this wasn't enough. Software doesn't exist in a vacuum, and what happens outside of the vendor is just as important as the system and vendor themselves. Software needs to be supported by a network of resources, known as an "ecosystem."

The ecosystem of software platform consists of:

- An active community
- Accurate, comprehensive documentation
- Qualified integration partners

We'll unpack these items individually.

Community can be both formal and informal. The vendor probably provides a set of discussion forums, and good vendors encourage or even require their professional services team to spend time there answering questions. Many vendors reward members with some type of "MVP" status for helping other members of the community.

Informal communities matter too. A lot of problems can be resolved via social channels like Twitter or forum sites like Stack Overflow, provided there's critical mass around the product. It can be instructive to search Stack Overflow for questions people are asking about the CMS you're evaluating.[1] Evaluate both the volume of questions and their content. what problems are people having? How many people are willing to answer those questions?

Back in my corporate days, my company purchased a very expensive enterprise content management system. I spent months trying to figure it out, and all that time, I felt like I was working alone. In the back of my mind, I was sure that there was a group of people *somewhere* who were suffering through the same growing pains I was, but I just hadn't been able to connect. Eventually, almost accidentally, I found a Yahoo! Group for this software, and there it was. I had finally stumbled upon another group of customers of this particular vendor, struggling with the same problems, and willing to talk about it. It was revelatory, and it changed how the software was perceived and implemented at my organization.

Documentation ranges from auto-generated API docs to blog posts to sample code to accurate feature lists. Lack of documentation is probably Complaint #1 among developers trying to work with a vendor's system.

I remember months of working with a particular vendor's software without enough documentation. My last resort was simply to decom-

1. Bonus points if the CMS is well-known enough to have earned its own tag.

pile their code (undoubtedly in gross violation of their license agreement), and wade through it line-by-line to figure out what was going on.

For many developers, sample code is the pinnacle of documentation. It tends to get vilified for encouraging blind copying and pasting, but reverse engineering is a common way to figure a system out – give a developer code that works, and they'll examine it, take it apart, and modify it to do what they want. More software has been learned in this fashion than probably any other.

Many vendors offer "reference implementations" of their software. For CMS, this would be a sample website the vendor uses for demos, and offers as a "sandbox" or trial for customers who want hands-on experience with the system. The hope is that this implementation has enough breadth of functionality and represents good coding and usage practices so that it can be examined as reference.

Often, customers will start new projects from the reference implementation, modifying it for their particular needs. The advisability of this varies greatly, depending on the quality of the reference site and the applicability of it to the customer's requirements. Some vendors are horrified that customers do this, while others have specifically designed their reference implementations for this purpose.

Look through this documentation and reference code before you buy. Ask your developers to look through it. Make sure it's public.

I've seen vendors who lock away their documentation behind a login, which doesn't make sense. If you can get a free account – by joining their developer community, for example – then that's fine, but in some cases, vendors will only allow access to paying customers, as if their documentation was some trade secret.

Look for third-party documentation. Are people writing about the platform? Are there blog posts about how to implement or work with features? Go search Amazon. Has anyone written a book about it? Visit independent training sites like PluralSight or O'Reilly. Does anyone maintain video tutorials on YouTube?

The level of documentation and community activity is a reasonable proxy for how well that CMS has been received by the market and the level of adoption it's received.

Finally, since that conversation with my analyst friend about Episerver 11 years ago, I've come to understand how much partner networks matter, the hard way. The partner network of a CMS vendor is your safety net. It's the thing you can fall back on if the vendor or integrator lets you down. It's a soft place to land.

Partner networks vary greatly in size and intensity. To be a "partner" of a vendor means an integrator gets some special perks – some extra licenses, recognition, more communication, and hopefully sales leads. In exchange, the vendor hopes that the integrator will evangelize their software and sell it for projects.

Sometimes an integrator only works with a single system and both sides benefit from a close affiliation. However, it's not uncommon for the relationship to be asymmetrical, where an integrator is just looking for targets of opportunity. They'll promote a vendor's software if it works for them, otherwise they'll gladly use something else. They're "badge collectors" who will claim expertise in any system if it will help them obtain services work.

This means that the generic descriptor of "partner" is not absolute. Not all partners are equal, and you can't trust this means an integrator actually has practical skill in a system.

A few years back, I watched a large customer purchase a CMS from a small, little-known vendor. Unfortunately, the vendor was small enough that this customer quickly overwhelmed the vendor's professional services team.

The customer asked me to help them find a third-party integrator for help. The vendor's "Partners" page listed a dozen companies, but every company I contacted had only done one implementation, and had only become a partner to get a development license (a common perk of partnership). Some of them didn't even know their logo was on the vendor's website.

We came to understand that there really was no partner network. And the vendor's professional services staff was less than a dozen. *There were less than 12 people in the world available as competent engineers of this system*. And they were busy, all the time.

That's not a good thing to find out when you're six figures deep in licensing fees.

Honestly, I don't enjoy emphasizing ecosystems, because the lack of one is the highest bar to entry for a new vendor to break into the mainstream. Newer vendors have to contend with building their system, supporting their customers, and, at the same time, convincing new people to use it to the extent that an ecosystem develops.

It's a classic chicken and egg situation – few people are using a new system, so there's little in way of ecosystem; and there's little ecosystem because few people are using it. In the best-case scenario, it becomes a positive reinforcement loop. More users mean more ecosystem, and more ecosystem means more users. But the opposite is sadly more likely to be true. Smaller vendors can wallow in a tiny market share and never break out because they can't develop their ecosystem.

If you consider the major CMS vendors compared to all the others, the difference that stands out are the size of intensity of their partner networks. Look outside those systems, and you'll still find some great software, but drastically fewer partners. Some very solid CMS platforms might only have one or two focused partners in North America.

I genuinely feel badly for smaller vendors that struggle with their ecosystem. The lack of one is a huge impediment, and they're not easy to develop.

The fact remains that the lack of a viable ecosystem can put customers in a very bad position. You need to ask and verify questions about your vendor's ecosystem, and vendors need to start cultivating their ecosystem and partner network like it was a core feature of the product.

Lesson #12

A Request for Proposal can sometimes be abusive and this doesn't help anyone

If there's one thing that infuriates me, it's an abusive RFP. You see these every once in a while – an RFP written with no regard to the people who have to respond to it, and sometimes even openly contemptuous of them.

Often, this is reflected in ridiculous timeframes. While responsiveness is important, and vendors certainly want to make a sale, they have other projects they're working on. They need time to respond to your request.[1]

Other times, we see lists of hundreds of questions, many of which are obviously never going to be reviewed and were just thrown in because someone had the power to ask them. A lot of this is systemic to the organization,[2] but sometimes you can imagine a bad actor at the

1. For a large, comprehensive RFP with scenarios and demo requirements, at least four weeks is a fair amount of time to turn something like this around. If you want it in a week, the quality will be poor, if it's returned at all.

customer organization demanding responses to pointless questions as a way to prove their own value to the organization or derail a project they don't want to see happen.

And sometimes, it's not timeframes or bloated size as much as it's...tone. I've seen RFPs full of specific admonitions and punitive requirements that sound like they were written by someone with a mile-high chip on their shoulder who's just looking to slap down a respondent to make themselves feel better.

I'll speculate on a few reasons why this happens:

- Sometimes people are just unpleasant. They're on a power trip and enjoy being in control. They view themselves as the puppeteer who holds all the strings.
- Sometimes respondents get dragged into turf wars. Certain factions inside a customer organization might not want a project to happen, and these groups have influence over the RFP, so they intentionally make it unpleasant.
- Sometimes there's a vendor or integrator that the customer wants to select (they have someone "in their back pocket"), but policy dictates they have to issue a public RFP. They write the RFP in such a way to dissuade anyone other than their preferred vendor from responding.
- Sometimes people are terrified that they're going to get taken advantage of. They're afraid of the respondents – sometimes rationally – and this manifests itself as a punitive process. They're going to show you that they're in charge, because they're secretly worried they have no idea what they're doing.

---

2. Government agencies are famous for this. I once reviewed an RFP for a government agency that ran to 50 pages, *49 of which were instructions on how to respond to the RFP*. The description of services the agency wanted a vendor to perform consisted of one single page. Every vendor and integrator can tell you similar stories.

These situations are difficult for the respondent. Yes, we're in sales, but we're also human beings. We have feelings, we take pride in our work, and we have lives outside the office. We don't like working on things that sap our morale.

Your ongoing relationship with your vendor and integrator is different than when you buy consumer products. You have to live with both them. You are entering into a critical relationship with both of these organizations. The quality of that relationship matters. It can be more or less productive, based on intangible factors like how it was conceived.

This is especially true with the integrator. We'll have an entire chapter on this later, but know that you are, to some extent, selling your project to the integrator.

Good integrators are in demand and often have little unused capacity available to sell. They have to pick their projects carefully.

The opposite is also true. Bad integrators aren't in demand. They need any project they can get their hands on, and they're willing to be abused in exchange for revenue.

If you find that you can completely manhandle an integrator during the proposal stage – you get them to agree with anything, and cut an insanely good deal for yourself to their detriment – perhaps ask why this integrator is so willing to tolerate your abuse? The answer should scare you.

Vendors and integrators are not the enemy. Occasionally, you'll work with one and realize they're not being totally honest or reliable with you, *but don't simply assume this*. If you set up an antagonistic relationship from the outset, the results are not likely to be what you want.

I've simply discarded several RFPs because I took one read and realized I'd never in a million years want to work for the people who wrote it. Call it wishful thinking, but my gut tells me that most of those projects didn't end well.

Lesson #13

Know your budget target in advance and be prepared to share it

At some point, you're going to have to come to grips with your own budget. Do you know what it is? Do you have any idea of how much you have to spend?

If you're answer is no, then you need to figure this out before you embark on your process. You shouldn't go into a selection process completely blind. And you shouldn't use it as a fishing expedition for pricing.

Whenever I was consulting on a selection, I would push the client for their expectations on budget. If they claimed no idea how much they could spend, I would make sure they had as much as an enterprise CMS vendor would require at the mid-range, and as much as an integrator would require to implement it.

If they protested at this number (which would be well into the six figures), that was still progress. Then I could have a rational discussion with them about how much budget they *did* have and tailor the discussion accordingly.

