НЕПРЕРЫВНАЯ ИНТЕГРАЦИЯ И ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ

два ключевых фактора разработки качественной информационной системы

epam

ВВЕДЕНИЕ

Применение методики непрерывной интеграции в рамках разработки ПО позволяет автоматизировать этапы создания, сборки и установки программного кода. Благодаря этому можно постоянно контролировать его качество, уже на ранних этапах создания ИТ-системы выявлять ошибки и быстро устранять их. Последующее функциональное тестирование обеспечит тщательную проверку готового решения и сделает ПО максимально соответствующим требованиям заказчика. В конечном итоге это гарантирует высокое качество и надежность информационной системы.

Бизнес многих современных компаний довольно сильно зависит от работы информационных систем, которые применяются для обслуживания клиентов, поддержки работы бэк-офиса, ведения управленческого учета и ряда других задач. Досадные ошибки в ИТ-приложениях или некорректное выполнение функций может привести к финансовым или репутационным потерям.

Какие шаги нужно предпринять, чтобы снизить риски появления подобного рода проблем при использовании информационных систем? Эксперты EPAM считают одним из наиболее эффективных вариантов — применить комбинированный подход: использовать методологию непрерывной интеграции (continuous integration, CI) при разработке ИТ-решения в сочетании с традиционными инструментами функционального тестирования.

1 НЕПРЕРЫВНАЯ ИНТЕГРАЦИЯ И ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ: ПОЧЕМУ ВМЕСТЕ ЭФФЕКТИВНЕЕ?

Как правило, при разработке информационной системы программный код пишут различные группы специалистов. Затем эти части собираются в более крупные модули, которые в свою очередь объединяются в систему и далее тестируются. Однако процесс сборки начинается только тогда, когда модули практически полностью готовы. Соответственно, если на этом этапе в коде обнаруживаются ошибки или возникают проблемы с взаимодействием отдельных компонентов, переписывать приходится очень много.

Напротив, практика непрерывной интеграции предполагает, что сборка кода начинает проводиться как можно раньше и повторяется как можно чаще (как минимум — при появлении каждой новой части кода). Почему такой вариант эффективнее? Здесь появляется возможность найти ошибки или противоречия в различных компонентах еще на ранних стадиях создания информационной системы. А чем раньше выявляется ошибка, тем легче и быстрее ее можно исправить. В результате снижаются риски затягивания сроков разработки решения и обеспечивается высокое качество программного кода.

После завершения разработки готовая система подвергается функциональному тестированию (автоматическому и ручному). Оно позволяет определить, соответствует ли система требованиям заказчика, и дает возможность проверить ее работу в ситуациях, максимально близких к реальным бизнес-сценариям.
Тем самым сочетание практики CI (при создании информационной системы) и функционального тестирования дает уверенность в качестве информационной системы, позволяет однозначно подтвердить, что все требования заказчика были реализованы, система им полностью отвечает и не имеет ошибок в программном коде.

2 СОЗДАНИЕ ИСХОДНОГО КОДА

В практике непрерывной интеграции можно выделить несколько основных этапов разработки ИТ-решения: создание различных частей исходного программного кода, их сборка (компиляция и компоновка), оформление в установочный пакет, установка на сервер и подготовка к тестированию.

ЭТАПЫ ПРОЦЕССА НЕПРЕРЫВНОЙ ИНТЕГРАЦИИ

Программный код будущей информационной системы может создаваться несколькими группами специалистов, причем находиться они могут в различных офисах компании-разработчика. Для повышения качества и эффективности совместной работы над кодом часто используются специальные ИТ-решения — системы контроля версий (Version Control System, VCS). К числу наиболее распространенных из них относятся open source-приложения Subversion (SVN) и Git. Подобные системы позволяют программистам составлять код, реализуя новый функционал, и при этом не мешать друг другу. Кроме того, всегда есть возможность посмотреть и проанализировать историю кода, в любой момент вернуться к предыдущему удачному варианту.

Программный код сохраняется в специальном хранилище — репозитории. При необходимости можно создавать ответвления от основного направления разработки — так называемые «ветки» или потоки (branches). С точки зрения удобства ведения проекта обычно рекомендуется открывать отдельную «ветку» для каждого нового элемента функционала, который планируется реализовать в будущей системе. После написания кода «ветка», предварительно пройдя проверку, добавляется в общее хранилище и вливается в основной поток разработки.

