ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
ИНСТИТУТА КОМПЬЮТЕРНЫХ НАУК И КИБЕРБЕЗОПАСНОСТИ
ВЫСШАЯ ШКОЛА ПРОГРАММНОЙ ИНЖЕНЕРИИ

Отчет о прохождении преддипломной практики на тему: «Разработка плагина Jenkins для визуализации статистики работы сборок Jenkins»

Кухто Андрея Денисовича, гр. з5130903/90301

Направление подготовки: 09.03.03 Прикладная информатика.
Место прохождения практики: СПбПУ, ИКНК, ВШПИ г. Санкт-Петербург, ул. Обручевых, д. 1, лит. В.
Сроки практики: с 19.12.2023 по 09.01.2024.
Руководитель практической подготовки от ФГАОУ ВО «СПбПУ»: Пархоменко Владимир Андреевич, ст. преподователь ВШПИ.
Оценка: _____

Руководитель практической подготовки
от ФГАОУ ВО «СПбПУ» В.А. Пархоменко

Обучающийся А.Д. Кухто

Дата: 09.01.2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

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

Для упрощение 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. Существует множество средств CI, 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 [10].

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 [8].

Одним из лучших средств CI, 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 в сравнении 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 Build Configuration Statistics. Часть плагинов реализует частичный функционал модуля TeamCity, подробнее о недостатках таких средств будет описано в сравнительном анализе и обзоре аналогов. 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.

В данной работе будет разработан плагин, который обеспечит визуализацию статистики работы сборок.

Объект исследования — инструменты для сборки приложений.

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.

ГЛАВА 1. АНАЛИЗ СРЕДСТВ СБОРКИ И ВИЗУАЛИЗАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В первой главе рассмотрим:

A. Процесс и инструменты сборки приложения.
B. Обзор и сравнительный анализ средств CI/CD в контексте сборок приложения.
C. Понятие статистик и визуализации сборок в контексте инструментов CI/CD.
D. Обзор и сравнительный анализ плагинов Jenkins по визуализации статистики работы сборок.
E. Требования к разработке.

1.1. Обзор и сравнительный 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 [25]:

– 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 [9]:

– 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 отличии 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. Особенности и сравнительный анализ CI/CD систем

CI/CD — это технология автоматизации тестирования и доставки/развертывания готового приложения заказчику [14]. 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 [6]:

- 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, становиться 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.

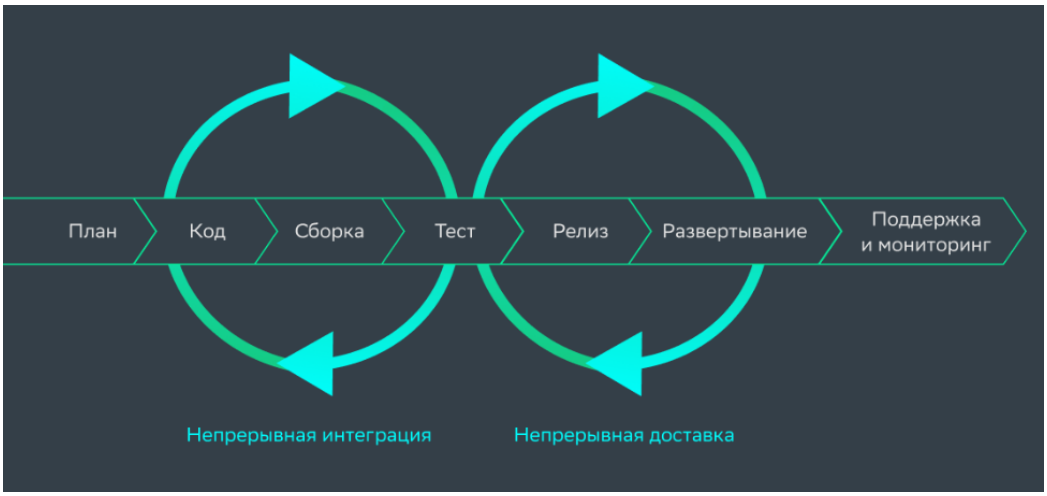

Рис.1.1. Цикл CI/CD [2]

1.3. Обзор и сравнительный анализ средств CI/CD

На данный момент существует множеств 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 [24].

TeamCity - это сервер CI от компании Jetbrains [28], 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 [19], 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 [16], 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 [12], который связывает автоматизированные сборки, тесты и выпуски в единый рабочий процесс.

В таблице 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 лицензировании 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

Критерий	Jenkins	TeamCity	GitLab CI	CircleCI	Bamboo
Открытый исходный код	+	-	+	-	-
Цена	Бесплатно	от 45$ в месяц	от 5$ в месяц	от 15$ в месяц	от 1200$ в год
Поддержка вендора	-	+	+	+	+
Поддержка репозиториев Git	Любой репозиторий	GitHub, GitLab, Bitbucket	GitHub, GitLab, Bitbucket	GitHub, GitLab, Bitbucket	Любой репозиторий

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 [15], в сравнении со всеми остальными инструментами, за исключением Jenkins.

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 количество 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.

1.4. Статистика и визуализация работы сборок в контексте CI/CD

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 производит мониторинг и визуализацию информацию статистики работы сборок во времени. Под статистикой работы сборок, подразумеваются следующие параметры относительно собираемых метрик, таких как время продолжительности исполнения сборки, время проведенного в очереди сборкой, размеров полученных по итогом сборки артефактов:

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

1.5. Обзор и сравнительный анализ плагинов Jenkins по визуализации статистики сборок

В данной работе производится разработка плагина для визуализации метрик сборки Jenkins, основание для разработки является отсутствие плагина, который полностью визуализирует метрики сборок в Jenkins, аналогично встроенному модулю в Teamcity. Многие российские ИТ-компании широко использовали TeamCity, этот модуль позволял отслеживать состояние отдельных конфигураций сборки с течением времени,собирать статистические данные по всей истории сборки и отображать их в виде наглядных диаграмм. В данной работе будет разработан плагин для воссоздания это модуля в Jenkins, с дополнительным функционалом, которого не хватало в TeamCity.

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 сборок 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.

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

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.

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.

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 отслеживания аномальных результатов метрик, которое позволит определить проблемные сборки, у которых возникают временные проблемы с процессом CI/CD. Также данные плагины не предлагают динамическое изменение графиков по мере изменения временного интервала или установления фильтров.

Критерий	Build Monitor Plugin	Global Build Stats Plugin	Build Time Blame	Плагин разрабатываемый
Наличие отслеживания аномальных результатов метрик	-	-	-	+
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	-	+	-	+

1.6. 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 и за весь промежуток времени, т.е. если, например, был выбран промежуток времени месяц, то должен выполняться следующий набор действий:

A. Должна собираться информация о требуемой метрики у всех сборок.
B. Производиться фильтрация сборок - т.е. отбираться только сборки за последний месяц (в том числе упавшие, если был выбран данный чекбокс).
C. Полученные сборки должны группироваться по дням т.е. на итоговом графике должно быть 30/31 точка или столбца.
D. Если необходимо производиться вычисление среднего среди всех сгруппированных за день метрик сборок.
E. 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.

Помимо реализации перечисленных функция, которые полностью аналогичны функциям TeamCity, плагин будет вычислять аномальные значения за определенный период - т.е. можно будет наглядно увидеть, например, в какие дни произошли сбои в работе сборок, это может быть например слишком большой размер артефактов у одной сборки за какой-то промежуток времени.

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 расчитаны веса каждой сборки/с-

борок 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 понятно, 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.

1.7. Выводы

По всем описанным выше разделам можно прийти к выводу, что данный плагин актуален для ИТ-компаний, которые ранее отдавали предпочтению 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 компания, которые приняли решение отказаться от TeamCity в пользу Jenkins из-за больших денежных затрат на лицензию. Также будет 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 в приложении П3.1. 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 в приложении П2.1.-3. Основное внимание на диаграмме уделяется визуализации статистики сборок, поскольку это изначально является целью разработки. Также там будут отражены дополнительные функции такие как фильтрация, и высчитывание статистик метрик.

2.2. Архитектура Jenkins

Перед объяснением построения архитектуры плагинов Jenkins, необходимо привести схему архитектуры Jenkins, где будет отображено место разрабатываемых плагинов в CI системе (рисунок 2.1). Установленные плагины Jenkins-CI, а также локальные сценарии и приложения выполняются на сервере Jenkins-CI и предоставляют расширяемый набор функций управления и обработки данных [11].

Архитектура плагинов использует точки расширения, которые, предоставляют разработчикам плагинов возможности реализации для расширения функциональности системы Jenkins [29]. 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.

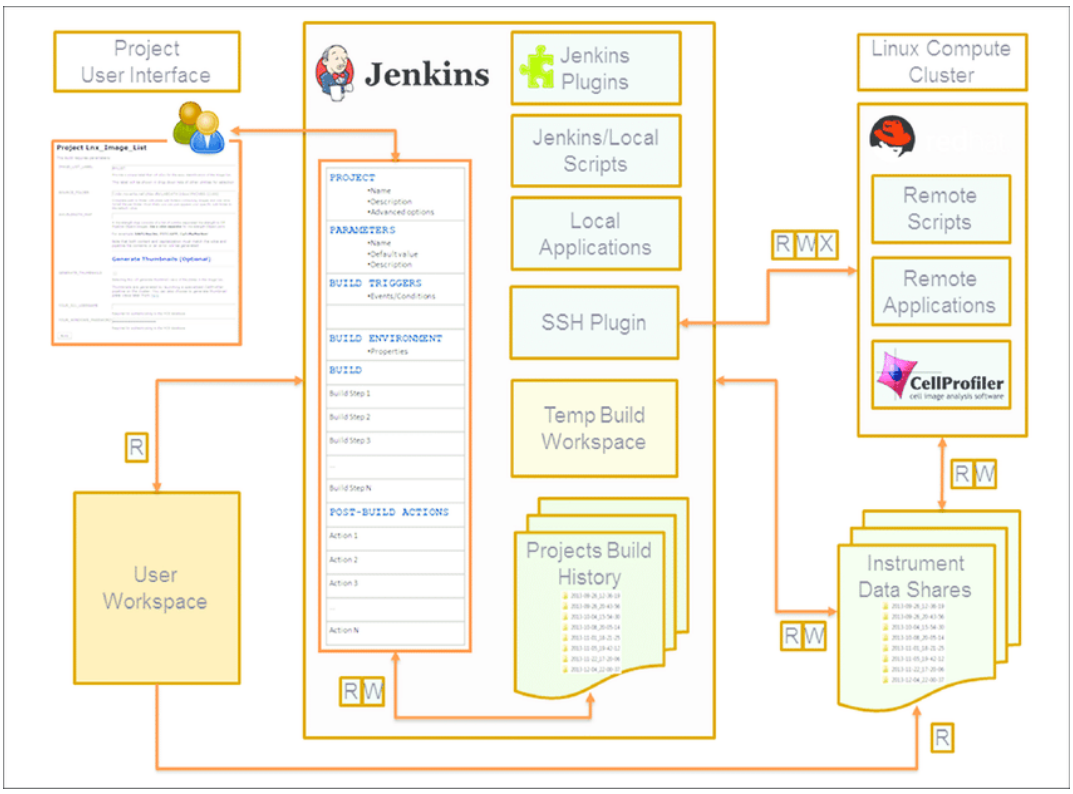

Рис.2.1. Архитектура Jenkins [11]

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 описаны в приложении П1.1.

При рассмотрении диаграммы необходимо отметить, что два класса являются встроенными в Jenkins, это TransientActionFactoryб 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 визуализиовать 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 серевер 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 2.2 представлено изображение архитектуры плагина.

