

Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности

Работа допущена к защите
Руководитель ОП
_____ А.В. Щукин
« _____ » _____ 2024 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
РАБОТА БАКАЛАВРА
РАЗРАБОТКА ПЛАГИНА JENKINS ДЛЯ ВИЗУАЛИЗАЦИИ
СТАТИСТИКИ РАБОТЫ СБОРОК JENKINS

по направлению подготовки 09.03.03 Прикладная информатика
Направленность (профиль) 09.03.03_03 Интеллектуальные инфокоммуникацион-
ные технологии

Выполнил
студент гр. з5130903/90301

А.Д. Кухто

Руководитель
ст. преподаватель ВШ ПИ

В.А. Пархоменко

Консультант
по нормоконтролю

В.А. Пархоменко

Санкт-Петербург
2024

**САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО**

Институт компьютерных наук и кибербезопасности

УТВЕРЖДАЮ

Руководитель ОП

_____ А.В. Щукин

« _____ » _____

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

студенту Кухто Андрею Денисовичу гр. з5130903/90301

1. Тема работы: Разработка плагина Jenkins для визуализации статистики работы сборок Jenkins.
2. Срок сдачи студентом законченной работы: 09.01.2024.
3. Исходные данные по работе: документация по Jenkins [3.2], Java [3.1], TeamCity [3.3].
 - 3.1. Java Docs. — URL: <https://docs.oracle.com/en/java/> (visited on 02.10.2023).
 - 3.2. Jenkins Documentation. — URL: <https://www.jenkins.io/doc/> (visited on 02.10.2023).
 - 3.3. TeamCity Docs. — URL: <https://www.jetbrains.com/help/teamcity/teamcity-documentation.html> (visited on 02.10.2023).
4. Содержание работы (перечень подлежащих разработке вопросов):
 - 4.1. Обзор инструментов сборки для совместной работы.
 - 4.2. Исследование плагинов визуализации статистики работы сборки Jenkins.
 - 4.3. Разработка плагина визуализации статистики работы сборки для системы Jenkins.
 - 4.4. Тестирование и апробация плагина в системе Jenkins.
5. Перечень графического материала (с указанием обязательных чертежей):
 - 5.1. Use-case, Class и IDEF0 диаграммы.
 - 5.2. Архитектура разработанного плагина.
6. Консультанты по работе:

6.1. Ст. преподаватель ВШ ПИ, В.А. Пархоменко (нормоконтроль).

7. Дата выдачи задания: 02.10.2023.

Руководитель ВКР _____ В.А. Пархоменко

Задание принял к исполнению 02.10.2023

Студент _____ А.Д. Кухто

РЕФЕРАТ

На 59 с., 20 рисунков, 4 таблицы, 4 приложения

КЛЮЧЕВЫЕ СЛОВА: ВИЗУАЛИЗАЦИЯ, СТАТИСТИКА, JENKINS, ПЛАГИН, СБОРКА, МЕТРИКИ СБОРКИ.

Тема выпускной квалификационной работы: «Разработка плагина Jenkins для визуализации статистики работы сборок Jenkins».

Целью работы является разработка плагина Jenkins для визуализации статистики работы сборок в инструментах совместного использования. **Объект исследования** — инструменты для сборки приложений. **Предмет исследования** — визуализация статистики работы сборок. Основными **методами** проведения работы являются методы сравнительного анализа аналогичных решений и методы объектно-ориентированного программирования. В результате работы разработан прототип плагина для визуализации статистики работы сборок Jenkins. Для визуализации и обработки метрик сборок применялись статистические показатели, разные типы диаграмм, а также анализ данных для прогнозирования значения метрик. Проведена апробация и тестирование разработки на реальном проекте с открытым исходным кодом в системе Jenkins. **Область применения** разработанного плагина - промышленная разработка программных продуктов, которые собираются и тестируются с использованием CI/CD инструмента Jenkins.

ABSTRACT

59 pages, 20 figures, 4 tables, 4 appendices

KEYWORDS: VISUALIZATION, STATISTICS, JENKINS, PLUGIN, BUILD, BUILD METRICS.

The subject of the graduate qualification work is «Title of the thesis».

The purpose of the work is to develop a Jenkins plugin for visualizing build statistics in sharing tools. **The object** of study is tools for building applications. **The subject** of the study is visualization of assembly operation statistics. The main **methods** of carrying out the work are methods of comparative analysis of similar solutions and methods of object-oriented programming. As a result of the work, a prototype plugin was developed for visualizing statistics on the operation of Jenkins builds. To visualize and process assembly metrics, statistical measures, different types of charts, and data

analysis were used to predict the value of metrics. Approbation and testing of the development was carried out on a real open source project in the Jenkins system. **The scope of application** of the developed plugin is industrial development of software products that are assembled and tested using the Jenkins CI/CD tool.

СОДЕРЖАНИЕ

Введение	8
Глава 1. Анализ средств сборки и визуализации программного обеспечения	10
1.1. Анализ средств сборки программного обеспечения	10
1.2. Анализ средств CI/CD	11
1.3. Анализ существующих плагинов Jenkins по визуализации статистики сборок	14
1.4. Требования к разработке	16
1.5. Выводы	19
Глава 2. Проектирование архитектуры плагина	19
2.1. Модель системы	19
2.2. Архитектура Jenkins	22
2.3. Архитектура плагина	24
2.4. Языки программирования	25
2.5. Инструменты сборки	26
2.6. Библиотеки	26
2.6.1. Chart.js	26
2.6.2. Gson	27
2.6.3. Apache Commons Math	27
2.6.4. Junit	28
2.6.5. Mockito	28
2.7. Выводы	28
Глава 3. Реализация прототипа плагина	28
3.1. Описание разработанных классов	29
3.1.1. BuildConfigurationStatisticsAction	29
3.1.2. DateTimeHandler	30
3.1.3. IntervalDate	32
3.1.4. Statistics	33
3.1.5. TimeInQueueFetcher	33
3.1.6. BuildLogic	33
3.1.7. BuildArtifactSizeLogic	34
3.1.8. BuildDurationLogic	34
3.1.9. BuildSuccessRateLogic	35
3.1.10. BuildTestCountLogic	36
3.1.11. BuildTimeQueueLogic	36

3.1.12. LinearRegressionHandler.....	37
3.1.13. Файлы JS и Jelly.....	37
3.2. Результаты разработки плагина	38
3.3. Выводы	40
Глава 4. Тестирование и апробация плагина в Jenkins	41
4.1. Методы тестирования	41
4.1.1. Unit тестирование.....	43
4.1.2. BDD тестирование	44
4.1.3. UI тестирование	44
4.2. Апробация плагина	45
4.2.1. Подготовка и генерация набора сборок для апробации.....	45
4.2.2. Апробация на проекте с открытым исходным кодом.....	49
4.2.3. Оценка результатов работы	53
4.3. Выводы	54
Заключение	56
Список использованных источников.....	57
Приложение 1. Программный код плагина	60
Приложение 2. Программный код тестов на языке Java	151
Приложение 3. Программный код ui тестов на языке Python	164
Приложение 4. Программный код для обработки результатов апробации ..	170

ВВЕДЕНИЕ

Сегодня разработка информационных систем достаточно сложный процесс, который состоит, как правило, из следующих основных этапов: анализ требований заказчика, проектирование системы, разработка, тестирование и доставка приложения потенциальному заказчику.

Для упрощения процесса разработки программного обеспечения в настоящий момент широко применяются практики DevOps, одной из которых является Continuous Integration, Continuous Delivery – CI/CD - непрерывная интеграция, сборка и доставка [27]. Существует множество средств CI/CD, которые применяются в промышленной разработке: TeamCity, Jenkins, Gitlab CI и другие.

Под *сборкой программного продукта* будем подразумевать процесс объединения отдельных файлов и компонентов программы в единый исполняемый файл или пакет, который включает в себя компиляцию, связывание модулей, оптимизацию и другие операции, необходимые для создания готового к выполнению приложения [11].

Будем различать понятие сборки программного продукта и *сборки* в инструментах CI/CD, таких как Jenkins, которые обычно используются *командой для совместной работы над одним проектом*. Сборка Jenkins — это набор задач, которые выполняются последовательно, как определено пользователем [9].

Одним из лучших средств CI/CD, в котором доступно много функций ”из коробки” является TeamCity компании JetBrains. TeamCity - мощный и сложный инструмент, который использовался крупными ИТ компаниями в промышленной разработке до 2022 года. Одним из главных недостатков TeamCity является то, что это платное решение, лицензия обходится ИТ компания достаточно дорого, также недостатком является то, что компания JetBrains покинула ИТ сектор РФ. Для того, чтобы преодолеть данные проблемы ИТ компании РФ начали поиск бесплатных средств с открытым исходным кодом. Одним из таких средств является Jenkins - средство CI, которое всегда пользовалось популярностью у разработчиков при локальной разработке решений с открытым исходным кодом.

Jenkins обладает меньшим функционалом по сравнению с TeamCity, но имеет много подключаемых плагинов, которые могут помочь заменить или даже улучшить те функции, которые требуется разработчикам в процессе тестирования, сборки и доставки приложений.

Актуальность исследования. На данный момент в Jenkins нет плагина, который полностью заменяет модуль визуализации статистики (Statistics) из TeamCity. Этот плагин/модуль требуется для того чтобы отслеживать состояние отдельных конфигураций сборки с течением времени, плагин собирает статистические данные по всей истории сборки и отображает их в виде наглядных диаграмм.

Степень разработанности проблемы. Среди имеющихся плагинов Jenkins есть те, которые реализует частичный функционал модуля из TeamCity, например, отображение графика продолжительности сборок за период. Подробнее о недостатках таких средств будет описано в сравнительном анализе и обзоре аналогов.

Объект исследования — инструменты для сборки приложений в инструментах совместного использования.

Предмет исследования — визуализация статистики работы сборок.

Цель - разработать плагин Jenkins для визуализации статистики работы сборок в инструментах совместного использования.

Задачи:

- A. Изучить инструменты сборки приложений.
- B. Изучить особенности CI/CD, Jenkins, работу и характеристики сборок Jenkins.
- C. Описать метрики и статистики, которые могут собираться по результатам работы сборок Jenkins.
- D. Изучить методы разработки плагинов Jenkins.
- E. Провести проектирование плагина и описать архитектуру.
- F. Реализовать плагин.
- G. Провести тестирование и апробацию плагина.

Основными **методами** проведения работы являются методы сравнительного анализа аналогичных решений и методы объектно-ориентированного программирования.

ГЛАВА 1. АНАЛИЗ СРЕДСТВ СБОРКИ И ВИЗУАЛИЗАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В первой главе рассмотрим:

- А. Процесс и инструменты сборки приложения.
- В. Анализ средств сборки программного обеспечения
- С. Понятие статистик и визуализации сборок в контексте инструментов CI/CD.
- Д. Обзор и сравнительный анализ плагинов Jenkins по визуализации статистики работы сборок.
- Е. Требования к разработке.

1.1. Анализ средств сборки программного обеспечения

Для осуществления сборки программного продукта существует множество инструментов, какое средство использовать определяют не только из преимуществ и недостатков этих средств, но и исходя из того, какой используется язык программирования, фреймворк и окружение. На данный момент существует большое количество инструментов сборки приложения.

Maven — инструмент для автоматизации сборки проектов, который используется с Java приложениями. Maven решает несколько проблем [33]:

- упрощение процесса сборки;
- обеспечение единой системы сборки;
- предоставление информации о проекте;
- упрощение работы с зависимостями, включая их автоматическое обновление.

Gradle — система автоматизации сборки, которая также часто используется для Java разработки. Gradle включает в себя следующие возможности [10]:

- декларативное описание сборки;
- управление зависимостями;
- создание многомодульных проектов;
- плагины.

Для проектов на JavaScript для управления зависимостями и сборками может использоваться npm в связке с Webpack. Webpack это инструмент для сборки и оптимизации приложений Node.js. Преимущества инструмента [1]:

- разбивки пакетов на мелкие фрагменты;
- поддержка плагинов;
- большое сообщество.

Также данный сборщик обладает такими недостатками, как высокий порог вхождения и низкая скорость сборки.

Также необходимо отметить, что в отличие от компилируемых языков, таких как Java, приложения на интерпретируемом языке Python могут запускаться без сборки прямо из командной строки, а для управления зависимостями в Python используется инструмент `pip`.

Существуют также и другие инструменты сборки для приложений написанных на разных языках программирования. Все их удобно использовать при локальной разработке над небольшими проектами, но когда рассматривается вопрос о разработке большого продукта, в работе над которым задействуется целая команда разработчиков и тестировщиков, для уменьшения затрат на разработку, в первую очередь временных, следует внедрять DevOps практики и CI/CD подходы.

Инструменты CI/CD позволят взаимодействовать с репозиторием гита, проводить сборку продукта автоматически по заданному времени или по наличию новых коммитов, прогонять тесты после каждого изменения разработчика, производить установку на различные стенды, а также выполнять сборку различных компонентов системы одновременно и доставлять продукт заказчику. Перейдем к рассмотрению особенностей CI/CD инструментов, а затем рассмотрим сравнение их между собой.

1.2. Анализ средств CI/CD

CI/CD — это технология автоматизации тестирования и доставки/развертывания готового приложения заказчику [18]. Данная технология стала неотъемлемой составляющей DevOps методологии и помогает сократить временные ресурсные затраты в процессе современного жизненного цикла приложения, когда до заказчика изначально доходит минимально жизнеспособный продукт (MVP), а затем дорабатывается с учетом новых требований заказчика, т.е. идет непрерывная разработка новых версий продукта.

Преимущества CI/CD подхода [7]:

- упрощение разработки - позволяет разработчикам распределять приоритеты и сконцентрироваться на самых важных аспектах;

- улучшение качества кода - качество кода проверяется до того, как он достигнет среды тестирования, проблемы в коде могут быть выявлены на ранних стадиях;
- более короткие циклы тестирования - меньший объем кода для проверки, становится проще определить проблемы в процессе развертывания;
- более простой мониторинг изменений - меньший объем кода для проверки;
- более легкий откат - меньшие усилия для отката приложения к предыдущей версии при возникновении проблем в новой версии.

Этапы разработки и принцип CI/CD подхода можно отразить с помощью рисунка 1.



Рис.1.1. Цикл CI/CD [3]

На данный момент существует множество инструментов CI/CD, которые обладают своими преимуществами и недостатками, были выделены самые распространенные системы:

- TeamCity;
- Jenkins;
- GitLab CI;
- CircleCI;
- Bamboo.

Jenkins — это автономный сервер автоматизации с открытым исходным кодом, который можно использовать для автоматизации всех видов задач, связанных со сборкой, тестированием, доставкой или развертыванием программного обеспечения [31].

TeamCity - это сервер CI от компании JetBrains [38], который позволяет запускать параллельные сборки одновременно на разных платформах и средах, а

также настраивать статистику по продолжительности сборки, уровню успешности, качеству кода и пользовательским метрикам.

GitLab CI - сервер CI от компании GitLab [23], которая также предоставляет одноименный репозиторий Git. GitLab CI/CD может обнаруживать ошибки на ранних этапах цикла разработки и гарантировать, что весь код, развернутый в рабочей среде, соответствует установленным стандартам кода.

CircleCI - сервер CI [20], который позволяет настроить для эффективной работы очень сложных конвейеров кэширование, кэширование уровня Docker и классы ресурсов для работы на более быстрых машинах.

Bamboo — это инструмент непрерывной интеграции и доставки [14], который связывает автоматизированные сборки, тесты и выпуски в единый рабочий процесс.

В таблице 1.1 указан, краткий сравнительный анализ плагинов. Стоит сразу отметить, что *сравнение платных и бесплатных решений не корректно*, поскольку организации, которые выпускают коммерческие продукты обладают куда большими возможностями в сравнении с компаниями, которые не берут плату за использование своего продукта. Что наглядно видно из сравнительного анализа Jenkins с остальными средствами CI.

Особое внимание следует уделить критерию OpenSource, этот критерий является достаточно важным с учетом, того, что многие компании после 2022 года ушли из РФ, тем самым стали либо недоступны, либо прекратили лицензирование и стали менее безопасными, т.к. новые версии продуктов больше недоступны и проблемы с безопасностью и другими дефектами не будут исправлены/доступны на территории РФ. Также критерий важен тем, что даже при наличии действия продуктов компаний, они обходилось крупным ИТ-компаниям достаточно дорого.

Также необходимо отметить еще 2 критерия для более объективной оценки: интеграции - количество интеграций инструмента со сторонними средствами и встроенная функциональность - количество встроенных функций. Критерий интеграция показывает сколько можно подключить к системе плагинов и интеграций со сторонними сервисами, а встроенный функционал сколько функций из коробки поддерживает, то или иное средство, наличие инструментов, которые позволят облегчить работу.

Если сравнивать описанные выше средства, то TeamCity обладает самой мощной встроенной функциональностью, а также достаточно большим количеством интеграций и плагинов [19], в сравнении со всеми остальными инструментами, за исключением Jenkins.

Таблица 1.1

Сравнительный анализ инструментов CI/CD

Критерий	Jenkins	TeamCity	GitLab CI	CircleCI	Bamboo
Открытый исходный код	+	-	+	-	-
Цена	Бесплатно	от 45\$ в месяц [19]	от 21\$ в месяц [2]	от 15\$ в месяц [19]	от 1200\$ в год [19]
Поддержка вендора	-	+	+	+	+
Поддержка репозитория Git	Любой репозиторий	GitHub, GitLab, Bitbucket	GitHub, GitLab, Bitbucket	GitHub, GitLab, Bitbucket	Любой репозиторий

Jenkins в отличие от перечисленных коммерческих средств обладает самой низкой встроенной функциональностью, также отсутствует возможность построения конвейеров, но при этом все эти недостатки закрываются большим количеством плагинов, которые постоянно пишутся разработчиками, среди плагинов имеется и pipeline, который и нужен для построения конвейеров, количество плагинов около 2 тысяч, что в несколько раз больше, чем у TeamCity, в котором около 500 плагинов и интеграций.

Среди всех средств особо ярко выделяется Jenkins, поскольку он является бесплатным и с открытым исходным кодом, а также обладает большим количеством интеграций и плагинов, которые постоянно пишутся, что позволяет устранить основной его недостаток по наличию встроенных функций. После 2022 года в РФ это стал самый востребованный инструмент для настройки CI конвейеров, и обосновывает важность разработки плагина для устранения недостатков функциональности, которые есть в других средствах.

1.3. Анализ существующих плагинов Jenkins по визуализации статистики сборок

Поскольку для работы со сборками приложений был выбран Jenkins, то разработка плагина будет производиться в этой системе. В любой системе с

большим количеством приложений, сборок и тестов будет удобно производить мониторинг и визуализацию информации по статистике работы сборок во времени.

Под статистикой работы сборок будем понимать следующие статистические характеристики собираемых метрик (продолжительность исполнения сборки, время проведенное в очереди сборкой, размеры полученных по итогам сборки артефактов):

- среднее арифметическое;
- мода;
- медиана;
- размах;
- среднеквадратическое отклонение;
- среднеквадратическое отклонение несмещенное;
- дисперсия.

В данной работе будет разработан плагин для воссоздания модуля Statistics в Jenkins, с дополнительным функционалом, которого не хватало в TeamCity, который позволял отслеживать состояние отдельных конфигураций сборки с течением времени, собирать статистические данные по всей истории отдельного задания и отображать их в виде наглядных диаграмм.

Для оценки плагинов, необходимо понять какие метрики требуется для сбора статистики работы сборок. Требуется реализовать следующие метрики:

- визуализация метрики success rate (SR) - процент успешности сборок, который будет показывать сколько сборок завершилось успешно;
- визуализация метрики Build Duration (BD) - время выполнения сборок, в том числе должен быть доступен фильтр на добавления в график упавших сборок, а также возможность вычислять не только суммарно время сборок, а также среднее время всех сборок за определенный интервал времени;
- визуализация метрики Time Spent in queue (TQ) - время проведенное в очереди сборок, в том числе среднее время, вычисляемое аналогично Build Duration;
- визуализация метрики Test Count (TC) - количество выполненных тестов в сборке, в том числе количество выполненных тестов в упавших сборках, если таковые успели выполниться;
- визуализация метрики Artifacts Size (AS) - размер созданных во время сборки артефактов, в том числе средний размер за определенный интервал

времени, а также учет артефактов, которые успели создаться в сборках до падения.

Сначала рассмотрим уже разработанные плагины визуализации и их недостатки и преимущества в сравнении с разрабатываемым решением. Результаты сравнения приведены в таблице 1.2.

Build Monitor Plugin - плагин, который обеспечивает наглядное представление статуса выбранных заданий Jenkins. Отображает состояние и ход выполнения выбранных заданий [15].

Global Build Stats Plugin - плагин, который позволит собирать и отображать глобальную статистику результатов сборки, а также позволяющий отображать глобальную тенденцию сборки Jenkins/Hudson с течением времени [24].

Build Time Blame - плагин, который сканирует вывод консоли на наличие успешных сборок и генерирует отчет, показывающий, как эти шаги повлияли на общее время сборки. Это предназначено для того, чтобы помочь проанализировать, какие этапы процесса сборки являются подходящими кандидатами на оптимизацию [16].

После проведения сравнения аналогичных решений, были выявлены преимущества разрабатываемого плагина, которые обосновывают его разработку, это отсутствие у данных плагинов функционала по визуализации Artifacts Size, Time Spent in queue, Success Rate истории сборок, а также наличие прогнозирования метрик следующей сборки. Также данные плагины не предлагают динамическое изменение графиков по мере изменения временного интервала или установления фильтров.

1.4. Требования к разработке

Поскольку разрабатываемый плагин является аналогом модуля статистики сборок в TeamCity (поскольку TeamCity является лучшим из коммерческих инструментов и имеет удобный модуль визуализации статистики), то функционал должен как минимум реализовывать функции модуля Statistics в TeamCity. В первую очередь должна производиться визуализация метрик сборок с помощью графиков и диаграмм.

На всех графиках и диаграммах должна быть возможность выбора значения из выпадающего списка интервала времени, за который будет производиться сбор статистики за день, месяц, квартал, неделю, год.

Таблица 1.2

Сравнительный анализ плагинов Jenkins

Критерий	Build Monitor Plugin [15]	Global Build Stats Plugin [24]	Build Time Blame [16]	Плагин разрабатываемый
Прогнозирование метрик следующей сборки	-	-	-	+
Открытый исходный код	+	+	+	+
Визуализация времени выполнения и статуса последней сборки	+	+	- (только время)	+
Визуализация SR истории сборок	-	+/- (в TeamCity гистограммы, которые показывают процентное соотношение нагляднее)	-	+
Визуализация BD истории сборок (в числе average)	-	+	+	+
Визуализация TQ	-	-	-	+
Визуализация TC	-	-	+	+
Визуализация AS	-	-	-	+
Отображение всех графиков на одной странице по одному диапазону времени для наглядного отображения всех метрик в один момент и во времени	-	+	-	+

Например, был выбран промежуток времени месяц, то должен выполняться следующий набор действий:

- А. Должна собираться информация о требуемой метрики у всех сборок.
- В. Производиться фильтрация сборок т.е. должны отбираться только сборки за последний месяц (в том числе упавшие, если был выбран данный чекбокс).
- С. Полученные сборки должны группироваться по дням т.е. на итоговом графике должно быть 30/31 точка или столбца.
- Д. Если необходимо производиться вычисление статистической обработки среди всех сгруппированных за день метрик сборок.
- Е. Отображение всей информации о метриках сборки на одном графике или диаграмме.

Также все графики должны располагаться друг под другом на одной странице, что может наглядно показать (если на каждом графике был выбран один период), все вычисленные метрики за один период, например при выборе месяца все перечисленные метрики будут отображены на странице и можно будет увидеть, что происходило, например вчера по результатам запуска всех сборок.

На метрикам BD, AS, TQ должна быть возможность выбрать статистический показатель, в соответствии с которым должна производиться обработка итоговых значений. Возможные показатели перечислены в разделе 1.4.

Помимо прочего требуется, чтобы при визуализации можно было выбрать различные типы диаграмм: столбчатые, линейные тренды, круговые.

Также было принято решения добавить анализ данных, чтобы делать предположение, о том какими метриками будет обладать следующая запущенная сборка, при вычислении данного значения должно быть рассчитаны веса каждой сборки/сборок по графику за определенный период, и если сборка была собрана, например, месяц назад - она должна иметь меньший вес, чем сборка, собранная вчера.

Также к разработке будет предъявлено требование об удобстве интерфейса: все графики должны быть удобными, не перегруженными информацией, а также интерфейс должен быть интуитивно понятен, чтобы данный плагин не усложнял восприятие собранной статистики сборок и не вызывал желание воспользоваться другим плагином или разработать другой более удобной, или отказаться от идеи смотреть статистику по сборкам.

1.5. Выводы

По всем описанным выше разделам можно прийти к выводу, что данный плагин актуален для ИТ-компаний, которые ранее отдавали предпочтение многофункциональному инструменту TeamCity, в котором уже были все необходимые для работы функции, особенно это актуально для компаний в РФ, но также может понадобиться и другим компаниям, которые приняли решение отказаться от TeamCity в пользу Jenkins из-за больших денежных затрат на лицензию. Также будут реализованы дополнительный функционал по сравнению с модулем TeamCity, что даст преимущества не только в цене. После проведенного обзора аналогичных решений становится понятно, что сейчас в Jenkins нет полнофункциональной замены модуля статистики TeamCity, также необходимо учесть и визуальную составляющую, чтобы при установке данного плагина разработчики выбирали его не только из-за отсутствия другого решения.

ГЛАВА 2. ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПЛАГИНА

В данной главе будет проведено проектирование разрабатываемого плагина: будет описана архитектура построения плагинов в Jenkins, а также архитектура разработки, будут выбраны инструменты разработки, а также рассмотрена функциональная модель системы. Поскольку плагин разрабатывается для системы Jenkins, то отладку и тестирование будем проводить в этой системе.

2.1. Модель системы

Диаграмма вариантов использования, показывающая функционал плагина отображена на рисунке 2.1. На данной диаграмме основное внимание также уделяется процессу визуализации статистики метрик сборок. Основное действующее лицо одно - это пользователь системы, который запускает сборки и работает в CI системе, это может быть любой участник команды, который задействован в разработке, тестировании, доставке и внедрению приложения. В данном случае все эти роли представлены на диаграмме как разработчик.

Функциональная модель в нотации IDEF0 отображена на рисунках 2.2.-3. Основное внимание на диаграмме уделяется визуализации статистики сборок,

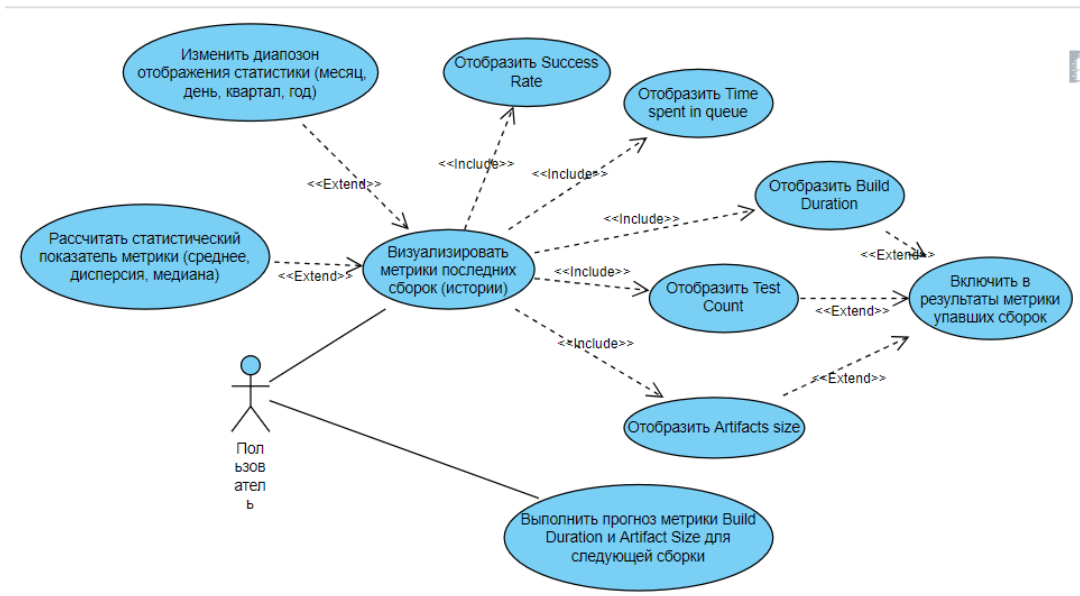


Рис.2.1. Use case

поскольку это изначально является целью разработки. Также там будут отражены дополнительные функции такие как фильтрация, и высчитывание статистик метрик.