ЖИЗНЕННЫЙ ЦИКЛ ВЕТКИ КОДА

Чтобы получить готовое решение, нужно собрать различные части программного кода. Это можно выполнить вручную, что хорошо для небольшой программы, хотя и не всегда. Но лучше это сделать автоматически.

3 КОМПИЛЯЦИЯ И КОМПОНОВКА

Команда программистов, как правило, одновременно ведет сразу несколько потоков разработки, периодически изменяя или даже удаляя их. С точки зрения эффективности для объединения результатов работы программистов рационально применять специальный инструмент — планировщик, который позволит создавать, хранить и автоматически запускать задачи по сборке программного кода. В качестве примера планировщиков можно привести решения CruiseControl, TeamCity, Hudson (и его наследник Jenkins) и другие.
В рамках выполнения каждой такого рода задачи планировщик с заданной периодичностью (например, по правилу «появление нового изменения» или в строго определенное время) получает части кода из нужных веток репозитория и запускает их сборку так, как это было предварительного сконфигурировано билд-инженером. Чаще всего такие задачи однотипны, так что нет ничего сложного в том, чтобы их автоматически создавать по определенному шаблону.

Во время сборки кода (компиляции и компоновки) параллельно может быть запущено и юнит-тестирование (unit tests): вызов части кода с определенными параметрами, который должен привести к строго обозначенному результату. Это позволит проверить код еще до его включения в конечную программу (систему), посмотреть, вызывают ли тесты все строки кода, проверить его на логические ошибки и т.д. Тем самым появляется возможность выявить дефекты, которые есть в исходном коде и которые возникают при взаимодействии компонентов программы, а также проверить, как были исправлены найденные ошибки.

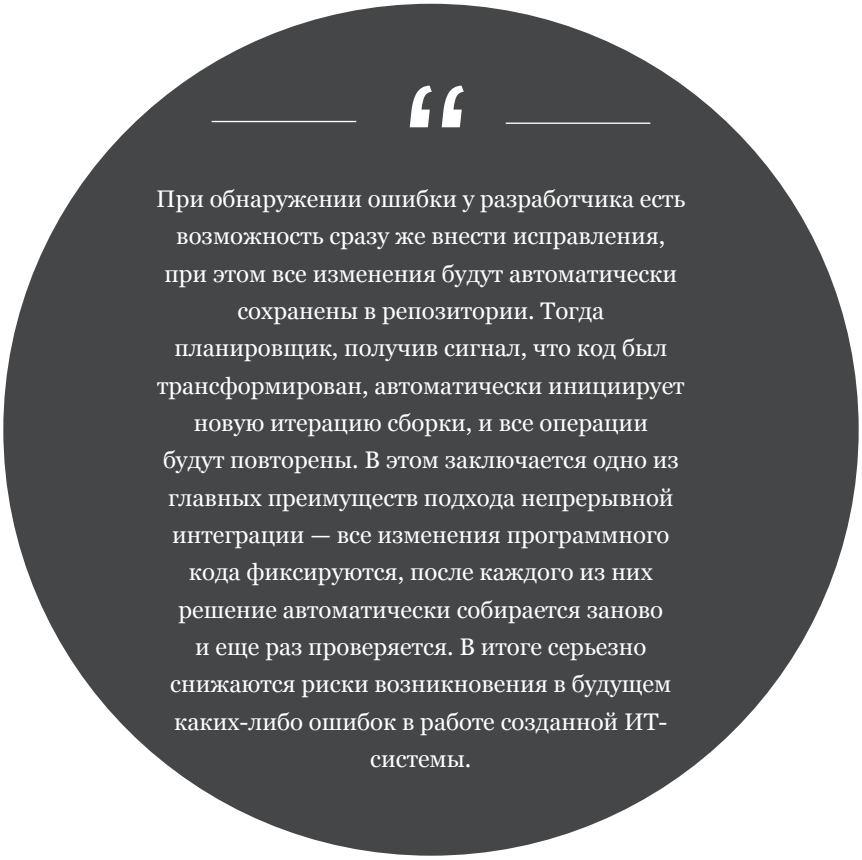
Результаты такого предварительного тестирования могут быть автоматически обработаны и представлены участникам проекта в специальной системе, позволяющей просмотреть итоги тестирования частей кода в единообразном виде.