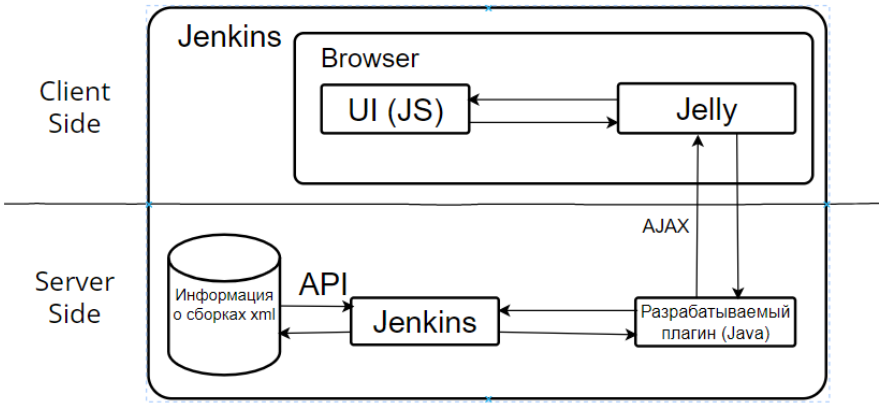

Рис.2.2. Архитектура плагина

2.4. Языки программирования

Для программирования плагина будет использоваться язык Java. Поскольку Jenkis 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. Groov это динамический язык с возможностями статической типизации и статической компиляции для платформы Java[20], 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 сравнии с Groovy, Java обладает большей производительностью[27], 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 [21]. 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 легковесный, интерпретируемый или JIT-компилируемый, объектно-ориентированный язык [22], 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 [17], 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 [23]. В Jelly можно вызывать функции Java, использовать такие синтакситеские 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

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 [13]. 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.

2.7. 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 приложения 1, а также полученные результаты.

3.1. Описание разработанных классов

В процессе написания кода плагины были запрограммированы классы в соответствии с диаграммой классов из приложения 1.

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 average) - 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 average) - метод для получения информации о созданных артефактов 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 average) - метод для получения информации о времени нахождения в очереди запусков сборки, за определенный период, с заданными настройками.

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, Double> createDateMonthMap() - метод, создающий начальную структуру данных за прошедший месяц по дням, где в качестве значений словаря используются нули;
- HashMap<String, 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, Double> createDateDayMap() - метод, создающий начальную структуру данных за прошедший день по часам, где в качестве значений словаря используются нули;
- 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, Double> createDateYearMap() - метод, создающий начальную структуру данных за прошедший год по месяцам, где в качестве значений словаря используются дробные нули;
- HashMap<String, Double> createDateQuarterMap() - метод, создающий начальную структуру данных за прошедший квартал по месяцам, где в качестве значений словаря используются дробные нули;
- HashMap<String, Double> createDateWeekMap() - метод, создающий начальную структуру данных за прошедшую неделю по дням, где в качестве значений словаря используются дробные нули;
- 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 предопределенные константы:

- DAY - день;
- WEEK - неделя;
- MONTH - месяц;
- YEAR - год;
- QUARTER - квартал;
- ALL - константа для определения, того что будут вычисляться равные периоды для данных за все время, от начального запуска сборки до конечного.

3.1.4. 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.

3.1.5. 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 отборок только тех запусков, которые удовлетворяют заданному периоду;

– void filterFailedBuild() - метод, который производит отборок только тех запусков, которые удовлетворяют полю failed, т.е. в зависимости от значения флага, либо включает в выборку упавшие сборки, либо нет.

3.1.6. 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 артифакта 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(Boolean average) - 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.

3.1.7. 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(Boolean average) - 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.

3.1.8. 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.

3.1.9. 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 TS, 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.

3.1.10. 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(Boolean average) - 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.

3.1.11. Файлы JS и Jelly

В JS определяются функции событий для выбора элемента из выпадающего списка и взаимодействия с флажками. Для каждой метрики используется своя функции, 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(period) - функция в которой происходит подготовка данных, формирование настроек и создание графика по метрике SR;
- createTestCountChart(period) - функция в которой происходит подготовка данных, формирование настроек и создание графика по метрике TC;
- createBuildDurationChart(period) - функция в которой происходит подготовка данных, формирование настроек и создание графика по метрике BD;
- createArtifactSizeChart(period) - функция в которой происходит подготовка данных, формирование настроек и создание графика по метрике AS;
- createTimeQueueChart(period) - 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 одно периода, например, месяца, будет получена сводная информация по каждой сборке или нескольких сборок запущенных в один день. Графики отображаются посредством переходна 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, усреднить метрику.

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

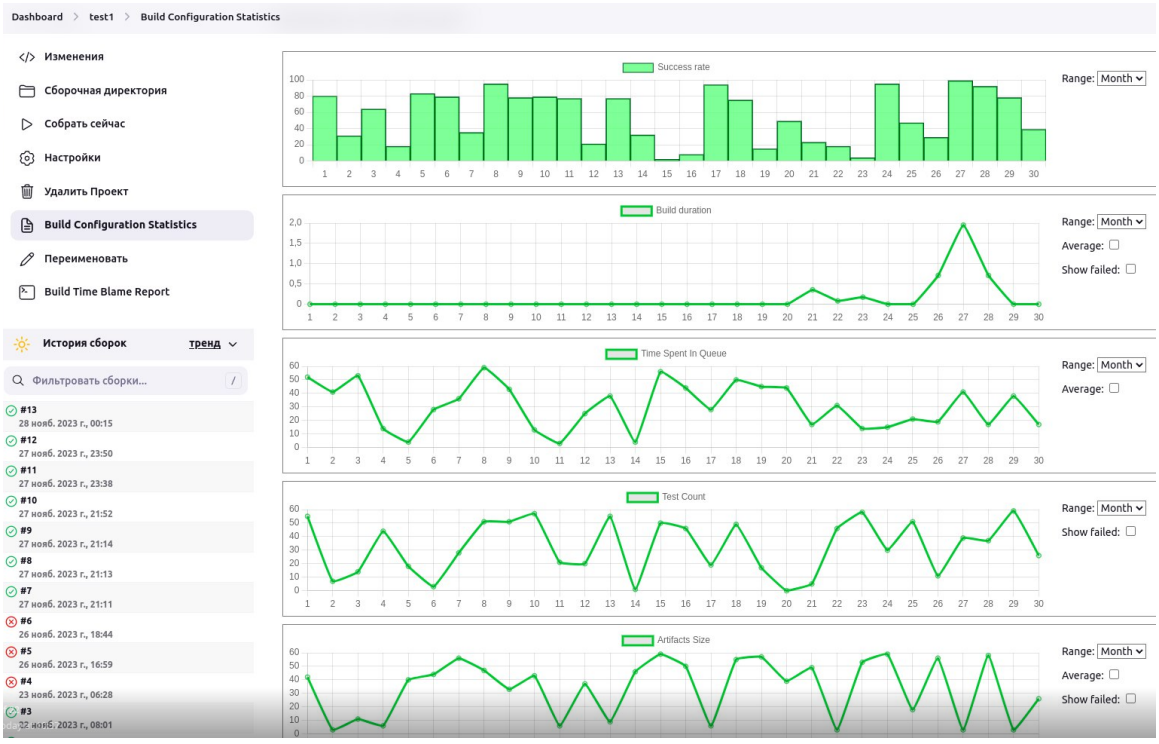

Рис.3.1. Интерфейс плагина Jenkins

Интерфейс страницы плагина с графиками в системе Jenkins на странице задания при выборе всех включенных настроек, а также с выбранным периодом неделя показан на рисунке 3.2.

Рис.3.2. Интерфейс графиков с включенным настройками и периодом неделей

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

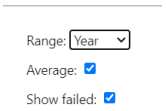

Рис.3.3. Интерфейс элементов управления

При взаимодействии с раскрывающимся списком должен вызываться Java метод, который пересчитает и отфильтрует необходимые сборки Jenkins и динамически отобразит результаты по выбранными периоду, также динамически должна производить обработка метрик сборок, при выборе основной метрики ни как суммы всех значений, а как среднего, 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.

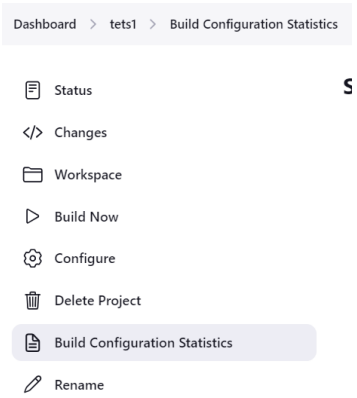

Рис.3.4. Интерфейс элементов управления

Код плагина представлен в приложении 4.

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:

A. Описаны методы тестирования.
B. Проведена апробация плагина в системе Jenkins.
C. Разработан код тестирования плагина.

4.1. Методы тестирования

Тестирование программного обеспечения — обширное понятие, которое включает планирование, проектирование и, собственно, выполнение тестов [7]. 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 требования, 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 [4].

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 [3], 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 совместить с подробным описание 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 [5] ручные UI кейсы, находятся в самой верхней ее части и не должны занимать достаточно большое место в системе тестирования. С дрогой стороны для улучшения покрытия требования, предъявляемых к продукты 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 [30].

№	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 соотвествует всем часам прошедшего дня, на каждый час отображены корректные значения размера итоговых артефактов полученного по результатам задания
6	Проверка корректного отображения при выборе периода ALL	Отображены сборки со всего периода прошедшего, информация поделена на равное части
7	Проверка корректного отображения аномальных значений	Отображены номера сборок, возле каждого графика, у которых аномальное значений метрики соответствующей графику
8	Проверка корректного расчета предугаданной ’следующей’ сборки	Метрики предугаданной сборки рассчитываются корректно для каждого графика

4.1.1. Unit тестирование

При написание юнит тестов используется метод белого ящика. Данный метод предоставляет тестировщику полное знание тестируемого приложения, включая доступ к исходному коду и проектной документации [32], 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 5.

4.1.2. 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 [31]. 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 6.

4.2. Апробация плагина

Апробация плагина будет проводится в системе CI Jenkins для которой и был разработан плагин визуализации. Для того чтобы провести апробацию 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 [26] 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 апробации

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.

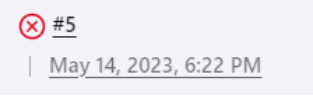

Рис.4.1. Сборка с датой в прошлом

В ходе апробации плагина были сгенерированы следующие сборки.

Рис.4.2. Сгенерированные сборки

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;
– 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.

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 разделы в сборке Post-build Actions, в котором выплавлялась команда Archive the artifacts.

