## **Анализ существующих систем контроля версий**

**Введение**

Системы контроля версий (Version Control Systems - VCS) стали неотъемлемой частью современного процесса разработки программного обеспечения. Они позволяют эффективно управлять изменениями кода, отслеживать историю изменений, сотрудничать в командах и восстанавливать предыдущие версии проекта в случае необходимости. На сегодняшний день существует множество различных VCS, каждая из которых обладает своими преимуществами и недостатками.

Данный доклад направлен на анализ существующих систем контроля версий, их сравнение по ключевым критериям и выявление их сильных и слабых сторон. Исследование будет проводиться с целью предоставления практического руководства по выбору VCS для различных типов проектов и команд разработчиков.

**1. Основные понятия и принципы VCS**

**1.1. Понятие системы контроля версий**

Система контроля версий (СКВ) – это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов. Под версионный контроль можно поместить файлы практически любого типа, а не только программный код.

Если вы графический или веб-дизайнер и хотели бы хранить каждую версию изображения или макета – а этого вам наверняка хочется – то пользоваться системой контроля версий будет очень мудрым решением. СКВ даёт возможность возвращать отдельные файлы к прежнему виду, возвращать к прежнему состоянию весь проект, просматривать происходящие со временем изменения, определять, кто последним вносил изменения во внезапно переставший работать модуль, кто и когда внёс в код какую-то ошибку, и многое другое. Вообще, если, пользуясь СКВ, вы всё испортите или потеряете файлы, всё можно будет легко восстановить. Вдобавок, накладные расходы за всё, что вы получаете, будут очень маленькими.

**1.1.1. Локальные системы контроля версий**

Многие предпочитают контролировать версии, просто копируя файлы в отдельный каталог (как правило добавляя текущую дату к названию каталога). Такой подход очень распространён, потому что прост, но он и чаще даёт сбои. Очень легко забыть, что ты не в том каталоге, и случайно изменить не тот файл, либо скопировать файлы не туда, куда хотел, и затереть нужные файлы.

Чтобы решить эту проблему, программисты уже давно разработали локальные СКВ с простой базой данных, в которой хранятся все изменения нужных файлов (рисунок 1.1).

Одной из наиболее популярных СКВ такого типа является rcs, которая до сих пор устанавливается на многие компьютеры. Даже в современной операционной системе Mac OS X утилита rcs устанавливается вместе с Developer Tools. Эта утилита основана на работе с наборами патчей между парами версий (патч – файл, описывающий различие между файлами), которые хранятся в специальном формате на диске. Это позволяет пересоздать любой файл на любой момент времени, последовательно накладывая патчи.

**1.1.2. Централизованные системы контроля версий**

Следующей основной проблемой оказалась необходимость сотрудничать с разработчиками за другими компьютерами. Чтобы решить её, были созданы централизованные системы контроля версий (ЦСКВ). В системах такого типа, например CVS, Subversion и Perforce, есть один единственный центральный сервер, на котором хранятся все файлы, находящиеся под версионным контролем, и несколько клиентов, которые получают копии файлов с него. Много лет это был стандарт для систем контроля версий (рис. 1.2).

Такой подход имеет множество преимуществ, особенно над локальными СКВ. К примеру, все знают, кто и чем занимается в проекте. У администраторов есть чёткий контроль над тем, кто и что может делать, и, конечно, администрировать ЦСКВ намного легче, чем локальные базы на каждом клиенте.

Однако у такого подхода есть и несколько серьёзных недостатков. Наиболее очевидный – централизованный сервер является уязвимым местом всей системы. Если сервер выключается на час, то в течение часа разработчики не могут взаимодействовать, и никто не может сохранить новой версии своей работы. Если же повреждается диск с центральной базой данных и нет резервной копии, вы теряете абсолютно всё – всю историю проекта, разве что за исключением нескольких рабочих версий, сохранившихся на рабочих машинах пользователей. Локальные системы контроля версий подвержены той же проблеме: если вся история проекта хранится в одном месте, вы рискуете потерять всё.

