**Пример ВКР «SPbPU-student-thesis-template». Версия от 30 января 2024 г.**

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

Работа допущена к защите
Руководитель ОП
_____ А.В. Щукин
«_____» _____ 2023 г.

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

по направлению подготовки 09.03.03 Прикладная информатика
Направленность (профиль) 09.03.03_YY Наименование направленности (профиля) образовательной программы

Выполнил
студент гр. з5130903/90301 А.Д. Кухто

Руководитель
ст. преподователь ВШПИ,
звание[1] В.А. Пархоменко

Консультант
по нормоконтролю[2] В.А. Пархоменко

Санкт-Петербург
2023

[1]Должность указывают сокращенно, учёную степень и звание — при наличии, а подразделения — аббревиатурами. «СПбПУ» и аббревиатуры институтов не добавляют.

[2]Обязателен, из числа ППС по решению руководителя ОП или подразделения. Должность и степень не указываются. Сведения помещаются в последнюю строчку по порядку. Рецензенты не указываются.

**САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО**

**Институт компьютерных наук и кибербезопасности**

УТВЕРЖДАЮ
Руководитель ОП
_____ А.В. Щукин
«_____» _____

# ЗАДАНИЕ
## на выполнение выпускной квалификационной работы

студенту Кухто Андрею Денисовичу гр. з5130903/90301

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

6.1. Ст. преподователь ВШПИ, В.А. Пархоменко (нормоконтроль).

7. Дата выдачи задания: 02.10.2023.

Руководитель ВКР _____ В.А. Пархоменко

Задание принял к исполнению 02.10.2023

Студент _____ А.Д. Кухто

## РЕФЕРАТ

На 53 с., 15 рисунков, 4 таблицы, 2 приложения

КЛЮЧЕВЫЕ СЛОВА: СТИЛЕВОЕ ОФОРМЛЕНИЕ САЙТА, УПРАВЛЕНИЕ КОНТЕНТОМ, PHP, MYSQL, АРХИТЕКТУРА СИСТЕМЫ.[3]

Тема выпускной квалификационной работы: «Разработка плагина Jenkins для визуализации статистики работы сборок Jenkins»[4].

В данной работе изложена сущность подхода к созданию динамического информационного портала на основе использования открытых технологий Apache, MySQL и PHP. Даны общие понятия и классификация IT-систем такого класса. Проведен анализ систем-прототипов. Изучена технология создания указанного класса информационных систем. Разработана конкретная программная реализация динамического информационного портала на примере портала выбранной тематики...[5]

В данной работе изложена сущность подхода к созданию динамического информационного портала на основе использования открытых технологий Apache, MySQL и PHP. Даны общие понятия и классификация IT-систем такого класса. Проведен анализ систем-прототипов. Изучена технология создания указанного класса информационных систем. Разработана конкретная программная реализация динамического информационного портала на примере портала выбранной тематики...

## ABSTRACT

53 pages, 15 figures, 4 tables, 2 appendices

KEYWORDS: STYLE REGISTRATION, CONTENT MANAGEMENT, PHP, MYSQL, SYSTEM ARCHITECTURE.

---

[3]Всего **слов**: от 3 до 15. Всего **слов и словосочетаний**: от 3 до 5. Оформляются в именительном падеже множественного числа (или в единственном числе, если нет другой формы), оформленных по правилам русского языка. *Внимание! Размещение сноски после точки является примером как запрещено оформлять сноски.*

[4]Реферат **должен содержать**: предмет, тему, цель ВКР; метод или методологию проведения ВКР: результаты ВКР: область применения результатов ВКР; выводы.

[5]ОТ 1000 ДО 1500 печатных знаков (ГОСТ Р 7.0.99-2018 СИБИД) на русский или английский текст. Текст реферата повторён дважды на русском и английском языке для демонстрации подхода к нумерации страниц.

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

In the given work the essence of the approach to creation of a dynamic information portal on the basis of use of open technologies Apache, MySQL and PHP is stated. The general concepts and classification of IT-systems of such class are given. The analysis of systems-prototypes is lead. The technology of creation of the specified class of information systems is investigated. Concrete program realization of a dynamic information portal on an example of a portal of the chosen subjects is developed...

