DOI 10.37882/2223-2966.2024.8.37

АЛГОРИТМ ВНЕДРЕНИЯ АВТОМАТИЗИРОВАННОГО ТЕСТИРОВАНИЯ ПО И ПЕРЕХОДУ К CONTINUOUS DEPLOYMENT

ALGORITHM FOR IMPLEMENTING AUTOMATED SOFTWARE TESTING AND TRANSITION TO CONTINUOUS DEPLOYMENT

A. Chumakova

Summary. The article presents a practical algorithm for the implementation of automated software testing in a project and the subsequent transition to a continuous release cycle of Continuous Deployment. The author notes that the integration of information technologies creates the basis for the effective operation of companies and directly affects their level of success in the market. Special attention is paid to the need for property organization of testing on the project to improve the quality of the software being developed and, accordingly, customer loyalty. The article discusses the practical aspects of the implementation of automated testing in the project and provides a detailed methodology for the transition to a continuous cycle of deployment — Continuous Deployment. The main problems that company face when implementing various information system testing technologies are also noted.

Keywords: information technology, software testing, test automation, Continuous Deployment, modern technologies.

Чумакова Анна Александровна
Ведущий QA инженер, ООО «ГК «Иннотех»
anna-a-chumakova@yandex.ru

Аннотация. В статье представлен практический алгоритм по внедрению автоматизированного тестирования программного обеспечения в проект и последующий переход к непрерывному релизному циклу разработки Continuous Deployment. Автор отмечает, что интеграция информационных технологий создает основу для эффективной работы компаний и непосредственным образом влияет на уровень их успешности на рынке. Особое внимание уделяется необходимости правильной организации тестирования на проекте для повышения качества разрабатываемого программного обеспечения и, соответственно, лояльности клиентов. В статье рассматриваются практические аспекты внедрения автоматизированного тестирования в проекте, а также дается детальная методология по переходу к непрерывному циклу развертывания сборок в промышленную эксплуатацию — Continuous Deployment. Также отмечаются основные проблемы, с которыми сталкиваются компании при внедрении различных технологий тестирования информационных систем.

Ключевые слова: информационные технологии, тестирование программного обеспечения, автоматизация тестирования, Continuous Deployment, современные технологии.

Введение

Интеграция информационных технологий в различные аспекты человеческой деятельности создаёт предпосылки для формирования новых условий функционирования рынка, по этому компании, занимающиеся разработкой и внедрением программного обеспечения (ПО), очень востребованы в современном мире. В условиях рыночной экономики для большинства таких компаний ключевое значение приобретают повышение качества выпускаемого продукта и минимизация затрат на его разработку. Для достижения данных целей используется множество различных инструментов и технологий, самый популярный из которых является автоматизация тестирования разработанного продукта, несущий в себе неоспоримые плюсы: увеличение скорости и качества проверок за счет отсутствия человеческого фактора, широкое покрытие функционала, а также экономию бюджета за счет сокращения трудозатрат и времени на ручное тестирование. Но также имеются и минусы: актуализация авто-тестов требует больше трудозатрат, чем для ручного тестирования; авто-тесты проверяют только то, что заложено в коде, пропуская другие детали; риск необходимости частого обновления авто-тестов при наличии большого количества нового функционала; дороговизна внедрения; невозможность автоматизации тестирования некоторых тест-кейсов.

Таким образом, вопрос, связанный с оправданностью внедрения автоматизированного тестирования программного обеспечения, остается открытым и дискуссионным, что обуславливает актуальность данной темы.

Алгоритм внедрения автоматизированного тестирования ПО и переходу к Continuous Deployment

Принимать решение о необходимости автоматизации тестирования необходимо на этапе старта проекта, но для этого нужно провести предварительный анализ. Обозначим критерии проекта, требующего автоматизации тестирования:

1. Регресс требует большого количества трудозатрат и времени;
2. Нет возможности отказаться от поддержки старых версий ПО;
3. Большой и долгий проект;
4. В ручных проверках прослеживается влияние человеческого фактора.

Если проекту характерны эти признаки, то, однозначно, автоматизация тестирования в нем оправдана и целесообразно использовать алгоритм внедрения процесса, описанный в данной статье.

