Тестирование ПО. Уровень 1

Регрессионное тестирование

Регрессионное тестирование. Виды регрессионного тестирования. Регрессионное и повторное тестирование Тест-план и тестовая стратегия.

Оглавление

[Регрессионное тестирование](#_cajrpgf2jwwp)

[Как меняется ПО в течение жизненного цикла?](#_8pyj8cw065g1)

[Пример прохождения фаз разработок и добавления новых функций](#_azpjdvwzn9fx)

[Регрессионное тестирование vs повторное тестирование](#_cjcp80zem3em)

[Тест-план и стратегия тестирования](#_tdboc6ty0gj1)

[Практики, применяемые на проектах для снижения рисков появления новых дефектов](#_fg9nuttmncpp)

[Модульные тесты (Unit tests)](#_4p4i68n1pt9x)

[Интеграционные тесты](#_7sqlgik70mi1)

[Виды регрессионного тестирования](#_l5ufd89i64jc)

[Как проводится регрессионное тестирование и на каких этапах](#_46h7xb8zt12i)

[Когда надо добавлять новый регрессионный тест?](#_17jsegmzjngn)

[Приоритизация регрессионных тестов](#_35yv55wgpa90)

[Пример](#_yv44rwv4yal)

[Глоссарий](#_3znysh7)

[Практическое задание](#_guw98rsna002)

[Требования к выполненной работе](#_4d34og8)

[Дополнительные материалы](#_17dp8vu)

[Используемые источники](#_26in1rg)

# 

# Регрессионное тестирование

## Как меняется ПО в течение жизненного цикла?

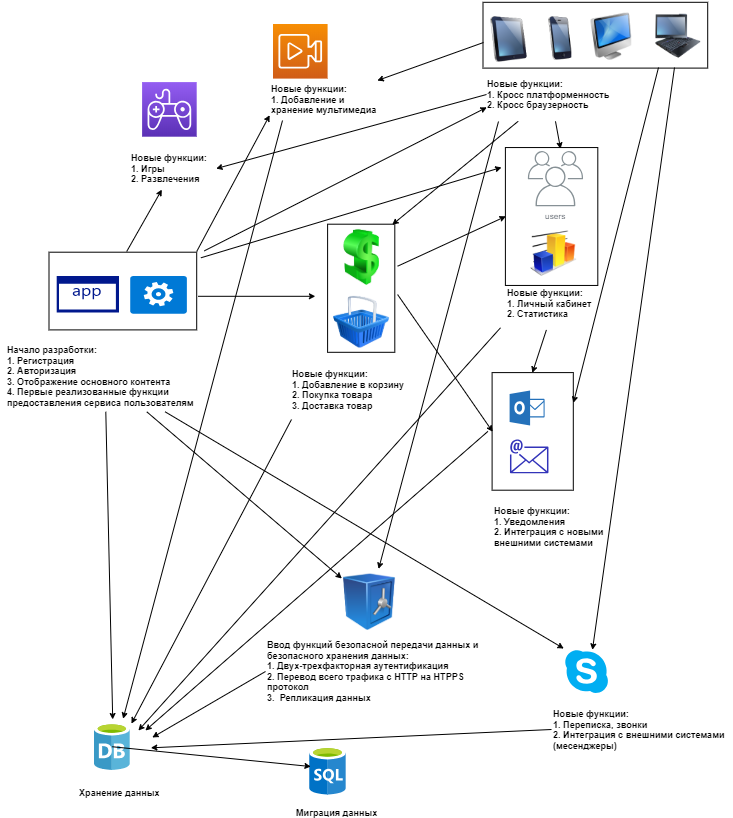
Прежде чем непосредственно перейти к рассмотрению регрессионного тестирования, необходимо вспомнить что такое инкрементальная и итеративная модели разработки ПО.



На левой части рисунка представлена инкрементная модель, поскольку каждый раз происходит добавление нового инкремента или функциональности.

На правой части предоставлена итеративная модель: мы видим, что идет «прорисовка» или доработка существующего функционала.

Рассмотрим теперь пример реальных функций и функционала, который может добавляться или изменяться со временем в ходе каждой итерации.



Вы можете посмотреть, как происходят фазы разработки и добавления новых функций или сервисов для пользователя или какого-то приложения, и доработка функций и сервисов.

### Пример прохождения фаз разработок и добавления новых функций

**Первая итерация.** Мы создаём саму платформу, тот базовый набор функций, который нужен заказчику или конечному пользователю. Это может быть:

* Форма регистрации.
* Форма авторизации.
* Предоставление определенного контента пользователю (конечному пользователю).

Не всегда такой базовый набор функционала может сразу выйти на рынок, в реальную жизнь, быть доступен конечному пользователю.

Ещё раз вспомним, что преимущества итерационно-инкрементальных моделей и гибких методологий (которые основаны на итерационно-инкрементальных моделях) в том, что мы уже на ранних этапах разработки можем показывать заказчику прототипы или ранние версии работающего продукта.

**Вторая итерация.** Это может быть добавление нового функционала. К примеру:

* покупка товара,
* доставка товара.

При покупке товара могут использоваться обычные способы электронной оплаты — банковские карты Visa, к примеру.

**Третья итерация.** Добавление новых функций и сервисов для конечного пользователя. К примеру:

* игры,
* развлечения,
* добавление и хранение мультимедиа,
* добавление новых способов оплаты (новые банковские карты Mastercard, МИР, криптовалюта, оплата через электронные кошельки),
* доработка предыдущего функционала — корзины, каталога товара.

**Четвёртая итерация** — добавление нового функционала. К примеру:

* личный кабинет пользователя,
* статистика по покупкам и другим действиям пользователя,
* уведомления и рассылка сообщений,
* интеграция с внешними сервисами (к примеру, Skype, соцсети).

Здесь очень важно отметить, что разработка нового добавляемого функционала или доработка уже существующего может затронуть прежде работающий функционал, а может его никак не трогать. Стрелками на рисунке выше представлены возможные взаимосвязи между модулями или функциями приложения.

