WHITE PAPER

Five Reasons Why You Should Use a Git-Based CMS

CRAFTER SOFTWARE

Table of Contents

Introduction

Traditional content management systems (CMS) are still based on architecture and technology that was introduced more than 20 years ago. To this day, the typical CMS vendor sees the primary goal of the CMS as directly enabling the business user, typically marketing. Support for developers and operations was and is a second-class concern. The whole concept of DevOps wasn't even around when most CMS platforms were built. Today's digital economy runs on the paradigm of Continuous Integration and Continuous Delivery (CI/CD). Where is the support for CI/CD of digital experience apps — content and code? It has been non-existent.

In the last 10 years the CMS industry almost completely ignored these concerns in favor of support of marketing enablement technology — mainly by bolting on various "martech" tools to the CMS. Times have passed this approach by. Today companies need to create and deploy innovative digital experiences - multi-channel and multifeature websites, mobile apps, virtual reality, mixed reality, digital signage, personalized e-commerce front-ends, IoT, and more — as often and as frequently as content authors can dream them up. Organizations are finding their legacy CMS is now the bottleneck. Content authors need to freeze content while IT tests and/or deploys new code, and new feature development takes forever. The CMS not only fails to aid in the creation, deployment, and scale out of new features, it fights it at every step of the way.

Next generation CMS platforms must serve authors, developers and operations as equal first class citizens allowing them to work both independently and collaboratively to create, manage and deploy any digital experiences with agility, speed and flexibility with ease at any time any where.

One of the primary issues with traditional CMS platforms can be found at the repository level. These platforms use centralized data and content storage services that not only limit, but also dictate the overall capabilities of the repository — and by extension the CMS. Meanwhile, modern software development has moved to

Git for source code management for a variety of reasons, including the massive productivity improvements gained from decentralization and distributed versioning and workflow. Some had even started to use Git to store HTML and other artifacts of static websites. So we asked the question: What if Git could be used to store not only code, but also all content, for dynamic digital experience applications? We answered the question through several years of R&D and found that it solves the major problems of traditional CMS solutions outlined above.

Crafter CMS is a next generation content management platform based on Git that replaces the broken paradigm of traditional content management systems, and ushers in a new era of fast, agile and easier development of innovative digital experiences - serving the needs for developers, content editors, and operations. In this White Paper, we'll outline the five main reasons you should use a Git-based CMS like Crafter.

1. Event-Based, Multi-Object Versioning

Traditional CMS platforms like Drupal, Wordpress, Adobe Experience Manager, Sitecore and most others either have very limited versioning, or provide basic versioning capabilities that track single object graphs, or maintain clunky data structures to track relationships.

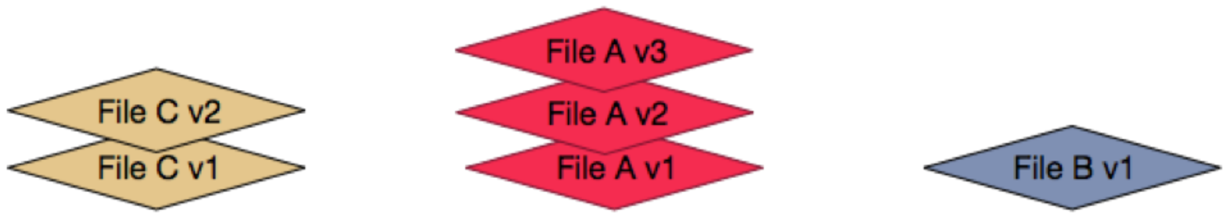


Figure 1: Single file versioning model. Each object has its own version tree. How and whether relationships are tracked between objects differ from one system to the next.

Such simplistic approaches work for basic content management needs like blogs or boring websites but largely fall down in the face of managing today's multi-object, multi-asset digital experiences. Today's content models are component-based, and they have many relationships and dependencies. Further, there is often a relationship between the content and the code (CSS, Javascript, templates, etc.) that needs to be considered. Tracking the edits of any one specific object in isolation is simply not enough.