На первый взгляд сделать это достаточно легко: после того как требования сформулированы, и разработка приступила к написанию кода, ручные тестировщики (Manual QA) готовят тестовую модель, а автоматизаторы (Automation QA) переносят часть тест кейсов во фреймворк. При этом компании хотят достичь максимального покрытия функционала авто-тестами, но даже при высоком уровне его достижения, в таком формате тесты остаются не стабильными и трудно поддерживаемыми, качество покрытия кода зачастую недостаточное. В итоге релизный цикл становится тяжело управляемым и неповоротливым. Сборки, которые, согласно scrum-методологии, выходят один раз в две недели, покрываются ручными регрессионными проверками и нестабильными авто-тестами, что в итоге создает хаос в проекте и порождает дефекты в среде промышленной эксплуатации (ПРОМ). Подобная практика наблюдается во всех больших корпорациях и является наиболее актуальной проблемой в сфере тестирования ПО.

Таким образом, можно выделить следующие проблемы, характерные для большинства крупных проектов, решив которые можно приблизиться к Continuous Deployment:

1. Медленный и сложный ручной регресс;
2. Отсутствие взаимодействия Automation QA с командой проекта;
3. Реальное тестирование возложено только на Manual QA;
4. Большое количество бесполезных, не стабильных сложно поддерживаемых авто-тестов, которые пишутся медленно и не помогают выявить дефекты.

Следующий алгоритм внедрения автоматизированного тестирования в проект позволяет избежать вышеописанных проблем:

1. Необходимо выстроить удобную систему отчетов и запусков авто-тестов;
2. Создать и запустить алгоритм автоматического запуска тестов на каждый коммит (в идеале, каждая часть функционала, каждая задача и история должны быть проверены авто-тестами);
3. Организовать слаженную работу всей команды — вовлечь Manual QA, Automation QA и разработчиков в тестирование (что бы каждый член команды умел тестировать тот функционал, за который он ответственен);
4. Сделать авто-тесты стабильными, быстрыми и простыми в их написании и поддержке.

Рассмотрим каждый шаг подробнее.

