**ПРЕДЛОЖЕНИЯ**

**по развертыванию и настройке удаленного репозитария**

**для разработки, хранения и автоматизации запуска сценарных и дымовых тестов**

1. **Этап 1. Внедрение git и связанных с ним технологий.**

На первом этапе для разработки и хранения файлов тестов, а также исходного кода обработчиков тестов предлагается:

- установить и настроить корпоративный git-сервер (в случае невозможности сделать это – можно использовать корпоративный аккаунт на github);

- создать репозитарий, где ветка master будет защищена от коммитов контрибьютерами проекта (обычные разработчики и консультанты). Изменения в ветку master будут приниматься только мэйнтейнером или владельцем (owner) проекта (архитектор или ведущий разработчик) через запросы на слияние (pull request или merge request);

- разработку тестов вести по технологии git-flow (ветвление). Это означает: новый тест – новая ветка, исправление существующего теста – новая ветка;

- установить стандарты именования веток и файлов тестов (фича-файлов). Пример для имени ветки: *«feature – Краткая суть теста - № задачи в трекере»*. Также *«fix – Краткая суть исправления - № задачи в трекере»*. Для имен тестов (фича-файлов): *«Раздел учета – Краткая суть теста - № задачи в трекере»*;

- провести обучение сотрудников Заказчика средствами и силами Подрядчика по работе со следующим стеком технологий:

- git и основные команды для работы с ним;

- командная работа в git «Аналитик (консультант) – Разработчик – Архитектор (Код-ревьюер), методология git-flow;

- git-клиенты (Source Tree и аналоги), работа с git через графический интерфейс;

- Visual Studio Code и основные плагины для комфортной работы;

- язык формализации бизнес-требований в сценарии Gherkin;

- свободный интерпретатор языка BSL – OneScript и основные библиотеки, нужные в работе (их настройки);

- BSL Languaege Server - как настроить в VSC и получать пользу от статического анализа программного кода;

- веб-сервер графических отчетов Яндекс Allure – как настроить и пользоваться;

- фреймворк сценарного тестирования Vanessa ADD – как писать тесты, методология правильного создания тестов, покрытие сценариев программным кодом;

- методологии ldd и tdd (покрытие тестами legasy-функционала и разработка нового функционала «через тестирование»);

1. **Этап 2. Автоматизация запуска сценарных и дымовых тестов с помощью сервера сборок.**

После освоения сотрудниками Заказчика вышеперечисленных технологий, а также после накопления в репозитарии сколько-нибудь значимого количества тестов предлагается:

