Когда нужны автотесты

(те у кого есть ссылка - можете комментировать).

Нужно ли автотестирование ? Когда оно нужно ? Какую ценность оно приносит?
В статье разобраны когда и зачем нужно тестирование как таковое и в каких случаях нужна его автоматизация.

В ходе дискуссии по этому вопросу, проводимой в клубе им. Френсиса Бекона (“ВебПосиделки КиФБ” в teleram), коллеги обменялись опытом и записали свои мысли.

Начнем с того, что автоматизированное тестирование не отменяет тестирование постановки, тестирование дизайна и написания тест кейсов по постановке! И не заменяет их. Если этого нет, то и автоматизацией не следует заниматься. При этом под автотестами следует понимать не только само выполнение сценария, но и подготовку к их прогону и использование результатов.

Автоматизация тестирования нужна, если это приносит ценность. Когда же само тестирование приносит ценность? Выявлено два случая.

1. Если процесс отлавливает ошибки в ПО перед выкладкой на бой.

Если тестирование новой версии выявило ошибки, которые были в последующем исправлены, то это тестирование было проведено не зря.

Но что если ситуация обратная? Если тестирование не нашло ошибки, но они проявились на бою? Если ошибки при этом были обнаружимы на тестовом стенде (в т.ч. тестовый стенд настроен наиболее близко к бою).
Утверждается, что в этом случае тестирование проведено некачественно.
Как измерять качество тестирования? Для этого случая пригодна метрика = количество ошибок на тестовом окружении / (количество ошибок на тестовом + боевом окружении). При этом количество ошибок берется взвешенное по уровню их критичности.

А что если тестирование не нашло ошибок и их не было обнаружено на боевом сервере?
Утверждается, что этом случае тестирование, как таковое, не принесло ценности и эти работы были сделаны зря (за исключением следующего случая, про который мы расскажем позже). С точки зрения Lean это потери.

Когда возможна такая ситуация ? Когда тестируемый модуль не изменялся. Что может изменить модуль?

- Изменение кода модуля.

- Изменение версии используемых библиотек (включая ОС, БД и пр). Изменение может быть обусловлено требованиями регуляторов, из-за чего библиотеку требуется обновить в жестко заданные сроки.
- Изменение настроек или данных, серьезно влияющих на поведение универсального функционала, всестороннее тестирование которого провести избыточно трудоемко тяжело и от тестирования в итоге отказались. Примеры:
    - Настройки промоакций в решениях типа Siebel CRM или Oracle ATG может привести к деградации производительности функционала расчета промо, а возможность всесторонней проверки невозможна в разумные сроки из-за излишней гибкости и универсальности этих решений
    - html описание карточки товара может содержать нарушенную структуру документа или ошибки в написанном внутри описании js-кода, что ломает страницу карточки товара
- Применение функционала не предназначенным для этого способом (микроскопом гвозди забивать). Например, изменение типа нагрузки, не заложенное в требованиях и как следствие не учтенное при проведенном тестировании. К примеру, перед черной пятницей стоит провести отдельное нагрузочное тестирование на лендинг, куда будет идти трафик, если к этому типу страниц не было таких требований по нагрузке.
- Изменение поведения API других модулей, который использует разрабатываемый модуль. Особенно если функционал API не покрыт регрессионным тестированием.

Так как варианты изменений могут быть разные, а проведение полного регрессионного тестирования стоит денег, то стоит проводить не все тесты. Один из вариантов управления состоит в разметке тестов тегами и перед тестированием. Тест менеджер определяет какой набор тестов стоит направить на выполнение для заданной порции изменений.

Когда нужно писать автотесты в этом случае?
Если их писать после создания кода, то это приведет к увеличению time2market (что автоматически приведет к увеличению связного капитала). Поэтому если принято решение покрывать код автотестами, то следует писать эти автотесты параллельно основному коду, в парадигме разработки “TestFirst” или “TDD”.
Основная ценность создаваемая при этом автоматизацией тестирования - это сокращение time2market за счет более быстрой выкладки новой версии.

2. Для гарантии работоспособности критических процессов

Несмотря на то, что автомобиль ни разу не загорелся, тем не менее наличие огнетушителя в нем не бесполезно.
Ошибка на сайте с большим трафиком, не позволяющая добавлять товар в корзину, может привести к более значимым потерям, чем расходы на разработку и проведение тестирования этой функции за год.