Шаг 1. Внедрение системы запуска авто-тестов и отчетности

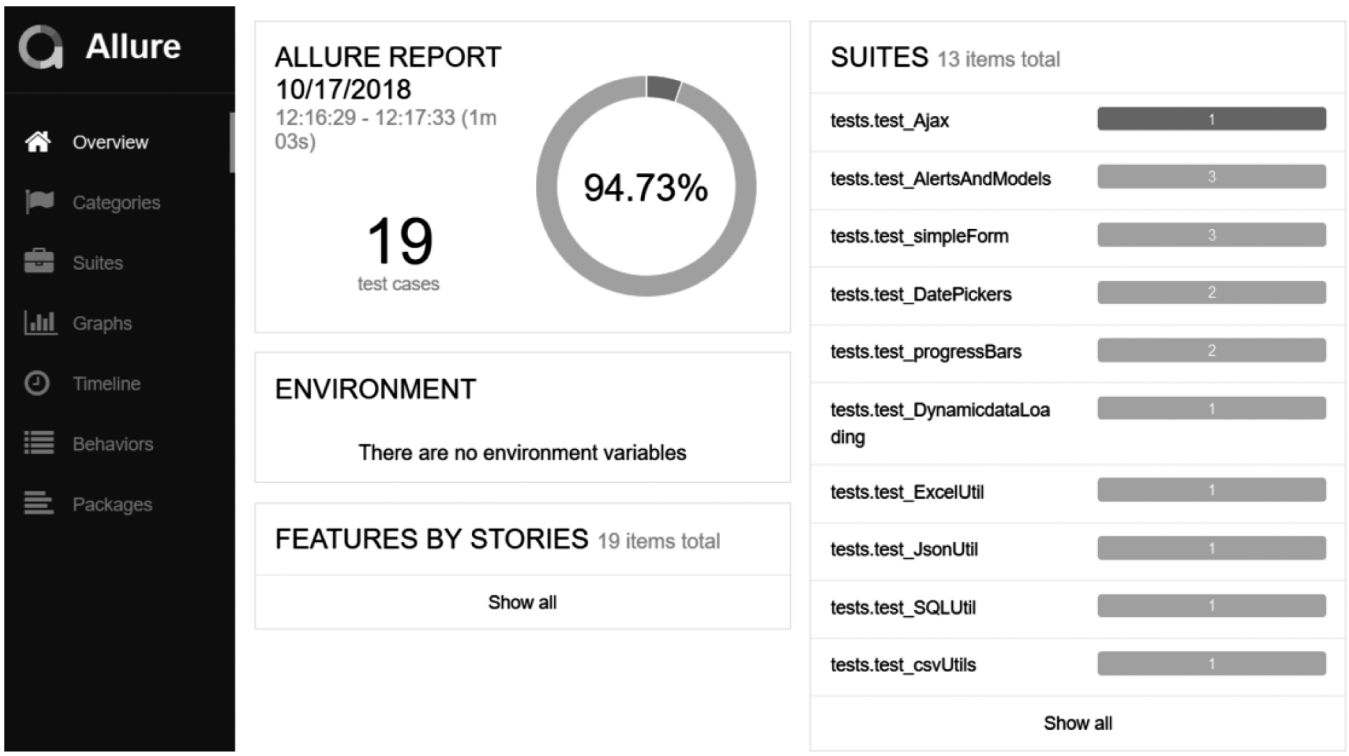

Рис. 1. Внешний вид системы отчетности Allure TestOps

Источник: официальный сайт системы Allure https://qameta.io

В первую очередь необходимо внедрить систему отчетов и запуска авто-тестов, например Allure TestOps, в которой можно хранить тест кейсы, запускать тест планы для разных сервисов и просматривать все прогоны в режиме онлайн:

В авто-тестах желательно добавить описания шагов что бы любой человек, даже далекий от кода смог понять, что там происходит, организовав таким образом максимальную прозрачность.

Для отслеживания падений авто-тестов добавить понятные сообщения об ошибках что бы сразу же можно было заводить дефекты и разбирать прогоны, не тратя лишнее время и не привлекая разработку на локализацию проблем.

Шаг 2. Каждый коммит обязательно должен быть проверен

Все должно выглядеть так: когда разработчик мержит свою задачу в мастер ветку, которая в последствии должна оказаться в ПРОМе, в первую очередь она должна установиться на тестовый стенд, быть проверена, а результаты отобразиться на дашборде (Messenger channel).

Схема выглядит следующим образом (см. рис. 2).

На схеме можно увидеть, что сервисы находятся в GitLab, и при мерже какого-либо из них запускается GitLab CI Pipline который отдает всю информацию (о сервисе, о комите, об авторе этого коммита и другую) в «Autorun job», которая в свою очередь запускает джобу деплоя (Deploy job), а она в свою очередь результат обратно, и если он не успешный, то посылается сообщение в мессенджер и весь алгоритм останавливается. Если же все было хорошо и сервис успешно задеплоился, то происходит запуск авто-тестов в «Autotests run job», ее результаты отправляются в определенный канал в маттер мосте и так же передаются на дашборд (Messenger channel).

Шаг 3. Модификация команды

В типичном общераспространенном типе устройства команд каждый ее член автономно занимается своей функцией, коммуникации сведены к минимуму. Этот подход абсолютно не верен.

После внедрения новой системы отчетов, описанной на шагах выше, роли в команде изменяются: разработчики присоединяются к тестированию продукта, ручные тестировщики тестируют и передают свои знания, автотестеры помогают в регрессе.

Стоит подробнее отметить каким образом разработчики могут помочь в тестировании. У них так же есть тесты: unit тесты, которые могут сильно помочь в автоматической проверке функционала т. к. запускаются раньше, чем авто-тесты, они стабильнее и быстрее, но есть необходимость в их контроле и анализе их качества. Чаще всего покрытие unit-тестов очень ограничено и покрывает только максимально критичные вещи.

