УДК 004.94:378

СОВРЕМЕННАЯ МЕТОДИКА ОРГАНИЗАЦИИ ПРОЦЕССА РАЗРАБОТКИ ПРОГРАММ

Канд. техн. наук, доц. ДАДЫКИН А.К.
Магистрант ЕРМОЛАЕВ А.А.

Аннотация:
В статье «СОВРЕМЕННАЯ МЕТОДИКА ОРГАНИЗАЦИИ ПРОЦЕССА РАЗРАБОТКИ ПРОГРАММ» рассматривается методика организации процесса разработки программного обеспечения на основе непрерывной интеграции, которая объединяет в единое целое множество техник, облегчающих разработку и сопровождение программ в промышленных масштабах.

Существуют множество техник, облегчающих разработку и сопровождение программ в промышленных масштабах – тестирование, система управления версиями, система отслеживания ошибок, автоматизированная система сборки и развертывания и т.д. Непрерывная интеграция(ContinuousIntegration, CI) – объединяет все эти компоненты в единое целое.

Когда большое число разработчиков совместно работают над сложными программными проектами, интеграция разных частей кода может превратиться в длительный процесс с непредсказуемыми результатами. Однако на проектах, где процесс разработки строится на основе методик CI,проблемы и риски, связанные с интеграцией, сведены к минимуму.

В проекте, выполняемом одним человеком, интеграция программного обеспечения (ПО) – не слишком существенная проблема, но при увеличении сложности проекта, возникает необходимость в интеграции и проверке слаженной работы компонентов ПО. Дожидаться конца проекта для проведения интеграции и выявления всего спектра возможных ошибок – неразумно и к тому же не способствует качеству ПО, а зачастую даже приводит к удорожанию и задержке сдачи проекта. CIснижает трудоёмкость интеграции и делает её более предсказуемой, за счет наиболее раннего обнаружения и устранения ошибок и противоречий.

В сущности, CI гарантирует совместимость недавних изменений с остальной частью ПО. На более высоком уровне CI повышает коллективную ответственность группы и снижает трудоемкость проекта, уменьшая объем ручного труда, выполняемого при каждой интеграции. Если описать данную методику в несколько фраз, то это: “всегда есть рабочая версия”, “автоматизированная сборка”, “всегда известно, в каком состоянии прибывает проект”.Структуру CI можно представить в виде модулей,изображённых на рисунке 1.

Организация единого репозитория и правила возврата исходного кода

Программные проекты состоят из множества файлов, необходимых для построения сборки. Когда в разработку вовлечено множество людей, отслеживание всех изменений требует значительных усилий, поэтому команды разработчиков используют специализированные утилиты для организации данного процесса – сервера исходного кода с системой контроля версий.

Процесс нужно организовать таким образом, чтобы все участники проекта знали, где репозиторий находится, и хранили все исходные файлы только там. Многие допускают ошибку, храня относящиеся к проекту файлы такие как: тестировочные скрипты, скрипты баз данных, установочные скрипты и сторонние библиотеки вне репозитория. Всё это неотъемлемые части проекта, необходимые для того, чтобы

Рисунок 1. Из чего состоит CI

можно было получить исходные файлы из репозитория на любой машине и суметь собрать рабочую версию проекта. Единственное, что должно понадобиться на целевой машине, это самостоятельные продукты, такие как: операционная система, сервер баз данных, среда разработки или средство, способное собрать проект. Также является хорошей практикой держать на сервере исходного кода файлы настроек для интегрированной среды разработки и для систем оценки качества кода, так как участникам проектажелательно использовать одни и те же настройки.Таким образом, на сервере исходного кода должно храниться все, что требуется для построения сборки, и ничего, что в итоге строится.Если хранить на сервере исходного кода готовые сборки, это явный признак того, что в существующем процессе разработки есть проблемы, не позволяющие получитьконкретную версию сборки в любой момент времени.

Отличной функцией систем контроля версий является создание веток для разветвления процесса разработки. Это очень полезно, но создание веток нужно свести к минимуму. Как правило, создаётся главная ветка, которая называется стволом проекта (trunk), в ней ведётся основная разработка. Ветки,как правило, создаются для отдельных фаз разработки и для предрелизных версий, в которых ведется устранение ошибок. Так же нужно хранить предыдущие версии продукта,чтобы было возможно собрать ранние релизы.