When I set the expectation for an enterprise CMS and implementation, I was specifically coming in at the high end of all solutions. But

neither CMS software nor implementations are set in stone. They can bend and flex along multiple axes.

Consider this adage:

> You can have it fast, cheap, or good. Pick two.

I've always thought this should read "eliminate one," rather than "pick two," because it accurately represents that projects have inflection points. Most all projects have a soft spot on which they can flex.

- **Schedule:** How quickly you go from idea to launch ("fast")
- **Budget:** How much the entire solution costs ("cheap")
- **Features:** What's included in the project ("good")

Here's how your project might flex:

- The CMO has freed up what's left of the marketing budget, but it has to be spent by the end of the year. You have a rigid **budget** and **schedule**, so you're going to flex on **features** by figuring out what you can pay for and get done in time.
- The organization has given you a set budget for each year, and laid out the grand vision for their digital strategy. They want to know how long it's going to take. You have a rigid **budget** and **features**, which means you will need to flex by developing a phased **schedule** which launches features in multiple cycles.
- You are required to have a specific project completed by June 30 to pass an audit. You have rigid **features** and **schedule**, so you're going to have to flex on **budget** and be prepared to pay more in contractors to get it done[1]

---

1. For the record, throwing money at a project to get it done faster is actually a terrible way to make a deadline and it rarely works. The legendary book *The Mythical Man Month* explained this phenomenon in detail. My business part-

When you consider your project, know that it's not monolithic. There are undoubtedly some things that need to go in by launch, but there's likely many other features that can be phased in over time.

So when it comes time to discuss your integration budget, what do you reveal? RFPs tend to concentrate on features and schedule, then have an implied flex on budget. It's common to get an RFP that boils down to, "This is what we want, and this is when we need it by, so how much will it cost?"

Budget is somehow always expected to be the missing variable that vendors need to fill in.

But it's not really a missing variable at all, is it? Because *you* likely know how much budget you have. You just want a vendor to fill it in and see if it matches the number you have in your head.

Hopefully you're not just looking for the lowest number, because there are *so* many other factors that go into picking an integrator. Assuming you're not, then what you're looking for is the best integrator inside the budget range you have.

Consider: Would you select an integrator you don't love in order to save $10,000? Probably not. For projects like this, customers don't tend to chase deals.

When we combine this with the fact that projects can flex on multiple points, it usually makes sense to just state your budget in advance. Right in your RFP, say how much you'd like to spend on the project. "This is what we have, tell us how you might deliver what we want inside that budget."

For some of you, the hair on the back of your neck just stood up. There are organizations that refuse, under any circumstances, to state how much budget they have because they think they might have *more* budget than the project needs, and if they reveal this number,

---

ner put it more bluntly: "Nine women can't make a baby in one month."

some integrator is going to soak them for all of it when they didn't need to. Customers are afraid they'll overpay.

But after 25 years in this industry, I find it hard to believe this is the case. I can count on one hand the number of times someone came to Blend Interactive with more money than they needed. Best case, they're somewhere in the right ballpark. Often, they're under the budget level they need to get to for the services and software Blend worked with.

In these cases, we'd simply let them know the number we thought they needed to be at and why, then work with them to figure out if it was possible to get there. If we knew the budget range, we could consult with the prospect to figure out how to make it work.

For every project, there are options:

- Some of your functionality might not be worth its budget. If you're swimming in money, fine, but you might have a feature that's expensive and likely won't provide a return. We might cut some of those.
- You can phase your implementation. There are probably some advanced features that you aren't going to use right away. Let's push those off to your next budget cycle and implement them when everyone is comfortable with the new system and ready to expand their skills.
- We might simply go with cheaper technology. Every integrator has their preferred system, but they can often "step down" if the budget calls for it.

Now, that last point may have you pointing your finger and saying, "That's why I wasn't going to reveal my budget! Because if cheaper technology will do the job, then I want them to use it!"

The problem here is that you believe that integrators will price as low as they can, without a budget target. You think that every integrator

will sweat, trim, and cut their number before they put it in a proposal, in the belief that they need to be the lowest number you consider.

But that's just not the case. Most integrators aren't going to compete on price, because these aren't "projects" as much as longer-term relationships. Most integrators know that they'll be working with a customer for years, so going in at a rock-bottom price is not helpful. The customer will then expect to keep paying those low prices forever, or the integrator will raise the price to their normal range and their now-angry customer will go find someone else.

Put another way: If you're hiding your budget in the hopes that an integrator drops their price, you're not likely to reach the goal you really want in the long term.

Alternately, you might think that if you reveal your price, lots of integrators will choose not to respond to your RFP at all. This is undoubtedly true, but *that's the point*. If your budget is not at a level that an integrator is willing to work at, then why go down that road at all? It's not helpful to anyone to start something you can't finish. Your budget is what it is, so have that discussion early, openly, and honestly.

Not stating your budget forces the respondent to make assumptions, and sometimes they're not accurate, which doesn't work to anyone's advantage. I once passed on an RFP because there was no budget stated and they refused to provide any insight to their range. The features they wanted, combined with the type of organization, just didn't make me feel like there was going to be enough budget there. Responding to an RFP takes considerable effort, and I couldn't justify it based on the information and expectations I had.

In response to my declination email, I got an annoyed response that *then* stated their budget, which turned out to be quite reasonable. Their email had the tone of, "Ha! Aren't you sad you didn't bid now?"

Well, sure, but I was sadder for *them*. My company would have done a nice job with their project. But because they refused to state their

budget, my only rational choice was to pass on it. As such, they lost the opportunity for a well-prepared, informative response from an experienced integrator. They were the only real loser in this situation.[2]

Of all the advice in this book, this might be the most controversial. There are many who maintain that your budget should remain a well-kept secret. In my experience, I've found that discussions centered around a known budget are more productive, more honest, and lay better groundwork overall for a relationship over the long-term.

2. This is an example of where a selection consultant is helpful. If someone hired a consultant who was familiar with Blend Interactive, then we knew the customer had been vetted for budget. If the budget wasn't stated, we still knew we could trust that they were in the right ballpark.

Lesson #14

If you don't know how to write an RFP, get help

There's a tendency in an RFP for customers to try and sound very smart. No customer wants to seem naïve, so they posture in an RFP to try to get the vendors to believe they're experienced and cannot be taken advantage of.

This rarely works. I promise you that every vendor and every integrator has seen more RFPs than any customer – they do this for a living, remember. Larger organizations will have a person that literally reads and evaluates RFPs all day long. And when you see enough of them, it becomes obvious when someone has no idea what they're talking about.

Worse, a lot of posturing manifests as rigid demands on the response format and content. When you do this, you're going to force an artificial response out of someone, which might not be the best plan for what you want to accomplish. Plan Y might be the right answer, but you're never going to see that because *you specifically asked for Plan X* and refused to consider anything else.

For example, I saw an RFP once that listed this among a long, official-looking list of "Requirements."

> Design update plan with template, etc.

... what even is this? A plan for...updates to the design? And what does a template have to do with it – a template for the *design*, or a template for the *plan*? The qualifier of "etc." is inherently imprecise. And then this was under the heading of "Training," which made even less sense.

I don't know everything, and I'm willing to admit it. So I called the sender and asked what this was. Honestly, I thought I'd learn something new.

But when pressed for an explanation, the woman who wrote the RFP didn't even know what it meant. She seemed surprised I was asking, not because the definition was obvious, but because she expected me to posture as much as she had. She clearly didn't want to admit that she didn't know what this thing was, and she absolutely never expected me to admit that I didn't know either.

After dancing around the subject for a few minutes, I said, "I appreciate the list of requirements, but can we just propose the project as we would build it in our process?"

Long pause, then "Sure."

The jig was up. Her relief was palpable.

What's terrifying is that a lot of respondents would have simply offered up...something. Rather than admit ignorance, they would have just fabricated something on the spot to sound as if they could meet a requirement which was copied-and-pasted without any thought or consideration.

When you posture in an RFP, you run the risk of hamstringing the respondent. They don't want to insult you, and they don't want you to think *they're* dumb, so they'll launch into the Classic Escalating Arms Race of BS, in which both sides try to bluff the other one into thinking they know exactly what's going on. It can be pathetically hysterical.

More cynically, if you posture in an RFP and get caught, you've damaged your position. The respondent now knows more about your mindset and your level of knowledge. Bad actors will use this to their advantage.

Sometimes, I would see an RFP that was so inept it made me genuinely worry for the organization who issued it. It was painfully clear that whomever wrote the document had little to no understanding of the market or the software acquisition process. They were asking for too much, providing too little information, or both.

I've seen customers attempt to procure half a million dollars in software and services from a single page of bullet points. I would shudder every once in a while when I imagined how easily some customers were going to be manipulated by respondents. They were never going to get what they were looking for.

In these situations, we would always ask for a phone call to get more information. If we were refused this call because of their process, we would pass on the opportunity. And even if we did bid it, we would usually always just bid a discovery engagement – essentially, we bid a process to help them define all the information that *should* have been in the RFP to start with. In effect, they would end up paying us to help them re-write their RFP.

In one such case, a customer's RFP had a classic list of bullet points which barely scratched the surface of what a vendor or integrator would need to provide a binding quote. We requested a discovery meeting, but were denied in the interest of "fairness." We told the customer that we simply couldn't respond on the information we had.

Six weeks later, we were re-contacted by the customer. They indicated, sheepishly, that the responses "weren't what we expected," and asked if we would re-consider. We got our discovery meeting after all, and eventually won the work after we persuaded the customer to drastically limit the scope of what they were requesting.

Remember, you probably asked for a proposal from someone because they're an expert in a thing that you are not. That being the

case, *let them be experts*. At the risk of sounding harsh, the average integrator knows how to build websites better than you do. That's why you're asking them for a proposal, remember?

If you tightly script how you want a response, then you're going to get just what you asked for, which is something you already know. If you give the respondent some leeway in how to respond, they'll give you some insight into their process, methods, and toolset, which can be enormously valuable.

Feel free to explain what information you need to see from the respondent. But don't be overly prescriptive, or you might be missing out on some good information.