Таким образом, для увеличения качества покрытия функционала тестами можно выделить следующие пути и решения:

1. Совместное обсуждение функционала аналитиками, разработчиками и тестировщиками для лучшего понимания возможностей тестирования;
2. «Парный тест-дизайн» — встреча разработчиков (frontend и backend) и тестировщиков (Manual QA и Automation QA) по обсуждению кейсов по каждой задаче, их дополнение, исправление и распределение для исключения возможной повторной проверки и избыточности. На этом этапе,

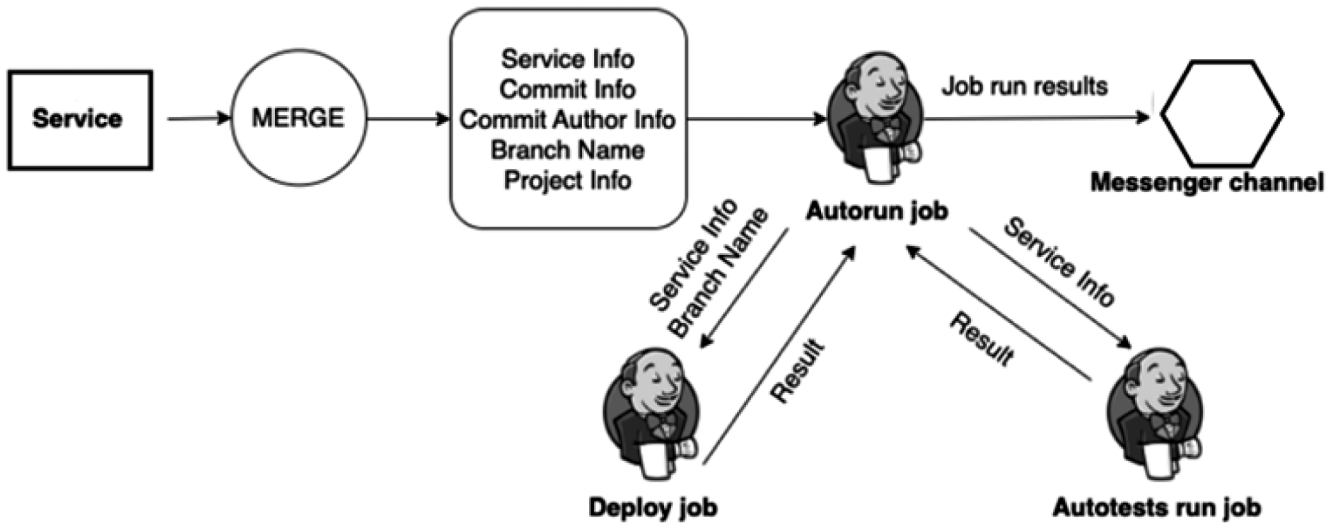

Рис. 2. Схема внедрения авто-тестирования в релизный цикл

перед совместным обсуждением, тестировщик предварительно описывает тест-кейсы, после чего происходит их распределение;
3. Ревью тест кейсов друг друга тестировщиками на предмет полноты проверок, и аналитиками на предмет правильности понимания работоспособности функционала;
4. Распределение тест кейсов и проверок по ролям: фронт разработчикам, бэк разработчикам, Manual QA, Automation QA для исключения дублей проверок.

Окончательный отказ от ручного регресса возможен в следующей схеме релизного цикла (см. рис. 3).

В схеме отражено:

Получение требований — первоначальный этап в процессе разработки нового функционала, которым занимается бизнес аналитик или системный аналитик, после завершения которого, готовые требования передаются команде в виде задач или других описаний.

Груминг — разбор задачи — еженедельный созвон, где разбираются задачи на спринт и прописываются заметки для всех членов команды.

Написание тех дизайна — проектирование пользовательских веб-интерфейсов для интернет-ресурсов.

Парный тест дизайн — как уже было описано выше, это встреча разработчиков (frontend и backend) и тестировщиков (Manual QA и Automation QA) по обсуждению кейсов, их дополнение, исправление и распределение между членами команды.

Разработка — непосредственное написание кода.

Проверка задачи unit тестами разработчиков и, при необходимости, написание новых unit тестов.

Merge функционала — это перенос кода программы из одной ветки в другую.

Проведение регрессионного тестирования авто-тестами.

Проверка задачи ручными тестировщиками.

Релиз в production — установка нового функционала в промышленную эксплуатацию.