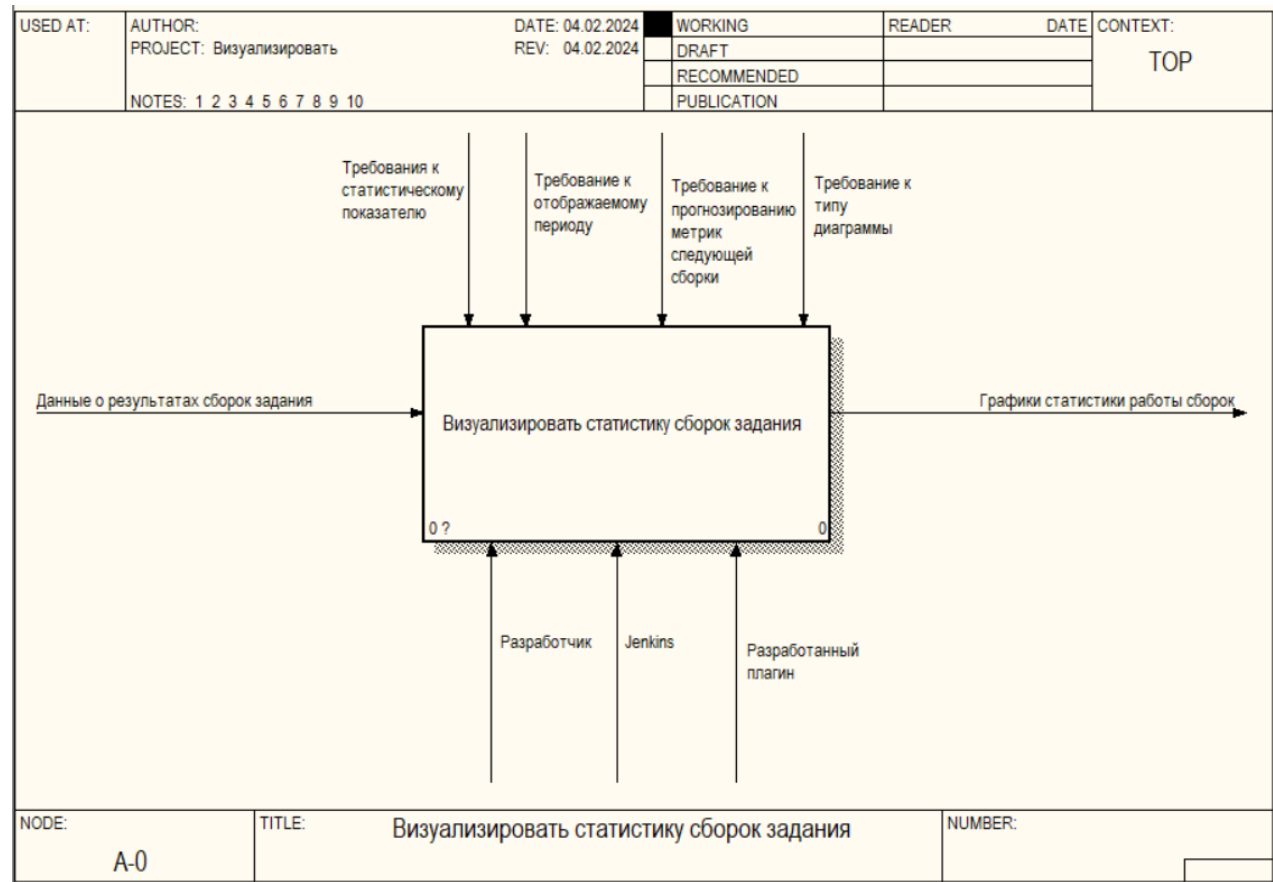


Рис.2.2. Процесс визуализации сборок

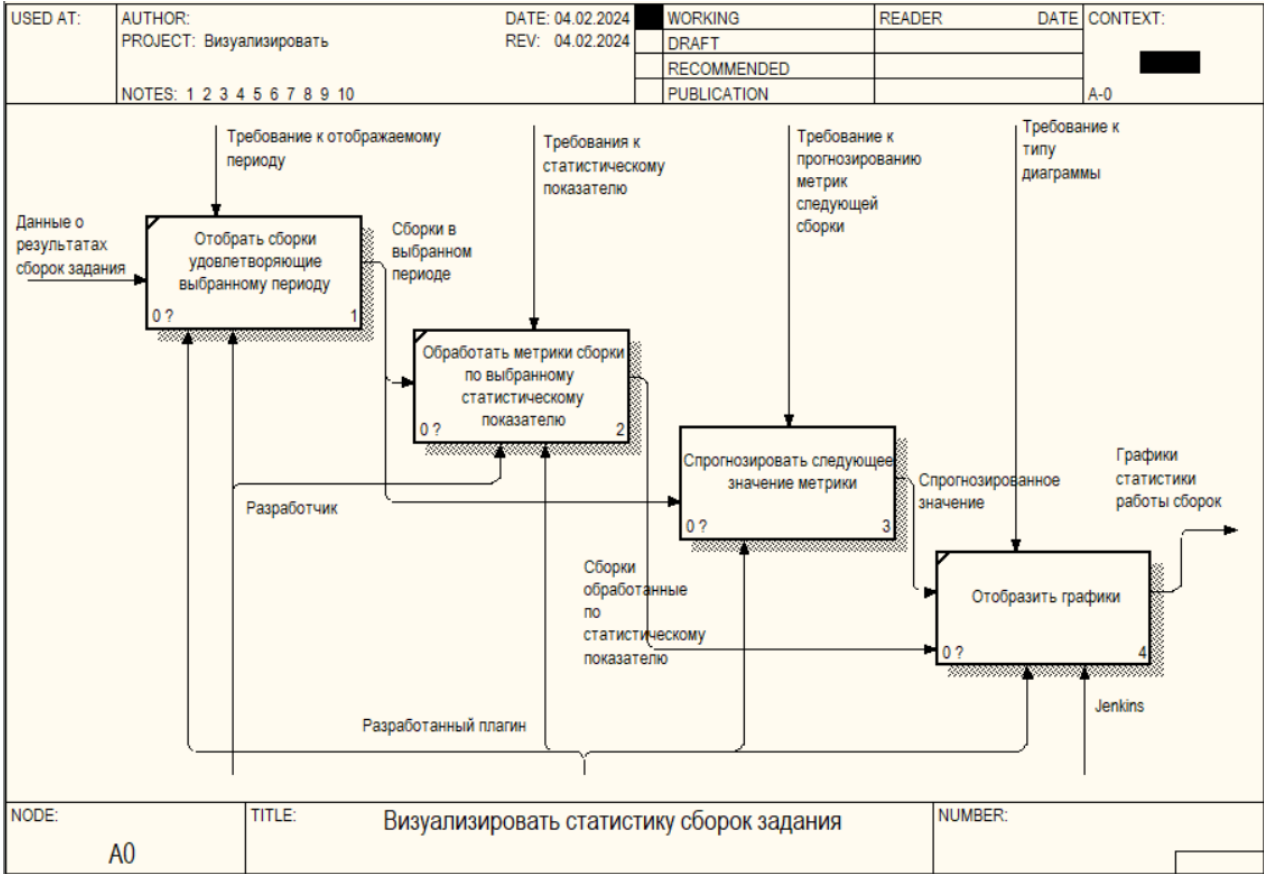


Рис.2.3. Детализация процесса визуализации

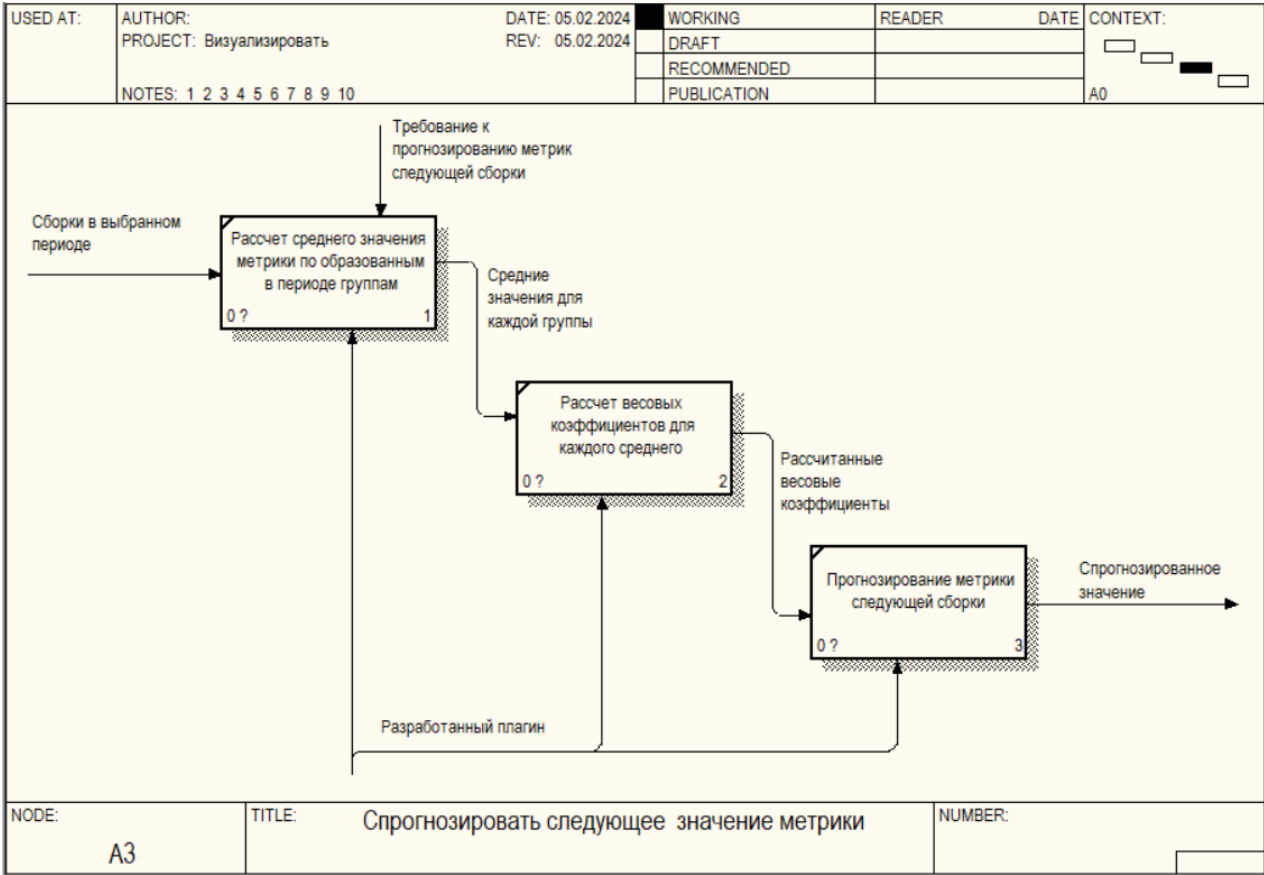


Рис.2.4. Процесс прогнозирования метрики

2.2. Архитектура Jenkins

Перед объяснением построения архитектуры плагинов Jenkins, необходимо привести схему архитектуры Jenkins, где будет отображено место разрабатываемых плагинов в CI системе. Архитектура Jenkins представлена на рисунке 2.5. Установленные плагины Jenkins-CI, а также локальные сценарии и приложения выполняются на сервере Jenkins-CI и предоставляют расширяемый набор функций управления и обработки данных [12].

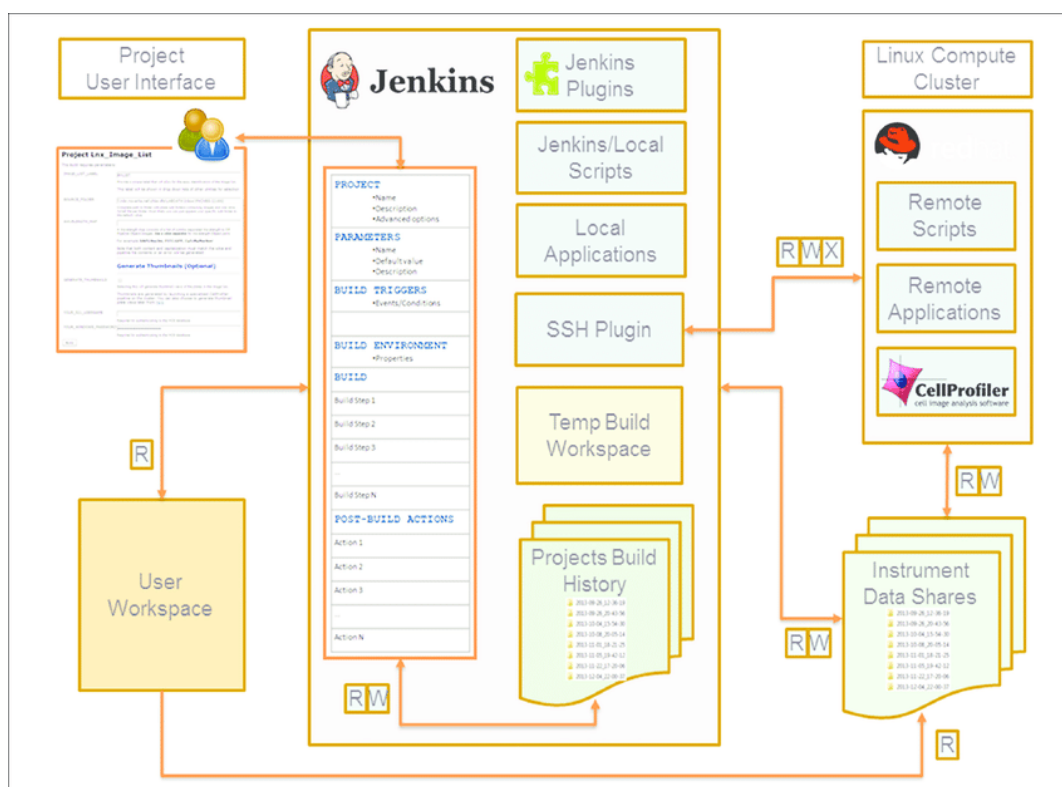


Рис.2.5. Архитектура Jenkins [12]

Архитектура плагинов использует точки расширения, которые, предоставляют разработчикам плагинов возможности реализации для расширения функциональности системы Jenkins [39]. Точки расширения автоматически обнаруживаются Jenkins во время загрузки системы.

В разрабатываемом плагине реализация будет происходить через класс Action. Actions являются основным строительным блоком расширяемости в Jenkins: их можно прикреплять ко многим объектам модели, хранить вместе с ними и при необходимости добавлять в их пользовательский интерфейс.

Помимо класса Action для того чтобы создать временные действия, которые будут прикреплены к заданию Jenkins будет использован класс TransientActionFactory, который позволяет создавать действия, которые будут

отображаться на страницах Jenkins только при наличии соответствующего объекта - задания.

Разработка будет выполняться в объектно-ориентированной парадигме, т.е. приложение будет разбито на классы, будет применяться наследование, полиморфизм и инкапсуляция. Все классы, которые будут разработаны для плагина отображены на рисунке 2.6.

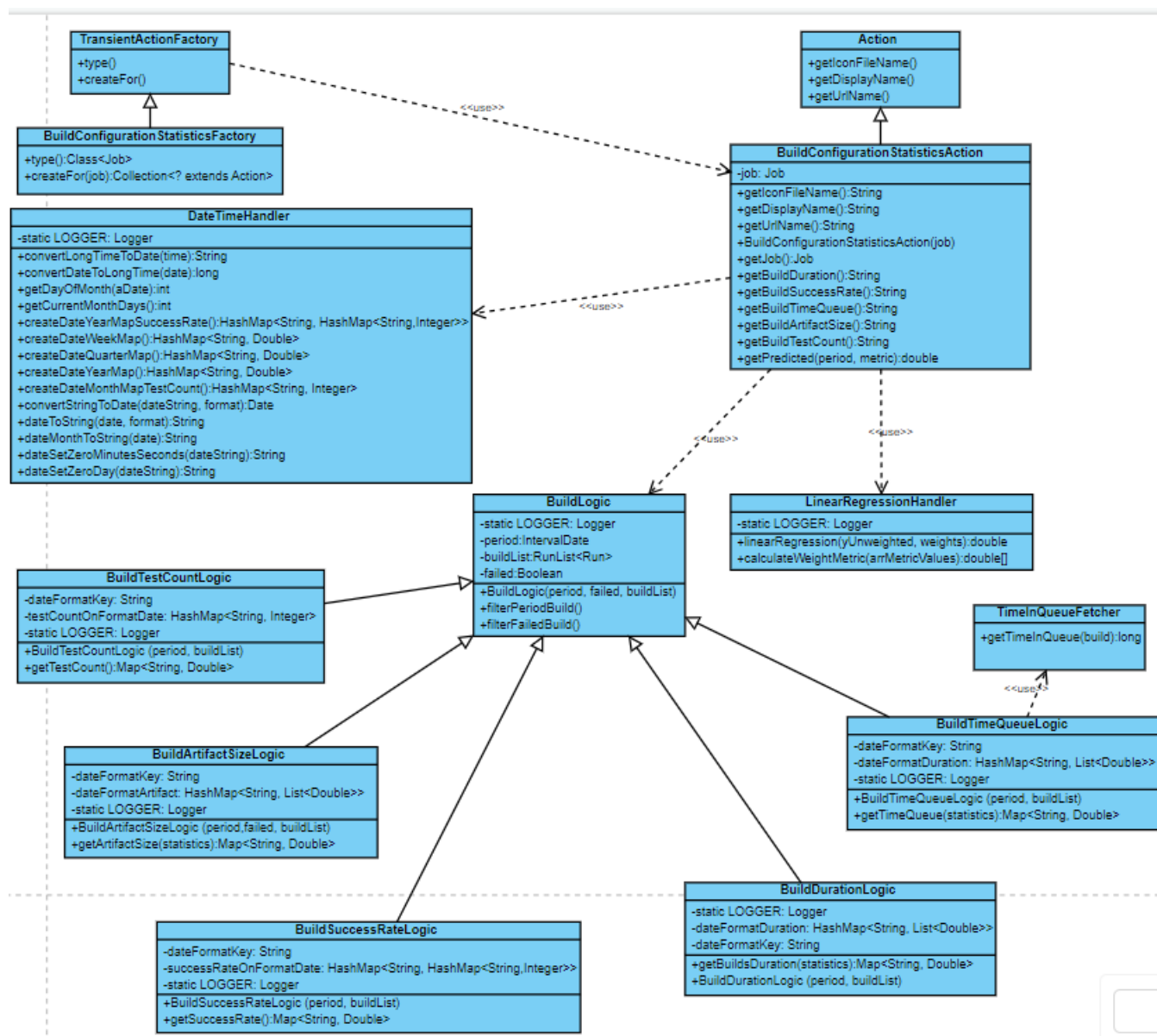


Рис.2.6. Диаграмма классов плагина

При рассмотрении диаграммы необходимо отметить, что два класса являются встроенными в Jenkins, это `TransientActionFactory`, который позволяет добавлять действия к любому типу объекта, а также интерфейс `Action` - добавленный к объекту модели, создает дополнительное подпространство URL-адресов под родительским объектом модели, через которое он может взаимодействовать с пользователями.

Actions также способны открывать доступ к левому меню в интерфейсы Jenkins, по которому обычно производится навигация при конфигурировании сборки.

Для удобства использования плагина, предполагается добавить дополнительную ссылку в меню слева, для перехода на страницу визуализации метрик, а также динамически обновлять страницу при изменении параметров и фильтров, что и обосновывает использование данных встроенных классов.

Основная часть остальных классов требуется для работы с определенной метрикой статистики выполнения сборок Jenkins, что следует из их названия. Также будет разработан дополнительный класс `DateTimeHandler`, который позволит создать методы для удобной работы с датой и временем, что необходимо поскольку будет производиться преобразования одних типов дат к другим, сравнение дат между собой, а также получение определенных частей дат.

2.3. Архитектура плагина

Для того чтобы визуализировать и обработать данные о сборках, необходимо получить эти данные. Для этого необходимо использовать различные методы и классы Jenkins, такие как `Job` - для работы с проектом (статическая сущность), а также `Run` для работы со сборкой (конкретные запуски `Job`, со временем выполнения и результатом). Внутри методов этих сущностей при их вызове будет отправляться API запрос на сервер Jenkins, который будет возвращать данные из хранилища xml файлов для каждой конкретной сборки.

После получения данных в плагине, идет их обработка и подготовка структур данных для визуализации. Вызов методов обработки данных о сборках будут происходить из Jelly файлов, в которых с помощью специальных тегов будет производиться связывание между объектами бизнес-логики Java и JS файлами, где будут создаваться графики визуализации.

Jelly для получения данных из Java использует AJAX запросы, а затем полученные данные сохраняет в DOM структуре страницы плагина. Затем с помощью JS происходит получение данных о сборках из DOM структуры и отправка в методы построения графиков.

Все взаимодействие между Java, Jelly и JS происходит с помощью JSON структур, такое решение было принято ввиду удобства работы со структурой с помощью этих инструментов. На рисунке 2.7 представлено изображение архитектуры плагина.

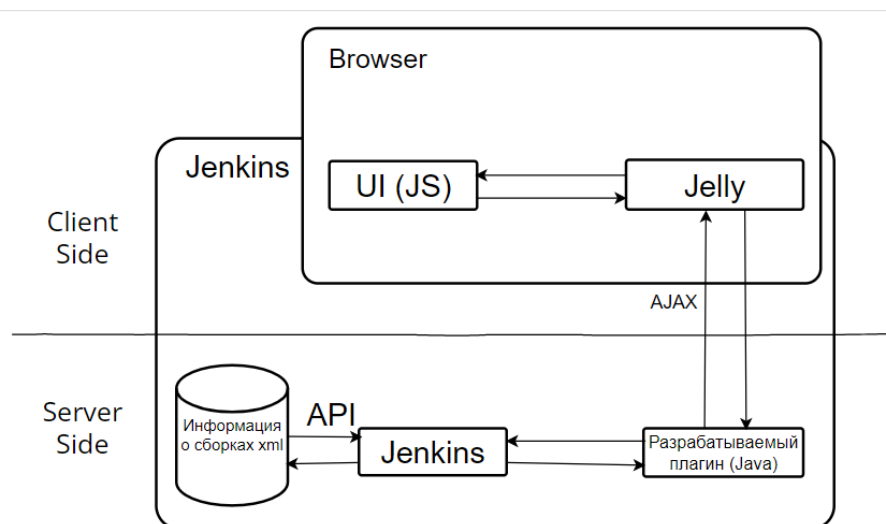


Рис.2.7. Архитектура плагина

2.4. Языки программирования

Для программирования плагина будет использоваться язык Java. Поскольку Jenkins написан на Java, то все плагины необходимо писать на том же языке. Это является главным минусом, а возможно и сложностью при разработке плагинов на Jenkins, поскольку ограничивает свободу разработчика.

Есть возможность разработки плагина с использованием языка программирования Groovy. Groovy это динамический язык с возможностями статической типизации и статической компиляции для платформы Java[25], нацеленный на повышение производительности разработчиков, который плавно интегрируется с любой программой Java.

Недостатком такого выбора является то, что абсолютное большинство плагинов написано на чисто Java, а значит сообщества и поддержка при разработке на Java будет значительно большей. Также в сравнении с Groovy, Java обладает большей производительностью[36], статической типизацией и подходит для разработки приложений в парадигме ООП.

Java — это язык высокого уровня, который можно охарактеризовать следующими словами: объектно-ориентированный, многопоточный, динамический, высокопроизводительный и безопасный [28]. Java используется для разработки высоконагруженных информационных систем, мобильных приложений, плагинов, десктопного ПО и др. К преимуществам Java также можно отнести компилируемость, что обеспечивает высокое быстродействие.

Java будет использоваться для программирования ядра плагина и бизнес-логики. Также для программирования графических компонентов, графиков и диаграмм будет использоваться язык программирования JavaScript. JS - это легкий, интерпретируемый или JIT-компилируемый, объектно-ориентированный язык [29], основное предназначение которого выполнять сценарии на веб-страницах, что необходимо при разработке плагина, результаты которого отображаются на веб-страницах.

Помимо прочего, для стилизации компонентов веб-интерфейса будет использоваться язык каскадных таблиц стилей CSS [21], который позволит настроить удобное отображение и позиционирование элементов на странице плагина Jenkins.

Верстка страниц будет осуществляться с помощью инструмента Jelly - все разрабатываемые плагины используют данный инструмент в Jenkins, поскольку с ним можно легко интегрировать Java, XML и JS. Jelly — это средство для преобразования XML в исполняемый код, это механизм сценариев и обработки на основе Java и XML [30]. В Jelly можно вызывать функции Java, использовать такие синтаксические конструкции, как циклы, условия и переменные, также он позволяет легко обратиться к объектам в Java.

2.5. Инструменты сборки

В качестве инструмента сборки проекта был выбран Maven, который можно использовать для создания и управления любым проектом на основе Java. Преимущества Maven были описаны в первой главе при рассмотрении инструментов сборок приложения.

Абсолютное большинство разработанных плагинов для Jenkins использует Maven, поскольку Maven предоставляет удобные архетипы для начала разработки плагинов, что делает использование того же Gradle не рациональным.

2.6. Библиотеки

2.6.1. Chart.js

Поскольку проект предполагает использование графиков и диаграмм, то необходимо было выбрать инструмент для работы с графиками в Jenkins и Java, который позволит отображать графики прямо на странице задания Jenkins. В

качестве этого инструмента была выбрана библиотека Chart.js, которая на данный момент является самой популярной JavaScript библиотекой по оценкам GitHub и загрузок npm [17]. К преимуществам данной библиотеки можно отнести:

- у Chart.js очень подробная документация;
- отрисовка canvas делает Chart.js очень производительным, особенно для больших наборов данных и сложных визуализаций;
- строит отзывчивый интерфейс - перерисовывает диаграммы при изменении размера окна для идеальной детализации масштаба.

2.6.2. *Gson*

Для удобства разработки и взаимодействия клиентской части с серверной необходимо, чтобы обработанные данные из Jenkins в браузер передавались в формате JSON. Для обеспечения преобразования объектов и структур Java в JSON была использована библиотека Gson от компании google. К важным особенностям и преимуществам данной библиотеки можно отнести:

- легкость в использовании;
- отсутствие требования размещения аннотаций Java в пользовательских классах [26];
- поддержка Java Generics.

2.6.3. *Apache Commons Math*

Поскольку одним из требований к плагину является работа с математической статистикой для обработки метрик сборок, а также прогнозирование значений метрик, то необходима библиотека, которая реализует данный функционал. В качестве такой библиотеки была выбрана Apache Commons Math. Commons Math это библиотека легких, автономных математических и статистических компонентов, решающая наиболее распространенные проблемы [13], связанные со статистикой. Преимущества использования библиотеки:

- скорость математических вычислений выше, чем у стандартных средств Java;
- открытый исходный код;
- функции для статистической обработки и анализа данных (линейная регрессия).

2.6.4. Junit

Для того чтобы, протестировать разработанный плагин, будут написаны unit тесты. Поскольку разработка плагина будет вестись на языке Java, то для написания тестов будет использована Java инструмент. В качестве такого инструмента был выбран Junit, который состоит из следующих компонентов: JUnit Platform, JUnit Jupiter, JUnit Vintage [32]. Основными преимуществами инструмента являются:

- популярность и большое сообщество;
- использование для написания тестов Java (без дополнительных фреймворков и языков);
- совместимость с Java инструментами (Maven, Gradle).

2.6.5. Mockito

Поскольку функционал плагина использует обращение к структурам Jenkins (Job, Run), для тестирования необходимо использовать заглушки этих структур. Для работы с заглушками была выбрана библиотека mockito, которая была выбрана из-за следующих особенностей, причин и преимуществ:

- сообщество StackOverflow назвало Mockito лучшим фреймворком для заглушек на Java [34];
- входит в число 10 лучших библиотек Java среди всех библиотек;
- позволяет писать читабельные тесты, использовать BDD подход.

2.7. Выводы

В данной главе было проведено проектирование плагина, составлена use-case диаграмма и диаграмма классов, построена функциональная модель системы, описана архитектура Jenkins, а также описана архитектура, разрабатываемого плагина. Затем были выбраны инструменты разработки плагина.

ГЛАВА 3. РЕАЛИЗАЦИЯ ПРОТОТИПА ПЛАГИНА

В 3 главе будут рассмотрены и описаны основные классы, которые были разработаны в соответствии с диаграммой классов из главы 2, а также полученные результаты.

3.1. Описание разработанных классов

В процессе написания кода плагина были запрограммированы классы в соответствии с диаграммой классов из главы 2.

3.1.1. *BuildConfigurationStatisticsAction*

Основной класс приложения, который реализует интерфейс действия, через этот класс происходит взаимодействие с Jelly, а также вызов всех остальных методов бизнес-логики плагина, и определены поля для работы со сборками, все методы для получения информации о конкретной метрике сборки помечены аннотацией @JavaScript для того, чтобы можно было их вызывать через JS в Jelly, также во всех этих методах тип возвращаемого объекта приведен к JSON, который и передается в DOM страницы плагина при взаимодействии с элементами пользовательского интерфейса.

В классе реализованы только одно закрытое поле job, с помощью которого происходит взаимодействие со сборкой в проекте, а также обработка выполненных запусков. А также следующие методы:

- BuildConfigurationStatisticsAction(Job job) - конструктор класса;
- String getIconFileName() - метод, определяющий иконку приложения в боковом меню;
- String getDisplayName() - метод, определяющий отображаемое имя плагина в боковом меню и других частях страницы;
- String getUrlName() - метод, определяющий url, по которому доступна страница плагина;
- Job getJob() - метод-геттер для получения текущей сборки;
- String getBuildDuration(String period, String fail, String statistics) - метод для получения информации о времени продолжительности запусков сборки, за определенный период, с заданными настройками;
- String getBuildSuccessRate(String period) - метод для получения информации о проценте успешности выполнения запусков сборки, за определенный период;
- String getBuildArtifactSize(String period, String fail, String statistics) - метод для получения информации о созданных артефактах запусков сборки, за определенный период, с заданными настройками;

- `String getBuildTestCount(String period, String fail)` - метод для получения информации о количестве выполненных тестов при запуске сборки, за определенный период, с заданными настройками;
- `String getBuildTimeQueue(String period, String statistics)` - метод для получения информации о времени нахождения в очереди запусков сборки, за определенный период, с заданными настройками.
- `double getPredicted(String period, String metric)` - метод для прогнозирования метрик (BD, AS) следующей сборки.

3.1.2. DateTimeHandler

Статический класс, созданный для взаимодействия с датами и их обработки при создании структур данных, которые также создаются в рамках этого класса, формирования структуры данных зависит от метрики и от периода за который нужно получить информацию.