In the given work the essence of the approach to creation of a dynamic information portal on the basis of use of open technologies Apache, MySQL and PHP is stated. The general concepts and classification of IT-systems of such class are given. The analysis of systems-prototypes is lead. The technology of creation of the specified class of information systems is investigated. Concrete program realization of a dynamic information portal on an example of a portal of the chosen subjects is developed...

# СОДЕРЖАНИЕ

Введение ..... 8
Глава 1. Анализ средств сборки и визуализации программного обеспечения 10
1.1. Обзор и сравнительный анализ средств сборки программного обеспечения ..... 10
1.2. Особенности и сравнительный анализ CI/CD систем ..... 11
1.3. Обзор и сравнительный анализ средств CI/CD ..... 12
1.4. Статистика и визуализация работы сборок в контексте CI/CD ..... 15
1.5. Обзор и сравнительный анализ плагинов Jenkins по визуализации статистики сборок ..... 15
1.6. Требования к разработке ..... 18
1.7. Выводы ..... 19
Глава 2. Проектирование архитектуры плагина ..... 19
2.1. Модель системы ..... 20
2.2. Архитектура Jenkins ..... 20
2.3. Архитектура плагина ..... 22
2.4. Языки программирования ..... 23
2.5. Инструменты сборки ..... 24
2.6. Библиотеки ..... 24
2.7. Выводы ..... 25
Глава 3. Реализация прототипа плагина ..... 25
3.1. Описание разработанных классов ..... 25
3.1.1. BuildConfigurationStatisticsAction ..... 25
3.1.2. DateTimeHandler ..... 26
3.1.3. IntervalDate ..... 29
3.1.4. Statistics ..... 29
3.1.5. TimeInQueueFetcher ..... 30
3.1.6. BuildLogic ..... 30
3.1.7. BuildArtifactSizeLogic ..... 30
3.1.8. BuildDurationLogic ..... 31
3.1.9. BuildSuccessRateLogic ..... 32
3.1.10. BuildTestCountLogic ..... 32
3.1.11. BuildTimeQueueLogic ..... 33
3.1.12. Файлы JS и Jelly ..... 34
3.2. Результаты разработки плагина ..... 35
3.3. Выводы ..... 36

Глава 4. Тестирование и апробация плагина в Jenkins ..... 37
4.1. Методы тестирования ..... 37
4.1.1. Unit тестирование..... 38
4.1.2. UI тестирование ..... 40
4.2. Апробация плагина ..... 41
4.2.1. Подготовка и генерация набора сборок для апробации..... 41
4.2.2. Апробация на проекте с открытым исходным кодом..... 45
4.2.3. Оценка результатов работы ..... 47
4.3. Выводы ..... 48
Заключение ..... 50
Список использованных источников..... 51
Приложение 1. UML диаграмма классов плагина..... 54
Приложение 2. Idef0 ..... 55

# 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 [11].

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

Одним из лучших средств CI, 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 [29]:

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

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

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

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

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

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

Также необходимо отметить, что в 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 [17]. Данная технология стала неотъемлемой составляющей DevOps методологии и помогает сократить временные ресурсные затраты в процессе современного жизненного цикла приложения, когда до заказчика изначально доходит минимально жизнеспособный продукт (MVP), а затем дорабатывается с учетом новых требований заказчика, т.е. идет непрерывная разработка новых версий продукта.

Преимущества CI/CD подхода [7]:

- упрощение разработки - позволяет разработчикам распределять приоритеты и сконцентрироваться на самых важных аспектах;
- улучшение качества кода - качество кода проверяется до того, как он достигнет среды тестирования, проблемы в коде могут быть выявлены на ранних стадиях;
- более короткие циклы тестирования - меньший объем кода для проверки, 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.

