ТЕХНОЛОГИЯ НЕПРЕРЫВНОЙ ИНТЕГРАЦИИ В ПРОЦЕССЕ РАЗРАБОТКИ ПРОГРАММНЫХ СРЕДСТВ

Турчин Артур Чеславович

бакалавр, Белорусский государственный университет информатики и радиоэлектроники, Беларусь, г. Минск

В наше время сложно представить разработку программного обеспечения без использования методологий гибкой разработки: манифеста, определяющего способ мышления и содержащего основные ценности и принципы, на которых базируются подходы к разработке программного обеспечения, подразумевающих под собой интерактивную разработку, периодическое предоставление обновленных требований заказчика и их реализацию посредством самоорганизующихся групп, сформированных из специалистов различных профилей (разработчики, тестировщики, хостинг-администраторы и т.д.).

Непрерывная интеграция – это практика разработки программного обеспечения, которая заключается в слиянии рабочих копий в общую основную ветвь разработки несколько раз в день и выполнении частых автоматизированных сборок проекта для скорейшего выявления потенциальных дефектов и решения интеграционных проблем. В обычном проекте, где над разными частями системы разработчики трудятся независимо, стадия интеграции является заключительной. Она может непредсказуемо задержать окончание работ, так как дефекты программного средства будут выявлены на стадии интеграции изменений. Переход к непрерывной интеграции позволяет снизить трудоёмкость циклического внедрения изменений и сделать процесс разработки более предсказуемым за счет раннего обнаружения и устранения ошибок и противоречий. Но основным преимуществом является сокращение стоимости исправления дефекта за счёт раннего его выявления. Непрерывная интеграция впервые названа и предложена Гради Бучем в 1991 г [3, с. 209].

Обратимся к принципам методологии гибкой разработки программного обеспечения и выделим те, на которые непосредственно влияет или может повлиять процесс непрерывной интеграции:

- удовлетворение потребностей заказчика благодаря ранней и регулярной поставке программного обеспечения;

- изменение требований даже на поздних этапах разработки;

- учащение выпуска обновлений продукта;

- повышение мотивации команды за счет повышения технологичности процесса разработки;

- стабильный и работающий продукт – основной показатель прогресса;

- устойчивый и постоянный ритм процесса разработки;

- постоянное внимание к техническому совершенству и качеству проектирования.

Частая поставка нового функционала в уже работающий проект ведет к тому, что кроме потери качества отдельных частей продукта со временем начинают проявляться проблемы развертывания и доставки обновлений. Как правило, кодовая база и список доступного функционала со временем растет и усложняется и наступает критический момент, когда

доставка занимает 60 и более минут рабочего времени, в течении которого часть команды сидит и ожидает появления нового функционала у заказчика, бездействуя и нерационально используя рабочее время. Проблемы проявляются при осуществлении доставки обновления клиенту и зачастую обнаруживаются уже на стороне заказчика в работающем продукте, а не на предыдущих стадиях, когда их устранение не повлияло бы на итоговое качество программного средства. В результате от заказчика поступает претензия к качеству поставляемых обновлений, репутация продукта страдает, а на ведение переговоров и устранение недостатков уходит незапланированное время. Таким образом, если проблему качества обновлений не решать своевременно, то разработчики, даже с хорошей подготовкой и опытом, теряют чувство профессионализма и хуже выполняют свои обязанности, находятся в угнетенном состоянии, пока не наступает эмоциональное выгорание, часто приводящее к смене места работы.

Методология гибкой разработки подразумевает, что все члены команды должны находится в устойчивом работоспособном состоянии постоянно. Иначе команда с течением времени будет терять темп, а также большую часть времени тратить на исправление ошибок вместо разработки нового функционала, в свою очередь продукт потеряет предсказуемость в разработке.

Таким образом, поддержка процесса непрерывного внедрения изменений в программное средство требует определенных навыков команды, ритмичности и самодисциплины. Чтобы обеспечить стабильность и безопасность процесса разработки при использовании гибкой методологии в сложных программных средствах существует практика непрерывной интеграции, суть которой будет раскрыта в следующем разделе данной работы.

Обязательные требования к продукту, соблюдение которых необходимы для внедрения процесса непрерывной интеграции, заключаются в следующем:

- исходный код продукта и все, что необходимо для сборки и тестирования, должны храниться в системе контроля версий;

- операции сборки и тестирования всего проекта автоматизированы и легко могут быть вызваны из стороннего программного средства.

На рисунке 1 представлены стадии процесса непрерывной интеграции. Данная практика направлена на то, чтобы своими преимуществами компенсировать недостатки методологии гибкой разработки:

- Проблемы интеграции выясняются сразу же после возникновения, что в результате оказывается дешевле, чем обнаружение дефектов на поздней стадии.

- Предполагает немедленный запуск модульных тестов после каждых зафиксированных изменений, повышая тем самым устойчивость и надежность программного средства.

- Гарантирует наличие стабильной версии в любой момент времени, например, для демонстрации или тестирования.

- Немедленная реакция на неполный или нерабочий код приучает команду к коротким продуктивным циклам работы.

Рисунок 1. Структурная схема стадий непрерывной интеграции

Рассмотрим перечисленные свойства более подробно. Как правило, обнаружение проблемы на ранних стадиях ведет к более дешевому их устранению: у команды все еще в фокусе внедряемый функционал, как с точки зрения написания кода, так и с точки зрения требований к функциональности, причем не только у разработчика, но и у остальных участников команды. Таким образом, шанс выявления проблем после завершения интеграции значительно снижается [1].