В классе определено поле `Logger LOGGER`, с помощью которого записываются логи плагина. Также в классе определены методы:

- `Date convertLongTimeToDate(long time)` - метод, преобразующей время в миллисекундах, прошедших с 1970 года, в дату типа `Date`;
- `long convertDateToLongTime(Date date)` - метод, преобразующей дату в миллисекунды, прошедшие с 1970 года;
- `int getDayOfMonth(Date aDate)` - метод, получающий номер дня из даты в месяце;
- `int getLastMonthDays()` - метод для получения дней в прошлом месяце;
- `String dateToString(Date date, String format)` - метод для преобразования типа `Date`, в строку в соответствии с заданным форматом даты;
- `String dateMonthToString(Date date)` - метод для преобразования типа `Date`, в строку в соответствии с форматом "yyyy-MM";
- `String dateSetZeroMinutesSeconds(String dateString)` - метод для обнуления минут и секунд в строке с датой;
- `HashMap<String, List<Double>> createDateMonthMap()` - метод, создающий начальную структуру данных за прошедший месяц по дням, где в качестве значений словаря используются пустые списки;
- `HashMap<String, List<Double>> createDateAllMap(RunList<Run> runs)` - метод для создания начальной структуры данных, за весь прошедший

период, который вычисляет на какие равные интервалы следует разбить общий промежуток времени, от создания первой сборки, до создания последней сборки;

- `HashMap<String, List<Double> createDateDayMap()` - метод, создающий начальную структуру данных за прошедший день по часам, где в качестве значений словаря используются пустые списки;
- `HashMap<String, HashMap<String,Integer> createDateDayMapSuccess()` - метод, создающий начальную структуру данных за прошедший день по часам, для вычисления процента успешности выполнения сборок, где в качестве значений словаря используется словарь, в котором записано сколько запусков сборки выполнено успешно, а сколько с ошибками;
- `HashMap<String, HashMap<String,Integer> createDateWeekMapSuccessRate()` - метод, создающий начальную структуру данных за прошедшую неделю по дням, для вычисления процента успешности выполнения сборок, где в качестве значений словаря используется словарь, в котором записано сколько запусков сборки выполнено успешно, а сколько с ошибками;
- `HashMap<String, HashMap<String,Integer> createDateMonthMapSuccessRate()` - метод, создающий начальную структуру данных за прошедший месяц по дням, для вычисления процента успешности выполнения сборок, где в качестве значений словаря используется словарь, в котором записано сколько запусков сборки выполнено успешно, а сколько с ошибками;
- `HashMap<String, HashMap<String,Integer> createDateQuarterMapSuccessRate()` - метод, создающий начальную структуру данных за прошедший квартал по месяцам, для вычисления процента успешности выполнения сборок, где в качестве значений словаря используется словарь, в котором записано сколько запусков сборки выполнено успешно, а сколько с ошибками;
- `HashMap<String, Integer> createDateMonthMapTestCount()` - метод, создающий начальную структуру данных за прошедший месяц по дням, для вычисления количества выполненных тестов во время запуска сборки, где в качестве значений словаря используется целочисленные нули;
- `HashMap<String, Integer> createDateDayMapTestCount()` - метод, создающий начальную структуру данных за прошедший день по часам, для вычисления количества выполненных тестов во время запуска сборки, где в качестве значений словаря используется целочисленные нули;

- `HashMap<String, Integer> createDateYearMapTestCount()` - метод, создающий начальную структуру данных за прошедший год по месяцам, для вычисления количества выполненных тестов во время запуска сборки, где в качестве значений словаря используется целочисленные нули;
- `HashMap<String, Integer> createDateQuarterMapTestCount()` - метод, создающий начальную структуру данных за прошедший квартал по месяцам, для вычисления количества выполненных тестов во время запуска сборки, где в качестве значений словаря используется целочисленные нули;
- `HashMap<String, Integer> createDateWeekMapTestCount()` - метод, создающий начальную структуру данных за прошедшую неделю по дням, для вычисления количества выполненных тестов во время запуска сборки, где в качестве значений словаря используется целочисленные нули;
- `HashMap<String, List<Double>> createDateYearMap()` - метод, создающий начальную структуру данных за прошедший год по месяцам, где в качестве значений словаря используются пустые списки;
- `HashMap<String, List<Double>> createDateQuarterMap()` - метод, создающий начальную структуру данных за прошедший квартал по месяцам, где в качестве значений словаря используются пустые списки;
- `HashMap<String, List<Double>> createDateWeekMap()` - метод, создающий начальную структуру данных за прошедшую неделю по дням, где в качестве значений словаря используются пустые списки;
- `HashMap<String, HashMap<String,Integer>> createDateYearMapSuccessRate()` - метод, создающий начальную структуру данных за прошедший год по месяцам, для вычисления процента успешности выполнения сборок, где в качестве значений словаря используется словарь, в котором записано сколько запусков сборки выполнено успешно, а сколько с ошибками.

3.1.3. IntervalDate

Перечисляемый тип для удобства работы с датами-периодами. Содержит следующие predefined константы:

- `DAY` - день;
- `WEEK` - неделя;
- `MONTH` - месяц;
- `YEAR` - год;

- QUARTER - квартал.

3.1.4. Statistics

Перечисляемый тип для удобства работы с показателями статистики. Содержит следующие предопределенные константы:

- SUM - сумма;
- AVG - среднее;
- MEDIAN - медиана;
- RANGE - размах;
- DISPERSION - дисперсия;
- SDUNBIASED - среднеквадратичное отклонение несмещенное;
- MODE - мода;
- SD - среднеквадратичное отклонение.

3.1.5. TimeInQueueFetcher

Класс отвечающий за расчет времени, которая сборка провела в очереди перед тем как отправилась на выполнение. В классе определен один метод `long getTimeInQueue(Run build)` с помощью которого вычисляется нахождение времени в очереди в миллисекундах для конкретного запуска сборки.

3.1.6. BuildLogic

Базовый класс бизнес-логики, от которого наследуются все остальные более специфичные классы по каждой метрике, в классе определяются методы фильтрации по периоду и наличию упавших сборок в итоговых результатах. В классе определены следующие поля:

- `IntervalDate period` - период за который производится отбор запусков сборок для дальнейшей обработки и визуализации;
- `RunList<Run> buildList` - список запусков у конкретной сборки;
- `Boolean failed` - поле, которое определяет нужно ли учитывать при обработке и визуализации упавшие сборки (`true` - надо учитывать);
- `Logger LOGGER` - поле, с помощью которого записываются логи плагина при выполнении методов класса.

Также в классе определены методы, которые наследуются всеми остальными классами бизнес-логики, которые отвечают за работу с определенной метрикой:

- BuildLogic(IntervalDate period, Boolean failed, RunList<Run> buildList) - конструктор класса;
- void filterPeriodBuild() - метод, который производит отбор только тех запусков, которые удовлетворяют заданному периоду;
- void filterFailedBuild() - метод, который производит отбор только тех запусков, которые удовлетворяют полю failed, т.е. в зависимости от значения флага, либо включает в выборку упавшие сборки, либо нет.

3.1.7. BuildArtifactSizeLogic

Класс для работы с метрикой AS, в нем происходит пересчет параметров в зависимости от периода и настроек подданных на вход, а также высчитывается размер артефакта в Кб. В классе определены следующие поля:

- HashMap<String, Double> dateFormatArtifact - структура данных для работы с запусками сборки относительно метрики AS, ключи даты за выбранный период, значения размер артефактов, созданных во время запуска за выбранный период;
- String dateFormatKey - поле, которое определяет формат даты за выбранный период, по которому будет происходить обработка и визуализация;
- Logger LOGGER - поле, с помощью которого записываются логи плагина при выполнении методов класса.

Также в классе определены методы:

- BuildArtifactSizeLogic(IntervalDate period, Boolean failed, RunList<Run> buildList) - конструктор класса;
- Map<String, Double> getArtifactSize(Statistics statistics) - метод, в котором происходит фильтрация данных запусков сборок по периоду и флагу failed, определение формата дат и вычисление размера артефактов в Кб, а также расчет по статистической величине (например, среднее арифметическое), в зависимости от заданных настроек.

3.1.8. BuildDurationLogic

Класс для работы с метрикой BD, в нем происходит пересчет параметров в зависимости от периода и настроек подданных на вход, а также высчитывается продолжительность сборки в секундах. В классе определены следующие поля:

- `HashMap<String, Double> dateFormatDuration` - структура данных для работы с запусками сборки относительно метрики BD, ключи даты за выбранный период, значения время выполнения запусков сборки за выбранный период;
- `String dateFormatKey` - поле, которое определяет формат даты за выбранный период, по которому будет происходить обработка и визуализация;
- `Logger LOGGER` - поле, с помощью которого записываются логи плагина при выполнении методов класса.

Также в классе определены методы:

- `BuildDurationLogic(IntervalDate period, Boolean failed, RunList<Run> buildList)` - конструктор класса;
- `Map<String, Double> getBuildsDuration(Statistics statistics)` - метод, в котором происходит фильтрация данных запусков сборок по периоду и флагу failed, определение формата дат и вычисление времени выполнения запусков сборок, а также расчет по статистической величине (например, среднее арифметическое), в зависимости от заданных настроек.

3.1.9. BuildSuccessRateLogic

Класс для работы с метрикой SR, в нем происходит пересчет параметров в зависимости от периода, а также высчитывается процент успешности выполненных сборок за заданный промежуток времени. В классе определены следующие поля:

- `HashMap<String, HashMap<String,Integer>> successRateOnFormatDate` - структура данных для работы с запусками сборки относительно метрики SR, ключи даты за выбранный период, значения процент успешности выполнения запусков сборки за выбранный период;
- `String dateFormatKey` - поле, которое определяет формат даты за выбранный период, по которому будет происходить обработка и визуализация;
- `Logger LOGGER` - поле, с помощью которого записываются логи плагина при выполнении методов класса.

Также в классе определены методы:

- `BuildSuccessRateLogic(IntervalDate period, RunList<Run> buildList)` - конструктор класса;

- Map<String, Double> getSuccessRate() - метод, в котором происходит фильтрация данных запусков сборок по периоду, определение формата дат и вычисление процента успешности выполнения запусков сборок.

3.1.10. BuildTestCountLogic

Класс для работы с метрикой TS, в нем происходит пересчет параметров в зависимости от периода и настроек подданных на вход, а также высчитывается количество выполненных тестов во время работы сборок за определенный период. В классе определены следующие поля:

- HashMap<String, Integer> testCountOnFormatDate - структура данных для работы с запусками сборки относительно метрики ТС, ключи даты за выбранный период, значения количество выполненных тестов запусков сборки за выбранный период;
- String dateFormatKey - поле, которое определяет формат даты за выбранный период, по которому будет происходить обработка и визуализация;
- Logger LOGGER - поле, с помощью которого записываются логи плагина при выполнении методов класса.

Также в классе определены методы:

- BuildTestCountLogic(IntervalDate period, RunList<Run> buildList) - конструктор класса;
- Map<String, Integer> getTestCount() - метод, в котором происходит фильтрация данных запусков сборок по периоду и флагу failed, определение формата дат и вычисление количества выполненных тестов в процессе исполнения запусков сборки.

3.1.11. BuildTimeQueueLogic

Класс для работы с метрикой TQ, в нем происходит пересчет параметров в зависимости от периода и настроек подданных на вход, а также высчитывается время ожидание сборки в очереди в миллисекундах. В классе определены следующие поля:

- HashMap<String, Double> dateFormatDuration - структура данных для работы с запусками сборки относительно метрики TQ, ключи даты за выбранный период, значения время нахождения в очереди запусков сборки за выбранный период;

- String dateFormatKey - поле, которое определяет формат даты за выбранный период, по которому будет происходить обработка и визуализация;
- Logger LOGGER - поле, с помощью которого записываются логи плагина при выполнении методов класса.

Также в классе определены методы:

- BuildTimeQueueLogic(IntervalDate period, RunList<Run> buildList) - конструктор класса;
- Map<String, Double> getTimeQueue(Statistics statistics) - метод, в котором происходит фильтрация данных запусков сборок по периоду и флагу failed, определение формата дат и вычисление времени нахождения в очереди запусков сборок, а также расчет по статистической величине (например, среднее арифметическое), в зависимости от заданных настроек.

3.1.12. LinearRegressionHandler

Класс для прогнозирования метрик следующей сборки с помощью линейной регрессии с весовыми коэффициентами. В классе определено одно поле Logger LOGGER, с помощью которого записываются логи плагина при выполнении методов класса.

Также в классе определены методы:

- double linearRegression(double[] yUnweighted, double[] weights) - метод для прогнозирования метрик следующей сборки на основе весовых коэффициентов и предыдущих значений метрики за период;
- double[] calculateWeightMetric(double[] arrMetricValues) - метод расчета весовых коэффициентов, с учетом пустых значений, например, если за период не было запущено сборок.

3.1.13. Файлы JS и Jelly

В JS определяются функции событий для выбора элемента из выпадающего списка и взаимодействия с флажками. Для каждой метрики используется своя функция, внутри определяются настройки данных и отображения для визуализации отдельной метрики в виде определенного графика/диаграммы, вызывается метод для сортировки агрегированных по датам значений метрик в структуре JSON, а также формируются метки-подписи для каждого типа периода.

Также в JS определены следующие функции:

- `formatLabelsDate(arrLabels, dateFormat, period)` - функция, в которой происходит формирование меток-подписей к графикам в зависимости от формата дат и выбранного периода;
- `sortOnKeys(dict, period)` - функция в которой происходит сортировка значений словаря с данными о запусках сборок по ключам-датам;
- `createSuccessRateChart()` - функция в которой происходит подготовка данных, формирование настроек и создание графика по метрике SR;
- `typeChartHandler(typeChart, labels, title, dictValues)` - функция, которая обрабатывает событие изменение типа графика/диаграммы, и обновляет представление на странице плагина;
- `createTestCountChart()` - функция в которой происходит подготовка данных, формирование настроек и создание графика по метрике ТС;
- `createBuildDurationChart()` - функция в которой происходит подготовка данных, формирование настроек и создание графика по метрике BD;
- `createArtifactSizeChart()` - функция в которой происходит подготовка данных, формирование настроек и создание графика по метрике AS;
- `createTimeQueueChart()` - функция в которой происходит подготовка данных, формирование настроек и создание графика по метрике TQ.

В `jelly` файле с помощью `html` формируется структура документа, а также выполняется привязка `Java` объектов к объектам `JS`. Определяются обработчики событий, который при взаимодействии с пользователем вызывают определенный запрос-метод `AJAX`.

3.2. Результаты разработки плагина

При разработке плагина надо было учитывать, что требуется отображать все графики на одной странице задания друг под другом, поскольку при выборе одного периода, например, месяца, будет получена сводная информация по каждой сборке или нескольких сборок запущенных в один день. Графики отображаются посредством перехода на соответствующую ссылку, оставляя при этом пользователя в том же задании (странице с результатами последних сборок).

В интерфейсе у каждого графика были реализованы те дополнительные функции отображения, которые могут быть применены к визуализируемой метрике: отобразить статистику по упавшим сборкам/тестам, обработать метрику в соответствии со статистическим показателем.

Интерфейс страницы плагина с графиками в системе Jenkins на странице задания показан на рисунке 3.1.

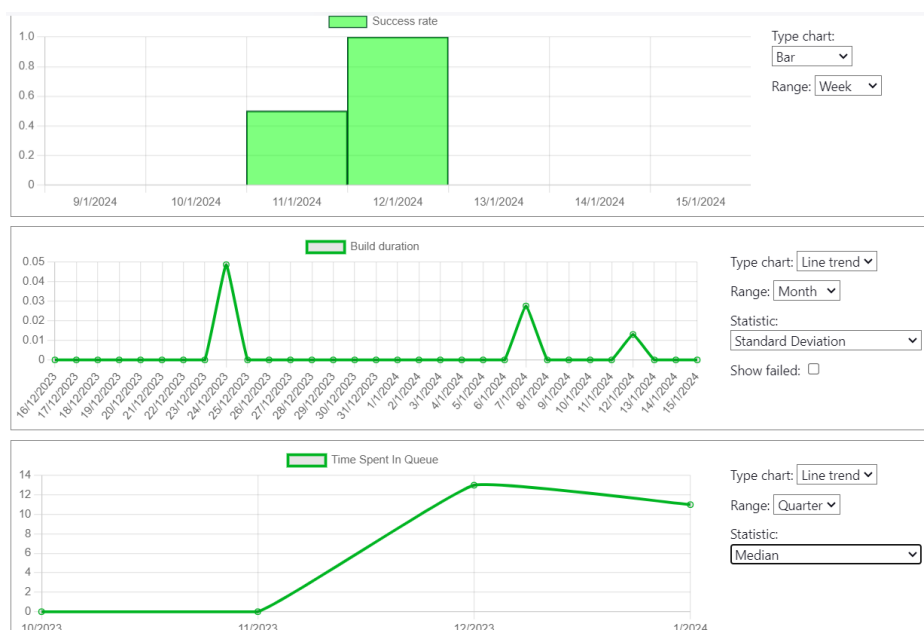


Рис.3.1. Интерфейс плагина Jenkins

Интерфейс страницы плагина в системе Jenkins с графиком AS на странице задания при выборе типа графика Radar, за период месяц, обработанные по показателю среднее арифметическое и учитывающий упавшие сборки представлен на рисунке 3.2.

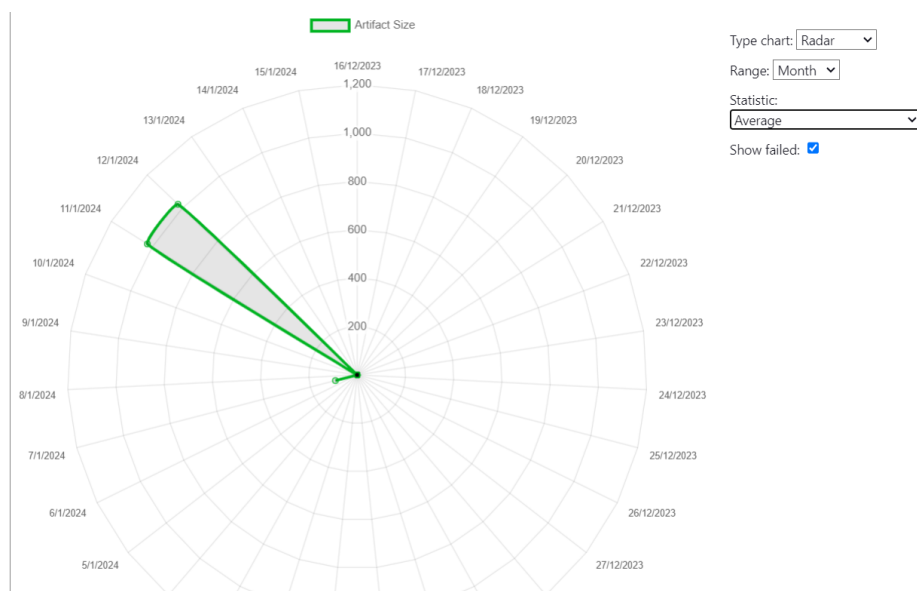
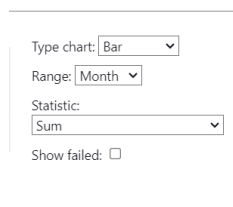


Рис.3.2. Интерфейс графика Radar для AS

Основное взаимодействие с графиками будет производить через меню, которое есть напротив каждого графика со своими параметрами, отображенном на рисунке 3.3:



Type chart: Bar
 Range: Month
 Statistic: Sum
 Show failed: ☐

Рис.3.3. Интерфейс элементов управления

При взаимодействии с раскрывающимся списком должен вызываться Java метод, который пересчитает и отфильтрует необходимые сборки Jenkins и динамически отобразит результаты по выбранным периоду, также динамически должна производить обработка метрик сборок, при выборе статистического показателя, а также включение в графики данных об упавших сборках, при выборе соответствующих чекбоксов.

Интерфейс изменения навигационной панели отображен на рисунке 3.4. В данном случае видно что изменения видны при открытии конфигурации конкретной сборки, т.е. не надо будет переключаться между окном плагина и сборкой для визуализации метрик, при открытии данного пункта меню, также происходит изменения URL, с которым в дальнейшем и происходит взаимодействие.

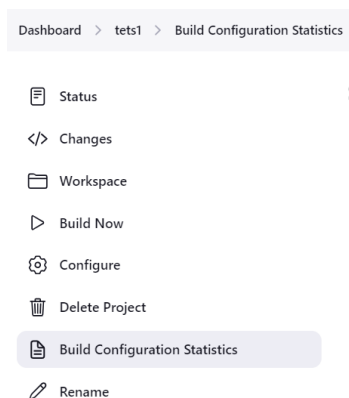


Рис.3.4. Интерфейс элементов управления

Код плагина представлен в приложении 1.

3.3. Выводы

В главе 3 была проведена реализация плагина, а также описаны классы, разработанные при написании плагина и файлы, которые участвуют во взаимодействии с этими классами и отображаемым интерфейсом пользователя. Также были приведены результаты разработки, приведены скриншоты интерфейсов, а также описаны добавленные на страницу Jenkins элементы, после установки плагина.

ГЛАВА 4. ТЕСТИРОВАНИЕ И АПРОБАЦИЯ ПЛАГИНА В JENKINS

В главе описано проведенное тестирование и апробация плагина:

- А. Описаны методы тестирования.
- В. Проведена апробация плагина в системе Jenkins.
- С. Разработан код тестирования плагина.

4.1. Методы тестирования

Тестирование программного обеспечения — обширное понятие, которое включает планирование, проектирование и, собственно, выполнение тестов [8]. В процессе CI/CD производится непрерывное тестирование разработанного кода, а также тестирование разработанного приложения. Сам плагин является частью CI/CD процессе, но также требует тестирования корректной работы своей функциональности, тестирование разработанного кода, а также тестирования на соответствие исходным требованиям, которые были предъявлены к разработке в главе 1.

Существует множество методов тестирования и техник тест дизайна, в процессе анализ функциональных требования были отобраны те, которые наиболее релевантные для разработанного плагина Jenkins.

Будет проведено как ручное, так и автоматизированное тестирование плагина. Ручное тестирование поможет выявить нетипичные тест-кейсы, которые не покрываются автоматизированными тестами.

Ручные тесты будут проведены методом черного ящика. Данный метод это процедура получения и выбора тестовых случаев на основе анализа функциональности и технического задания, без применения знаний о внутреннем устройстве системы [5].

Были составлены тест-кейсы, которые отражены в таблице 4.1.

В данном случае тест-кейсы были описаны с помощью техники тест-дизайна Матрица трассировки. Если обратиться к определению, то матрица трассировки — двумерная таблица, содержащая соответствие функциональных требований и подготовленных тестовых сценариев [4], а на пересечении столбцов и строк ставится метка, о том, что данное требование покрывается данным тест-кейсом. В случае данной работы из соображений удобства и оптимизации тестовой документации, было принято решение модифицировать матрицу трассировки и

Таблица 4.1

Составленные тест-кейсы

№	Описание	Ожидание
1	Проверка выбранного периода на графике SR (месяц)	Количество дней соответствует прошлому месяцу, на каждый день отображены корректные значения процента успешности сборок
2	Проверка выбранного периода на графике BD (неделя) с учетом выбора настройки среднего значения и упавших сборок	Количество дней 7 в соответствии со всеми днями недели, на каждый день отображены корректные значения среднего времени продолжительности сборок, учтены упавшие сборки
3	Проверка выбранного периода на графике TQ (квартал) с учетом выбора настройки среднего значения	Количество кварталов 4 в соответствии с кварталами года, на каждый квартал отображены средние значения проведенного в очереди времени сборок
4	Проверка выбранного периода на графике TC (год) с учетом выбора настройки упавших сборок	Количество месяцев 12 соответствует прошедшему году, на каждый месяц отображены корректные значения количества тестов, с учетом тестов выполненных на упавших сборках
5	Проверка выбранного периода на графике AS (день)	Количество часов 24 соответствует всем часам прошедшего дня, на каждый час отображены корректные значения размера итоговых артефактов полученного по результатам задания
6	Проверка корректного отображения аномальных значений	Отображены номера сборок, возле каждого графика, у которых аномальное значений метрики соответствующей графику
7	Проверка корректного расчета предугаданной 'следующей' сборки	Метрики предугаданной сборки рассчитываются корректно для каждого графика

совместить с подробным описанием в формате чек-листа: для оптимизации идет проверка, что каждый тип графика-метрики (Success Rate) корректно себя ведет на определенном периоде (неделя), таким образом не придется проверять, каждый график на каждом периоде при каждой доработке кода продукта.

Данные тест-кейсы оптимизированы, поскольку в соответствии с пирамидой тестирования [6] ручные UI кейсы, находятся в самой верхней ее части и не должны занимать достаточно большое место в системе тестирования. С другой стороны для улучшения покрытия требования, предъявляемых к продукту должны использоваться гораздо в большем объеме unit-тесты, т.е. тесты в которых самые маленькие компоненты системы - модули (модули, классы, методы), индивидуально проверяются на предмет правильной работы [40].

4.1.1. Unit тестирование

При написании юнит тестов используется метод белого ящика. Данный метод предоставляет тестирующему полное знание тестируемого приложения, включая доступ к исходному коду и проектной документации [42], т.е. тестирование происходит на основе знания исходного кода, таким образом, юнит тестирование методом белого ящика поможет нам достаточно широко покрыть все модули плагина. Для запуска тестов требуется перейти в корень директории плагина и выполнить команду `mvn test`.

Код юнит тестов расположен в классе `BuildConfigurationStatisticsBuilderTest`. В этом классе проводятся проверки, такие как:

- проверка корректности системы в целом - `testWorkingSystem()`;
- проверка успешного завершения сборки - `testSuccessBuildFromCustomBuild()`;
- проверка падения сборки при некорректных входных данных - `testFailBuildFromCustomBuild()`;
- проверка формирования структур для начальной инициализации данных - `testCreateDateWeekMapSuccessRate()`, `testCreateDateMonthMap()`;
- проверки корректности написанных методов работы с датой - `testDateMonthToString()`, `testGetLastMonthDays()`;
- проверка работоспособности модулей обработки времени сборок - `testGetTimeInQueue()`.

Код юнит тестов приведен в приложении 2.

4.1.2. BDD тестирование

Помимо юнит тестов, были написаны автоматизированные тесты с использованием методологии Behavior-driven development (BDD) или разработки через поведение. В соответствии с BDD тесты следует писать на естественном, удобочитаемом языке, ориентированном на поведение приложения [37].

Для написания BDD тестов использовался фреймворк Cucumber, использующий Gherkin нотацию для описания сценария тестов на естественном языке, а также заглушки для структур Jenkins.

Тесты написанные с BDD подходом, проверяют работоспособность основного функционала плагина, а именно получения обработанной информации по метрикам сборок, в соответствии с указанными настройками/фильтрами пользователя.

Код BDD тестов приведен в приложении 2.

4.1.3. UI тестирование

Также для автоматизации тестирования UI части плагина, был применен Selenium web driver и язык программирования python. WebDriver управляет браузером, как это делает пользователь, с использованием сервера Selenium [41]. Данные тест-кейсы будут в автоматическом режиме проверять реакцию элементов веб интерфейса на действия пользователя. Для запуска тестов требуется перейти в корень проекта Selenium и запустить команду pytest, при необходимости указать браузер и url, с которыми необходимо запустить UI тесты.

Код UI тестов расположен в отдельном проекте в классе TestCase. В этом классе проводятся проверки, такие как:

- проверка корректности открытия и наличия элементов во вкладке на странице плагина - `test_open_tab(self)`;
- проверка наличия и корректного отображения графика SR - `test_success_rate_chart(self)`;
- проверка наличия и корректного отображения графика BD - `test_build_duration_chart(self)`;
- проверка наличия и корректного отображения графика TC - `test_test_count_chart(self)`;
- проверка наличия и корректного отображения графика BQ - `test_time_spent_queue_chart(self)`;

- проверка наличия и корректного отображения графика AS - `test_artifacts_size_chart(self);`
- проверка корректности реакция элемента выпадающего списка - `test_change_value_select_period(self);`
- проверка корректности реакция чекбоксов - `test_change_value_checkbox(self);`

Код UI тестов приведен в приложении 3.

4.2. Аprobация плагина

Аprobация плагина будет проводится в системе CI Jenkins для которой и был разработан плагин визуализации. Для того чтобы провести аprobацию плагина на локальном сервере Jenkins, запущенном на локальном или удаленном ПК, потребуется произвести несколько операций. Для начала, нужно будет клонировать репозиторий с кодом плагина на GitHub, перейти в папку проекта и выполнить [35] `Maven mvn install` и скопировать `.hpi` в папку `/plugins/`.

Затем потребуется на запущенном сервере Jenkins перейти в Управление Jenkins и среди доступных плагинов выбрать Build Configuration Statistics, установить, после чего напротив каждой сборки Jenkins в боком меню, откуда можно запустить и отредактировать сборку, появится пункт меню Build Configuration Statistics, при нажатии на которой должны отобразиться все графики с собранной статистикой по метрикам каждого задания в Jenkins.

4.2.1. Подготовка и генерация набора сборок для аprobации

Для того чтобы графики отображали какие-то данные, необходимо сначала сгенерировать сборки разной длительности, статусов, с разным количеством тестов и размером артефактов.

Для генерации запусков сборки, можно задать конфигурацию сборки через меню Configuration, а затем с помощью действия Build Now в меню сборки, запустить сборку на выполнение. Также можно использовать плагин Pipeline, для того чтобы декларативно с помощью groovy скрипта задать конфигурацию сборки с шагами, которые будут выполнять при запуске сборки.

Пример скрипта для создания для создания сборки, в которой будет выполняться два шага: создание файла и создания артефакта сборки из созданного файла с помощью cmd Windows.

```

pipeline {
  agent any
  stages {
    stage('Create file') {
      steps {
        bat 'echo Hello World > myfile.txt'
      }
    }
    stage('Archive file') {
      steps {
        archiveArtifacts artifacts: 'myfile.txt',
          onlyIfSuccessful: true
      }
    }
  }
}

```

Для того чтобы проверить корректность обработки данных во времени, можно после запуска сборки, отредактировать (в логах выбранного запуска в папке запуска в файле build.xml) xml теги `<timestamp>1684077728000</timestamp>` и `<startTime>1684077728013</startTime>`, в которых в формате timestamp задать нужное время в прошлом. Для конвертации даты и времени в timestamp можно использовать веб-ресурс <https://www.epochconverter.com/>.

Например, для даты Sunday, May 14, 2023 3:22:08 PM получаем следующий результат в формате timestamp 1684077728000 в миллисекундах. После редактирования xml файла с информацией о сборке получим необходимое дату и время в интерфейсе Jenkins. Пример отредактированной сборки с датой в прошлом указан на рисунке 4.1.

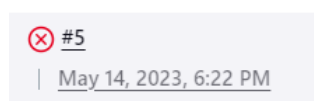


Рис.4.1. Сборка с датой в прошлом

В ходе апробации плагина были сгенерированы сборки, которые отображены на рисунке 4.2.