Finally, if you have questions about your RFP, get help from a consultant. Some consultants would consider a small engagement just to review what you're put together, to identify problem spots or information you might be missing. Some analyst firms even explicitly offer a "document review" service.

But, please, don't try to posture or bluff your way through a complex acquisition process without help. It won't work the way you hope, and you'll suffer for it in the end.

Lesson #15

Scenario-based demos are helpful, but can be restricting

At some point in the selection process, a software vendor will demonstrate their product to you. You need to exercise some control over what the vendors demo...but not too much.

If you let the vendor control the demo, you'll watch something that has very little relevance to your actual situation, and that's so canned and contrived that it has little chance of translating to actual value after you've purchased it. You need to drive the demo toward usage patterns to which you expect to apply the software and that reasonably represent your organization's staffing model

Therefore, the most valuable and effective demos are scenario-based, meaning you effectively state: "This is a use case we need to perform. Now show us how we'd do it with your system."

For example:

> Bob needs to post a press release. He has the content for the press release in a Word document, and he has an oversized image to go with it. This press release needs to be approved by Mary. Then it needs to be scheduled to publish tomorrow at noon. It needs to

> have its own page with a custom URL, and it needs to automatically appear on the list of press releases both by date, and grouped by one or more assigned categories.

That's something a vendor can work with and that a customer can evaluate.

This specific example is a very mainstream use case, to be sure, but it's also helpful as an introduction to the basic editorial workflow tools of the CMS. As this demo progresses, other questions will be asked, and the vendor will take some detours to show you some flashier stuff. You'll get a nice introduction.

Beyond this basic example, your scenarios should get more and more specific to what you do at your organization, both right now, and things you want to do in the future.

If a vendor refuses to demo your provided scenarios, move on. If they have had the scenarios but haven't bothered to go through them, then take a hard pass on that vendor. Make it clear in your RFP that you want to see the scenarios you asked for.

But here's a key point: allow the vendor at least a little time to demo what *they* want you to see as well. As someone who did these demos, I was frustrated when I wasn't allowed to show a customer anything else, because occasionally I had functionality that I was reasonably sure would have benefited them greatly. Leaving some open-ended time for a vendor or an integrator to spread their wings a little and show you what makes their solution unique can be genuinely interesting, educational, and even inspirational.

Another reason for that second one: if you dictate all the scenarios, you're going to sit and watch the same things over and over.

I once sat through five vendor demos over two days, working from a list of scenarios I had written. By the end of the two days, I was a little numb. I realized that all the vendors performed basically the same way, and they all ran together in my head.

At the end of each scenario, every vendor would sit back, smile, and say something like, "I think you can see how unique we are in our ability to do this." By the fifth demo, I was snarky enough to tell the vendor that the four others had basically shown us the exact same thing.

But the problem was this: we didn't actually know which vendor was any better than the others, because we asked them all to do the same things, and they all did them acceptably well. All we knew afterwards is that we couldn't eliminate any of them based on what we had seen. The team had seen the same basic functionality five times and was left to answer the question of which version they liked better, and the gradients were sliced pretty thin.

Feature differences can sometimes be binary – one vendor has something, while another vendor doesn't. But lots of vendors are at parity with one another, and so your differentiation becomes a process of teasing apart small differences and simply deciding which one you like better.

Sometimes it's just preference. Two vendors might show you the same feature or process, and either version will work just fine. So you either eliminate that as a decision point, or just pick the one that somehow agrees with you more. Sometimes this is frustratingly unscientific, for both sides.

Scenarios based on your current experience are backwards looking. You're writing them based on what you know right now, about your process as you've been living it. To move the organization ahead, you need to allow for at least a small amount of forward-looking projection of things you might be able to do down the road. Give vendors some wiggle-room to show you how their solution might improve on what you're looking for.

And, while it's your right to drive what you want to see, have some consideration for the vendor's position when developing and preparing for a demo.

Some caveats:

- **Be reasonable about time.** If you're asking for something to be custom developed for your demo, you need to give the vendor time to do this. Don't ask for the world on one week's notice.
- **Be reasonable about scope and depth.** The vendor might fully develop part of the demo, and then talk through other parts, explaining what they would do and how it would work, in order to deliver what you want. Be accepting of this. If you're not paying them, then understand that they might not show you something end to end.
- **Be reasonable about code quality.** A lot of these custom demos are thrown together quickly. This will not be production-level code with fault tolerance and error handling. It might run slowly. It might throw errors. Debugging takes time, and unless you have paid for production-level code, you need to look past the occasional error.

Custom demos are often fun for vendors and integrators. They get to see and solve new problems. But it's frustrating when a customer somehow expects perfection in a custom demo that they're not paying for and have allowed you one week to prepare.

And, of course, keep it all in perspective, because a demo is a lot like a wedding – it's a tightly choreographed event designed to run on a specific schedule with participants who are artificially engineered to look as good as possible. The events of a wedding bear virtually no resemblance to the marriage that follows.

Likewise, the software is never going to look better than it does when it's demoed by the vendor. I'm only exaggerating a little when I say that's all downhill from there.

Lesson #16

Pay careful attention to how much vendors and integrators are willing to teach

Vendors and integrators, if you want to sell, you need to teach.

Your job is to get the customer to grok the system to some level. "Grok" is one of my favorite words, from Robert Hienlein's *Stranger in a Strange Land*.

> Grok means [...to] understand it so thoroughly that you merge with it and it merges with you.

So, to grok something means to understand something in such a way that your knowledge goes beyond simple task completion. When you start to grok a system, you can be presented with a scenario not explicitly covered in the manual or your training, and you can use the principles you've learned to extrapolate and speculate on ways to solve the problem.

This matters, because when someone groks something, they usually enjoy working with it. When they enjoy working with something, they usually want to buy it.

Customers, how well vendors and integrators do at teaching is a handy look forward to what they're going to be like to work with in the future.

While it may be hard to get down to the pure "bones" of a system during the evaluation process, a sales team needs to get you to some solid level of understanding, because that breeds comfort.

Some questions to think about:

- How much did the vendor's pitch actually discuss the product and its capabilities? And how much was vague platitudes and unverifiable promises about what their system *might* do for you? Did they spin tales of success more than they showed you the product?
- Do you have an over-arching understanding of how the system works? Is there a clear mental model in your head of how the big pieces fit together?
- Do you have unanswered questions about concepts that you specifically indicated were important to your organization?
- How much live demo time did you observe? How much time were you able to get hands-on with the product?
- How responsive was the vendor to your questions?
- Did the vendor ever concede a weakness or admit to something that their system was not designed for or not the best at? Or did they maintain that it did everything perfectly?
- When presented with a problem the software didn't solve, did the vendor show you how some underlying principles of the system might be employed to find a creative solution?

Helpful vendors will talk *to* you, not *at* you. They'll listen to your questions and provide thoughtful answers, even if it means conceding a weakness.

If the vendor can't explain their system in general terms you can understand, be suspicious. Perhaps their system is poorly architected from a conceptual level. Perhaps they're glossing over problems. Perhaps they're hard people to work with. You have a right to understand, and they need to teach you.

If you don't feel some level of mastery over something during the sales process, I suspect this confusion will just deepen once you purchase the product and dig into it. Don't buy something you don't understand, or don't feel like you *can* understand. Paying six figures isn't going to magically eliminate confusion.

A purchasing decision is fundamentally about conquering fear. I've seen customers pick less sophisticated software primarily because they felt they understood it. And in understanding it, they were no longer afraid of it.

When you're spending a non-trivial amount of your budget, a vendor owes you an explanation of how their system works, both at a practical level and a theoretical level. Make sure you have at least some foundational understanding of how the big pieces work, not just how to put Tab A in Slot B.

Lesson #17
It's easy to get excited about something new and interesting

Humans love novelty. We love to see a new and different way of doing something. And often, we just like change for the sake of change.

Change can make our productivity and effectiveness increase for no reason other than the change itself. The "Novelty Effect" says that new technology will cause increased interest in its purpose, which will temporarily provide better results. The effect tends to wear off over time as the new technology gets routine and boring.[1]

Remember how Netflix used to be amazing? When I first got a streaming box, I watched movie after movie after movie, because the mere idea of having any movie on-demand was amazing to me. As time wore on, I watched less and less because I started to take the functionality for granted and it no longer seemed amazing. Now all I seem to do is browse previews without actually watching anything.[2]

1. This is sometimes mistaken for The Hawthorne Effect, which says that the novelty of being observed and measured will temporarily increase productivity. It's not quite the same thing.

Customers, you need to be careful that you evaluate demonstrated features in terms of their *actual value*, not just their novelty compared to what you currently have. How a different CMS does something might be new and interesting, but the end result actually better?

When humans are exposed to a new way of doing something, they often get smitten with it and think it's a leap forward. In reality, they may just be bored of the way they're doing things now, and they're confusing novelty for actual productivity gains.

This is exacerbated by the mindset of a team searching for new technology. Remember that your current CMS represents all the reasons why you want a new one. If you loved your current CMS, you probably wouldn't be out looking for something else. The current CMS is boring and tedious, so anything that's new symbolizes a bold and exciting new future where all your problems are solved. We mentally extrapolate a new system to magically fill in any gaps or problems we're struggling with, and this is a narrative that vendors are only too happy to reinforce.

I've been doing this for a couple of decades, and I still have to watch out for this in demos. I've had multiple vendors show me their systems which seemed very glamorous and exciting...until I reminded myself that they didn't actually show me anything new, or anything I couldn't already do.

A couple of years ago, my partner and I got a demo of a popular, emerging CMS. I loved it – it looked amazing, fresh, new, exciting. The user interface (UI) took advantage of all the new client-side capabilities, and it was skinned with the latest design trends. I imagined all the amazing things we could do with it.

---

2. The Onion nailed this: Netflix Introduces New 'Browse Endlessly' Plan (https://www.theonion.com/netflix-introduces-new-browse-endlessly-plan-1819595604)

Then my partner rained on my parade with, “It looks great, but I can’t see that we can do anything new with it.”

Sadly, I had to agree. We were already using equivalent platforms so there just wasn’t a value-add for us. None of the systems we were using looked and operated quite as slick on a UI level, but UI polish doesn’t solely make for a great CMS. And, sure enough, when we got “under” the UI to deeper architectural considerations, the new CMS was lacking.