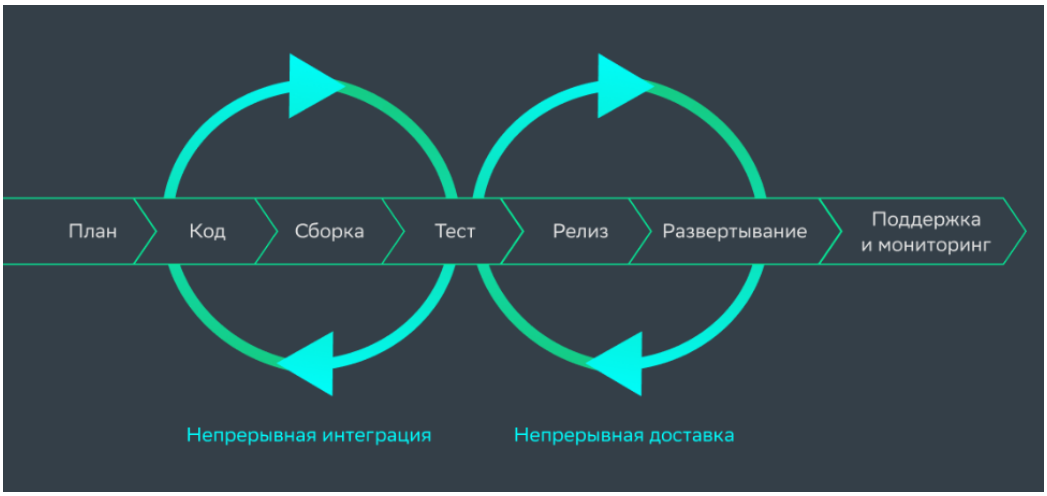


Рис.1.1. Цикл CI/CD [3]

## 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 [28].

TeamCity - это сервер CI от компании Jetbrains [32], 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 [22], 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 [19], 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 [13], который связывает автоматизированные сборки, тесты и выпуски в единый рабочий процесс.

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

Особое внимание следует уделить критерию OpenSource, этот критерий является достаточно важным с учетом, того, что многие компании после 2022 года ушли из РФ, тем самым стали либо недоступны, либо прекратили 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

Таблица 1.1

Сравнительный анализ инструментов CI/CD

| Критерий | Jenkins | TeamCity | GitLab CI | CircleCI | Bamboo |
|---|---|---|---|---|---|
| Открытый исходный код | + | - | + | - | - |
| Цена | Бесплатно | от 45$ в месяц [18] | от 21$ в месяц [2] | от 15$ в месяц [18] | от 1200$ в год [18] |
| 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 [18], 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 производить мониторинг и визуализацию информации по статистике работы сборок во времени. Под статистикой работы сборок, подразумеваются следующие показатели относительно собираемых метрик (продолжительность исполнения сборки, время проведенное в очереди сборкой, размеры полученных по итогам сборки артефактов):

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

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

В данной работе производится разработка плагина для визуализации метрик сборки Jenkins, основанием для разработки является отсутствие плагина, который полностью визуализирует метрики сборок в Jenkins, аналогично встроенному модулю в TeamCity. Многие российские ИТ-компании 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 этого 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. Результаты сравнения приведены в таблице 1.2.

Build Monitor Plugin - плагин, который обеспечивает наглядное представление статуса выбранных заданий Jenkins. Отображает состояние и ход выполнения выбранных заданий [14].

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 [23].

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 [15].

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.

Таблица 1.2

Сравнительный анализ плагинов Jenkins

| 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.

*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. должны 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 статистической обработки 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.

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

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

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, а также 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 рассчитаны веса каждой сборки/сборок по графику за определенный период, и если сборка была собрана, например, месяц назад - она должна иметь меньший вес, чем сборка, собранная вчера.

Также к разработке будет предъявлено требование об удобстве интерфейса: все графики должны быть удобными, не перегруженными информацией, а также интерфейс должен быть интуитивно 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. Архитектура Jenkins представлена на рисунке 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 [12].

Архитектура плагинов использует точки расширения, которые, предоставляют разработчикам плагинов возможности реализации для расширения функциональности системы Jenkins [33]. 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.