При обнаружении ошибки у разработчика есть возможность сразу же внести исправления, при этом все изменения будут автоматически сохранены в репозитории. Тогда планировщик, получив сигнал, что код был трансформирован, автоматически инициирует новую итерацию сборки, и все операции будут повторены. В этом заключается одно из главных преимуществ подхода непрерывной интеграции — все изменения программного кода фиксируются, после каждого из них решение автоматически собирается заново и еще раз проверяется. В итоге серьезно снижаются риски возникновения в будущем каких-либо ошибок в работе созданной ИТ-системы.
Кроме того, это позволяет избежать задержек в работе в том случае, если в проекте участвуют команды, которые находятся в различных регионах и часовых поясах и работают посменно. Без использования CI программист, написав свою часть кода и не получив подтверждения (нотификации) по электронной почте об отсутствии ошибок, спокойно уходит домой. Но его коллеги, которые в этот момент в другом регионе только начинают свой рабочий день, будут вынуждены потратить свое время на выявление дефектов и их исправление. Сроки разработки системы увеличатся. В случае когда разработка ведется в рамках непрерывной интеграции, подобные ситуации исключены.

На некоторых проектах иногда дополнительно применяется так называемый «анализ кода» (code review). Его смысл заключается в том, что программист, закончив разработку части функционала, которую он хочет включить в общий поток разработки, сначала передает ее для анализа своему коллеге, который специализируется в той же или смежной области. Такой взгляд со стороны зачастую позволяет быстро найти потенциальные проблемы, а иногда и способы их решения. Нелишним может быть и применение инструментов статистического анализа кода (PMD, FindBugs) — они помогают автоматически выявить стандартные и наиболее распространенные ошибки, которые могут допускаться разработчиками.

Кроме того, на данном этапе проверяется стилистика кода: его соответствие тем правилам оформления и формату, которые действуют на проекте. Это позволяет сделать код «читабельным», что значительно облегчает дальнейшую работу с ним. Подобное тестирование проводится в автоматическом режиме (например, с использованием программы Checkstyle). Частично ошибки исправляются также автоматически, но иногда это приходится делать вручную (определяются конкретные строки кода, где были сделаны ошибки, и дальше вносятся исправления). Стоит заметить, что провести подобную проверку можно только в рамках практики continuous integration. В других случаях, когда тестированию подвергаются уже готовые системы, QA-специалисты фокусируются на приведение кода в соответствие с требованиями заказчика и не смотрят, написан ли он в едином стиле или нет.

В результате после окончания этапа у проектной команды появляется готовая программа, которая по своей структуре, покрытию юнит-тестами, документированности и другим параметрам полностью отвечает ключевым стандартам качества.

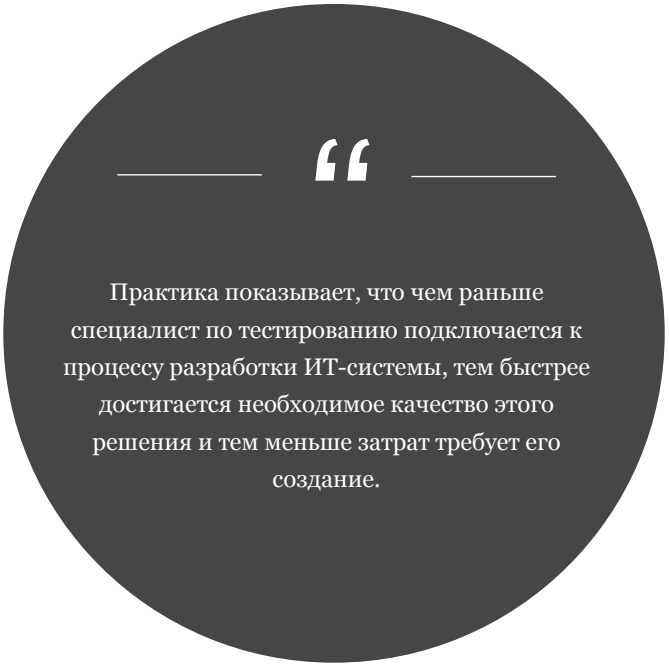


4 ПОДГОТОВКА УСТАНОВОЧНОГО ПАКЕТА

Скомпилированный код должен быть запакован соответствующим образом: это может быть zip-архив, установочная программа Windows или пакет для Unix/Linux дистрибутива. Для хранения готового пакета чаще всего используется файл-сервер (FTP, NFS, общая папка в сети и т.д.), который может быть как локальным (размещен в компании-разработчике системы), так и удаленным (например, располагаться у заказчика).