And that should be a benchmark for you: if you temporarily ignore UI improvements, can you literally do something new with what you’ve just seen? True, a new CMS might make an existing process easier, and there’s some undeniable value there. But does what you’ve just seen in this snazzy demo actually expand your capabilities? Will you derive some differential benefit from it?

We also talked earlier about not be blinded by “revolutionary” features to the point where you ignore the fundamentals. There are a lot of base, expected features to a CMS. Things like modeling, permissions, workflow, version management, version conflict management, etc. Don’t be so overwhelmed by something new and exciting that you overlook gaping flaws in the fundamentals.

Remember too that the joy and accomplishment of learning can masquerade as an improvement. Some systems can be a challenge to get on top of, and the thrill of mastery can be deceptive.

When I finally got a particularly challenging CMS figured out, for example, I got a rush of endorphins and happiness. And when that wore off, I realized I could basically do the same things as I could do before in a dozen different systems...I could just also do them in yet another CMS now. Outside of the novelty of being able to do this in a new system, I really wasn’t any better off than I was before.

I’ve often compared this to learning how to say “where is the bathroom?” in Klingon. You might feel like you’ve accomplished something, but you really haven’t done anything except re-learn what you already knew in a language that you’re probably never going to use.

Sure, it's an interesting party trick, but you're simply no better off in any practical sense than you were before.

As I write this in early 2020, the business phrase du jour is "move the needle," as in "We hope our Q1 campaign will be able to move the needle."

As much as I dislike trite sayings, there's some depth to this one. If you imagine an instrument panel full of gauges, the needles in those gauges are impartial – they don't care what you *do*, they simply react to changes in measurable values. In marketing, we often do a lot of activities and make a lot of noise, but if we don't change some measurable value somewhere, the needles don't move.

Remember, just because Vendor X does it a little differently than Vendor Y doesn't necessarily mean it's better. It might just be...well, *different*. And being different doesn't move any needles.

Lesson #18

RFP responses are often a team effort of multiple providers, which can be confusing

We've discussed this a bit before, but it bears repeating: Software implementation projects are usually a combination of product and services. Consequently, these projects exist on a spectrum of the number of parties involved.

On one extreme, perhaps you're planning to integrate the software and host it yourself. In this case, you literally only need software, so that's all your proposal needs to ask for. You should get responses from software vendors for their product, full stop.

Or perhaps you're looking at an open-source system, and you just need implementation help. Again, you're getting a proposal from an integrator only.

However, the other extreme is quite common: you have nothing but a need of varying levels of definition – exact and detailed or vague and amorphous – and you're looking for a coterie of vendors:

1. You need someone to plan the content, design, and marketing strategy

2. You need new CMS software
3. You need someone to integrate it to meet your needs
4. You need someone to host the resulting website
5. You need someone to support it over time

There are all sorts of companies to cater to these needs. The problem comes when the RFP is sent to organizations not capable of providing all the needed services, and then responses are evaluated with some bias against an assembled response.

Looking at the list of needs above, understand that no single organization will probably do all of that. Most integrators work with software they didn't develop – they partner with a commercial vendor, or they use open-source software developed by a larger community.

Some vendors do offer professional services, but often this is just advanced technical support, and doesn't include entire site implementations. If the vendor does do full implementation work, there's very little chance they would also provide the marketing strategy services necessary for higher-level site planning.

So, where do you send your RFP?

You can divide it up into pieces – a software RFP, a hosting RFP, a services RFP, etc. – and send them to different companies. However, this is a lot of work, and the pieces don't exist in isolation. The software vendor will need to work with the integration vendor, the support vendor will need to work with the hosting vendor, and so on. Now you're in the position of both evaluating the vendors and evaluating whether or not they can work together.

More likely, you'll send a single RFP for *all* services to a single recipient and understand that they'll recruit other companies to fulfill all your needs.

- If you send the RFP to a software vendor, they'll select an integrator to provide that part of the response

- If you send the RFP to an integrator, they'll select a software vendor to provide that part of the response[1]

The result is that the response to your RFP will be a combination of companies who claim they can work together to do what you need. By submitting a combined response, the group of companies is tacitly certifying they have self-vetted their skillsets and relationships and they believe they're offering full coverage and coordination to the problem presented in the RFP.

Often, questions in the RFP alternate between those directed to a software vendor, integrator, hosting vendor, etc. When these companies come together to coordinate a response, they have to negotiate who is going to answer what.

However, some customers still get confused by this. They expect a seamless, uniform response, and they have trouble processing a collaborative response from more than one party. And sometimes they don't understand this before they write the RFP.

To sort this out, you need to understand the relationship between the vendor, the integrator, a bunch of other service providers, and you. When the software is sold, do you still have a relationship with the vendor, or does everything go through the integrator? What if you and the integrator have a falling out? Will the vendor help you find a new integrator?

Furthermore, for the purposes of the demo, you need to know who is showing what. If a vendor and an integration partner are combining on a demo, is the project going to be vendor-led or integrator-led? These relationships run the spectrum – in some cases, the integrator is just kind of along for the ride or vice-versa. Who is your prima-

1. "Select" is probably not accurate here. Very rarely will the integrator survey the entire market and select a vendor just for your project. More likely, they'll just propose the vendor they use for everything else. The only selection criteria will likely be what they believe your budget capacity is.

ry relationship going to be with? Know that before you agree to buy anything.

If you're determined to only have one company to deal with, then I recommend selecting an integrator – either one who represents the software vendor you're interested in, or one from which you're willing to accept whatever software they believe is right for your project. Look back at the list of services you might need, and it's clear that an integrator has a better chance of providing most of them. Most can do all the strategy, planning, and implementation work, and many offer managed hosting as well.

You might still have a separate software vendor, unless the integrator somehow "fronts" the vendor.

1. The integrator might implement an open-source CMS, which, as we've discussed before, doesn't have formal vendor representation
2. The integrator might resell the vendor, which means that all business dealings with the vendor are "proxied" through the integrator

Only in those limited cases are you genuinely going to deal with a single organization.

Yes, it would be wonderful if everything was handled by one company so you always knew who was responsible for what, and you never had to worry about communication issues between parties. But this usually isn't realistic.

So, be flexible about responses. You might want a clean, single-sourced relationship, but don't automatically reject other options. You can probably get the same result you're looking for around accountability, it just takes clear communication about your concerns. The best you can do is ask good questions and make sure you're clear on who is responding to what.

Lesson #19
If you have no CMS experience, get help for your evaluations

Picking software is usually a problem of information asymmetry. In most cases, the vendors and integrators will know more than customers, just because they do this all the time. Vendors sell software all day long, and integrators do dozens of implementations a year, but customers maybe do a project like this every five years, if that.

The industry moves fast. There's a good chance that the technology and the content environment has changed remarkably from the last time the customer changed their website.

We used to just build simple web pages, now we publish content into all sorts of channels. We used to generate HTML pages server-side, now we've built entire applications from JavaScript which run in the browser. We used to have all our content in a single CMS, now we're aggregating content from all sorts of different sources.

Additionally, CMS architectures can be confusing, and different systems sometimes have wildly different ways of doing things. There's a temptation to concentrate on the methods rather than the results, and get hung up on *how* a system does something, rather than *what* it's actually doing. It's like interrogating a car salesman about the dis-

placement of the engine, how many cylinders it has, and what kind of gas it takes, only to find out that it's electric.

Occasionally, this will cause customers to fixate on odd reasons for elimination. This can be frustrating for the vendor and the integrator.

These reasons make sense to the customer based on their perspective, but they either don't understand how factual the reason really is, or they don't understand the relative priority of it. Sometimes, they just have some vague, uneasy feeling about the software that they're subconsciously projecting onto a weirdly specific reason to justify an elimination in their own mind.

Two examples, one of a feature and one of a service.

- During a selection process I was managing, a client didn't like a CMS because, "I don't like how when you rollback to a prior version, it just copies that old version and makes it the newest version, so you have another version." This statement is not untrue, and they were right in the sense that you do get two similar versions now...but this is just the way most CMS software works. It's not an outlier, so it's not like this is the only system that does this, and all the others will "solve" it.
- While we were selling as an integrator, a client started to question us deeply on how many implementations we had done on Microsoft's Azure cloud-hosting platform. We had done exactly one on Azure, but we'd done dozens on Amazon's AWS platform and we were experts in the CMS itself. I had to resist the urge to tell the customer, "This is just not important. AWS and Azure are not that different, and we know the software so well that this just isn't going to matter that much in the big picture." We won the engagement, but I was nervous we were going to be eliminated for a reason I knew wasn't relevant to the overall success of the project.

In both cases, the customer is concerned about something unnecessarily.

In the first case, it's because they're not familiar with systems; in the second, because they're not familiar with integrations. In the first case, the reason was not valid; in the second, it was valid, but just not that important.

How does a vendor or integrator handle this?

When these situations come up, it's hard for the provider not to sound arrogant ("Trust me. I know this better than you.") or like you're deflecting valid problems ("Yeah, the engine is missing, but have you seen these amazing floormats?!"). Most integrators have a vastly wider frame of perspective than the customer, but they're hobbled by the inherent conflict of interest that the customer is convinced will influence their opinion.

If I was managing a selection process, I had no problem telling my client that their reasoning was invalid. But, if I was the one selling, it was harder because the process is easy to frame in adversarial terms – a salesman is viewed as having a clear priority above the customer's best interest, and the customer will evaluate everything they say in light of that. With some hard work and honesty, I would have built up trust and credit with the customer to the point where I could overcome it and steer them in the right direction, but that's neither automatic nor easy.

My only advice here is to ask questions judiciously and carefully, to figure out the relative importance of your concern. If something is troubling you, explain what your desired end state is, and evaluate the resulting solution, not necessarily the process. I'm not saying the process doesn't matter, but think critically about whether the vendor or integrator is getting you where you want to go, just via a different route than you expected.

Additionally, talk about concerns internally as a group. What you need to avoid is one person harboring an issue which manifests as projected negativity about the entire proposal, when their specific issue could be resolved if it was surfaced and discussed.