While simple object versioning supports basic editing, simple review and basic reversion, these use cases are only the tip of the iceberg in a real-world environment. Scenarios like legal audits, company re-brandings and concurrent feature development drive the need for much more sophisticated CMS capabilities like a "time-machine" preview, multi-object reversion and content/ code base branching.

Instead of the single file versioning we see in the CMS space, what's needed is a multi-object versioning approach like we see in the programming space. We require an approach that tracks "the entire state of the universe on each change." With this level of version detail, a system can provide real previews at any point in time, make intelligent decisions about what must be reverted and support a host of branching and workflow needs.

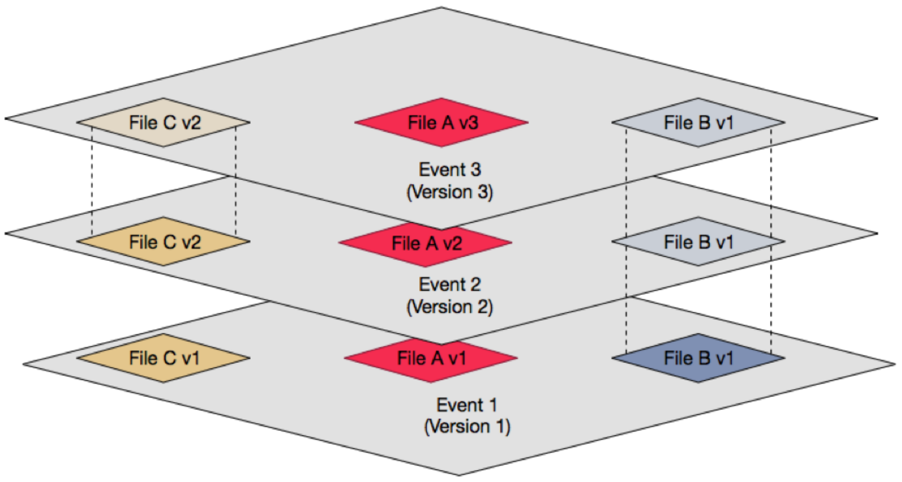


Figure 2: Multi-object ("striped") versioning model. Each event tracks the state of the entire repository at the time of the event.

This type of solution already exists in the enterprise software development space. With software, one source file is often related to many others. Versions between objects matter. Modern Source Code Management (SCM) has evolved to support this need. Git is today's most popular and widely used source code management system. It's clear that the content and technical components of today's digital experiences share many of the same needs that we see in the software development space. Rather than re-invent Git to achieve the same versioning capabilities in the context of content management, we've based Crafter CMS on Git's versioning mechanics.

Because Crafter CMS is based on Git, every content change event is tracked with an event ID known as a “Commit ID.” Using this ID, it is possible to know the state of every content object in the system at the time of the event. For the sake of simplicity we can say that we’ve created a version “stripe” across the entire repository at a given moment in time. The system does not make a copy of every object on every edit. That would be too slow and cost too much in terms of storage. Instead, this is done in an efficient and effective manner by leveraging Git’s own proven versioning mechanics.

Moreover, because of the way Git stores and manages versions, traversals to any point in time are extremely fast. Performance is very important for the types of use cases we discussed earlier. Let’s take for example, an auditing scenario: legal needs to see what the site looked like 46 days, 2 hours and 42 minutes ago. With most CMS platforms, this scenario is impossible to support. At best a systems group can attempt to restore a backup from that date and staff can be diverted to give the lawyers what they need. Even if your CMS claims to support this kind of review, the speed at which it can be provided is of key importance. If it’s too slow it won’t be practical. I’ve seen demos of CMS platforms that take minutes to render a previous version of a dynamic site. That’s too slow when you are doing a triage. It’s worse if you are traversing for editorial reasons. Crafter CMS simply doesn’t have this issue. Because of the way Git stores versions, traversal of version in our Git-based CMS is extremely fast.

Finally, Crafter’s Git-based versioning approach itself hints at another important and related characteristic of Crafter CMS: content is managed in a document-oriented, file based store. In short, content is stored as XML. Git is a file based versioning system. Storing content as files is not only necessary, but the filebased approach has several major advantages. Because we’re dealing with files, content is easy to move among environments (Dev, QA, Prod, etc.) and migrate between systems. It’s much easier to integrate the content with other 3rd party systems, such as for language translation, e-commerce and marketing automation. And because we store content in an XML format, it’s multi-byte character set friendly and totally extensible.