Рис.4.4. Короткая успешная сборка с артефактом

```
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

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 текущес 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 4.4.

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.

Для создания небольших файлов-артефактов использовалась команда cmd *echo "tempbuild1111111111111111111"1>tembuild.txt*. А для генерация файлов в 1Кб команда *fsutil file createnew tem.txt 1048576*.

4.2.2. Апробация на проекте с открытым исходным кодом

Для того, чтобы убедиться, что плагин работает также на уже существующих проектах, необходимо провести апробацию на стороннем проекте. В качестве такого проекта был выбран frontend-maven-plugin (https://github.com/eirslett/frontend-maven-plugin).

Это плагин, который загружает/устанавливает Node и NPM локально для вашего проекта, запускает npm install, а затем любую комбинацию Bower , Grunt , Gulp , Jspm , Karma или Webpack и может работать в Windows, OS X и Linux [18]. У проекта 865 forks на GitHub, а также более 4 тысяч звезд, из чего следует, что его разработка была полезна для ИТ сообщества и активно используется разработчиками.

Следуя, указаниям из документации для сборки проекта необходимо вызвать команду *mvn clean install*. В случае тестирования разработанного плагина в системе Jenkins, также использовался ключ -l, который сохранит логи сборки проекта в отдельный файл clean. Также в настройках сборки Jenkins было настроено действие после сборки для создания артефакта из полученных логов.

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

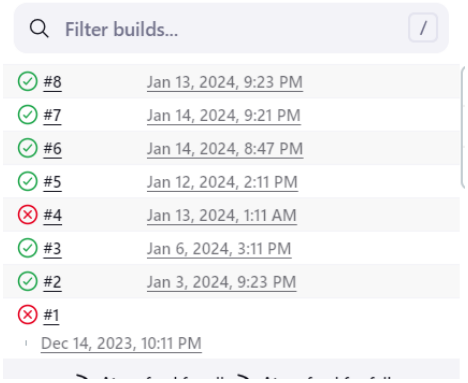

Рис.4.6. Сгенерированные сборки реального проекта

Рис.4.7. Результаты апробации на реальном проекте

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.

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.

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

4.3. 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 отрабатывает при разных поданных на вход исходных данных и настройках.

ЗАКЛЮЧЕНИЕ

В результате проведенной работы был разработан прототип плагина для визуализации статистики сборок Jenkins. Были выбраны средства и инструменты разработки, спроектирована архитектура плагина, описаны функциональные возможности, а также разработан программный код и интерфейс плагина.

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. Краткий обзор методологии CI/CD: принципы, этапы, плюсы и минусы. — URL: https://cloud.ru/ru/blog/cicd-about (дата обращения: 20.11.2023).

3. Матрица трассабилити. — URL: https://habr.com/ru/companies/simbirsoft/articles/412677/ (дата обращения: 22.12.2023).

4. *Панина О.* Особенности тестирования «черного ящика». — 2017. — URL: https://quality-lab.ru/blog/key-principles-of-black-box-testing/ (дата обращения: 22.12.2023).

5. Подробнее про пирамиду тестирования. — URL: https://habr.com/ru/articles/672484/ (дата обращения: 22.12.2023).

6. *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).

7. Тестирование ПО: суть профессии, требования и заработная плата. — URL: https://habr.com/ru/companies/habr_career/articles/517812/ (дата обращения: 22.12.2023).

8. Учимся создавать и настраивать Jenkins Jobs. — URL: https://habr.com/ru/companies/slurm/articles/742504/ (дата обращения: 20.11.2023).

9. Что такое Gradle. — URL: https://apptractor.ru/develop/gradle.html (дата обращения: 20.11.2023).

10. Что такое сборка в программировании. — URL: https://uchet-jkh.ru/i/cto-takoe-sborka-v-programmirovanii (дата обращения: 20.11.2023).

11. 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.

12. Bamboo Docs. — URL: https://confluence.atlassian.com/bamboo/bamboo-documentation-289276551.html (visited on 20.11.2023).

13. Chart.js Documentation. — URL: https://www.chartjs.org/docs/latest/ (visited on 20.11.2023).

14. CI/CD. — URL: https://blog.skillfactory.ru/glossary/ci-cd (visited on 20.11.2023).

15. 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).

16. Circle CI Docs. — URL: https://circleci.com/docs/about-circleci/ (visited on 20.11.2023).

17. CSS Documentation. — URL: https://developer.mozilla.org/en-US/docs/Web/CSS (visited on 20.11.2023).

18. frontend-maven-plugin. — URL: https://github.com/eirslett/frontend-maven-plugin (visited on 25.12.2023).

19. GitLab CI Docs. — URL: https://docs.gitlab.com/ee/ci/ (visited on 20.11.2023).

20. Groovy Docs. — URL: https://groovy-lang.org/documentation.html (visited on 20.11.2023).

21. Java Docs. — URL: https://docs.oracle.com/en/java/ (visited on 20.11.2023).

22. JavaScript Documentation. — URL: https://developer.mozilla.org/ru/docs/Web/JavaScript (visited on 20.11.2023).

23. Jelly Documentation. — URL: https://commons.apache.org/proper/commons-jelly/ (visited on 20.11.2023).

24. Jenkins Documentation. — URL: https://www.jenkins.io/doc/ (visited on 20.11.2023).

25. Maven Documentation. — URL: https://maven.apache.org/what-is-maven.html (visited on 20.11.2023).

26. *Pathare A.* Tutorial: Developing Complex Plugins for Jenkins. — 2022. — URL: https://www.velotio.com/engineering-blog/jenkins-plugin-development (visited on 22.12.2023).

27. *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).

28. TeamCity Docs. — URL: https://www.jetbrains.com/help/teamcity/teamcity-documentation.html (visited on 20.11.2023).