Возврат изменений и сборка проекта на интеграционном сервере выполняется в первую очередь для того, чтобы оповестить других разработчиков о том, как быстро развивается разработка. Так же это помогает писать рабочий код другим разработчикам, которые ориентируются на последние изменения. Перед каждым возвратом изменений разработчик должензабрать изменения из “trunk”а проекта и разрешить все конфликты локально, и только затем возвращать код.

Чем чаще происходят возвраты, тем чаще будут обнаружены конфликты между различными изменениями и как следствие решены быстро. Не следует допускать долгой локальной разработки без возврата изменений, так как это повлечёт за собой долгое разрешение конфликтов.

Должно быть выработано правило, что каждый разработчик должен возвращать свои

изменения не реже чем раз в день, если у него было несколько заданий, то лучше возвращать изменения после выполнения каждого из них. Это помогает отслеживать прогресс, а также даёт ощущение прогресса. Это даёт людям осознание того, что они могут сделать что-то значимое за небольшой отрезок времени, и это стимулирует их работу.

Автоматизация сборки и ускоренное построение

Преобразование исходного кода в исполняемые файлы (артефакты) может оказаться сложным процессом, включающим компиляцию, перемещение файлов и т.д. Все эти задачи должны быть автоматизированы. Принуждение участников команды вводить не простые команды и постоянно нажимать кнопки во множестве окон является пустой тратой времени, и влечёт за собой потенциальные ошибки.

Сборка большого проекта часто может занимать много времени, но не обязательно каждый раз выполнять абсолютно все шаги. В зависимости от поставленных целей можно варьировать процесс сборки.Должна быть возможность выбора запуска сборки с последующими тестами, без них, или же с переделённым набором тестов и т.д. Автоматизированные скрипты построения должны быть достаточно гибкими, чтобы позволить настроить процесс сборки в соответствии с различными ситуациями.

Важной целью CI является обеспечение как можно быстрой обратной связи. Поэтомустремиться надо к тому, чтобы обычное построение занимало около 10 минут, и полное построение со всеми тестами не более часа[1]. Для ускорения процесса сборки можно реализовать поэтапное построение. К примеру, не обязательно каждый раз разворачивать базу данных с нуля, или разворачивать тестовые данные, если просто поменялось оформление приложения. Так же можно делить тесты на части, и затем предусмотреть возможность запуска только части из них. Если тесты занимают большое количество времени, то можно организоватьраспределённый запуск на нескольких машинах одновременно.

Автоматизациясамотестирования сборки

Многие считают, что сборкой является компиляцияпроекта,и в итоге получение исполняемой части программы. ПО возможно будет после этого запускаться, но это вовсе не означает, что оно будет работать корректно. Современные строго типизированные языки позволяют отлавливать множество ошибок, но гораздо большее количество ошибок не отлавливается на уровне компиляции.

Хорошим способом быстро и эффективно отловить ошибки является включение автоматизированных тестов в процесс сборки. Тестирование, конечно, не сможет полностью избавить нас от ошибок, но будет возможность отловить достаточно количество из них, что делает самотестирование очень важным и полезным процессом. Если во время сборки какие-нибудь из тестов не были пройдены успешно, это должно означать, что сборка прошла неуспешно.

Сервер непрерывной интеграции

Совершая ежедневные возвраты, команда получает результаты тестов сборки. Всё это делается для того, чтобы постоянно содержать “trunk” проекта в рабочем состоянии. На практике всё конечно не так гладко, одна из причин это дисциплина, люди часто не обновляют локальную копию проекта, забрав изменения из “trunk”а, перед тем как возвращать код. Но есть и другие, к примеру, различные особенности окружения. Именно по этой причине требуется гарантия, что сборка была успешной на интеграционном сервере, только после этого она будет считаться рабочей.

За каждое изменение на интеграционном сервере отвечает разработчик, который его зафиксировал, поэтому если изменения повлекли за собой провал сборки, то разработчик, который их зафиксировал, назначается ответ-

ственным за исправление данной проблемы. Поэтому перед уходом с рабочего места лучше не фиксировать изменения, если потом не будет возможности их исправить.