2. Distributed Repository

Most databases are not easily distributable from a geographic sense, and more importantly, they are not distributable from a versioning and workflow sense.

We could spend a lot of time talking about how scaling and distributing a database geographically matters in the context of CMS and why it's so difficult. If you have the need for a CMS with high availability and global distribution you already know why it matters. If you have tried to make this work with a CMS based on a traditional database or a JCR repository, you already know it's a difficult and sometimes impossible errand.

What is distributed versioning and workflow? The easiest way to get at this is by example. In the software development space we've had Source Code Management (SCM) systems for a long time. These SCM systems allow teams of developers to work on a single code base as a team without stepping on each others toes by checking out work locally, working on it and then checking back in edits. Hint: This is not much different from what a CMS provides to content authors behind its UI.

Back to developers: In the past we had CVS, SVN along with many others. These SCM systems provided basic version management as well as branching and tagging but fundamentally the system was a centralized model. With such solutions, there is a single central store and source of truth for the code base.

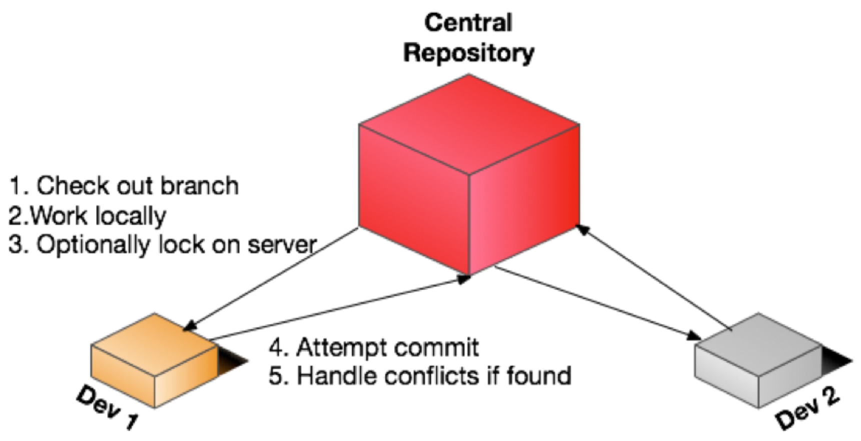


Figure 3: Centralized source code management. Fails for large scale development.

This SCM model worked well for smaller teams and smaller code bases but for large projects like the Linux operating system it failed completely. Linux has so many developers spread out all over the world, working on many separate but related projects. A single, centralized system simply does not scale (in several ways) to meet this need. To make a long story short (collapsing a lot of history and detail), Linus Torvalds created Git as a lightning fast, open source solution to solve this problem. Git allows developers to have their own local and intermediary repositories that are all born from a parent repository. This makes distributing developers easy, it makes concurrency simple and most importantly to us, it distributes the versioning and workflow which makes "flowing" code to and from these independent repositories possible, fast and easy. Yes!

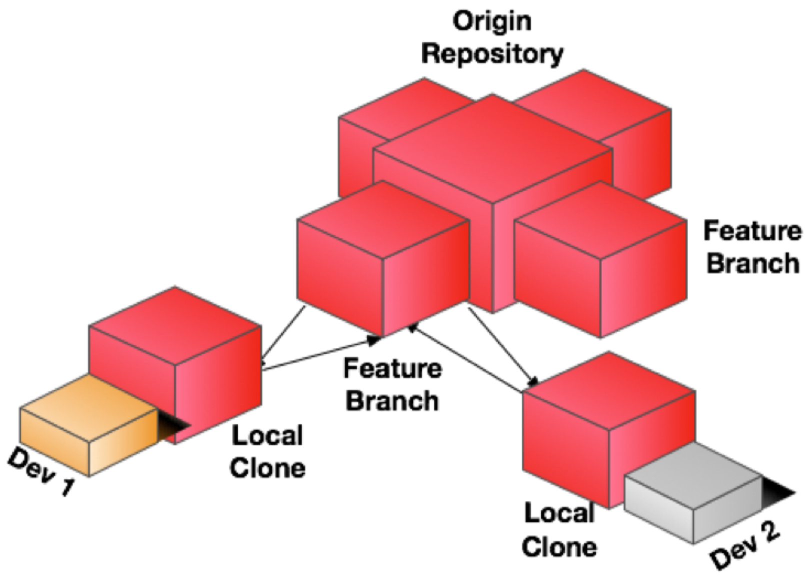


Figure 4: Decentralized source code management.

