Enterprise Software *Confidential*

By Philip Yurchuk, CEO of *Made Up Name*

The account rep from the vendor was exasperated. We kept saying the same thing, week after week, in our status meetings: we need you to catch up. We've got a deadline to meet, and you're slowing us down. Finally, he lost his cool.

"Look. We're all **really** impressed with what you're doing. But you have to understand, it typically takes us nine months to launch an ecommerce site, and it costs a million dollars. So, can you be a little patient?"

Did he just-? In front of the **customer**? He did. He announced to everyone at the meeting that our small, independent team, which had never even seen this software before, was delivering for less than half the cost, in a third of the time.

This is an insider's irreverent guide to enterprise software. You'll never read this content in a Gartner Group report because there's no sugar coating to protect vendors and unscrupulous service providers. In it I will explain:

- Why enterprise software is so complex.
- The wrong place that everyone looks for answers, and where to go instead.
- The one question nobody asks, but should.

For over 20 years, I've been researching and selecting enterprise software for organizations like NASA, Boeing, and Toyota, and have worked as a system integrator for enterprise ecommerce software on several major brands. If you're lucky, you may not need it!

What do we mean by *enterprise software*?

Enterprise software can handle the needs of the world's largest and most complex businesses. This might include:

- thousands of transactions per second (Black Friday, major product launches, celebrity mentions)
- large or complex data
- highly complex business logic
- integration (data sharing and process triggers) with similarly complex systems

How do you write software that can do all this? You don't create a single program, like QuickBooks. You create a software **platform.** Something that can be configured and extended to meet your business needs within that specific domain (ecommerce, CRM, ERP, payroll, accounting, travel management, etc.).

Now, you don't have to be a multi-billion-dollar company to have complex needs. For ecommerce, it typically begins when you hit 8-figure revenue. Nor do you need to pay 6-7 figure license fees. There are many free, open source enterprise software packages.

SaaS or On Premise?

Let's talk about how the software is delivered.

Software as a Service (**SaaS**) includes vendors like Salesforce, Shopify, Mailchimp, Google Analytics, etc. They host it, and all their customers are on the same version. It's sometimes called "cloud based" because marketing thinks that's cool. There might be different tiers of prices/features, and they will likely do something extraordinary for you if you throw enough money at them. Maybe even give you an on-premise version.

On-Premise implies it's installed on your servers, but it doesn't have to be. Could be hosted by the vendor, integrator, or anywhere you like. The key is that each installation is a one-off that requires separate installation and maintenance. Almost all on premise enterprise software includes the source code for you to customize, so it can also be a unique snowflake (for better or worse).

OOTB, Configuration, and Customization

OOTB is out of the box. It refers to the features you get right after installation and minimal configuration. For instance, user management, and, um… well probably something else. You did just pay six figures for this.

Configuration requires no coding, just enabling features and modifying settings through the user interface. A product expert or business analyst (typically referred to as an administrator/admin in this case) can handle this; no need for a coder.

Note: Salesforce enables a ton of functionality via configuration, to the point where a Salesforce certified "developer" doesn't develop any software. In fact, they may never have coded in their life. Salesforce does have a real software development platform with a proprietary programming language called Apex. But why not use confusing, overloaded vernacular when you can? Welcome to enterprise software (and marketing).

Customization requires coding, which is typically the most expensive part. I frequently see **professional services** (what they call this work, along with the installation and configuration) running 1x-3x the license cost, which is often six figures.

Good enterprise software will have clear extension points (**APIs**) for customization. Most SaaS software has a **web service** API for integration. Web services enable your integration code to run anywhere, written in any language on any platform. Has your IT department standardized on .NET, but your new platform is in Java (or something totally unknown)? No problem.

Through the web services, you can download your orders, upload changes to your customer database, destroy valuable data, etc.

A couple SaaS platforms, like Salesforce and Demandware, have a proprietary programming language in addition to their web service. On premise software might have a web service interface, and/or a language API like Java or C#.

None of that is rocket surgery, but takes time to get the requirements right and cover all the edge cases. The problem comes when a core feature is implemented incorrectly for your business and there's no way to customize it through the API. You must change the core code (and maybe the database) to meet your requirements. I often suggest changing your process to match the software, if this is possible. But maybe you're doing it differently because this is a competitive advantage. Or you've already changed your process to accommodate another, more expensive piece of software. In that case, implementing this is only possible with on-premise software that includes the source code.

Basic Architecture

It's important to understand, at a high level at least, the different parts of the software.