Сервер непрерывной интеграции мониторит все изменения на сервере исходного кода, и как только изменения будут обнаружены, он приступает к сборке проекта.Затем извещает разработчиков,которые вернулиэти изменения, о результате сборки, например по почте. Хорошей практикойявляются регулярные построения по расписанию, такие как, например, ночью. Но этоне тоже самое, что и непрерывная сборка. Ночные сборки делаются для того, чтобы можно было работать с последней рабочей сборкой по состоянию на утро.Ключевой частью процесса CI является быстрое восстановление рабочей версии сборки, если вдруг она стала не рабочей вследствие внесённых изменений. Весь смысл работы по принципу CI в том, что вы всегда ведёте разработку, отталкиваясь от последней стабильной точки.

Не рабочая сборка не является катастрофой, это лишь воспитывает в людях привычку устранять конфликты у себя на локальных машинах, прежде чем возвращать изменения в общий репозиторий. Но со временем все учатся данным правилам, у командыв итоге будет очень малое количество неуспешных сборок.

Тестирование в клоне рабочей среды

Ещё одной целью тестирования является избавление от ошибок, которые могут возникнуть в связи с особенностями окружения. Поэтому следует проводить тестирование в среде идентичной производственной. Если тестировать сборку в среде отличной от производственной, то с каждой особенностью среды будет риск получить иной результат.

Тестовое окружение должно соответствовать производственному настолько, насколько это возможно. Используйте такое же программное обеспечение базы данных и соответствующей версии, такую же версию операционной системы и запускайте всё на таком же оборудовании. Не исключено, что воссоздать клон производственной среды может оказаться сложной задачей потому как там может быть установлено дорогостоящее оборудование и программное обеспечение. Но всё равно нужно постараться сделать всё возможное, и учитывать те риски, которые могут быть связаны с различиями между окружениями.

Ещё одним способом построить многомашинную среду для тестирования и имитации различных окружений является виртуализация. Этот способ существенно может сократить средства, затраченные на оборудование.

Беспрепятственный доступ к последним артефактам для каждого участника команды

В процессе разработки, каждый раз, когда требуется убедиться в работоспособности той или иной части приложения, необходимо запустить это приложение и проверить его на работоспособность. Потому как необходимость в запуске приложения возникает часто, нужно убедиться, что последняя собранная версия приложения легкодоступна для теста, или просто для просмотра последних изменений.

Сделать это не так сложно: убедитесь, что все участники проекта знают, где располагаются данные исполняемые файлы, и что главное, предоставляться должно только последнее рабочее приложение. Особенно, если оно предназначается для демонстрации, и демонстратору не очень бы хотелось оказаться в ситуации, когда во время показа у него что-либо начнёт работать не так, как ожидалось.

Обратная связь

Задачей CI стоит организовать хорошее взаимодействие между участниками проекта, и поэтому каждый должен иметь возможность в любой момент с лёгкостью посмотреть список изменений и состояние сборки, или причину неуспешного выполнения сборки.

Большинство серверов непрерывной интеграции предоставляют для просмотра данной информации веб-сайт, который предостав-