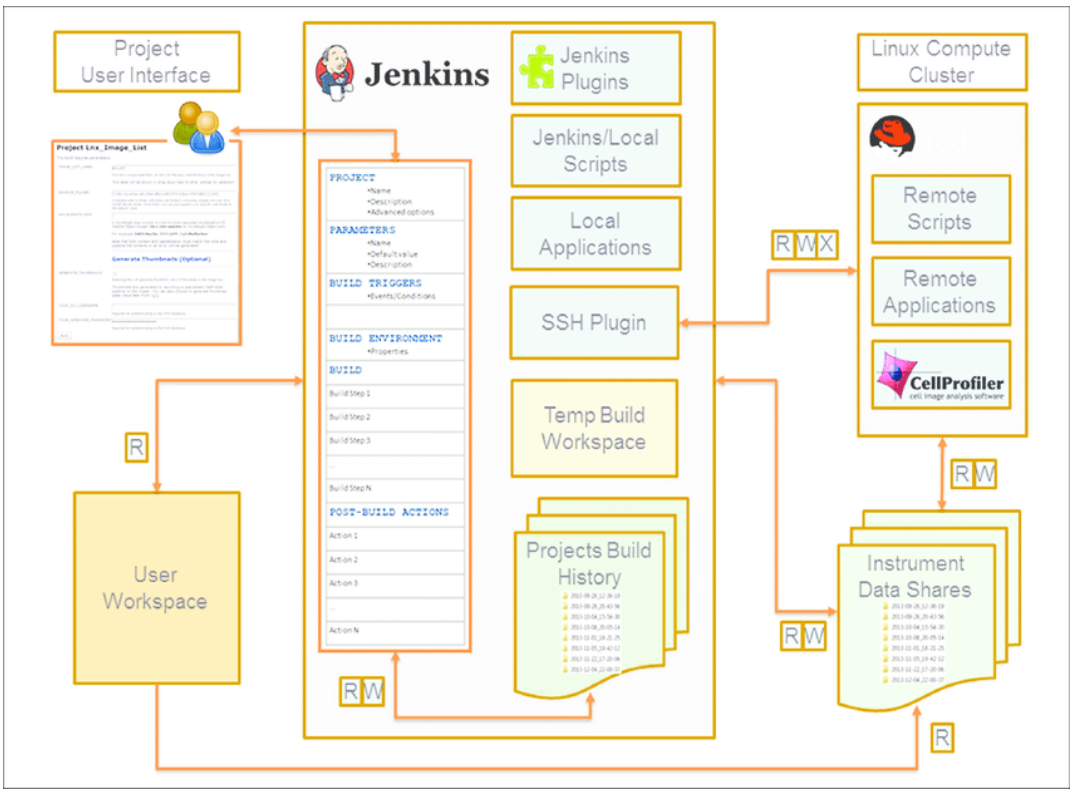


Рис.2.1. Архитектура Jenkins [12]

Помимо класса Action для того чтобы создать временные действия, которые будут прикреплены к заданию Jenkins будет использован класс TransientActionFactory, который позволяет создавать действия, которые будут отображаться на страницах Jenkins только при наличии соответствующего объекта - задания.

Разработка будет выполняться в объектно-ориентированной парадигме, т.е. приложение будет разбито на классы, будет применяться наследование, полиморфизм и инкапсуляция. Все классы, которые будут разработаны для плагина описаны в приложении П1.1.

При рассмотрении диаграммы необходимо отметить, что два класса являются встроенными в Jenkins, это TransientActionFactory, который позволяет добавлять действия к любому типу объекта, а также интерфейс Action - добавленный к объекту модели, создает дополнительное подпространство URL-адресов под родительским объектом модели, через которое он может взаимодействовать с пользователями. Actions также способны открывать доступ к левому меню в интерфейсы Jenkins, по которому обычно производится навигация при конфигурировании сборки.

Для удобства использования плагина, предполагается добавить дополнительную ссылку в меню слева, для перехода на страницу визуализации метрик, а

также динамически обновлять страницу при изменении параметров и фильтров, что и обосновывает использование данных встроенных классов.