Запуски были сгенерированы с разной датой начала:

- 5 успешных запусков с датой 24.12.2023, с разным временем начала с 2 до 9 часов вечера;
- 1 успешный запуск 4.01.2024;

✓ #20	7 янв. 2024 г., 21:17
✗ #19	7 янв. 2024 г., 21:16
✓ #18	7 янв. 2024 г., 21:13
✓ #17	7 янв. 2024 г., 21:06
✗ #16	7 янв. 2024 г., 2:44
✗ #15	7 янв. 2024 г., 2:42
✗ #14	7 янв. 2024 г., 1:27
✗ #13	7 янв. 2024 г., 1:25
✓ #12	7 янв. 2024 г., 1:24
✗ #11	7 янв. 2024 г., 1:21
✗ #10	7 янв. 2024 г., 1:21
✓ #9	5 янв. 2024 г., 23:12
✗ #8	5 янв. 2024 г., 23:12
✗ #7	5 янв. 2024 г., 00:12
✓ #6	4 янв. 2024 г., 17:15
✓ #5	
✓ #4	24 дек. 2023 г., 21:27
✓ #3	24 дек. 2023 г., 16:45
✓ #2	24 дек. 2023 г., 16:45
✓ #1	24 дек. 2023 г., 14:40
✓ #1	24 дек. 2023 г., 14:40

Рис.4.2. Сгенерированные сборки

- 3 запуска 5.01.2024 - 2 из которых закончилось с результатом падение (в 0 часов и 23 часа), а один с положительным результатом в 23 часа;
- 11 запусков 7.01.2024 из которых 7 закончилось падением, а 4 с успешным результатом, запуски имеют разное время начала с 1 до 21 часа ;
- 2 запуска 11.01.2024 один из которых закончился падением с временем начала 14 часов, а другой с успешным результатом в 14 часов;
- 2 успешных запуска 12.01.2024 со временем начала 13 часов и 16 часов.

Среди сборок присутствуют, упавшие сборки со специально завышенным временем выполнения 100 секунд, сборки без завышенного времени выполнялись около 20 секунд. Такая сборка отображена на рисунке 4.3.



Рис.4.3. Длинная упавшая сборка

А также сборки, с созданными артефактами, часть из которых в статусе успешного выполнения, с короткой продолжительностью, а часть из которых завершены падением с большой продолжительностью выполнения. Были определены разные файлы-артефакты для генерации с размером от 70 байт до 1 Кб. Перенос созданных файлов в артефакты выполнялся с помощью конфигурационного раздела в сборке Post-build Actions, в котором выплавлялась команда Archive the artifacts. Сборка с артефактом отображена на рисунке 4.4.



Рис.4.4. Короткая успешная сборка с артефактом

Часть сборок выполнялось посредством созданного класса в плагине `BuildConfigurationStatisticsBuilder`, который добавлял еще один вариант запуска шагов сборки `Build Steps`. Этот класс выполнял вывод информации о текущем запуске в консоль, а также вывод информации с именами всех запусков, уже выполненных в сборке, а также вывод параметра, который задавался при создании шага в конфигурации сборки. Информация из консоли с процессом выполнения шага представлена на рисунке 4.5.

```
Started by user unknown or anonymous
Running as SYSTEM
Building in workspace C:\buildConfigurationStatistics\work\workspace\tets1
Hello, Andrei!
Hello, tets1 #4!
Hello, [tets1 #4, tets1 #3, tets1 #2, tets1 #1]!
Hello, <?xml version='1.1' encoding='UTF-8'?>
<project>
  <description></description>
  <keepDependencies>false</keepDependencies>
  <properties/>
  <scm class="hudson.scm.NullSCM"/>
  <canRoam>true</canRoam>
  <disabled>false</disabled>
  <blockBuildWhenDownstreamBuilding>false</blockBuildWhenDownstreamBuilding>
  <blockBuildWhenUpstreamBuilding>false</blockBuildWhenUpstreamBuilding>
  <triggers/>
  <concurrentBuild>false</concurrentBuild>
  <builders>
    <io.jenkins.plugins.sample.BuildConfigurationStatisticsBuilder plugin="buildConfigurationStatistics@1.0-SNAPSHOT">
      <name>Andrei</name>
    </io.jenkins.plugins.sample.BuildConfigurationStatisticsBuilder>
  </builders>
  <publishers/>
  <buildWrappers/>
</project>!
Finished: SUCCESS
```

Рис.4.5. Консоль шага с разработанным Builder

Для части сборок был добавлен 1 шаг с выполнением `cmd` команды `echo 123`, для создания упавших сборок, команды `cmd` намеренно прописывались с ошибками в синтаксисе, например `echo1 123`.

Для того чтобы увеличить время выполнения сборок использовалась команда `cmd` `waitfor SomethingThatIsNeverHappening /t 100 2>NUL`, которая обеспечивала время выполнения запуска 100 секунд.

- A. Сделать fork проекта в личный репозиторий.
- B. Найти подходящий коммит, в котором были выполнены значительные изменения (более 500 строк измененного кода).
- C. Откатиться до найденного коммита.
- D. Создание новой ветки на основе коммита, до которого произошел откат.
- E. Отправка ветки в личный удаленный репозиторий на GitHub.

После того как произведен откат до выбранного коммита, необходимо повторить процедуру начиная с коммита, до которого был произведен откат. Процедура была повторена 12 раз т.е. было сгенерировано 12 версий проекта на 12 месяцев года (для генерации сборок на год).

Для отката к более ранним версиям был написан скрипт на языке Python, представленный в приложении П4.

Далее при генерации сборок было произведено по 2 запуска на каждую версию продукта, время в каждой сборке было отредактировано на каждый месяц за прошедший год. Для редактирования времени запуска сборки был написан скрипт на языке Python, представленный в приложении П4, который обрабатывает xml файлы с информацией о сборках. Перед выполнением каждого запуска был отредактирован параметр, по которому определяется из какой ветки берется исходный код продукта.

При формировании графиков на апробируемом проекте, даты запуска сборок были отредактированы по месяцам года (одна версия - один месяц), а не по реальным датам коммитов в репозитории. Рекомендуется смотреть на этап жизненного цикла продукта и проводить визуализацию по кварталам или месяцам.

Сгенерированные на этом проекте сборки отображены на рисунке 4.6.

Результаты работы плагина на описанном выше проекте отображены на рисунках 4.7-8. На рисунке 4.7 видно на графике SR, как менялся процент успешности сборок в течении года на столбчатой диаграмме. Также видно на графике BD динамику изменения продолжительности сборок за последний год на линейном графике, в данном случае можно отследить как менялось среднее значение продолжительности. Видно, что время сборки незначительно увеличивалось, т.е. при увеличении кода приложения, в продолжительности сборки также увеличивалось время, за исключением 8 и 11 месяцев.

В 8 месяце в изменениях кода была повышена версия npm и node, что оптимизировало время выполнения сборки, которое уменьшилось с 19.3 до 18.3

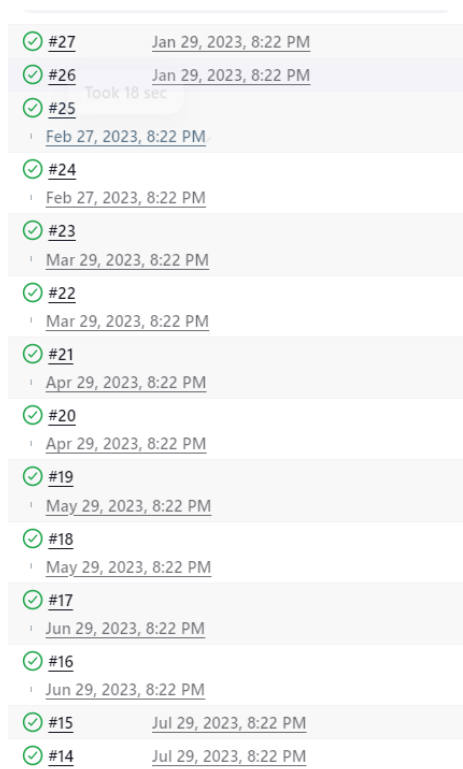


Рис.4.6. Сгенерированные сборки реального проекта

Statistics for job test-Build-frontend-maven-plugin_version



Рис.4.7. Результаты апробации на реальном проекте SR, BD

секунд. А версия в 11 месяце, вероятно была не протестировано перед загрузкой в главную ветку, поскольку результат из 3 запусков сборки закончился падением. Что также видно на графике SR, т.е. можно сделать вывод что в коммите был дефект, а не оптимизация, которая ускорила время сборки в несколько раз.

На рисунке 4.8 можно увидеть с помощью радарной диаграммы общий размер, сгенерированных логов-артефактов за последний год. Видно, что с каждым месяцем размер артефактов увеличивался, что также можно объяснить увеличением исходного кода продукта, также наглядна видна разницу между первым и

последним месяцем года, а также то что при отображении упавших сборок видно, то что генерировались артефакты, хоть и с небольшим размером.

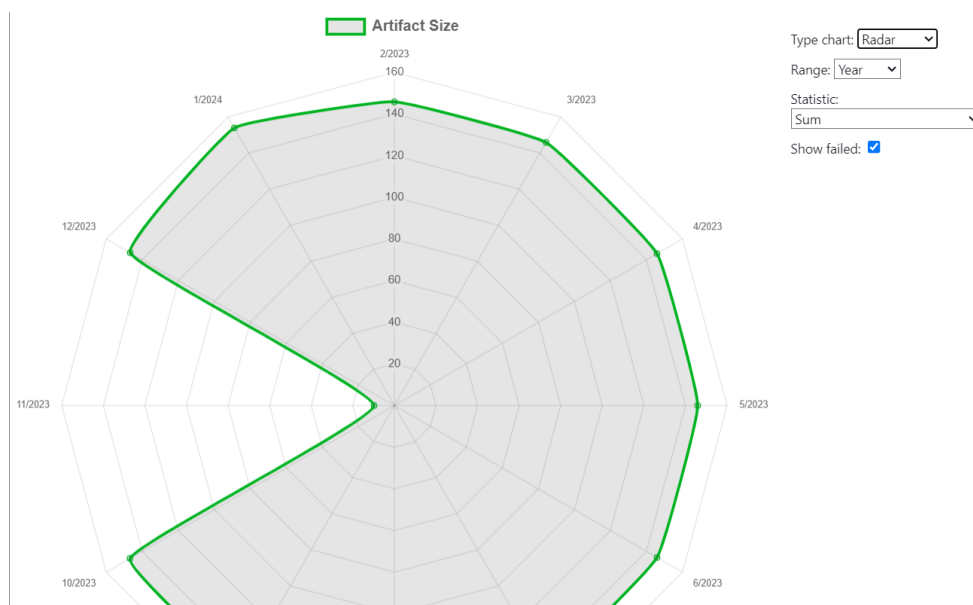


Рис.4.8. Результаты апробации на реальном проекте AS

По описанным выше результатам визуализации можно прийти к выводу, что плагин полезен тем, что отображает изменение различных метрик запусков сборки с течением времени, т.е. можно увидеть тенденцию изменений в созданных сборках Jenkins в течении цикла разработки за нужный период времени и если метрики в какой момент изменили свои значения, можно определить, что это за момент (или период) и проанализировать как изменения в сборке, тестах или коде могли повлиять на это.

Для удобства пользователей, как видно на рисунках выше, результаты отображаются на разных типах диаграмм, чтобы каждой участник команды мог изучать динамику изменения метрик, так как ему удобно. Также видно, что возле метрик BD, SR, AS можно выбрать статистический показатель, в соответствии с которым будут обработана метрика. Тем самым, можно определить является отклонение случайностью, нестабильностью сборки или это какая-то закономерность.

После анализа разработчики, тестировщики и DevOps-инженеры могут принять решение, насколько критичны данные изменения для процессов CI/CD и если потребуется оптимизировать сборки, тесты или, возможно, какую-то часть кода.

Также по данным диаграммам можно обнаружить и другие проблемы, например, аномалии в процессах сборки или тестирования или окружении, в котором производится сборка или установка компонентов системы.

Таблица 4.2

Количественные результаты работы

Критерий	Разработанное решение	TeamCity	Build Monitor Plugin	Global Build Stats Plugin	Build Time Blame
Количество визуализируемых метрик	5	5	1	2	2
Количество статистических показателей	7	1	0	1	1
Количество типов диаграмм	3	2	1	1	1

4.2.3. Оценка результатов работы

Для того чтобы оценить результаты работы, будет проведено сравнение функционала, который присутствует в аналогичных решениях: плагинах, сравнительный анализ, которых проводился в разделе 1.5 и модуль Statistics, реализованный в средстве CI TeamCity, который и послужил причиной для создания аналогичного модуля в Jenkins.

В разработанном плагине реализовано 7 статистических показателей, которые применяются к метрикам сборок, что является значительным преимуществом в сравнении с аналогичными решениями, поскольку в аналогичных плагинах Jenkins, а также в модуле TeamCity реализован только расчет среднего арифметического значения, и при том не во всех аналогичных плагинах Jenkins.

Также преимуществом разработанного плагина является наличие 3 типов диаграмм для каждой метрики, что больше чем у всех аналогичных решений.

Все результаты сравнения с аналогичными решениями приведены в таблице 4.2.

Если сравнивать по количеству визуализируемых метрик, то видно, что все 5 метрик, которые были реализованы в TeamCity, удалось реализовать и в

разработанном плагине, аналогичные плагины Jenkins визуализируют определенные из перечисленных в разделе 1.5 метрик, но их количество меньше 5.

4.3. Выводы

После проведения этапа апробации и тестирования плагина визуализации статистики сборок Jenkins можно прийти к выводу, что тестирование проведено в полном объеме и затронуло разные методы, техники тест-дизайна и соответствует методологиям и устоявшимся практикам тестирования программного обеспечения. Апробация протестированного плагина, дала понять, что плагин корректно отображает при разных поданных на вход исходных данных и настройках. Корректно высчитываются и визуализируются статистические показатели обработанных метрик:

- среднее арифметическое;
- мода;
- медиана;
- размах;
- среднеквадратическое отклонение;
- среднеквадратическое отклонение несмещенное;
- дисперсия.

Визуализация метрик со статистическими показателями была проверена на различных типах диаграмм/графиков:

- столбчатая диаграмма;
- линейный тренд;
- радарная диаграмма.

В результате разработки автоматизированных тестов было разработано 22 юнит теста, 2 BDD теста и 8 UI тестов.

Работоспособность плагина проверена на проекте открытым с исходным кодом frontend-maven-plugin. В ходе апробации были получено 24 запуска сборок на 12 разных версий продукта, в том числе 2 упавших запуска на одной из версий. Также было написано 2 python скрипта для формирования окружения для апробации.

Плагин рекомендуется использовать для:

- прогнозирования метрик будущих сборок за выбранный период для планирования и оптимизации процесса разработки;

- отслеживания тенденций по изменению метрик сборки за период, в том числе выявления аномальных значений (резкое увеличение времени сборок и размера артефактов после добавления нового кода);
- по отслеженным данным можно определить узкие места в процессе сборки и оптимизировать этот процесс для ускорения развертывания;
- по отслеженным данным можно оценить эффективность CI/CD процесса, с целью улучшения процесса поставки;
- по отслеженным данным можно оценить качество кода и эффективности тестов и принять решения по улучшению этого качества.

ЗАКЛЮЧЕНИЕ

В результате проведенной работы был разработан прототип плагина для визуализации статистики сборок Jenkins. Была проанализирована предметная область, проведен сравнительный анализ аналогичных решений.

Были выбраны средства и инструменты разработки, спроектирована архитектура плагина, описаны функциональные возможности, а также разработан программный код и интерфейс плагина.

В процессе проектирования и реализация были выбраны статистические показатели и типы диаграмм, по которым должна происходить визуализация метрик сборок Jenkins. Было проведено тестирование плагина различными методами и апробация на реальном проекте `frontend-maven-plugin` (более 4 тысяч звезд и 863 forks).

Плагин рекомендуется использовать для: оценки эффективности написанного кода и тестирования, а также для оптимизации и улучшения процессов CI/CD, разработки и тестирования.

По итогу реализации объем кода проекта составляет 3225 строк, из которых:

- 1686 строк - Java код на сервере Jenkins;
- 788 строк - Jelly и JS на клиентской части;
- 263 строк - Java unit тесты;
- 214 строк - Java bdd тесты;
- 188 строк - Python код UI тестов;
- 86 строк - python скрипты для апробации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 5 сборщиков модулей для приложений Node.js. — URL: <https://tproger.ru/articles/5-razlichnyh-instrumentov-dlya-obedineniya-prilozhenij-node-js> (дата обращения: 20.11.2023).
2. Как использовать GitLab в условиях санкций? — URL: <https://habr.com/ru/companies/ruvds/articles/715010/> (дата обращения: 20.11.2023).
3. Краткий обзор методологии CI/CD: принципы, этапы, плюсы и минусы. — URL: <https://cloud.ru/ru/blog/cicd-about> (дата обращения: 20.11.2023).
4. Матрица трассабили. — URL: <https://habr.com/ru/companies/simbirsoft/articles/412677/> (дата обращения: 22.12.2023).
5. Панина О. Особенности тестирования «черного ящика». — 2017. — URL: <https://quality-lab.ru/blog/key-principles-of-black-box-testing/> (дата обращения: 22.12.2023).
6. Подробнее про пирамиду тестирования. — URL: <https://habr.com/ru/articles/672484/> (дата обращения: 22.12.2023).
7. Соколова А. Подходит CI/CD вашему бизнесу: плюсы и минусы конвейеров // Cloud Networks. — 2021. — URL: <https://cloudnetworks.ru/analitika/podhodit-ci-cd-vashemu-biznesu-plyusy-i-minusy-konvejerov/> (дата обращения: 20.11.2023).
8. Тестирование ПО: суть профессии, требования и заработная плата. — URL: https://habr.com/ru/companies/habr_career/articles/517812/ (дата обращения: 22.12.2023).
9. Учимся создавать и настраивать Jenkins Jobs. — URL: <https://habr.com/ru/companies/slurm/articles/742504/> (дата обращения: 20.11.2023).
10. Что такое Gradle. — URL: <https://appttractor.ru/develop/gradle.html> (дата обращения: 20.11.2023).
11. Что такое сборка в программировании. — URL: <https://uchet-jkh.ru/i/cto-takoe-sborka-v-programmirovanii> (дата обращения: 20.11.2023).
12. Jenkins-CI, an Open-Source Continuous Integration System, as a Scientific Data and Image-Processing Platform / I. Moutsatsos [и др.] // Journal of Biomolecular Screening. — 2016. — Т. 22. — С. 1087057116679993. — DOI 10.1177/1087057116679993.
13. Apache Commons Math Documentation. — URL: <https://commons.apache.org/proper/commons-math/> (visited on 20.11.2023).

14. Bamboo Docs. — URL: <https://confluence.atlassian.com/bamboo/bamboo-documentation-289276551.html> (visited on 20.11.2023).
15. Build Monitor View. — URL: <https://plugins.jenkins.io/build-monitor-plugin/> (visited on 20.11.2023).
16. Build Time Blame. — URL: <https://plugins.jenkins.io/build-time-blame/> (visited on 20.11.2023).
17. Chart.js Documentation. — URL: <https://www.chartjs.org/docs/latest/> (visited on 20.11.2023).
18. CI/CD. — URL: <https://blog.skillfactory.ru/glossary/ci-cd> (visited on 20.11.2023).
19. CI/CD Tools Comparison: Jenkins, TeamCity, Bamboo, Travis CI, and More. — URL: <https://www.altexsoft.com/blog/cicd-tools-comparison/> (visited on 20.11.2023).
20. Circle CI Docs. — URL: <https://circleci.com/docs/about-circleci/> (visited on 20.11.2023).
21. CSS Documentation. — URL: <https://developer.mozilla.org/en-US/docs/Web/CSS> (visited on 20.11.2023).
22. frontend-maven-plugin. — URL: <https://github.com/eirslett/frontend-maven-plugin> (visited on 25.12.2023).
23. GitLab CI Docs. — URL: <https://docs.gitlab.com/ee/ci/> (visited on 20.11.2023).
24. global-build-stats. — URL: <https://plugins.jenkins.io/global-build-stats/> (visited on 20.11.2023).
25. Groovy Docs. — URL: <https://groovy-lang.org/documentation.html> (visited on 20.11.2023).
26. Gson Documentation. — URL: <https://github.com/google/gson> (visited on 20.11.2023).
27. *Irani Z.* 5 common pitfalls of CI/CD—and how to avoid them // Cloud Networks. — 2017. — URL: <https://www.infoworld.com/article/3113680/5-common-pitfalls-of-cicd-and-how-to-avoid-them.html> (visited on 20.11.2023).
28. Java Docs. — URL: <https://docs.oracle.com/en/java/> (visited on 02.10.2023).
29. JavaScript Documentation. — URL: <https://developer.mozilla.org/ru/docs/Web/JavaScript> (visited on 20.11.2023).
30. Jelly Documentation. — URL: <https://commons.apache.org/proper/commons-jelly/> (visited on 20.11.2023).

31. Jenkins Documentation. — URL: <https://www.jenkins.io/doc/> (visited on 02.10.2023).
32. Junit Documentation. — URL: <https://junit.org/junit5/docs/current/user-guide/> (visited on 20.11.2023).
33. Maven Documentation. — URL: <https://maven.apache.org/what-is-maven.html> (visited on 20.11.2023).
34. Mockito Documentation. — URL: <https://site.mockito.org/> (visited on 20.11.2023).
35. *Pathare A.* Tutorial: Developing Complex Plugins for Jenkins. — 2022. — URL: <https://www.velotio.com/engineering-blog/jenkins-plugin-development> (visited on 22.12.2023).
36. *Puzhevich V.* Groovy vs Java: Detailed Comparison and Tips on the Language Choice. — 2020. — URL: <https://scand.com/company/blog/groovy-vs-java/> (visited on 20.11.2023).
37. Quick Guide to BDDMockito. — URL: <https://www.baeldung.com/bdd-mockito> (visited on 22.12.2023).
38. TeamCity Docs. — URL: <https://www.jetbrains.com/help/teamcity/teamcity-documentation.html> (visited on 02.10.2023).
39. The architecture of Jenkins plugins. — URL: <https://subscription.packtpub.com/book/programming/9781784390891/10/ch10lvl1sec62/the-architecture-of-jenkins-plugins> (visited on 20.11.2023).
40. Unit testing. — URL: <https://www.techtarget.com/searchsoftwarequality/definition/unit-testing> (visited on 22.12.2023).
41. WebDriver Docs. — URL: <https://www.selenium.dev/documentation/webdriver/> (visited on 20.12.2023).
42. What is White Box Testing? — URL: <https://www.checkpoint.com/cyber-hub/cyber-security/what-is-white-box-testing> (visited on 22.12.2023).

Программный код плагина

Основной класс BuildConfigurationStatisticsAction.

```
package io.jenkins.plugins.sample;

import com.google.gson.Gson;
5 import hudson.model.Action;
import hudson.model.Job;
import org.kohsuke.stapler.bind.JavaScriptMethod;

import java.text.DecimalFormat;
10 import java.text.DecimalFormatSymbols;
import java.text.ParseException;
import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;
15 import java.util.stream.Collectors;

public class BuildConfigurationStatisticsAction implements
    Action {
    private Job job;

20     public BuildConfigurationStatisticsAction(Job job) {
        this.job = job;
    }

25     @Override
    public String getIconFileName() {
        return "document.png";
    }

30     @Override
    public String getDisplayName() {
        return "Build Configuration Statistics";
    }

35     @Override
    public String getUrlName() {
        return "buildConfigurationStatistics";
    }
}
```

```

40 public Job getJob() {
    return job;
}

@JavaScriptMethod
45 public String getBuildDuration(String period, String fail,
    String statistics) throws ParseException {
    Logger LOGGER = Logger.getLogger("uuu");
    LOGGER.log(Level.INFO, "arg jelly: " + period);
    LOGGER.log(Level.INFO, "failed status: " + fail);
    LOGGER.log(Level.INFO, "statistics status: " +
50     statistics);
    IntervalDate intreval = IntervalDate.valueOf(period);
    Statistics statisticsEnum = Statistics.valueOf(
        statistics);
    Boolean failed = fail.equals("1");
    //Boolean averageTime = average.equals("1");
    Gson gson = new Gson();
55 Map<String, Double> map = new BuildDurationLogic(
    intreval, failed, job.getBuilds()).getBuildsDuration(
    statisticsEnum);
    String gsonData = gson.toJson(map);
    LOGGER.log(Level.INFO, "gson: " + gsonData);
    return gsonData;
}

60 @JavaScriptMethod
public String getBuildSuccessRate(String period) throws
    ParseException {
    Logger LOGGER = Logger.getLogger("uuu1");
    LOGGER.log(Level.INFO, "arg jelly period success: " +
        period);
    IntervalDate intreval = IntervalDate.valueOf(period);
65 Gson gson = new Gson();
    Map<String, Double> map = new BuildSuccessRateLogic(
    intreval, job.getBuilds()).getSuccessRate();
    String gsonData = gson.toJson(map);
    LOGGER.log(Level.INFO, "gson: " + gsonData);
    return gsonData;
70 }

@JavaScriptMethod
public String getBuildArtifactSize(String period, String
    fail, String statistics) throws ParseException {
    Logger LOGGER = Logger.getLogger("artifact");

```

```

75     LOGGER.log(Level.INFO, "arg jelly artifact: " + period
        );
        LOGGER.log(Level.INFO, "failed artifact: " + fail);
        LOGGER.log(Level.INFO, "statistics artifact: " +
            statistics);
        IntervalDate intreval = IntervalDate.valueOf(period);
        Statistics statisticsEnum = Statistics.valueOf(
            statistics);
80     Boolean failed = fail.equals("1");
    //     Boolean averageTime = average.equals("1");

        Gson gson = new Gson();
        Map<String, Double> map = new BuildArtifactSizeLogic(
            intreval, failed, job.getBuilds()).getArtifactSize(
            statisticsEnum);
85     String gsonData = gson.toJson(map);
        LOGGER.log(Level.INFO, "gson artifact: " + gsonData);
        return gsonData;
    }

90     @JavaScriptMethod
    public String getBuildTestCount(String period, String fail
        ) throws ParseException {
        Logger LOGGER = Logger.getLogger("TestCount");
        LOGGER.log(Level.INFO, "arg jelly TestCount: " +
            period);
        LOGGER.log(Level.INFO, "failed TestCount: " + fail);
95     IntervalDate intreval = IntervalDate.valueOf(period);
        Boolean failed = fail.equals("1");

        Gson gson = new Gson();
        Map<String, Integer> map = new BuildTestCountLogic(
            intreval, job.getBuilds()).getTestCount();
100    String gsonData = gson.toJson(map);
        LOGGER.log(Level.INFO, "gson TestCount: " + gsonData);
        return gsonData;
    }

105    @JavaScriptMethod
    public String getBuildTimeQueue(String period, String
        statistics) throws ParseException {
        Logger LOGGER = Logger.getLogger("queue");
        LOGGER.log(Level.INFO, "arg jelly queue: " + period);
        LOGGER.log(Level.INFO, "statistics queue: " +
            statistics);

```

```

110         IntervalDate intreval = IntervalDate.valueOf(period);
        Statistics statisticsEnum = Statistics.valueOf(
            statistics);
        // Boolean averageTime = average.equals("1");

        Gson gson = new Gson();
115         Map<String, Double> map = new BuildTimeQueueLogic(
            intreval, job.getBuilds()).getTimeQueue(
            statisticsEnum);
        String gsonData = gson.toJson(map);
        LOGGER.log(Level.INFO, "gson TimeQueue: " + gsonData);
        return gsonData;

120     }

    @JavaScriptMethod
    public double getPredicted(String period, String metric)
        throws ParseException {
        Logger LOGGER = Logger.getLogger("getPredicted");
125         LOGGER.log(Level.INFO, "getPredicted period: " +
            period);
        IntervalDate intreval = IntervalDate.valueOf(period);
        Map<String, Double> map;

        if (metric.equals("BD")) {
130             map = new BuildDurationLogic(intreval, true, job.
                getBuilds()).getBuildsDuration(Statistics.AVG);
        } else {
            map = new BuildArtifactSizeLogic(intreval, true,
                job.getBuilds()).getArtifactSize(Statistics.AVG
            );
        }

135

        String formatDate;

        Map<Long, Double> newMap = new HashMap<Long, Double>()
            ;
        for(Map.Entry<String, Double> entry : map.entrySet())
        {
140             //LOGGER.log(Level.INFO, "map entrySet: " + map.
                entrySet());
            LOGGER.log(Level.INFO, "entrySet: " + entry.getKey
                () + " - " + entry.getValue());

```

```

    LOGGER.log(Level.INFO, "string date zero: " +
        DateTimeHandler.dateSetZeroDay(entry.getKey()))
    ;
    String parsedDate;
    if (intreval == IntervalDate.YEAR || intreval ==
        IntervalDate.QUARTER) {
145         parsedDate = DateTimeHandler.dateSetZeroDay(
            entry.getKey());
        formatDate = "yyyy-MM-dd";
    } else if (intreval == IntervalDate.DAY){
        parsedDate = entry.getKey();
        formatDate = "yyyy-MM-dd HH";
150     }
    else {
        parsedDate = entry.getKey();
        formatDate = "yyyy-MM-dd";
155     }

    LOGGER.log(Level.INFO, "string date: " +
        DateTimeHandler.convertStringToDate(parsedDate,
            formatDate));
    LOGGER.log(Level.INFO, "long time: " +
        DateTimeHandler.convertDateToLongTime(
            DateTimeHandler.convertStringToDate(parsedDate,
                formatDate)));
    newMap.put(DateTimeHandler.convertDateToLongTime(
        DateTimeHandler.convertStringToDate(parsedDate,
            formatDate)), entry.getValue());
    }
160 LOGGER.log(Level.INFO, "newMap: " + newMap.keySet());
    SortedSet<Long> keys = new TreeSet<>(newMap.keySet());
    double[] listValuesMetric = new double[keys.size()];

    int i = 0;
165 for (Long key : keys) {
        Double value = newMap.get(key);
        LOGGER.log(Level.INFO, "sorted key and value: " +
            key + " - " + value);
        // do something
        listValuesMetric[i] = value;
170 LOGGER.log(Level.INFO, "listValuesMetric[i]: " +
            listValuesMetric[i]);
        LOGGER.log(Level.INFO, "listValuesMetric: " +
            Arrays.toString(listValuesMetric));
        i++;

```



```

    }
    LOGGER.log(Level.INFO, "listValuesMetric: " + Arrays.
        toString(listValuesMetric));
175 double[] arrWeights = LinearRegressionHandler.
        calculateWeightMetric(listValuesMetric);
    LOGGER.log(Level.INFO, "arrWeights: " + Arrays.
        toString(arrWeights));
    LOGGER.log(Level.INFO, "listValuesMetric: " + Arrays.
        toString(listValuesMetric));
    double predictedNextValue = LinearRegressionHandler.
        linearRegression(listValuesMetric, arrWeights);

180    DecimalFormatSymbols separator = new
        DecimalFormatSymbols();
    separator.setDecimalSeparator('.');

    DecimalFormat df = new DecimalFormat("#.##", separator
        );

185    LOGGER.log(Level.INFO, "predictedNextValue format: " +
        df.format(predictedNextValue));
    predictedNextValue = Double.valueOf(df.format(
        predictedNextValue));

    return predictedNextValue;

190 }
}