Ошибки, которые обнаруживаются в ходе тестирования прежде работающего функционала, называют регрессионными дефектами. Иногда выражаются, что внесена регрессия.

# Регрессионное тестирование vs повторное тестирование

В соответствии с ISTQB, регрессионное тестирование (regression testing) — тестирование протестированной ранее программы, проводящееся после модификации ПО для уверенности в том, что процесс изменения или дополнения функциональности не внёс или не активизировал ошибки в областях, не подвергавшихся изменениям.

Регрессионное тестирование, как правило, проводится всегда после изменений в коде программного продукта или его окружения. Каждый раз, когда в ходе новой итерации происходит разработка нового или доработка существующего функционала, проводится регрессионное тестирование.

Это дает информацию о том, что ранее работающий код до сих пор функционирует в соответствии с требованиями. Регрессионное тестирование проводится практически на всех проектах. Как правило, разработчики — первые, кто проверяет, что новый функционал (или модификации старого) работает. Как правило, они проверяют это на своих локальных машинах до того, как отправляют свой код в общий репозиторий кода разрабатываемого продукта.

Поскольку регрессионное тестирование с каждой итерацией занимает всё больше времени, сценарии становятся сложнее, и, как правило, для проведения регрессионного тестирования создаётся набор проверок или тест-кейсов.

Поскольку физически невозможно запомнить не только весь объем необходимых проверок, но и их последовательность (исходя из приоритета/важности проверяемого функционала), составление тест-кейсов и их наборов значительно облегчает работу по тестированию.

Набор проверок может быть в виде:

* Чек-листа.
* Набора тест-кейсов, относящихся к одному модулю приложения или сгруппированных по какому-то признаку.

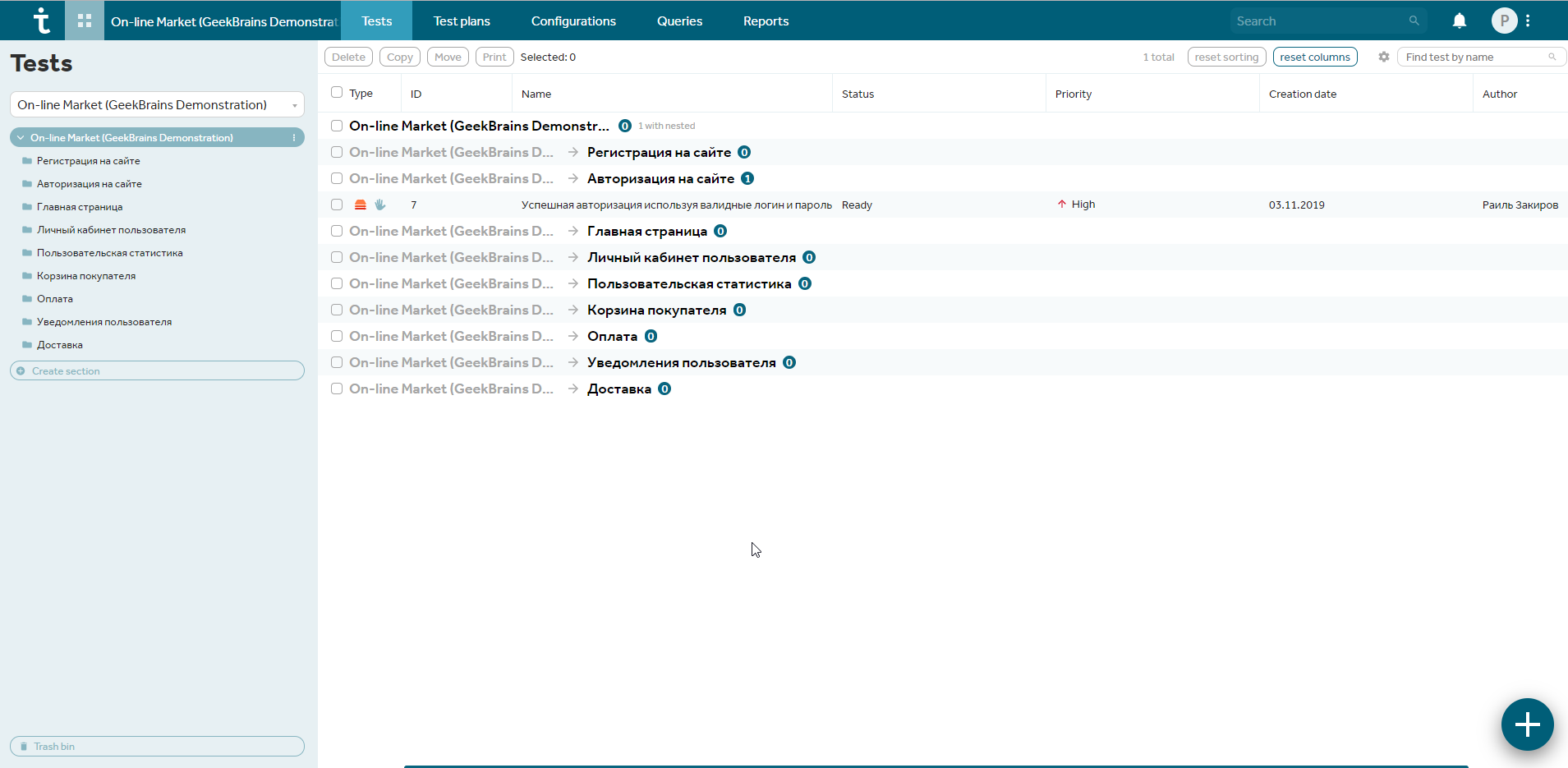
Фредерик Брукс в своей книге «Мифический человеко-месяц» писал: «Фундаментальная проблема при сопровождении программ состоит в том, что исправление одной ошибки с большой вероятностью (20–50 %) влечёт появление новой. Поэтому весь процесс идет по принципу «два шага вперед, шаг назад». Также Фредерик Брукс объяснял, почему не удается устранять ошибки более аккуратно.