Таким образом Quality Gates (автоматические проверки качества, которые устанавливают пороговые значения для продвижения продукта по конвейеру разработки) выглядят следующим образом (см. рис. 4).

Первым этапом к задаче формируются актуальные правильные требования, при необходимости проводится тех дизайн, парный тест-дизайн и описываются основные тест-кейсы, после чего она может быть передана в разработку. Перед мержем задачи проводится обязательное код-ревью, проходятся unit-тесты разработчиков. Далее запускаются авто-тесты регресса (при необходимости пишутся новые авто-тесты), а также проводится ручное тестирование нового функционала, после чего сборка устанавливается в ПРОМ.

Рис. 3. Схема релизного цикла, исключающая проведение ручного регресса

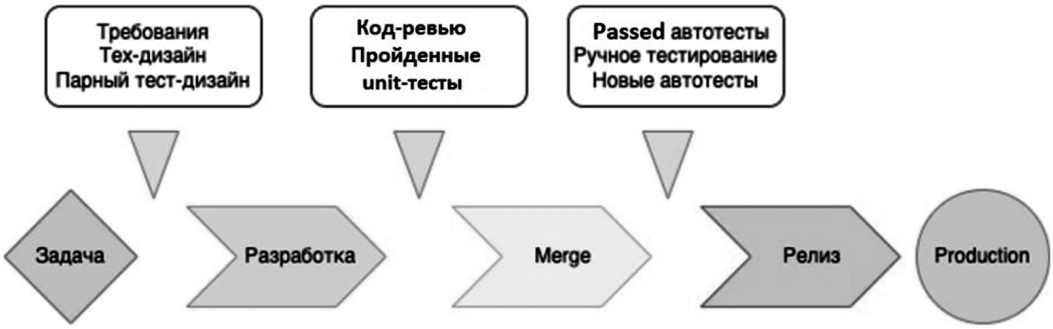

Рис. 4. Quality Gates в релизном цикле

Таким образом, Continuous Deployment получается двухэтапным: merge и deploy, что является идеальным вариантом релизного цикла для больших проектов.

Шаг 4. Сделать авто-тесты стабильными, быстрыми и простыми в их написании и поддержке

Далее рассмотрим, какие тесты и в каком распределении должны быть на проекте.

Рис. 5. Пирамида оптимального распределение тестовых проверок

На рисунке изображено правильное пирамидное тестирование, которое в основании имеет unit тесты: как UI, так и backend, где распределение может быть иным в зависимости от специфики функционала. В середине пирамиды имеются скриншотные UI-тесты и мок-интеграционные тесты бэкэндеров, и закрывает это все end-to-end (E2E) авто-тесты.

Итого, при внедрении в проект всех вышеперечисленных рекомендаций можно получить:

- Отсутствие ручного регресса и налаженные автоматические процессы релиза;
- Сплоченную команду и вовлеченных в процессы сотрудников;
- Тестирование системы всей командой, а не только Manual QA;
- Качественные, стабильные, легко поддерживаемые авто-тесты, которые являются универсальным инструментом тестирования.

При этом важно правильно внедрять авто-тесты и, что бы они приносили плоды и действительно помогали в работе, а не отнимали ресурсы команды, необходимо своевременно решать возникшие проблем и трудности. Следование рекомендациям из этой статьи гарантирует, что в проекте процесс авто-тестирования будет внедрен надлежащим образом, будет прозрачным и полезным.

ЛИТЕРАТУРА

1. Ефимов С.Н. Проектирование вычислительной сети эффективной архитектуры для распределенного решения сложных задач / С.Н. Ефимов, В.В. Тынченко, В.С. Тынченко; под ред. проф. Г.П. Белякова // Вестник Сибирского государственного аэрокосмического университета: сб. науч. тр. — Красноярск: СибГАУ. — 2007. — Вып. 3(16). — С. 36–41.
2. Попов И.И. Компьютерные сети / И.И. Попов, Н.В. Максимов. — М.: Форум, 2004. — 326 с.
3. Datasheet HOKUYO URG [Electronic resource]. — Access mode: https://www.hokuyo-aut.jp/dl/ Specifications_URG-04LX_1513063395.pdf.

Журнал «Современная наука: актуальные проблемы теории и практики»