```

Класс для добавления действия BuildConfigurationStatisticsFactory.

```

package io.jenkins.plugins.sample;

import hudson.Extension;
5 import hudson.model.Action;
import hudson.model.Job;
import java.util.Collection;
import java.util.Collections;
import javax.annotation.Nonnull;
10 import jenkins.model.TransientActionFactory;

@Extension
public class BuildConfigurationStatisticsFactory extends
    TransientActionFactory<Job> {
    @Override
15    public Class<Job> type() {

```

```

        return Job.class;
    }

    @Nonnull
    @Override
20 public Collection<? extends Action> createFor(@Nonnull Job
        job) {
        return Collections.singletonList(new
            BuildConfigurationStatisticsAction(job));
    }
}

```

Класс для обработки продолжительности сборки BuildDurationLogic.

```

package io.jenkins.plugins.sample;

import hudson.model.AbstractBuild;
5 import hudson.model.Run;
import hudson.model.Queue;
import hudson.util.RunList;
import jenkins.model.Jenkins;

10 import java.io.IOException;
import java.text.ParseException;
import java.util.*;
import java.util.function.IntFunction;
import java.util.logging.Level;
15 import java.util.logging.Logger;
import java.util.concurrent.TimeUnit;
import hudson.model.AbstractBuild;
import org.apache.commons.math3.stat.StatUtils;
import org.apache.commons.math3.stat.descriptive.
    DescriptiveStatistics;
20 import org.apache.commons.math3.util.FastMath;

public class BuildDurationLogic extends BuildLogic {
    static Logger LOGGER = Logger.getLogger(BuildDurationLogic
        .class.getName());
    HashMap<String, List<Double>> dateFormatDuration;
25 //HashMap<String, List<Double>>
    dateFormatDurationListValues;
    String dateFormatKey;

    public BuildDurationLogic(IntervalDate period, Boolean
        failed, RunList<Run> buildList) {
        super(period, failed, buildList);
    }
}

```

```

30     }

    public Map<String, Double> getBuildsDuration(Statistics
        statistics) throws ParseException {
        filterPeriodBuild();
        filterFailedBuild();
35     switch (this.period){
        case MONTH:
            dateFormatDuration = DateTimeHandler.
                createDateMonthMap();
            //dateFormatDurationListValues =
            //    DateTimeHandler.createDateMonthMap();
            dateFormatKey = "yyyy-MM-dd";
40            break;
        case WEEK:
            dateFormatDuration = DateTimeHandler.
                createDateWeekMap();
            dateFormatKey = "yyyy-MM-dd";
            break;
45        case YEAR:
            dateFormatDuration = DateTimeHandler.
                createDateYearMap();
            dateFormatKey = "yyyy-MM";
            break;
        case QUARTER:
50            dateFormatDuration = DateTimeHandler.
                createDateQuarterMap();
            dateFormatKey = "yyyy-MM";
            break;
        case DAY:
            dateFormatDuration = DateTimeHandler.
                createDateDayMap();
55            dateFormatKey = "yyyy-MM-dd HH";
            break;
        case ALL:
            dateFormatDuration = DateTimeHandler.
                createDateAllMap(this.buildList);
            dateFormatKey = "yyyy-MM-dd";
60            break;
    }

    //HashMap<String, Integer> dayDurationAverage = new
    //    HashMap<>();

```

```

65     LOGGER.log(Level.WARNING, "buildList: " + (this.
        buildList));

    for (Run run : this.buildList) {
        String dateFormatKeyAfterCheckPeriod =
70             DateTimeHandler.dateToString(
                DateTimeHandler.
                    convertLongTimeToDate(
                        run.getStartTimeInMillis()
                    ), dateFormatKey
                );
75     LOGGER.log(Level.INFO, "
        dateFormatKeyAfterCheckPeriod: " +
        dateFormatKeyAfterCheckPeriod);
        if (this.period == IntervalDate.DAY) {
            dateFormatKeyAfterCheckPeriod =
                DateTimeHandler.dateSetZeroMinutesSeconds(
                    dateFormatKeyAfterCheckPeriod);
            LOGGER.log(Level.INFO, "
                dateFormatKeyAfterCheckPeriod for day zero:
                " + dateFormatKeyAfterCheckPeriod);
        }
80 //         if (dateFormatDuration.get(
            dateFormatKeyAfterCheckPeriod) == 0.0) {
            //             dateFormatDuration.put(
                dateFormatKeyAfterCheckPeriod, run.getDuration() / 1000.0);
            //             LOGGER.log(Level.WARNING, "getDuration: " +
                run.getDuration());
            //             dayDurationAverage.put(
                dateFormatKeyAfterCheckPeriod, 1);
            //         } else {
85 //             dateFormatDuration.put(
                dateFormatKeyAfterCheckPeriod, dateFormatDuration.get(
                    dateFormatKeyAfterCheckPeriod) + run.getDuration() /
                    1000.0);
            //             dayDurationAverage.put(
                dateFormatKeyAfterCheckPeriod, dayDurationAverage.get(
                    dateFormatKeyAfterCheckPeriod) + 1);
            //         }
            dateFormatDuration.get(
                dateFormatKeyAfterCheckPeriod).add(run.
                getDuration() / 1000.0);

```

```

90     LOGGER.log(Level.WARNING, "
        dateFormatDurationListValues: " +
        dateFormatDuration);

    }

95     HashMap<String, Double> dayDurationMetric = new
        HashMap<String, Double>();

    for (Map.Entry<String, List<Double>> entry :
        dateFormatDuration.entrySet()) {
        // work with one date array metric [1.2, 2.09,
        // 5.09]
        DescriptiveStatistics descriptiveStatistics = new
            DescriptiveStatistics();
        for (double v : entry.getValue()) {
            descriptiveStatistics.addValue(v);
        }

        if (entry.getValue().size() == 0) {
            dayDurationMetric.put(entry.getKey(), 0.0);
            continue;
        }

        switch (statistics){
            case SUM:
                double sum = descriptiveStatistics.getSum
                    ();
                LOGGER.log(Level.WARNING, "sum: " + sum);
                dayDurationMetric.put(entry.getKey(), sum)
                    ;
                break;
            case RANGE:
                double range = descriptiveStatistics.
                    getMax() - descriptiveStatistics.getMin
                    ();
                //LOGGER.log(Level.WARNING, "range: " +
                //    range + "max" + descriptiveStatistics.
                //    getMax() + "min" + descriptiveStatistics
                //    .getMin());
                dayDurationMetric.put(entry.getKey(),
                    range);
                break;

```

```

120 case AVG:
    double mean = descriptiveStatistics.
        getMean();
    LOGGER.log(Level.WARNING, "mean: " + mean)
        ;
    dayDurationMetric.put(entry.getKey(), mean
        );
    break;
125 case MEDIAN:
    double median = descriptiveStatistics.
        getPercentile(50);
    LOGGER.log(Level.WARNING, "median: " +
        median);
    dayDurationMetric.put(entry.getKey(),
        median);
    break;
130 case DISPERSION:
    double dispersion = descriptiveStatistics.
        getPopulationVariance();
    LOGGER.log(Level.WARNING, "sum: " +
        dispersion);
    dayDurationMetric.put(entry.getKey(),
        dispersion);
    break;
135 case SDUNBIASED:
    double sdUnbiased = descriptiveStatistics.
        getStandardDeviation();
    LOGGER.log(Level.WARNING, "sdUnbiased: " +
        sdUnbiased);
    dayDurationMetric.put(entry.getKey(),
        sdUnbiased);
    break;
140 case SD:
    double sd = FastMath.sqrt(
        descriptiveStatistics.
        getPopulationVariance());
    LOGGER.log(Level.WARNING, "sd: " + sd);
    dayDurationMetric.put(entry.getKey(), sd);
    break;
145 case MODE:
    //prepare for mode with StatUtils methods
    double[] doublesArray = entry.getValue().
        stream().mapToDouble(d -> d).toArray();
    double[] modes = StatUtils.mode(
        doublesArray);

```

```

double mode;

150     if (modes.length == doublesArray.length) {
        mode = 0;
    } else {
        mode = modes[0];
155     }

    LOGGER.log(Level.WARNING, "mode: " + mode)
        ;
    dayDurationMetric.put(entry.getKey(), mode
        );
    break;
160 }

    }

165     LOGGER.log(Level.INFO, "dayDurationMetric: " +
        dayDurationMetric);
    return dayDurationMetric;
    }
}

```

Класс для работы с датами и инициализации структур для метрик
DateTimeHandler.

```

package io.jenkins.plugins.sample;

import hudson.model.Run;
5 import hudson.util.RunList;

import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
10 import java.time.*;
import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;

15 public class DateTimeHandler {

    static Logger LOGGER = Logger.getLogger(DateTimeHandler.
        class.getName());

    public static Date convertLongTimeToDate(long time) {

```

```

20         Date date = new Date(time);
           return date;
       }

       public static Date convertStringToDate(String dateString,
25         String format) throws ParseException {
           SimpleDateFormat formatter = new SimpleDateFormat(
               format);
           Date date = formatter.parse(dateString);
           return date;
       }

30
       /*
         * date format = yyyy MM dd HH:mm:ss
         *
35         * */
       public static long convertDateToLongTime(Date date) throws
           ParseException {
           return date.getTime();
       }

40       public static int getDayOfMonth(Date aDate) throws
           ParseException {
           Calendar cal = Calendar.getInstance();
           cal.setTime(aDate);
           return cal.get(Calendar.DAY_OF_MONTH);
       }

45       public static int getCurrentMonthDays() {
           Calendar c = Calendar.getInstance();
           return c.getActualMaximum(Calendar.DAY_OF_MONTH);
       }

50       public static int getLastMonthDays() {
           Calendar c = Calendar.getInstance();
           c.add(Calendar.MONTH, -1);
           return c.getActualMaximum(Calendar.DAY_OF_MONTH);
55       }

       // public static int getLastMonths() {
       //     Calendar c = Calendar.getInstance();
       //     c.add(Calendar.MONTH, -1);
60 //     return c.getActualMaximum(Calendar.DAY_OF_MONTH);

```



```

// }

public static String dateToString(Date date, String format
    ) {
    DateFormat dateFormat = new SimpleDateFormat(format);
65    String strDate = dateFormat.format(date);
    return strDate;
}

public static String dateMonthToString(Date date) {
70    DateFormat dateFormat = new SimpleDateFormat("yyyy-MM"
        );
    String strDate = dateFormat.format(date);
    return strDate;
}

public static String dateSetZeroMinutesSeconds(String
75    dateString) throws ParseException {

    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-
        dd HH");
    Date date = sdf.parse(dateString);
    Calendar cal = Calendar.getInstance();
80    cal.setTime(date);
    cal.set(Calendar.MINUTE, 0);
    cal.set(Calendar.SECOND, 0);
    return dateToString(cal.getTime(), "yyyy-MM-dd HH:mm:
        ss");
}

85
public static String dateSetZeroDay(String dateString)
    throws ParseException {

    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM")
        ;
    Date date = sdf.parse(dateString);
90    Calendar cal = Calendar.getInstance();
    cal.setTime(date);
    cal.set(Calendar.DAY_OF_MONTH, 1);
    return dateToString(cal.getTime(), "yyyy-MM-dd");
}

95
/**
 * Create map format {23.12: [], 24.12: [] ...}
 * on 30-31 days

```

```

100      *
      * **/
public static HashMap<String, List<Double>>
    createDateMonthMap() {
        ZonedDateTime dateTime = ZonedDateTime.now().
            minusMonths(1);
        Logger LOGGER;
        LOGGER = Logger.getLogger(DateTimeHandler.class.
            getName());
105    LOGGER.log(Level.INFO, "dateTime" + dateTime);
        HashMap<String, List<Double>> dayDuration = new
            HashMap<String, List<Double>>();
        int lenMonth = getLastMonthDays();
        LOGGER.log(Level.INFO, "lenMonth: " + lenMonth);
        for (int i = 1; i <= lenMonth; i++) {
110
            DateFormat dateFormat = new SimpleDateFormat("yyyy
                -MM-dd");
            String strDate = dateFormat.format(
                Date.from(dateTime.plusDays(i).toInstant())
                    .getTime());
            LOGGER.log(Level.INFO, "strdate i: " + i + " - " +
                strDate);
115            dayDuration.put(strDate, new ArrayList<Double>());
            //dayDuration.get(strDate).add(0.0);
        }
        LOGGER.log(Level.INFO, "dayDuration: " + dayDuration.
            entrySet());
        return dayDuration;
120    }

    /**
     * Create map format {23.12.2023: 0.0, 24.12.2023: 0.0
     * ...}
125     * on 8 equal periods
     *
     * **/
public static HashMap<String, List<Double>>
    createDateAllMap(RunList<Run> runs) {
        //ZonedDateTime dateTime = ZonedDateTime.now().
            minusMonths(1);
130        Logger LOGGER;
        LOGGER = Logger.getLogger(DateTimeHandler.class.
            getName());

```

```

    LOGGER.log(Level.INFO, "run1" + runs.getFirstBuild());
    LOGGER.log(Level.INFO, "run2" + runs.getLastBuild());
    long timeStart = runs.getFirstBuild().
        getStartTimeInMillis();
135    long timeEnd = runs.getLastBuild().
        getStartTimeInMillis();
    LocalDate startDate =
        LocalDate.ofInstant(Instant.ofEpochMilli(
            timeStart),
            TimeZone.getDefault().toZoneId());
    LocalDate endDate =
140    LocalDate.ofInstant(Instant.ofEpochMilli(
        timeEnd),
        TimeZone.getDefault().toZoneId());
    Period period = Period.between(startDate, endDate);
    LOGGER.log(Level.INFO, "period between first and last
        build: " + period);
    HashMap<String, List<Double>> dayDuration = new
        HashMap<String, List<Double>>();
145    int lenAll = 8;
    LOGGER.log(Level.INFO, "lenMonth: " + lenAll);
    //    for (int i = 1; i <= lenAll; i++) {
    //
    //        DateFormat dateFormat = new SimpleDateFormat("
150    yyyy-MM-dd");
    //        String strDate = dateFormat.format(
    //            Date.from(dateTime.plusDays(i).toInstant
    //                ()).getTime());
    //        LOGGER.log(Level.INFO, "strdate i: " + i + " - "
    //            + strDate);
    //        dayDuration.put(strDate, 0.0);
    //    }
155    LOGGER.log(Level.INFO, "dayDuration: " + dayDuration.
        entrySet());
    return dayDuration;
}

/**
160    * Create map format {1:00:00 0.0, 2:00:00: 0.0 ...}
    * on 24 hours
    *
    * **/
    public static HashMap<String, List<Double>>
        createDateDayMap() throws ParseException {

```

```

165     ZonedDateTime dateTime = ZonedDateTime.now().
        minusHours(24);
    Logger LOGGER;
    LOGGER = Logger.getLogger(DateTimeHandler.class.
        getName());
    LOGGER.log(Level.INFO, "dateTime days" + dateTime);
    HashMap<String, List<Double>> hourDuration = new
        HashMap<String, List<Double>>();
170     int lenDay = 24;
    LOGGER.log(Level.INFO, "lenDay: " + lenDay);
    for (int i = 1; i <= lenDay; i++) {

        DateFormat dateFormat = new SimpleDateFormat("yyyy
            -MM-dd HH:mm:ss");
175     String strDate = dateFormat.format(
        Date.from(dateTime.plusHours(i).toInstant
            ())).getTime();
        strDate = DateTimeHandler.
            dateSetZeroMinutesSeconds(strDate);
        LOGGER.log(Level.INFO, "strdate i: " + i + " - " +
            strDate);
        hourDuration.put(strDate, new ArrayList<Double>())
            ;
180     //hourDuration.put(strDate, 0.0);
    }
    LOGGER.log(Level.INFO, "hourDuration: " + hourDuration
        .entrySet());
    return hourDuration;
}

185
/**
 * Create map format {1:00:00 {success: 0, fail : 0},
 * 2:00:00: {success: 0, fail : 0} ...}
 * on 24 hours
 *
190 * **/
public static HashMap<String, HashMap<String,Integer>>
    createDateDayMapSuccess() throws ParseException {
    ZonedDateTime dateTime = ZonedDateTime.now().
        minusHours(24);
    Logger LOGGER;
    LOGGER = Logger.getLogger(DateTimeHandler.class.
        getName());
195     LOGGER.log(Level.INFO, "dateTime days" + dateTime);

```

```

200     HashMap<String, HashMap<String,Integer>>
        successFailSuccess = new HashMap();
        int lenDay = 24;
        LOGGER.log(Level.INFO, "lenDay: " + lenDay);
        for (int i = 1; i <= lenDay; i++) {

            DateFormat dateFormat = new SimpleDateFormat("yyyy
                -MM-dd HH:mm:ss");
            String strDate = dateFormat.format(
                Date.from(dateTime.plusHours(i).toInstant
                    ()).getTime());
            strDate = DateTimeHandler.
                dateSetZeroMinutesSeconds(strDate);
205         LOGGER.log(Level.INFO, "strdate i: " + i + " - " +
                strDate);
            successFailSuccess.put(strDate, new HashMap(){
                put("fail", 0);
                put("success", 0);
            });
210     }
        LOGGER.log(Level.INFO, "hour successFailSuccess: " +
            successFailSuccess.entrySet());
        return successFailSuccess;
    }

215     public static HashMap<String, HashMap<String,Integer>>
        createDateWeekMapSuccessRate() {
            ZonedDateTime dateTime = ZonedDateTime.now().
                minusWeeks(1);
            LOGGER.log(Level.INFO, "dateTime - 1 week" + dateTime)
                ;
            HashMap<String, HashMap<String,Integer>>
                successFailSuccess = new HashMap();
            int lenWeek = 7;
220         LOGGER.log(Level.INFO, "lenWeek" + lenWeek);
            for (int i = 1; i <= lenWeek; i++) {

                DateFormat dateFormat = new SimpleDateFormat("yyyy
                    -MM-dd");
                //get dateTime previous week + i = 1...7 day and
                getTime, after in strDate=2022-03-22
225         String strDate = dateFormat.format(
                Date.from(dateTime.plusDays(i).toInstant()
                    ).getTime());

```

```

        LOGGER.log(Level.INFO, "strdate i: " + i + " - " +
            strDate);
        successFailSuccess.put(strDate, new HashMap(){
            put("fail", 0);
230         put("success", 0);
        });
    }
    LOGGER.log(Level.INFO, "successFailSuccess: " +
        successFailSuccess.entrySet());
    return successFailSuccess;
235 }
/**
 * Create map format {23.12: {success: 0, fail : 0},
 *    24.12: {success: 0, fail : 0} ...}
 * on 30-31 days
 *
240 * **/

public static HashMap<String, HashMap<String,Integer>>
createDateMonthMapSuccessRate() {
    ZonedDateTime dateTime = ZonedDateTime.now().
        minusMonths(1);
    LOGGER.log(Level.INFO, "dateTime" + dateTime);
245     HashMap<String, HashMap<String,Integer>>
        successFailSuccess = new HashMap();
    int lenMonth = getLastMonthDays();
    LOGGER.log(Level.INFO, "lenMonth: " + lenMonth);
    for (int i = 1; i <= lenMonth; i++) {

250         DateFormat dateFormat = new SimpleDateFormat("yyyy
            -MM-dd");
        //get dateTime previous month + i = 1...31 day and
        //getTime, after in strDate=2022-03-22
        String strDate = dateFormat.format(
            Date.from(dateTime.plusDays(i).toInstant()
                ).getTime());
        LOGGER.log(Level.INFO, "strdate i: " + i + " - " +
            strDate);
255         successFailSuccess.put(strDate, new HashMap(){
            put("fail", 0);
            put("success", 0);
        });
    }
260     LOGGER.log(Level.INFO, "successFailSuccess: " +
        successFailSuccess.entrySet());

```

```

        return successFailSuccess;
    }

    /**
265     * Create map format {2/2022: {success: 0, fail : 0},
        3/2022: {success: 0, fail : 0} ...}
    * on 4 month
    *
    * **/

270 public static HashMap<String, HashMap<String,Integer>>
    createDateQuarterMapSuccessRate() {
        ZonedDateTime dateTime = ZonedDateTime.now().
            minusMonths(3);
        LOGGER.log(Level.INFO, "dateTime QuarterMapSuccess" +
            dateTime);
        HashMap<String, HashMap<String,Integer>>
            successFailSuccess = new HashMap();
        int lenQuarterSuccess = 3;
275 LOGGER.log(Level.INFO, "lenQuarterSuccess
            QuarterMapSuccess: " + lenQuarterSuccess);
        for (int i = 1; i <= lenQuarterSuccess; i++) {

            DateFormat dateFormat = new SimpleDateFormat("yyyy
                -MM");
            //get dateTime previous quarter + i = 1...4 month
            and getTime, after in strDate=2022-03
280 String strDate = dateFormat.format(
                Date.from(dateTime.plusMonths(i).toInstant
                    ()).getTime());
            LOGGER.log(Level.INFO, "strdate i: " + i + " - " +
                strDate);
            successFailSuccess.put(strDate, new HashMap(){
                put("fail", 0);
285 put("success", 0);
            });
        }
        LOGGER.log(Level.INFO, "successFailSuccess: " +
            successFailSuccess.entrySet());
        return successFailSuccess;
290 }

    /**
    * Create map format {23.12: 0, 24.12: 0 ...}
    * on 30-31 days

```

```

295      *
      * **/

public static HashMap<String, Integer>
createDateMonthMapTestCount() {
    ZonedDateTime dateTime = ZonedDateTime.now().
        minusMonths(1);
300    LOGGER.log(Level.INFO, "dateTime test count" +
        dateTime);
    HashMap<String, Integer> testCount = new HashMap();
    int lenMonth = getLastMonthDays();
    LOGGER.log(Level.INFO, "lenMonth test count: " +
        lenMonth);
    for (int i = 1; i <= lenMonth; i++) {
305
        DateFormat dateFormat = new SimpleDateFormat("yyyy
            -MM-dd");
        //get dateTime previous month + i = 1...31 day and
        //getTime, after in strDate=2022-03-22
        String strDate = dateFormat.format(
            Date.from(dateTime.plusDays(i).toInstant()
                ).getTime());
310        LOGGER.log(Level.INFO, "strdate i: " + i + " - " +
            strDate);
        testCount.put(strDate, 0);
    }
    LOGGER.log(Level.INFO, "testCount: " + testCount.
        entrySet());
    return testCount;
315 }

/**
 * Create map format {12 10:00: 0, 12 11:00: 0 ...}
 * on 24 hours
320 *
 * **/

public static HashMap<String, Integer>
createDateDayMapTestCount() throws ParseException {
    ZonedDateTime dateTime = ZonedDateTime.now().
        minusHours(24);
325    Logger LOGGER;
    LOGGER = Logger.getLogger(DateTimeHandler.class.
        getName());

```



```

    LOGGER.log(Level.INFO, "dateTime days test count" +
        dateTime);
    HashMap<String, Integer> hourDuration = new HashMap<
        String, Integer>();
    int lenDay = 24;
330    LOGGER.log(Level.INFO, "lenDay: " + lenDay);
    for (int i = 1; i <= lenDay; i++) {

        DateFormat dateFormat = new SimpleDateFormat("yyyy
            -MM-dd HH:mm:ss");
        String strDate = dateFormat.format(
335            Date.from(dateTime.plusHours(i).toInstant
                ()).getTime());
        strDate = DateTimeHandler.
            dateSetZeroMinutesSeconds(strDate);
        LOGGER.log(Level.INFO, "strdate i: " + i + " - " +
            strDate);
        hourDuration.put(strDate, 0);
    }
340    LOGGER.log(Level.INFO, "hourDuration: " + hourDuration
        .entrySet());
    return hourDuration;
}

/**
345  * Create map format {2/2022: 0, 3/2022: 0 ...}
  * on 12 month
  *
  * **/
public static HashMap<String, Integer>
    createDateYearMapTestCount() {
350    ZonedDateTime dateTime = ZonedDateTime.now().
        minusYears(1);
    LOGGER.log(Level.INFO, "last year dateTime test" +
        dateTime);
    HashMap<String, Integer> monthDuration = new HashMap<
        String, Integer>();
    int lengthYear = 12; // any year length in month
    LOGGER.log(Level.INFO, "lengthYear test: " +
        lengthYear);
355    for (int i = 1; i <= lengthYear; i++) {

        DateFormat dateFormat = new SimpleDateFormat("yyyy
            -MM");

```

```

360         //get dateTime previous year + i = 1...12 and
           getTime, after in strDate=2022-03
String strDate = dateFormat.format(
    Date.from(
        dateTime.plusMonths(i).toInstant()
    ).getTime()
);
365
    LOGGER.log(Level.INFO, "strdate i: " + i + " - " +
        strDate);
    monthDuration.put(strDate, 0);
}
    LOGGER.log(Level.INFO, "monthDuration: " +
        monthDuration.entrySet());
370    return monthDuration;
}

/**
 * Create map format {2/2022: 0, 3/2022: 0 ...}
375 * on 3 month
 *
 * **/
public static HashMap<String, Integer>
createDateQuarterMapTestCount() {
    ZonedDateTime dateTime = ZonedDateTime.now().
        minusMonths(3);
380    LOGGER.log(Level.INFO, "last quarter dateTime test" +
        dateTime);
    HashMap<String, Integer> quarterDuration = new HashMap
        <String, Integer>();
    int lengthQuarter = 3; // any year length in month
    LOGGER.log(Level.INFO, "lengthQuarter test: " +
        lengthQuarter);
    for (int i = 1; i <= lengthQuarter; i++) {
385
        DateFormat dateFormat = new SimpleDateFormat("yyyy
            -MM");

        //get dateTime previous year + i = 1...3 and
           getTime, after in strDate=2022-03
String strDate = dateFormat.format(
    Date.from(
        dateTime.plusMonths(i).toInstant()
    ).getTime()
);
390

```

```

395         LOGGER.log(Level.INFO, "strdate i test: " + i + "
            - " + strDate);
        quarterDuration.put(strDate, 0);
    }
    LOGGER.log(Level.INFO, "quarterDuration test: " +
        quarterDuration.entrySet());
    return quarterDuration;
400 }

/**
 * Create map format {1/2/2022: 0.0, 3/2/2022: 0.0 ...}
 * on 1 week
405 *
 * **/
public static HashMap<String, Integer>
    createDateWeekMapTestCount() {
    ZonedDateTime dateTime = ZonedDateTime.now().
        minusWeeks(1);
    LOGGER.log(Level.INFO, "last week dateTime test" +
        dateTime);
410    HashMap<String, Integer> weekDuration = new HashMap<
        String, Integer>();
    int lengthWeek = 7; // any week length in days
    LOGGER.log(Level.INFO, "lengthWeek test: " +
        lengthWeek);
    for (int i = 1; i <= lengthWeek; i++) {

415        DateFormat dateFormat = new SimpleDateFormat("yyyy
            -MM-dd");

        //get dateTime previous week + i = 1...7 and
        //getTime, after in strDate=2022-03-01
        String strDate = dateFormat.format(
            Date.from(
420                dateTime.plusDays(i).toInstant()
            ).getTime()
        );

        LOGGER.log(Level.INFO, "strdate i test: " + i + "
            - " + strDate);
425        weekDuration.put(strDate, 0);
    }
    LOGGER.log(Level.INFO, "weekDuration test: " +
        weekDuration.entrySet());

```

```

    return weekDuration;
}

430

/**
 * Create map format {2/2022: 0.0, 3/2022: 0.0 ...}
 * on 12 month
435
 *
 * **/
public static HashMap<String, List<Double>>
createDateYearMap() {
    ZonedDateTime dateTime = ZonedDateTime.now().
        minusYears(1);
    LOGGER.log(Level.INFO, "last year dateTime" + dateTime
440
    );
    HashMap<String, List<Double>> monthDuration = new
        HashMap<String, List<Double>>();
    int lengthYear = 12; // any year length in month
    LOGGER.log(Level.INFO, "lengthYear: " + lengthYear);
    for (int i = 1; i <= lengthYear; i++) {

445
        DateFormat dateFormat = new SimpleDateFormat("yyyy
            -MM");

        //get dateTime previous year + i = 1...12 and
        //getTime, after in strDate=2022-03
        String strDate = dateFormat.format(
450
            Date.from(
                dateTime.plusMonths(i).toInstant()
            ).getTime()
        );

        LOGGER.log(Level.INFO, "strdate i: " + i + " - " +
            strDate);
455
        monthDuration.put(strDate, new ArrayList<Double>()
            );
    }
    LOGGER.log(Level.INFO, "monthDuration: " +
        monthDuration.entrySet());
    return monthDuration;
}

460

/**
 * Create map format {2/2022: 0.0, 3/2022: 0.0 ...}
 * on 3 month

```

```

465      *
      * **/
public static HashMap<String, List<Double>>
    createDateQuarterMap() {
        ZonedDateTime dateTime = ZonedDateTime.now().
            minusMonths(3);
        LOGGER.log(Level.INFO, "last quarter dateTime" +
            dateTime);
        HashMap<String, List<Double>> quarterDuration = new
            HashMap<String, List<Double>>();
470        int lengthQuarter = 3; // any year length in month
        LOGGER.log(Level.INFO, "lengthQuarter: " +
            lengthQuarter);
        for (int i = 1; i <= lengthQuarter; i++) {

            DateFormat dateFormat = new SimpleDateFormat("yyyy
                -MM");

475            //get dateTime previous year + i = 1...3 and
                getTime, after in strDate=2022-03
            String strDate = dateFormat.format(
                Date.from(
                    dateTime.plusMonths(i).toInstant()
480                ).getTime()
            );

            LOGGER.log(Level.INFO, "strdate i: " + i + " - " +
                strDate);
            quarterDuration.put(strDate, new ArrayList<Double>
                >());
485        }
        LOGGER.log(Level.INFO, "quarterDuration: " +
            quarterDuration.entrySet());
        return quarterDuration;
    }

490    /**
        * Create map format {1/2/2022: 0.0, 3/2/2022: 0.0 ...}
        * on 1 week
        *
        * **/
495    public static HashMap<String, List<Double>>
        createDateWeekMap() {
            ZonedDateTime dateTime = ZonedDateTime.now().
                minusWeeks(1);