29. 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).

30. Unit testing. — URL: https://www.techtarget.com/searchsoftwarequality/definition/unit-testing (visited on 22.12.2023).

31. WebDriver Docs. — URL: https://www.selenium.dev/documentation/webdriver/ (visited on 20.12.2023).

32. What is White Box Testing? — URL: https://www.checkpoint.com/cyber-hub/cyber-security/what-is-white-box-testing (visited on 22.12.2023).

Приложение 1

UML диаграмма классов плагина

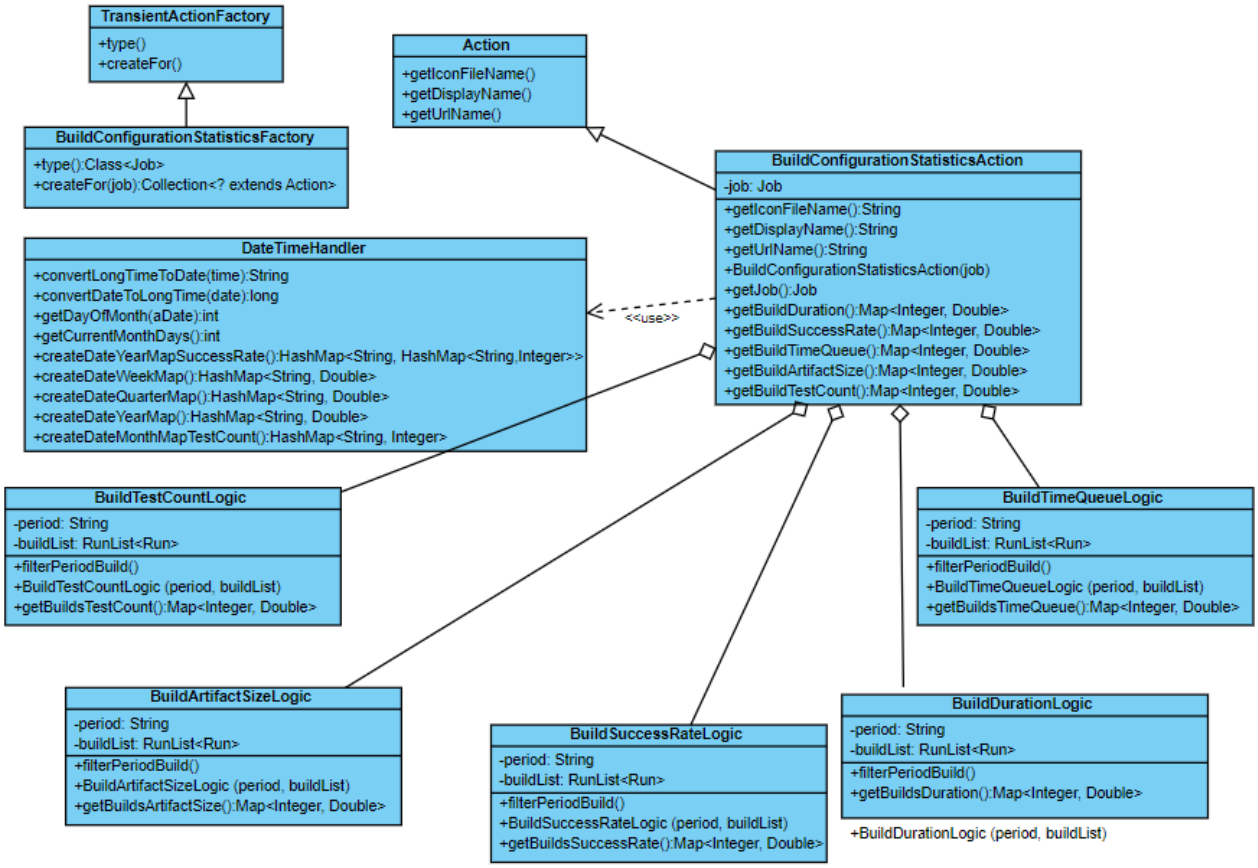

Рис.П1.1. Диаграмма классов плагина

Приложение 2

Idef0

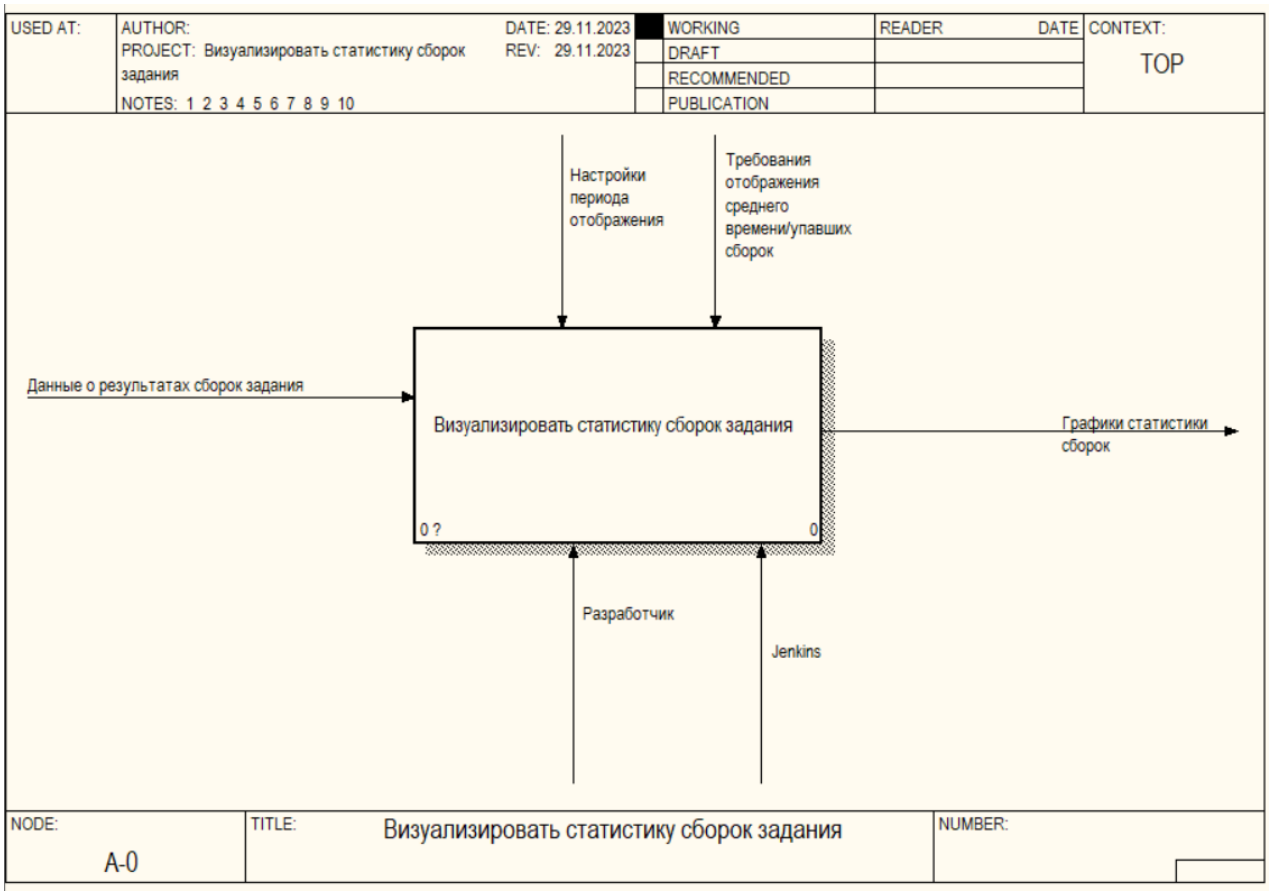

Рис.П2.1. Процесс визуализации сборок

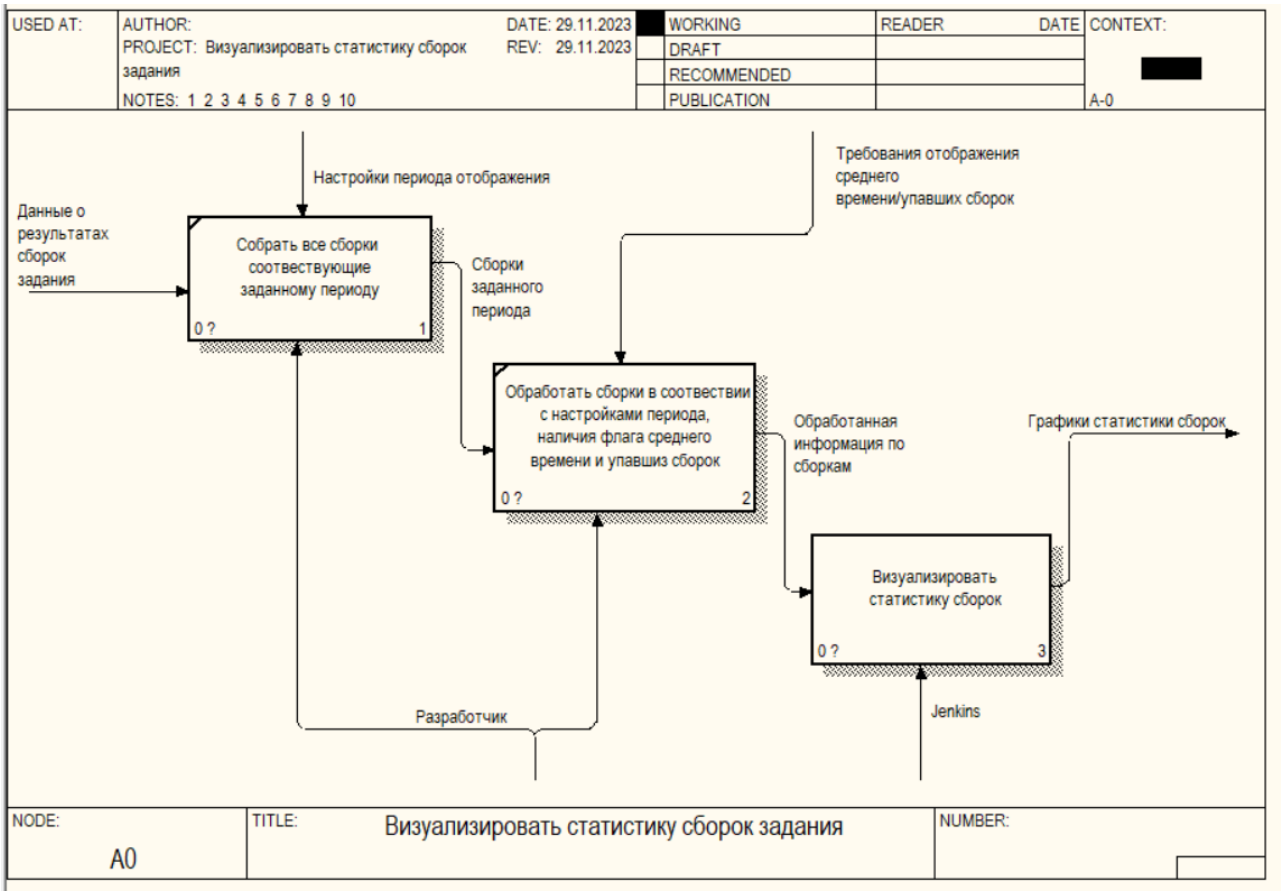

Рис.П2.2. Детализация процесса визуализации

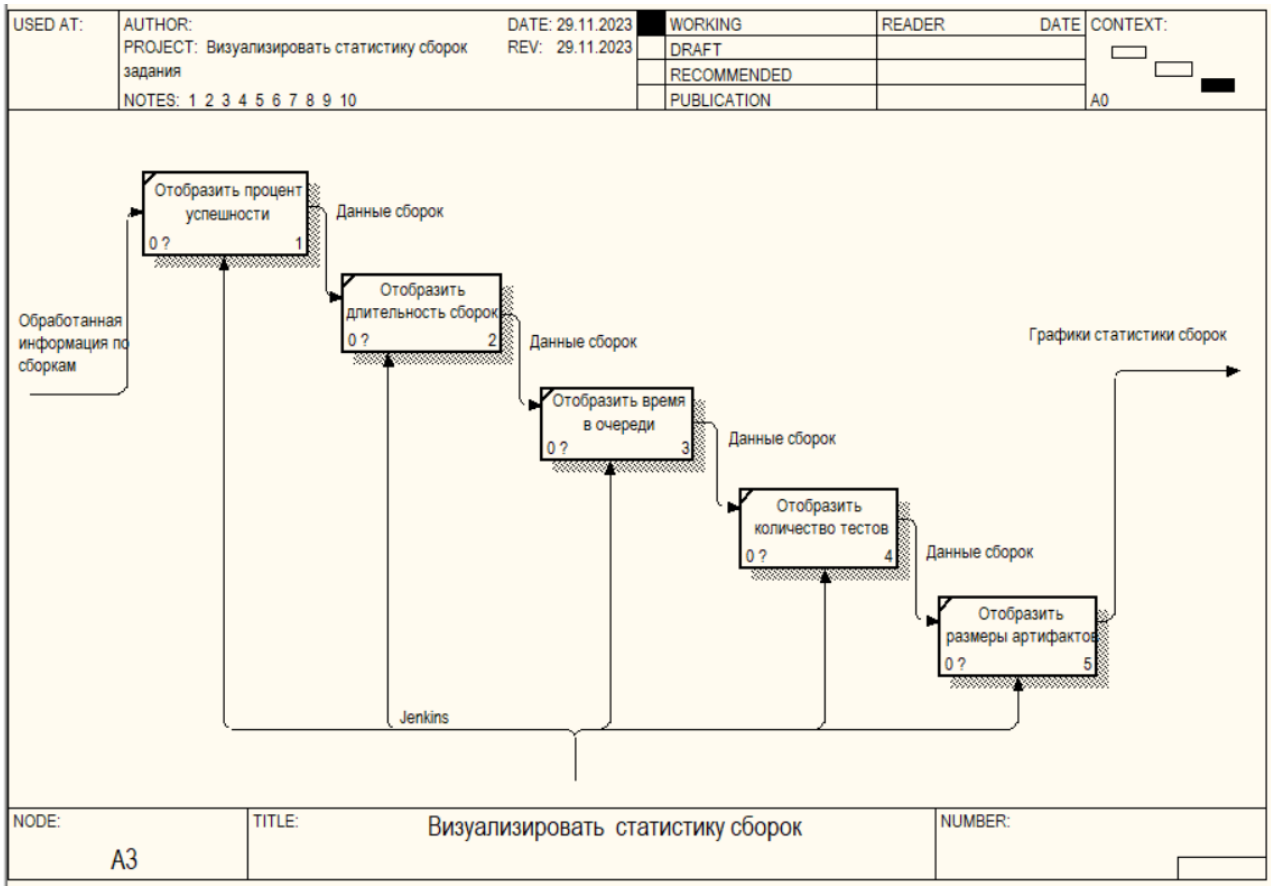

Рис.П2.3. Процесс построения графиков

Приложение 3

Use-case

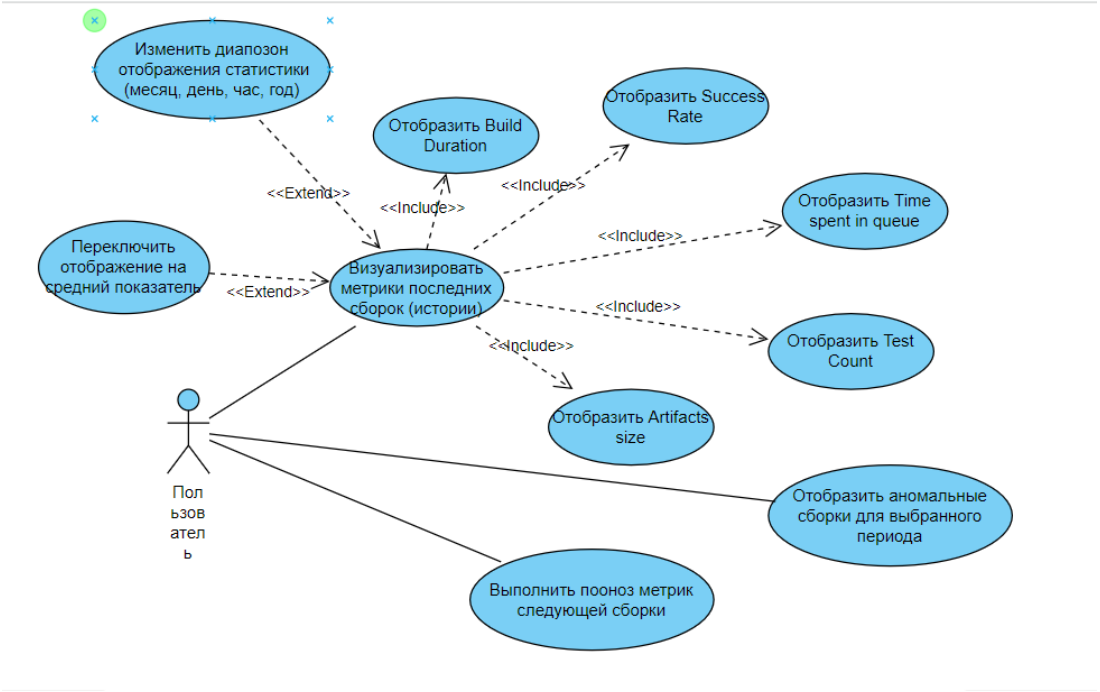

Рис.П3.1. Use case

Приложение 4

Программный код плагина

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

import hudson.model.Action;
import hudson.model.Job;
import jenkins.security.stapler.StaplerDispatchable;
import org.kohsuke.stapler.StaplerRequest;
import org.kohsuke.stapler.StaplerResponse;
import org.kohsuke.stapler.bind.JavaScriptMethod;
import org.kohsuke.stapler.interceptor.RequirePOST;

import javax.servlet.ServletException;
import java.io.IOException;
import java.text.ParseException;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

public class BuildConfigurationStatisticsAction implements
   Action {



    private Job job;

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

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

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

    @Override
    public String getUrlName() {
```

```
        return "buildConfigurationStatistics";
    }

    public Job getJob() {
        return job;
    }

    public Map<String, Double> getBuildDuration(String period,
        String fail, String average) 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, "avg status: " + average);
        IntervalDate intreval = IntervalDate.valueOf(period);
        Boolean failed = fail.equals("1");
        Boolean averageTime = average.equals("1");
        return new BuildDurationLogic(intreval, failed,job.
            getBuilds()).getBuildsDuration(averageTime);
    }

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

    public Map<String, Double> getBuildArtifactSize(String
        period, String fail, String average) throws
        ParseException {
        Logger LOGGER = Logger.getLogger("artifact");
        LOGGER.log(Level.INFO, "arg jelly artifact: " + period
            );
        LOGGER.log(Level.INFO, "failed artifact: " + fail);
        LOGGER.log(Level.INFO, "avg artifact: " + average);
        IntervalDate intreval = IntervalDate.valueOf(period);
        Boolean failed = fail.equals("1");
        Boolean averageTime = average.equals("1");
        return new BuildArtifactSizeLogic(intreval, failed,
            job.getBuilds()).getArtifactSize(averageTime);
    }
```

```
    public Map<String, Integer> 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);
        IntervalDate intreval = IntervalDate.valueOf(period);
        Boolean failed = fail.equals("1");
        return new BuildTestCountLogic(intreval, job.getBuilds
           ()).getTestCount();
    }

    public Map<String, Double> getBuildTimeQueue(String period
       , String average) throws ParseException {
        Logger LOGGER = Logger.getLogger("queue");
        LOGGER.log(Level.INFO, "arg jelly queue: " + period);
        LOGGER.log(Level.INFO, "avg queue: " + average);
        IntervalDate intreval = IntervalDate.valueOf(period);
        Boolean averageTime = average.equals("1");
        return new BuildTimeQueueLogic(intreval, job.getBuilds
           ()).getTimeQueue(averageTime);
    }
    // methods for js -> jelly -> java communication
    @JavaScriptMethod
    public int add(int x, int y) {
        return x+y;
    }

        @JavaScriptMethod
        public Map<String, Double> buildto(String job1) throws
            ParseException {
            return new BuildDurationLogic(IntervalDate.YEAR,
               true, job.getBuilds()).getBuildsDuration(false)
               ;
        }
        public void doAction(StaplerRequest request,
           StaplerResponse response) throws IOException,
           ServletException,
     ParseException {
            IntervalDate intervalDate = IntervalDate.valueOf(
               request.getParameter("IntervalDate"));
            BuildConfigurationStatisticsAction myPluginAction
               = new BuildConfigurationStatisticsAction(job);
            myPluginAction.getBuildDuration("MONTH", "1", "0")
               ;
```

```
            response.sendRedirect2(request.getContextPath() +
               "/jenkins");
        }
    @JavaScriptMethod
    @RequirePOST
    public void mark(String job) {
        Logger LOGGER = Logger.getLogger("uuu2");
        LOGGER.log(Level.WARNING, "js test");
    }

    @StaplerDispatchable
    public void doMark(StaplerRequest req, StaplerResponse res
       ) throws IOException {
        Logger LOGGER = Logger.getLogger("uuu3");
        LOGGER.log(Level.WARNING, "js stepler");
    }
}
```