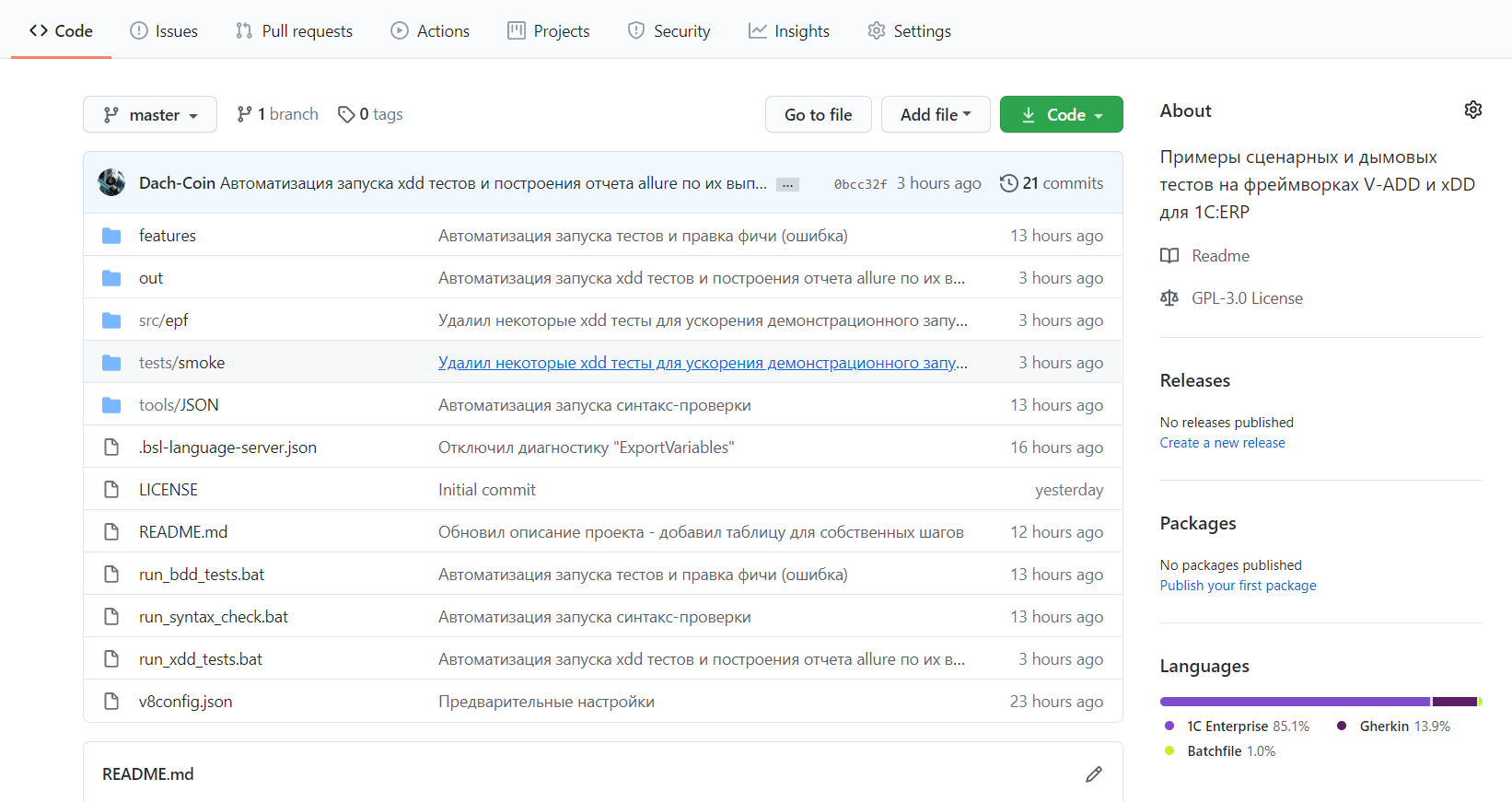
- перенос в данный репозитарий технологии Заказчика по сборке релизов (из ветки dev в ветку master). Автоматизация этого процесса с помощью существующей обработки для переносов коммитов хранилища 1С (cherry-pick);

- создание и наладка основной сборочной линии («пайп-лайна») для автоматизации подготовки релиза («деплоя») на выбранном Заказчиком сервере сборок (GitLab CI, Jenkins, TeamCity и т.д.): Создание файла поставки из ветки master, обновление данным файлом тестовой ИБ, запуск сценарных и дымовых тестов, формирование артефактов сборки для отчетов Allure и Junit;

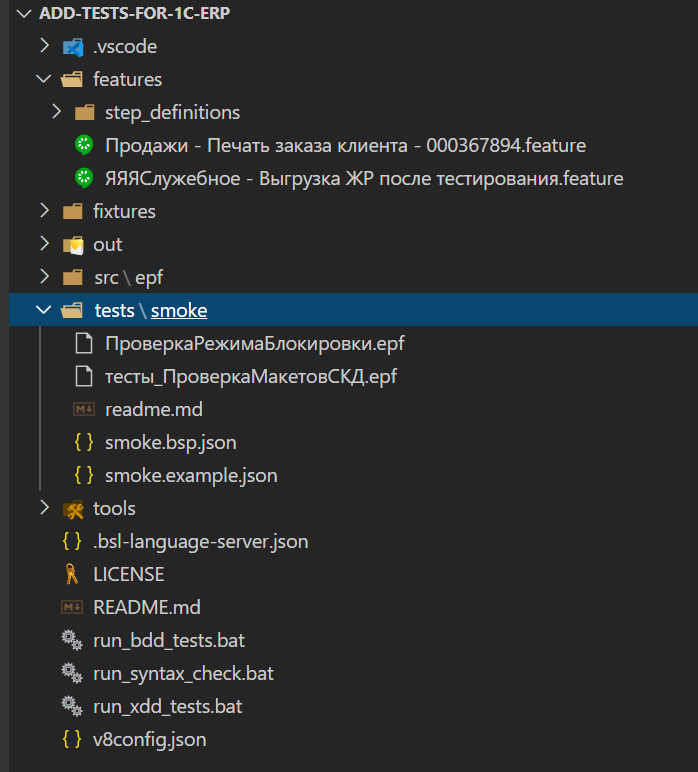
- опционально – разворачивание и настройка сервера статического анализа программного кода SonarQube, включение статического анализа в основную сборочную линию;