- **Core code**: The creamy center. It has most of the functionality and underlying services. It includes the APIs to integrate with other software (say, ecommerce to ERP or CRM) and add/modify features (customization).
- **Admin interface**: The interface to configure the software.
- **User interface (UI)**: What your customers or employees see.
- **External integration code**: This uses the web service API, file upload, or other external integration method. This code may be stored and run completely separately.
- **Custom code**: What you write to add/modify features. It's typically stored somewhere it uses or overrides – not over**writes** – the existing code. *Typically.* It could include…
    - modifying the user interfaces mentioned
    - code using the Core APIs to add features and integrate with other software
    - changing (**overwriting**) the core code, if it's *really* required

As you'll see in a second, modifying the core code is a lot like going back in time. You fix one thing, like, you didn't total your classic car. But you somehow wiped out all the pandas. And we're at war with Eastasia. (Luckily, we've always been at war with Eastasia.)

Upgrades

Upgrades can be very tricky depending on the platform and level of customization. With most SaaS, upgrades should be automatic and you get all the **admin** UI features and core fixes and improvements. The rest will require some combination of new configuration, modifying the user interface, and adding code to access new features in the API. You also need to check if any APIs were deprecated, breaking software that relies on it. But usually, SaaS upgrades are safe.

With on premise, some of that will be the same. A new version means a new core. You drop it in, run regression tests, and you should get the fixes and performance improvements to the core code, plus any new admin screens.

In both cases, you may have modified the UI. That means your code is overriding theirs, so new features in the UI won't be exposed until you add them to your custom templates. With on premise, you may have modified the admin screens as well – same effect.

Finally, you may have modified the core code. This means you must merge your changes with the new core. Which means understanding all the new code they added, plus any changes to the existing core code. Improving existing code is called refactoring and is an industry best practice. Lucky for you, sort of, enterprise software companies rarely improve existing code. No matter, the whole process is a pain, and error prone. When finished, run a smoke test: are there still pandas?

Testing

In all cases, upgrades will be vastly aided by running your regression test suite. These are all the end to end tests created by your QA team that prove everything is working as it should, and they should be automated. However, when a project is running late, QA is the first thing to go. If you scrapped these tests during the initial rollout, you're going to have to factor them into the cost of the upgrade, along with everything else. Also, scrapping these tests means you have bugs you don't know about, which can be really embarrassing for you.

While I'm on testing, I'll point out that software quality insurance is a field that people devote their lives to. It's not something you can assign to interns and office admins. They will miss lots of bugs, which won't be caught until it's in production. By then, the warranty may be gone. The original developer may be on another project, or forgot what he did.

Bottom line:

- Have QA professionals test your software.
- Make sure they are writing automated tests.
- The earlier you catch a bug, the cheaper it is to fix.

I know of one development shop who stopped accepting clients who don't have their own QA professionals. Made Up Name takes on clients without even an IT department, but we'll insist on proper testing, by us or a third party.

Ask a Repair Shop

Some years ago, I had a broken GE washer. Pretty sure I knew the culprit, but if I was wrong I'd waste a bunch of money on a part I couldn't return. I called a local repair shop who serviced many brands, including mine. "Oh, we don't repair GE anymore. They're pretty much throwaways now. When they break, you just buy a new one."

This shocked me, beyond learning of GE's quality nosedive. If their products frequently broke, wouldn't that mean a steady stream of work? Then I realized a flaw in human reasoning. The first time it breaks, you blame the manufacturer. The second time, and every time after that, you blame the repair shop. For a repair shop, servicing appliances from a bad vendor can damage your reputation. Upon realizing this, I called him right back. "If GE is bad, who do you recommend?" I got a full education on washers, including a lot of industry dirty laundry.

Since then, every time I make an appliance purchase – vacuum cleaner, dishwasher, car – I go to a repair shop. They're always excited to talk to someone who will listen. They may even have a used model that will last you for years.

System integrators are a lot like repair shops. They aren't just connecting your new enterprise software to your other systems, but customizing the code to conform your business processes. Hopefully there are APIs to do everything you want, but they might have to modify the core code. The good ones are experienced, know the domain (ecommerce, ERP, CRM, etc.), and work with multiple vendors.
Their developers, at least, have opinions. Likely strong ones. I've been in this position in the past. I won't name names, but I've had the following conversation with developers from other integrators:

> "You guys are FooSoft partners, right? How is it?"
> "It's alright."
> "Really? Because we use BarSoft and it's a pile of garbage. It's got *rabble rabble rabble…*"
> "*Sigh.* FooSoft is terrible too, it has all the same problems."
> "I hear NewSoft is actually decent."
> "Yeah, I was working on it for a short while before getting pulled off to help a late project. It's pretty nice!"

This is how I determined who the top vendors were in terms of quality. The problem is, most companies don't buy software with this way. They cut out the system integrators entirely.

Typically, the CTO/CIO/CMO, etc. will first try to determine what they need. Come up with a list of features. Ballpark a budget. Then start talking with vendors directly.