При создании или помещении на хранение пакет, содержащий готовую программу, может быть подписан цифровым сертификатом, либо авторизован каким-либо другим способом. Дополнительно можно добавить сведения о размере, указать контрольную сумму или задать какие-либо другие отметки, которые помогут отследить, не был ли поврежден пакет при передаче в репозиторий на файл-сервер или в результате воздействия вирусной программы. Для удобства процесс подписи также может быть автоматизирован — такие возможности есть у тех же решений-планировщиков, с помощью которых запускаются задачи по сборке кода. Другой вариант — использование штатного планировщика операционной системы.

В любом случае лучшей практикой будет хранение сразу нескольких версий готовой программы или ее модулей (в этом случае обычно номер версии включается в название пакета или в название папки на файл-сервере, где хранится этот пакет).



5 АВТОМАТИЧЕСКАЯ УСТАНОВКА И ПОДГОТОВКА К ТЕСТИРОВАНИЮ

Хорошим тоном при разработке программы считается предусмотреть в ней вариант так называемой «бесшумной» (silent) установки, когда система ставится на компьютер или сервер, не требуя каких-либо действий от ИТ-специалистов (без всплывающих окон и запросов). При этом она может использовать как заданные разработчиком параметры «по умолчанию», так и файл, предоставленный заказчиком, с аналогичными параметрами, но уже специально подобранными под конкретную ИТ-среду компании. Штатные пакеты для Unix/Linux- систем обладают такой возможностью по определению. Все это позволяет автоматизировать установку решений, снизить влияние человеческого фактора на этот процесс и существенно сэкономить время.

Для автоматической установки готового решения на сервер или группу серверов, где затем будет проводиться его тестирование, может также использоваться планировщик. Он отследит факт помещения решения в хранилище и запустит процесс его установки. В последнее время в связи с широкой популярностью виртуализации и вычислительных «облаков» распространена практика, когда программа ставится на специально созданную виртуальную машину. И, что важно, этот виртуальный сервер тоже может быть создан автоматически — с помощью планировщика. Например, этот процесс можно привязать к запуску задачи по сборке кода, который регулируется планировщиком. Тогда от попадания программы в репозиторий до ее установки на сервер для последующего тестирования будут проходить считанные минуты.

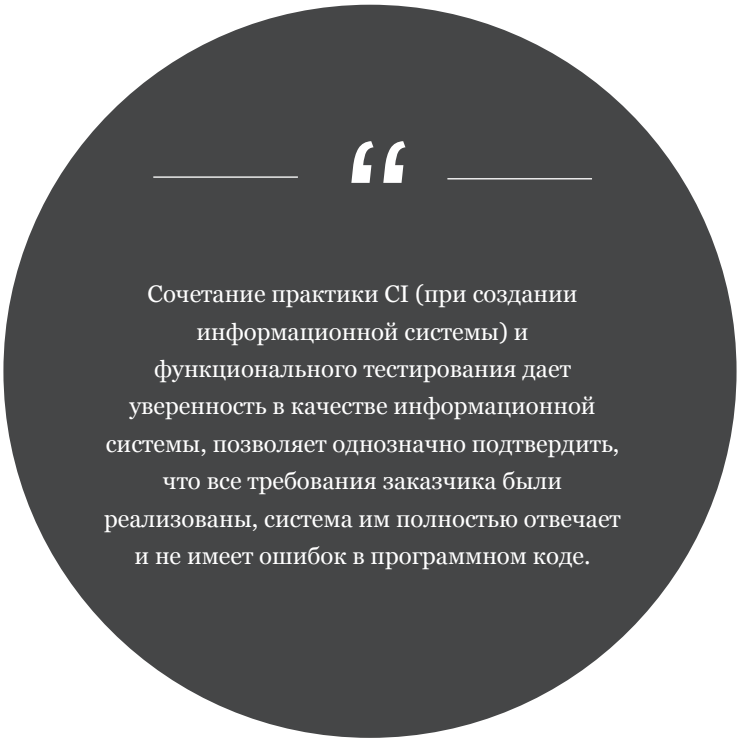
Отдельно стоит упомянуть об одном из перспективных направлений развития практики continuous integration, когда весь процесс полностью — от написания кода до запуска готового решения в эксплуатацию — выстраивается как «облачный» сервис. Заказчик обозначает свои требования — и после завершения проекта получает готовую систему. Все вопросы, связанные со сроками и рисками разработки и поддержки решения, выпуска релизов, регулируются с помощью заключения SLA (Service Level Agreement — соглашение об уровне услуг) между исполнителем и заказчиком. К примеру, SLA помогает снизить зависимость успешного проведения разработки и тестирования от конкретных специалистов, которые управляют процессом непрерывной интеграции. Даже в случае их временного отсутствия на проекте будет обеспечено выполнение всех запланированных операция точно в установленные сроки.