In the CMS space, for more than 20 years all the way up to this day, we've had repository solutions of various capabilities and quality. All of these solutions have no real, workable solutions for moving content back from production to lower environments like Staging, QA, Development, Load Testing and local developer machines. Yes, you can do it. But it's a nightmare. You end up doing an export/ import process and it's difficult. Some systems are easier than others but they all stink. CMS consumers rig up all kinds of replication and publishing work-arounds to try and deal with this problem. It's all a hack. There's no technical solution in the

CMS space that was built to handle the problem specifically. For this reason and many others, development and operations teams HATE the CMS options available today. They do nothing to help the team work — worse, they fight them in almost every way. The technical members of the team put up with CMS technology because their business counterparts need content creation and editing capabilities. That's all.

Today we understand that to some degree, in the digital experience space, "code is content." Just as weneed to be able to move content back to environments, we also must be able to move code (templates, javascript, CSS, etc.) forward through the environments. Developers have a process that they use to ensure quality and performance. With traditional CMS, moving code forward through environments is even harder than moving content back. Wholesale export/import doesn't work!

Because Crafter CMS is Git-based and because we've specifically built capability in Crafter CMS to handle these needs, the world finally has a CMS that solves this problem. The same approach developers use to make and promote source code changes with Git is used by Crafter CMS to move code forward and content back.

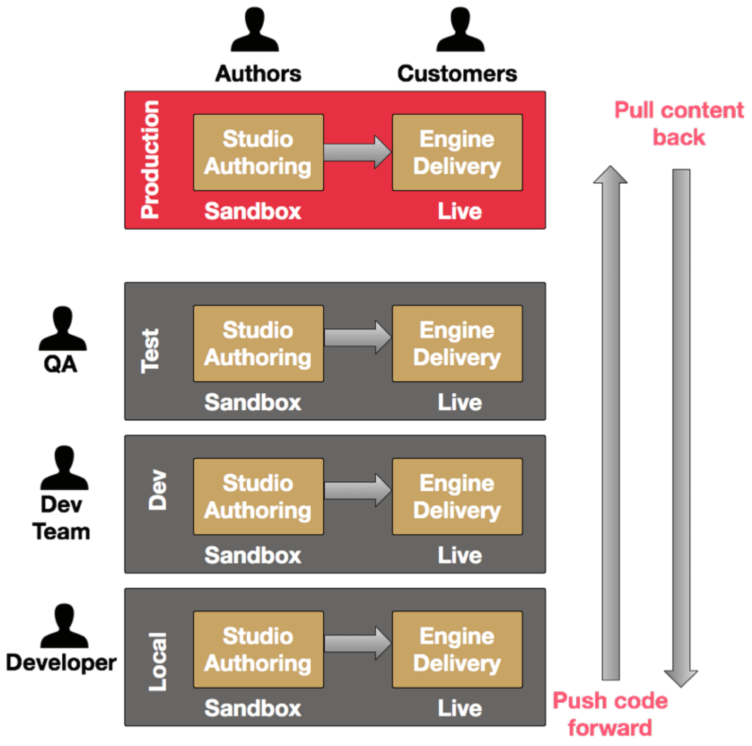


Figure 5: Extending the concepts of Git to content management

Every organization that uses a CMS for more than simple edits and blog posts can readily see the benefits here. Today, it's understood that customer experience is one of the biggest competitive advantages an organization can have. Further, beyond the human element, digital enablement and innovation is the most important component of delivering great customer experience. Because content and code are inseparable from customer experience, the CMS is a mission critical component of any and all customer experience solutions. Here's the kicker: nearly the entire world is using a CMS technology that not only fails to enable the organization to innovate faster — it actually fights them!