**1.1.3. Распределённые системы контроля версий**

И в этой ситуации в игру вступают распределённые системы контроля версий (РСКВ). В таких системах как Git, Mercurial, Bazaar или Darcs клиенты не просто выгружают последние версии файлов, а полностью копируют весь репозиторий. Поэтому в случае, когда «умирает» сервер, через который шла работа, любой клиентский репозиторий может быть скопирован обратно на сервер, чтобы восстановить базу данных. Каждый раз, когда клиент забирает свежую версию файлов, он создаёт себе полную копию всех данных (рисунок 1.3).

Кроме того, в большей части этих систем можно работать с несколькими удалёнными репозиториями, таким образом, можно параллельно работать с разными группами людей по-разному в рамках одного проекта. Так, в одном проекте можно одновременно вести несколько типов рабочих процессов, что невозможно в централизованных системах.

**1.2. Краткая история Git**

Как и многие замечательные вещи, Git начинался с, в некотором роде, разрушения во имя созидания и жарких споров. Ядро Linux – действительно очень большой открытый проект. Большую часть существования ядра Linux (1991-2002) изменения к нему распространялись в виде патчей и заархивированных файлов. В 2002 году проект перешёл на проприетарную РСКВ BitKeeper.

В 2005 году отношения между сообществом разработчиков ядра Linux и компанией, разрабатывавшей BitKeeper, испортились, и право бесплатного пользования продуктом было отменено. Это подтолкнуло разработчиков Linux (и в частности Линуса Торвальдса, создателя Linux) разработать собственную систему, основываясь на опыте, полученном за время использования BitKeeper. Основные требования к новой системе были следующими:

* Скорость
* Простота дизайна
* Поддержка нелинейной разработки (тысячи параллельных веток)
* Полная распределённость
* Возможность эффективной работы с такими большими проектами как ядро Linux (как по скорости, так и по размеру данных)

С момента рождения в 2005 году Git развивался и эволюционировал, становясь проще и удобнее в использовании, сохраняя при этом свои первоначальные качества. Он невероятно быстр, очень эффективен для больших проектов, а также обладает превосходной системой ветвления для нелинейной разработки (см. главу 3).

**1.3. Основы Git**

Так что же такое Git в двух словах? Эту часть важно усвоить, поскольку если вы поймёте, что такое Git, и каковы принципы его работы, вам будет гораздо проще пользоваться им эффективно. Изучая Git, постарайтесь освободиться от всего, что вы знали о других СКВ, таких как Subversion или Perforce. В Git совсем не такие понятия об информации и работе с ней как в других системах, хотя пользовательский интерфейс очень похож. Знание этих различий защитит вас от путаницы при использовании Git.

**1.3.1. Слепки вместо патчей**

Главное отличие Git от любых других СКВ (например, Subversion и ей подобных) – это то, как Git воспринимает свои данные. В принципе, большинство других систем хранит информацию как список изменений (патчей) для файлов. Эти системы (CVS, Subversion, Perforce, Bazaar и другие) относятся к хранимым данным как к набору файлов и изменений, сделанных для каждого из этих файлов во времени, как показано на рисунке 1.4.

Git не хранит свои данные в таком виде. Вместо этого Git считает хранимые данные набором слепков небольшой файловой системы. Каждый раз, когда вы фиксируете текущую версию проекта, Git, по сути, сохраняет слепок того, как выглядят все файлы проекта на текущий момент. Ради эффективности, если файл не менялся, Git не сохраняет файл снова, а делает ссылку на ранее сохранённый файл. То, как Git подходит к хранению данных, похоже на рисунок 1.5.

Это важное отличие Git от практически всех других систем контроля версий. Из-за него Git вынужден пересмотреть практически все аспекты контроля версий, которые другие системы переняли от своих предшественниц. Git больше похож на небольшую файловую систему с невероятно мощными инструментами, работающими поверх неё, чем на просто СКВ. В главе 3, коснувшись работы с ветвями в Git, мы узнаем, какие преимущества даёт такое понимание данных.

**1.3.2. Почти все операции – локальные**

Для совершения большинства операций в Git необходимы только локальные файлы и ресурсы, т.е. обычно информация с других компьютеров в сети не нужна. Если вы привыкли пользоваться централизованными системами, где практически на каждую операцию накладывается сетевая задержка, вы, возможно, подумаете, что боги наделили Git неземной силой. Так как вся история проекта хранится прямо у вас на диске, большинство операций кажутся практически мгновенными.

