ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФФЕСИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

**Факультет Санкт-Петербургская школа
физико-математических и компьютерных наук**

Суходольский Максим Андреевич

Разработка интегрированной системы для приоритизации и уведомления о падении тестов в реальном времени

Выпускная квалификационная работа - БАКАЛАВРСКАЯ РАБОТА
по направлению подготовки 01.03.02 Прикладная математика и информатика
образовательная программа «Прикладная математика и информатика»

Научный руководитель:
кандидат ф.-м. н., доцент Москвин Д. Н.

Научный рецензент:
ООО «В Контакте», директор по Quality Assurance, Шваркунов М. С.

Научный консультант:
ООО «В Контакте», программист-разработчик, Спирин Е. С.

Санкт-Петербург
2023

Оглавление

Аннотация

Системы непрерывной интеграции, являясь ключевым элементом современных методологий разработки программного обеспечения, позволяют производить автоматическое тестирование всех новых изменений, вносимых в код. Это важный процесс, однако он может стать очень времязатратным, особенно в контексте больших проектов, где количество тестов может насчитывать тысячи, десятки тысяч или даже сотни. Однако, автоматически задавая нужный порядок выполнения тестов и внедрив систему мгновенного уведомления разработчика об их падениях, мы можем значительно ускорить процесс выявления и исправления ошибок в коде. Определение приоритета запуска тестов может быть осуществлено на основе множества данных, предоставляемых системой непрерывной интеграции или системой контроля версий. Например, могут быть учтены результаты предыдущих запусков тестов, данные о недавно измененных файлах, а также другая метаинформация. Изменение порядка тестов, совмещенное с уведомлением об их неудачном прохождении, сократит время до обнаружения ошибок в коде и, следовательно, до их исправления. Представленная в рамках данной работы система является продвинутым инструментом, разработанным с целью интеграции приоритизации тестов, а также уведомления о их падении в режиме реального времени. Она была успешно внедрена в проект по тестированию API в компаниии ВКонтакте. Использование этой системы не только позволяет снизить вероятность выпуска на рынок программного продукта с ошибками, что может привести к негативным последствиям для пользователей, но и значительно экономит время дежурного тестировщика и разработчика, позволяя им быстрее и эффективнее реагировать на возникающие проблемы.

Ключевые слова: приоритизация тестов, тестирование программного обеспечения, непрервыная интеграция, обнаружение ошибок, машинное обучение

Continuous integration systems, being a key element of modern software development methodologies, allow for the automatic testing of all new changes made to the code. This is an important process, but it can become very time-consuming, especially in the context of large projects, where the number of tests can reach thousands, tens of thousands, or even hundreds. However, by automatically setting the correct order of tests and implementing a system for instantly notifying the developer of their failures, we can significantly speed up the process of identifying and correcting errors in the code. Determining the priority of test execution can be done based on a variety of data provided by the continuous integration system or version control system. For example, the results of previous test runs, data on recently changed files, and other meta-information can be taken into account. Changing the order of tests, combined with notification of their unsuccessful completion, will reduce the time of error detection in the code and, therefore, the time to correct them. The system presented in this work is an advanced tool developed with the aim of integrating test prioritization and notification of their failure in real time. It was successfully implemented in the API testing project at VK. The use of this system not only reduces the likelihood of a software product with errors being released to the market, which could have negative consequences for users, but also significantly saves time for the quality assurance engineer and developer on duty, allowing them to respond more quickly and effectively to emerging issues.

Keywords: test prioritization, software testing, continuous integration, fault detection, machine learning

Введение

Описание предметной области

В современном быстро развивающемся мире информационных технологий, успешная разработка программного обеспечения требует гибких и эффективных методологий. В этом контексте, концепция непрерывной интеграции становится неотъемлемой частью процесса разработки.

Непрерывная интеграция (Continuous Integration, CI) — это практика разработки программного обеспечения, при которой команды разработчиков регулярно (часто несколько раз в день) объединяют свои изменения в общий репозиторий. После каждого такого слияния происходит автоматическая сборка и тестирование кода, что позволяет быстро обнаруживать и исправлять ошибки, улучшать качество продукта, сокращать время на развертывание и доставку новых функций до конечного пользователя.

В современном мире существует множество инструментов, поддерживающих практику непрерывной интеграции, включая Jenkins, GitLab CI/CD, Travis CI, CircleCI и другие. Однако, в контексте данной работы, особое внимание следует уделить системе TeamCity.

TeamCity [3] — это мощный инструмент, разработанный компанией JetBrains. Он предоставляет платформу для непрерывной интеграции и непрерывной доставки (Continuous Delivery, CD), обеспечивая автоматическую сборку и тестирование кода, а также мониторинг выполнения процессов на каждом этапе разработки. TeamCity поддерживает множество языков программирования и систем контроля версий, что делает его универсальным инструментом в арсенале разработчиков. Важной особенностью TeamCity является его гибкость и масштабируемость, позволяющие настраивать процессы сборки и тестирования в соответствии с уникальными требованиями конкретного проекта.

TeamCity также предлагает широкие возможности для интеграции с

другими инструментами и сервисами, что делает его идеальной платформой для создания и внедрения сложных систем непрерывной интеграции, включая предлагаемую в рамках данной дипломной работы систему приоритизации и уведомления о падении тестов в реальном времени.

Автоматическое тестирование программного обеспечения (ПО) играет ключевую роль в процессе разработки. Его основная цель — проверка функциональности и производительности кода для обеспечения качества программного продукта. Весь процесс тестирования основан на предварительно разработанных тестовых сценариях, которые автоматически выполняются с целью обнаружения ошибок и несоответствий спецификациям.

В больших проектах, где тестовые сценарии могут насчитывать тысячи, автоматическое тестирование становится весьма времязатратным процессом. Время тестирования может достигать нескольких часов, а в некоторых случаях и дней. Причинами этого являются разнообразие и сложность функциональности, которую нужно проверить, а также необходимость тестирования в различных конфигурациях и условиях.

Однако, несмотря на время и ресурсы, затраченные на автоматическое тестирование, его значимость не может быть недооценена. Оно является первой линией обороны против багов и ошибок в коде, и важно обнаружить падение теста как можно раньше, чтобы ускорить процесс исправления ошибок и сократить общее время разработки.

В этом контексте, возможность автоматически выбирать порядок тестов становится критически важной для больших проектов. Приоритизация тестов на основе различных факторов, таких как история исполнения, скорость выполнения, связанные с тестом части кода и другие, может значительно ускорить процесс обнаружения ошибок. Если система способна выбирать такой порядок тестов, который приводит к их раннему падению, это поможет ускорить обнаружение проблем и обеспечит более быстрое исправление ошибок.

Описание проекта API тестов ВКонтакте

ВКонтакте, одной из крупнейших социальных платформ в России, характерен быстрый релизный цикл. Новые изменения в коде уже через час становятся доступны пользователям, что было подробно описано в статье Михаила Шваркунова [11]. Благодаря такому подходу, ВКонтакте может быстро реагировать на потребности пользователей и внедрять новые функции. Однако это также делает скорость обнаружения ошибок на стадии тестирования критически важной. Уменьшение времени обнаружения ошибки даже на 5 минут может иметь значительное влияние на общую эффективность процесса разработки.

