Factors to be Considered in Evaluating Version Control Options

Flosum's Native Version Control vs. Open Source Tools: How it Stacks Up

Flosum

CONTENTS

Executive Summary

At the heart of every DevOps solution is the Version Control System, which works to maintain a central source of truth for all code versions. However, many other functions fall under the Version Control umbrella, and the technology selected to manage these processes can transform application development. This document examines Version Control in its entirety, and how it's specific functions relate to the Salesforce platform. Additionally, it provides a side-by-side comparison between Git-based and native Version Control tools.

It cannot be emphasized enough, that selecting the correct Version Control tool is a critical component of any Development Operations Department's success. If you are currently evaluating competing solutions or utilizing Git-based Version Control, this document will help you to evaluate the differences and ultimately make an informed decision. With many options on the market, understanding these differences is more important than ever, and selecting the most optimal tool becomes a critical part of any strategic DevOps plan.

Considerations

- Git and Salesforce are both powerful pieces of technology, however they were created for very different purposes and as such the functionality between the two tools is often misaligned.
- Salesforce has created a unique culture largely comprised of Admins and Citizen Developers, many of whom do not hold computer science degrees. Git on the other hand was built to be used by professional developers with extensive background knowledge. This often impedes the use of the software for a large portion of the Salesforce development community.
- Native, purpose-built Version Control tools are congruous to Salesforce, providing functionality that reflects an intimate understanding and tight alignment of the force.com platform and its unique demands.
- When evaluating Version Control tools it becomes essential to not only consider the features and functionality that are presented at face value, but to also further evaluate their execution. Taking time to bench-mark the tools on their performance becomes an important step in the selection process. For example, consider the time that must be spent moving changes from development into production for each tool. Deep analysis and direct comparison of each feature will ensure satisfaction and success when the final selection is made and the tool is adopted.
- While Git is a widely used Version Control tool, special consideration should be taken when evaluating its fit for Salesforce development because it was not purpose built for the force.com platform. Native Solutions are Salesforce specific and enterprise-ready leading to quicker adoption rates and improved performance.

32%

Business value generated by saving developer time in branch management

29%

Increased productivity

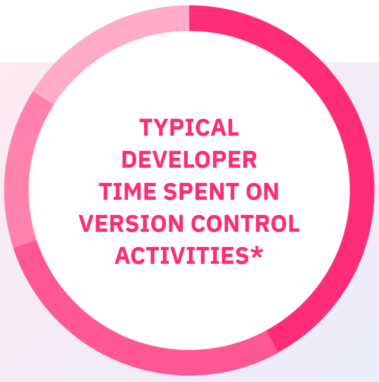
41%

Technology and consulting costs saved

Key Functions of a Version Control System

We conducted an in-depth study of Development Operations in which we took a close look at Version Control as a whole. Our research identified the six most significant functions that make up complete Version Control performance including conflict management, isolation of work, synchronization of work, collaboration, visibility/team capabilities, code review/approval process, as well as accountability/compliance.

Our research also allowed us to develop a clear picture of which Version Control functionalities require significant attention from developers. The pie chart below illustrates our findings, pointing out that the most substantial amount of time is spent merging changes and completing pull requests and code reviews.



- ● 42% Merging Changes
- ● 28% Pull Requests and Code Reviews
- ● 14% Branch Management Functions
- ● 16% Other Version Control Activities

* Based on a study of 127 developers over six-month period

How is the Salesforce Platform Unique?

The Force.com platform is a rich development environment designed to enable developers to rapidly create and deploy trusted cloud applications. Its unique architecture is designed to enable swift innovation, deployment, and adoption. However, because of this - it also differs greatly from other development platforms in the following ways:

- Development does not occur in an Integrated Development Environment, but rather in a Salesforce Environment called a Sandbox. The Sandbox is a copy of the production organization and multiple copies can be created in separate environments for different purposes such as development, testing and training, without compromising the data and applications in the production organization. And because every sandbox contains the entire production codebase it is a complete repository in and of itself.
- The Force.com Platform is the only platform that allows development teams to build powerful enterprise applications without writing a line of code. In fact, almost seventy percent of the code is written in a declarative or XML based fashion rather than traditional programmatic language.
- Since applications can be written without writing complex code, unique roles to Salesforce have emerged including Salesforce Admins and Citizen Developers. These individuals are business users, nearly 60% of whom do not hold Computer Science degrees but take part in development activities. Many are in charge of making certain code changes which can lead to a need for increased compliance as well as a robust toolset that does not require extensive background knowledge.
- Most application development platforms are hosted in-house, and all the data is safely held within the corporate network. However, the Force.com platform is cloud-based, making security one of the most important factors to be considered when building new applications.