К примеру, чтобы показать историю проекта, Git’y не нужно скачивать её с сервера, он просто читает её прямо из вашего локального репозитория. Поэтому историю вы увидите практически мгновенно. Если вам нужно просмотреть изменения между текущей версией файла и версией, сделанной месяц назад, Git может взять файл месячной давности и вычислить разницу на месте, вместо того чтобы запрашивать разницу у СКВ-сервера или качать с него старую версию файла и делать локальное сравнение.

Кроме того, работа локально означает, что мало чего нельзя сделать без доступа к Сети или VPN. Если вы в самолёте или в поезде и хотите немного поработать, можно спокойно делать коммиты, а затем отправить их, как только станет доступна сеть. Если вы пришли домой, а VPN-клиент не работает, всё равно можно продолжать работать. Во многих других системах это невозможно или же крайне неудобно. Например, используя Perforce, вы мало что можете сделать без соединения с сервером. Работая с Subversion и CVS, вы можете редактировать файлы, но сохранить изменения в базу данных нельзя (потому что база данных недоступна). Возможно, вы думаете, что не так уж это всё и важно, но потом вы удивитесь, насколько это меняет дело.

**1.3.3. Git следит за целостностью данных**

Для всего, что сохраняется в Git, сначала вычисляется контрольная сумма, и в дальнейшем она же и используется для поиска содержимого. Поэтому невозможно изменить содержимое файла или каталога так, чтобы Git не узнал об этом. Эта функциональность встроена в сам фундамент Git и является важной составляющей его философии. Если информация потеряется при передаче или повредится на диске, Git всегда сможет это обнаружить.

Механизм, используемый Git’ом для вычисления контрольных сумм, называется SHA-1 хешем. Это строка из 40 шестнадцатеричных символов (0-9 и a-f), вычисляемая в Git на основе содержимого файла или структуры каталога. SHA-1 хеш выглядит примерно так:

24b9da6552252987aa493b52f8696cd6d3b00373

Работая с Git’ом, вы будете встречать эти хеши повсюду, поскольку он их очень широко использует. Фактически, в своей базе данных Git сохраняет всё не по именам файлов, а по хешам их содержимого.

**1.3.4. Чаще всего данные в Git только добавляются**

Практически все действия, которые вы совершаете в Git, только добавляют данные в базу. Очень сложно заставить систему удалить данные или сделать что-то неотменяемое. Можно, как и в любой другой СКВ, потерять данные, которые вы ещё не сохранили, но как только они зафиксированы, их очень сложно потерять, особенно если вы регулярно отправляете изменения в другой репозиторий.

**1.3.5. Три состояния**

В Git файлы могут находиться в одном из трёх состояний: зафиксированном, изменённом и подготовленном. «Зафиксированный» значит, что файл уже сохранён в вашей локальной базе. К изменённым относятся файлы, которые поменялись, но ещё не были зафиксированы. Подготовленные файлы – это изменённые файлы, отмеченные для включения в следующий коммит.

Таким образом, в проектах, использующих Git, есть три части: каталог Git (Git directory), рабочий каталог (working directory) и область подготовленных файлов (staging area).

* Каталог Git – это место, где Git хранит метаданные и базу данных объектов вашего проекта. Это наиболее важная часть Git, и именно она копируется, когда вы клонируете репозиторий с другого компьютера.
* Рабочий каталог – это извлечённая из базы копия определённой версии проекта. Эти файлы достаются из сжатой базы данных в каталоге Git и помещаются на диск для того, чтобы вы их просматривали и редактировали.
* Область подготовленных файлов – это обычный файл, обычно хранящийся в каталоге Git, который содержит информацию о том, что должно войти в следующий коммит. Иногда его называют индексом (index), хотя в последнее время становится общепринятым называть его областью подготовленных файлов (staging area).

Стандартный рабочий процесс с использованием Git выглядит примерно так:

1. Вы вносите изменения в файлы в своём рабочем каталоге.
2. Подготавливаете файлы, добавляя их слепки в область подготовленных файлов.
3. Делаете коммит, который берёт подготовленные файлы из индекса и помещает их в каталог Git на постоянное хранение.

Если рабочая версия файла совпадает с версией в каталоге Git’a, файл считается зафиксированным. Если файл изменён, но добавлен в область подготовленных данных, он подготовлен. Если же файл изменился после выгрузки из БД, но не был подготовлен, то он считается изменённым.