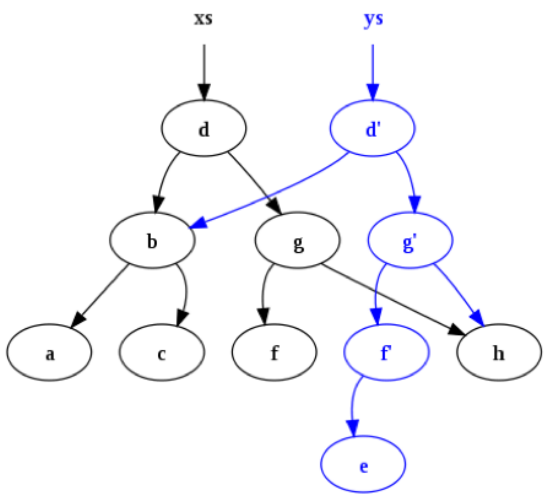
The Git-based distributed capabilities in Crafter CMS allow your organization to have many environments that are all related to one another -- syncing and moving objects between them is natural and part and parcel to the technology itself. This means it's easy to move content back and code forward.

Because the system is distributed and Git-based, developers can work locally and still be part of the CMS. That means they can use the tools they know and like, and they are not working on an island. The best way to make a developer love the CMS is to let them work with the CMS without having to work in the CMS. Organizations that want to win, need to innovate without impedance.

3. Distributed, Scalable, Consistent Publishing

Crafter CMS uses Git mechanics to publish content to its decoupled delivery space. When Git reports that its repository is set at a specific version that means that every file is guaranteed to be present and in the proper state for that version, it is. Fact. It's provable.

The reason it's provable is due to the fact that the Git mechanics that underlie Crafter CMS' content repository are based on Git's purely functional data structures. "The main difference between an arbitrary data structure and a purely functional one is that the latter is (strongly) immutable" (Wikipedia). What this means is that as commits happen within the repository an entirely new immutable data structure is created containing the changes for the commit. No action is taking on the previous data structure(s.) Nothing you ever change can be lost or corrupted by an operation once the change has been committed. Moreover, in Git, the ID for the commit is essentially a SHA1-hash of metadata and the content in the directory tree. By definition, if a single bit changes anywhere in the tree a new SHA1-hash must be generated.



While this explanation is an oversimplification of Git's algorithm, it is essentially the model of how it works. The point is that two repositories on two different machines with the same commit ID are mathematically guaranteed to be the same. That's an extremely useful mechanism for versioning but it also is a very large helping hand in publishing. Crafter CMS publishes (replicates) content based on Git commits. If you want to know if an endpoint on the other side of the world is the same as what you expect, you only have to compare the commit ID(s).

In today's elastically scalable, globally distributed world you can have any number of servers. You need a means to make sure they are all in sync. As you can see above, Git's internal mechanics give us just that. Crafter CMS is the first decoupled CMS with the capacity to scale geographically across an elastic cloud and at the same time make 100% certain that remote instances are consistently running the same version of content and code.

4. Branching

Nearly every CMS still sits on an architecture and repository that was devised 20 years ago. Back then innovation was important but the world moved a lot slower. A single pipeline of features got the job done and the CMS was less of a bottleneck. Today, in contrast, digital channels are at the core of many organization's strategies for serving the customer better and beating the competition. The organization who moves the fastest often wins. Agility is key.

Meanwhile, infrastructure costs have gone down while development costs continue to rise. Balancing costs with volume and speed of innovation is a major challenge of our day. Today it's all about great DevOps. To make DevOps really work with CMS you need to be able to branch.