Во-первых, даже скрытый дефект проявляет себя как отказ в каком-то одном месте. В действительности же он часто имеет разветвления по всей системе, обычно неочевидные. Попытка исправить его минимальными усилиями приведет к исправлению локального и очевидного, но, если только структура не очень ясная или документация не очень хороша, отдаленные последствия этого исправления останутся незамеченными.

Во-вторых, ошибки обычно исправляет не автор программы, а зачастую младший программист или стажер.

Наборы регрессионных тестов хранятся в системах по хранению и управлению тест-кейсами, или системах по управлению тестированием.

Один из примеров — TestIT.



Задачи, решаемые такими системами:

1. Хранение тестовой документации (тест-кейсов, чек-листов, тест-планов, требований).
2. Анализ тестового покрытия требований (не во всех системах).
3. Запуск (прохождение) тест-кейсов во время тестового цикла.
4. Хранение результатов каждого из тестовых циклов.
5. Отображение статистики/результатов каждого из тестовых циклов.
6. Запуск автоматизированных тест-кейсов (не во всех системах).
7. Интеграция с внешними системами, к примеру, с баг-трекинговыми для связывания тест-кейса с обнаруженным дефектом.
8. Распределение задач между членами команды тестирования.

**Подтверждающее тестирование (confirmation testing)** — тестирование, при котором выполняются тестовые сценарии, которые были не пройдены при последнем запуске, чтобы подтвердить успешность исправлений.

Как мы уже знаем, в ходе проведения цикла тестирования может быть обнаружено определённое количество дефектов (ошибок) в работающем продукте (программе). После того, как отчёты о дефектах были созданы, дефекты — назначены на разработчика, а разработчик провел исправление дефекта, он передаётся (возвращается обратно) команде тестирования. Команда тестирования подтверждает или не подтверждает исправление дефекта.

Как правило, разработчики стараются сами первоначально проверить, что их исправления действительно работают, и дефект исправлен. Разные авторы и специалисты в области тестирования считают по-разному: кто-то говорит, что подтверждающее тестирование — часть регрессионного, кто-то — что это отдельный вид тестирования. В курсе мы будем рассматривать эти два вида тестирования как отдельные.

Чтобы провести подтверждающее тестирование (иногда говорят re-testing), необходимо пройти по шагам отчёта о дефекте, либо пройти тест-кейс, который был заблокирован дефектом.

# Тест-план и стратегия тестирования

В соответствии с ISTQB, **тест-план** — документ, который описывает цели тестирования, которые должны быть достигнуты, а также средства/инструменты и график для их достижения, призванные координировать тестирование.

Часто можно встретить разделение тест-плана на два вида:

* Мастер-тест-план.
* Детальный тест-план.

Мастер-тест-план составляется на весь проект. Детальный тест-план составляется на отдельный тестовый модуль.

Цели создания и написания тест-плана:

1. Урегулирование процессов тестирования.
2. Выставление приоритетов.
3. Планирование ресурсов.
4. Учёт программного обеспечения и человеческих ресурсов.

Урегулирование процессов тестирования — общее описание целей и методов тестирования, а также подходов к нему.

Выставление приоритетов — перечисление тех функций и нефункциональных особенностей, которые будут подвергаться тестированию, в нужном порядке.

Планирование ресурсов — планирование, какие тестовые стенды и серверы будут использоваться для тестирования.

Учёт человеческих ресурсов — кто из членов команды будет задействован и какое время заложено на проведение того или иного вида тестирования.

**Основные пункты, которые может включать в себя тест-план:**

1. Цель.
2. Области, подвергаемые тестированию.
3. Области, не подвергаемые тестированию.
4. Тестовая стратегия и подходы.
5. Критерии:
   1. Приёмочные критерии, критерии качества.
   2. Критерии начала тестирования.
   3. Критерии приостановки тестирования.
   4. Критерии возобновления тестирования.
   5. Критерии завершения тестирования.
6. Ресурсы.
7. Расписание.
8. Роли и ответственность.
9. Оценка рисков.
10. Документация.
11. Метрики.

Цель кратко описывает цели разработки приложения.

Области, подвергаемые тестированию — перечень функций и/или нефункциональных особенностей̆ приложения, которые будут подвергнуты тестированию.

Области, не подвергаемые тестированию — перечень функций и/или нефункциональных особенностей̆ приложения, которые не будут подвергнуты тестированию. Причины исключения той или иной̆ области из списка тестируемых могут быть самыми различными — от предельно низкой их важности для заказчика до нехватки времени или иных ресурсов.

Тестовая стратегия и подходы — перечисление всех методов и подходов, которые будут использоваться во время тестирования.

Критерии включают в себя:

* Приёмочные критерии — такие показатели качества, которым должен соответствовать разрабатываемый продукт как с точки зрения заказчика/пользователя, так и с точки зрения технических требований.
* Критерии начала тестирования — условия, при которых команда может приступать к работе.
* Критерии приостановки тестирования — условия, при которых тестирование приостанавливается. Это может быть появление блокирующего дефекта или если планируемый разрабатываемый функционал не был реализован полностью.
* Критерии возобновления тестирования — перечень условий, при выполнении которых тестирование возобновляется. Например, исправление блокирующего дефекта или получение нового билда (сборки), удовлетворяющего всем требованиям для проведения тестирования.
* Критерии завершения тестирования — условия, при которых тестирование завершается. К примеру, прохождение всех ручных тестов, запуск всех автотестов или законченное нагрузочное тестирование.

Ресурсы — список всех необходимых для успешной̆ реализации стратегии тестирования ресурсы, которые в общем случае можно разделить на:

* программные ресурсы (какое ПО необходимо команде тестировщиков, сколько копий и с какими лицензиями (если речь идёт о коммерческом ПО));
* аппаратные ресурсы (какое аппаратное обеспечение, в каком количестве и к какому моменту необходимо команде тестировщиков);
* человеческие ресурсы (сколько специалистов, какого уровня и со знаниями в каких областях должно подключиться к команде тестировщиков в тот или иной̆ момент времени);
* временные ресурсы (сколько по времени займёт выполнение тех или иных работ).