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

## 2.3. Архитектура плагина

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

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

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

Все взаимодействие между Java, Jelly и JS происходит с помощью JSON структур, такое решение было принято ввиду удобства работы со структурой с помощью этих инструментов. На рисунке 2.2 представлено изображение архитектуры плагина.

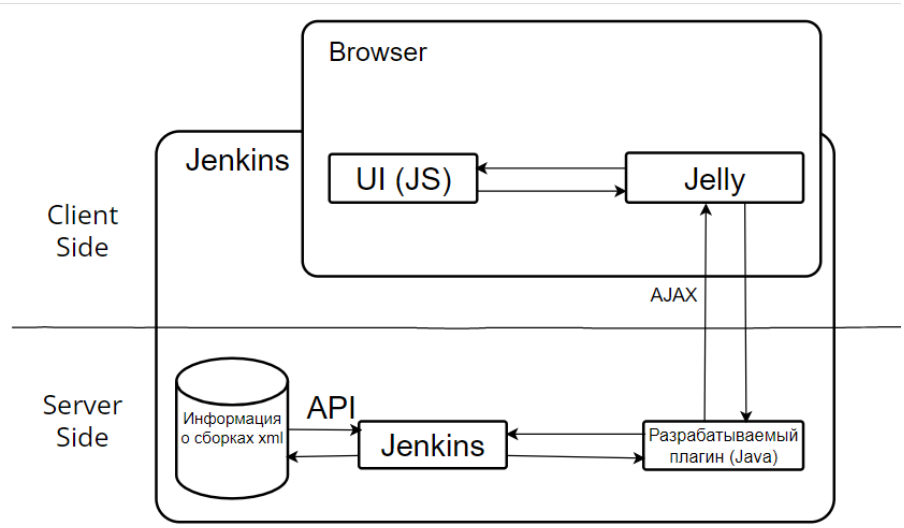


Рис.2.2. Архитектура плагина

## 2.4. Языки программирования

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

Есть возможность разработки плагина с использованием языка программирования Groovy. Groovy это динамический язык с возможностями статической типизации и статической компиляции для платформы Java[24], нацеленный на повышение производительности разработчиков, который плавно интегрируется с любой программой Java.

Недостатком такого выбора является то, что абсолютное большинство плагинов написано на чисто Java, а значит сообщества и поддержка при разработке на Java будет значительно большей. Также в сравнении с Groovy, Java обладает большей производительностью[31], 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 [25]. 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 [26], 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 [20], 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 [27]. В Jelly можно вызывать функции Java, использовать такие синтаксические конструкции, как циклы, условия и переменные, также он позволяет легко обратиться к объектам в Java.

## 2.5. Инструменты сборки

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

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

## 2.6. Библиотеки

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 [16]. 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 statistics) - метод для получения информации о времени продолжительности запусков сборки, за определенный период, с заданными настройками;
- String getBuildSuccessRate(String period) - метод для получения информации о проценте успешности выполнения запусков сборки, за определенный период;
- String getBuildArtifactSize(String period, String fail, String statistics) - метод для получения информации о созданных 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 statistics) - 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, List<Double» createDateMonthMap() - метод, создающий начальную структуру данных за прошедший месяц по дням, где в качестве значений словаря используются пустые списки;
- HashMap<String, List<Double» createDateAllMap(RunList<Run> runs) - метод для создания начальной структуры данных, за весь прошедший период, который вычисляет на какие равные интервалы следует разбить общий промежуток времени, от создания первой сборки, до создания последней сборки;
- HashMap<String, List<Double» createDateDayMap() - метод, создающий начальную структуру данных за прошедший день по часам, где в качестве значений словаря используются пустые списки;
- HashMap<String, HashMap<String,Integer» createDateDayMapSuccess() - метод, создающий начальную структуру данных за прошедший день по часам, для вычисления процента успешности выполнения сборок, где в качестве значений словаря используется словарь, в котором записано сколько запусков сборки выполнено успешно, а сколько с ошибками;
- HashMap<String, HashMap<String,Integer» createDateWeekMapSuccessRate() - метод, создающий начальную структуру данных за прошедшую неделю по дням, для вычисления процента успешности выполнения сборок, где