**2. Сравнительный анализ VCS**

**2.1. Критерии сравнения:**

* **Модель хранения данных:** централизованная, распределенная, локальная. Этот критерий определяет способ хранения информации о версиях файлов и влияет на скорость работы, безопасность и гибкость системы.
* **Скорость работы:** производительность операций с файлами (добавление, изменение, удаление, слияние). Скорость работы VCS является важным фактором для производительности разработки, особенно для больших проектов с множеством файлов.
* **Удобство использования:** интерфейс пользователя, документация, наличие графических клиентов. Удобство использования VCS влияет на скорость обучения и удобство работы разработчиков с системой.
* **Функциональность:** поддержка ветвления, слияния, отката, тегирования, истории изменений. Функциональность VCS определяет ее возможности по управлению изменениями в проекте и позволяет решать различные задачи разработки.
* **Безопасность:** защита данных от потери, несанкционированного доступа, возможность восстановления данных в случае сбоя. Безопасность VCS является важным фактором для защиты проекта от потери данных и несанкционированного доступа.
* **Интеграция:** возможность интеграции с другими инструментами разработки (IDE, CI/CD, платформы хостинга). Интеграция VCS с другими инструментами разработки позволяет автоматизировать процессы разработки и увеличить производительность команды.
* **Сообщество:** активность разработки, наличие документации, форумов, поддержки. Активное сообщество VCS обеспечивает более быструю реакцию на проблемы, более качественную документацию и большую возможность получения помощи от других разработчиков.

**2.2. Таблица сравнения:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **VCS** | **Модель хранения данных** | **Скорость работы** | **Удобство использования** | **Функциональность** | **Безопасность** | **Интеграция** | **Сообщество** |
| **Git** | Распределенная | Высокая | Средняя | Высокая | Высокая | Высокая | Очень высокая |
| **SVN** | Централизованная | Средняя | Средняя | Средняя | Средняя | Средняя | Высокая |
| **Mercurial** | Распределенная | Высокая | Средняя | Высокая | Высокая | Средняя | Средняя |
| **Perforce** | Централизованная | Высокая | Низкая | Высокая | Высокая | Средняя | Средняя |
| **CVS** | Централизованная | Низкая | Низкая | Низкая | Низкая | Низкая | Низкая |
| **Bazaar** | Распределенная | Средняя | Средняя | Средняя | Средняя | Средняя | Средняя |
| **Darcs** | Распределенная | Низкая | Низкая | Высокая | Средняя | Низкая | Низкая |

**2.3. Подробное описание VCS:**

**2.3.1. Git**

* **Описание:** Распределенная система контроля версий, отличающаяся высокой скоростью работы, гибкостью и мощными возможностями ветвления. Git широко используется в современной разработке и является стандартом de-facto для многих проектов.
* **Преимущества:**
  + **Высокая скорость работы:** Git оптимизирован для быстрого обращения с файлами и эффективного управления версиями. Это особенно важно для больших проектов с множеством файлов.
  + **Гибкость и мощные возможности ветвления:** Git позволяет легко создавать и управлять ветвями, что позволяет разработчикам работать над различными частями проекта одновременно, без влияния друг на друга.
  + **Широкое сообщество и документация:** Git имеет большое и активное сообщество разработчиков, что обеспечивает отличную документацию, многочисленные ресурсы и быструю помощь при возникновении проблем.
  + **Интеграция с многими инструментами разработки:** Git интегрируется с большинством современных IDE, CI/CD систем и платформ хостинга, что позволяет автоматизировать процессы разработки
* **Недостатки:**
  + **Сложный интерфейс пользователя:** Git имеет более сложный интерфейс пользователя, чем некоторые другие VCS, и требует времени на изучение. Однако, существуют множество графических клиентов для Git, которые упрощают его использование.