Расписание — календарь, в котором указано, что и к какому моменту должно быть сделано. Особое внимание уделяется т. н. «ключевым точкам» (milestones), к моменту наступления которых должен быть получен некий значимый̆ ощутимый̆ результат.

Роли и ответственность. Перечень необходимых ролей (например, ведущий̆ тестировщик, инженер по нагрузочному тестированию, инженер по автоматизированному тестированию) и область ответственности специалистов, выполняющих эти роли.

Оценка рисков. Перечень рисков, которые с высокой вероятностью могут возникнуть в процессе работы над проектом. По каждому риску даётся оценка представляемой̆ им угрозы и приводятся варианты выхода из ситуации.

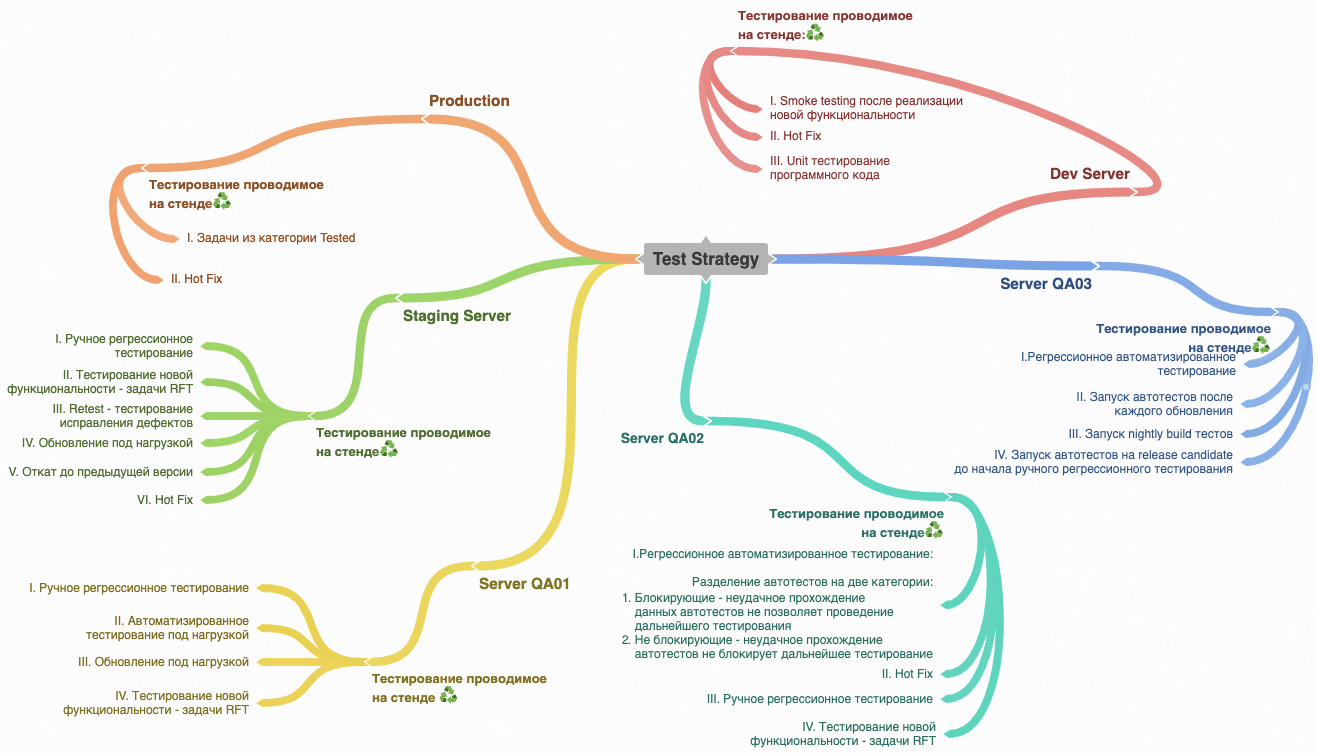
Документация. Перечень используемой̆ тестовой документации с указанием, кто и когда должен её готовить и кому передавать.

Метрики. Числовые характеристики показателей качества, способы их оценки, формулы, графики их изменения.

**Стратегия тестирования** — документация, в которой выражены общие требования к тестированию одного или нескольких проектов, выполняемых в организации. В ней предоставлены подробные сведения о порядке проведения тестирования и она приведена в соответствие с политикой тестирования.

Может быть указано, какие методы тестирования и подходы к тестированию мы будем использовать, технологии и инструменты. Тестовая стратегия входит в тест-план.

Один из графических примеров стратегии тестирования:



В примере указано распределение, на каком из тестовых стендов должен производиться тот или иной вид (подход) тестирования.

# Практики, применяемые на проектах для снижения рисков появления новых дефектов

Поскольку регрессионное тестирование направлено на проверку функций, которые ранее были протестированы и работали, чтобы убедиться, что они продолжают работать, есть ряд практик, применяемых на проектах, которые делают качество кода лучше и снижают риски внесения регрессий (ошибок, связанных с изменением или модификацией программного кода).

Среди таких практик:

1. Составление Unit-тестов (юнит или модульных тестов) для покрытия кода.
2. Составление Integration-тестов (интеграционных тестов) для покрытия кода.
3. Использование стандартов написания программного кода.
4. Использование взаимной проверки (peer review/pull request) и проведение рефакторинга кода (оптимизация уже существующего кода).