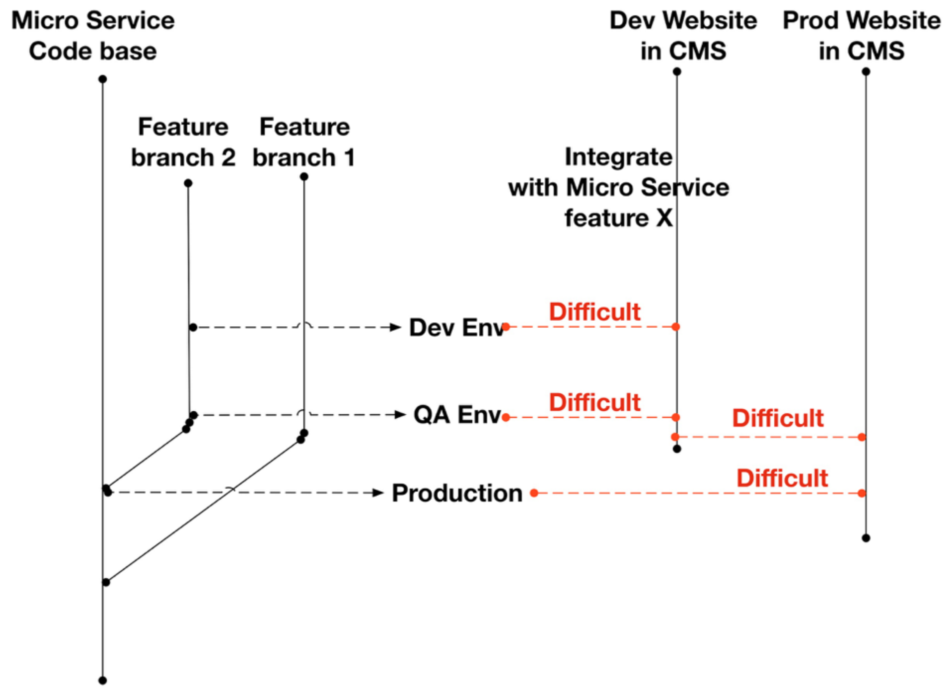


Figure 6: Traditional CMS development is difficult

Traditional software development process has included branching for eons. Developers and DevOps have long since figured out that they need to be able to work in teams, isolate work and control the order in which work is merged into the critical path for go-live. The CMS track runs right alongside the traditional development track, and at some point it merges and the last few steps require the CMS. It's only now that the demand for the speed of innovation has increased that the connective tissues between development and the CMS have been put under so much stress that they are completely failing. The fact is that today, we need that same agility all the way through the CMS and right up to the very last step of production deployment. Even if the majority of your development is outside the CMS you still need to integrate.

Because the website needs to integrate with, and ultimately deliver the functionality of the microservice, we need to perform development. We also need to support daily content edits and continuous publishing. With traditional CMS we have no option but to use multiple CMS environments to support this scenario. Does this approach give us multiple tracks and control over my releases? Yes, technically it does. But practically speaking? No. Not at all.

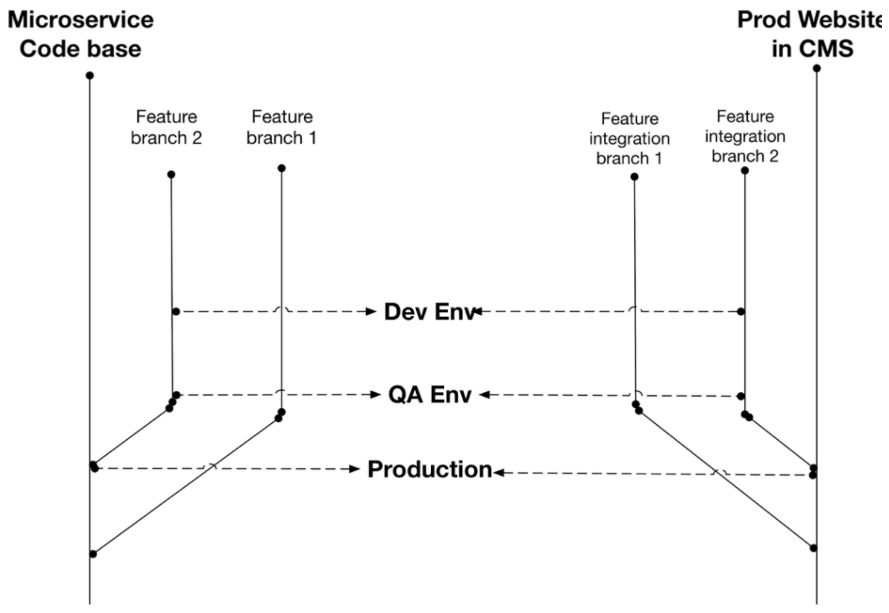


Figure 7: Branching mechanics of Git streamline

As we learned before, moving content and code between CMS environments with traditional CMS architecture is extremely difficult. The process of spinning up environments and loading code and content into them is so difficult and time consuming for any DevOps team that most won't even consider it unless absolutely forced to. Even then the size of the team may not support the need. The rate of innovation crawls to a near stop.