Модульный, или юнит-тест – это программный код, задача которого проверить корректность отдельных составляющих программы. Главная обязанность разработчиков – писать тесты для каждого нетривиального метода или функции, что обеспечивает проверку изменений на появление ошибок в уже оттестированной функциональности, тем самым сводя к минимуму трудозатраты на обнаружение и устранение данных дефектов [2, с. 82]. Так как практика непрерывной интеграции обеспечивает автоматический запуск данных тестов, шанс того, что клиенту будет доставлен код с игнорированием результатов модульного-тестирования, сведен к нулю.

Благодаря системе контроля версий, без которой в принципе невозможно использовать практику непрерывной интеграции, в любой момент времени есть возможность получить стабильную версию. Обычно стабильная версия хранится в так называемых мастер-ветках, но даже если команда по какой-либо причине испортила ее, всегда есть возможно вернуться на точку фиксации изменений до внесения дефекта [4, с. 21]. Тем самым, в любую единицу времени может быть запущено новое окружение с полностью рабочим продуктом, будь то требование заказчика или падение работоспособности сервера.

Последнее из преимуществ – самое неочевидное на первый взгляд. Может показаться, что задачи на разработку функционала большого объема и требующие больших трудозатрат с

множеством условий, особенно когда они довольно замысловаты, вызовут у разработчика неподдельный интерес, тем самым мотивируя его тщательно изучить техническое задание и продумать реализацию, обеспечив максимальное качество исполнения. Как показывает практика, даже после тщательного изучения поставленной задачи, с течением времени мотивация у сотрудника теряется с каждым днем работы. В случае, когда все делается без разделения на этапы, т.е. все требования изначального технического задания реализуются единовременно, а затем изменяется несколько довольно значимых условий, потребность изменений приводит к еще большей демотивации сотрудника, у него появляется желание завершить задачу как можно скорее.

Таким образом, инициация новых изменений еще больше отсрочит дату завершения задачи. Проще потратить больше времени на общение с заказчиком, разбить задачу на короткие этапы, которые можно реализовать максимум за два рабочих дня. Это минимизирует шанс появления ошибок при условии, что интеграция всех этих частей идет постоянно. Благодаря такому подходу, у команды есть возможность всегда взять стабильный функционал данной задачи из зафиксированных точек в системе контроля версий и проще его адаптировать под новые условия.

Однако стоит помнить о том, что мелкие части, изолированные от программного средства в целом, не несут никакой бизнесс-ценности, т.к. выполняют строго атомарную операцию, которая сама по себе ничего не дает. Поэтому следует объединять такие атомарные части в группы и держать их в фокусе, особенно акцентируя внимание в момент, когда осталась последняя задача до получения результата, но менеджмент хочет изменить план. В противном случае, команду постоянно будет преследовать ощущение незавершенности, о вреде которого уже было сказано выше.

Не лишена непрерывная интеграция и недостатков, а именно:

- постоянные издержки на поддержку работы;

- необходимость вычислительных ресурсов.

Постоянные издержки на поддержку непрерывной интеграции вызваны тем, что программное средство каждый день подвергается изменениям, обусловленными гибкой методологией разработки. Это приводит к тому, что появляются новые условия или изменяются существующие принципы и требования при развертывании проекта, его тестировании. Постоянно возникает необходимость тратить время на доработку и исправление модульных тестов, а также инструкций по сборке и тестированию.

Стадии процесса запускаются автоматически, насколько это возможно, следовательно должна быть серверная машина, которая выполнит инструкции каждой из стадий. Требуется периодическая поддержка этого сервера.

Стоимость и обоснованность перечисленных недостатков непрерывной интеграции, возможные пути внедрения и результаты оцениваются каждой командой персонально.

Выводы:

1) Непрерывная интеграция защищает от срыва сроков сдачи функциональности заказчику. Она обеспечивает гарантию того, что все внесенные изменения не начнут конфликтовать между собой при поставке заказчику.

2) Непрерывная интеграция поощряет разработку большого функционала маленькими частями, шаг за шагом, что дает большую предсказуемость и лучшую изменяемость на всех стадиях разработки.

3) Непрерывная интеграция дает возможность собирать отчеты на каждой из своих стадий, обеспечивая возможность анализировать работу команды, обнаруживать проблемы в управленческих процессах и отслеживать снижение качества выполнения работы у отдельных сотрудников или всей команды.

4) Непрерывная интеграция повышает надежность программного средства, автоматизируя ручную работу по сборке и запуске тестов, подготовке и отправке отчетов о тестировании, лишая возможности скрыть факт дефекта, тем самым заставляя своевременно выделить ресурсы на его исправление.

Список литературы:

1. Блог Мартина Фаулера [Электронный ресурс]. – Электронные данные.– Режим доступа: https://www.martinfowler.com/articles/continuousIntegration.html (дата обращения: 15.07.2014)

2. Фредерик Брукс. Мифический человеко-месяц, 1975. – 82 с.

3. Booch Grady. Object Oriented Design: With Applications. – Benjamin Cummings, 1991. – 209 с.

4. Scott Chacon, Ben Straub. Pro Git, 1975. – 21 с.