ВКонтакте использует различные проекты для обеспечения автоматического тестирования различных аспектов своих сервисов. Один из наиболее значимых проектов занимается тестированием API и включает в себя массив из более чем трех тысяч тестов. Каждый проход этого обширного набора тестов требует как минимум 20 минут времени, и тестирование осуществляется в TeamCity на основном домене, а также на этапе разработки с использованием 1% пользователей и в других ветках кода.

Процедура тестирования достаточно частая – каждые 30 минут запускается специфический набор из свыше 2200 надежных тестов, которые абсолютно необходимы для гарантии качества на этапе тестирования. Любое падение такого теста автоматически активизирует процесс исследования причин, которое осуществляется дежурным специалистом.

Если углубиться в технические аспекты, то можно обратить внимание на то, что используется тестовый фреймворк TestNG [1], с помощью которого тесты распределяются на 40 потоков для более эффективного и быстрого процесса тестирования. Все тесты, не прошедшие успешно при прогоне на тестовом домене, автоматически перезапускаются в конце. Перезапуск каждого теста может происходить до трёх раз. Если же после этого тест продолжает падать, он перезапускается уже на

основном домене. Важным элементом процесса тестирования является анализ причин падения теста, который осуществляется дежурным специалистом по тестированию на основе отчета Allure [8], генерируемого в TeamCity по завершении каждой сборки.

В таком контексте становится очевидной актуальность разработки и внедрения системы, которая позволила бы приоритизировать тесты и уведомлять о их падении в реальном времени. Такая система способна значительно повысить эффективность процесса тестирования, ускорить обнаружение и исправление ошибок, что в целом сократит время, необходимое для решения возникающих проблем в процессе разработки и тестирования.

Цели и задачи работы

Целью данной работы является реализация и внедрение в проект API тестов системы, которая будет способствовать автоматическому изменению порядка выполнения тестов и уведомлению о их падении.

Для достижения этой цели были поставлены следующие задачи:

- Разработать подход к приоритизации тестов: Процесс приоритизации должен быть основан на ряде факторов, таких как история прохождения тестов, скорость выполнения, связанные с тестом части кода и другие.
- Изменить механизм перезапуска тестов: Механизм перезапуска тестов должен быть оптимизирован таким образом, чтобы тесты перезапускались сразу после падения, а не в конце,
- Ускорить формирование отчета об упавших тестах: Нужно ускорить создание отчета, чтобы получать его при первом же падении теста, а не в конце всего прогона.
- Реализовать механизм уведомления дежурного: При падении теста система должна автоматически уведомлять дежурного специалиста в чате ВКонтакте.

- Внедрить механизм приоритизации тестов: Разработанный подход и механизм приоритизации должны быть успешно интегрированы в существующий процесс тестирования.

1. Обзор литературы

1.1. Научные работы

Несмотря на то, что автоматическое тестирование является важным аспектом разработки программного обеспечения, идея приоритизации тестов и использование алгоритмов для оптимизации порядка их выполнения является относительно новым направлением в исследованиях. Однако некоторые важные работы уже были выполнены в этом направлении.

Наиболее известной работой в этом области является Predictive Test Selection [7], автором которой являются инженеры Facebook. Эта работа описывает, как Facebook смог значительно сократить время регрессионного тестирования внутри своей инфраструктуры. Авторы разработали модель, которая для каждого теста определяет вероятность обнаружения регрессии при определенном изменении кода. Затем исполняется только подмножество тестов с высокой вероятностью обнаружения регрессий. Для обучения своей модели авторы использовали стандартные методы машинного обучения на большом наборе данных историй сборок проектов компании. В результате, в среднем выполнялась только около трети всех тестов, которые изначально планировались к выполнению.

Многие другие работы в этой области ссылались на исследование Facebook. В работе под названием Improving Regression Test Selection [10] оценивалось количество времени, сэкономленного благодаря использованию выборочного тестирования, описанного в статье Facebook. В среднем, это позволило сэкономить около 24% времени, которое было бы потрачено на выполнение всех тестов.

Авторы другой работы [6] пытались реализовать выборочное тестирование с помощью анализа текста и связи между измененным файлом и соответствующим тестом. Однако результаты этого подхода были неудовлетворительными. Доля правильно обнаруженных падений

тестов составила всего 49%.

Из этих работ видно, что приоритизация и выборочное тестирование могут привести к значительной экономии времени и улучшению эффективности процесса тестирования. Однако необходимо провести дополнительные исследования для улучшения этих методов и создания более надежных систем.

1.2. Существующие аналоги

Несмотря на то, что научные исследования в области выборочного тестирования продолжают активно развиваться, существуют и промышленные решения, внедрившие подходы, в основе которых лежит опыт крупных компаний, таких как Facebook. Например, в системе непрерывной интеграции TeamCity реализован механизм, который даёт возможность приоритизировать новые тесты и тесты, которые не прошли при последней сборке, располагая их в начале очереди. Тем не менее, стоит отметить, что данная функциональность не предоставляет пользователю возможностей для дополнительных настроек или кастомизации, что может ограничивать применимость подхода в различных ситуациях.

В числе коммерческих сервисов, применяющих концепцию выборочного тестирования, выделяется Gradle Enterprise. Этот сервис представил свой вариант «Predictive Test Selection» [2] в апреле 2022 года. Однако, публично доступная информация о внутреннем устройстве и принципах работы этого сервиса остается достаточно скудной. Говоря о возможностях Gradle Enterprise, стоит отметить, что он предлагает дополнительные параметры конфигурации при сборке для тестов JUnit Platform. Но стоит учесть, что это является платным решением с применением, ограниченным тестирующим фреймворком JUnit.

Другой сервис, Launchable [4], также предлагает функциональность, похожую на выборочное тестирование. Однако, как и в случае с Gradle Enterprise, информация о работе сервиса ограничена, и доступ к сервису

предоставляется на платной основе.

Кроме платных сервисов, приоритизация тестов интегрирована в фреймворке для тестирования Jest [5] для языка JavaScript. Данный фреймворк имеет открытых исходный код, проанализировав который, можно заключить, что приоритизация реализована простейшим методом, как и в TeamCity, а именно ставит в начало упавшие тесты в прошлом запуске.

1.3. Выводы

Существующие научные работы и промышленные аналоги, посвящённые приоритизации и автоматизированному тестированию, не лишены определённых недостатков.

В частности, у научных исследований, связанных с этой темой, отсутствует открытая реализация, которую можно было бы использовать в качестве отправной точки или в качестве основы для дальнейшего развития.

С другой стороны, у промышленных аналогов есть свои проблемы. Например, Jest и TeamCity показывают неэффективность в приоритизации тестов, что ограничивает их полезность в контексте регрессионного тестирования.

Сервисы вроде Launchable и Gradle Enterprise, несмотря на их возможности в области выборочного тестирования, сталкивается с проблемой неполноты тестирования, так как сервис предлагает запускать только часть тестов. Это может упустить некоторые потенциальные проблемы, которые могли бы быть обнаружены при полном тестировании.

Очевидным подходом, представляющим альтернативу выборочному тестированию, является возможность запускать все тесты, но менять их порядок. Это приводит к более раннему обнаружению ошибок, сохраняя при этом полноту тестирования.

В целом, перечисленные недостатки промышленных решений не поз-

воляют использовать их в рамках проекта API тестов компании ВКонтакте. Из-за этого было решено разработать свой инструмент, реализующий подход к приоритизации тестов, созданный на основе идей из перечисленный научных статей.