## Модульные тесты (Unit tests)

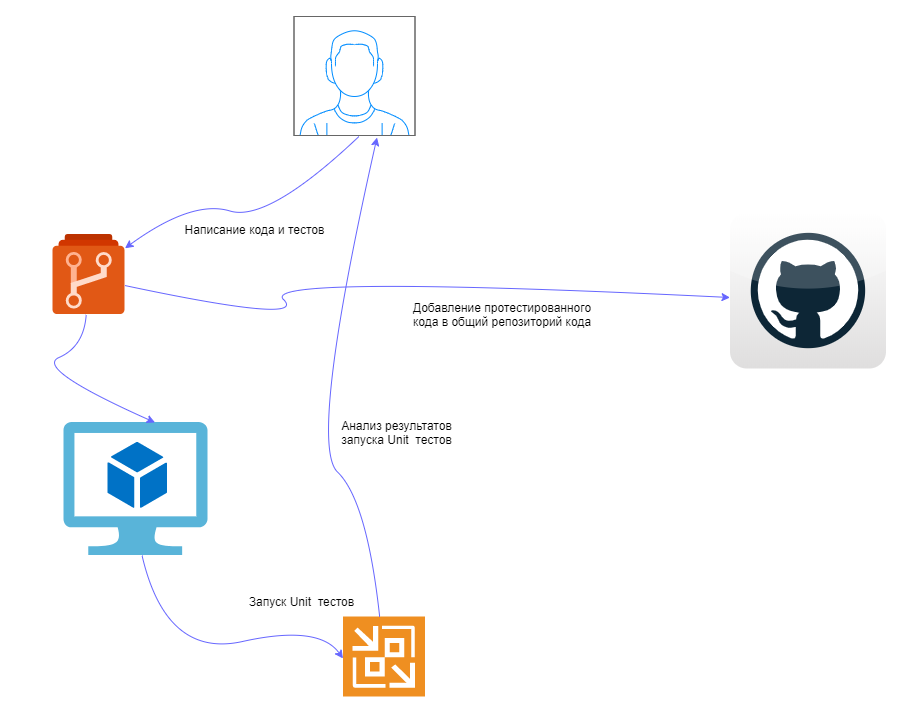
Как говорилось ранее, модульное тестирование (Unit testing) направлено на проверку отдельных небольших частей приложения, которые, как правило, можно исследовать изолированно от других подобных частей.

Как правило, когда говорят про unit-тесты, подразумевают тесты, которые пишут разработчики ПО. По мере написания программного кода они создают и юнит-тесты. Среди преимуществ юнит-тестирования можно выделить следующие:

1. Низкоуровневые, фокусируются на маленькой части ПО.
2. Тесты пишут сами разработчики.
3. Тесты выполняются очень быстро, можно делать это несколько раз в минуту.
4. При разработке можно выполнять не все тесты, а только те, которые необходимы именно вам.

Использование юнит-тестов помогает избегать появления дефектов на уровне функций. Поскольку разработчики пишут программный код, им быстрее написать юнит-тесты, чем это делали бы инженеры по тестированию. Поскольку эти тесты проверяют отдельно функции, их запуск происходит за очень короткое время.

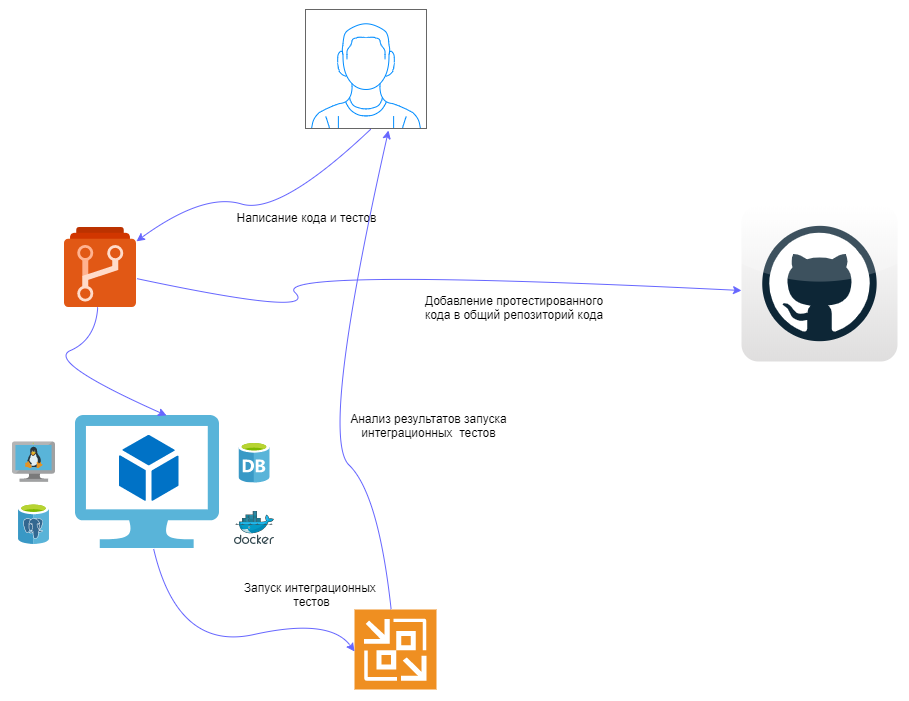
Как правило, перед тем, как добавлять код в общий репозиторий, разработчик запускает Unit-тесты.



## Интеграционные тесты

Помимо модульных тестов, направленных на тестирование сугубо изолированных частей кода, которые пишут разработчики, создаются и интеграционные тесты.