Barriers that exist when utilizing Salesforce with Git-based Version Control Systems and the Argument for Native Version Control

Traditionally, open source tools such as Git, have been widely adopted for Version Control needs. However, because Salesforce differs greatly from traditional development platforms (as illustrated above), the application of these existing Version Control technologies is limited and becomes increasingly difficult for developers to utilize in the Salesforce environment.

Therefore, a business case is made for emerging technologies to step in and solve these long-held nuances, creating solutions that enable developers to utilize Version Control in the Salesforce environment. And a significant advantage is afforded by Native Version Control Systems that are built directly within the force.com platform designed specifically to handle its unique needs.

459% ROI	**6 months** Maximum payback	**$4.8MM** NPV	**$5.8MM** Benefits PV

Side-by-Side Comparison of Native Version Control vs. Git-based Version Control

Below we examine the following areas: merging capabilities, code review, branch management, quality gates, ease of use and security, in order to provide a holistic view of Native Version Control vs. Git-based Version Control. The native version control tool used in the comparisons is Flosum, a comprehensive DevOps Tool built entirely on the Salesforce platform, specifically designed to handle the unique needs of the Force.com platform. As you will see below, these two pieces of technology both provide a similar foundation for version control, but vary greatly in the complexity of use and time required to execute basic Version Control functions.

Merging capabilities

Salesforce developers using Git-based Version Control systems spend nearly half of their time merging changes. The reason this function is so time-consuming is that Git-based Version Control systems weren't originally intended to handle certain Salesforce specific components (i.e. lightning components, aura definition bundles, static resources) and are only capable of direct line-by-line comparison.

> Salesforce developers using Git-based Version Control systems spend 40% of their time merging changes.

Alternatively, because Native Version Control was built specifically for Salesforce, it includes algorithms for all Salesforce component types and utilizes logic to place changes in order, automatically alleviating a significant amount of developer burden in resolving conflicts.

The following examples illustrate this concept:

Git Based Version Control

Developer 1		Developer 2
<list_of_countries>		<list_of_countries>
<Argentina>		<Mexico>
<Uruguay>	→	<Argentina>
<Brazil>		<Uruguay>
<Mexico>		<Brazil>
<list_of_countries>		<list_of_countries>

Conflicts: 4

Flosum Version Control

Developer 1		Developer 2
<list_of_countries>		<list_of_countries>
<Argentina>		<Argentina>
<Uruguay>	→	<Uruguay>
<Brazil>		<Brazil>
<Mexico>		<Mexico>
<list_of_countries>		<list_of_countries>

Conflicts: 0

The chart above depicts two developers who are merging their versions. As you can see, the developer on the left has the same countries listed as the developer on the right, but they are out of order. Because Git can only do line-by-line comparison, it will show four conflicts that need to be resolved. The developer will then have to manually go through these lines and eventually determine that these conflicts were false.

Flosum on the other hand, uses powerful algorithms that are able re-order these changes in the same sequence and in turn will show 0 conflicts, saving the developer from having to resolve any conflicts at all.

In this next example below, the second developer simply removed Argentina, but because Git can only compare line-by-line, it will again show four conflicts. Flosum will determine that the rest of the countries are the same and only show one.

Git Based Version Control

Developer 1		Developer 2
<list_of_countries>		<list_of_countries>
<Argentina>		<Uruguay>
<Uruguay>	→	<Mexico>
<Brazil>		<Brazil>
<Mexico>		
<list_of_countries>		<list_of_countries>