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


import hudson.Extension;
import hudson.model.Action;
import hudson.model.Job;
import jenkins.model.TransientActionFactory;

import javax.annotation.Nonnull;
import java.util.Collection;
import java.util.Collections;

@Extension
public class BuildConfigurationStatisticsFactory extends
   TransientActionFactory<Job> {
    @Override
    public Class<Job> type() {
        return Job.class;
    }

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

```
}
```

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

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

import java.io.IOException;
import java.text.ParseException;
import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.concurrent.TimeUnit;
import hudson.model.AbstractBuild;

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

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

    public Map<String, Double> getBuildsDuration(Boolean
       average) throws ParseException {
        filterPeriodBuild();
        filterFailedBuild();
        switch (this.period){
            case MONTH:
                dateFormatDuration = DateTimeHandler.
                   createDateMonthMap();
                dateFormatKey = "yyyy-MM-dd";
                break;
            case WEEK:
                dateFormatDuration = DateTimeHandler.
                   createDateWeekMap();
                dateFormatKey = "yyyy-MM-dd";
                break;
            case YEAR:
```

```
                dateFormatDuration = DateTimeHandler.
                    createDateYearMap();
                dateFormatKey = "yyyy-MM";
                break;
            case QUARTER:
                dateFormatDuration = DateTimeHandler.
                    createDateQuarterMap();
                dateFormatKey = "yyyy-MM";
                break;
            case DAY:
                dateFormatDuration = DateTimeHandler.
                    createDateDayMap();
                dateFormatKey = "yyyy-MM-dd HH";
                break;
            case ALL:
                dateFormatDuration = DateTimeHandler.
                    createDateYearMap();
                dateFormatKey = "yyyy-MM";
                break;
        }


        HashMap<String, Integer> dayDurationAverage = new
            HashMap<>();
        for (Run run : this.buildList) {
            String dateFormatKeyAfterCheckPeriod =
                    DateTimeHandler.dateToString(
                            DateTimeHandler.
                                convertLongTimeToDate(
                                    run.getStartTimeInMillis()
                            ), dateFormatKey
                    );
            LOGGER.log(Level.INFO, "
                dateFormatKeyAfterCheckPeriod: " +
                dateFormatKeyAfterCheckPeriod);
            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 {
```

```
                dateFormatDuration.put(
                   dateFormatKeyAfterCheckPeriod,
                   dateFormatDuration.get(
                   dateFormatKeyAfterCheckPeriod) + run.
                   getDuration() / 1000.0);
                dayDurationAverage.put(
                   dateFormatKeyAfterCheckPeriod,
                   dayDurationAverage.get(
                   dateFormatKeyAfterCheckPeriod) + 1);
            }
        }
        if (average) {
            for (Map.Entry<String, Integer> entry :
               dayDurationAverage.entrySet()) {
                LOGGER.log(Level.INFO, "sum time duration: " +
                    dateFormatDuration.get(entry.getKey()));
                LOGGER.log(Level.INFO, "count runs: " + entry.
                   getValue());
                dateFormatDuration.put(entry.getKey(),
                        dateFormatDuration.get(entry.getKey())
                           /entry.getValue()
                );
            }
        }
        LOGGER.log(Level.INFO, "dateFormatDuration: " +
           dateFormatDuration);
        LOGGER.log(Level.INFO, "dayDurationAverage: " +
           dayDurationAverage);
        return dateFormatDuration;
    }
}
```

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

import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.ZonedDateTime;
import java.util.Calendar;
import java.util.Date;
import java.util.HashMap;
import java.util.logging.Level;
import java.util.logging.Logger;

public class DateTimeHandler {
```

```
15
    static Logger LOGGER = Logger.getLogger(DateTimeHandler.
       class.getName());

    public static Date convertLongTimeToDate(long time) {
        Date date = new Date(time);
20      return date;
    }
    /*
     * date format = yyyy MM dd HH:mm:ss
     *
25   * */
    public static long convertDateToLongTime(Date date) throws
        ParseException {
        return date.getTime();
    }

30  public static int getDayOfMonth(Date aDate) throws
       ParseException {
        Calendar cal = Calendar.getInstance();
        cal.setTime(aDate);
        return cal.get(Calendar.DAY_OF_MONTH);
    }
35
    public static int getCurrentMonthDays() {
        Calendar c = Calendar.getInstance();
        return c.getActualMaximum(Calendar.DAY_OF_MONTH);
    }
40
    public static int getLastMonthDays() {
        Calendar c = Calendar.getInstance();
        c.add(Calendar.MONTH, -1);
        return c.getActualMaximum(Calendar.DAY_OF_MONTH);
45  }

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

    public static String dateToString(Date date, String format
       ) {
        DateFormat dateFormat = new SimpleDateFormat(format);
55      String strDate = dateFormat.format(date);
```

```
        return strDate;
    }

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

    /**
     * Create map format {23.12: 0.0, 24.12: 0.0 ...}
     * on 30-31 days
     *
     * **/
    public static HashMap<String, Double> createDateMonthMap()
        {
        ZonedDateTime dateTime = ZonedDateTime.now().
            minusMonths(1);
        Logger LOGGER;
        LOGGER = Logger.getLogger(DateTimeHandler.class.
            getName());
        LOGGER.log(Level.INFO, "dateTime" + dateTime);
        HashMap<String, Double> dayDuration = new HashMap<
            String, Double>();
        int lenMonth = getLastMonthDays();
        LOGGER.log(Level.INFO, "lenMonth: " + lenMonth);
        for (int i = 1; i <= lenMonth; i++) {

            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);
            dayDuration.put(strDate, 0.0);
        }
        LOGGER.log(Level.INFO, "dayDuration: " + dayDuration.
            entrySet());
        return dayDuration;
    }

    /**
     * Create map format {1:00:00 0.0, 2:00:00: 0.0 ...}
```

```
     * on 24 hours
     *
     * **/
    public static HashMap<String, Double> createDateDayMap() {
        ZonedDateTime dateTime = ZonedDateTime.now().
            minusHours(24);
        Logger LOGGER;
        LOGGER = Logger.getLogger(DateTimeHandler.class.
            getName());
        LOGGER.log(Level.INFO, "dateTime days" + dateTime);
        HashMap<String, Double> hourDuration = new HashMap<
            String, Double>();
        int lenDay = 24;
        LOGGER.log(Level.INFO, "lenDay: " + lenDay);
        for (int i = 1; i <= lenDay; i++) {

            DateFormat dateFormat = new SimpleDateFormat("yyyy
                -MM-dd HH");
            String strDate = dateFormat.format(
                    Date.from(dateTime.plusHours(i).toInstant
                        ()).getTime());
            LOGGER.log(Level.INFO, "strdate i: " + i + " - " +
                strDate);
            hourDuration.put(strDate, 0.0);
        }
        LOGGER.log(Level.INFO, "hourDuration: " + hourDuration
            .entrySet());
        return hourDuration;
    }

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

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

            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);
            successFailSuccess.put(strDate, new HashMap(){{
                put("fail", 0);
```

```
                put("success", 0);
            }});
        }
        LOGGER.log(Level.INFO, "successFailSuccess: " +
            successFailSuccess.entrySet());
        return successFailSuccess;
    }

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

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

            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);
            testCount.put(strDate, 0);
        }
        LOGGER.log(Level.INFO, "testCount: " + testCount.
            entrySet());
        return testCount;
    }

    /**
     * Create map format {2/2022: 0.0, 3/2022: 0.0 ...}
     * on 12 month
     *
```

```
     * **/
    public static HashMap<String, Double> createDateYearMap()
       {
195      ZonedDateTime dateTime = ZonedDateTime.now().
            minusYears(1);
         LOGGER.log(Level.INFO, "last year dateTime" + dateTime
            );
         HashMap<String, Double> monthDuration = new HashMap<
            String, Double>();
         int lengthYear = 12; // any year length in month
         LOGGER.log(Level.INFO, "lengthYear: " + lengthYear);
200      for (int i = 1; i <= lengthYear; i++) {

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

             //get dateTime previous year + i = 1...12 and
                getTime, after in strDate=2022-03
205          String strDate = dateFormat.format(
                     Date.from(
                             dateTime.plusMonths(i).toInstant()
                     ).getTime()
             );
210
             LOGGER.log(Level.INFO, "strdate i: " + i + " - " +
                 strDate);
             monthDuration.put(strDate, 0.0);
         }
         LOGGER.log(Level.INFO, "monthDuration: " +
            monthDuration.entrySet());
215      return monthDuration;
    }

    /**
     * Create map format {2/2022: 0.0, 3/2022: 0.0 ...}
220  * on 3 month
     *
     * **/
    public static HashMap<String, Double> createDateQuarterMap
       () {
         ZonedDateTime dateTime = ZonedDateTime.now().
            minusMonths(3);
225      LOGGER.log(Level.INFO, "last quarter dateTime" +
            dateTime);
```

```
        HashMap<String, Double> quarterDuration = new HashMap<
            String, Double>();
        int lengthQuarter = 4; // any year length in month
        LOGGER.log(Level.INFO, "lengthQuarter: " +
            lengthQuarter);
        for (int i = 1; i <= lengthQuarter; i++) {

            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()
            );

            LOGGER.log(Level.INFO, "strdate i: " + i + " - " +
                strDate);
            quarterDuration.put(strDate, 0.0);
        }
        LOGGER.log(Level.INFO, "quarterDuration: " +
            quarterDuration.entrySet());
        return quarterDuration;
    }

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