**ПРИМЕР РЕАЛИЗАЦИИ**

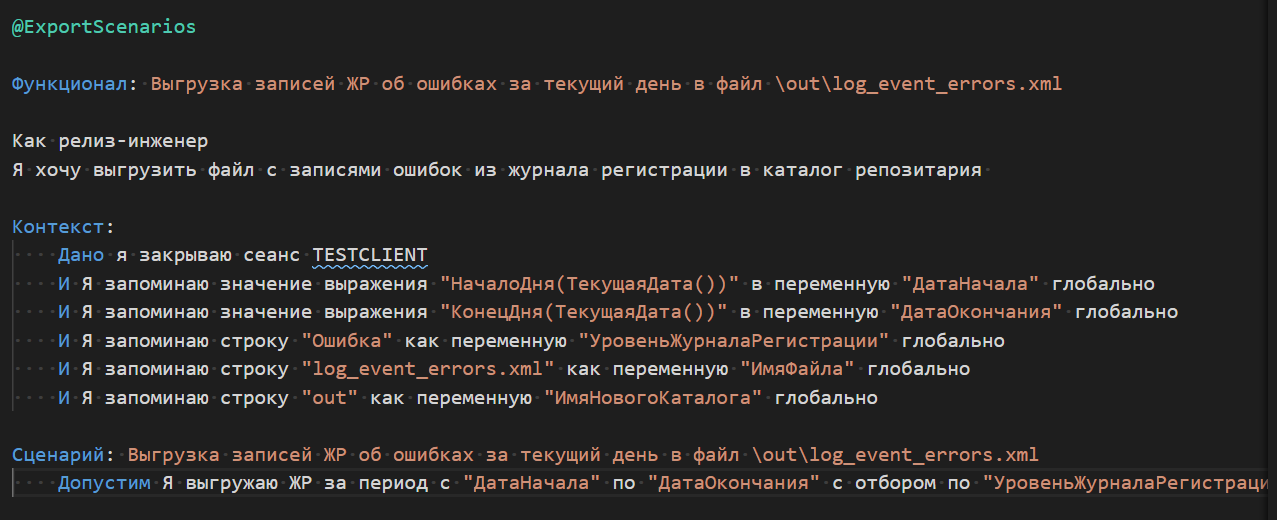
1. Приватный репозитарий-пример на github:



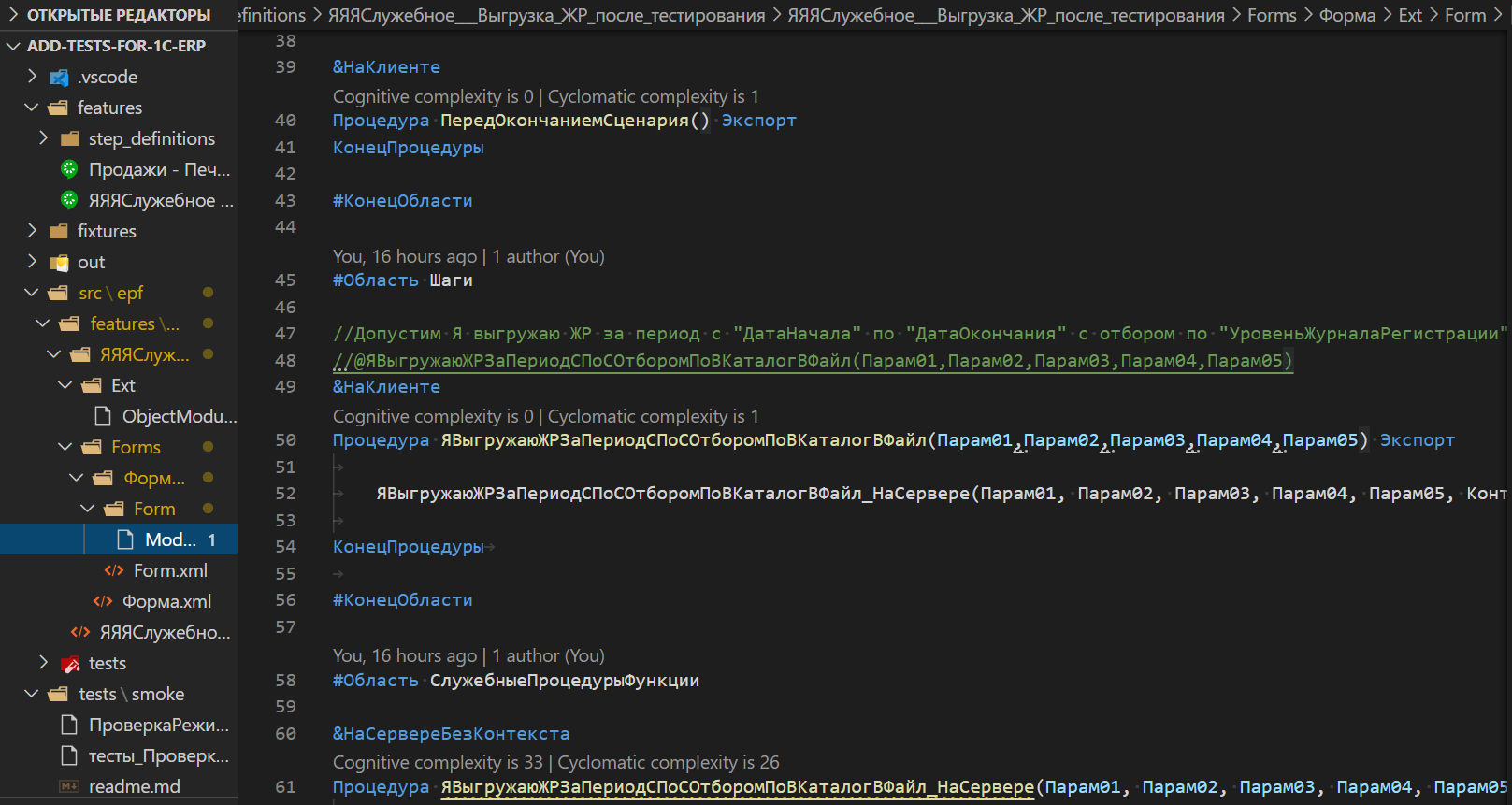
1. Состав файлов репозитария:



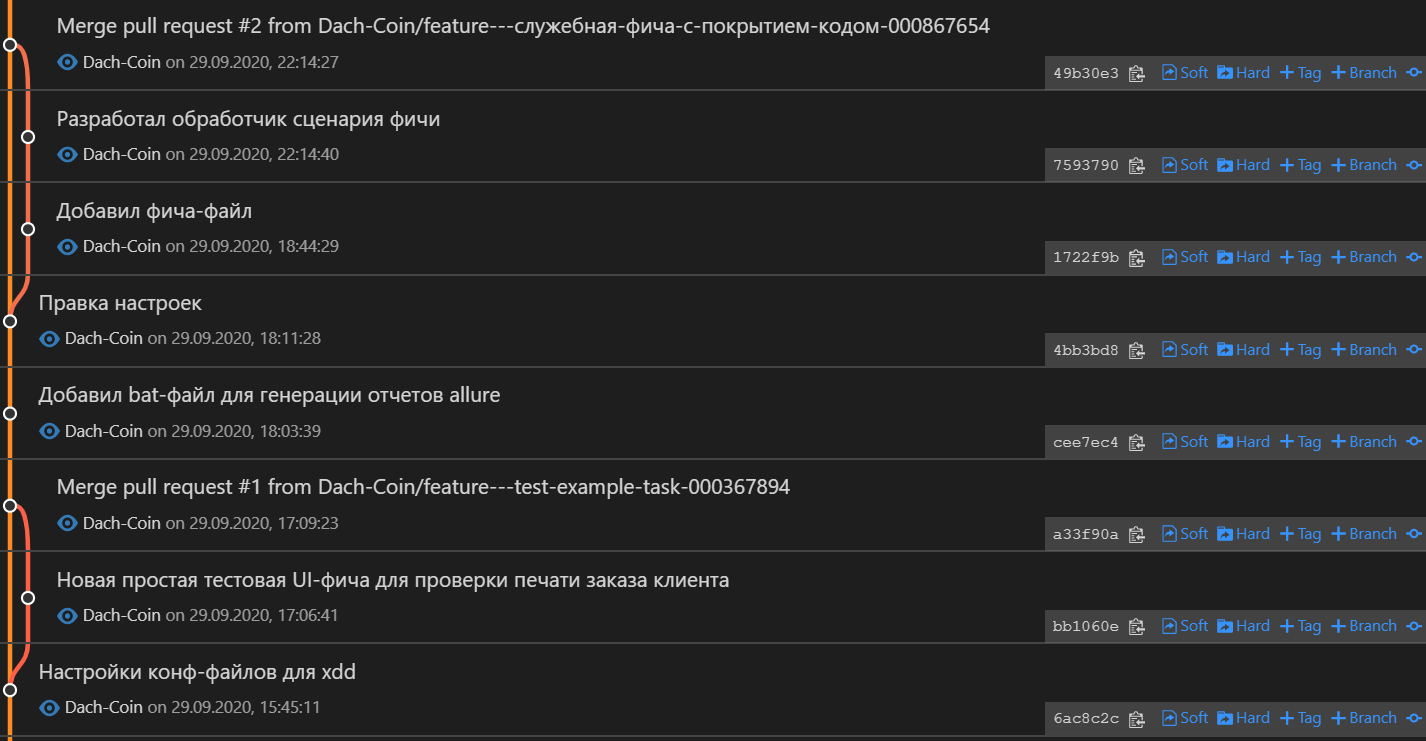
1. Фича-файлы (тесты), пример:



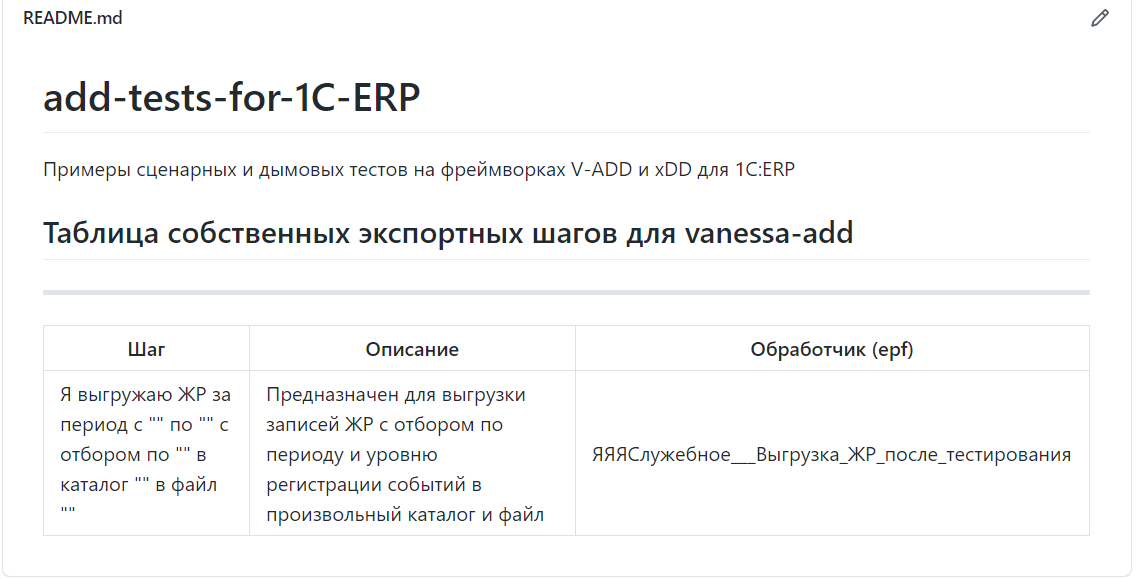
1. Исходный код обработчиков тестов:



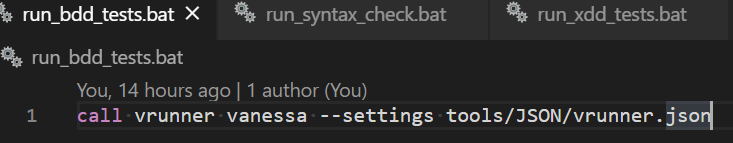
1. Технология git-flow применительно к разработке тестов:

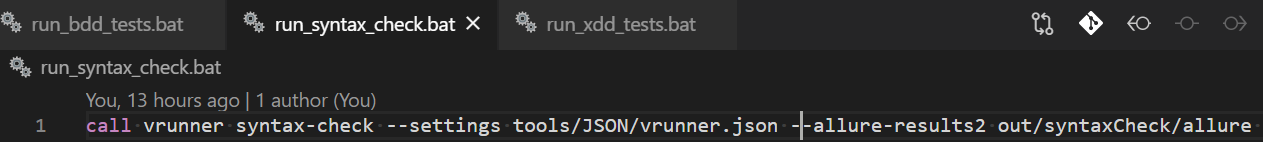


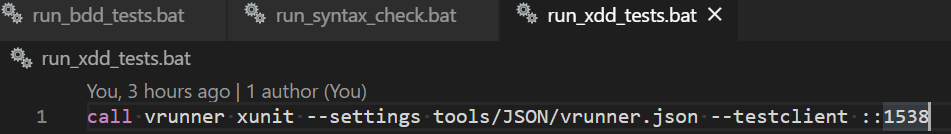
1. Стандартизация и хранение описания собственных сценарных шагов в readme проекта:



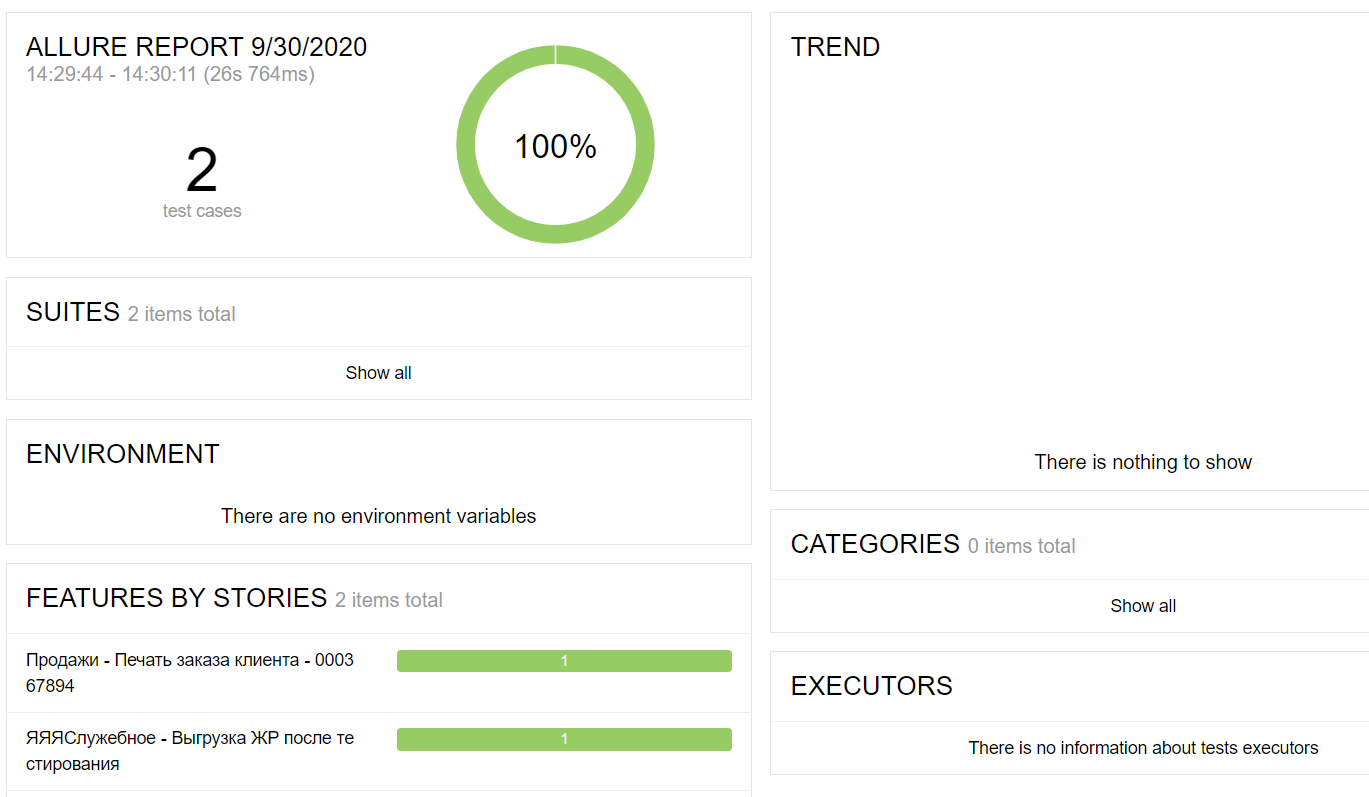
1. Скрипты для автоматизации запуска тестов и прочих операций (с применением библиотеки OneScript – vanessa-runner)