Следовательно, нужно выделить критические процессы, которые войдут в регулярные проверки (которые стоит делать если происходит какое-то изменение). Сопоставлять следует:

- потери от простоя функции за время от обнаружения до его исправления,
- расходы ручное регулярное тестирование или на его автоматизацию и последующее его проведение в течении периода окупаемости.

А что если регулярное тестирование не находит ошибок в течении долгого времени и их не возникает на бою? Тогда тестирование не приносит ценности, а следовательно оно не нужно. Когда это возможно?

- Разрабатываемый модуль не очень большой.
- Стабильная высоко компетентная команда.
- Интеграции закрыты тестами либо на той стороне не происходит изменений.

Возможно ли не делать тестирование вообще?
Это возможно через запуск нескольких инсталляций решения и тестирования новых beta-версий на “кошечках”, если это технически возможно и если найдутся такие добровольцы. После выкладки новой версии мониторится телеметрия и производится откат, в случае деградации показателей. (Напомним, что телеметрия на бою обязана быть вне зависимости от наличия тестирования).

Еще один случай полезности регрессионного автотестирования - это тестирование API (тестирование контракта API), если это API требуется для обеспечения критического процесса. Особенно это важно, если разработчики чужого модуля что-то меняют и не делают качественное тестирование изменений на своей стороне.

Когда не нужна автоматизация тестов

Если вам достался большой объем унаследованного не очень качественного кода. Покрывать автотестами такой хаос это увеличивать хаос.

В этом случае стоит выделить из этого решения логический модуль. После выделения слоя взаимодействия этого модуля с остальным кодом нужно покрыть взаимодействие автотестами. Это обеспечит гарантии сохранности поведения модуля и целостности после его перекодирования.

Автотесты не заменяют ручного тестирования. Ручное тестирование чаще всего провести быстрее и дешевле чем написать и в последующем сопровождать автотесты. В частности, если тестирование проводится после разработки кода, то из этого тестирования автоматизировать нужно только тот кусок, который войдет в регулярное тестирование критического функционала.

И напоследок - чеклист готовности компании к автотестам

Сразу оговоримся этот чек лист не для всех, он написан для тестировщиков компаний для которых разработка софта есть основной источник дохода.

Чек лист

- ❏ В компании отрисован основной бизнес процесс доставки и есть понимание где вы конкретно находитесь.
- ❏ В компании отрисован процесс доставки ценности заказчикам.
- ❏ Поставлено управление задачами, это означает что все причастные переводят задачи в нужный статус. И задачи при этом типизированы.
- ❏ В компании сформулированы цели тестирования.
- ❏ Заголовки задач в трекере “причесаны”, иными словами, по заголовку можно понять что это за задача.
- ❏ Реестр задач управляем, в любой момент времени он отражает текущий статус и политику проекта/продукта.
- ❏ Реестр требований есть и он управляем.
- ❏ Существует трассируемость требований на задачи.
- ❏ Существует трассируемость требований на тесты. Известно какие требования какими тестами покрыты.
- ❏ Существует трассируемость тестов на задачи. Известно что уже протестировано, где и как.
- ❏ Существуют матрица влияния компонент друг на друга.
- ❏ Задачи трассируются на компоненты.
- ❏ Все стоит на версионном контроле.
- ❏ Существует версионная политика, кто, как и зачем. Есть понимание почему git flow плохая модель.
- ❏ Существую стандраты: кодирования и прочего
- ❏ Есть CI
- ❏ Есть релизная политика, где в частности прописаны способы версионирования, всего чего надо.
- ❏ Есть репозиторий для артефактов, откуда вы можете однозначно вынуть готовый к установке продукт.
- ❏ Есть политика по маркировке артефактов по разным критериям. Не забыт статический анализ кода.
- ❏ Среда для развертки продукта поднимается по щелчку пальца. Среда тоже на версионном контроле.
- ❏ Среда полностью автоматизировано проверяются на правильность. Порты, версия джавы, ….
- ❏ Продукт разворачивается по щелчку с проверкой
- ❏ Продукт конфигурируется полностью автоматически под необходимую задачу. Кстати это относится и бизнес конфигурации. И это тоже проверятся в автоматически.

- ❏ У вас есть способ повторяемо и автоматически генерировать все необходимые тестовые данные. Скрипты генерации тоже на версионном контроле и связаны с артефактами продукта.
- ❏ Все выше указанное работает для любой версии продукта.
- ❏ У вас прописан конвейер поставки, внутри релизной политики.