2. Разработка подхода к приоритизации

2.1. Исторический и Байесовский подходы

Различные подходы к приоритизации тестов, описанные в научных статьях, включают использование исторических данных на основе прошлых запусков, байесовскую классификацию, а также подход, когда анализируются только последние запуски, а не полная история. Однако, они часто включают нереалистичные предположения о коде и отдаляются от реальных условий. Несмотря на это, наши подходы к приоритизации тестов будут основаны на этих стратегиях, но с учетом их ограничений.

Важной метрикой в этих подходах является средняя позиция упавшего теста. Эта метрика, вместе с позицией первого или последнего упавшего теста, может помочь нам лучше понять, как наши тесты работают в контексте регрессионного тестирования. Время, конечно, является важной метрикой, поскольку мы стремимся минимизировать время до обнаружения ошибки в тесте. Для того чтобы эффективно сравнивать наши метрики с существующими, мы будем рассматривать разницу между нашими результатами и результатами без приоритизации, которые по умолчанию запускаются в TeamCity.

Существует два подхода к приоритизации тестов, которые мы реализовали. Первый — исторический подход, где мы сортируем тесты по убыванию приоритета, вычисленного как доля падений данного теста в прошлых запусках. В этом подходе мы также учитываем затухание, умножая приоритет на затухающий коэффициент после каждого сборки. Хорошим стартовым значением для затухающего коэффициента считается 0.95.

Для реализации второго подхода мы используем очевидный факт: текущие изменения в файлах влияют на успешность прохождения тестов. Поэтому по истории прохождения и падения тестов, получаемых из системы непрерывной интеграции, и истории изменения файлов, по-

лучаемых из системы контроля версий, можно отследить зависимость: при изменении каких файлов, изменяется успешность прохождения каких тестов. В этом подходе приоритет теста вычисляется (рис. 1) как вероятность падения теста при изменении определенных файлов, с учетом априорной вероятности и вероятности изменения этих файлов при падении теста. Этот подход также требует применения сглаживания Лапласа для получения более надежных результатов.

$$P(t \text{ упал} \mid \text{изменены } f_i) = P(t \text{ упал}) \cdot \prod_i P(f_i \text{ изменился} \mid t \text{ упал})$$

Рис. 1: Формула наивного Байеса

Оба подхода требуют настройки определенных коэффициентов, таких как коэффициент затухания и коэффициент сглаживания, наилучшие значения которых могут отличаться в разных проектах. Эти коэффициенты могут быть уточнены в процессе работы модели для достижения наилучших результатов.

Итоговая модель, показавшая наилучшие теоретические результаты, является линейной комбинацией двух описанных моделей с коэффициентами α для исторической и $(1 - \alpha)$ для наивного Байеса. Этот коэффициент также может различаться в разных проектах, но хорошим приближением выявлен $\alpha = 0.7$.

2.2. Выводы и результаты по главе

В данной главе исследованы и адаптированы различные подходы к приоритизации тестов из существующих научных статей, включая использование исторических данных, байесовскую классификацию и ограниченный анализ последних запусков. Выявленные метрики, такие как средняя позиция упавшего теста и время до обнаружения ошибки, оказались особенно полезными и будут использованы для тестирования подходов. Реализованы два подхода к приоритизации тестов: исторический подход, использующий затухающий коэффициент, и наивный

Байес, где приоритет определяется на основе вероятности сбоя теста при изменении конкретных файлов. При использовании этих подходов было выявлено, что коэффициенты затухания и сглаживания требуют тонкой настройки для каждого проекта, чтобы добиться наилучших результатов. Итоговой моделью является линейная комбинация двух существующих. Тестирование этой модели на предмет эффективности приведено в следующей главе.

3. Тестирование приоритизации с помощью плагина

3.1. Накопление статистики по предыдущим сборкам

Для тестирования разработанных подходов приоритизации, был создан плагин для TeamCity. Этот плагин автоматически изменяет порядок выполнения тестов для ускорения обнаружения падений и исправлений тестов, предоставляя более оперативную обратную связь команде разработки. Реализация этого плагина включала разработку системы для автоматического ранжирования тестов, а также оценку его эффективности на реальных проектах.

В процессе непрерывной интеграции с использованием инструмента TeamCity, ключевую роль играет сервер, который служит основной точкой координации всех сборок. Наиболее часто это включает в себя запросы на выполнение определенных действий, таких как запуск набора тестов или сборку кода, основанную на конкретной ветке или ревизии. Сервер TeamCity обрабатывает эти запросы и делегирует их соответствующим агентам сборки, обеспечивая точную конфигурацию, которая требуется для каждого задания. Агенты сборки выполняют назначенные им задачи и сохраняют результаты выполнения в виде артефактов. Артефакты, в свою очередь, представляют собой конкретные выводы или продукты процесса сборки, которые могут быть сохранены для последующего использования или анализа.

Для реализации возможности сохранения и накапливания статистики по уже прошедшим сборкам, было решено записывать её в файл определенного формата, который публикуется в качестве артефакта в конце сборки. Этот файл (рис. 2) включает данные о том, какие тесты и сколько раз запускались, сколько раз они падали, и какие файлы были изменены в процессе.

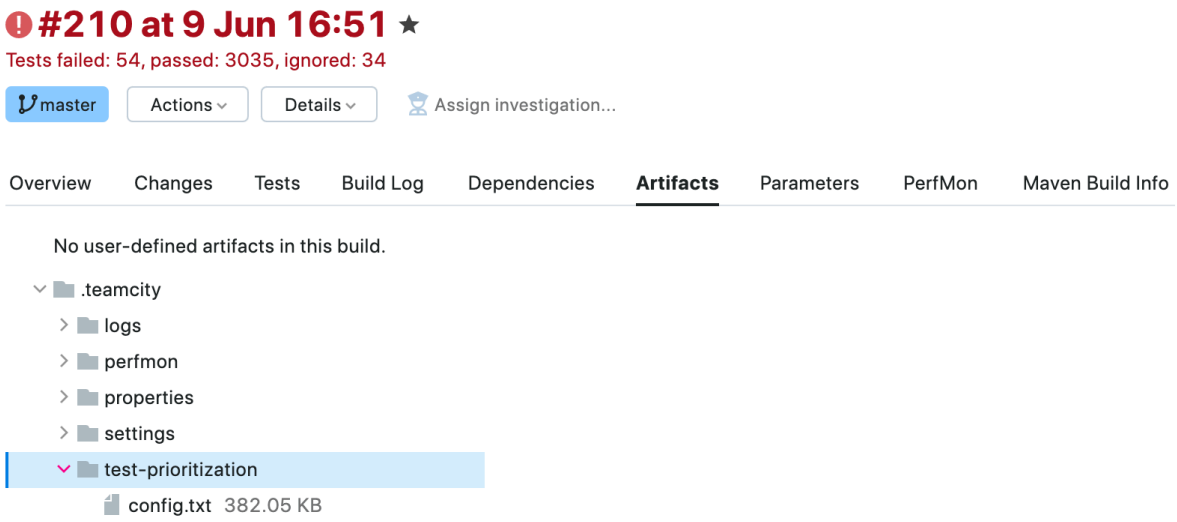

Рис. 2: Артефакт с накопленной статистикой

3.2. Алгоритм нахождения предыдущей статистики

Статистика предыдущих сборок хранится в артефактах каждой сборки, так как история сборок имеет нелинейную структуру: сборки можно запускать на разных ветках и ревизиях, и для каждой сборки нужно найти предыдущую ревизию, откуда берется собранная статистика. При единственном ревизии-предке, плагин пытается найти артефакт со статистикой там. В случае же сборок на основе merge-коммитов, ревизий-предков может быть несколько, поэтому статистика берется из последней завершенной сборки из всех возможных предков. Таким образом, при нелинейной истории сборок на рис. 3, сборка номер 1 возьмет статистику из сборки номер 2, которая является последней по дате окончания среди предков ревизии 1 под номерами 2, 3 и 4.