Look for the "question behind the question." Back to the versioning example from above, some investigation was needed about why this person was concerned about rollback resulting in two versions. It turned out they were concerned about a proliferation of versions affecting the compliance process. This was resolved when we talked through it and demonstrated that compliance would be more impacted by a "true" rollback to a prior version, since then each version could represent what the content looked like at multiple points in time, whereas at creation of a new version was timestamped and therefore more accurate. In the end, the "problem" was actually an advantage.

Finally, seek help with your evaluation if you're still not comfortable with your level of experience, either with software, with integrations, or both. Consultants exist to help you identify issues, clarify them, put them in priority and perspective, and ultimately find answers to them.

You need to lean into issues by asking questions, forcing discussion, and keeping an open mind about the responses.

Lesson #20

An adversarial relationship with your integrator is never helpful

People love to negotiate. We're compelled to seek "a good deal" on everything.

We tend to frame selling situations in adversarial terms. It's us against them in a zero-sum game. Whatever they get, we don't get, and vice-versa.

Sometimes this is appropriate. If you're negotiating on a 2009 Toyota Camry for your teenager,[1] then sure, fight to the death with Larry the used car salesman. Take him to the cleaners. After all, you're never going to see Larry again. Once you drive a used car off the lot, it's your problem, and there are hundreds of repair shops in town, so your experience with Larry will be over.

Your CMS implementation, however, is different. It's not something you buy and walk away from. You're probably going to be working

1. I did this exact thing once. It cost $4,500 and has 225,000 miles. In Camry terms, this means it's just getting broken in.

with your integrator for a long time. Your relationship with them matters. If you try to be too adversarial, it can backfire.

If the integrator is any good, then they're in demand, and they've probably set rates based on that demand. Since they're still in business, they likely have people willing to pay them the hourly rate they're asking and work within their scheduling restrictions.

And you want to pay them 60% of that rate instead? And you're demanding that they start on your project tomorrow? How do you expect that to work out?

When your integrator's capacity gets tight, you might be competing for their attention with other clients. They'll have to make decisions about how to organize and prioritize their workload. When this happens, I promise that you don't want to be "the cheap client" or "the unreasonable client" or – *gulp* – both.

Opportunity cost matters.[2] In services work, when you're doing anything at a fraction of your rate while full-rate work exists, you're effectively paying someone the difference between the discounted rate and your full rate.

If a company is willing to lose money to work on your project, you need to ask yourself why. There might be a valid reason – perhaps your work is easy and they can staff it with cheaper talent, or perhaps you're offering a larger, longer project that increases their utilization rate[3] – but it usually means there's no opportunity cost to lose, which means they have no other work paying them a higher rate.

---

2. Opportunity Cost is what it costs to do lower-priced work. If Project A pays $100 and Project B pays $150, and they are otherwise identical, then it effectively *costs* you $50 to do Project A instead of Project B.

3. Utilization Rate is the percent of an employee's time which is billable. If you're paying a developer for 40 hours, and they can invoice 30 of them, that's a Utilization Rate of 75%. Smaller projects with gaps between them will drive down the Utilization Rate, which is why service providers like longer projects to which employees can apply a steady stream of billable hours.

When I was in services, I occasionally discounted, but only in exchange for something else of value – usually a large advance purchase of hours, or a contractual baseline of hours purchased per month. Even then, we had sufficient reserves of work that we were hesitant to compromise for anything less than our full rate.

If a vendor or integrator will let you get away with cutting their rate while getting nothing in return, that's probably not a good sign.

It's also worth noting that hourly rate is a terrible metric of value. If someone is charging $150 an hour, and someone else is charging $300 an hour, does that mean the second person is worth twice as much?

The idea of "value" when paying a set rate for an hour of work depends on a few things:

1. **Productivity:** How much do they get done in an hour?
2. **Quality:** Was that work done well, or did it cause problems and have to be re-done?
3. **Utility:** Was what they did even the right thing to do in the first place? Are they giving you good advice?

These factors make it difficult to compare rates across integrators. The person to whom you're paying $150 an hour won't look so cheap when they take three times as long to deliver anything. Remember, if someone consistently takes twice as long to do something than it should take, then their hourly rate is effectively double what they quoted.

When negotiating services, understand that you're really negotiating the assumption of risk. You're deciding who is going to own the risk of loss if things don't work out.

A safe rule here: If someone accepts a risk, they will seek to be compensated for it.

Let's look at risks on both sides:

- The **integrator's risk** is that they will have to expend more paid labor than they receive in revenue. The customer might have requirements that were hidden or misunderstood that the integrator has to fulfill.
- The **customer's risk** is that they will pay for something – or pay more than they planned for something – and not receive what they were promised. The website might not launch, or it might launch and have stability problems or not have all the functionality they were promised.

Both sides will seek to either reduce these risks or be compensated for accepting them.

- The **customer** will try to cap the cost of the project as much as possible – everyone wants a "fixed fee." This cap represents a risk for the integrator because the requirements might expand but the revenue from the project will not.
- The **integrator** will try to reveal all requirements so there's clear agreement on what they're responsible for. If the requirements are vague or unknown, the integrator will (1) not commit to a set price, or (2) pad the quoted price to ensure that they're covered if the requirements expand. If the requirements don't expand, the integrator gets to keep that extra money, which might seem like a windfall, but is really just *compensation for the risk they assumed*.

What should be clear from both those items is that the requirements of the project are the key point of negotiation – they will both deeply affect both parties in the project. It's in everyone's benefit to get all the requirements out on the table as unambiguously as possible.

What you're trying to avoid is a hidden or unknown requirement that comes up after the price has been set. Unfortunately, this still happens all the time, and then the disagreement becomes whether or not both parties knew about the requirement.

Ideally, you would fall back to project documentation, but sometimes that's incomplete, non-existent, or the parties will claim it doesn't accurately represent what was discussed or assumed.

When disputes like this arise, who takes the loss? It depends on a lot of factors, but unless some fault is clear, then one side is going to simply concede or negotiate the loss in hope of benefiting the long-term relationship. More than once, my company provided free labor to salvage or improve a relationship in the longer term.

This means it's to your benefit that the integrator can see a genuine, profitable long-term relationship with you. If they can't, then there's nothing to work toward, and they're much less likely to work with you when disputes like this come up. Worst case, if they know the relationship with you is over once the project is complete, then they have nothing more to lose so their negotiating position becomes very rigid.

Some customers think that strict contracts are the ultimate protection – all they need are ironclad clauses and penalty payments if the project runs off schedule. This is fine, but remember that any smart integrator is going to want to be compensated for risk, and a penalty payment represents a risk. If you want a $1,000 discount for every day that the project is late, then the integrator is going to mark up the cost to balance that risk out.

Additionally, before getting too smug about your punitive and airtight contract, you might want to role-play your worst-case scenario. If the project runs off the rails to the point where you start to lawyer up, how do you think that's going to play out? The legal system doesn't turn on a dime.

If you do actually sue your integrator, understand it might be *years* before you're compensated – if at all – and that won't help you in terms of this specific project. If anything, it will set it back further since you'll clearly have to find another integrator.[4] You may win in the long term, but your project is still going to fail in the short term.

The basic point is this: Your relationship with your integrator is likely to be a long-term one, and it needs to be mutually beneficial to both sides. When one side no longer feels like there's long-term value, then it becomes difficult to work through any issues. And there *will* be issues.

No relationship is proven until it's tested. The honeymoon *will* end at some point, and you don't really know the other side until you've had a disagreement and worked through it.

Remember, this is not like buying a used car. This relationship is not purely adversarial. You need to be concerned with the well-being of the other party, because their long-term success will very much affect yours.

4. Also worth noting: If your new integrator knows you sued your last one, they're going to make assumptions about you, which they might interpret as risk. They will build in compensation for this. If a customer is difficult, word gets around – this is a smaller industry than you might think.

Lesson #21

The lure of "out-of-the-box" functionality is usually misplaced and illusory

"Out of the box" (OOTB) is the marketing phrase I probably dislike the most. These are things a system supposedly does with no development or configuration, as in "How much does the system do out of the box?"

OOTB isn't inherently bad, it's just misunderstood and frequently misrepresented. If a vendor is honest about their product's capabilities, and customers know and understand what they're getting (and, more importantly, what they're *not* getting), then some OOTB functionality can be positive. But, sadly, that's rare on both counts.

Having a lot of OOTB functionality is invariably marketed as a good thing. If a system does a lot "out of the box," then that's great, and supposedly the sign of a competent system. If the system arrives "unassembled" and has to be developed, then this is a sign of a bad system, like when you're having to assemble toys until the wee hours of Christmas morning.

I was initially tempted to say the phrase is dishonest, but it might not be – the software being advertised might actually do a bunch of

stuff "out of the box." Everything does *something* out of the box, the question is really one of quality and applicability. Is the OOTB functionality that's offered any good, and does it apply to what you want to do?

The truth: What a CMS does OOTB is probably not specifically applicable to your situation. It's usually very general, and there's only a slim chance it works the way you want it to work. A system can't be instantly good at everything. So it's either poor at a wide range of functionality, mediocre at a smaller range, or actually *good* at a narrow range.

I'm reminded of the Danish physicist Niels Bohr, who said: "An expert is a person who has made all the mistakes that can be made in a very narrow field." To be really skilled at something, you have to narrow your field dramatically.

A lot of allure of OOTB is for customers who had a bad experience with whomever was integrating their website, be it their internal team, or an external integrator. This was often manifested in long delays between request and delivery, and therefore the customer mindset is that the end user should be able to do everything without having any external help – we just push some buttons and everything works...

...until it doesn't.

Beyond very basic websites, I've never seen the OOTB story end well. As mainstream as your use case is, I promise there's something that will be different than whatever was "in the box."

Pre-built features can only address "patterns." These are common usage scenarios that the vendor can predict in advance, and develop pre-built solutions for. The value of this depends on the strength of the usage pattern – is it something that *everyone* using this genre of software wants to do, *all the time?*

For some types of software, this might be true. Consider accounting software – there are a lot of patterns baked into accounting that

are true throughout that industry. The patterns in accounting are so strong they're built into a set of guidelines that students learn in college called the Generally Accepted Accounting Practices (GAAP), and sometimes they're even legally enforced by organizations like the Federal Accounting Standards Advisory Board (FASAB). Large swaths of accounting practice are factual, non-debatable, and enforceable.