Не войдет в публикацию

Приложу еще памятку от Сергея Титкова

В данном документе представлен чек лист помогающий ответить на очень популярный вопрос: Когда надо начинать заниматься автоматизированным тестированием. Сразу оговоримся этот чек лист не для всех, он написан для тестировщиков компаний для которых разработка софта есть основной источник дохода.

Чек лист создан в помощь веб посиделкам КиФБ. Если у вас при чтении уже все горит, это норма. Мы разжигаем! И да, тут все про деньги и только про них.

Если вы дошли до конца и у вас везде проставлены птички, поздравляем, автоматизируйте!

Чек лист

- ❏ В компании отрисован основной бизнес процесс доставки и есть понимание где вы конкретно находитесь
- ❏ В компании отрисован процесс доставки ценности заказчику или нескольким заказчика не принципиально.
- ❏ Поставлено управление задачами, это означает что все причастные переводят задачи в нужный статус. И задачи типизированы
- ❏ В компании сформулированы цели тестирования.
- ❏ Заголовки задач в трекере “причесаны”, иными словами, по заголовку можно понять что это за задача, да не умная девочка служанка должна понимать такой заголовок. Почитать тут
- ❏ Реестр задач управляем, в любой момент времени он отражает текущий статус и политику проекта/продукта
- ❏ Реестр требований есть и он то же управляем
- ❏ Существует трассируемость требований на задачи.
- ❏ Существует трассируемость требований на тесты. Да, мы знаем что и как покрыли.
- ❏ Существует трассируемость тестов на задачи, да мы знаем что протестировано, где и как.
- ❏ Существуют матрица влияния компонент друг на друга

- ❏ Задачи трассируются на компоненты, да мы всегда знаем почему мы пишем или удаляем код
- ❏ Все стоит на версионном контроле
- ❏ Хорошо, вы особый случай, да текстуру весом 25 гб не надо класть в систему контроля версий. Кстати а сколько раз вы их уже того…. ?
- ❏ Существует версионная политика, кто, как и зачем.
  Да есть понимание почему git flow плохая модель.
- ❏ Существую стандраты: кодирования и прочего
- ❏ У вас есть CI
- ❏ Да! У вас конечно должна быть релизная политика, где в частности прописаны способы версионирования, всего чего надо.
- ❏ У вас есть репозиторий для артефактов, откуда вы можете однозначно вынуть готовый к установке продукт. Рука сама стала писать yum…
- ❏ У вас есть политика по маркировке артефактов. По разным критериям, статический анализ кода, кстати не забыт
- ❏ Среда для развертки продукта поднимается по щелчку пальца. Да она то же на версионном контроле. Все как код!
- ❏ Среда полностью автоматизировано проверяются на правильность. Да порты, да версия джавы, да этот ….
- ❏ Продукт разворачивается по щелчку. Разумеется с проверкой :) Поставить все могут а вот, что бы работало!
- ❏ Продукт конфигурируется полностью автоматически под необходимую задачу. Кстати это относится и бизнес конфигурации! И это тоже проверятся в автоматически! Кто сказал про стык с платежными системами????
- ❏ У вас есть способ повторяемо и автоматически генерить все необходимые тестовые данные. Да это то же на версионном контроле. Да оно связано с артефактами продукта.
- ❏ Я упомянул, что все выше указанное работает для любой версии продукта?
- ❏ У вас прописан конвейер поставки, он обычно внутри релизной политики, но я вытащил наружу.

Поздравляем!
Вы готовы к написанию автотестов!

Еще нет.

- ❏ Интерфейсы, будь то GUI или API проектируются ДО того, как программист сядет писать код. Да, в соответствии со стандартом проектирования интерфейсов. Да, там прописана политика именования идентификаторов элементов управления (для GUI).
- ❏ Задачу на написание тестов тестировщик получает, как правило, раньше, чем программист получит задачу на написание кода.

И не надо начинать автоматизацию тестирования, с написания регрессионных тестов. Вот совсем не надо.

P.S.

По проектированию и архитектуре полезны буду книги (по рекомендации П. Озолина):

- https://en.wikipedia.org/wiki/Design_Patterns
- Crispin, Lisa and Janet Gregory. Agile Testing: A Practical Guide for Testers and Agile Teams (не смотря на название, по отзывам в основном про архитектуру :)