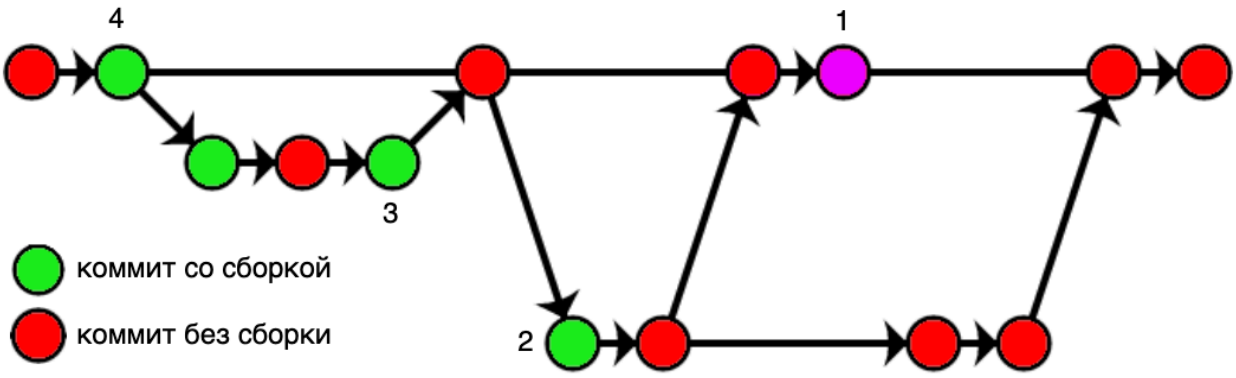

Рис. 3: Пример нелинейной структуры ревизий

После нахождения предыдущей подходящей сборке по данному алгоритму, сервер TeamCity передает её идентификатор агенту (рис. 4). После окончания сборки, сервер пересчитывает статистику и сохраняет её в артефакте уже своей сборки. Агент, в свою очередь, загружает статистику из предыдущей сборки и применяет конфигурацию перед началом своей сборки.

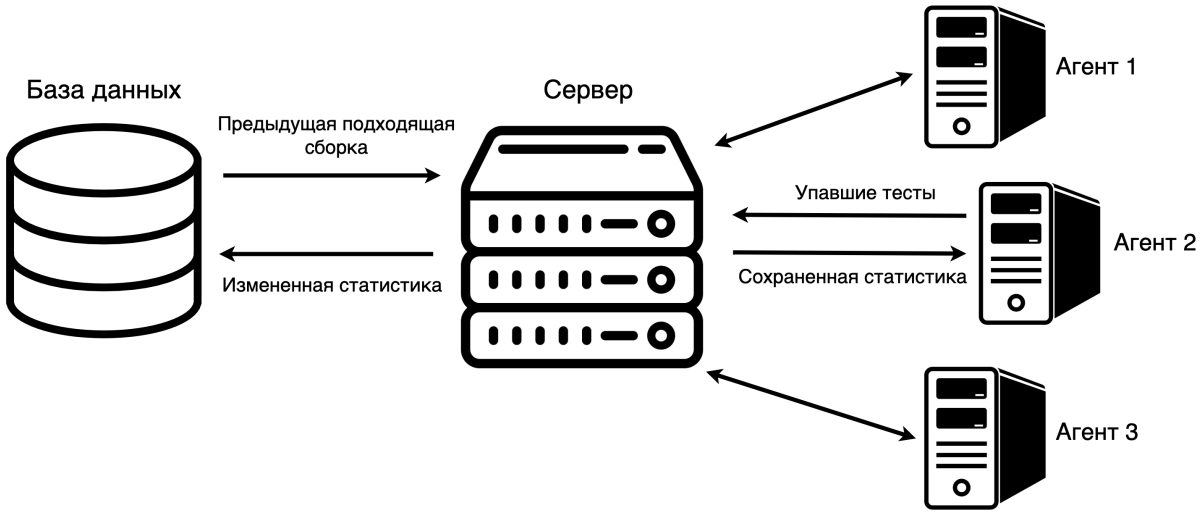

Рис. 4: Взаимодействие агента и сервера TeamCity

3.3. Реализация приоритизации

Агент использует собранную статистику для упорядочивания тестов согласно разработанному подходу. Конкретная реализация зависит от используемого языка программирования и тестового фреймворка, так как каждый из них требует своего подхода к настройке переупорядочивания тестов. В рамках этой работы было решено использовать JUnit, так как он является самым популярным тестовым фреймворком для Java и используется в большом количестве крупных проектов. Однако нужно отметить, что разработанный подход может быть применен ко всем фреймворкам, поддерживающим переупорядочивание тестов.

В отличие от теоретической модели, где порядок тестов можно устанавливать произвольно, в реальности JUnit позволяет переупорядочивать только тестовые классы между собой и методы внутри них. Кон-

фигурация JUnit осуществляется посредством внедрения файла .properties, который указывает на применяемый Java-класс. Этот класс в свою очередь реализует алгоритм переупорядочивания на основе собранной статистики в специальном переопределенном методе API JUnit.

При работе с проектами, состоящими из нескольких модулей, возникают дополнительные сложности. Внутри каждого отдельного модуля можно переупорядочить тесты, однако перемещение тестов между модулями невозможно. При этом возможно внедрение конфигурации в каждый модуль, а также переупорядочение самих модулей. Однако, это уже выходит за рамки JUnit и входит в область системы сборки.

В связи с этим, для реализации переупорядочивания модулей потребовалось также интегрироваться с системой сборки. Это было успешно реализовано для системы сборки Maven, которая используется во всех Java/Kotlin проектах, на которых проводились тесты с использованием JUnit.

3.4. Выводы и результаты по главе

Для тестирования эффективности подхода приоритизации в плагине были выбраны три крупных Java проекта, использующие JUnit: Apache Flink [13], Dolphin Scheduler [12] и Quarkus [9]. В каждом из них запускались тесты в порядке по умолчанию, а также в ранжированном с помощью плагина порядке. Метрики из прошлой главы описаны далее и их значения предоставлены в таблице 1.
На сколько раньше с плагином появляется первый упавший тест:

- На First секунд раньше

На сколько раньше с плагином появляется средний упавший тест:

- На Avg секунд раньше
- На AvgPos процентов от общего числа тестов раньше

На сколько раньше с плагином появляется последний упавший тест:

	Apache Flink	**Dolphin Scheduler**	**Quarkus**
Time	**210.2**	**721.8**	**250.8**
First	107.3	285.0	120.3
Avg	127.9	308.8	143.2
AvgPos	31%	38%	61%
Last	160.1	511.9	195.1

Таблица 1: Метрики плагина приоритизации

- На Last секунд раньше

Результаты показывают, что первый упавший тест обнаруживается на 40-50% раньше по времени от общей длительности сборки. Средний же тест обнаруживается раньше на 40-60%, а последний упавший тест раньше на 70-78%.