```
260             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(
                                dateTime.plusDays(i).toInstant()
265                     ).getTime()
                );

                LOGGER.log(Level.INFO, "strdate i: " + i + " - " +
                    strDate);
270             weekDuration.put(strDate, 0.0);
            }
            LOGGER.log(Level.INFO, "weekDuration: " + weekDuration
                .entrySet());
            return weekDuration;
        }
275
        public static HashMap<String, HashMap<String,Integer>>
            createDateYearMapSuccessRate() {
            ZonedDateTime dateTime = ZonedDateTime.now().
                minusYears(1);
            LOGGER.log(Level.INFO, "last year dateTime success" +
                dateTime);
            HashMap<String, HashMap<String,Integer>>
                successFailSuccess = new HashMap();
280         int lenMonth = 12;
            LOGGER.log(Level.INFO, "lenMonth: " + lenMonth);
            for (int i = 1; i <= lenMonth; i++) {

                DateFormat dateFormat = new SimpleDateFormat("yyyy
                    -MM");
285             //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());
                LOGGER.log(Level.INFO, "strdate i: " + i + " - " +
                    strDate);
                successFailSuccess.put(strDate, new HashMap(){{
290                 put("fail", 0);
                    put("success", 0);
                }});
```

```
        }
        LOGGER.log(Level.INFO, "successFailSuccess: " +
           successFailSuccess.entrySet());
        return successFailSuccess;
    }
}
```

Код BuildLogic.java:

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

import hudson.model.Result;
import hudson.model.Run;
import hudson.util.RunList;
import java.text.ParseException;
import java.time.ZonedDateTime;
import java.util.Date;

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

    Boolean failed;

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

```
35                  break;
                case YEAR:
                    Date dateYear = Date.from(ZonedDateTime.now().
                       minusYears(1).toInstant());
                    this.buildList = buildList.filter(run -> {
                        try {
40                          return run.getStartTimeInMillis() >=
                               DateTimeHandler.
                               convertDateToLongTime(dateYear);
                        } catch (ParseException e) {
                            throw new RuntimeException(e);
                        }
                    });
45                  break;
                case DAY:
                    Date dateDay = Date.from(ZonedDateTime.now().
                       minusHours(24).toInstant());
                    this.buildList = buildList.filter(run -> {
                        try {
50                          return run.getStartTimeInMillis() >=
                               DateTimeHandler.
                               convertDateToLongTime(dateDay);
                        } catch (ParseException e) {
                            throw new RuntimeException(e);
                        }
                    });
55                  break;
                case WEEK:
                    Date dateWeek = Date.from(ZonedDateTime.now().
                       minusWeeks(1).toInstant());
                    this.buildList = buildList.filter(run -> {
                        try {
60                          return run.getStartTimeInMillis() >=
                               DateTimeHandler.
                               convertDateToLongTime(dateWeek);
                        } catch (ParseException e) {
                            throw new RuntimeException(e);
                        }
                    });
65                  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);
            });
        }
    }
}
```

Код BuildArtifactSizeLogic.java:

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

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

public class BuildArtifactSizeLogic extends BuildLogic {

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

```
    }

    public Map<String, Double> getArtifactSize(Boolean average
       ) throws ParseException {
        filterPeriodBuild();
        filterFailedBuild();
        switch (this.period){
            case MONTH:
                dateFormatArtifact = DateTimeHandler.
                   createDateMonthMap();
                dateFormatKey = "yyyy-MM-dd";
                break;
            case WEEK:
                dateFormatArtifact = DateTimeHandler.
                   createDateWeekMap();
                dateFormatKey = "yyyy-MM-dd";
                break;
            case YEAR:
                dateFormatArtifact = DateTimeHandler.
                   createDateYearMap();
                dateFormatKey = "yyyy-MM";
                break;
        }

        HashMap<String, Integer> dayArtifactAverage = new
           HashMap<>();
        for (Run run : this.buildList) {
            String dateFormatKeyAfterCheckPeriod =
                    DateTimeHandler.dateToString(
                            DateTimeHandler.
                               convertLongTimeToDate(
                                    run.getStartTimeInMillis()
                            ), dateFormatKey
                    );
            LOGGER.log(Level.INFO, "
               dateFormatKeyAfterCheckPeriod artifact: " +
               dateFormatKeyAfterCheckPeriod);
            LOGGER.log(Level.WARNING, "getArtifacts: " + run.
               getArtifacts());
            List<Run.Artifact> listArtifacts = run.
               getArtifacts();
            double artifactsRunSize = 0;

            for (Run.Artifact artifact : listArtifacts) {
```

```
                artifactsRunSize += artifact.getFileSize()
                    /1024.0;
                LOGGER.log(Level.WARNING, "artifact.
                    getFileSize(): " +artifact.getFileSize());
            }

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

            if (dateFormatArtifact.get(
                dateFormatKeyAfterCheckPeriod) == 0.0) {
                dateFormatArtifact.put(
                    dateFormatKeyAfterCheckPeriod,
                    artifactsRunSize);
                dayArtifactAverage.put(
                    dateFormatKeyAfterCheckPeriod, 1);
            } else {
                dateFormatArtifact.put(
                    dateFormatKeyAfterCheckPeriod,
                    dateFormatArtifact.get(
                    dateFormatKeyAfterCheckPeriod) +
                    artifactsRunSize);
                dayArtifactAverage.put(
                    dateFormatKeyAfterCheckPeriod,
                    dayArtifactAverage.get(
                    dateFormatKeyAfterCheckPeriod) + 1);
            }
        }
        if (average) {
            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()
                );
            }
        }
        LOGGER.log(Level.INFO, "dateFormatArtifact: " +
            dateFormatArtifact);
        LOGGER.log(Level.INFO, "dayArtifactAverage: " +
            dayArtifactAverage);
```

```
        return dateFormatArtifact;
    }
}
```

Код BuildSuccessRateLogic.java:

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

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

public class BuildSuccessRateLogic extends BuildLogic {

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

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

//    public Map<String, Double> getSuccessRate() throws
   ParseException {
//        filterPeriodBuild();
//        Map<String, Double> successRate = new HashMap<String
   , Double>();
//        for (Run run : this.buildList) {
//            String day ="kjk";
//                    //DateTimeHandler.dateToString(
   DateTimeHandler.convertLongTimeToDate(run.
   getStartTimeInMillis()));
//            if (successRate.containsKey(day)) {
//                successRate.put(day, successRate.get(day) +
   run.getDuration() / 1000.0);
//            } else {
//                successRate.put(day, run.getDuration() /
   1000.0);
```

```
//            }
//        }
//        return successRate;
//    }

    public Map<String, Double> getSuccessRate() throws
       ParseException {
        filterPeriodBuild();
        switch (this.period){
            case MONTH:
                successRateOnFormatDate = DateTimeHandler.
                   createDateMonthMapSuccessRate();
                dateFormatKey = "yyyy-MM-dd";
                break;
            case WEEK:
                successRateOnFormatDate = DateTimeHandler.
                   createDateWeekMapSuccessRate();
                dateFormatKey = "yyyy-MM-dd";
                break;
            case YEAR:
                successRateOnFormatDate = DateTimeHandler.
                   createDateYearMapSuccessRate();
                dateFormatKey = "yyyy-MM";
                break;
        }

        for (Run run : this.buildList) {
            String dateFormatKeyAfterCheckPeriod =
                    DateTimeHandler.dateToString(
                            DateTimeHandler.
                               convertLongTimeToDate(
                                    run.getStartTimeInMillis()
                            ), dateFormatKey
                    );
            LOGGER.log(Level.INFO, "
               dateFormatKeyAfterCheckPeriod: " +
               dateFormatKeyAfterCheckPeriod);
            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);
            }
            LOGGER.log(Level.INFO, "successRateOnFormatDate: "
                + successRateOnFormatDate);

//            if (successRateOnFormatDate.get(
dateFormatKeyAfterCheckPeriod).get("fail") == 0.0) {
//                successRateOnFormatDate.put(
dateFormatKeyAfterCheckPeriod,
//                        //(run.getResult().isBetterOrEqualTo
(Result.SUCCESS)) ? );
//
//            } else {
//                successRateOnFormatDate.put(
dateFormatKeyAfterCheckPeriod, successRateOnFormatDate.get(
dateFormatKeyAfterCheckPeriod) + run.getDuration() /
1000.0);
//            }
        }
        HashMap<String, Double> successRateMap = new HashMap<
            String, Double>();
        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"))
                     );
                }
        }
        LOGGER.log(Level.INFO, "successRateMap: " +
            successRateMap);
```

```
        return successRateMap;
    }
}
```

Код BuildTestCountLogic.java:

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

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

public class BuildTestCountLogic extends BuildLogic {

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

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

    public Map<String, Integer> getTestCount() throws
       ParseException {
        filterPeriodBuild();
        filterPeriodBuild();
        switch (this.period) {
            case MONTH:
                testCountOnFormatDate = DateTimeHandler.
                   createDateMonthMapTestCount();
                dateFormatKey = "yyyy-MM-dd";
                break;
            case WEEK:
                testCountOnFormatDate = DateTimeHandler.
                   createDateMonthMapTestCount();
                dateFormatKey = "yyyy-MM-dd";
                break;
```

```
            case YEAR:
                testCountOnFormatDate = DateTimeHandler.
                   createDateMonthMapTestCount();
                dateFormatKey = "yyyy-MM";
                break;
        }

        for (Run run : this.buildList) {
            String dateFormatKeyAfterCheckPeriod =
                    DateTimeHandler.dateToString(
                            DateTimeHandler.
                               convertLongTimeToDate(
                                    run.getStartTimeInMillis()
                            ), dateFormatKey
                    );
            LOGGER.log(Level.INFO, "
               dateFormatKeyAfterCheckPeriod: " +
               dateFormatKeyAfterCheckPeriod);
            if (testCountOnFormatDate.get(
               dateFormatKeyAfterCheckPeriod) == 0) {
                testCountOnFormatDate.put(
                   dateFormatKeyAfterCheckPeriod,
                   getTestCountForRun(run));
            } else {
                testCountOnFormatDate.put(
                   dateFormatKeyAfterCheckPeriod,
                   testCountOnFormatDate.get(
                   dateFormatKeyAfterCheckPeriod) +
                   getTestCountForRun(run));
            }
            LOGGER.log(Level.INFO, "testCountOnFormatDate: " +
                testCountOnFormatDate);

        }

        LOGGER.log(Level.INFO, "testCountMap: " +
           testCountOnFormatDate);

        return testCountOnFormatDate;
    }


    public int getTestCountForRun(Run run) {
        int testCount = 0;
//          Jenkins jenkinsInstance = Jenkins.getInstance();
```