Итак, программный код скомпилирован, запакован и установлен. Что дальше? Функциональное тестирование.

6 ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ: АНАЛИЗ ТРЕБОВАНИЙ

Проведение функционального тестирования может включать в себя несколько этапов: анализ требований, которые были выдвинуты заказчиком к информационной системе, разработка функциональных тест-кейсов, проведение автоматизированного и ручного функционального тестирования, анализ результатов тестирования.
Практика показывает, что чем раньше специалист по тестированию подключается к процессу разработки ИТ-системы, тем быстрее достигается необходимое качество этого решения и тем меньше затрат требует его создание. Еще перед началом разработки команда тестирования анализирует требования, которые были сформулированы заказчиком: выявляются противоречия, уточняются неясные моменты, добавляются отсутствующие, но явно необходимые требования. Как правило, при этом тестировщики работают в тесном взаимодействии с бизнес-аналитиками и разработчиками, задействованными в проекте. Кроме того, на этом же этапе оценивается тестируемость требований (можно ли в принципе будет проверить их выполнение), что немаловажно для успешного проведения последующих этапов.

Стоит отметить, что в зависимости от выбранной методологии реализации проекта анализ требований может быть проведен в строго определенный срок (это характерно для тех случаев, когда разработка ведется по водопадной (каскадной) модели) или продолжаться на протяжении всего проекта (если выбрана Agile-методология или итеративная разработка). В любом случае тестирование требований позволяет минимизировать риски на ранней стадии проекта и, как следствие, снизить издержки на устранение ошибок.



7 РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТЕСТ-КЕЙСОВ И АВТОМАТИЗИРОВАННОЕ ТЕСТИРОВАНИЕ

Параллельно с оценкой требований начинается разработка функциональных тест-кейсов — они описывают, какие действия или операции будут выполняться при проверке системы и какие результаты должны быть получены.
Степень детализации тест-кейсов зависит от жизненного цикла и потребностей проекта. На данном этапе часто привлекаются бизнес-аналитики: их участие помогает обеспечить более точное и полное покрытие тестами различных сценариев будущего использования информационной системы.

После того, как готовый программный код был собран и установлен на сервер, начинается следующий этап функционального тестирования — запуск автоматизированных тестов. В отличие от юнит тестов, которые применялись после сборки кода для проверки его работоспособности, эти тесты используются для автоматической проверки функционала и интерфейсов ИТ-системы.

Настройкой авто-тестов занимаются инженеры по автоматическому тестированию. Начинать процесс тестирования можно либо в ручном режиме, либо автоматически (в этом случае билд-инженер — специалист, который руководит процессом непрерывной интеграции, — настраивает процесс таким образом, чтобы проверка запускалась сразу после установки программы на сервер).

В дальнейшем результаты автоматизированных тестов анализируются, определяются дефекты, которые привели к получению отрицательных результатов. Если в проекте используется гибкая модель разработки программного обеспечения, то найденные ошибки немедленно устраняются, и процесс тестирования запускается снова. В иных случаях дефекты регистрируются в системе для фиксации ошибок (bag tracking system) и исправляются после завершения всего тестирования.