В реальности это позволило бы реагировать на падения первых тестов быстрее на половину времени сборки, а узнавать про падения всех тестов быстрее на 75%. Данный результат показывает эффективность разработанного подхода приоритизации, что подчеркивает важность его реализации в проекте API тестирования ВКонтакте в качестве первоочередного задания, вместо исследования более сложных и точных моделей.

4. Внедрение системы в проект API тестов ВКонтакте

4.1. Изменения в перезапусках тестов

Проведённая работа в ВКонтакте связана с модернизацией текущего процесса обработки тестов. Ранее используемая схема перезапуска тестов после завершения всего набора (сьюта) тестов приводила к задержке получения информации об упавших тестах до окончания процесса сборки. Для решения этой проблемы, схема была изменена (рис. 5) на мгновенные перезапуски тестов с задержкой 5 секунд между ними. Это позволяет получать информацию об упавших тестах в середине процесса сборки.

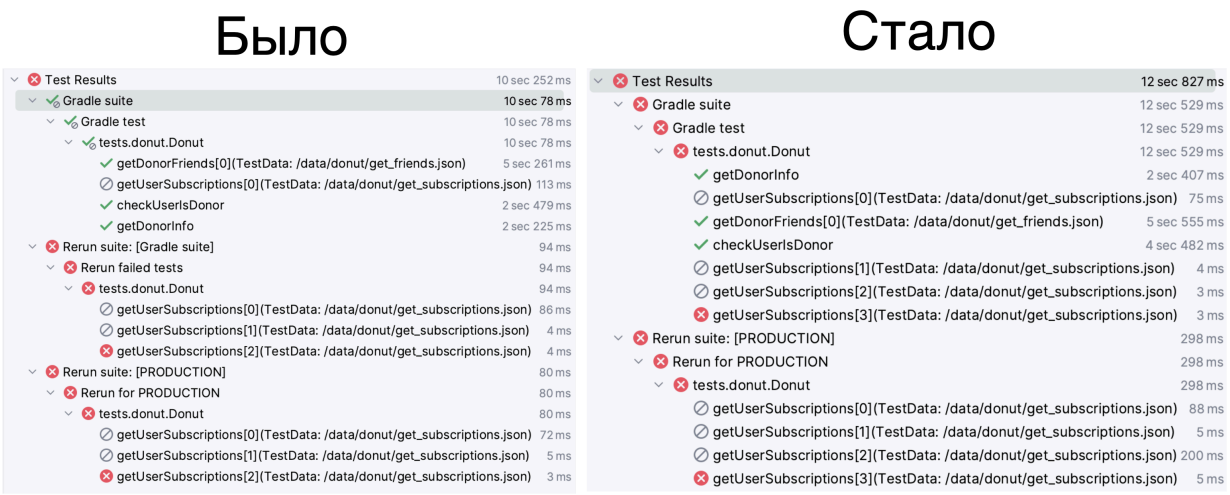

Рис. 5: Изменения в перезапусках тестов

4.2. Изменения в генерации отчета

Однако, проблема генерации отчёта оставалась актуальной — отчет генерировался средствами TeamCity только по завершении сборки (рис. 6). Сделать это ранее на стороне TeamCity невозможно, из-за чего требовалось реализовать другое решение.

Было предложено два решения: первое включает пересылку информации об упавших тестах на отдельный сервер, который генерирует веб-

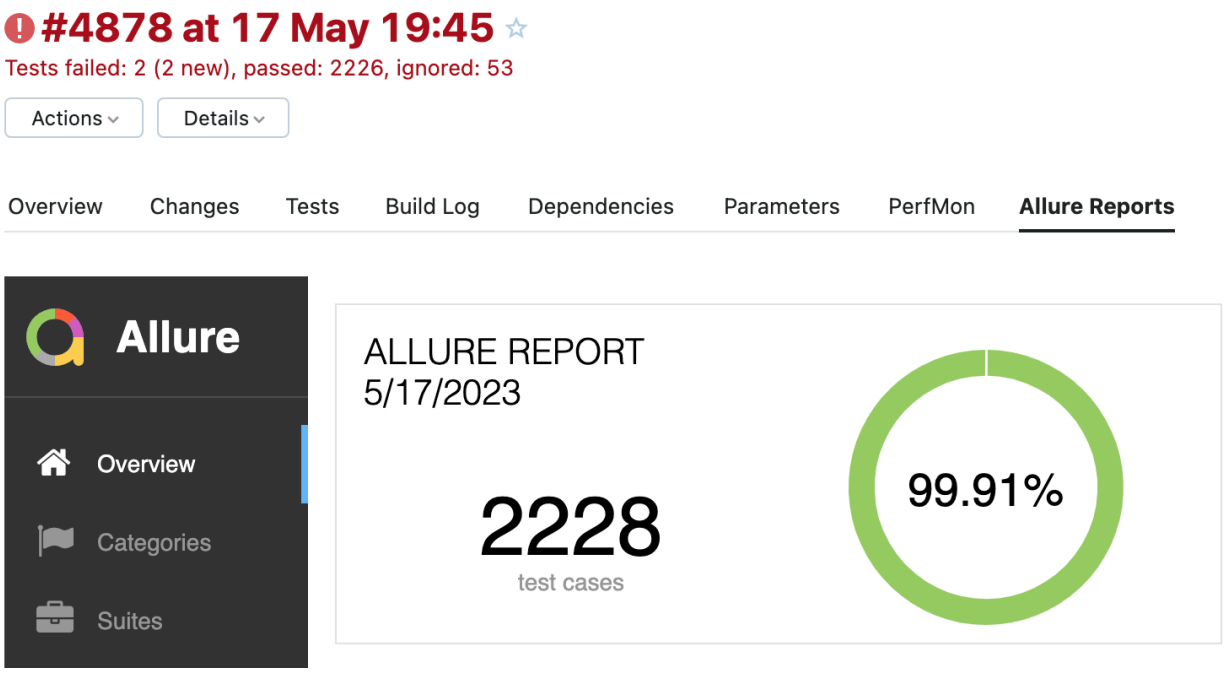

Рис. 6: Отчет Allure в конце сборки TeamCity

страницу с отчетом Allure и предоставляет её на определенном порту. Сервер обновляет страницу с отчётом по мере появления новых упавших тестов и способен обрабатывать до десяти параллельно происходящих сборок, что в данный момент является достаточным.

Второй вариант предполагал использование Allure Testops [8], которое предоставляет аналогичные возможности ”из коробки”. Однако, при использовании этого решения мы столкнулись с проблемами.

Первая проблема связана со слишком высокой нагрузкой на сеть при выгрузке всех логов и прикрепленных файлах каждого теста, вследствие чего время выгрузки отчета увеличилось до 20 минут, что сопоставимо с длительностью самой сборки, и поэтому недопустимо. Проблема была решена удалением отладочной информации о прошедших тестах. Из-за того что падающих тестов в среднем значительно меньше, чем успешно прошедших, мы добились уменьшения объема пересылаемой информации по сети в более, чем 100 раз. При этом удаленная отладочная информация не используется дежурным тестировщиком, поэтому её раннее получение бесполезно. Получить эту информацию остается возможным, но только после завершения всей сборки, как раньше.