There is absolutely *no equivalent* for GAAP or FASAB in digital marketing. A "best practice" in digital marketing is a glorified opinion, perhaps qualified by "this worked for me."

Occasionally, you see strong patterns in certain applications of CMS. Intranets, for example – employees interacting with company data fall into common usage scenarios (searching an employee directory for a phone number, for example). The "intranet-in-a-box" market is quite strong because the usage patterns in that space are relatively well-known. They're narrow and deep.

When considering a mainstream CMS – which is a generalized framework for managing an open-ended repository of any kind of information with the intention to deliver it through myriad channels to a wide variety of audiences – the patterns are much weaker. Every client wants to do something a little bit differently, because they think their customers are going to interact with their particular organization and information in unique ways. The usage patterns are broad and shallow.

You'll usually always have to modify the system to an extent that requires a technical resource – hence our usage of the word "integrate." And systems that have gone out of their way to build a lot of OOTB functionality will usually have many built-in paradigms and methodologies that differ from how you want the system to work, or how your developer thinks it *should* work. Working around how a vendor built an OOTB feature always takes more time and is less functional or stable than if the vendor just provided the tools for a developer to fine-tune it from code to start with.

This is called "The Last 10% Trap." The first 90% of functionality is handled OOTB, but trying to get that last 10% of what you want ends up taking twice as long as that first 90%, because you're working with things that weren't designed to change. But by that point, you're in too far to back out – you're trapped.

A mobile home is a way to get shelter quickly, but when you need to add two bedrooms, well, good luck with that. You either live with the limitation, modify it foundationally so it doesn't much resemble what you started with,[1] or throw the entire thing out and start with something built in a more foundational, expandable way.

Every feature decision is based on the opinions of the developer who designed it. Systems that can only work as-designed are said to be "highly opinionated." These opinions can be rigid, and you might not agree with all of them, but they enforce their opinions, whether you agree or not.

Unless you're totally on-board with the opinions of the feature, the tail is going to wag the dog. You're going to compromise your requirements because "that's the way the software works" and it's just too hard to change.

There's a balance, of course, and I'm not advocating that you develop everything from scratch. Competent systems, however, have common-sense boundaries around what has to be built from scratch, what can be configured from code, what can be configured from the interface, and what they claim "just works" OOTB.

When you remodel your house, you can move furniture around and paint, and maybe knock down an interior wall or two. At some point, you will run up against a load-bearing wall that holds the roof up.

---

1. A friend once explained this by saying they could technically win the Indy 500 by modifying a 1973 Ford Pinto. But by the time they were done, their Ford Pinto would look and function *exactly* like an Indy car. Also, there wouldn't be any single part of the Pinto left remaining.

You can't knock this down yourself, and you'll have to call a contractor to perform a more complicated building project.

Different CMS systems have different numbers and placements of the load-bearing walls. You need to find out where these are. If a system claims it has none and you can "gut the entire house" if you want, be skeptical.

A decade ago, a particular CMS advertised itself as having a library of pre-built components, and all you had to do was "drag them out on the page." In reality, the widgets were horribly designed, generated terrible HTML, and could not be changed – all the HTML code was compiled directly into them. I did 25+ implementations with this system, and never used the OOTB components even one time. I spent most of my time – and my client's budget – working around the limitations these components imposed.[2]

If you're still convinced you want a system that's promoted as having everything OOTB, then know you're agreeing to this statement

> I completely trust the opinions of whomever designed and built this feature, and I swear I will use it only in the way it was intended and mercilessly suppress any desire to ever customize beyond the specific options they have built into it. I promise to be happy with what I got.

Put another way: What is driving the final product you are launching – your vision, or the features in the CMS? If the latter, then sure, buy something that works only one way and let it define what you release. But if you have a specific plan you want to execute to arrive at a specific final product, then you need a framework on which you can build.

---

2. I went to this company's annual customer conference once. Every session seemed to be a lesson in how to rewrite massive sections of the CMS to do what you *really* wanted to do, to the point where the original CMS was almost unrecognizable.

Lesson #22

Poor governance and vague ownership do far more damage than a lack of technology

Humans are good at projecting issues. We take issues that come from a specific problem, and we project (verb form) them onto something else. We often do this with organizational issues by making them look like technology issues.

Customers will sometimes get into deep arguments internally about features and functionality, when they're really talking about larger, underlying paradigms of how a CMS exists within their organizations. They think they need "digital transformation," when they really just need to resolve some basic organizational issues – or at least discuss them to some degree before embarking on a large-scale implementation project.

Customers are taking the *real* problem and morphing it into some other issue – like the lack of brand-new software – because that's easier to discuss. The lack of software is a clear gap that can be filled, thus re-formulating the problem into something binary and solvable: "We don't have The Thing™, but we can get The Thing™, and that will fix the problem!"

When the problem is hard to solve, we tend to turn it into a straw-man[1] problem – a façade of the actual problem that's easier to solve.

These issues inevitably poke through the surface when discussing staffing, management, or ownership issues with the website. Many organizations have defaulted the website ownership to Marketing, but then never give them the tools or autonomy to make it work. Marketing ends up fighting for development resources with IT, when IT has conflicting priorities between the website and other pressing infrastructure issues.

This is often the *real* problem. But it's buried under layers of misunderstanding, lack of communication, and sometimes outright animosity. Organizations usually find it's easier to just throw software at the problem.

I remember a discussion about a website which contained a detailed and complex mortgage loan application. There was lots of disagreement about all sorts of features and their relevance. Marketing and IT were both at the table, and it became obvious that their disagreement was foundational and *way* deeper than just software.

- Marketing felt that the website was content based with some application functionality
- IT felt that the website was a line-of-business concern, with some marketing functionality

When I gave voice to this framing, there was silence around the room. Not because everyone agreed (if only), but because they realized how far apart they were, and, eventually, that their views on it could never be reconciled. They simply did not agree who owned the website, which was a very basic point of contention.

---

1. In logic and debate, a "strawman argument" is when you twist your opponent's actual position into something that's easier for you to discredit and dismantle. You turn it into a "man made of straw," which is easier to defeat.

IT had been deferring to Marketing, but if the organization was going to spend six figures on a new software platform, IT considered that an "enterprise software acquisition" and it wanted to control that decision.

I call these "worldview" issues. These problems come from the most basic lens of how you view your world, not from specifics about the actual project. These issues can be so foundational that people don't even see them. You lose sight of the forest for the trees.

So often we hear the same argument from Marketing when investigating a new CMS: "We need to be able to do things without having to involve IT." This means that organizational and governance disagreements are so acute that Marketing is completely switching technology platforms as a solution.

In situations like this, I've often wondered if Marketing should just have a set of developers assigned solely to them. So, Marketing would have development talent they could manage and allocate themselves.

When Marketing complains about having to involve IT, what they're really saying is that they don't want to fight for human resources amidst all the other concerns that IT is working on. If Marketing can't remove the bottleneck and become a priority, then it plans to go around the bottleneck altogether. Purchasing enterprise software and paying an outside firm to completely change the website is easier than just getting a new job requisition or reassigning someone internally.

I'm not claiming that every CMS purchase could be avoided, but at least some portion is unnecessary, and here's the worst part: they often won't even solve the organization's problems. They'll get a new CMS, and they'll either be arguing about the same problems, or they'll create new ones by introducing a new software stack and a new expense to have a third party manage it.

No one wants to deal with human issues, because they're frequently messy and awkward. They can involve value and competency judg-

ments about co-workers and existing work product, and they can shift the balance of power (and budget) inside an organization. No one wants to lose an area of responsibility because they've been evaluated as lacking in its management or execution.

The other problem with organizational issues goes back to humans' love of novelty. Organizational and management issues can be easier and lower cost to implement, but they're...boring. They don't involve new software or tools.

Additionally, there's always the unspoken question of why the organization didn't fix the issues before. If someone has implied that new software will make a situation better, then they're also claiming that *not* having the software is clearly the reason why the problem persisted so long in the first place.

If an organizational change is proposed as the solution, it begs the question of why the problem wasn't fixed earlier, so we stay away from that.

Patrick Lencioni wrote in *The Advantage*:

> Most leaders prefer to look for answers where the light is better[2]...and the light is certainly better in the measurable, objective, and data-driven world...than in the messier, more unpredictable world of organizational health.

In the end, it's much easier to deflect and defer problems in search of a software solution that will Make Everything Better™.

Before you start on a software and services project, answer these questions:

---

2. A reference to an old joke about a guy looking for his keys under a streetlight, even though he lost them on the other side of the street. When someone asks why, he responds, "Because the light is better over here." The larger point is that there's a human tendency to contort actual problems into something that we find easier or less awkward to solve.

- Who owns the purchasing decision for both software and services?
- Who is responsible for the improved result the project is supposed to bring about?
- Who has the resources – budgetary and human – to manage and develop the new platform post-launch?
- If the answers to the above questions are not the same person or group...*why not?*

Cynically, neither vendors nor integrators should complain about this state of affairs – it keeps us in business. Still, it's tough to watch an organization that just can't get out of its own way. Great software and a great implementation will wither and die in the face of organizational confusion and disagreements.

Lesson #23

Launch day is not the finish line, it's the starting line

This book is about preparing to select software and services for your digital transformation project. So, you're preparing for the project that comes before your *actual* project.

The actual implementation project might have a 6-9 month timeframe. When you include selection, the entire project cycle might run for over a year.

And then it's finally done. Whew.

Just kidding. It's not done. It's never done.

The strict definition of a project is something that:

> [...] is not a routine operation, but a specific set of operations designed to accomplish a singular goal; a project is temporary in that it has a defined beginning and end in time.[1]

1. This definition was cobbled together from various resources offered by the Project Management Institute.

The implementation of your website is a project, sure. Once it launches, it becomes a *product*.

The companion to project management is product management, defined as the maintenance and development of a set of features over time. Your website project eventually becomes a product, which creates an entirely new set of challenges.