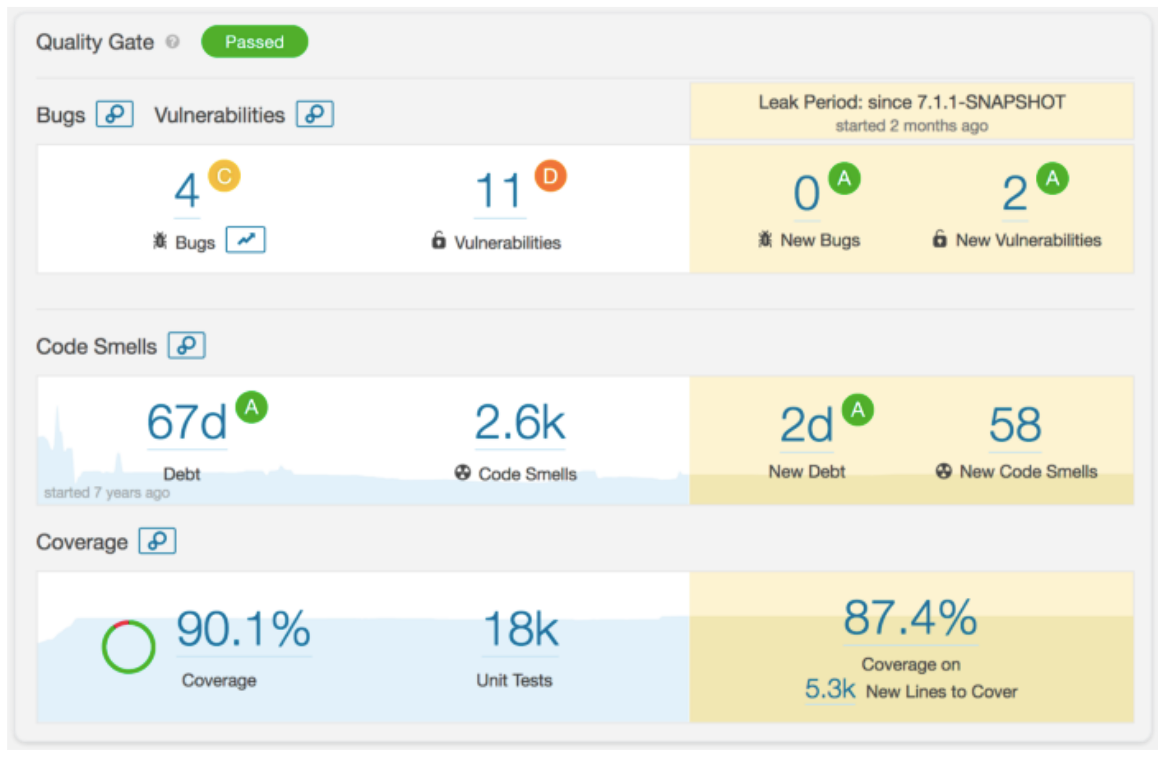
Цель интеграционных тестов — проверка интеграции с другими компонентами. К примеру, проверка интеграции с базой данных. Для запуска интеграционных тестов часто необходима установка дополнительных компонентов, тестирование с которыми необходимо провести. К примеру, базы данных.



Как правило, интеграционных тестов пишется меньше, чем модульных. Запуск интеграционных тестов занимает больше времени.

С точки зрения тестировщика, если разработчик пишет (если их нет) юнит-тесты и интеграционные тесты, меньше вероятность того, что он внесет ошибку.

Есть специальные программы, которые анализируют код и подсчитывают, какой его процент покрыт юнит-тестами. Один из примеров такого ПО — SonarCube. SonarCube проводит статический анализ кода и предоставляет отчёт такого анализа (см. картинку ниже).

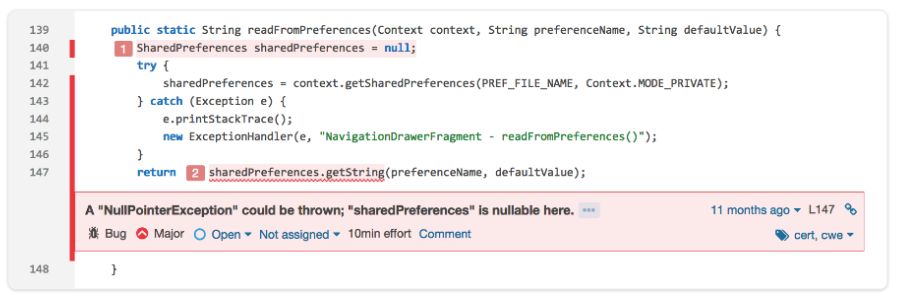


Что в себя включает отчёт:

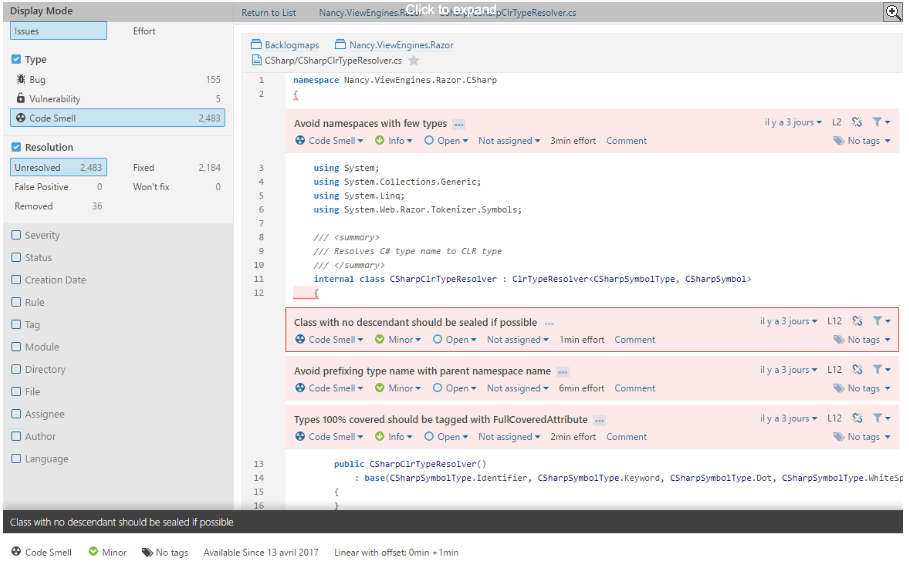
* Количество новых дефектов.
* Количество новых уязвимостей в коде.
* Количество строчек кода CodeSmells, то есть строчек кода, которые сложно поддерживать, либо не соответствующих стандартам тестирования, имеющих недостаточно или слишком много комментариев и так далее.
* Покрытие программного кода модульными тестами.

Примеры отчётов из SonarCube:

1. Обнаруженный дефект уровня Major:



1. CodeSmells:



Инструмент позволяет настраивать метрики качества программного кода, чтобы дальше пропускать (или нет) код для последующей сборки нового билда (релиза) и передачи релиза на тестирование в команду QA. Использование подобных инструментов позволяет обнаружить и исправить дефекты в ПО до выпуска нового билда, что сэкономит время для дальнейшей разработки продукта.

# Виды регрессионного тестирования

Как правило, выделяют такие виды регрессионного тестирования:

* Автоматизированное тестирование.
* Ручное тестирование.

Помимо этого, встречается разделение на:

* полное регрессионное тестирование,
* выборочное регрессионное тестирование.