Вторая проблема — потеря возможности отслеживания истории тестов, вызванная расхождением в определении уникальности тестовых кейсов, так и осталась нерешенной. Testops идентифицирует одни и те же тестовые методы, выполненные при помощи различных тестовых аккаунтов ВКонтакте, как разные, поэтому с помощью него невозможно оперативно отследить падения одного и того же теста на протяжении нескольких сборок. В связи с этим, было решено временно использовать первый вариант решения через выделенный сервер, пока вторая обозначенная проблема не будет решена разработчиками самого Allure Testops.

4.3. Изменения в уведомлении дежурного

Существовавшая система уведомления (рис. 7), так же как и генерация отчета, осуществлялась через TeamCity только по окончании сборки. Такие уведомления не позволяют получить информацию об упавших тестах в процессе сборки, поэтому была реализована другая и более умная система уведомлений.

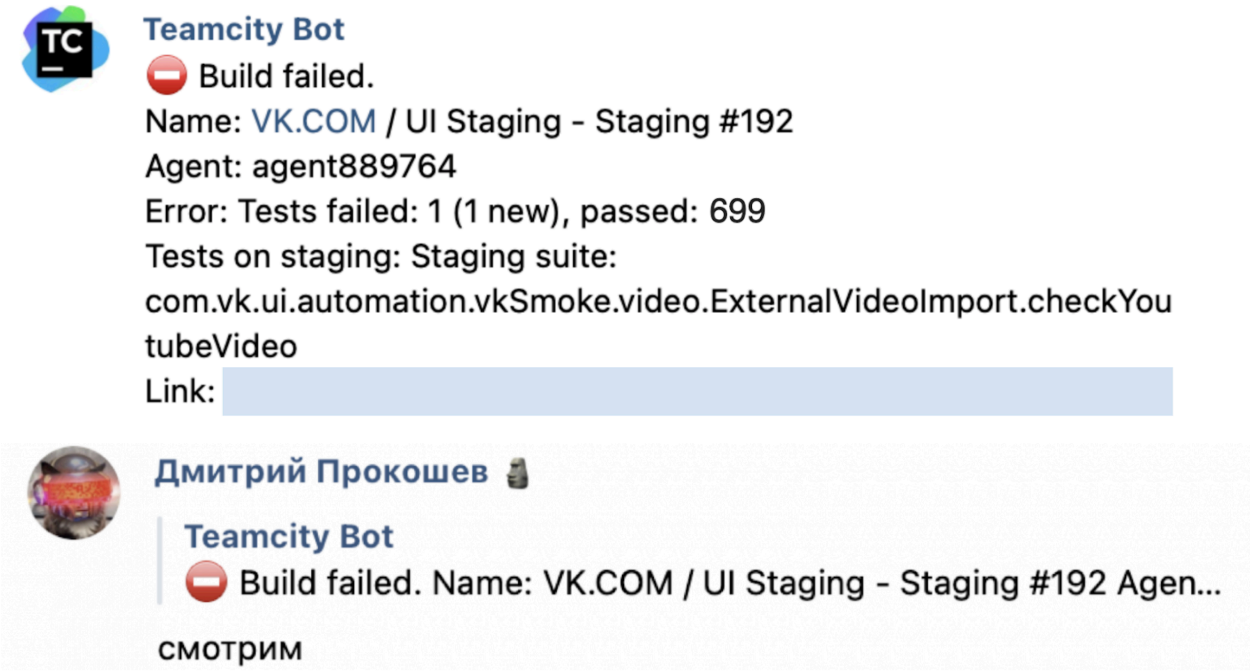

Рис. 7: Уведомление о падении сборки

Новые уведомления приходят в реальном времени, то есть через несколько секунд после фактического падения теста. Сообщения с информацией о падении теста (рис. 8) передаются через социальную сеть ВКонтакте ответственному за тестирование специалисту. При этом сообщение включает в себя все данные, необходимые для анализа проблемы и принятия дальнейших действий: название теста, ссылку на сборку, отчет Allure и другую сопутствующую информацию. Был также предусмотрен важный аспект оптимизации функционала уведомлений. Падения тестов, произошедшие по одинаковым причинам или зарегистрированные в течение одной минуты, группируются в одно сообщение. Такой подход исключает большое количество уведомлений, предотвращая избыточное количество сообщений и поддерживая порядок и чистоту рабочего чата.

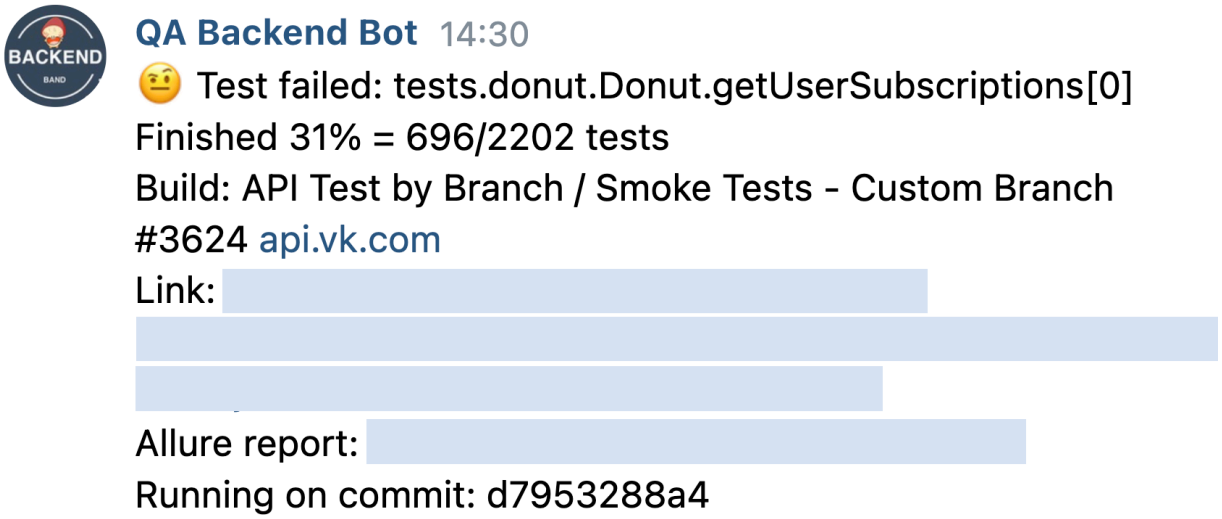

Рис. 8: Новые уведомление о падении теста

В такой же функциональности заинтересованы другие проекты тестов внутри ВКонтакте, например Android тесты и UI тесты, поэтому задачей было сделать встраиваемую реализацию. Она представляет из себя JVM библиотеку, которая позволяет подключить функциональность уведомлений одной строкой кода.

Новая система уведомлений позволяет получать информацию о падении тестов через секунды после происшествия, что на 10 минут быст-

рее, чем предыдущие уведомления.