**2.3.2. SVN**

* **Описание:** Централизованная система контроля версий, считающаяся более простой в использовании, чем Git. SVN широко использовалась в прошлом, но в последнее время ее популярность снизилась из-за возникновения более современных распределенных VCS, таких как Git.
* **Преимущества:**
  + **Простой интерфейс пользователя:** SVN имеет более простой интерфейс пользователя, чем Git, что делает ее более доступной для новичков.
  + **Широкое сообщество и документация:** SVN имеет большое и активное сообщество разработчиков, что обеспечивает хорошую документацию и поддержку.
  + **Хорошо подходит для малых и средних проектов:** SVN является хорошим выбором для малых и средних проектов, которые не требуют сложных возможностей ветвления.
* **Недостатки:**
  + **Ниже скорость работы:** SVN работает медленнее, чем Git, особенно для больших проектов с множеством файлов.
  + **Ограниченные возможности ветвления:** SVN имеет более ограниченные возможности ветвления, чем Git, что может быть неудобно для проектов с большим количеством ветвей.

**2.3.3. Mercurial**

* **Описание:** Распределенная система контроля версий, похожая на Git, но с более простым интерфейсом пользователя. Mercurial является популярным выбором для проектов, которые ищут более простую в использовании альтернативу Git.
* **Преимущества:**
  + **Простой интерфейс пользователя:** Mercurial имеет более простой интерфейс пользователя, чем Git, что делает ее более доступной для новичков.
  + **Высокая скорость работы:** Mercurial работает быстро и эффективно, особенно для проектов с большим количеством файлов.
  + **Гибкость и мощные возможности ветвления:** Mercurial позволяет легко создавать и управлять ветвями, как и Git.
* **Недостатки:**
  + **Меньше сообщества и документации, чем у Git:** Mercurial имеет меньшее сообщество и документацию, чем Git, что может усложнить поиск помощи и решение проблем.

**2.3.4. Perforce**

* **Описание:** Централизованная система контроля версий, ориентированная на большие проекты с множеством разработчиков. Perforce известна своей высокой скоростью работы и надежностью, что делает ее популярным выбором для крупных коммерческих проектов.
* **Преимущества:**
  + **Высокая скорость работы:** Perforce работает очень быстро и эффективно, даже для проектов с миллионами файлов.
  + **Мощные возможности управления доступом:** Perforce имеет мощные возможности управления доступом, что позволяет контролировать доступ к проекту на уровне файлов и папок.
  + **Хорошо подходит для больших проектов:** Perforce является хорошим выбором для больших проектов с множеством разработчиков, которые требуют высокой скорости работы и надежности.
* **Недостатки:**
  + **Сложный интерфейс пользователя:** Perforce имеет более сложный интерфейс пользователя, чем некоторые другие VCS, и требует времени на изучение.
  + **Высокая стоимость лицензии:** Perforce является платной системой и имеет высокую стоимость лицензии.

**2.3.5. CVS**

* **Описание:** Одна из первых централизованных систем контроля версий. CVS была популярна в прошлом, но в настоящее время ее использование ограничено из-за возникновения более современных VCS, таких как SVN и Git.
* **Преимущества:**
  + **Простая в использовании:** CVS относительно проста в использовании и имеет несложный интерфейс.
* **Недостатки:**
  + **Низкая скорость работы:** CVS работает медленно, особенно для больших проектов с множеством файлов.
  + **Ограниченные возможности ветвления:** CVS имеет ограниченные возможности ветвления, что делает ее неудобной для проектов с большим количеством ветвей.
  + **Плохая безопасность:** CVS имеет плохую безопасность и не обеспечивает защиту данных от потери и несанкционированного доступа.

**2.3.6. Bazaar**

* **Описание:** Распределенная система контроля версий, считающаяся более простой в использовании, чем Git. Bazaar является хорошим выбором для проектов, которые ищут более простую в использовании альтернативу Git с возможностями распределенного управления версиями.
* **Преимущества:**
  + **Простой интерфейс пользователя:** Bazaar имеет более простой интерфейс пользователя, чем Git, что делает ее более доступной для новичков.
  + **Гибкость и мощные возможности ветвления:** Bazaar позволяет легко создавать и управлять ветвями, как и Git.
* **Недостатки:**
  + **Меньше сообщества и документации, чем у Git:** Bazaar имеет меньшее сообщество и документацию, чем Git, что может усложнить поиск помощи и решение проблем.