Customers often don't plan for the day after launch. Once the website launches, there's usually some immediate clean-up and stabilization issues to handle, but once things are running smoothly in production, you'll often lose several things:

- Your **budget** will likely be slashed. The organization was probably pumping money into the project, but they expect to be able to turn that flow off now.
- You might have had **extra staff** working on the project, from a variety of departments. They're probably thankful to go back to their regular work now.
- If you hired one or more **contractors** to do the work, their role might have ended with launch.
- The **attention of the executive-level** at your organization might have been focused on the project and prepared to fight organizational battles when called on. Now they consider it done and are ready to check off that box and move on.
- The entire organization might have had a **general sense of urgency** and had banded together to get all the work done.
- You might have even lost **your own enthusiasm** for the project. It's been a long haul, and now you'd just like to not think about the website for a couple of months.[2]

---

2. Early in my career, I built the website for an NFL team. I spent month after month in "death march" mode to get the project completed. For years afterward, the color combination of that particular team's uniforms gave me low-

The day after launch can feel like New Year's Day in Times Square. It's obvious that some exciting things happened the night before, but no one is left except the clean-up crews trying to get a handle on the mess.

Customers tend to get "development tunnel vision" where you're so focused on getting to launch that they don't consider what comes after. Some projects never make the leap to a continuing, managed product. They get launched to great fanfare, then just linger with no subsequent progress.

Here are some things you need to consider:

- Features *will* get thrown overboard at the end of the project. Some things just won't make it under the schedule and budget, and they'll get pushed off to "Phase 2." The day after launch, you need to take stock of these things, and decide if you're actually going to execute a follow-on project for them.
- You might need new staff. Remember all the advantages you promised the CEO so she would agree to the budget? Who is going to do all that stuff?
- Existing staff will need to be re-trained. Sure, you probably did some training during the project, but when they sit down to work with the system, they'll have forgotten half of it, they'll have new questions, they'll find edge cases, etc. It's a great myth that training is a singular activity. It needs to be an ongoing program.
- You'll need to work out support issues. The first time you have a problem with the website, you'll probably realize you didn't plan comprehensively enough – or at all. Do you have monitoring so you even know a problem exists? Who is your first call? How do you escalate? How do you communicate the issue internally? What level of post-mortem will you require? What will you do

---

level anxiety.

with the resulting information?

- What happens when you want to make that first planned change? You might not have your integrator anymore, nor all the developers that IT loaned you during the project. Who is going to do this work? How do you get it authorized?
- You'll need to develop new processes around the new functionality. All those new webforms you're making – who is going to route and respond to the resulting emails? Yes, personalization is great, but who is going to maintain the rules? And can anyone interpret all the new analytics?
- You'll realize that some things just don't work as well as you thought. Some new features the CMS offered aren't as great in practice as they were in theory. Perhaps your staff can't adapt to them, or perhaps there are limitations that didn't show up in the demo. You'll have to make decisions about whether to keep or abandon them.
- Lots of people around the organization will have new ideas. Some won't like the changes. Others will be inspired by the new website and platform, and they'll want to get their pet project started. How will you evaluate and prioritize these?
- You might be juggling new vendors. You'll have the new CMS vendor, and perhaps a continuing relationship with the integrator, plus any number of subcontractors. How do you manage lines of communication and accountability?
- You might have new budget management issues. A new website can be an expensive thing to maintain, so you'll need to plan your spending around this Brave New World of tools, processes, and organizations.

Understand that a project like the one you're considering will probably create more work than it will resolve.

There are certainly some situations where outdated technology was creating work for your organization, and a good CMS and implementation can fix this. But what companies often find is that a new set of functionality creates *new* work. The organization expects some return on its investment, so it's going to want to see those new features in use. Hopefully, the new work will be productive and provide benefits to the organization, provided it's staffed and managed comprehensively.

There's nothing more demoralizing for an integrator than to get a phone call from their customer a full year after launch asking, "Can you show us how to log in to the CMS again?" At that moment, it becomes obvious that the customer considered the long-past launch day to be the finish line for a project that they just wanted to end. And that project never turned into a product that was actively managed in a manner designed to manifest all the benefits that were promised along the way.

You need to be constantly looking *past* launch day. What happens on day two?

Lesson #24

A lot of results you're promised will require considerable effort from humans

Vendors love to talk about salesly concepts like "digital transformation," and they'll show you amazing case studies of what people did with their software. What they'll gloss over is how much work this will require from salaried human beings to make it happen.

This is true of anything. Humans can improve their tools, but they still have to *use* them.

I've been a car guy my entire life. I would watch auto racing and imagine how fun it would be to have a sports car I could toss around like that.

And then my wife bought me a Porsche 911 for my 40th birthday, and I quickly learned something: I just wasn't that great of a driver.

I wasn't afraid of spirited driving every once in a while, but the car's capabilities were *way* above my skill level. In fact, the car still has the same set of winter tires it came with. Sometimes I'm tempted to buy higher-performance tires, then I remember that I've never even

approached the outer limits of the snow tires, so I'd be buying capability I didn't have the skill to get any real use out of.

To race a car requires (1) a capable car, and (2) *a really good driver*. I only had one of those things. If it could, my Porsche would have rolled its eyes whenever I sat in the driver's seat.

When a vendor shows off a case study of how Customer X did something amazing, understand that this wasn't done on autopilot. There's no magic switch or menu option that makes this happen. The software was just a cog in a much larger machine, which included adequate staffing, intelligent processes, and often external consultants.

If you're considering software *and* services, this can be easier, because many integrators offer marketing strategy and operations services to help you get this done – so the agency provides the people. But everything has a price attached, and qualified people aren't cheap. For a large-scale marketing transformation, be prepared to pay four- or five-figures *per month* after the implementation just to run the software.

Some software vendors are even moving into this market. They're selling the services to run the product along with the license. It's like selling a car so complex that the customer has to lease a driver when they buy it.[1]

Any selling process is a zooming in and out of small scale and large scale concepts. When talking about results, vendors will make large-scale claims that often leave out a smaller-scale point: "Our software provides a toolset of functionality which can be used by humans to achieve larger goals. Our software is a necessary *but not sufficient*

1. I visited Hanoi, Vietnam a few years ago. This city is legendary for its traffic and driving patterns. My host was from England, living in Vietnam temporarily. She picked me up at the airport in her car, piloted by a Vietnamese driver. It turned out that, for non-natives, the dealership sold the car and driver services as a package, because driving in Hanoi is just that difficult.

tool to achieve this larger goal." They're assuming you will provide the human skill to use the tool they're selling you.

Be a little guarded when vendors speak of "experiences" and "journeys." What they're saying is that their software is theoretically able to help you do all this stuff...with considerable effort from humans – you or other humans you are paying.

Remember that the software you're examining probably has newer features and functionality than what you have now. Almost by definition, it's different than what you're used to. If it wasn't, then why would you be considering a change? This being the case, you're going to need to develop or buy new skills to use the software. The current processes and skills you have might not be enough to run it.

For example, personalization is great, but are you really in a position to perform a fine-grained analysis of all your audiences, figure out how to identify them all, and create and maintain multiple versions of all your content? Most organizations struggle to maintain *one* version of their content, much less five or six.

Some customers believe their skill level and processes are far in excess of what their current software can support, and they need new software to help them fill their potential.

And this is what humans do, naturally. We're evaluating something new, we project ourselves into a world where we're competent professionals who clearly *need* this amazing functionality. If a vendor shows us a new feature, we *want* to be a professional who needs that. We love the idea that this advanced technology will fulfill some imagined "software gap" in how our current technology is not fulfilling our needs – as if we're so much further ahead.

And in our heads, this means we don't need more people. The new software just means our existing people can use the full array of their skills. The software is the *only* puzzle piece missing.

Vendors perpetuate this. Not out of dishonesty, but purely because they have no alternative. No vendor is going to tell a customer, "You

aren't nearly sophisticated enough to use this." Don't confuse vendors with consultants or therapists – they're not being paid to give you the unvarnished truth about anything.

Certainly, some customers *are* being held back by a "software gap." But more often, this is not the case, and I've seen customers bite off way more than they could chew by not considering the human element required to get where they wanted to go. They found they could never do any of the amazing things they were sold because they simply didn't have the headcount or skillset necessary. Worse, new capabilities often had the effect of creating *more* work.

Many times, the gap is in human skills and capacity, not software. Often, the customer isn't even using their *current* software to its full potential.

As the customer, it's your job to be honest with yourself about what your organization is capable of. Do you have the staff and skill necessary to use all the tools you're being sold to the level necessary to justify the investment?

Lesson #25
Software is not your savior

Here's a strong statement: in most cases, *you* are the biggest risk to the successful completion of your project. Sure, projects sometimes fail due to technology shortcomings or a bad integrator, but that's far less common than the customer being the ultimate source of why the project didn't fulfill its expectations.

But first, let's talk a bit about the concept of "failure," because it's not clear cut. We can actually divide it up into multiple types of failure.

- An **Abortive** failure means the project failed to complete or launch at all. Sometimes, this isn't even a failure. If you get 15% into your project and realize it's not going to work, canceling it at that stage is the right decision and should probably be considered a win.
- A **Quantitative** failure means the project launched, but failed on some numeric metric, like budget or schedule. Usually, it cost too much or took too long. These failures are very common (even expected, in many cases), but are recoverable. If something is late or over-budget, but then goes on to succeed on every other metric, then the sins of development will be quickly forgotten.

- A failure of **ROI** or **Goals** means the project may have launched on-schedule or within budget, but it never achieved the intended end goal. These failures are sadly rare because to fail on metrics, *you have to have some*, and few projects do. It's rare that a customer has identified the goals of their project in terms of numeric values that can be tracked and compared, assuming you even had a baseline to start with. So even if a project does fail on goals, the goals are usually implied or assumed, so no one can tell for sure.
- Which leaves us with failures of **Expectations**. This is when your project launches on-schedule and within budget, but there were no goals or metrics to track. All you had were expectations. Sometimes the project fulfills them and is considered a success because no one feels let down, but other times, one or more people assess the result after the dust has settled and say something like, "I just thought it would be better than this" or "We still have Problem X, and I thought the project was going to fix that." Often, they won't verbalize it at all, and it will remain felt yet unspoken.

Truth: A project's "success" or "failure" is often chalked up to gut feeling.