```

```

500     LOGGER.log(Level.INFO, "last week dateTime duration" +
        dateTime);
    HashMap<String, List<Double>> weekDuration = new
        HashMap<String, List<Double>>();
    int lengthWeek = 7; // any week length in days
    505     LOGGER.log(Level.INFO, "lengthWeek: " + lengthWeek);
    for (int i = 1; i <= lengthWeek; i++) {

        DateFormat dateFormat = new SimpleDateFormat("yyyy
            -MM-dd");

        505         //get dateTime previous week + i = 1...7 and
            getTime, after in strDate=2022-03-01
        String strDate = dateFormat.format(
            Date.from(
                dateTime.plusDays(i).toInstant()
            ).getTime()
        510    );

        LOGGER.log(Level.INFO, "strdate i: " + i + " - " +
            strDate);
        weekDuration.put(strDate, new ArrayList<Double>())
            ;
    }
    515     LOGGER.log(Level.INFO, "weekDuration: " + weekDuration
        .entrySet());
    return weekDuration;
}

public static HashMap<String, HashMap<String,Integer>>
createDateYearMapSuccessRate() {
    520     ZonedDateTime dateTime = ZonedDateTime.now().
        minusYears(1);
    LOGGER.log(Level.INFO, "last year dateTime success" +
        dateTime);
    HashMap<String, HashMap<String,Integer>>
        successFailSuccess = new HashMap();
    int lenMonth = 12;
    525     LOGGER.log(Level.INFO, "lenMonth: " + lenMonth);
    for (int i = 1; i <= lenMonth; i++) {

        DateFormat dateFormat = new SimpleDateFormat("yyyy
            -MM");
        //get dateTime previous year + i = 1...12 and
            getTime, after in strDate=2022-03

```

```

530         String strDate = dateFormat.format(
            Date.from(dateTime.plusMonths(i).toInstant
                ()).getTime());
        LOGGER.log(Level.INFO, "strdate i: " + i + " - " +
            strDate);
        successFailSuccess.put(strDate, new HashMap(){
            put("fail", 0);
            put("success", 0);
535        });
    }
    LOGGER.log(Level.INFO, "successFailSuccess: " +
        successFailSuccess.entrySet());
    return successFailSuccess;
    }
540 }

```

Код BuildLogic.java:

```

package io.jenkins.plugins.sample;

5 import hudson.model.Result;
import hudson.model.Run;
import hudson.util.RunList;
import java.text.ParseException;
import java.time.ZonedDateTime;
10 import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;

public class BuildLogic {
15     IntervalDate period;
    RunList<Run> buildList;

    Boolean failed;
    static Logger LOGGER = Logger.getLogger(BuildLogic.class.
        getName());
20

    public BuildLogic(IntervalDate period, Boolean failed,
        RunList<Run> buildList) {
        this.period = period;
        this.buildList = buildList;
25        this.failed = failed;
    }
}

```

```

public void filterPeriodBuild() {
    switch (period) {
        case MONTH:
            Date dateMonth = Date.from(ZonedDateTime.now().
                .minusMonths(1).toInstant());
            this.buildList = buildList.filter(run -> {
                try {
                    return run.getStartTimeInMillis() >=
                        DateTimeHandler.
                            convertDateToLongTime(dateMonth);
                } catch (ParseException e) {
                    throw new RuntimeException(e);
                }
            });
            break;
        case YEAR:
            Date dateYear = Date.from(ZonedDateTime.now().
                .minusYears(1).toInstant());
            this.buildList = buildList.filter(run -> {
                try {
                    return run.getStartTimeInMillis() >=
                        DateTimeHandler.
                            convertDateToLongTime(dateYear);
                } catch (ParseException e) {
                    throw new RuntimeException(e);
                }
            });
            break;
        case DAY:
            Date dateDay = Date.from(ZonedDateTime.now().
                .minusHours(24).toInstant());
            this.buildList = buildList.filter(run -> {
                try {
                    LOGGER.log(Level.WARNING, "runTime: "
                        + DateTimeHandler.
                            convertLongTimeToDate(run.
                                getStartTimeInMillis()));
                    LOGGER.log(Level.WARNING, "runTime: "
                        + run.getStartTimeInMillis() + "now
                        " + DateTimeHandler.
                            convertDateToLongTime(dateDay));
                    LOGGER.log(Level.WARNING, "bool check:
                        " + (run.getStartTimeInMillis() >=
                            DateTimeHandler.
                                convertDateToLongTime(dateDay)));
                }
            });
    }
}

```



```

        return run.getStartTimeInMillis() >=
            DateTimeHandler.
                convertDateToLongTime(dateDay);
    } catch (ParseException e) {
        throw new RuntimeException(e);
    }
});
LOGGER.log(Level.WARNING, "dat filter: " + (
    this.buildList));
break;
case WEEK:
    Date dateWeek = Date.from(ZonedDateTime.now().
        minusDays(6).toInstant());
    this.buildList = buildList.filter(run -> {
        try {
            return run.getStartTimeInMillis() >=
                DateTimeHandler.
                    convertDateToLongTime(dateWeek);
        } catch (ParseException e) {
            throw new RuntimeException(e);
        }
    });
    break;
case QUARTER:
    Date dateQuarter = Date.from(ZonedDateTime.now
        ().minusMonths(3).toInstant());
    this.buildList = buildList.filter(run -> {
        try {
            return run.getStartTimeInMillis() >=
                DateTimeHandler.
                    convertDateToLongTime(dateQuarter);
        } catch (ParseException e) {
            throw new RuntimeException(e);
        }
    });
    break;
case ALL:
    break;
}
}

public void filterFailedBuild() {
    if (!failed) {
        this.buildList = buildList.filter(run -> {

```

```

        return run.getResult().isBetterOrEqualTo(
            Result.SUCCESS);
    });
}
95  }
}

```

Код BuildArtifactSizeLogic.java:

```

package io.jenkins.plugins.sample;

5  import hudson.model.Run;
import hudson.util.RunList;
import org.apache.commons.math3.stat.StatUtils;
import org.apache.commons.math3.stat.descriptive.
    DescriptiveStatistics;
import org.apache.commons.math3.util.FastMath;
10
import java.text.ParseException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
15 import java.util.logging.Level;
import java.util.logging.Logger;

public class BuildArtifactSizeLogic extends BuildLogic {

20     static Logger LOGGER = Logger.getLogger(BuildDurationLogic
        .class.getName());
    HashMap<String, List<Double>> dateFormatArtifact;
    String dateFormatKey;
    public BuildArtifactSizeLogic(IntervalDate period, Boolean
        failed, RunList<Run> buildList) {
        super(period, failed, buildList);
25    }

    public Map<String, Double> getArtifactSize(Statistics
        statistics) throws ParseException {
        filterPeriodBuild();
        filterFailedBuild();
30        switch (this.period){
            case MONTH:
                dateFormatArtifact = DateTimeHandler.
                    createDateMonthMap();
                dateFormatKey = "yyyy-MM-dd";

```

```

35         break;
case WEEK:
    dateFormatArtifact = DateTimeHandler.
        createDateWeekMap();
    dateFormatKey = "yyyy-MM-dd";
    break;
40 case YEAR:
    dateFormatArtifact = DateTimeHandler.
        createDateYearMap();
    dateFormatKey = "yyyy-MM";
    break;
case QUARTER:
    dateFormatArtifact = DateTimeHandler.
        createDateQuarterMap();
45     dateFormatKey = "yyyy-MM";
    break;
case DAY:
    dateFormatArtifact = DateTimeHandler.
        createDateDayMap();
    dateFormatKey = "yyyy-MM-dd HH";
50     break;
case ALL:
    dateFormatArtifact = DateTimeHandler.
        createDateYearMap();
    dateFormatKey = "yyyy-MM";
    break;
55     }

//     HashMap<String, Integer> dayArtifactAverage = new
HashMap<>();
    for (Run run : this.buildList) {
60         String dateFormatKeyAfterCheckPeriod =
            DateTimeHandler.dateToString(
                DateTimeHandler.
                    convertLongTimeToDate(
                        run.getStartTimeInMillis()
                    ), dateFormatKey
65         );
        LOGGER.log(Level.INFO, "
            dateFormatKeyAfterCheckPeriod artifact: " +
            dateFormatKeyAfterCheckPeriod);
        if (this.period == IntervalDate.DAY) {

```

```

        dateFormatKeyAfterCheckPeriod =
            DateTimeHandler.dateSetZeroMinutesSeconds(
                dateFormatKeyAfterCheckPeriod);
        LOGGER.log(Level.INFO, "
            dateFormatKeyAfterCheckPeriod for day zero:
            " + dateFormatKeyAfterCheckPeriod);
70    }
    LOGGER.log(Level.WARNING, "getArtifacts: " + run.
        getArtifacts());
    List<Run.Artifact> listArtifacts = run.
        getArtifacts();
    double artifactsRunSize = 0;

75    for (Run.Artifact artifact : listArtifacts) {
        artifactsRunSize += artifact.getFileSize()
            /1024.0;
        LOGGER.log(Level.WARNING, "artifact.
            getFileSize(): " +artifact.getFileSize());
    }

80    LOGGER.log(Level.WARNING, "artifactsRunSize: " +
        artifactsRunSize);

    //        if (dateFormatArtifact.get(
        dateFormatKeyAfterCheckPeriod) == 0.0) {
    //            dateFormatArtifact.put(
        dateFormatKeyAfterCheckPeriod, artifactsRunSize);
    //            dayArtifactAverage.put(
        dateFormatKeyAfterCheckPeriod, 1);
85 //        } else {
    //            dateFormatArtifact.put(
        dateFormatKeyAfterCheckPeriod, dateFormatArtifact.get(
        dateFormatKeyAfterCheckPeriod) + artifactsRunSize);
    //            dayArtifactAverage.put(
        dateFormatKeyAfterCheckPeriod, dayArtifactAverage.get(
        dateFormatKeyAfterCheckPeriod) + 1);
    //        }

    dateFormatArtifact.get(
        dateFormatKeyAfterCheckPeriod).add(
        artifactsRunSize);
90    LOGGER.log(Level.WARNING, "dateFormatArtifact: " +
        dateFormatArtifact);
    }

```

```

95      HashMap<String, Double> dayArtifactMetric = new
        HashMap<String, Double>();

    for (Map.Entry<String, List<Double>> entry :
        dateFormatArtifact.entrySet()) {
        // work with one date array metric [1.2, 2.09,
        5,09]
        DescriptiveStatistics descriptiveStatistics = new
            DescriptiveStatistics();
        for (double v : entry.getValue()) {
100            descriptiveStatistics.addValue(v);
        }

        if (entry.getValue().size() == 0) {
            dayArtifactMetric.put(entry.getKey(), 0.0);
105            continue;
        }

        switch (statistics){
            case SUM:
110                double sum = descriptiveStatistics.getSum
                    ();
                LOGGER.log(Level.WARNING, "sum: " + sum);
                dayArtifactMetric.put(entry.getKey(), sum)
                    ;
                break;
            case AVG:
115                double mean = descriptiveStatistics.
                    getMean();
                LOGGER.log(Level.WARNING, "mean: " + mean)
                    ;
                dayArtifactMetric.put(entry.getKey(), mean
                    );
                break;
            case RANGE:
120                double range = descriptiveStatistics.
                    getMax() - descriptiveStatistics.getMin
                    ();
                LOGGER.log(Level.WARNING, "range: " +
                    range);
                dayArtifactMetric.put(entry.getKey(),
                    range);
                break;
            case MEDIAN:

```

```

125         double median = descriptiveStatistics.
            getPercentile(50);
        LOGGER.log(Level.WARNING, "median: " +
            median);
        dayArtifactMetric.put(entry.getKey(),
            median);
        break;
130     case DISPERSION:
        double dispersion = descriptiveStatistics.
            getPopulationVariance();
        LOGGER.log(Level.WARNING, "sum: " +
            dispersion);
        dayArtifactMetric.put(entry.getKey(),
            dispersion);
        break;
135     case SDUNBIASED:
        double sdUnbiased = descriptiveStatistics.
            getStandardDeviation();
        LOGGER.log(Level.WARNING, "sdUnbiased: " +
            sdUnbiased);
        dayArtifactMetric.put(entry.getKey(),
            sdUnbiased);
        break;
140     case SD:
        double sd = FastMath.sqrt(
            descriptiveStatistics.
            getPopulationVariance());
        LOGGER.log(Level.WARNING, "sd: " + sd);
        dayArtifactMetric.put(entry.getKey(), sd);
        break;
145     case MODE:
        //prepare for mode with StatUtils methods
        double[] doublesArray = entry.getValue().
            stream().mapToDouble(d -> d).toArray();
        double[] modes = StatUtils.mode(
            doublesArray);
        double mode;

150         if (modes.length == doublesArray.length) {
            mode = 0;
        } else {
            mode = modes[0];
        }
155

```

```

        LOGGER.log(Level.WARNING, "mode: " + mode)
        ;
        dayArtifactMetric.put(entry.getKey(), mode
        );
        break;
    }

160
    }

//    if (average) {
165 //        for (Map.Entry<String, Integer> entry :
        dayArtifactAverage.entrySet()) {
//            LOGGER.log(Level.INFO, "sum time duration: "
//                + dateFormatArtifact.get(entry.getKey()));
//            LOGGER.log(Level.INFO, "count runs: " +
//                entry.getValue());
//            dateFormatArtifact.put(entry.getKey(),
//                dateFormatArtifact.get(entry.getKey
//                ())/entry.getValue()
170 //            );
//        }
//    }
    LOGGER.log(Level.INFO, "dayArtifactMetric: " +
        dayArtifactMetric);
//    LOGGER.log(Level.INFO, "dayArtifactAverage: " +
        dayArtifactAverage);
175    return dayArtifactMetric;
    }
}

```

Код BuildSuccessRateLogic.java:

```

package io.jenkins.plugins.sample;

import hudson.model.Result;
5 import hudson.model.Run;
import hudson.util.RunList;
import java.text.ParseException;
import java.util.HashMap;
import java.util.Map;
10 import java.util.logging.Level;
import java.util.logging.Logger;

public class BuildSuccessRateLogic extends BuildLogic {

```

```

15 static Logger LOGGER = Logger.getLogger(
    BuildSuccessRateLogic.class.getName());
    HashMap<String, HashMap<String,Integer>>
        successRateOnFormatDate;
    String dateFormatKey;

    public BuildSuccessRateLogic(IntervalDate period, RunList<
        Run> buildList) {
20         super(period,false, buildList);
    }

    public Map<String, Double> getSuccessRate() throws
        ParseException {
25         filterPeriodBuild();
        switch (this.period){
            case MONTH:
                successRateOnFormatDate = DateTimeHandler.
                    createDateMonthMapSuccessRate();
                dateFormatKey = "yyyy-MM-dd";
30                 break;
            case WEEK:
                successRateOnFormatDate = DateTimeHandler.
                    createDateWeekMapSuccessRate();
                dateFormatKey = "yyyy-MM-dd";
                break;
35             case YEAR:
                successRateOnFormatDate = DateTimeHandler.
                    createDateYearMapSuccessRate();
                dateFormatKey = "yyyy-MM";
                break;
            case QUARTER:
40                 successRateOnFormatDate = DateTimeHandler.
                    createDateQuarterMapSuccessRate();
                dateFormatKey = "yyyy-MM";
                break;
            case DAY:
                successRateOnFormatDate = DateTimeHandler.
                    createDateDayMapSuccess();
45                 dateFormatKey = "yyyy-MM-dd HH";
                break;
            case ALL:
                successRateOnFormatDate = DateTimeHandler.
                    createDateWeekMapSuccessRate();
                dateFormatKey = "yyyy-MM";

```



```

50         break;
    }

    for (Run run : this.buildList) {
        String dateFormatKeyAfterCheckPeriod =
55             DateTimeHandler.dateToString(
                DateTimeHandler.
                    convertLongTimeToDate(
                        run.getStartTimeInMillis()
                    ), dateFormatKey
                );
60         LOGGER.log(Level.INFO, "
            dateFormatKeyAfterCheckPeriod: " +
            dateFormatKeyAfterCheckPeriod);
        if (this.period == IntervalDate.DAY) {
            dateFormatKeyAfterCheckPeriod =
                DateTimeHandler.dateSetZeroMinutesSeconds(
                    dateFormatKeyAfterCheckPeriod);
            LOGGER.log(Level.INFO, "
                dateFormatKeyAfterCheckPeriod for day zero:
                " + dateFormatKeyAfterCheckPeriod);
        }
65         if (run.getResult().isBetterOrEqualTo(Result.
            SUCCESS)){
            successRateOnFormatDate.get(
                dateFormatKeyAfterCheckPeriod).put("success
            ", (successRateOnFormatDate.get(
                dateFormatKeyAfterCheckPeriod).get("success
            ")) + 1);
        } else {
            successRateOnFormatDate.get(
                dateFormatKeyAfterCheckPeriod).put("fail",
            (successRateOnFormatDate.get(
                dateFormatKeyAfterCheckPeriod).get("fail"))
            + 1);
        }
70         LOGGER.log(Level.INFO, "successRateOnFormatDate: "
            + successRateOnFormatDate);

    }

    HashMap<String, Double> successRateMap = new HashMap<
        String, Double>();
75     for (Map.Entry<String, HashMap<String, Integer>> entry
        : successRateOnFormatDate.entrySet()) {

```

```

        LOGGER.log(Level.INFO, "succes value on date:
            " + entry.getValue());
        LOGGER.log(Level.INFO, "key success rate: " +
            entry.getKey());
        if ((entry.getValue().get("success")+entry.
            getValue().get("fail")) == 0) {
            successRateMap.put(entry.getKey(), 0.0);
        } else {
            successRateMap.put(entry.getKey(),
                Double.valueOf(entry.getValue().
                    get("success"))/(entry.getValue()
                    .get("success")+entry.
                    getValue().get("fail"))
                );
        }
    }
    85    LOGGER.log(Level.INFO, "successRateMap: " +
        successRateMap);

    return successRateMap;
    }
    90 }

```

Код BuildTestCountLogic.java:

```

package io.jenkins.plugins.sample;

import hudson.model.AbstractProject;
5 import hudson.model.Result;
import hudson.model.Run;
import hudson.tasks.test.AbstractTestResultAction;
import hudson.tasks.test.AggregatedTestResultAction;
import hudson.util.RunList;
10 import java.text.ParseException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.logging.Level;
15 import java.util.logging.Logger;

public class BuildTestCountLogic extends BuildLogic {

    20    static Logger LOGGER = Logger.getLogger(
        BuildTestCountLogic.class.getName());
    HashMap<String, Integer> testCountOnFormatDate;

```

```

String dateFormatKey;

public BuildTestCountLogic(IntervalDate period, RunList<
    Run> buildList) {
25     super(period, true, buildList);
}

public Map<String, Integer> getTestCount() throws
    ParseException {
30     filterPeriodBuild();
    filterPeriodBuild();
    switch (this.period) {
        case MONTH:
            testCountOnFormatDate = DateTimeHandler.
                createDateMonthMapTestCount();
            dateFormatKey = "yyyy-MM-dd";
35             break;
        case WEEK:
            testCountOnFormatDate = DateTimeHandler.
                createDateWeekMapTestCount();
            dateFormatKey = "yyyy-MM-dd";
            break;
40         case YEAR:
            testCountOnFormatDate = DateTimeHandler.
                createDateYearMapTestCount();
            dateFormatKey = "yyyy-MM";
            break;
        case QUARTER:
45         testCountOnFormatDate = DateTimeHandler.
            createDateQuarterMapTestCount();
            dateFormatKey = "yyyy-MM";
            break;
        case DAY:
            testCountOnFormatDate = DateTimeHandler.
                createDateDayMapTestCount();
50         dateFormatKey = "yyyy-MM-dd HH";
            break;
        case ALL:
            testCountOnFormatDate = DateTimeHandler.
                createDateDayMapTestCount();
            dateFormatKey = "yyyy-MM";
55         break;
    }

    for (Run run : this.buildList) {

```

```

        LOGGER.log(Level.FINEST, "testCountMap: " +
            getTestCountForRun(run));

        String dateFormatKeyAfterCheckPeriod =
            DateTimeHandler.dateToString(
                DateTimeHandler.
                    convertLongTimeToDate(
                        run.getStartTimeInMillis()
                    ), dateFormatKey
            );
        LOGGER.log(Level.INFO, "
            dateFormatKeyAfterCheckPeriod: " +
            dateFormatKeyAfterCheckPeriod);
        if (this.period == IntervalDate.DAY) {
            dateFormatKeyAfterCheckPeriod =
                DateTimeHandler.dateSetZeroMinutesSeconds(
                    dateFormatKeyAfterCheckPeriod);
            LOGGER.log(Level.INFO, "
                dateFormatKeyAfterCheckPeriod for day zero:
                " + dateFormatKeyAfterCheckPeriod);
        }
        if (testCountOnFormatDate.get(
            dateFormatKeyAfterCheckPeriod) == 0) {
            testCountOnFormatDate.put(
                dateFormatKeyAfterCheckPeriod,
                getTestCountForRun(run));
            LOGGER.log(Level.WARNING, "getTestCountForRun:
                " + getTestCountForRun(run));
        } else {
            testCountOnFormatDate.put(
                dateFormatKeyAfterCheckPeriod,
                testCountOnFormatDate.get(
                    dateFormatKeyAfterCheckPeriod) +
                getTestCountForRun(run));
        }
    }
    LOGGER.log(Level.INFO, "testCountOnFormatDate: " +
        testCountOnFormatDate);
    return testCountOnFormatDate;
}

public int getTestCountForRun(Run run) {
    int testCount = 0;

```

```

List<AbstractTestResultAction> testActions = run.
    getActions(AbstractTestResultAction.class);
for (AbstractTestResultAction testAction : testActions
    ) {
    LOGGER.log(Level.INFO, "testAction: " + testAction
        );
    LOGGER.log(Level.INFO, "getPassedTests: " +
        testAction.getPassedTests());
    LOGGER.log(Level.INFO, "getPassedTests: " +
        testAction.getPassedTests().size());
    LOGGER.log(Level.INFO, "getFailedTests: " +
        testAction.getFailedTests());
    testCount+=testAction.getPassedTests().size();
90
95
    }

    return testCount;

    }
100 }

```

Код BuildTimeQueueLogic.java:

```

package io.jenkins.plugins.sample;

5 import hudson.model.Run;
import hudson.util.RunList;
import org.apache.commons.math3.stat.StatUtils;
import org.apache.commons.math3.stat.descriptive.
    DescriptiveStatistics;
import org.apache.commons.math3.util.FastMath;
10
import java.text.ParseException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
15 import java.util.concurrent.TimeUnit;
import java.util.logging.Level;
import java.util.logging.Logger;

public class BuildTimeQueueLogic extends BuildLogic {
20     static Logger LOGGER = Logger.getLogger(BuildDurationLogic
        .class.getName());

    HashMap<String, List<Double>> dateFormatDuration;

```

```

String dateFormatKey;
public BuildTimeQueueLogic(IntervalDate period, RunList<
    Run> buildList) {
25     super(period, true, buildList);
}

public Map<String, Double> getTimeQueue(Statistics
    statistics) throws ParseException {
    filterPeriodBuild();
30
    switch (this.period){
        case MONTH:
            dateFormatDuration = DateTimeHandler.
                createDateMonthMap();
            dateFormatKey = "yyyy-MM-dd";
35             break;
        case WEEK:
            dateFormatDuration = DateTimeHandler.
                createDateWeekMap();
            dateFormatKey = "yyyy-MM-dd";
            break;
40         case YEAR:
            dateFormatDuration = DateTimeHandler.
                createDateYearMap();
            dateFormatKey = "yyyy-MM";
            break;
        case QUARTER:
45         dateFormatDuration = DateTimeHandler.
            createDateQuarterMap();
            dateFormatKey = "yyyy-MM";
            break;
        case DAY:
            dateFormatDuration = DateTimeHandler.
                createDateDayMap();
50         dateFormatKey = "yyyy-MM-dd HH";
            break;
        case ALL:
            dateFormatDuration = DateTimeHandler.
                createDateYearMap();
            dateFormatKey = "yyyy-MM";
55         break;
    }

```

```

//      HashMap<String, Integer> dayDurationAverage = new
HashMap<>();
60      for (Run run : this.buildList) {
          String dateFormatKeyAfterCheckPeriod =
              DateTimeHandler.dateToString(
                  DateTimeHandler.
                      convertLongTimeToDate(
70                      run.getStartTimeInMillis()
                          ), dateFormatKey
                      );
          LOGGER.log(Level.INFO, "
              dateFormatKeyAfterCheckPeriod time Queue: " +
              dateFormatKeyAfterCheckPeriod);
          if (this.period == IntervalDate.DAY) {
              dateFormatKeyAfterCheckPeriod =
                  DateTimeHandler.dateSetZeroMinutesSeconds(
                      dateFormatKeyAfterCheckPeriod);
70              LOGGER.log(Level.INFO, "
                  dateFormatKeyAfterCheckPeriod for day zero:
                  " + dateFormatKeyAfterCheckPeriod);
          }
          long runTimeInQueue = new TimeInQueueFetcher().
              getTimeInQueue(run);
          //      if (dateFormatDuration.get(
              dateFormatKeyAfterCheckPeriod) == 0.0) {
              //
75 //      LOGGER.log(Level.WARNING, "getTimeInQueue
              long: " + runTimeInQueue);
              //      dateFormatDuration.put(
                  dateFormatKeyAfterCheckPeriod, (double) runTimeInQueue);
              //      LOGGER.log(Level.WARNING, "getTimeInQueue
                  double: " + (double) runTimeInQueue);
              //      dayDurationAverage.put(
                  dateFormatKeyAfterCheckPeriod, 1);
              //      } else {
80 //      dateFormatDuration.put(
                  dateFormatKeyAfterCheckPeriod, dateFormatDuration.get(
                      dateFormatKeyAfterCheckPeriod) + (double) runTimeInQueue);
              //      dayDurationAverage.put(
                  dateFormatKeyAfterCheckPeriod, dayDurationAverage.get(
                      dateFormatKeyAfterCheckPeriod) + 1);
              //      }
              dateFormatDuration.get(
                  dateFormatKeyAfterCheckPeriod).add((double)
                  runTimeInQueue);

```

```

        LOGGER.log(Level.WARNING, "
            dateFormatDurationListValues: " +
            dateFormatDuration);
85    }
    //    if (average) {
    //        for (Map.Entry<String, Integer> entry :
    //            dayDurationAverage.entrySet()) {
    //            LOGGER.log(Level.INFO, "sum time queue: " +
    //                dateFormatDuration.get(entry.getKey()));
    //            LOGGER.log(Level.INFO, "count runs time
    //                queue: " + entry.getValue());
    //            dateFormatDuration.put(entry.getKey(),
    //                dateFormatDuration.get(entry.getKey
    //                    ())/entry.getValue()
    //            );
    //        }
    //    }
95    HashMap<String, Double> dayTimeQueueMetric = new
        HashMap<String, Double>();

    for (Map.Entry<String, List<Double>> entry :
        dateFormatDuration.entrySet()) {
        // work with one date array metric [1.2, 2.09,
        //    5,09]
        DescriptiveStatistics descriptiveStatistics = new
            DescriptiveStatistics();
    100    for (double v : entry.getValue()) {
            descriptiveStatistics.addValue(v);
        }

        if (entry.getValue().size() == 0) {
    105            dayTimeQueueMetric.put(entry.getKey(), 0.0);
            continue;
        }

        switch (statistics){
    110            case SUM:
                double sum = descriptiveStatistics.getSum
                    ();
                LOGGER.log(Level.WARNING, "sum: " + sum);
                dayTimeQueueMetric.put(entry.getKey(), sum
                    );
                break;
    115            case AVG:

```



```

double mean = descriptiveStatistics.
    getMean();
LOGGER.log(Level.WARNING, "mean: " + mean)
    ;
dayTimeQueueMetric.put(entry.getKey(),
    mean);
break;
120 case RANGE:
    double range = descriptiveStatistics.
        getMax() - descriptiveStatistics.getMin
            ();
    LOGGER.log(Level.WARNING, "range: " +
        range);
    dayTimeQueueMetric.put(entry.getKey(),
        range);
    break;
125 case MEDIAN:
    double median = descriptiveStatistics.
        getPercentile(50);
    LOGGER.log(Level.WARNING, "median: " +
        median);
    dayTimeQueueMetric.put(entry.getKey(),
        median);
    break;
130 case DISPERSION:
    double dispersion = descriptiveStatistics.
        getPopulationVariance();
    LOGGER.log(Level.WARNING, "sum: " +
        dispersion);
    dayTimeQueueMetric.put(entry.getKey(),
        dispersion);
    break;
135 case SDUNBIASED:
    double sdUnbiased = descriptiveStatistics.
        getStandardDeviation();
    LOGGER.log(Level.WARNING, "sdUnbiased: " +
        sdUnbiased);
    dayTimeQueueMetric.put(entry.getKey(),
        sdUnbiased);
    break;
140 case SD:
    double sd = FastMath.sqrt(
        descriptiveStatistics.
            getPopulationVariance());
    LOGGER.log(Level.WARNING, "sd: " + sd);

```

```

        dayTimeQueueMetric.put(entry.getKey(), sd)
        ;
        break;
145     case MODE:
        //prepare for mode with StatUtils methods
        double[] doublesArray = entry.getValue().
            stream().mapToDouble(d -> d).toArray();
        double[] modes = StatUtils.mode(
            doublesArray);
        double mode;

150         if (modes.length == doublesArray.length) {
            mode = 0;
        } else {
            mode = modes[0];
155         }

        LOGGER.log(Level.WARNING, "mode: " + mode)
        ;
        dayTimeQueueMetric.put(entry.getKey(),
            mode);
        break;
160     }

    }
    LOGGER.log(Level.INFO, "dayTimeQueueMetric time queue:
        " + dayTimeQueueMetric);
165    //LOGGER.log(Level.INFO, "dayDurationAverage time
        queue: " + dayDurationAverage);
    return dayTimeQueueMetric;
    }
}

```

Код TimeInQueueFetcher.java:

```

package io.jenkins.plugins.sample;

5 import hudson.model.Run;

import java.util.concurrent.TimeUnit;

public class TimeInQueueFetcher {
10     public long getTimeInQueue(Run build) {

```

```

        long queuedTime = build.getStartTimeInMillis() - build
            .getTimeInMillis();
        return TimeUnit.MILLISECONDS.toMillis(queuedTime);
    }
}

```

Код `dateIntervalEnum.java`:

```

package io.jenkins.plugins.sample;

enum IntervalDate {
5     DAY,
    WEEK,
    MONTH,
    YEAR,
    QUARTER,
10    ALL
}

```

Код `Statistics.java`:

```

package io.jenkins.plugins.sample;

enum Statistics {
5     SUM,
    AVG,
    MEDIAN,
    RANGE,
    DISPERSION,
10    SD,
    SDUNBIASED,
    MODE
}