```
//        if (jenkinsInstance != null) {
//            Job job = (Job) jenkinsInstance.getItem(jobName)
   ;
//            if (job != null && job instanceof
   AbstractProject) {
//                AbstractProject abstractProject = (
   AbstractProject) job;
//                AbstractTestResultAction
   abstractTestResultAction = abstractProject.
   getLastCompletedBuild().getAction(AbstractTestResultAction.
   class);
//                if (abstractTestResultAction != null) {
//                    return abstractTestResultAction.
   getTotalCount();
//                }
//            }
//        }
        return testCount;
    }
}
```

Код BuildTimeQueueLogic.java:

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

import hudson.model.Run;
import hudson.util.RunList;
import java.text.ParseException;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.TimeUnit;
import java.util.logging.Level;
import java.util.logging.Logger;

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

    HashMap<String, Double> dateFormatDuration;
    String dateFormatKey;
    public BuildTimeQueueLogic(IntervalDate period, RunList<
       Run> buildList) {
        super(period, true, buildList);
    }
```

```
    public Map<String, Double> getTimeQueue(Boolean average)
       throws ParseException {
        filterPeriodBuild();

        switch (this.period){
            case MONTH:
                dateFormatDuration = DateTimeHandler.
                   createDateMonthMap();
                dateFormatKey = "yyyy-MM-dd";
                break;
            case WEEK:
                dateFormatDuration = DateTimeHandler.
                   createDateWeekMap();
                dateFormatKey = "yyyy-MM-dd";
                break;
            case YEAR:
                dateFormatDuration = DateTimeHandler.
                   createDateYearMap();
                dateFormatKey = "yyyy-MM";
                break;
        }


        HashMap<String, Integer> dayDurationAverage = new
           HashMap<>();
        for (Run run : this.buildList) {
            String dateFormatKeyAfterCheckPeriod =
                    DateTimeHandler.dateToString(
                            DateTimeHandler.
                               convertLongTimeToDate(
                                    run.getStartTimeInMillis()
                            ), dateFormatKey
                    );
            LOGGER.log(Level.INFO, "
               dateFormatKeyAfterCheckPeriod time Queue: " +
               dateFormatKeyAfterCheckPeriod);
            long runTimeInQueue = new TimeInQueueFetcher().
               getTimeInQueue(run);
            if (dateFormatDuration.get(
               dateFormatKeyAfterCheckPeriod) == 0.0) {

                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 {
                dateFormatDuration.put(
                   dateFormatKeyAfterCheckPeriod,
                   dateFormatDuration.get(
                   dateFormatKeyAfterCheckPeriod) + (double)
                   runTimeInQueue);
                dayDurationAverage.put(
                   dateFormatKeyAfterCheckPeriod,
                   dayDurationAverage.get(
                   dateFormatKeyAfterCheckPeriod) + 1);
            }
        }
        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()
                );
            }
        }
        LOGGER.log(Level.INFO, "dateFormatDuration time queue:
            " + dateFormatDuration);
        LOGGER.log(Level.INFO, "dayDurationAverage time queue:
            " + dayDurationAverage);
        return dateFormatDuration;
    }
}
```

Код TimeInQueueFetcher.java:

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

import hudson.model.Run;
```

```
import java.util.concurrent.TimeUnit;

public class TimeInQueueFetcher {
    public long getTimeInQueue(Run build) {
        long queuedTime = build.getStartTimeInMillis() - build
           .getTimeInMillis();
        return TimeUnit.MILLISECONDS.toMillis(queuedTime);
    }
}
```

Код dateIntervalEnum.java.java:

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

enum IntervalDate {
    DAY,
    WEEK,
    MONTH,
    YEAR,
    QUARTER,
    ALL
}
```

Код js файла с парсингом данных и визуализацией:

```


function formatLabelsDate(arrLabels, dateFormat, period) {
switch(period) {
    case 'WEEK':
    case 'MONTH':
                arrLabels.push(
                          dateFormat.getDate()+
                                    "/"+(dateFormat.
                                       getMonth()+1)+
                                    "/"+dateFormat.
                                       getFullYear()
                        );
    break;

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

```
}
}


function sortOnKeys(dict) {

    var sorted = [];
    for(var key in dict) {
        sorted[sorted.length] = key;
    }
    sorted.sort();

    var dataBuildDurationValues = [];
    var labelsB = [];
    for(var i = 0; i < sorted.length; i++) {
       dataBuildDurationValues.push(dict[sorted[i]]);
       var dateFormat= new Date(parseInt(sorted[i]));
       console.log("dateFormat", dateFormat);
       var period = document.querySelector(".period").
          textContent;
       formatLabelsDate(labelsB, dateFormat, period);
    }
    return [dataBuildDurationValues, labelsB];
}
// build Duration create chart settings
var buildDuration = document.querySelectorAll(".buildDuration
   ");

var dataBuildDurationValues = [];
var dataBuildDurationDict = {};

for (var i=0; i<buildDuration.length; i++){

    dataBuildDurationDict[Date.parse(buildDuration[i].
       querySelector('.key').textContent)]
        = parseFloat(buildDuration[i].querySelector('.value').
           textContent);

}
console.log("dataBuildDurationDict: ", dataBuildDurationDict);

dict = sortOnKeys(dataBuildDurationDict)[0];
labelsB = sortOnKeys(dataBuildDurationDict)[1];
```

```
console.log("build duration dict values",dict);

// build Duration create chart settings
var successRate = document.querySelectorAll(".successRate");

var dataSuccessRateValues = [];
var dataSuccessRateDict = {};

for (var i=0; i<successRate.length; i++){

    dataSuccessRateDict[Date.parse(successRate[i].
       querySelector('.key').textContent)]
        = parseFloat(successRate[i].querySelector('.value').
           textContent);

}
console.log("dataSuccessRateDict: ", dataSuccessRateDict);

dictSuccess = sortOnKeys(dataSuccessRateDict)[0];
labelsSuccess = sortOnKeys(dataSuccessRateDict)[1];
console.log("success rate dict values",dictSuccess);


// time Queue create chart settings
var timeQueue = document.querySelectorAll(".timeQueue");

var dataTimeQueueValues = [];
var dataTimeQueueDict = {};

for (var i=0; i<timeQueue.length; i++){

    dataTimeQueueDict[Date.parse(timeQueue[i].querySelector('.
       key').textContent)]
        = parseFloat(timeQueue[i].querySelector('.value').
           textContent);

}
console.log("dataTimeQueueDict: ", dataTimeQueueDict);

dictTimeQueue = sortOnKeys(dataTimeQueueDict)[0];
labelsTimeQueue = sortOnKeys(dataTimeQueueDict)[1];
console.log("time spent queue dict values",dictTimeQueue);

// artifact create chart settings
var artifactSize = document.querySelectorAll(".artifactSize");
```

```
var dataArtifactSizeValues = [];
var dataArtifactSizeDict = {};

for (var i=0; i<artifactSize.length; i++){

    dataArtifactSizeDict[Date.parse(artifactSize[i].
       querySelector('.key').textContent)]
        = parseFloat(artifactSize[i].querySelector('.value').
           textContent);

}
console.log("dataArtifactSizeDict: ", dataArtifactSizeDict);

dictArtifactSize = sortOnKeys(dataArtifactSizeDict)[0];
labelsArtifactSize = sortOnKeys(dataArtifactSizeDict)[1];
console.log("artifact size dict values",dictArtifactSize);



const labels = Array.from({length: 30}, (_, i) => i + 1);

const data = {
  labels: labelsSuccess,
  datasets: [{
    label: 'Success rate',
    data: dictSuccess,
    backgroundColor: [
      'rgba(0, 255, 0, 0.5)',
    ],
    borderColor: [
      'rgb(0, 69, 36)',
    ],
    categoryPercentage: 1,
    borderWidth: 1,
    barPercentage: 1,
  }]
};


const dataBuildDuration = {
  labels: labelsB,
  datasets: [{
    label: 'Build duration',
```

```
    data: dict,
    borderColor: [
      'rgba(0, 180, 33, 1)',
    ],
    tension: 0.1

  }]
};

const dataTimeSpentQueue = {
  labels: labelsB,
  datasets: [{
    label: 'Time Spent In Queue',
    data: dictTimeQueue,
    borderColor: [
      'rgba(0, 180, 33, 1)',
    ],
    tension: 0.1

  }]
};
const dataTestCount = {
  labels: labels,
  datasets: [{
    label: 'Test Count',
    data: Array.from({length: 30}, () => Math.floor(Math.
       random() * 60)),
    borderColor: [
      'rgba(0, 180, 33, 1)',
    ],
    tension: 0.1

  }]
};
const dataArtifactsSize = {
  labels: labelsArtifactSize,
  datasets: [{
    label: 'Artifacts Size',
    data: dictArtifactSize,
    borderColor: [
      'rgba(0, 180, 33, 1)',
    ],
    tension: 0.1

  }]
```

```
};
```

```
var allPerf = {
                        type: 'bar',
                        data: data,
                        options: {
                            scales: {
                                  y: {
                                    beginAtZero: true
                                  }
                                }
                        }
                      };

var settingsBuildDuration = {
                        type: 'line',
                        data: dataBuildDuration,
                      };
var settingsTimeSpentQueue = {
                        type: 'line',
                        data: dataTimeSpentQueue,
                      };
var settingsTestCount = {
                        type: 'line',
                        data: dataTestCount,
                      };
var settingsArtifactsSize = {
                        type: 'line',
                        data: dataArtifactsSize,
                      };
var ctx = document.getElementById("successRateChart").
   getContext("2d");
var ctxBuild = document.getElementById("buildDurationChart").
   getContext("2d");
var ctxTimeSpentQueue = document.getElementById("
   timeSpentQueue").getContext("2d");
```