Полное регрессионное тестирование проводится, если нужно проверить весь программный продукт.

Выборочное регрессионное тестирование проводится, если изменения в коде затрагивают только определенные модули программного продукта, и делать полное регрессионное тестирование нет смысла.

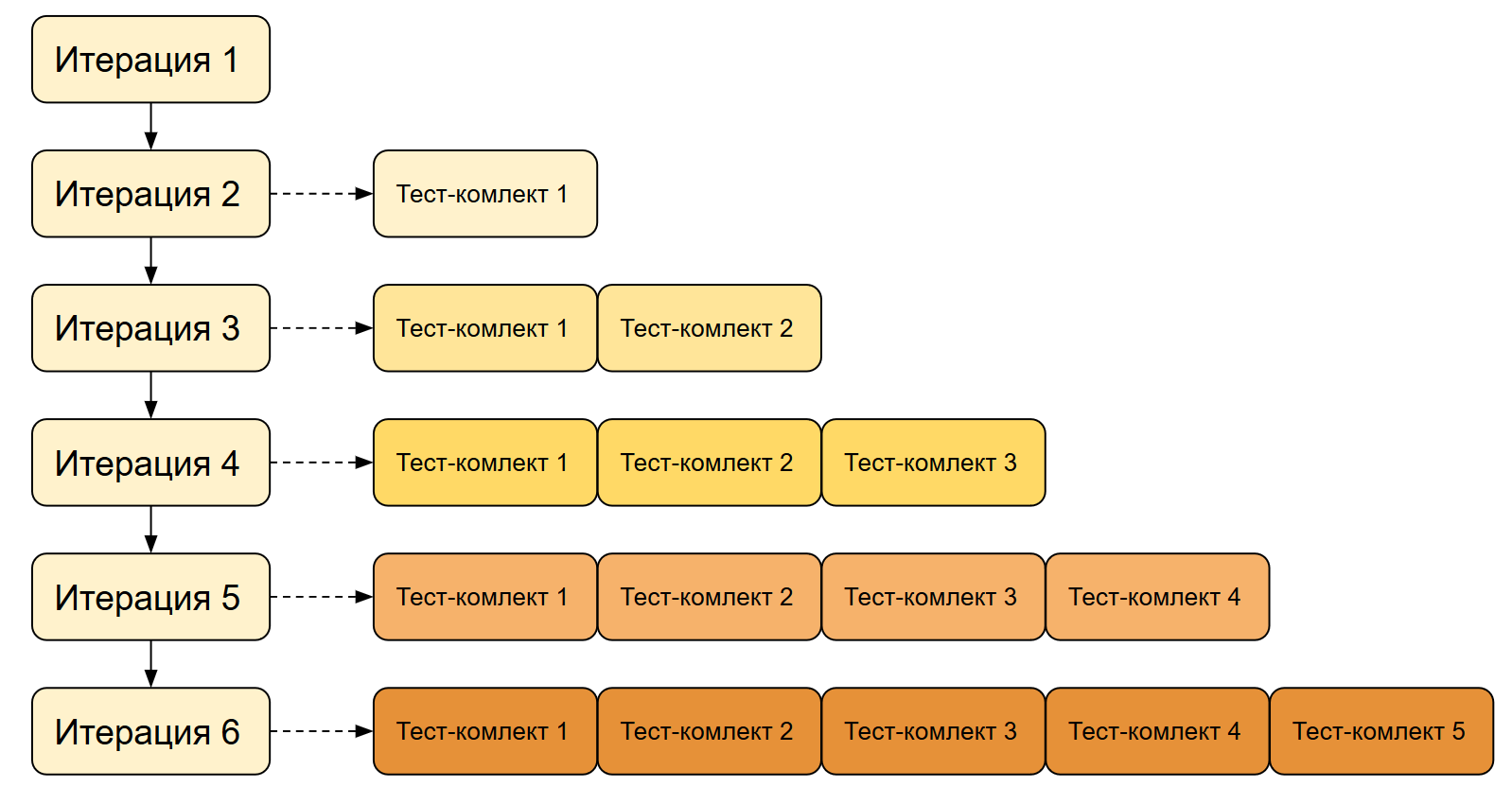
## Как проводится регрессионное тестирование и на каких этапах

Перед самым первым выпуском первой сборки (первая часть разрабатываемого функционала), регрессионного тестирования не будет, поскольку этот код ещё не тестировали. Для первичного тестирования будет создан, к примеру, тестовый комплект №1.

Когда мы разрабатываем вторую часть функционала, надо будет провести не только тестирование новых функций (тестовый комплект №2), но и регрессивное тестирование в тех модулях или частях, которые могли быть затронуты при разработке, используя тестовый комплект №1.

И так далее. По факту получается, что чем больше мы разрабатываем программный продукт, тем больше у нас добавляется регрессивных тест-кейсов (тестовых наборов), и тем больше времени может уходить на регрессионное тестирование. В случае, когда на полное проведение регрессионного тестирования нет времени, мы делаем выборку наборов тест-кейсов для выборочного регрессионного тестирования.

Наглядно прирост тестов с каждой итерацией можно изобразить таким образом:



Преимущество полного регрессионного тестирования — уверенность в качестве продукта из-за возможного большего количества обнаруженных и устранённых багов. Недостаток — большие затраты времени. Поэтому при ограниченном времени на тестирование необходимо взвешивать риски и проводить лишь частичное регрессионное тестирование. Чтобы нивелировать недостаток, связанный со временем, проводят автоматизацию регрессионного тестирования.

В чём преимущества и недостатки автоматизации?

Преимущества:

1. Скорость.
2. Отсутствие человеческого фактора.
3. Минимизация затрат.
4. Выполнение задач, которые порой непосильны человеку.
5. Сбор, первичная аналитика и агрегирование результатов автоматизированного тестирования.

Недостатки:

1. Необходимость в высококвалифицированном персонале.
2. Высокие затраты на средства автоматизации.
3. Требование к более тщательному планированию и управлению рисками.
4. Чувствительность к малейшим изменениям в коде.