Six months after launch – just long enough for the novelty of a new thing to wear off – a bunch of people are going to ask themselves, "Are we any better off now than we were before?" The answer to that is then chalked up as success or failure.

And where does this gut feeling come from?

A lot of it is based on "perceived differential competence." Organizations will base their opinion of their own competence on what they see other organizations doing. And since what organizations do is often hidden – it's hard, for instance, to conclusively identify or evaluate an A/B test being performed on you – then we often fall back to what we *think* other organizations are doing.

And where do we get this fear from? Mainly from what other people talk about. This means articles we read, case studies and white papers we find on LinkedIn, conference presentations we sit through, and reports that analysts write.

Facebook gave us FOMO – Fear of Missing Out – on a personal level. LinkedIn did the same thing professionally.

The problem is survivor bias, which says that history is written by the victors – the only things that survive to be written and talked about are the things that are interesting or that worked. No one writes about the projects that didn't accomplish anything interesting. No one writes a post on LinkedIn that boils down to, "We tried this thing, and it didn't work."[1]

And so we chase stories of success. We run after the things people are claim were successful when we have no firsthand knowledge of what those actual results were or if they apply to our business. We just want to resolve the fear that our competitors are doing more than we are.

Steven Furtick is a pastor from North Carolina. He made a tweet back in 2011 that has resonated with me ever since:

> One reason we struggle with insecurity: we're comparing our behind the scenes to everyone else's highlight reel.

Without a way to measure the results, any victory is hollow. We often have no idea what end state we're even looking for, much less how to measure whether that made an impact on the state of the business.

---

1. This is well-known in the scientific community as "publication bias." The only experiments that get published are those which came to some clear conclusion. This gives the impression that science is much more determinable than it really is. For every paper in a peer-reviewed journal, there were likely several dozen experiments that yielded no clear conclusion and were therefore never documented.

External, marketing-focused initiatives are obvious, but we see this play out internally as well. Content managers live in fear that their content assets are "out of control." They believe that, at any given time, every single asset the organization owns should be in a controlled state and subject to complete transparency. Content managers want to get their arms around everything, because they're convinced this is the normal state that all their competitors are in.

There's a corollary here with new parents. With your first baby, you want to do everything "right." You might not know why you're buying organic diapers made from cotton fibers grown in the Brazilian rainforest, but there was some article in a parenting magazine that told you it was the "right" thing to do and that perfect couple down the street does it, so you figure it's normal.

When the second child comes, you've loosened up quite a bit. By the time that fourth child rolls around, you're just hoping they don't eat too much dirt when they play outside. And even that's negotiable.

Digitally, this manifests in what I call "control fixations." These are things like:

- **Dashboards and Reporting Tools:** Customers will over-emphasize these with the best of intentions, dreaming about the levels of control and transparency they'll enable. But even when this reporting exists, they hardly ever use it.
- **Form Building:** Customers want to be able to build highly complex webforms – even to point of replacing line-of-business applications. In reality, they might only have 2-3 forms on the website that hardly ever change.
- **Multi-Site Management:** Customers want to manage the entire array of their organization's digital presence from the same system, when sometimes the right answer is just to run your temporary campaign microsite out of a separate CMS instance.
- **Workflow Engines:** Customers dream of control by automation, with complex workflows routing content for approval and pub-

lication. Relatively few organizations (e.g. newspapers) push enough content through a sufficiently skilled team that this makes sense. For most, a simple, feature-thin system is better.

All of these feature studies speak to issues of control, and our fixation on it. We're convinced that we'll succeed by locking everything down with perfect software. That's the gap we just have to close.

But remember, *don't pin all your hopes on software*. There are people and process issues that need to be addressed inside your organization. Too many customers will simply throw software at a problem, then be upset when the software didn't fix everything that ailed them.

In reality, any "digital transformation" – insofar as that term actually means anything – is a combination of:

- A reasonable and clear vision of the desired end state
- Sufficient and appropriate technology
- The correct initial implementation of that technology
- Sufficient ongoing support and expert consultation around that technology
- Sufficient and ongoing training of the people who use that technology
- Correct oversight with ongoing budget allocation and clear executive support

And, most important of all –

- Constant evaluation and evolution of goals in the face of changing market conditions and organizational goals

Too many times, customers focus on the technology to the exclusion of everything else. "The software will save us!" is the standard rallying cry.

When software myopia is combined with the unspoken and unacknowledged reasons why a customer is pursing the project in the first place, that combination becomes the biggest threat. Customers are in denial about what they want to accomplish and why, and have unrealistic expectations of what raw software is going to do for them. When the results don't instantly duplicate what they're reading about in their LinkedIn timeline, they get a gut feeling that tells them the project didn't succeed.

Software is one part of a much larger landscape of solutions. Customers need to be honest about their reasons for pursuing a change, and realistic about its limits to improve an organization without the support of a larger transformational effort.

That is the biggest risk to any project. And that starts and ends with you.

Conclusion

This all feels very cynical, I know. And like I said in the introduction, it's a mix of procedural advice and human and social engineering. You need to exist somewhere in between a project manager and organizational therapist.

Any software and services selection process is a mixture of organizational need, human emotion, technical requirements, and business realities. At the risk of drama, it's a microcosm of the human condition all rolled up into six or eight weeks. It can be a roller coaster, except that you have to live with the results for years.

To distill it all of this down to some general principles –

First, accept that it's a messy, imperfect process that's not defined by a simple result of "success" or "failure." You just do as well as you can.

Technology selection will likely not doom your project, nor will it be the single thing that causes it to succeed. It's one factor among several others – in particular, people and processes – that all mix together to move you along a spectrum.

Success or failure in these projects is not binary. You don't even "succeed" or "fail" in simple terms, as much as you arrive at a point some distance to your goal.

Second, try to be humble and genuine. These processes feel adversarial by nature, but lots of problems can be chalked up simply to people – on both sides – posturing and bluffing. Know that you probably can't bluff as well as you think you can, and when this is revealed, you'll be in a worse position because of it. Be honest with vendors and integrators, and clearly communicate with them about your concerns and timelines.

Third, get help. If you're not an experienced buyer, then find a consultant to help you examine what you need, write the RFP, evaluate the responses, and sit in on the demos.

Selection consultants don't make the decision for you, they just help you put a breathtaking amount of information in perspective, so you can know what's important and what's less so. You can negotiate the scope of the engagement, so you don't need to worry they'll take over the entire process. If you just want some spot help, state that and look for someone willing to engage on that level.

Looking back through this book, there's one general analogy I've used more than any other: marriage.

Dozens of times, I've compared the software evaluation process to dating, the implementation to being engaged, and the system you're left with as marriage.

I'm reminded of a blog post written back in 2003 entitled "Why Content Management Systems Are Like Relationships." It's resonated with me in the 17 years since.

> **Why Content Management Systems Are Like Relationships**
>
> - There comes a point in your life when you feel that you really should be in one.
> - They're expensive up front, and you never stop paying for them.
> - People who are already in them tend to be a bit superior about being in one.

- Both aren't that easy to get into and you wonder if you're choosing correctly – you're sold this myth that there is "the one" out there for you.
- When you get into one, you have to adapt your behavior to suit your partner.
- Once you're in one you miss the freedom that you have before and you find it's invariably more work which you fail to see the need for.
- You'll be scorned or ridiculed by people who don't see the point in them.
- After a few months of excitement, you realize that you're basically doing everything that you were doing before.
- You occasionally have *wow* moments that make you realize it was all worthwhile.
- You can only really use one, conflicts arise if others hang around.
- Soon you notice new ones, and your affection begins to wander – but the upheaval of changing is just too much to consider moving on.
- After a while, keeping the set-up going becomes more important than anything you once thought you might get out of them, or even put in.
- Having been in one, you need some time to recover before you start your next.

Clearly, this post is dripping with cynicism,[1] but many of the things written here echo points I've made throughout this book.

1. Also, given the timeframe it was written, the author is comparing using a CMS to using static HTML, which might not be a valid comparison any longer.

In fact, let me quote from the same *New York Times* article that I mentioned earlier:

> We need to swap the romantic view for a tragic awareness that every human will frustrate, anger, annoy, madden and disappoint us – and we will do the same to them.
>
> [...] We should learn to accommodate ourselves to "wrongness," striving always to adopt a more forgiving, humorous and kindly perspective on its multiple examples in ourselves and in our partners.
>
> *Why You Will Marry the Wrong Person*
> *New York Times*
> *May 28, 2016*

You may be wondering why I'm finishing up the book with this, but the reason should have been staring you in the face all along:

Nothing is perfect in this industry, and no software exists that will magically meet your every need.

Instead you need to find a system and a service provider that comes as close to all your needs, but is flexible enough to modify itself to suit you in ways that are critical. And, in turn, you need to a take a long look at your organization and be ready to accommodate some software shortcomings, change some bad habits, and make another modifications as necessary to fit the tool you select.

Anything less than that will leave you profoundly unhappy, while you wait out the inevitably messy divorce sometime down the road, and start the process all over again.

Do your best. Good luck.

Are There More Things You Should Know?

Whenever you write a book, there's a struggle to know when you're done. At any point, you have a dozen other things that you wonder if you should include, and there are probably dozens more you're not thinking of.

My hope is that the book you've just finished is simply the *first* set of things. A few lessons didn't make the cut for this edition because I couldn't validate them past my own experience, or I couldn't sharpen their definition clearly enough.

Additionally, you might be thinking right now: "I can't believe he didn't talk about [insert your own situation here]!"

If this is the case, please reach out. I'd love for this book to spark a dialogue among customers, vendors, integrators, and analysts about other lessons I didn't cover here. When I get enough, I'll issue another edition.

Let's keep this conversation going. I'm all ears.

deane@deanebarker.net

Acknowledgements

Thanks to Episerver who agreed to publish this when most vendors would have run the other way.

Thanks to Heather Boutwell who read the original document, thought it would make a decent book, and became the champion and cheerleader for the project.

Thanks to Amberly Dressler for her editing.

Thanks to Justin Anovick and Ed Barrow for giving me the keyboard time to get it done.

Thanks for Sam Otis at Blend Interactive for his design.

And thanks to all the customers of Blend Interactive over the years that gave me the lessons and experiences – both good and bad – that I've written about here.