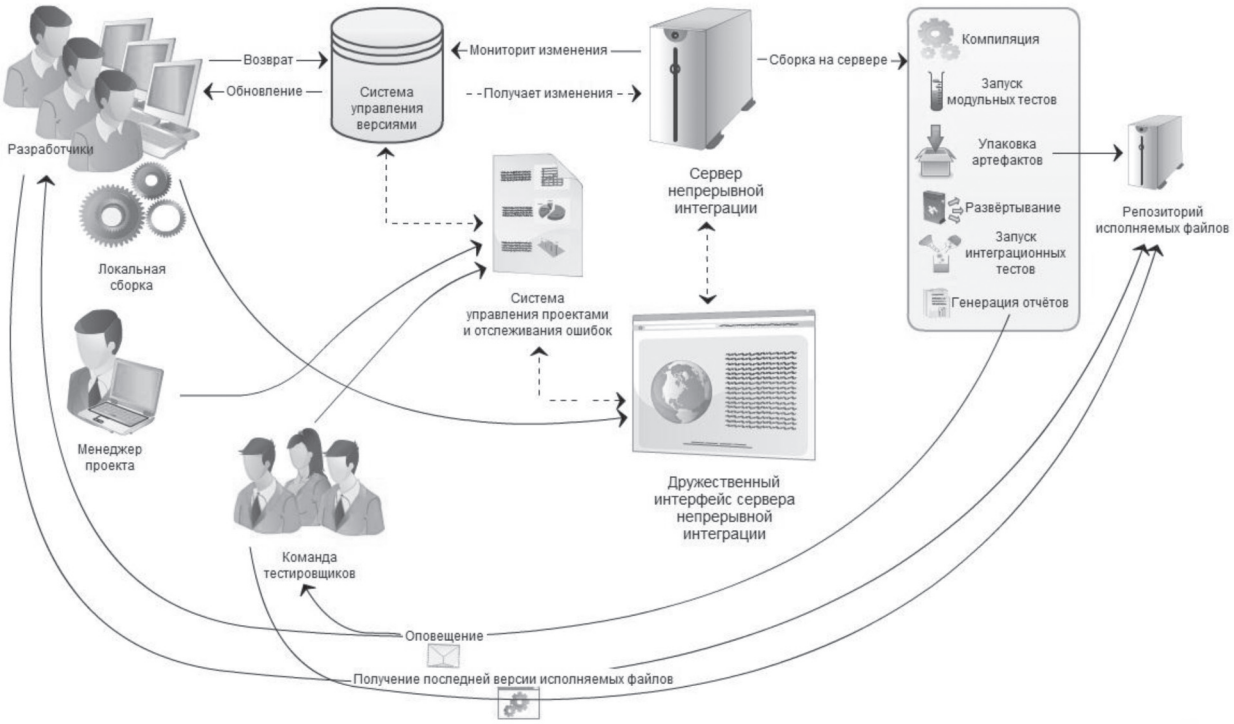

Рисунок 2.Организация процесса разработки

ляетмножество необходимой информации для отображения полной картины происходящего на сервере, такой как: состояние сборок, истории изменений, успешных и неуспешных сборок, время, затраченное на сборки, результаты пройденных тестов и многое другое.Данная информация особенно необходима, когда люди, работающее над проектом, разбиты на группы и работают в разных частях света, но всё время должны быть в курсе происходящего. Или когда команда работает сразу над несколькими проектами, которой так же необходимо быстро узнать состояние всех проектов.

Автоматизация развёртывания

Для организации контроля качества приложения необходимо иметь несколько окружений, одни – для построения и запуска автоматизированных тестов и другие – для развертывания исполняемых версий приложения для тестирования. Для автоматизированного построения, развёртывания и тестирования всех сред следует использовать один и тот же набор скриптов. Развёртывание в тестовой среде, не должно сильно отличаться от развёртывания в производственной.При развёртывании в производственной и предпроизводственной средах,необходимо предусмотреть возможность отката в исходное состояние, если что-либо пойдет не так. Время от времени случаются различные непредвиденные ситуации, при которых всё идёт не так как было запланировано и лучше сделать так, чтобы все это не вызывало остановку работы производственной среды, и было легко устранено.

Весь описанный выше процесс можно представить в виде диаграммы, изображённой на рисунке 2.

ВЫВОДЫ

Организация процесса разработки ПО вокруг методикиCI позволяет:

снизить трудоёмкость интеграции;

сделать её более предсказуемой за счет наиболее раннего обнаружения ошибок;

устранить ошибки и противоречия;

узнать в любой момент времени общее состоянием разрабатываемого ПО.

Основным преимуществом CI является избавление от рутины. Простота и повто-

ряемость процесса интеграции позволяет кому угодно – любому участнику проекта: тестировщику, разработчику, аналитику и другим собрать и запустить проект, и совершить ошибку при этом невозможно, потому что все эти операции автоматизированы.

У CI есть и недостатки, но они не так значительны как ее безусловные преимущества. Например, к недостаткам относят необходимость иметь выделенный сервер, тратить время на поддержку работы этого сервера. Первый аргумент с каждым днем все слабее, аппаратная часть стоит все меньше по сравнению с постоянно дорожающим временем разработчика. Время на поддержание сервера совершенно незначительно в сравнении со временем, затрачиваемым на обнаружение проблем другим способом, и запоздалое их устранение.

ЛИТЕРАТУРА

1. **Сайт** Мартина Фаулера[Электронный ресурс]– Режим доступа: http:// martinfowler. com/ articles/continuousIntegration.html – Дата доступа: 27.01.2013.

2. **Duvall**, PaulM. Continuous Integration Improving Software Qualityand Reducing Risk / PaulM. Duvall, Stephen Матиас, Andrew Glover. – USA,«Addison-Wesley», 2007. – 336 с.:ил.