## Когда надо добавлять новый регрессионный тест?

1. Когда в ПО появилась новая функциональность, вы покрываете новый функционал тестами.
2. Когда в ПО была найдена ошибка уровня Critical/Major, и этот дефект относится к функционалу, который не покрыт тест-кейсами.
3. Когда хотим улучшить тестовое покрытие ПО и расширяем количество тестов.

## Приоритизация регрессионных тестов

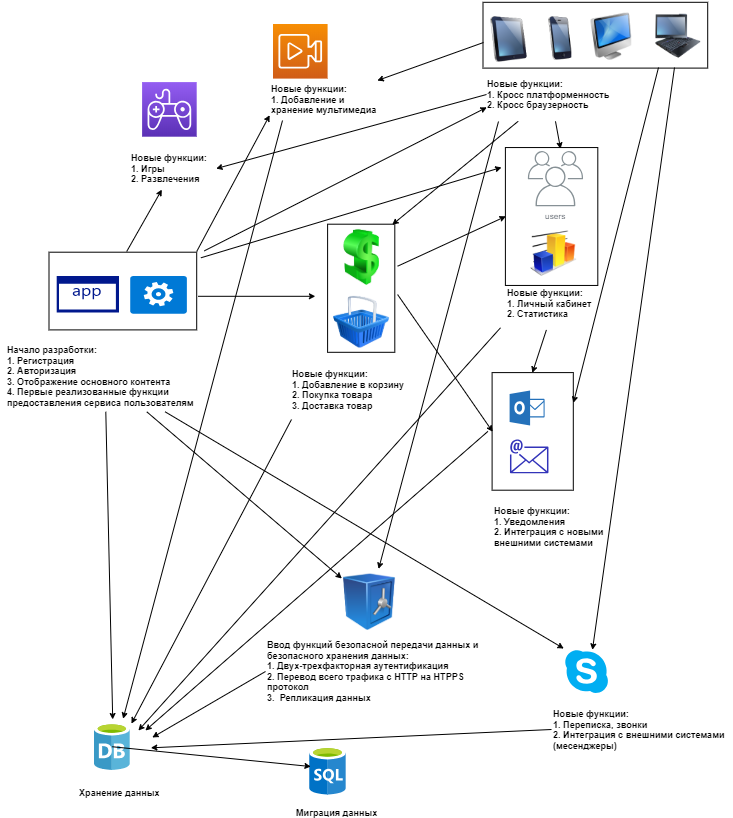
Даже при выборочном регрессионном тестировании может не хватать времени на выполнение всего набора. Поэтому набор нужно разбивать на поднаборы и каждому присваивать приоритеты.

Зачем нужна приоритизация?

1. 1. Чем раньше мы узнаем, что в ПО появилась серьезная регрессионная ошибка, тем скорее её исправят.
2. 2. Время на тестирование — ограниченный ресурс. Необходимо найти наибольшее число важных ошибок с учётом всех ограничений.

### **Пример**

Рассмотрим изменение или модификацию программного кода, относящегося к покупке товара и его добавлению в корзину, как представлено ниже.



В данном случае необходимо будет провести регрессионное тестирование следующих модулей и компонентов:

1. Корзина покупателя.
2. Модуль, относящийся к покупке товара.
3. Сохранение данных в базу данных.
4. Отображение покупок в статистике и в личном профиле пользователя.
5. Работа отправки уведомлений по покупке нового товара.

# Глоссарий

**Регрессионное тестирование** — тестирование уже протестированной ранее программы, проводящееся после модификации ПО для уверенности, что модификация (изменения или дополнения) не внесла или не активизировала ошибки в областях, не подвергавшихся изменениям.

**Подтверждающее тестирование (confirmation testing)** — тестирование, при котором выполняются тестовые сценарии, не пройденные при последнем запуске, чтобы подтвердить успешность исправлений.

**Тест-план** — документ, который описывает цели тестирования, которые должны быть достигнуты, а также средства/инструменты и график для достижения этих целей, организованные координировать деятельность по тестированию.

# Практическое задание

**Используя приложенную таблицуl, выполните следующие задания:**

1. Прочитать Святослава Куликова, страницы 208-217.
2. Составить тест-кейсы по чек-листу для раздела [«Вебинары»](https://geekbrains.ru/events) на GeekBrains.
3. Распределить тест-кейсы по тестовым наборам и приоритету.
4. \* Подготовить ответы к вопросам на собеседование**.**

## Требования к выполненной работе

1. Домашнее задание должно быть выполнено в приложенной форме.
2. Расширение файла должно быть формата Excel (XLS, XLSX).
3. Название файла не менять, и вместо ФИО укажите свои фамилию и имя.

# Дополнительные материалы

1. Фредерик Брукс — «Мифический человеко-месяц, или как создаются программные системы».
2. [13th Annual State Of Agile Report](https://explore.versionone.com/state-of-agile/13th-annual-state-of-agile-report).
3. [Пирамида тестов на практике](https://habr.com/ru/post/358950/).
4. [Проблема дублирования и устаревания знания в mock-объектах, или Интеграционные тесты — это хорошо](https://habr.com/ru/post/275249/).
5. [Обеспечение качества кода в масштабных проектах](https://habr.com/ru/company/airbnb/blog/252125/).
6. [Тестирование в Airbnb](https://habr.com/ru/company/airbnb/blog/238267/).
7. [Блог a1qa. О тестировании и качестве ПО](https://www.a1qa.ru/blog/regressionnoe-testirovanie-tseli-podhody-etapy-provedeniya/).

# Используемые источники

Для подготовки методического пособия мы использовали эти ресурсы:

1. Software Testing — Base Course (Svyatoslav Kulikov).
2. Фредерик Брукс — «Мифический человеко-месяц, или как создаются программные системы».
3. [13th Annual State Of Agile Report](https://explore.versionone.com/state-of-agile/13th-annual-state-of-agile-report).
4. [Юнит-тесты. Первый шаг к качеству](https://habr.com/ru/post/336030/).
5. [SonarQube documentation](https://docs.sonarqube.org/latest/).