How to Implement Sitecore the Right Way, the First Time

ENGAGENCY.COM

Website | Schedule a Meeting

So you've chosen Sitecore to power your digital transformation. It's a big investment.

You want to make sure that it delivers on all your expectations—in terms of budget, functionality, deployment time, and more.

For that to happen, you need to make sure you've got the right people and processes to implement Sitecore the right way, the first time. When it comes to people—whether you plan to implement internally or work with an outside vendor—you need to make sure you're working with a team of real experts.

We've talked about how to properly identify a qualified Sitecore partner before, but today, we're focusing on the process. Read on to learn about the best practices you need to follow when implementing Sitecore.

1. Start with a Sitecore Architecture Document

Any Sitecore implementation, whether you're building a new site or enhancing your current one, should begin with a comprehensive Sitecore Architecture Document.

This essential piece of documentation outlines all the requirements for your implementation, allowing everyone involved in the implementation (including your content editors, marketers, IT team, UX agency, and your Sitecore vendor) to get on the same page—and avoid any painful surprises.

Your Sitecore Architecture Document should define how you want your content to be structured on the site, what modules you'll need to use, and who will need to use them and with what permissions.

A Sitecore Architecture Document is crucial to establishing clear communication and alignment on expectations, which ensures your partner is setting you up for an implementation that will deliver the value you're expecting. Along with your SOW, it's a key ingredient in the discovery and planning stage for any successful Sitecore implementation.

Keep in mind, some Sitecore partners like Engagency will provide Sitecore Architecture documentation as a stand-alone service, so if you're planning to implement internally, that's something to consider taking advantage of.

2. Follow Helix best practices

For many years, Sitecore offered no comprehensive guidance on how to best architect a Sitecore solution. That all changed in September of 2016 when Sitecore's Helix architecture conventions and guidelines were published.

Helix's technical design principles and development process recommendations contribute to a higher quality architecture, faster time to market, and a future-proof implementation.

3. Follow development best practices

In addition to Helix guidelines, which are specific to Sitecore, you'll want to ensure your Sitecore development team has a firm grasp and is fully committed to following the myriad other development best practices.

Among others, these include:

- Peer-reviewing all code
- Using well-defined version control and release strategies
- Using configuration transforms
- Deploying first to a staging environment before production

Finally, don't reinvent the wheel. Don't spend time creating your own version of something that comes built-in with Sitecore. You'll just be introducing more complexities that can cause performance issues down the line, when your time would be better spent using Sitecore's built-in modules to create the functionality you need.

4. Deploy on Azure PaaS

Beginning with the release of Sitecore 9.0 in October 2017, Sitecore began to move aggressively toward a microservices architecture. This complicated the topology of a Sitecore environment, so spinning up a physical or cloud environment by hand took significantly more time than it did previously.

Fortunately, Sitecore also spent quite a bit of time developing an Azure Marketplace module. With this module, launching an appropriately-sized Sitecore environment is as simple as answering a few questions and clicking "Go!"

The value of Azure PaaS goes beyond deployment speed. Azure PaaS also makes it easy for you to automate scaling for your Sitecore environment, so you can adjust your hosting needs (and your bill) to meet your site's traffic demands.

Azure PaaS also supports deployment slots for production-tier app services, allowing you to implement a blue/green deployment strategy. Your future feature rollouts will be seamless.

Note: If you're unable or unwilling to deploy on Azure PaaS for any reason, we recommend having at least one more production Content Delivery (CD) instance than you would in a cloud environment, to handle your traffic. This way, it's possible to take individual machines out of the load balancer rotation and deploy seamlessly. Plus, your website will continue to operate if one server unexpectedly requires maintenance during peak traffic.

5. Size your environment appropriately

Sizing your environment appropriately is important because you don't want to overspend, but you also don't want to cut corners and pay the price in terms of performance and reliability.

Make sure you have enough Content Delivery (CD) instances and make sure you're consulting Sitecore experts on how many you actually need. Also make sure all instances (including Content Delivery, Content Management, and xConnect) and all databases and database servers have appropriate resources, including RAM and CPUs.

Again, make sure you're talking with experts about what you can get away with. The right Sitecore partner will architect a server environment that strikes an appropriate balance between cost and performance.