**2.3.7. Darcs**

* **Описание:** Распределенная система контроля версий, отличающаяся от других VCS своей нетрадиционной моделью хранения данных. Darcs известна своей мощной системой ветвления и способностью отслеживать изменения в истории проекта.
* **Преимущества:**
  + **Мощные возможности ветвления:** Darcs имеет мощные возможности ветвления и позволяет легко отслеживать изменения в истории проекта.
  + **Хорошо подходит для проектов с большим количеством ветвей:** Darcs является хорошим выбором для проектов с большим количеством ветвей, которые требуют гибкости и мощных возможностей управления ветвями.
* **Недостатки:**
  + **Низкая скорость работы:** Darcs работает медленнее, чем большинство других VCS.
  + **Сложный интерфейс пользователя:** Darcs имеет более сложный интерфейс пользователя, чем большинство других VCS, и требует времени на изучение.
  + **Небольшое сообщество:** Darcs имеет небольшое сообщество разработчиков, что может усложнить поиск помощи и решение проблем.

**3. Выбор VCS для разных проектов**

**3.1. Малые проекты:**

* **Рекомендуемые VCS:** SVN, Mercurial, Bazaar.
* **Причины:** простота в использовании, невысокие требования к ресурсам, хорошо подходят для проектов с небольшим количеством разработчиков и небольшим объемом кода.

**3.2. Средние проекты:**

* **Рекомендуемые VCS:** Git, SVN.
* **Причины:** хорошее сочетание функциональности и удобства использования, подходят для проектов с большим количеством разработчиков и большим объемом кода, обеспечивают хорошую скорость работы и безопасность.

**3.3. Большие проекты:**

* **Рекомендуемые VCS:** Git, Perforce.
* **Причины:** высокая скорость работы, мощные возможности ветвления, хорошая безопасность, подходят для проектов с очень большим количеством разработчиков и очень большим объемом кода, обеспечивают высокую производительность и надежность.

**3.4. Проекты с большим количеством ветвей:**

* **Рекомендуемые VCS:** Git, Darcs.
* **Причины:** гибкость и мощные возможности ветвления, позволяют легко создавать и управлять большим количеством ветвей, что необходимо для проектов с сложной структурой и множеством параллельных разработок.

**4. Заключение**

Выбор системы контроля версий зависит от конкретных требований проекта и команды разработчиков. Не существует идеальной VCS, каждая система имеет свои сильные и слабые стороны. Важно внимательно изучить характеристики разных VCS, сравнить их с требованиями проекта и выбрать систему, которая наиболее эффективно решает задачи разработки.

**5. Список литературы**

* [Название книги 1](https://word-edit.officeapps.live.com/we/%D1%81%D1%81%D1%8B%D0%BB%D0%BA%D0%B0)
* [Название книги 2](https://word-edit.officeapps.live.com/we/%D1%81%D1%81%D1%8B%D0%BB%D0%BA%D0%B0)
* [Название статьи 1](https://word-edit.officeapps.live.com/we/%D1%81%D1%81%D1%8B%D0%BB%D0%BA%D0%B0)
* [Название статьи 2](https://word-edit.officeapps.live.com/we/%D1%81%D1%81%D1%8B%D0%BB%D0%BA%D0%B0)
* [Название документации Git](https://word-edit.officeapps.live.com/we/%D1%81%D1%81%D1%8B%D0%BB%D0%BA%D0%B0)
* [Название документации SVN](https://word-edit.officeapps.live.com/we/%D1%81%D1%81%D1%8B%D0%BB%D0%BA%D0%B0)

**6. Дополнительные ресурсы:**

* [Веб-сайт Git](https://word-edit.officeapps.live.com/we/%D1%81%D1%81%D1%8B%D0%BB%D0%BA%D0%B0)
* [Веб-сайт SVN](https://word-edit.officeapps.live.com/we/%D1%81%D1%81%D1%8B%D0%BB%D0%BA%D0%B0)
* [Веб-сайт Mercurial](https://word-edit.officeapps.live.com/we/%D1%81%D1%81%D1%8B%D0%BB%D0%BA%D0%B0)
* [Веб-сайт Perforce](https://word-edit.officeapps.live.com/we/%D1%81%D1%81%D1%8B%D0%BB%D0%BA%D0%B0)

**Примечание:** Данный доклад является кратким обзором систем контроля версий. Для получения более глубокой информации рекомендуется изучить документацию по каждой системе и провести сравнительное тестирование. Также важно учитывать особенности конкретного проекта и команды разработчиков при выборе VCS.