```

Код `LinearRegressionHandler.java`

```

package io.jenkins.plugins.sample;

import org.apache.commons.math3.fitting.WeightedObservedPoints
;
5 import org.apache.commons.math3.stat.regression.
    OLSMultipleLinearRegression;
import org.apache.commons.math3.stat.regression.
    SimpleRegression;

import java.util.Arrays;
import java.util.logging.Level;
10 import java.util.logging.Logger;

```

```

public class LinearRegressionHandler {

    static Logger LOGGER = Logger.getLogger(
        LinearRegressionHandler.class.getName());

15    static double linearRegression(double[] yUnweighted,
        double[] weights) {

        double[] y = new double[yUnweighted.length];
20        double[][] x = new double[yUnweighted.length][2];

        for (int i = 0; i < y.length; i++) {
            LOGGER.log(Level.INFO, "yUnweighted i: " +
                yUnweighted[i]);
            LOGGER.log(Level.INFO, "weights i: " + weights[i])
                ;
25            y[i] = Math.sqrt(weights[i]) * yUnweighted[i];
            x[i][0] = Math.sqrt(weights[i]) * i + 1;
            x[i][1] = Math.sqrt(weights[i]);
            LOGGER.log(Level.INFO, "y[i]: " + y[i]);
            LOGGER.log(Level.INFO, "x[i][0]: " + x[i][0]);
30        }

        OLSMultipleLinearRegression regression = new
            OLSMultipleLinearRegression();
        regression.setNoIntercept(true);
        regression.newSampleData(y, x);
35

        double[] regressionParameters = regression.
            estimateRegressionParameters();
        double slope = regressionParameters[0];
        double intercept = regressionParameters[1];

40        double predictedValue = slope * (yUnweighted.length +
            1) + intercept;
        LOGGER.log(Level.INFO, "y = " + slope + "*x + " +
            intercept);
        LOGGER.log(Level.INFO, "predicted = " + predictedValue
            );
        return predictedValue;
45    }

```

```

public static double[] calculateWeightMetric(double[]
arrMetricValues) {
    LOGGER.log(Level.INFO, "arrMetricValues: " + Arrays.
        toString(arrMetricValues));
    double averageMetric = Arrays.stream(arrMetricValues).
        average().orElse(Double.NaN);
    LOGGER.log(Level.INFO, "averageMetric: " +
        averageMetric);
50    double initialWeight = 1.0;
    double stepWeight = initialWeight/(arrMetricValues.
        length *2);
    double[] arrWeights = new double[arrMetricValues.
        length];
    LOGGER.log(Level.INFO, "stepWeight: " + stepWeight);
    for (int i = arrMetricValues.length - 1; i >= 0; i--)
    {
55        if (arrMetricValues[i] < 0.5 * averageMetric) {
            arrWeights[i] = 0;
        } else {
            arrWeights[i] = initialWeight;
        }
60
        initialWeight -=stepWeight;
    }
    return arrWeights;
65 }
}

```

Код js файла с парсингом данных и визуализацией:

```

5
var ctx = document.getElementById("successRateChart").
    getContext("2d");
var ctxBuild = document.getElementById("buildDurationChart").
    getContext("2d");
10 var ctxArtifactsSize = document.getElementById("artifactsSize
    ").getContext("2d");
var ctxTimeSpentQueue = document.getElementById("
    timeSpentQueue").getContext("2d");

```

[illegible]

```

(255,
99,
132)
',
font:
{
size
:
16,
weight
:
,
bold
,
}
},
}
}
};

break;
case 'Line':
65 data = {
labels: labels,
datasets: [{
label: title,
data: dictValues,
70 borderColor: [
'rgba(0, 180, 33, 1)',
],
tension: 0.1

75 }]
};

allPerf = {

```

80

```
type: 'line',  
data: data,  
options: {  
  plugins: {
```

85


```

90
95
95      }
      };
100 break;
      case 'Radar':
105         data = {
            labels: labels,
            datasets: [{
110                label: title,
                data: dictValues,
                borderColor: [
                    'rgba(0, 180, 33, 1)',
                ],
                tension: 0.1
            }]
115         };
        allPerf = {
            type: 'radar',
            data: data,
            options: {
                scale: {
                    min:
                        0
                },
                plugins: {

```

125

130

135

},

};

140

break;

}

145 return allPerf;

}

150

function formatLabelsDate(arrLabels, dateFormat, period) {

switch(period) {

case 'DAY':

arrLabels.push(

dateFormat.getDate()+

" "+(dateFormat.

getHours()+":0"+(

dateFormat.

getMinutes())

155

);

break;

case 'WEEK':

case 'MONTH':

160

arrLabels.push(

dateFormat.getDate()+

"/"+(dateFormat.

getMonth()+1)+

"/"+dateFormat.

getFullYear()

);

165

break;

```

        case 'QUARTER':
        case 'YEAR':
            arrLabels.push(
170 (dateFormat.getMonth()+1)+"/"+dateFormat.getFullYear()
                );

        break;
175 }
    console.log("arrLabels", arrLabels);
}

180
function sortOnKeys(dict, period) {

    var sorted = [];
    for(var key in dict) {
185         console.log("key", key);
        sorted[sorted.length] = key;
        console.log("sorted", sorted);
    }
    sorted.sort();
190 console.log("sorted2", sorted);
    var dataBuildDurationValues = [];
    var labelsB = [];
    for(var i = 0; i < sorted.length; i++) {
        dataBuildDurationValues.push(dict[sorted[i]]);
195 console.log("dataBuildDurationValues",
            dataBuildDurationValues, dict[sorted[i]]);
        var dateFormat= new Date(parseInt(sorted[i]));
        console.log("dateFormat", dateFormat);
        //var period = document.querySelector(".period").
            textContent;
        console.log("period", period)
200 formatLabelsDate(labelsB, dateFormat, period);
        console.log("labelsB", labelsB);
    }
    return [dataBuildDurationValues, labelsB];
}

205
// success rate chart settings

```

```

var successRateSelect = document.querySelector("#selectSuccess
    ");
210 function createSuccessRateChart(){
    var period = document.getElementById("selectSuccess").value;
    var typeChart = document.getElementById("selectChartSR").value
        ;
    console.log("period", period);
    console.log("typeChart", typeChart)
215 var successRate2 = document.querySelector("#successRateData").
        textContent;
    console.log(successRate2);
    var obj = JSON.parse(successRate2);
    console.log("json", obj);

220     var dataSuccessRateValues = [];
    var dataSuccessRateDict = {};
    console.log(obj.count);
    for (var key in obj){
225         console.log("111", key);

        dataSuccessRateDict[Date.parse(key)] = parseFloat(obj[key]);
    }
    console.log("dataSuccessRateDict: ", dataSuccessRateDict);
230     dictSuccess = sortOnKeys(dataSuccessRateDict, period)[0];
    labelsSuccess = sortOnKeys(dataSuccessRateDict, period)[1];

    let allPerf = typeChartHandler(typeChart, labelsSuccess, '
        Success rate', dictSuccess);
235

    if (perfChartJsCharts["successRateChart"]) perfChartJsCharts
        ["successRateChart"].destroy();
    perfChartJsCharts["successRateChart"] = new Chart(ctx,
        allPerf);
240 }

// test count chart settings
245

```

```

var testCountSelect = document.querySelector("#selectTestCount
    ");

function createTestCountChart(){
var period = document.getElementById("selectTestCount").value;
250 var typeChart = document.getElementById("selectChartTC").value
    ;
    console.log("period", period)
var testCount2 = document.querySelector("#testCountData").
    textContent;
    console.log(testCount2);
    var obj = JSON.parse(testCount2);
255 console.log("json", obj);

    var dataTestCountValues = [];
    var dataTestCountDict = {};
    console.log(obj.count);
260 for (var key in obj){
    console.log("111", key);

dataTestCountDict[Date.parse(key)] = parseFloat(obj[key]);
265 }
    console.log("dataTestCountDict: ", dataTestCountDict);

    dictTestCount = sortOnKeys(dataTestCountDict, period)[0];
    labelsTestCount = sortOnKeys(dataTestCountDict, period)[1];
270
let settingsTestCount = typeChartHandler(typeChart,
    labelsTestCount, 'Test Count', dictTestCount);

    if (perfChartJsCharts["testCountChart"]) perfChartJsCharts["
        testCountChart"].destroy();
    perfChartJsCharts["testCountChart"] = new Chart(ctxTestCount
        , settingsTestCount);
275 }

// build duration chart settings
280
var buildDurationSelect = document.querySelector("#
    selectBuildDuration");

function createBuildDurationChart(){

```

```

var period = document.getElementById("selectBuildDuration").
    value;
285 var typeChart = document.getElementById("selectChartBD").value
    ;
    console.log("period", period)
var buildDuration = document.querySelector("#buildDurationData
    ").textContent;
    console.log(buildDuration);
    var obj = JSON.parse(buildDuration);
290 console.log("json", obj);

    var dataBuildDurationValues = [];
    var dataBuildDurationDict = {};
    console.log(obj.count);
295 for (var key in obj){
    console.log("111", key);
    console.log("Date.parse(key)", Date.parse(key));

300 dataBuildDurationDict[Date.parse(key)] = parseFloat(obj[key]);
    }
    console.log("dataBuildDurationDict: ", dataBuildDurationDict
        );

    dictBuildDuration = sortOnKeys(dataBuildDurationDict, period
        )[0];
305 labelsBuildDuration = sortOnKeys(dataBuildDurationDict,
    period)[1];

    let settingsBuildDuration = typeChartHandler(typeChart,
        labelsBuildDuration, 'Build duration', dictBuildDuration
    );
    if (perfChartJsCharts["buildDurationChart"])
        perfChartJsCharts["buildDurationChart"].destroy();
310 perfChartJsCharts["buildDurationChart"] = new Chart(ctxBuild
    , settingsBuildDuration);

}

315 // artifact size chart settings

```

```

var artifactSizeSelect = document.querySelector("#
    selectArtifactSize");

320 function createArtifactSizeChart(){
    var period = document.getElementById("selectArtifactsSize").
        value;
    var typeChart = document.getElementById("selectChartAS").value
        ;
    console.log("period", period)
    var artifactSize = document.querySelector("#artifactSizeData")
        .textContent;
325    console.log(artifactSize);
    var obj = JSON.parse(artifactSize);
    console.log("json", obj);

    var dataArtifactSizeValues = [];
330    var dataArtifactSizeDict = {};
    console.log(obj.count);
    for (var key in obj){
        console.log("111", key);
        console.log("Date.parse(key)", Date.parse(key));
335

    dataArtifactSizeDict[Date.parse(key)] = parseFloat(obj[key]);
    }
    console.log("dataArtifactSizeDict: ", dataArtifactSizeDict);
340

    dictArtifactSize = sortOnKeys(dataArtifactSizeDict, period)
        [0];
    labelsArtifactSize = sortOnKeys(dataArtifactSizeDict, period
        )[1];

    let settingsArtifactSize = typeChartHandler(typeChart,
        labelsArtifactSize, 'Artifact Size', dictArtifactSize);
345    if (perfChartJsCharts["artifactSizeChart"])
        perfChartJsCharts["artifactSizeChart"].destroy();
    perfChartJsCharts["artifactSizeChart"] = new Chart(
        ctxArtifactsSize, settingsArtifactSize);

    }

350 // time queue chart settings

var timeQueueSelect = document.querySelector("#selectTimeQueue
    ");

```



```

function createTimeQueueChart(){
355 var period = document.getElementById("selectTimeQueue").value;
    var typeChart = document.getElementById("selectChartTQ").value
        ;
    console.log("period", period)
    var timeQueue = document.querySelector("#timeQueueData").
        textContent;
    console.log(timeQueue);
360 var obj = JSON.parse(timeQueue);
    console.log("json", obj);

    var dataTimeQueueValues = [];
    var dataTimeQueueDict = {};
365 console.log(obj.count);
    for (var key in obj){
        console.log("111", key);
        console.log("Date.parse(key)", Date.parse(key));

370
        dataTimeQueueDict[Date.parse(key)] = parseFloat(obj[key]);
    }
    console.log("dataTimeQueueDict: ", dataTimeQueueDict);

375 dictTimeQueue = sortOnKeys(dataTimeQueueDict, period)[0];
    labelsTimeQueue = sortOnKeys(dataTimeQueueDict, period)[1];

    let settingsTimeQueue = typeChartHandler(typeChart,
        labelsTimeQueue, 'Time Spent In Queue', dictTimeQueue);
    if (perfChartJsCharts["timeQueueChart"]) perfChartJsCharts["
        timeQueueChart"].destroy();
380 perfChartJsCharts["timeQueueChart"] = new Chart(
        ctxTimeSpentQueue, settingsTimeQueue);
}

```

Код jelly файла, со взаимодействием Java, JS и интерфейса

```

<?jelly escape-by-default='true'?>
<j:jelly xmlns:j="jelly:core" xmlns:l="/lib/layout" xmlns:st="
    jelly:stapler" xmlns:f="/lib/form">
    <head>
5        <style>
            label {
                max-width:200px;
            }

```

```

10      .buildDuration, .period,
      .successRate, .timeQueue,
      .artifactSize, .testCount, #
        successRateData, #buildDurationData
      ,
      #testCountData, #artifactSizeData, #
        timeQueueData
    {
        display:none;
15    }
    .graph-container {
width: 95%;

    }
20    .graph-block{
        padding: 5px;
        border: 1px solid grey;
        margin: 10px;
        display: flex;

25    }

    .canvas-container{
width: 80%;

30    }

    .settings{
padding: 5px;
width:210px;
margin: 10px;
display: flex;
35    flex-direction: column;

    }

    form label{
40    margin: 5px;
    }

    </style>
</head>
45

<l:layout title="Build Configuration Statistics">

    <l:side-panel>

```

```

50         <st:include page="sidepanel.jelly" it
           ="${it.job}" optional="true" />
</l:side-panel>
<l:main-panel>

        <h1>Statistics for job ${it.job.name
          }</h1>

55        <div id="successRateData"></div>
        <div id="buildDurationData"></div>
        <div id="artifactSizeData"></div>
        <div id="testCountData"></div>
60        <div id="timeQueueData"></div>

        <script id="script1">

                var myObjectBuild = <st:bind
                  value="${it}"/>

65        myObjectBuild.
            getBuildSuccessRate('MONTH
              ', function(t) {
                document.getElementById('
                  successRateData').innerHTML
                  = t.responseObject();
                createSuccessRateChart();
              });
70        function myCheckSuccess(){
            var strPeriod = document.
              getElementById("
                selectSuccess").value;
            var typeChart = document.
              getElementById("
                selectChartAS").value;
            console.log(strPeriod, "Day");
            console.log(typeChart, "
              typeChart");
75        myObjectBuild.
            getBuildSuccessRate(
              strPeriod, function(t) {
                document.getElementById('
                  successRateData').innerHTML
                  = t.responseObject();
                createSuccessRateChart();
              });

```

80

}

85

```

myObjectBuild.getBuildDuration
    ('MONTH', '0', 'SUM',
    function(t) {
document.getElementById('
    buildDurationData').
    innerHTML = t.
    responseObject();
createBuildDurationChart();
});

```

90

```

function myCheckBuildDuration
(){
var strPeriod = document.
    getElementById("
    selectBuildDuration").value
;
var strStatistic = document.
    getElementById("
    selectStatisticBD").value;
var checkFailed = document.
    getElementById('
    checkboxFailedBuildDuration
').checked ? '1' : '0';

```

95

```

console.log(strPeriod, "
    strPeriod buildDuration");
console.log(strStatistic, "
    strStatistic buildDuration
");
myObjectBuild.getBuildDuration
    (strPeriod, checkFailed,
    strStatistic, function(t) {
document.getElementById('
    buildDurationData').
    innerHTML = t.
    responseObject();
createBuildDurationChart();
});

```

100

}

105

110

115

120

```

myObjectBuild.
    getBuildArtifactSize('MONTH
    ', '0', 'SUM', function(t)
    {
document.getElementById('
    artifactSizeData').
        innerHTML = t.
        responseObject();
createArtifactSizeChart();
});
function myCheckArtifactsSize
    (){
var strPeriod = document.
    getElementById("
        selectArtifactsSize").value
    ;
var strStatistic = document.
    getElementById("
        selectStatisticAS").value;
var checkFailed = document.
    getElementById('
        checkboxFailedArtifactsSize
        ').checked ? '1' : '0';

console.log(strPeriod, "
    strPeriod ArtifactSize");
console.log(strStatistic, "
    strStatistic ArtifactSize")
    ;
myObjectBuild.
    getBuildArtifactSize(
        strPeriod, checkFailed,
        strStatistic, function(t) {
document.getElementById('
    artifactSizeData').
        innerHTML = t.
        responseObject();
createArtifactSizeChart();
});
}

```

125

130

135

140

```

myObjectBuild.
    getBuildTimeQueue('MONTH',
        'SUM', function(t) {
document.getElementById('
    timeQueueData').innerHTML =
        t.responseObject();
createTimeQueueChart();
});
function myCheckTimeQueue(){
var strPeriod = document.
    getElementById("
        selectTimeQueue").value;
var strStatistic = document.
    getElementById("
        selectStatisticTQ").value;

console.log(strPeriod, "
    strPeriod TimeQueue");
console.log(strStatistic, "
    strStatistic TimeQueue");
myObjectBuild.
    getBuildTimeQueue(strPeriod
        , strStatistic, function(t)
        {
document.getElementById('
    timeQueueData').innerHTML =
        t.responseObject();
createTimeQueueChart();
});
}

myObjectBuild.
    getBuildTestCount('MONTH',
        '0', function(t) {
document.getElementById('
    testCountData').innerHTML =
        t.responseObject();
createTestCountChart();
});
function myCheckTestCount(){
var strPeriod = document.
    getElementById("
        selectTestCount").value;

```

145

```

var checkFailed = document.
    getElementById('
    checkboxFailedTestCount').
    checked ? '1' : '0';

console.log(strPeriod, "
    strPeriod TestCount");
myObjectBuild.
    getBuildTestCount(strPeriod
    , checkFailed, function(t)
    {
document.getElementById('
    testCountData').innerHTML =
        t.responseObject();
createTestCountChart();
    });
}

```

150

155

```

</script>

```

160

```

<div class="graph-container">
<div class="graph-block">
    <div class="canvas-container">
    <canvas id="successRateChart"
        width="90" height="25"></
        canvas>
    </div>
    <form class="settings">
        <label>

```

165

```

Type chart:
<select id="
    selectChartSR
    " onchange
    ="
    createSuccessRat
    () ">
    <
        option
        value
        ="

```

170

175

```
Bar
">
Bar
</
option
>
<
option

value
="
Line
">
Line

trend
</
option
>
<
option

value
="
Radar
">
Radar
</
option
>
</select>

</label>
<label>
Range:
<select id="
selectSuccess
" onchange
="
myCheckSuccess
()" ">
<
option

value
```


180

```
    ="
    MONTH
">
    Month
</
option
>
<
    option

    value
    ="
    DAY
">
    Day
</
option
>
<
    option

    value
    ="
    YEAR
">
    Year
</
option
>
<
    option

    value
    ="
    WEEK
">
    Week
</
option
>
<
    option

    value
    ="
```

		<div>QUARTER</div> <div>"></div> <div>Quarter</div> <div></</div> <div>option</div> <div>></div> <div><</div> <div>option</div> <div>value</div> <div>=</div> <div>ALL</div> <div>"></div> <div>All</div> <div></</div> <div>option</div> <div>></div> <div></select></div>
185		</label>
		</form>
		</div>
190		<div><div class="graph-block"></div> <div><div class="canvas-container"></div> <div><canvas id="buildDurationChart</div> <div>" width="90" height="25"></</div> <div>canvas></div> <div></div></div> <div><form class="settings"></div> <div><label></div>
195		<div>Type chart:</div> <div><select id="</div> <div>selectChartBD</div> <div>" onchange</div> <div>=</div> <div>createBuildDurat</div> <div>() ()"></div> <div><</div> <div>option</div> <div>value</div> <div>=</div> <div>Bar</div>

200

205

```
">
Bar
</
option
>
<
option

value
="
Line
">
Line

trend
</
option
>
<
option

value
="
Radar
">
Radar
</
option
>
</select>

</label>
<label>
Range:
<select id="
selectBuildDurat
" onchange
="
myCheckBuildDura
()" ">
<
option

value
="
```

210

```
MONTH
">
Month
</
option
>
<
option

value
="
DAY
">
Day
</
option
>
<
option

value
="
YEAR
">
Year
</
option
>
<
option

value
="
WEEK
">
Week
</
option
>
<
option

value
="
QUARTER
```

215

220

```
">
Quarter
</
option
>
<
option

value
="
ALL
">
All
</
option
>
</select>

</label>
<label>

Statistic:
<select id="
selectStatisticB
" onchange
="
myCheckBuildDura
()" ">
<
option

value
="
SUM
">
Sum
</
option
>
<
option

value
```

225

```
="
AVG
">
Average
</
option
>
<
option

value
="
MEDIAN
">
Median
</
option
>
<
option

value
="
RANGE
">
Range
</
option
>
<
option

value
="
DISPERSI
">
Dispersi
</
option
>
<
option

value
="
```

230

235

```
SD
">
Standard

Deviation
</
option
>
<
option

value
="
SDUNBIAS
">
Standard

Deviation

Unbiased
</
option
>
<
option

value
="
MODE
">
Mode
</
option
>
</select>

</label>
<label>
Show failed:
<input type="
checkbox"
id="
checkboxFailedBu
" onchange
="
```

240

245

250

```
myCheckBuildDura
() "/>

</label>

</form>
</div>
<div class="graph-block">
  <div class="canvas-container">
    <canvas id="timeSpentQueue"
      width="90" height="25"></
      canvas>
    </div>
    <form class="settings">
      <label>

Type chart:
<select id="
  selectChartTQ
  " onchange
  ="
  createTimeQueueC
  () ">
    <
      option
        value
        ="
        Bar
        ">
        Bar
      </
      option
    >
    <
      option
        value
        ="
        Line
        ">
        Line
      trend
    </
```


255

```
option
>
<
option

value
="
Radar
">
Radar
</
option
>

</select>

</label>
<label>
Range:
<select id="
selectTimeQueue
" onchange
="
myCheckTimeQueue
()" ">
<
option

value
="
MONTH
">
Month
</
option
>
<
option

value
="
DAY
">
Day
</
```

260

```
option
>
<
option

value
="
YEAR
">
Year
</
option
>
<
option

value
="
WEEK
">
Week
</
option
>
<
option

value
="
QUARTER
">
Quarter
</
option
>
<
option

value
="
ALL
">
All
</
```

265

270

```
option
>
</select>

</label>
<label>

Statistic:
<select id="
    selectStatisticT
    " onchange
    ="
    myCheckTimeQueue
    () ">
    <
        option

        value
        ="
        SUM
        ">
        Sum
    </
    option
    >
    <
        option

        value
        ="
        AVG
        ">
        Average
    </
    option
    >
    <
        option

        value
        ="
        MEDIAN
        ">
        Median
    </
```

275

```
option
>
<
option

value
="
RANGE
">
Range
</
option
>
<
option

value
="
DISPERSI
">
Dispersi
</
option
>
<
option

value
="
SD
">
Standard

Deviation
</
option
>
<
option

value
="
SDUNBIAS
">
Standard
```

		Deviation
		Unbiased
		</
		option
		>
	<	option
		value
		=
		MODE
		">
		Mode
		</
		option
		>
	</select>	
280	</label>	
	</form>	
	</div>	
285	<div class="graph-block">	
	<div class="canvas-container">	
	<canvas id="testCount" width	
	= "90" height="25"></canvas>	
	</div>	
	<form class="settings">	
290	<label>	
		Type chart:
		<select id="
		selectChartTC
		" onchange
		=
		createTestCountC
		() ">
295	<	option
		value
		=

300

```
Bar
">
Bar
</
option
>
<
option

value
="
Line
">
Line

trend
</
option
>
<
option

value
="
Radar
">
Radar
</
option
>
</select>

</label>
<label>
Range:
<select id="
selectTestCount
" onchange
="
myCheckTestCount
()" ">
<
option

value
```

305

```
    ="
    MONTH
    ">
    Month
  </
  option
>
<
  option

  value
  ="
  DAY
  ">
  Day
  </
  option
>
<
  option

  value
  ="
  YEAR
  ">
  Year
  </
  option
>
<
  option

  value
  ="
  WEEK
  ">
  Week
  </
  option
>
<
  option

  value
  ="
```

```

QUARTER
">
Quarter
</
option
>
<
option

value
="
ALL
">
All
</
option
>
310                                     </select>

</label>
<label>
315                                     Show failed:
                                     <input type="
                                     checkbox"
                                     id="
                                     checkboxFailedTe
                                     " onchange
                                     ="
                                     myCheckTestCount
                                     () "/>
</label>

</form>
</div>
320 <div class="graph-block">
    <div class="canvas-container">
    <canvas id="artifactsSize"
        width="90" height="25"></
        canvas>
    </div>
    <form class="settings">
325         <label>

```

Type chart:

330

335

```
<select id="
  selectChartAS
  " onchange
  ="
  createArtifactSi
  () ">
    <
      option
        value
        ="
        Bar
        ">
        Bar
      </
      option
    >
    <
      option
        value
        ="
        Line
        ">
        Line
        trend
      </
      option
    >
    <
      option
        value
        ="
        Radar
        ">
        Radar
      </
      option
    >
  </select>
</label>
<label>
```

```
Range :
<select id="
    selectArtifactsS
    " onchange
    ="
    myCheckArtifacts
    () ">
        <
            option
                value
                ="
                MONTH
                ">
                Month
            </
            option
        >
        <
            option
                value
                ="
                DAY
                ">
                Day
            </
            option
        >
        <
            option
                value
                ="
                YEAR
                ">
                Year
            </
            option
        >
        <
            option
                value
                ="
```

345

350

```

    WEEK
    ">
    Week
  </
option
>
  <
    option

    value
    ="
    QUARTER
    ">
    Quarter
  </
option
>
  <
    option

    value
    ="
    ALL
    ">
    All
  </
option
>
</select>

</label>
<label>

Statistic:
<select id="
  selectStatisticA
  " onchange
  ="
  myCheckArtifacts
  () ">
    <
      option

      value
      ="
```

355

```
SUM
">
Sum
</
option
>
<
option

value
="
AVG
">
Average
</
option
>
<
option

value
="
MEDIAN
">
Median
</
option
>
<
option

value
="
RANGE
">
Range
</
option
>
<
option

value
="
DISPERSI
```

```
">
Dispersi
</
option
>
<
option

value
="
SD
">
Standard

Deviation
</
option
>
<
option

value
="
SDUNBIAS
">
Standard

Deviation

Unbiased
</
option
>
<
option

value
="
MODE
">
Mode
</
option
>
</select>
```

```

360                                     </label>
                                     <label>
                                         Show failed:
                                         <input type="
                                             checkbox"
                                             id="
                                             checkboxFailedAr
                                             " onchange
                                             ="
                                             myCheckArtifacts
                                             () ">
365                                     </label>
                                     </form>
                                     </div>
                                     </div>
370
                                     </l:main-panel>
</l:layout>
<st:adjunct includes="io.jenkins.plugins.sample.
    BuildConfigurationStatisticsAction.
    declareChartJsClickArray"/>
<st:adjunct includes="io.jenkins.plugins.sample.
    BuildConfigurationStatisticsAction.chartLogicBox"/>
375 </j:jelly>

```

Программный код тестов на языке Java

Код юнит тестов.

```
package io.jenkins.plugins.sample;

import hudson.model.*;
5 import hudson.tasks.Shell;
import hudson.util.RunList;
import org.junit.Rule;
import org.junit.Test;
import org.jvnet.hudson.test.JenkinsRule;
10
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.*;
15
public class BuildConfigurationStatisticsBuilderTest {

    @Rule
    public JenkinsRule jenkins = new JenkinsRule();
20

    @Test
    public void testWorkingSystem() {
        assert 1 == 1;
    }
25

    @Test
    public void testSuccessBuildFromCustomBuild() throws
        Exception {
        FreeStyleProject project = jenkins.
            createFreeStyleProject();
        project.getBuildersList().add(new
            BuildConfigurationStatisticsBuilder());
30        jenkins.buildAndAssertSuccess(project);
    }

    @Test
    public void testFailBuildFromCustomBuild() throws
        Exception {
35        FreeStyleProject project = jenkins.
            createFreeStyleProject();
```

```

        project.getBuildersList().add(new Shell("echo1 hello")
        );
        jenkins.buildAndAssertStatus(Result.FAILURE, project);
    }

40  @Test
    public void testConvertLongTimeToDate() throws
        ParseException {
        DateFormat dateFormat = new SimpleDateFormat("dd/MM/
            yyyy");
        Date date = dateFormat.parse("23/09/2007");
        long time = date.getTime();
45  long resultDate = DateTimeHandler.
            convertDateToLongTime(date);
        assert resultDate == time;
    }

    @Test
50  public void testConvertDateToLongTime() throws
        ParseException {
        DateFormat dateFormat = new SimpleDateFormat("dd/MM/
            yyyy");
        Date date = dateFormat.parse("23/09/2007");
        long time = date.getTime();
        Date resultDate = DateTimeHandler.
55  convertLongTimeToDate(time);
        assert resultDate.equals(date);
    }

    @Test
    public void testGetDayOfMonth() throws ParseException {
        DateFormat dateFormat = new SimpleDateFormat("dd/MM/
            yyyy");
60  Date date = dateFormat.parse("23/09/2007");
        Date date2 = dateFormat.parse("29/02/2008");
        int daysDate = DateTimeHandler.getDayOfMonth(date);
        int daysDate2 = DateTimeHandler.getDayOfMonth(date2);
        assert daysDate == 23;
65  assert daysDate2 == 29;
    }

    @Test
    public void testGetCurrentMonthDays() {
70  int daysDate = DateTimeHandler.getCurrentMonthDays();
        Calendar mycal = new GregorianCalendar();
    }