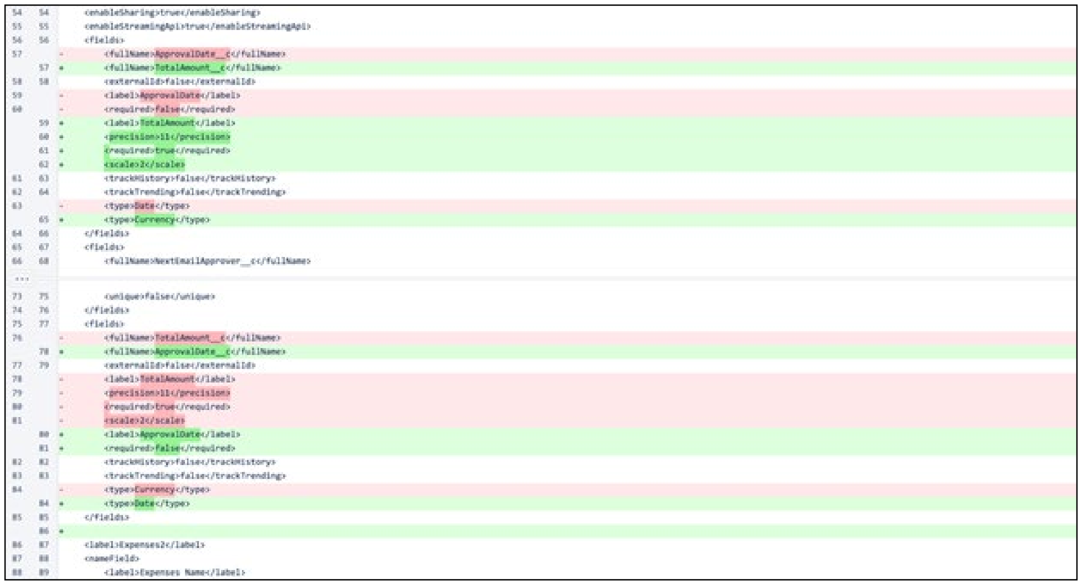
Conflicts: 3

Flosum Version Control

Developer 1		Developer 2
<list_of_countries>		<list_of_countries>
<Argentina>		
<Uruguay>	→	<Uruguay>
<Brazil>		<Brazil>
<Mexico>		<Mexico>
<list_of_countries>		<list_of_countries>