8 РУЧНОЕ ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ

Объем ручного тестирования во многом определяется тем, с какой степенью детализации и полноты автоматические тесты охватили функциональность создаваемой ИТ-системы. Если они были проработаны недостаточно детально, то сначала в ручном режиме проводится проверка работоспособности всех компонентов решения — с помощью «тестирования на дым» (Smoke-тесты). Затем найденные дефекты исправляются и проверяются еще раз — на предмет соответствия тем требованиям, которые были сформулированы изначально. По разработанным функциональным тест-кейсам прогоняются основные сценарии использования будущей системы.

Однако при наличии достаточно полных и подробных автоматизированных тестов эти этапы, как правило, опускаются. В такой ситуации задачу функционального тестирования можно обозначить так: с максимальной широтой охватить возможные бизнес-сценарии использования программы, а также протестировать логику взаимодействия продукта со сторонними компонентами.

> “
>
> Данные, полученные по итогам автоматизированного и ручного тестирования, обрабатываются и анализируются. Затем принимается решение либо о необходимости доработки программного кода (и тогда процесс разработки и непрерывной интеграции запускается снова), либо о передаче ИТ-системы заказчику.

9 ПОДВЕДЕНИЕ ИТОГОВ

На финальном этапе — после завершения разработки и тестирования информационной системы — возникает необходимость удалить «остатки производства»: убрать из планировщика задачи по сборке, стереть ненужные установочные пакеты, освободить уже неэксплуатируемую виртуальную машину и т.д. Для удобства и снижения затрат эти операции могут выполняться автоматически.

Стоит отметить, что одно из достоинств использования практики непрерывной интеграции заключается в том, что на протяжении всего процесса разработки на одном из серверов всегда присутствует самая свежая рабочая версия создаваемой системы. Причем, как показывает практический опыт, это программный код такого качества, работу которого можно в любой момент времени продемонстрировать заказчику (так называемый production quality code). В результате есть возможность показать готовые или готовящиеся компоненты ИТ-приложения клиенту и получить от него обратную связь — отзывы, которые помогут быстро внести в код необходимые изменения.

При применении методики непрерывной интеграции (в рамках разработки программного обеспечения) в сочетании с традиционными инструментами функционального тестирования появляется возможность автоматизировать этапы сборки, установки и тестирования программного кода. Это помогает постоянно контролировать качество кода, уже на ранних этапах создания ИТ-системы выявлять ошибки и быстро устранять их. Глубокая проработка функциональных требований, разработка детальных и полных тест-кейсов, использование различных средств ручного и автоматического тестирования обеспечивает тщательную проверку готового решения.

В конечном итоге это позволяет обеспечить максимально высокое качество и надежность информационной системы и ее соответствие ожиданиям заказчика.

Основанная в 1993 году компания EPAM Systems (NYSE: EPAM) специализируется на разработке программных решений. Штаб-квартира компании расположена в США. EPAM работает с клиентами по всему миру, используя свои отмеченные наградами центры разработки и офисы в 19 странах Северной Америки, Европы, Азии, Австралии. EPAM помогает клиентам повышать эффективность бизнеса, предлагая услуги по цифровому преображению и бизнес-трансформации, созданию умного предприятия и внедрению передовых технологий. По мнению ведущего независимого аналитического агентства, EPAM является лидером в сфере разработке программных решений, входит в Топ 10 поставщиков услуг для торговли и в число ключевых игроков на рынке Agile в мире. По данным издательского дома «Коммерсант» EPAM Systems возглавляет список крупнейших разработчиков программного обеспечения в России.

Более подробная информация доступна на www.epam-group.ru

РОССИЯ И СНГ

ул. 9-ая Радиальная, д.2
Москва, Россия, 115404

Тел.: +7-495-730-63-62
Факс: +7-495-730-63-61

ЕВРОПА

Corvin Offices I. Futó street 47-53
Budapest, H-1082 Hungary

Тел: +36-1-327-7400
Факс: +36-1-577-2384

США

41 University Drive Suite 202
Newtown, PA, USA 18940

Тел: +1-267-759-9000
Факс: +1-267-759-8989