4.4. Изменения в порядке тестов

Приоритизация реализована в той же JVM библиотеке, что и система уведомлений. Существовали другие варианты реализации, например, плагин для TeamCity. Однако, несмотря на наличие опыта разработки плагина, он представлял собой определенную угрозу безопасности, поскольку его установка на сервер, которым пользуются тысячи разработчиков, давала плагину достаточно прав, чтобы потенциально нарушить работу сервера. Как и плагин, библиотека может быть легко интегрирована в другие JVM проекты. Это решение не подходит для тестов на Python и PHP, но на данный момент в этом нет необходимости, поэтому выбор был сделан в пользу библиотеки. Вариант с использованием микросервиса был отклонён из-за его сложности в рамках временных ограничений.

В основе приоритизации лежит ранее разработанный универсальный подход, а также было запланировано улучшение модели специально для API тестов, с учётом покрытия методов API.

Каждый тест делает хотя бы один запрос к API ВКонтакте. С помощью изменений в тестовый фреймворк можно запомнить все запросы каждого тестового метода. Таким образом можно выявить, от каких методов API зависит успешность определенного теста. Далее, при обновлении API от одной версии к другой, необходимо проанализировать изменения и составить список измененных методов. Зная их, можно поднять приоритет тестам, завсящим от этих методов. Здесь можно также использовать модель наивного Байеса, как и в случае с измененными файлами.

Итоговое решение выглядит следующим образом (рис. 9): библиотека интегрируется в проект API тестов, предоставляя два класса, которые можно подключить одной строкой кода. Каждый из этих классов использует TestNG, VK API, TeamCity API и выделенный сервер для

реализации процесса тестирования, описанного ниже.

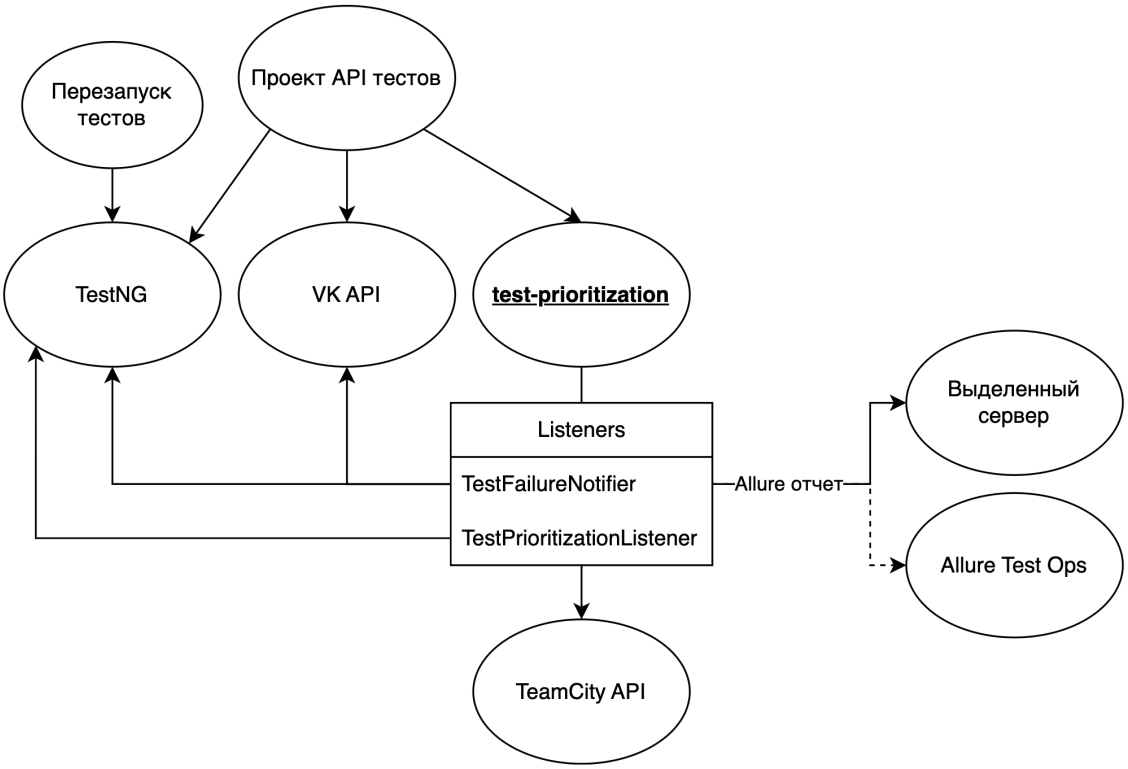

Рис. 9: Схема взаимодействия компонентов системы

В ходе автоматизированного процесса тестирования на TeamCity (рис. 10), производится запуск набора Acceptance тестов, состоящих из 2200 обязательных к прохождению тестов. Непосредственно перед инициализацией тестового набора TestNG, класс, отвечающий за приоритизацию, обрабатывает набор тестов, упорядочивая их в соответствии со статистическими данными, полученными из TeamCity. Эта статистика включает в себя историю падений тестов и измененных файлов.

В результате этого процесса, упорядоченный набор тестов предоставляется для выполнения в TestNG. После начала исполнения, каждый упавший тест обрабатывается отдельным классом, который отправляет данные об этом событии на специальный сервер. На этом сервере формируется отчет Allure, который становится доступным для просмотра через определенный порт.

После этого система автоматически генерирует уведомление о падении теста, содержащее информацию о соответствующем отчете Allure и другую информацию, и отправляет его сообщением ВКонтакте. Это

позволяет обеспечить своевременное уведомление дежурного о возникших проблемах и ускоряет процесс их решения.