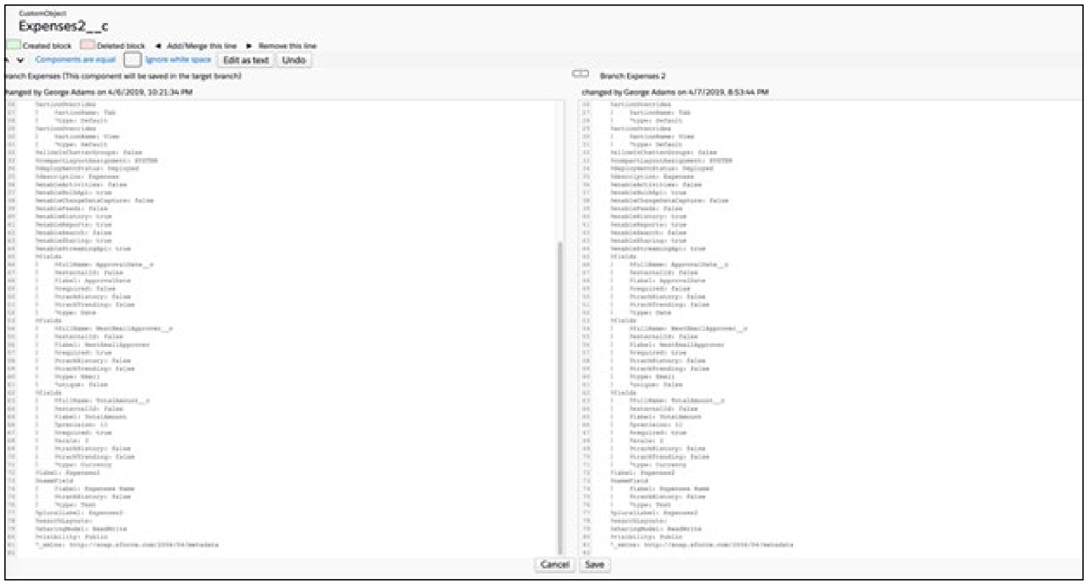
Conflicts: 1

Below we have provided a screenshot of a 100-line component commit in Git. As you can see, 21 conflicts have been detected (pink depicts code that has been removed, green depicts code that has been added). A typical admin may take 5-10 minutes to resolve all of these changes.



```
    <enableSharing>true</enableSharing>
    <enableStreamingApi>true</enableStreamingApi>
    <fields>
-       <fullName>ApprovalDate__c</fullName>
+       <fullName>TotalAmount__c</fullName>
        <externalId>false</externalId>
-       <label>ApprovalDate</label>
-       <required>false</required>
+       <label>TotalAmount</label>
+       <precision>11</precision>
+       <required>true</required>
+       <scale>2</scale>
        <trackHistory>false</trackHistory>
        <trackTrending>false</trackTrending>
-       <type>Date</type>
+       <type>Currency</type>
    </fields>
    <fields>
        <fullName>NextEmailApprover__c</fullName>
...
        <unique>false</unique>
    </fields>
    <fields>
-       <fullName>TotalAmount__c</fullName>
+       <fullName>ApprovalDate__c</fullName>
        <externalId>false</externalId>
-       <label>TotalAmount</label>
-       <precision>11</precision>
-       <required>true</required>
-       <scale>2</scale>
+       <label>ApprovalDate</label>
+       <required>false</required>
        <trackHistory>false</trackHistory>
        <trackTrending>false</trackTrending>
-       <type>Currency</type>
+       <type>Date</type>
    </fields>
+
    <label>Expenses2</label>
    <nameField>
        <label>Expenses Name</label>
```

Here is that same component merge in Flosum. Because the code is merged in a declarative fashion using XML technology, an orderly comparison has been achieved and Flosum was able to decipher that none of these changes were actually conflicts, saving the developer 10 minutes of time.



ranch Expenses (This component will be saved in the target branch)
hanged by George Adams on 4/6/2019, 10:21:34 PM

[illegible]

Branch Expenses 2
changed by George Adams on 4/7/2019, 8:53:44 PM

[illegible]

This was an example of a very small 100-line component. Some components in Salesforce can be tens of thousands of lines long. If Flosum was able to save the admin 10 minutes of time resolving this small conflict, it can be deduced that the time saved on large components would be significant.

If a typical developer is resolving a ten-thousand-line component, Flosum would save that developer 1,000 minutes that would have been spent resolving conflicts in Git. That is SIXTEEN HOURS of working time saved, just on one component!

Code Review

Figure 1: Factors to be considered in Git-based vs. Flosum Native Version Contol System in Code Review

	Git Version control system	Flosum's Native Repository
User Story Code Review	Git-based Version Control systems can only do a code review at the feature branch level	Flosum allows for code review at the User Story level with integration from Jira, Agile Accelerator, ServiceNow, AzureDevOps, etc. which provides a holistic picture of the user story vs. only seeing the code at a commit level.
Code Coverage	No such functionality in Git	Flosum allows the reviewer to ensure there is no drop in code coverage before the code review is completed. This ensures test classes are written along with main apex classes.
Line-by-line commenting	Available in Git	Available directly within the Salesforce UI
Sandbox and Org Comparison	Not feasible	Flosum allows users to easily compare the user story against any other org and easily merge the changes.
Approval	As determined by your Git Vendor	Flexible, customizable process using the Salesforce implementation.
Review of Declarative Differences	Comparison is done on a line-by-line basis	Comparison is done using XML formatting
Review of Lightning Components	Only top differences are shown	The entire bundle is unpackaged so that each file can be reviewed completely, and all the differences can be examined.
Dependencies	Git does not show which dependent components are missing.	Flosum shows missing dependent components.

Branch Management

Figure 1: Branch Management factors to be considered in Git-based vs. Flosum Native Version Control

	Git-Based Version Control Systems	Flosum's Native Version Control
Synching Sandboxes With a Target Org	If an admin or developer makes even a simple change to one of the orgs it is extremely difficult to decipher what change has been made and synch with that respective branch.	Flosum offers bi-directional, real-time synchronization between any branch in Flosum to any other Salesforce org, which ensures code-overwrites will never happen.
Destructive changes	Git tracks when changes are added, but it is very difficult to capture which components have been removed from a user story.	In Flosum, feature branches carry the complete information for destructive changes and can very easily visually map the changes for users.
Mapping with requirements management	With some exceptions, most open source Git-based tools cannot integrate with requirements management systems.	Flosum seamlessly integrates with all requirement management tools including Jira, ServiceNow, Azure DevOps and Agile Accelerator) and directly maps components changed for each ticket.
Code scan	Requires additional licensing and configuration of another solution.	Out of the box Static code analysis.