```



```

        int daysInMonth = mycal.getActualMaximum(Calendar.
            DAY_OF_MONTH);
        assert daysDate == daysInMonth;
    }

    @Test
    public void testGetLastMonthDays(){
        int daysDate = DateTimeHandler.getLastMonthDays();
        Date now = new Date();
        Calendar c = Calendar.getInstance();
        c.setTime(now);
        c.add(Calendar.MONTH, -1);
        int daysInMonth = c.getActualMaximum(Calendar.
            DAY_OF_MONTH);
        assert daysDate == daysInMonth;
    }

    @Test
    public void testDateToString() throws ParseException {
        DateFormat dateFormat = new SimpleDateFormat("dd/MM/
            yyyy");
        Date date = dateFormat.parse("23/09/2007");
        String strDate = DateTimeHandler.dateToString(date, "
            dd-MM-yyyy");
        assert strDate.equals("23-09-2007");
    }

    @Test
    public void testDateMonthToString() throws ParseException
    {
        DateFormat dateFormat = new SimpleDateFormat("dd/MM/
            yyyy");
        Date date = dateFormat.parse("23/09/2007");
        String strDate = DateTimeHandler.dateMonthToString(
            date);
        assert strDate.equals("2007-09");
    }

    @Test
    public void testCreateDateMonthMap() {
        int daysDate = DateTimeHandler.getLastMonthDays();
        HashMap<String, List<Double>> dictDateMonthZero =
            DateTimeHandler.createDateMonthMap();
        assert dictDateMonthZero.size() == daysDate;
        assert !dictDateMonthZero.isEmpty();
    }

```

```

    for (Map.Entry<String, List<Double>> entry :
        dictDateMonthZero.entrySet()) {
110         assert entry.getValue().isEmpty();
    }
}

@Test
115 public void testCreateDateWeekMapSuccessRate() {

    HashMap<String, HashMap<String, Integer>>
        dictDateMonthZero = DateTimeHandler.
            createDateWeekMapSuccessRate();
    assert dictDateMonthZero.size() == 7;
    assert !dictDateMonthZero.isEmpty();
120 for (Map.Entry<String, HashMap<String, Integer>> entry
        : dictDateMonthZero.entrySet()) {
        assert entry.getValue().equals(new HashMap(){{
            put("fail", 0);
            put("success", 0);
        }});
125     }
}

@Test
public void testCreateDateMonthMapSuccessRate() {
130     int daysDate = DateTimeHandler.getLastMonthDays();
    HashMap<String, HashMap<String, Integer>>
        dictDateMonthZero = DateTimeHandler.
            createDateMonthMapSuccessRate();
    assert dictDateMonthZero.size() == daysDate;
    assert !dictDateMonthZero.isEmpty();
    for (Map.Entry<String, HashMap<String, Integer>> entry
        : dictDateMonthZero.entrySet()) {
135         assert entry.getValue().equals(new HashMap(){{
            put("fail", 0);
            put("success", 0);
        }});
    }
140 }

@Test
public void testCreateDateMonthMapTestCount() {
    int daysDate = DateTimeHandler.getLastMonthDays();
145     HashMap<String, Integer> dictDateMonthZero =
        DateTimeHandler.createDateMonthMapTestCount();

```

```

        assert dictDateMonthZero.size() == daysDate;
        assert !dictDateMonthZero.isEmpty();
        for (Map.Entry<String, Integer> entry :
            dictDateMonthZero.entrySet()) {
            assert entry.getValue() == 0;
150     }
    }

    @Test
    public void testGetTimeInQueue() throws Exception {
155         FreeStyleProject project = jenkins.
            createFreeStyleProject();
        project.getBuildersList().add(new
            BuildConfigurationStatisticsBuilder());
        jenkins.buildAndAssertSuccess(project);
        Run run = project.getBuilds().getLastBuild();
        long time = new TimeInQueueFetcher().getTimeInQueue(
            run);
160         long queuedTime = run.getStartTimeInMillis() - run.
            getTimeInMillis();
        assert time == queuedTime;
    }

    @Test
165     public void testCreateDateYearMap() {
        HashMap<String, List<Double>> dictDateYearZero =
            DateTimeHandler.createDateYearMap();
        assert dictDateYearZero.size() == 12;
        assert !dictDateYearZero.isEmpty();
        for (Map.Entry<String, List<Double>> entry :
            dictDateYearZero.entrySet()) {
170             assert entry.getValue().isEmpty();
        }
    }

    @Test
175     public void testCreateDateWeekMap() {
        HashMap<String, List<Double>> dictDateWeekZero =
            DateTimeHandler.createDateWeekMap();
        assert dictDateWeekZero.size() == 7;
        assert !dictDateWeekZero.isEmpty();
        for (Map.Entry<String, List<Double>> entry :
            dictDateWeekZero.entrySet()) {
180             assert entry.getValue().isEmpty();
        }
    }

```

```

    }

    @Test
185 public void testCreateDateYearMapSuccessRate() {
        HashMap<String, HashMap<String, Integer>>
            dictDateYearZero = DateTimeHandler.
                createDateYearMapSuccessRate();
        assert dictDateYearZero.size() == 12;
        assert !dictDateYearZero.isEmpty();
        for (Map.Entry<String, HashMap<String, Integer>> entry
            : dictDateYearZero.entrySet()) {
190         assert entry.getValue().equals(new HashMap(){{
                put("fail", 0);
                put("success", 0);
            }});
        }
195     }

    @Test
    public void testFilterPeriodBuild() throws Exception {
        FreeStyleProject project = jenkins.
            createFreeStyleProject();
200     project.getBuildersList().add(new
        BuildConfigurationStatisticsBuilder());
        jenkins.buildAndAssertSuccess(project);
        jenkins.buildAndAssertSuccess(project);
        List<Run> runList = new RunList<>(project);

205     BuildLogic instance1 = new BuildLogic(IntervalDate.
        WEEK, true, (RunList<Run>) runList);
        instance1.filterPeriodBuild();

        assert instance1.buildList.size() == 2;

210     BuildLogic instance2 = new BuildLogic(IntervalDate.ALL
        , true, (RunList<Run>) runList);
        instance2.filterPeriodBuild();

        assert instance2.buildList.size() == 2;

215     BuildLogic instance3 = new BuildLogic(IntervalDate.
        MONTH, true, (RunList<Run>) runList);
        instance3.filterPeriodBuild();

        assert instance3.buildList.size() == 2;
    }

```

```

220         BuildLogic instance4 = new BuildLogic(IntervalDate.
            YEAR, true, (RunList<Run>) runList);
        instance4.filterPeriodBuild();

        assert instance4.buildList.size() == 2;
    }
225 @Test
    public void testFilterFailedBuild() throws Exception {
        FreeStyleProject project = jenkins.
            createFreeStyleProject();
        project.getBuildersList().add(new
            BuildConfigurationStatisticsBuilder());
        jenkins.buildAndAssertSuccess(project);
230 project.getBuildersList().add(new Shell("echo1 hello")
            );
        jenkins.buildAndAssertStatus(Result.FAILURE, project);
        List<Run> runList = new RunList<>(project);
        for (Run run :runList) {
            System.out.println(run.getResult());
235     }

        BuildLogic instance1 = new BuildLogic(IntervalDate.
            WEEK, false, (RunList<Run>) runList);
        instance1.filterFailedBuild();

240     assert instance1.buildList.size() == 1;

    }

    @Test
245 public void testCreateDateQuarterMap() {
        HashMap<String, List<Double>> dictDateQuarterZero =
            DateTimeHandler.createDateQuarterMap();
        assert dictDateQuarterZero.size() == 4;
        assert !dictDateQuarterZero.isEmpty();
        for (Map.Entry<String, List<Double>> entry :
            dictDateQuarterZero.entrySet()) {
250             assert entry.getValue().isEmpty();
        }
    }

    @Test
255 public void testCreateDateDayMap() throws ParseException {

```

```

        HashMap<String, List<Double>> dictDateDayZero =
            DateTimeHandler.createDateDayMap();
        assert dictDateDayZero.size() == 24;
        assert !dictDateDayZero.isEmpty();
        for (Map.Entry<String, List<Double>> entry :
            dictDateDayZero.entrySet()) {
260         System.out.println(entry.getKey());
            assert entry.getValue().isEmpty();
        }
    }
}

```

Код BDD тестов на Java.

```

package io.jenkins.plugins.sample;
import io.cucumber.java.Before;
5 import io.cucumber.java.PendingException;
import io.cucumber.java.ru.*;
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
import hudson.model.*;
10 import hudson.tasks.Shell;
import hudson.util.RunList;
import org.apache.commons.lang.time.DateUtils;
import org.assertj.core.api.Assertions;
import org.junit.Test;
15 import org.junit.jupiter.api.extension.ExtendWith;
import org.junit.runner.RunWith;
import org.jvnet.hudson.test.JenkinsRule;
import org.mockito.Mock;
import org.mockito.Mockito;
20 import org.mockito.junit.MockitoJUnit;
import org.mockito.junit.MockitoJUnitRunner;
import org.mockito.junit.MockitoRule;

import static org.assertj.core.api.BDDAssertions.then;
25 import static org.mockito.ArgumentMatchers.any;
import static org.mockito.BDDMockito.given;
import static org.mockito.Mockito.mock;

import java.text.DateFormat;
30 import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.*;

```

```

35 public class MyStepdefs {

    // @Rule public MockitoRule mockitoRule = MockitoJUnit.rule
    //      ();

40     Job job = mock(Job.class);

    FreeStyleBuild build = mock(FreeStyleBuild.class);
    FreeStyleBuild build2 = mock(FreeStyleBuild.class);
    FreeStyleBuild build3 = mock(FreeStyleBuild.class);
45     FreeStyleBuild build4 = mock(FreeStyleBuild.class);

    //      RunList<Run> buildList;

50     private BuildDurationLogic buildDurationLogic;

    BuildConfigurationStatisticsAction
        buildConfigurationStatisticsAction;

    Date now;
55     Date twoMonthAgo;

    Date fiveMonthAgo;
    Date twoWeekAgo;
    String formatNow;
60     String formatTwoMonthAgo;
    String formatFiveMonthAgo;

    String formatNowQuarter;
    String formatTwoMonthAgoQuarter;
65     String formatFiveMonthAgoQuarter;
    String formatTwoWeekAgo;

    Map<String, Double> map;
    Map<String, Object> jsonMap;

70

    @Before
    public void prepareData() throws ParseException {
        // подготовить данные
75         now = new Date();
        twoMonthAgo = DateUtils.addMonths(now, -2);

```

```

fiveMonthAgo = DateUtils.addMonths(now, -5);
//twoWeekAgo = DateUtils.addWeeks(now, -2);

80  formatNow = DateTimeHandler.dateToString(now, "yyyy-MM
    -dd");
    formatTwoMonthAgo = DateTimeHandler.dateToString(
        twoMonthAgo, "yyyy-MM-dd");
    formatFiveMonthAgo = DateTimeHandler.dateToString(
        fiveMonthAgo, "yyyy-MM-dd");
    //formatTwoWeekAgo = DateTimeHandler.dateToString(
        fiveMonthAgo, "yyyy-MM-dd");

85  // quarter date
    formatNowQuarter = DateTimeHandler.dateToString(now, "
        yyyy-MM");
    formatTwoMonthAgoQuarter = DateTimeHandler.
        dateToString(twoMonthAgo, "yyyy-MM");
    formatFiveMonthAgoQuarter = DateTimeHandler.
        dateToString(fiveMonthAgo, "yyyy-MM");

90  given(build.getStartTimeInMillis())
        .willReturn(DateTimeHandler.
            convertDateToLongTime(now));
    given(build2.getStartTimeInMillis())
        .willReturn(DateTimeHandler.
            convertDateToLongTime(twoMonthAgo));
    given(build3.getStartTimeInMillis())
        .willReturn(DateTimeHandler.
95         convertDateToLongTime(fiveMonthAgo));
    given(build4.getStartTimeInMillis())
        .willReturn(DateTimeHandler.
            convertDateToLongTime(twoMonthAgo));

    given(build.getDuration())
100         .willReturn(10000L);
    given(build2.getDuration())
        .willReturn(20000L);
    given(build3.getDuration())
        .willReturn(10000L);
105  given(build4.getDuration())
        .willReturn(10000L);

    given(build.getResult())
110         .willReturn(Result.FAILURE);

```



```

        given(build2.getResult())
            .willReturn(Result.SUCCESS);

        given(build3.getResult())
115         .willReturn(Result.SUCCESS);

        given(build4.getResult())
            .willReturn(Result.SUCCESS);
    }

120
    @Дано("^выбраны параметры отображения за период \"([^\"]*)\"
        \" и с флагом отображения упавших сборок \"([^\"]*)\" \"$\"
        )
    public void получениеСборокЗаПериодИСФлагомОтображенияУпав
        шихСборок(IntervalDate period, Boolean failed) throws
        Throwable {

        RunList<Run> buildList = RunList.fromRuns(Arrays.
            asList(build, build2));
125         given(job.getBuilds())
            .willReturn(buildList);
        buildDurationLogic = new BuildDurationLogic(period,
            failed, job.getBuilds());
    }

130
    @Когда("^выбран статистический показатель \"([^\"]*)\" \"$\"")
    public void выбранСтатистическийПоказатель(Statistics
        statistics) throws Throwable {
        map = buildDurationLogic.getBuildsDuration(statistics)
            ;
    }

135
    @Тогда("^отбираются успешные и упавшие сборки за месяц с в
        ычислением суммарного времени$")
    public void отбираютсяУспешныеИУпавшиеСборкиЗаМесяцСВычисл
        ениемСуммарногоВремени() throws Throwable {
        then(map)
            .as("Check that map is not contain date two
                month ago entry with 20.0 time build
                duration")
            .doesNotContainEntry(formatTwoMonthAgo, 20.0)
140         .as("Check that map is not contain date two
                month ago")
            .doesNotContainKey(formatTwoMonthAgo)
            .as("Check that map is contain date now")

```

```

        .containsKey(formatNow)
        .as("Check that map is contain date now value
            and initial values")
145     .containsValues(10.0, 0.0)
        .as("Check that map is contain date now entry
            with 10.0 time build duration")
        .containsEntry(formatNow, 10.0)
        .as("Check that map has size how last month of
            days")
        .hasSize(DateTimeHandler.getLastMonthDays());
150     }

    @Дано("^сформировано (\\d+) запуска задания$")
    public void сформированыЗапускиЗадания(int countRuns)
        throws Throwable {

155         if (countRuns == 4) {
            RunList<Run> buildList2 = RunList.fromRuns(Arrays.
                asList(build, build2, build3, build4));
            given(job.getBuilds())
                .willReturn(buildList2);
            buildConfigurationStatisticsAction = new
                BuildConfigurationStatisticsAction(job);
160         } else {
            throw new PendingException();
        }
    }

165     @Когда("^выбраны параметры отображения за период
        \"([^\"]*)\" и с флагом отображения упавших сборок
        \"([^\"]*)\" и статистический показатель \"([^\"]*)\" \"$")
        )
    public void получениеСборокЗаПериодИСФлагомОтображенияУпав
        шихСборокПоПоказателю(String period, String failed,
        String statistics) throws Throwable {
        String jsonData = buildConfigurationStatisticsAction.
            getBuildDuration(period, failed, statistics);

        jsonMap = new Gson().fromJson(
170             jsonData, new TypeToken<HashMap<String, Object
                >>() {}.getType()

        );
    }

```

```

@Тогда("^отбираются успешные сборки за последние (\\d+) ме
    сяца с вычислением среднего времени$")
175 public void отбираютсяУспешныеСборкиЗаКварталСВычислениемС
    реднегоВремени(int countMonth) throws Throwable {
        then(jsonMap)
            .as("Check that json is correct")
            .doesNotContainKey(formatFiveMonthAgoQuarter)
            .as("Check that map is not contain date two
                month ago")
180             .doesNotContainEntry(formatNowQuarter, 10.0)
            .as("Check that map is contain date now")
            .containsKey(formatTwoMonthAgoQuarter)
            .as("Check that map is contain date now value
                and initial values")
            .containsValues(15.0)
185             .as("Check that map is contain date now entry
                with 10.0 time build duration")
            .containsEntry(formatTwoMonthAgoQuarter, 15.0)
            .as("Check that map has size how last month of
                days")
            .hasSize(countMonth);
190     }
}

```

Описание BDD тестов на Gherkin.

```

# language: ru
Функция: Получение продолжительности выполнения сборок

5  Сценарий: Получение продолжительности сборок за месяц
    Дано выбраны параметры отображения за период "MONTH" и с ф
        лагом отображения упавших сборок "true"
    Когда выбран статистический показатель "SUM"
    Тогда отбираются успешные и упавшие сборки за месяц с вычи
        слением суммарного времени

10 Сценарий: Получение среднего продолжительности сборок за ква
    ртал в JSON
    Дано сформировано 4 запуска задания
    Когда выбраны параметры отображения за период "QUARTER" и
        с флагом отображения упавших сборок "0" и статистически
        й показатель "AVG"
    Тогда отбираются успешные сборки за последние 3 месяца с в
        ычислением среднего времени

```

Программный код ui тестов на языке Python

Код базовой страницы для упрощения взаимодействия с элементами в тестах.

```

import pytest
import locators
from selenium.webdriver.support import expected_conditions as
    EC
5 from selenium.webdriver.support.wait import WebDriverWait
import random
import string
from selenium.webdriver.common.action_chains import
    ActionChains

10
class Base:
    driver = None

    @pytest.fixture(scope='function', autouse=True)
15 def setup(self, driver):
        self.driver = driver

    def wait(self, timeout=None):
        if timeout is None:
20             timeout = 15
        return WebDriverWait(self.driver, timeout)

    def find(self, locator, timeout=None):
        return self.wait(timeout).until(EC.
            visibility_of_element_located(locator))
25

    def click(self, locator, timeout=None):
        self.find(locator, timeout)
        self.wait(timeout).until(EC.element_to_be_clickable(
            locator)).click()

30 def input_text(self, locator, text, timeout=None):
        self.find(locator, timeout)
        element = self.wait(timeout).until(EC.
            element_to_be_clickable(locator))
        element.clear()
        element.send_keys(text)

```

Код конфигурационного файла, в котором прописаны основные настройки и фикстуры для взаимодействия с драйвером.

```

import pytest
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
5 from webdriver_manager.chrome import ChromeDriverManager
from webdriver_manager.firefox import GeckoDriverManager

def pytest_addoption(parser):
10     parser.addoption('--browser', default='chrome')
    parser.addoption('--url', default='http://localhost:5000/
        jenkins/job/tets1/buildConfigurationStatistics/')

@pytest.fixture()
15 def config(request):
    browser = request.config.getoption('--browser')
    url = request.config.getoption('--url')
    return {'browser': browser, 'url': url}

20 @pytest.fixture()
def driver(config):
    browser = config['browser']
    url = config['url']
25     if browser == 'chrome':
        driver = webdriver.Chrome(service=Service(
            ChromeDriverManager().install()))
    elif browser == 'firefox':
        driver = webdriver.Firefox(service=Service(
            GeckoDriverManager().install()))
    else:
30         raise RuntimeError(f'Unsupported browser: "{browser}"')

    driver.get(url)
    driver.maximize_window()
    yield driver
    driver.quit()

```

Код локаторов.

```

from selenium.webdriver.common.by import By

```

```

HEADER_LINK_PLUGIN = (By.XPATH, "//li[@class='jenkins -
    breadcrumbs__list-item']//a[contains(@href, '
    buildConfiguration')]" )
5 SIDE_LINK_PLUGIN = (By.XPATH, "//span[@class='task-link-
    wrapper ']/a[contains(@href, 'buildConfiguration')]" )
HEADER_PLUGIN = (By.CSS_SELECTOR, "#main-panel h1")
SUCCESS_RATE_CHART = (By.CSS_SELECTOR, "#successRateChart")
SUCCESS_RATE_SELECT = (By.XPATH, "//canvas[@id='
    successRateChart']/../../../../form//select")
SUCCESS_RATE_SELECT_MONTH = (By.XPATH, "//canvas[@id='
    successRateChart']/../../../../form//select//option")
10 SUCCESS_RATE_SELECT_DAY = (By.XPATH, "//canvas[@id='
    successRateChart']/../../../../form//select//option[text()='Day']
    ")

15 BUILD_DURATION_CHART = (By.CSS_SELECTOR, "#buildDurationChart"
    )
BUILD_DURATION_SELECT = (By.XPATH, "//canvas[@id='
    buildDurationChart']/../../../../form//select")
BUILD_DURATION_SELECT_MONTH = (By.XPATH, "//canvas[@id='
    buildDurationChart']/../../../../form//select//option")
BUILD_DURATION_CHECKBOX_AVERAGE = (By.XPATH, "(//canvas[@id='
    buildDurationChart']/../../../../form//input)[1]" )
BUILD_DURATION_CHECKBOX_FAILED = (By.XPATH, "(//canvas[@id='
    buildDurationChart']/../../../../form//input)[2]" )
20 BUILD_DURATION_SELECT_WEEK = (By.XPATH, "//canvas[@id='
    buildDurationChart']/../../../../form//select//option[text()='
    Week']" )

TIME_SPENT_QUEUE_CHART = (By.CSS_SELECTOR, "#timeSpentQueue")
TIME_SPENT_QUEUE_SELECT = (By.XPATH, "//canvas[@id='
    timeSpentQueue']/../../../../form//select")
25 TIME_SPENT_QUEUE_SELECT_MONTH = (By.XPATH, "//canvas[@id='
    timeSpentQueue']/../../../../form//select//option")
TIME_SPENT_QUEUE_CHECKBOX_AVERAGE = (By.XPATH, "(//canvas[@id
    ='timeSpentQueue']/../../../../form//input)[1]" )

TIME_SPENT_QUEUE_SELECT_YEAR = (By.XPATH, "//canvas[@id='
    timeSpentQueue']/../../../../form//select//option[text()='Year']"
    )

```

```

30 TEST_COUNT_CHART = (By.CSS_SELECTOR, "#testCount")
TEST_COUNT_SELECT = (By.XPATH, "//canvas[@id='testCount
    ']/../../../../form//select")
TEST_COUNT_SELECT_MONTH = (By.XPATH, "//canvas[@id='testCount
    ']/../../../../form//select//option")
TEST_COUNT_CHECKBOX_FAILED = (By.XPATH, "(//canvas[@id='
    testCount']/../../../../form//input)[1]")

35 TEST_COUNT_SELECT_QUARTER = (By.XPATH, "//canvas[@id='
    testCount']/../../../../form//select//option[text()='Quarter']")

ARTIFACTS_SIZE_CHART = (By.CSS_SELECTOR, "#artifactsSize")
ARTIFACTS_SIZE_SELECT = (By.XPATH, "//canvas[@id='
    artifactsSize']/../../../../form//select")
ARTIFACTS_SIZE_SELECT_MONTH = (By.XPATH, "//canvas[@id='
    artifactsSize']/../../../../form//select//option")
40 ARTIFACTS_SIZE_CHECKBOX_AVERAGE = (By.XPATH, "(//canvas[@id='
    artifactsSize']/../../../../form//input)[1]")
ARTIFACTS_SIZE_CHECKBOX_FAILED = (By.XPATH, "(//canvas[@id='
    artifactsSize']/../../../../form//input)[2]")

ARTIFACTS_SIZE_SELECT_ALL = (By.XPATH, "//canvas[@id='
    artifactsSize']/../../../../form//select//option[text()='All']")

```

Код UI-тест-кейсов.

```

import time

from base import Base
5 import locators
import pytest

class TestCase(Base):
10     @pytest.mark.UI
    def test_open_tab(self):
        assert 'buildConfigurationStatistics' in self.driver.
            current_url
        assert 'Statistics for job tets1' in self.find(
            locators.HEADER_PLUGIN).text
        assert 'Build Configuration Statistics' in self.find(
            locators.HEADER_LINK_PLUGIN).text
15     assert 'Build Configuration Statistics' in self.find(
        locators.SIDE_LINK_PLUGIN).text

    @pytest.mark.UI

```

```

def test_success_rate_chart(self):
    self.find(locators.SUCCESS_RATE_CHART)
    self.find(locators.SUCCESS_RATE_SELECT)
    assert 'Month' in self.find(locators.
        SUCCESS_RATE_SELECT_MONTH).text

@pytest.mark.UI
def test_build_duration_chart(self):
    self.find(locators.BUILD_DURATION_CHART)
    self.find(locators.BUILD_DURATION_SELECT)
    assert 'Month' in self.find(locators.
        BUILD_DURATION_SELECT_MONTH).text
    assert not (self.find(locators.
        BUILD_DURATION_CHECKBOX_AVERAGE)).is_selected()
    assert not (self.find(locators.
        BUILD_DURATION_CHECKBOX_FAILED)).is_selected()

@pytest.mark.UI
def test_time_spent_queue_chart(self):
    self.find(locators.TIME_SPENT_QUEUE_CHART)
    self.find(locators.TIME_SPENT_QUEUE_SELECT)
    assert 'Month' in self.find(locators.
        TIME_SPENT_QUEUE_SELECT_MONTH).text
    assert not (self.find(locators.
        TIME_SPENT_QUEUE_CHECKBOX_AVERAGE)).is_selected()

@pytest.mark.UI
def test_test_count_chart(self):
    self.find(locators.TEST_COUNT_CHART)
    self.find(locators.TEST_COUNT_SELECT)
    assert 'Month' in self.find(locators.
        TEST_COUNT_SELECT_MONTH).text
    assert not (self.find(locators.
        TEST_COUNT_CHECKBOX_FAILED)).is_selected()

@pytest.mark.UI
def test_artifacts_size_chart(self):
    self.find(locators.ARTIFACTS_SIZE_CHART)
    self.find(locators.ARTIFACTS_SIZE_SELECT)
    assert 'Month' in self.find(locators.
        ARTIFACTS_SIZE_SELECT_MONTH).text
    assert not (self.find(locators.
        ARTIFACTS_SIZE_CHECKBOX_AVERAGE)).is_selected()
    assert not (self.find(locators.
        ARTIFACTS_SIZE_CHECKBOX_FAILED)).is_selected()

```



```

@pytest.mark.UI
def test_change_value_select_period(self):
55     self.find(locators.SUCCESS_RATE_SELECT_DAY).click()
        #time.sleep(4)
        self.find(locators.BUILD_DURATION_SELECT_WEEK).click()
        self.find(locators.TIME_SPENT_QUEUE_SELECT_YEAR).click()
            ()
        self.find(locators.TEST_COUNT_SELECT_QUARTER).click()
60     self.find(locators.ARTIFACTS_SIZE_SELECT_ALL).click()

@pytest.mark.UI
def test_change_value_checkbox(self):
    self.find(locators.BUILD_DURATION_CHECKBOX_AVERAGE).
        click()
65     self.find(locators.BUILD_DURATION_CHECKBOX_FAILED).
        click()
    assert (self.find(locators.
        BUILD_DURATION_CHECKBOX_AVERAGE)).is_selected()
    assert (self.find(locators.
        BUILD_DURATION_CHECKBOX_FAILED)).is_selected()
    #time.sleep(4)
    self.find(locators.TIME_SPENT_QUEUE_CHECKBOX_AVERAGE).
        click()
70     assert (self.find(locators.
        TIME_SPENT_QUEUE_CHECKBOX_AVERAGE)).is_selected()

    self.find(locators.TEST_COUNT_CHECKBOX_FAILED).click()
    assert (self.find(locators.TEST_COUNT_CHECKBOX_FAILED)
        ).is_selected()

75     self.find(locators.ARTIFACTS_SIZE_CHECKBOX_AVERAGE).
        click()
    self.find(locators.ARTIFACTS_SIZE_CHECKBOX_FAILED).
        click()
    assert (self.find(locators.
        ARTIFACTS_SIZE_CHECKBOX_AVERAGE)).is_selected()
    assert (self.find(locators.
        ARTIFACTS_SIZE_CHECKBOX_FAILED)).is_selected()

```

Приложение 4

Программный код для обработки результатов апробации

Код формирования веток frontend-maven-plugin с разными версиями.

```

from git import Repo
import re
repo = Repo('C:\\frontmavemplugversion\\frontend-maven-plugin'
    )
5
commits = list(repo.iter_commits('master'))
commit_first = commits[0]

i = 1
10 for commit in commits[1:]:
    diff_changes = repo.git.diff("--shortstat", commit_first,
        commit)
    list_changes = re.findall(r'\d+', diff_changes)
    if i == 5:
        diff_changes = repo.git.diff(commit_first, commit)
15 if len(list_changes) < 2:
        continue
    if len(list_changes) > 2:
        list_changes_count = int(list_changes[1]) + int(
            list_changes[2])
    else:
20     list_changes_count = int(list_changes[1])

    if list_changes_count > 500:
        print(i)
        print("500+: ", diff_changes, commit)
25     repo.git.reset(commit)
        new_branch_name = 'reverse_branch' + str(i)
        repo.git.branch(new_branch_name)

        repo.git.checkout(new_branch_name)
30     repo.git.push('origin', new_branch_name)
        commit_first = commit
        i+=1
        if i > 12:
            break

```

Код редактирования xml файлов с конфигурациями сборок.

```
import xml.etree.ElementTree as ET
```

```

from datetime import datetime
import dateutil.relativedelta
5 import re
from random import randrange

curr_dt = datetime.now()

10 len_builds = 18

path_to_builds = "C:\\buildConfigurationStatistics\\work\\jobs
    \\test-Build-frontend-maven-plugin\\builds\\"

name_file_build = "build.xml"
15
for i in range(0, len_builds):
    root = ET.parse(f"{path_to_builds}{i + 1}\\{
        name_file_build}")

    print("Current datetime: ", curr_dt)
20    timestamp = int(round(curr_dt.timestamp()))*1000

    print("Integer timestamp of current datetime: ", timestamp
        )

    new_date = curr_dt - dateutil.relativedelta.relativedelta(
        months=i)
25

    print("new_date: ", new_date)
    timestamp_new = int(round(new_date.timestamp()))*1000

    print("timestamp_new: ", timestamp_new)
30

    new_root = root.getroot()
    print(root)
    timestampText = new_root.find('timestamp').text
    print(timestampText)
35    root.find('timestamp').text = str(timestamp_new)

    startTimeText = new_root.find('startTime').text
    print(startTimeText)
    rand_milli = randrange(5, 25)
40    root.find('startTime').text = str(timestamp_new +
        rand_milli)
    print(rand_milli)

```

```
root.write(f'{{path_to_builds}}{{i + 1}}\\{{name_file_build}}',  
           encoding = "UTF-8", xml_declaration = True)  
  
45 # rewrite version xml  
  
with open(f"{{path_to_builds}}{{i + 1}}\\{{name_file_build}}", "  
         r") as rfile:  
    s = rfile.read()  
    rplce = re.sub('version=\'1.0\'', "version=\'1.1\'", s  
                  )  
50 with open(f"{{path_to_builds}}{{i + 1}}\\{{name_file_build}}", "  
         w") as wfile:  
    wfile.write(rplce)
```