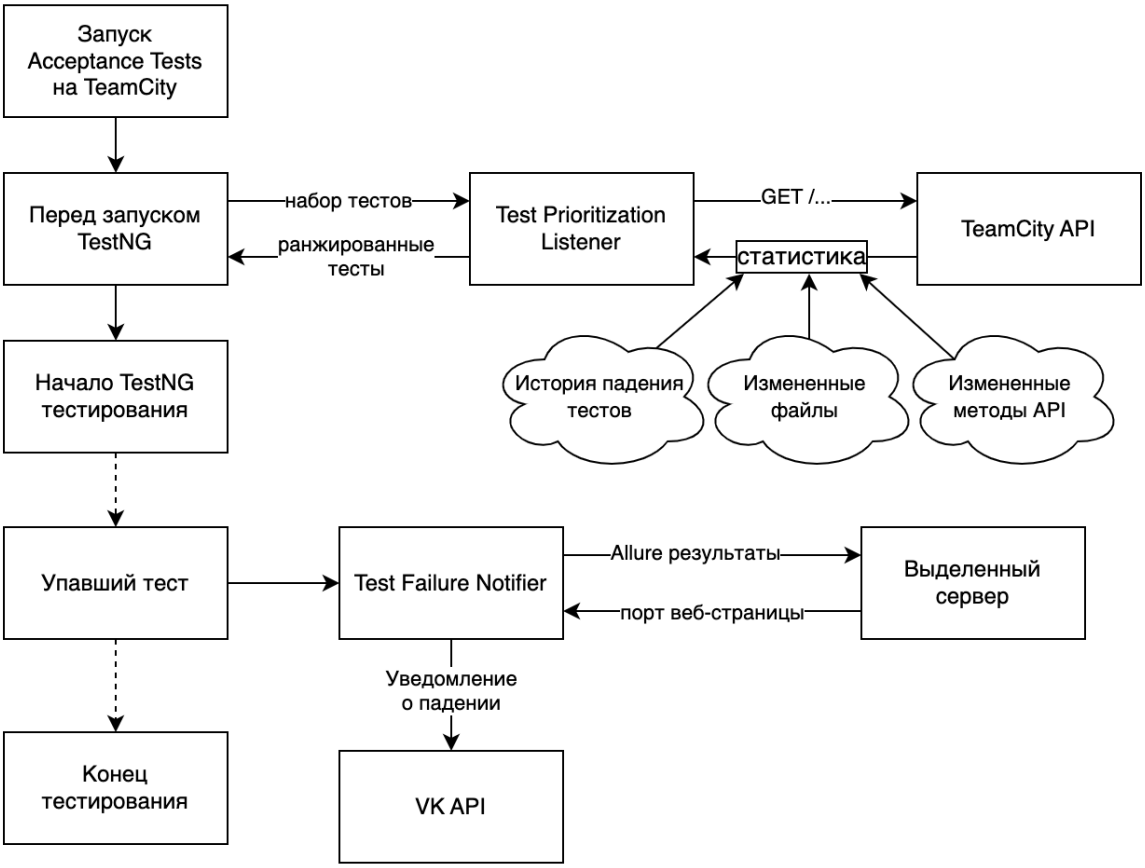

Рис. 10: Диаграмма потока данных

4.5. Оценивание эффективности системы

Для оценки эффективности внедренной системы были получены метрики, усредненные на 120 реальных сборках (таб. 2). Time показывает среднюю длительность сборки формата в формате минуты:секунды. Значения строк First/Avg/Last показывают время до уведомления дежурного тестировщика о падении первого/среднего/последнего теста соответственно.

Столбцы таблицы соотвествуют трем конфигурациям сборки: без представленной в этой работе системы, с подключенными быстрыми уведомлениями, а также с подключенными уведомлениями и приори-

тизацией. Время до получения информации о первом упавшем тесте сокращается на 19 минут, что делает уведомления в 10 раз быстрее, чем ранее. Узнать о падении всех тестов с системой приоритизации и быстрого уведомления становится возможным в среднем в 3 раза быстрее, чем без нее.

	Было	С уведомлениями	С приоритизацией
Time	**21:10**	**21:10**	**21:10**
First	21:10	07:33	02:17
Avg	21:10	11:47	04:54
Last	21:10	16:22	07:40

Таблица 2: Метрики эффективности приоритизации

4.6. Выводы и результаты по главе

Основные выводы данной главы можно сформулировать в следующих пунктах:

- Переделан механизм перезапуска тестов: Механизм повторного прогона тестов был оптимизирован и перепроектирован для мгновенных перезапусков с промежутком в 5 секунд между попытками. Это позволяет получать информацию об упавших тестах гораздо раньше, чем при предыдущем подходе, который предполагал перезапуск тестов только после завершения всего набора.
- Создана система генерации и хранения отчетов: Разработана и внедрена система, которая генерирует отчеты о проведении тестов в реальном времени. Отчеты доступны для просмотра на определенном порту выделенного сервера сразу после генерации, что значительно упрощает процесс отслеживания прогресса и результатов тестирования.

- Реализован умный механизм оповещения дежурного: Создана система уведомлений, которая автоматически информирует дежурного о любых упавших тестах. Оповещения группируют одинаковые падения и падения, произошедшие в течение одной минуты. Это позволяет дежурному быстро оценить общую картину и реагировать на проблемы в реальном времени.

- Внедрена приоритизации тестов: Реализован подход, который позволяет приоритизировать тесты на основе статистики. Это приводит к тому, что тесты, имеющие большую вероятность падения, запускаются раньше, что позволяет быстрее получить информацию о возможных проблемах и немедленно приступить к их решению.

- Собраны численные метрики эффективности системы: Метрики показывают эффективность системы приоритизации и быстрого уведомления о падении тестов по сравнению со стандартным подходом, ускоряя получение информации об упавших тестах в 3-7 раз.

Заключение

В ходе данной работы была успешно проведена разработка и внедрение модели приоритизации тестов и уведомления о падении тестов в реальной времени внутри проекта API тестов ВКонтакте.

Ключевые результаты работы:

- Разработка модели приоритизации: Разработана эффективная и универсальная модель приоритизации, которая может быть применена к различным тестовым фреймворкам и языкам программирования.
- Тестирование модели при помощи плагина к TeamCity: Разработанный подход был успешно протестирован и оценен на основе реальных данных с использованием плагина к TeamCity. Это позволило демонстрировать практическую применимость и эффективность разработанной модели приоритизации.
- Внедрение модели приоритизации и быстрых уведомлений в API тесты ВКонтакте: Разработанный подход успешно внедрен в промышленный проект по тестированию API ВКонтакте. Был также реализован механизм быстрых уведомлений, значительно улучшающий оперативность реакции на проблемы. Численные метрики показывают эффективность внедренной системы.

В перспективе дальнейшей деятельности предполагается:

- Внедрение разработнный библиотеки в другие проекты тестирования ВКонтакте, такие как Android тесты и UI-тесты.
- Проведение дополнительных исследований и экспериментов для уточнения и оптимизации модели приоритизации. Это может включать использование более сложных и точных статистических моделей, учет дополнительных факторов и т.д.

- Развитие системы быстрых уведомлений, например, с добавлением более детальных отчетов и интеграции с другими системами уведомлений.

Список литературы

[1] Beust Cédric. TestNG. — URL: https://testng.org/doc/ (online; accessed: 23.05.2023).

[2] Gradle, Inc. Gradle Enterprise Predictive Test Selection. — URL: https://gradle.com/gradle-enterprise-solutions/predictive-test-selection/ (online; accessed: 23.05.2023).

[3] Jetbrains s.r.o. Teamcity Open-source projects. — URL: https://teamcity.jetbrains.com/ (online; accessed: 23.05.2023).

[4] Launchable, Inc. Launchable Predictive Test Selection. — URL: https://www.launchableinc.com/ (online; accessed: 23.05.2023).

[5] Meta Platforms, Inc. Jest. — URL: https://jestjs.io/ (online; accessed: 23.05.2023).

[6] Predicting Test Case Verdicts Using Textual Analysis of Commited Code Churns / Khaled Al-Sabbagh, Miroslaw Staron, Regina Hebig, Wilhelm Meding // International Conference on Software Process and Product Measurement, 2019.

[7] Predictive Test Selection / Mateusz Machalica, Alex Samylkin, Meredith Porth, Satish Chandra // 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP).

[8] Qameta Software, Inc. Allure Testops. — URL: https://qameta.io/ (online; accessed: 23.05.2023).

[9] Red Hat, Inc. Quarkus. — URL: https://github.com/quarkusio/quarkus (online; accessed: 23.05.2023).

[10] Shi August, Zhao Peiyuan, Marinov Darko. Understanding and Improving Regression Test Selection in Continuous Integration //

2019 IEEE 30th International Symposium on Software Reliability Engineering.

[11] Shvarkunov Mikhail. Как обеспечивать качество при релизах раз в час // Хабрахабр. — 2022. — URL: https://habr.com/ru/companies/vk/articles/703230/ (online; accessed: 23.05.2023).

[12] The Apache Software Foundation. Apache Dolphin Scheduler. — URL: https://github.com/apache/dolphinscheduler (online; accessed: 23.05.2023).

[13] The Apache Software Foundation. Apache Flink. — URL: https://github.com/apache/flink (online; accessed: 23.05.2023).