To address this problem, we built Crafter CMS v3 on top of a Git-based repository. As a result, the Crafter CMS platform is built on a repository store that not only branches, but also is fully distributed. Not only are you able to easily control the order and rate of work, but it's a snap to move work from one environment to another.

Moreover, branching not only supports DevOps and but it makes development easier. Developers and authors can experiment, work on major features and other site enhancements in the safety of branch based sandboxes that keep them from stepping on each other's toes.

5. Familiarity

Throughout this White Paper, we've talked about a "Git-based" CMS. This is intentional and it's vitally important. We aren't just talking about a better mousetrap. We've tried to take a hard look at the kind of problems that remain in the CMS space and the needs that modern organizations have in terms of innovation at a competitive level with respect to ease, speed and scale of continuous delivery. It turns out that, yeah, some of the biggest bottlenecks go all the way down to the core of today's CMS platform: They way we store and manage content and code.

The features we are talking about are the first steps in truly making a real and major difference. What we're saying is: we need a repository with a specification that enables these kinds of capabilities. Developers and DevOps should not have to dance around the technology to get their jobs done. That said, the functional specifications of features and how you implement them are different things. We could have focused on a system with "Git-like" rather than "Git-based" functionality. We chose to do the latter. We did that on purpose.

Leveraging Git rather than re-inventing it has nothing to do with code-reuse or ease of implementation. If you look around, you will find a number of projects that enable non-technical content editing and publishing (basic CMS functionality) for statically generated static websites that leverage Git. Developers and DevOps who are familiar with Git, already recognize many of the benefits of leveraging Git for these kinds of needs. However, implementing a full enterprise-class CMS designed to support modern, dynamic and personalized multi-channel experiences isn't as simple as sticking Git under your CMS. The full range of content authoring, development and DevOps use cases are much different and more complex than what you see with simple editing and publishing of statically generated websites and standard source code management. There are a lot of decisions to make and problems to solve.

A Git-based implementation is important for a much bigger reason another reason: Familiarity matters. Familiarity covers both adoption and integration. Git is used all over the world. It's proven. Developers and operations teams know how to use it. A Git based CMS is part of the broader Git ecosystem -- which means your existing Git-friendly toolchain natively works with the CMS.

"Git-like" is not enough. It's got to be Git-based. Familiarity matters

Conclusion

Today's CMS systems are rooted in 20 year old architectures and technologies. As the demand for greater amounts of innovation and digital experience has grown and organizations are under pressure to deliver more at ever increasing rates CMS platforms have become more of a hindrance than a help. Crafter CMS, with its Git-based approach, not only solves these fundamental problems but also integrates well with existing developer processes and tools — enabling faster innovation. Finally, we have a CMS approach that accelerates development instead of blocking it.

Getting Started

LOOKING TO LEARN MORE?

- ◊ Video: **Introducing Crafter CMS 3.1**
- ◊ Download Crafter CMS: **Download**
- ◊ Launch an AWS Instance of Crafter CMS: **Launch AMI**
- ◊ Request a Trial: **Contact us**

ABOUT CRAFTER SOFTWARE

Crafter Software is on a mission to replace the broken paradigm of traditional content management, and to usher in a new era of fast, agile and easier development of innovative digital experiences. Our flagship product, Crafter CMS, is amazing for developers, easy for content authors, and fantastic for DevOps. We build our software solutions on the foundation of open source, transparency, robust architecture, high performance, superior quality and outstanding customer support. Available on premise in the enterprise or SaaS in the cloud.

Learn more at www.craftersoftware.com and craftercms.org.

LEARN MORE

CRAFTER SOFTWARE

www.craftersoftware.com

info@craftersoftware.com

+1.703.234.7744