Quality Gates

Git-based version control systems can only enforce quality gates based on code changes and are not customizable. Flosum's Version Control provides a robust set of quality control features that factor in the concepts of static code analysis, code coverage, and user story. An approval process can be configured to meet any company's requirements at any level using standard Salesforce application development tools such as workflow rules.

Ease of Use

Most Git-based Version Control systems require extensive developer knowledge and a substantial amount of training to utilize. Flosum, on the other hand, was built for admins as well as developers. It harnesses the power of the Salesforce User Interface and utilizes a click-not-code approach so that anyone who can use Salesforce can use Flosum. Furthermore, a Flosum implementation is swift and can be completed in about 48 hours, vs. the weeks and months that may be required for other solutions.

Security

Many customers store a substantial amount of confidential information in their Version Control System, and therefore cannot expose the system outside their corporate firewall. And because it is not possible to do this, they must choose a Native Version Control System which resides completely within the Salesforce platform. Flosum's entire DevOps toolset, including all of its Version Control functionality, resides within the Salesforce platform and therefore within the corporate firewall. Additionally, it offers adjacent capabilities for consent management, compliance, governance, and overall security.

Case Study

Visa finds security with native version control

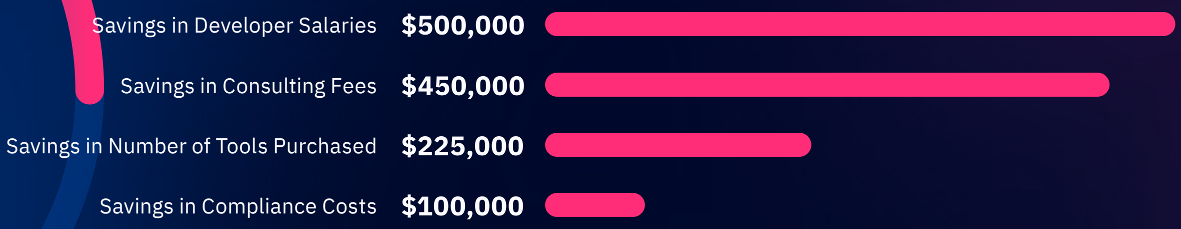
Visa is a fortune 500 company that handles a huge amount of extremely confidential customer data. Because of their strict security standards, they needed to choose a Version Control System that didn't expose their data outside their corporate firewall. Flosum is the only Version Control System that resides completely within the Salesforce Platform, and Visa was able to quickly determine that they would be able to use the tool without any security issues.

The Economic Impact of Salesforce Native Version Control

Financial Savings Obtained when utilizing a Version Control System specifically built within the Salesforce Platform

Summary of Cost Savings

Over a three-year period of ownership, utilizing a fifteen-person development team



274%
ROI

6 months
Maximum payback

$1.3MM
Savings

Case Study

Bio-Rad selects Flosum due to native Version control capabilities

Bio-Rad Laboratories, Inc. is a manufacturer of products for the life science research and clinical diagnostics markets. Bio-Rad had previously been utilizing a Git based version control system, but had found it difficult to use and struggled with the fact it couldn't support all of the Salesforce component types. Additionally, due to the sensitive nature of their healthcare driven industry, solution architecture and security were very important in their evaluations. Flosum met all of their criteria, delivering on Salesforce's intuitive clicks-not-code promise and the 100% native nature accommodated all of their security and compliance needs.

Integrations

It is understood that some development teams may want to continue to use their preference for Version Control tools in addition to Flosum. For that reason, Flosum seamlessly integrates with Github, Bitbucket, Gitlab, Azure DevOps and any other Git-based Version Control system.

Conclusion

When evaluating Git-based vs. Native Version Control it is clear that while the two technologies both provide Version Control functionality, native solutions deliver results faster, enable developer agility, application availability and resilience, reduce technical spending and offer a highly secure structure. Specifically, the architectural nature of Flosum designed with Salesforce's unique needs in mind, coupled with the culture Salesforce has built with both Admins and Developers partaking in Version Control activities, makes it a good fit for any company looking to increase ROI on their Version Control tool and proliferate developer productivity and team collaboration. Native Version Control builds lasting success within any strategic DevOps plan.

About Flosum

Flosum is an international software company that has pioneered the Salesforce application lifecycle management solution sector. Our firm focuses on successful development operations and provides customized assistance, training and knowledge transfer to our clients.

Contact us today to find out how we can help.

(844) 335 6786 — hello@flosum.com

www.flosum.com