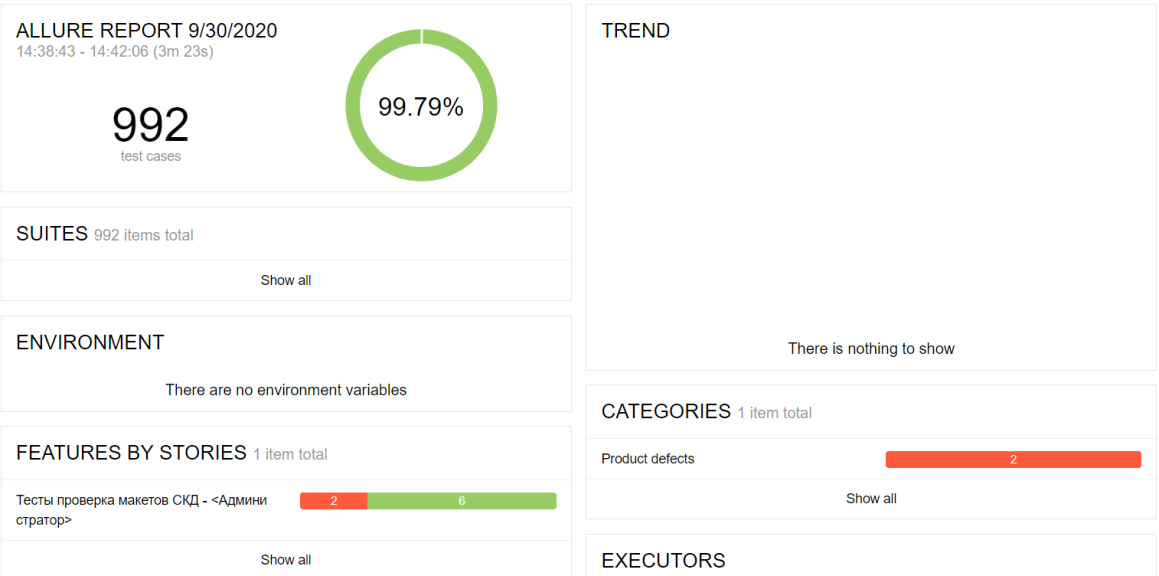




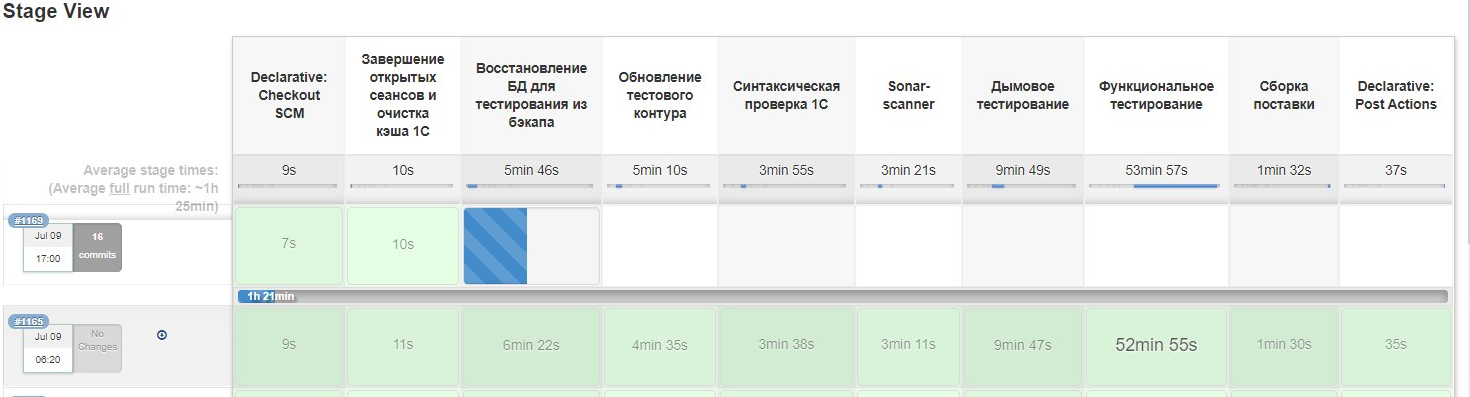


1. Результаты запуска тестов в Allure:





**Возможная реализация сборочной линии на сервере сборок Jenkins**



**МЕТОДОЛОГИЯ И ОПИСАНИЕ ПОНЯТИЯ «GIT-FLOW»**

Неплохое описание git-flow дано здесь:

<https://habr.com/ru/post/106912/>

<https://habr.com/ru/post/159107/>

Однако, мы рассмотрим более простой и приближенный к реалиям пример написания и хранения файлов тестов.

Представим, что у нас в отделе автоматизации есть сотрудники со следующими ролями: Аналитик (бизнес-консультант), Разработчик (программист), Архитектор (ведущий разработчик, «тимлид»).

Также примем следующие исходные данные. В ветке master нашего репозитария хранятся только заведомо рабочие тесты, выполняющиеся на «эталонной» ИБ. Эталонная ИБ – это база, не предназначенная для разработки, она создана путем копирования из продуктивной ИБ и усечения в ней данных (по необходимости), для уменьшения размера и облегчения восстановления из резервной копии (усечение должно быть выполнено с сохранением прикладной целостности данных). Конфигурация эталонной базы перед любым запуском любого теста всегда соответствует самой последней версии хранилища разработки 1С (или ветки dev нашего репозитария, если планируется собирать конфигурацию из нее). Если конфигурация не соответствует – она обновляется и только после этого запуск тестов можно считать выполненным верно с точки зрения методологии.