Additionally, for physical and SaaS virtual deployments, be sure you've selected the correct edition of SQL Server. SQL Server Professional limits the number of CPUs and RAM that SQL Server can access. SQL Server Enterprise does not have this limitation. So websites with high traffic and high content scenarios may require SQL Server Enterprise.

6. Plan for multiple languages from the beginning

Implementing a site in multiple languages in Sitecore is easy—if you plan ahead. Sitecore has built-in functionality that covers the basics of setting up languages and adding different versions of content using the language versioning feature.

But, a truly multi-language site also encompasses the following:

- **URL Handling:** How URLs are handled can have serious implications for both SEO and automated language selection. It's important to think through how your URLs will be structured and whether each language is treated as its own site in the eyes of Google or as a single site with a primary language selected as the default. Options for URL handling include using different subdomains, fully different domains, language codes in the url string, or no indication in the url at all. This will also affect how you implement selecting a language automatically based on GeoIP.

- **Language Switching:** What mechanism will be used to switch between language versions and how is this presented to the user? It could be in the utility navigation, an invitation in a modal, or something else. Will the language selection be stored for future visits?

- **Automated Language Selection:** Is it prudent to use the GeoIP of a user to select the language for them? Will the user find it helpful or irritating? What are the implications for testing? Is there a way to enable the user to select a different language, even if it was pre-selected?

- **Non-Editable Text:** Don't forget all the text on a website that doesn't get edited by a content editor. From the prompt text in the search box to the text displayed on buttons, these must be defined somewhere so that they too can be translated. Dictionary Items, a standard feature of Sitecore, can be a huge help here.

- **Content Inclusion / Exclusion Per Language:** Do all pages or sections of a site appear in all languages? Does the navigation need to reflect differences between language versions? These decisions are best made upfront so the proper implementation and fallbacks can be put into place.

- **Search:** Search results likely need to be in each language. So, indexing multiple language versions is essential, and as with anything in Sitecore, that can be done in multiple ways for multiple reasons.

7. Implement a testing environment

Let's face it: licensing additional non-production Sitecore instances takes budget away from more noteworthy investments—but it's vitally important.

Here's why.

You need to test new features in a UAT or QA environment that you can access, before deploying them to your live site. Consider the cost to your organization if poorly-tested code is deployed to production and renders part, or all, of your web presence unusable. You certainly don't want that kind of notoriety.

8. Configure HTML and data caching appropriately

Appropriately-configured caching can have a big impact on how well your Sitecore environment performs. Make sure this essential step is on your development checklist.

Understanding the functionality of each page and module plays a critical role in assessing the right kind of caching to apply. While the steps to apply caching are easy, if the wrong settings are applied or the cache is not cleared at the right time, some very serious production issues can happen. Incorrectly applying the settings can cause the wrong content to show to users, completely breaking the functionality of your website. It's critical to thoroughly understand these settings and get them right before going live with caching.

9. Use automated deployments

Manual deployments are a common source of problems. While the code itself might be fine, mistakes and inconsistencies in the deployment process can become problems in and of themselves.

There are a number of technologies that can be used to automate deployments, eliminating the human element from critical steps in the process. Automated deployments can also make it easier to reliably roll back to a known good state if necessary. This all means less risk that your website will become unusable or unstable due to deployment issues.

An experienced Sitecore partner will recommend technologies and processes that will build and deploy your solution in a consistent manner. While different environments may require different technologies, a few key objectives will be the same. The output should be consistent each time, it should require minimal human intervention, and it should be as safe as possible. A well-designed and implemented automated deployment will help achieve those objectives and will avoid many of the pitfalls of manual deployments.

Getting your Sitecore implementation right the first time

As you can see, implementing Sitecore correctly is a complex process. The best way to get it done right is with experts leading the way.

At Engagency, our team of Sitecore Certified Architects, Developers, and Support Engineers have over a decade of experience implementing and supporting Sitecore. Whether you want an ultra-fast Lift-and-Shift or a completely custom solution, our Sitecore Implementation Services team can get the job done on time, on budget, and the right way the first time.

Let's schedule a time to talk now.