These are smart people! Maybe they have a background in, say, networking engineering. Recently they spent a lot of money buying new network hardware and it was a big success. Then they did a desktop refresh, and that went well, too. So clearly, spending a lot of money on tech is something they do well, right?

Well, I'm highly technical, and I would not feel comfortable upgrading the network infrastructure of a medium-size enterprise without significant outside help. Yet many execs without software development experience (and possibly lacking domain expertise) go it alone. They get info straight from the vendor's formidable enterprise sales team, who knows exactly

what to say. I mean, have you seen a good enterprise sales team in action? They are amazing to watch.

This is like doing your appliance selection by talking to whoever is on the floor at Sears that day. In fact, it's worse, because at least the salesperson might know which models have the highest return rate. By going to the vendors, you have zero inside knowledge. Everything you hear is biased. It's more like visiting appliance manufacturer websites. We all know how helpful those are. And instead of risking hundreds of dollars, you're risking hundreds of thousands, if not millions.

Nothing is perfect. The system integrator may be biased toward specific vendors for a variety of reasons, like referral fees, or partnership building. Someone totally independent is ideal. But at least you're removing some of the bias.

What's the developer experience?

I started writing a comprehensive list of questions you want to ask when choosing software. But I don't have time for that, and it'd be misleading. You'd think you could use the guide and go on your own, and I don't think you can. Unless you have a software development background, bought that type of software before, and oversaw its implementation and use, you're not prepared. Hire a consultant. Doesn't have to be me, just someone who has done that before. What I will give you is the one question nobody asks, yet is so critical: what's the developer experience like?

As you just read, enterprise software is all about configuration and customization. If you've invested time and money to create competitive advantages in your processes, you don't throw them away to save a few engineering dollars.

Therefore, no matter the core feature set, there will be a bunch of custom programming to get you off the ground. Yet, somehow, I've never heard anyone ask if it will be pleasant for the developers. Maybe you don't care if the programmers like their job. Maybe you don't like your job! Misery loves company, right? Kidding aside, I assume neither is true and you care about the general happiness of your employees. But most of the time these are not your employees. They're at the system integrator or vendor, and you're paying them a lot to power through it without complaining.

However, there are solid business reasons to care about developer happiness.

Golden Handcuffs

Golden handcuffs are what we call the high salaries paid to good engineers so they don't leave. Much system integrator/agency work is repetitive, offers little growth, and uses frustrating technology. You pay a premium for good folks to power through, or you get not-so-good folks, which means you pay more either way. Sooner or later, the good ones leave. Maybe in the middle of your project. Programmers make lateral career moves, just to work on better tech, all the time.

Ill Communication

The experience could be bad because the vendor doesn't support system integrators well. Poor documentation, no access to the bug tracker, and firewalls between the integrators and the product team. When something goes wrong, it's on the integrator to prove it to the product team, which is a long and painful process when you can't talk to them directly. Many eye-rolling questions. ("Have you tried turning it on and off again?") And that time is billed to you!

A Pile of Garbage (Why Enterprise Vendors Fail)

Sadly, often the reason it's a bad developer experience is because it's bad software. This is a natural occurrence when your software company is driven by sales instead of engineering. And that's very common.

Sales will push for new features over fixing bugs. Marketing will mandate deadlines to meet their initiatives (trade shows, commercials, etc.), rather than how long it takes to do it right. Have you heard of the term **technical debt**? That's what you get when you rush developers instead of giving them time to do it right. Quick and dirty is expected occasionally, but developers need time to clean up the mess. In sales-driven organizations, that rarely happens. If the vendor spends a lot on sales, and that team is credited with driving revenue, they amass power. When that mindset grows to the point where engineering can't make a stand for quality, you're on a path to a pile of garbage.

Unfortunately, piles of garbage can have long shelf lives. Prohibitive sales costs keep out challengers, and customers rely too much on social proof. **Way** too much.

From a technical perspective, however, I have not seen a product recover from pile of garbage status. They become notorious among engineers, the butt of jokes at developer conferences. But they probably won't complain about it to you. Engineers aren't terribly confrontational, and know the value of discretion. Notice that I haven't called out any bad actors, and I know many. Engineers love the truth and want to be honest, but they're unlikely to openly bash products. The good ones simply find better work.

Get Help

You've seen the complexity of enterprise software and its pitfalls, and know you need professional guidance. Hopefully I've earned your trust and you'll consider using our services. But even if you don't, find someone who can help you:

- Gather requirements, with special attention to edge cases
- Determine budget based on projected revenue gains or cost savings
- Evaluate software packages and service providers
- Dig deep like a software Sherlock Holmes, finding out what nobody is telling you.

Contact us to learn more!