В отделе используется git-сервер, все сотрудники отдела обладают базовыми компетенциями по работе с ним.

**Пример 1. Разработчик получил задачу написать тест на уже существующий и критичный для компании функционал.**

Действия Разработчика:

- обновить в своем локальном репозитарии ветку master;

- «отпочковать» (или склонировать) от ветки master новую ветку в своем локальном репозитарии с названием «feature-суть теста-№задачи»;

- написать фича-файл на языке gherkin и epf-обработчик этого теста;

- убедиться в успешном выполнении теста на своей локальной базе, подключенной к хранилищу разработки (и обновленной из него), внести правки, если это необходимо;

- опубликовать свою ветку в удаленном центральном репозитарии и оформить запрос на слияние в ветку master (MR или PR);

Действия Архитектора:

- получив уведомление на e-mail о необходимости принять новый MR (PR) – прочитать имя ветки, найти задачу в трекере и понять суть проблемы;

- в своем локальном репозитарии перейти на опубликованную разработчиком ветку «feature-суть теста-№задачи». Состав файлов в локальном репозитарии Архитектора станет точно такой же, как у Разработчика;

- убедиться, что новый тест соответствует сути задачи и действительно выполняет покрытие функциональности;

- проверить, что разработанный тест успешно выполняется в эталонной базе (или в ее копии, если в эталонной базе не предполагается никакая работа, кроме автозапусков тестов);

- сделать Разработчику замечания в треде (переписка и комментарии с ответами) MR (PR), если тест не выполняется. Дождаться новых коммитов от Разработчика с исправлениями по замечаниям, получить эти коммиты в свой локальный репозитарий в данную ветку и выполнить проверку еще раз;

- так как тест новый – конфликтов при слиянии не будет и если тест выполняется – принять MR (PR).

В итоге в ветке master появится новый тест, который можно будет запускать вместе с остальными тестами по расписанию в эталонной базе.

**Пример 2. Аналитик получил задачу спроектировать новую функциональность и сразу покрыть ее тестом. Разработчик также назначен на эту задачу и ему предстоит техническая реализация функциональных и бизнес-требований.**

Действия Аналитика:

- обновить в своем локальном репозитарии ветку master;

- «отпочковать» (или склонировать) от ветки master новую ветку в своем локальном репозитарии с названием «feature-суть теста-новый-№задачи»;

- написать на языке gherkin тест на еще не существующую функциональность. Фактически, Аналитик формализует бизнес-требования по конкретной задаче в виде сценариев, описанных на языке gherkin. Такой тест кончено же будет «падать», так как функциональность еще не существует, поэтому в шапке теста ставится специальный тег «draft», который отключает его автовыполнение;

- опубликовать ветку и оформить MR (PR);

Действия Архитектора:

- получив уведомление на e-mail о необходимости принять новый MR (PR) – прочитать имя ветки, найти задачу в трекере и понять суть проблемы;

- в своем локальном репозитарии перейти на опубликованную разработчиком ветку «feature-суть теста-№задачи». Состав файлов в локальном репозитарии Архитектора станет точно такой же, как у Аналитика;

- убедиться, что новый тест соответствует сути задачи и действительно отражает реализацию бизнес-требований;

- убедиться, что в шапке теста есть тег «draft»;

- принять MR (PR) в ветку master;

- в треде MR (PR) можно упомянуть Разработчика и Аналитика по стандартному шаблону: @Аналитик, @Разработчик. Аналитик и Разработчик получат уведомления на e-mail. Либо уведомить их о принятии MR (PR) другим способом.

Действия Разработчика:

- разработать новую функциональность, провести ручное тестирование;

- обновить в своем локальном репозитарии ветку master;

- «отпочковать» (или склонировать) от ветки master новую ветку в своем локальном репо с названием «feature-суть теста-разработка-№задачи»;

- написать epf-обработчик для заранее подготовленного Аналитиком фича-файла (теста). Внести коррективы в сам тест, если это необходимо. Не забыть убрать тег «draft», чтобы обеспечить автовыполнение теста в будущем;

- убедиться в успешном выполнении теста на своей локальной базе, подключенной к хранилищу разработки (и обновленной из него), внести правки, если это необходимо;

- опубликовать свою ветку в удаленном центральном репозитарии и оформить запрос на слияние в ветку master (MR или PR);

Действия Архитектора:

- полностью аналогичны финальным действиям из примера 1.

В итоге в ветке master появится новый тест на новую функциональность, который можно будет запускать вместе с остальными тестами по расписанию в эталонной базе.

Технология разработки в этом примере называется «Разработка через тестирование» или «TDD – test driven development».