```
var ctxTestCount = document.getElementById("testCount").
   getContext("2d");
var ctxArtifactsSize = document.getElementById("artifactsSize
   ").getContext("2d");
perfChartJsCharts["successRateChart"] = new Chart(ctx, allPerf
   );
perfChartJsCharts["buildDurationChart"] = new Chart(ctxBuild,
   settingsBuildDuration);
perfChartJsCharts["timeSpentQueue"] = new Chart(
   ctxTimeSpentQueue, settingsTimeSpentQueue);
perfChartJsCharts["testCount"] = new Chart(ctxTestCount,
   settingsTestCount);
perfChartJsCharts["artifactsSize"] = new Chart(
   ctxArtifactsSize, settingsArtifactsSize);
```

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>
            <style>
                  .buildDuration, .period,
                  .successRate, .timeQueue,
                  .artifactSize, .testCount{
                        display:none;
                  }
                  .graph-container {
                  width: 90%;

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

                  }
                  .canvas-container{
                  width: 80%;

                  }
                  .settings{
                  padding: 5px;

                  margin: 10px;
```

```
                        display: flex;
                        flex-direction: column;

                        }
                        form label{
                        margin: 5px;
                        }

                </style>
        </head>

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

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

                        <h1>Statistics for job ${it.job.name
                           }</h1>
                        <div id="msg" />
<!--                        <script>-->
<!--                                var foo = <st:bind value="
   ${it}"/>-->

<!--                                foo.add(1,5, function(t)
   {-->
<!--                                document.getElementById('
   msg').innerHTML = t.responseObject();-->
<!--                                })-->
<!--                        </script>-->

<!--                        <td>-->
<!--                                <f:checkbox name="selected
   " onclick="myItem.mark('${it.job.fullName}')" />-->
<!--                        </td>-->

                        <p class="period">WEEK</p>
                        <j:forEach var="type" items="${it.
                           getBuildDuration('WEEK', '1','1')
                           }">
                                <p class="buildDuration">
```

```
                    <span class="key">${
                        type.key}</span>
                    <span class="value">${
                        type.value}</span>
                </p>
            </j:forEach>
            <j:forEach var="successRate" items="${
                it.getBuildSuccessRate('WEEK')}">
                <p class="successRate">
                    <span class="key">${
                        successRate.key}</
                        span>
                    <span class="value">${
                        successRate.value
                        }</span>
                </p>
            </j:forEach>
            <j:forEach var="timeQueue" items="${it
                .getBuildTimeQueue('WEEK', '1')}">
                <p class="timeQueue">
                    <span class="key">${
                        timeQueue.key}</
                        span>
                    <span class="value">${
                        timeQueue.value}</
                        span>
                </p>
            </j:forEach>
            <j:forEach var="artifactSize" items="$
                {it.getBuildArtifactSize('WEEK',
                '1','1')}">
                <p class="artifactSize">
                    <span class="key">${
                        artifactSize.key}</
                        span>
                    <span class="value">${
                        artifactSize.value
                        }</span>
                </p>
            </j:forEach>
            <j:forEach var="testCount" items="${it
                .getBuildTestCount('WEEK', '1')}">
                <p class="testCount">
```

```
                        <span class="key">${
                            testCount.key}</
                            span>
                        <span class="value">${
                            testCount.value}</
                            span>
                    </p>
                </j:forEach>

                <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>
                            Range:
                            <select>
                                <
                                    option
                                    >
                                    Month
                                    </
                                    option
                                    >
                                <
                                    option
                                    >
                                    Day
                                    </
                                    option
                                    >
                                <
                                    option
                                    >
                                    Year
                                    </
                                    option
                                    >
                                <
                                    option
                                    >
```

```
                                                    Week
                                                    </
                                                    option
                                                    >
                                                <
                                                    option
                                                    >
                                                    Quarter
                                                    </
                                                    option
                                                    >
110                                             <
                                                    option
                                                    >
                                                    All
                                                    </
                                                    option
                                                    >
                                            </select>

                                    </label>

115                         </form>
                    </div>
                    <div class="graph-block">
                            <div class="canvas-container">
                            <canvas id="buildDurationChart
                               " width="90" height="25"></
                               canvas>
120                         </div>
                            <form class="settings">
                                    <label>
                                            Range:
                                            <f:entry title
                                               ="My Plugin
                                                Action">
125                                         <select>
                                                    <
                                                    option
                                                    >
                                                    Month
                                                    </
                                                    option
                                                    >
```

```
                                        <
                                           option
                                           >
                                           Day
                                           </
                                           option
                                           >
                                        <
                                           option
                                           >
                                           Year
                                           </
                                           option
                                           >
                                        <
                                           option
                                           >
                                           Week
                                           </
                                           option
                                           >
                                        <
                                           option
                                           >
                                           Quarter
                                           </
                                           option
                                           >
                                        <
                                           option
                                           >
                                           All
                                           </
                                           option
                                           >
                                  </select>
                                  </f:entry>

                           </label>
                           <label>
                           Average:
                           <input type="checkbox
                              "/>
                           </label>
```

```
                        <label>
                                Show failed:
                                <input type="
                                   checkbox"/>
                        </label>
145
                </form>
        </div>
        <div class="graph-block">
                <div class="canvas-container">
150             <canvas id="timeSpentQueue"
                   width="90" height="25"></
                   canvas>
                </div>
                <form class="settings">
                        <label>
                                Range:
155                             <select>
                                        <
                                           option
                                           >
                                           Month
                                           </
                                           option
                                           >
                                        <
                                           option
                                           >
                                           Day
                                           </
                                           option
                                           >
                                        <
                                           option
                                           >
                                           Year
                                           </
                                           option
                                           >
                                        <
                                           option
                                           >
                                           Week
                                           </
```

```
                                                    option
                                                    >
160                                                 <
                                                    option
                                                    >
                                                    Quarter
                                                    </
                                                    option
                                                    >
                                                    <
                                                    option
                                                    >
                                                    All
                                                    </
                                                    option
                                                    >
                                            </select>

                                    </label>
165                                 <label>
                                            Average:
                                            <input type="
                                               checkbox"/>
                                    </label>

170                         </form>
                    </div>
                    <div class="graph-block">
                            <div class="canvas-container">
                            <canvas id="testCount" width
                               ="90" height="25"></canvas>
175                         </div>
                            <form class="settings">
                                    <label>
                                            Range:
                                            <select>
180                                                 <
                                                    option
                                                    >
                                                    Month
                                                    </
                                                    option
                                                    >
                                                    <
                                                    option
```

```
                                                    >
                                                    Day
                                                    </
                                                    option
                                                    >
                                                <
                                                    option
                                                    >
                                                    Year
                                                    </
                                                    option
                                                    >
                                                <
                                                    option
                                                    >
                                                    Week
                                                    </
                                                    option
                                                    >
                                                <
                                                    option
                                                    >
                                                    Quarter
                                                    </
                                                    option
                                                    >
                                                <
                                                    option
                                                    >
                                                    All
                                                    </
                                                    option
                                                    >
                                        </select>

                                </label>
                                <label>
                                        Show failed:
                                        <input type="
                                           checkbox"/>
                                </label>

                        </form>
                </div>
                <div class="graph-block">
```

```
<div class="canvas-container">
<canvas id="artifactsSize"
   width="90" height="25"></
   canvas>
</div>
<form class="settings">
        <label>
                Range:
                <select>
                        <
                           option
                           >
                           Month
                           </
                           option
                           >
                        <
                           option
                           >
                           Day
                           </
                           option
                           >
                        <
                           option
                           >
                           Year
                           </
                           option
                           >
                        <
                           option
                           >
                           Week
                           </
                           option
                           >
                        <
                           option
                           >
                           Quarter
                           </
                           option
                           >
```

```
                                                    <
                                                        option
                                                        >
                                                        All
                                                        </
                                                        option
                                                        >
                                                </select>

                                        </label>
                                        <label>
                                                Average:
                                                <input type="
                                                    checkbox"/>
                                        </label>
                                        <label>
                                                Show failed:
                                                <input type="
                                                    checkbox"/>
                                        </label>

                                </form>
                        </div>
                        </div>

                </l:main-panel>
                <st:bind var="myItem" value="${it}"/>
        </l:layout>
        <st:adjunct includes="io.jenkins.plugins.sample.
            BuildConfigurationStatisticsAction.
            declareChartJsClickArray"/>
        <st:adjunct includes="io.jenkins.plugins.sample.
            BuildConfigurationStatisticsAction.chartLogicBox"/>
</j:jelly>
```

Приложение 5

Программный код unit тестов на языке Java

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

import hudson.model.FreeStyleBuild;
import hudson.model.FreeStyleProject;
import hudson.model.Result;
import hudson.tasks.Shell;
import org.junit.Rule;
import org.junit.Test;
import org.jvnet.hudson.test.JenkinsRule;

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

public class BuildConfigurationStatisticsBuilderTest {

    @Rule
    public JenkinsRule jenkins = new JenkinsRule();

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

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

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

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

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

    @Test
    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.
            convertLongTimeToDate(time);
        assert resultDate.equals(date);
    }
    @Test
    public void testGetDayOfMonth() throws ParseException {
        DateFormat dateFormat = new SimpleDateFormat("dd/MM/
            yyyy");
        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;
        assert daysDate2 == 29;
    }

    @Test
    public void testGetCurrentMonthDays()  {
        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, Double> dictDateMonthZero =
            DateTimeHandler.createDateMonthMap();
        assert  dictDateMonthZero.size() == daysDate;
        assert  !dictDateMonthZero.isEmpty();
```

```
        for (Map.Entry<String, Double> entry :
           dictDateMonthZero.entrySet()) {
            assert entry.getValue() == 0.0;
        }
    }

    @Test
    public void testCreateDateWeekMapSuccessRate()  {

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

    @Test
    public void testCreateDateMonthMapSuccessRate()  {
        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()) {
            assert entry.getValue().equals(new HashMap(){{
                put("fail", 0);
                put("success", 0);
            }});
        }
    }

    @Test
    public void testCreateDateMonthMapTestCount()  {
        int daysDate = DateTimeHandler.getLastMonthDays();
        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;
        }
    }

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

    @Test
    public void testCreateDateYearMap()  {
        HashMap<String, Double> dictDateYearZero =
           DateTimeHandler.createDateYearMap();
        assert  dictDateYearZero.size() == 12;
        assert  !dictDateYearZero.isEmpty();
        for (Map.Entry<String, Double> entry :
           dictDateYearZero.entrySet()) {
            assert entry.getValue() == 0.0;
        }
    }

    @Test
    public void testCreateDateWeekMap()  {
        HashMap<String, Double> dictDateWeekZero =
           DateTimeHandler.createDateWeekMap();
        assert  dictDateWeekZero.size() == 7;
        assert  !dictDateWeekZero.isEmpty();
        for (Map.Entry<String, Double> entry :
           dictDateWeekZero.entrySet()) {
            assert entry.getValue() == 0.0;
        }
```

```
    }

    @Test
    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()) {
            assert entry.getValue().equals(new HashMap(){{
                put("fail", 0);
                put("success", 0);
            }});
        }
    }

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

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

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

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

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

        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());
230         jenkins.buildAndAssertSuccess(project);
        project.getBuildersList().add(new Shell("echo1 hello")
            );
        jenkins.buildAndAssertStatus(Result.FAILURE, project);
        List<Run> runList = new RunList<>(project);
        for (Run run :runList) {
235          System.out.println(run.getResult());
        }

        BuildLogic instance1 = new BuildLogic(IntervalDate.
            WEEK, false, (RunList<Run>) runList);
        instance1.filterFailedBuild();
240
        assert  instance1.buildList.size() == 1;

    }

245 @Test
    public void testCreateDateQuarterMap()  {
        HashMap<String, Double> dictDateQuarterZero =
            DateTimeHandler.createDateQuarterMap();
        assert  dictDateQuarterZero.size() == 4;
        assert  !dictDateQuarterZero.isEmpty();
250         for (Map.Entry<String, Double> entry :
            dictDateQuarterZero.entrySet()) {
            assert entry.getValue() == 0.0;
        }
    }

255 @Test
    public void testCreateDateDayMap()  {
```

```
        HashMap<String, Double> dictDateDayZero =
           DateTimeHandler.createDateDayMap();
        assert  dictDateDayZero.size() == 24;
        assert  !dictDateDayZero.isEmpty();
        for (Map.Entry<String, Double> entry : dictDateDayZero
           .entrySet()) {
            System.out.println(entry.getKey());
            assert entry.getValue() == 0.0;
        }
    }
}
```

Приложение 6

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
from selenium.webdriver.support.wait import WebDriverWait
import random
import string
from selenium.webdriver.common.action_chains import
    ActionChains


class Base:
    driver = None

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

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

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

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

    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 конфигурационно 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
from webdriver_manager.chrome import ChromeDriverManager
from webdriver_manager.firefox import GeckoDriverManager


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


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


@pytest.fixture()
def driver(config):
    browser = config['browser']
    url = config['url']
    if browser == 'chrome':
        driver = webdriver.Chrome(service=Service(
           ChromeDriverManager().install()))
    elif browser == 'firefox':
        driver = webdriver.Firefox(service=Service(
           GeckoDriverManager().install()))
    else:
        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')]")
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")

SUCCESS_RATE_SELECT_DAY = (By.XPATH, "//canvas[@id='
    successRateChart']/../../form//select//option[text()='Day']
    ")



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]")

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")
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']"
    )
```

```
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]")

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")
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
import locators
import pytest


class TestCase(Base):
    @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
        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):
        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()
        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()
        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()
        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()

        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()
```