в качестве значений словаря используется словарь, в котором записано сколько запусков сборки выполнено успешно, а сколько с ошибками;

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

- HashMap<String, List<Double» createDateQuarterMap() - метод, создающий начальную структуру данных за прошедший квартал по месяцам, где в качестве значений словаря используются пустые списки;
- HashMap<String, List<Double» createDateWeekMap() - метод, создающий начальную структуру данных за прошедшую неделю по дням, где в качестве значений словаря используются пустые списки;
- HashMap<String, HashMap<String,Integer» createDateYearMapSuccessRate() - метод, создающий начальную структуру данных за прошедший год по месяцам, для вычисления процента успешности выполнения сборок, где в качестве значений словаря используется словарь, в котором записано сколько запусков сборки выполнено успешно, а сколько с ошибками.

### *3.1.3. IntervalDate*

Перечисляемый тип для удобства работы с датами-периодами. Содержит следующие 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. Statistics*

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

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

– SD - среднеквадратичное отклонение.

### *3.1.5. TimeInQueueFetcher*

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

### *3.1.6. BuildLogic*

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

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

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

– BuildLogic(IntervalDate period, Boolean failed, RunList<Run> buildList) - конструктор класса;
– void filterPeriodBuild() - метод, который производит 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.

### *3.1.7. BuildArtifactSizeLogic*

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

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

Также в классе определены методы:

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

### *3.1.8. BuildDurationLogic*

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

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

Также в классе определены методы:

– BuildDurationLogic(IntervalDate period, Boolean failed, RunList<Run> buildList) - конструктор класса;
– Map<String, Double> getBuildsDuration(Statistics statistics) - метод, в котором происходит фильтрация данных запусков сборок по периоду и

флагу failed, определение формата дат и вычисление времени выполнения запусков сборок, а также расчет по статистической величине (например, среднее арифметическое), в зависимости от заданных настроек.

### *3.1.9. BuildSuccessRateLogic*

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

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

Также в классе определены методы:

- BuildSuccessRateLogic(IntervalDate period, RunList<Run> buildList) - конструктор класса;
- Map<String, Double> getSuccessRate() - метод, в котором происходит фильтрация данных запусков сборок по периоду, определение формата дат и вычисление процента успешности выполнения запусков сборок.

### *3.1.10. BuildTestCountLogic*

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

- HashMap<String, Integer> testCountOnFormatDate - структура данных для работы с запусками сборки относительно метрики TC, ключи даты за выбранный период, значения количество выполненных тестов запусков сборки за выбранный период;
- String dateFormatKey - поле, которое определяет формат даты за выбранный период, по которому будет происходить обработка и визуализация;

– Logger LOGGER - поле, с помощью которого записываются логи плагина при выполнении методов класса.

Также в классе определены методы:

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

### *3.1.11. BuildTimeQueueLogic*

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

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

Также в классе определены методы:

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

### *3.1.12. 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() - функция в которой происходит подготовка данных, формирование настроек и создание графика по метрике SR;
- typeChartHandler(typeChart, labels, title, dictValues) - функция, которая обрабатывает событие изменение типа графика/диаграммы, и обновляет представление на странице плагина;
- createTestCountChart() - функция в которой происходит подготовка данных, формирование настроек и создание графика по метрике TC;
- createBuildDurationChart() - функция в которой происходит подготовка данных, формирование настроек и создание графика по метрике BD;
- createArtifactSizeChart() - функция в которой происходит подготовка данных, формирование настроек и создание графика по метрике AS;
- createTimeQueueChart() - функция в которой происходит подготовка данных, формирование настроек и создание графика по метрике TQ.

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

## 3.2. Результаты разработки плагина

При разработке плагина надо было учитывать, что требуется отображать все графики на одной странице задания друг под другом, поскольку при выборе 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, обработать метрику в соответствии со статистическим показателем.

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

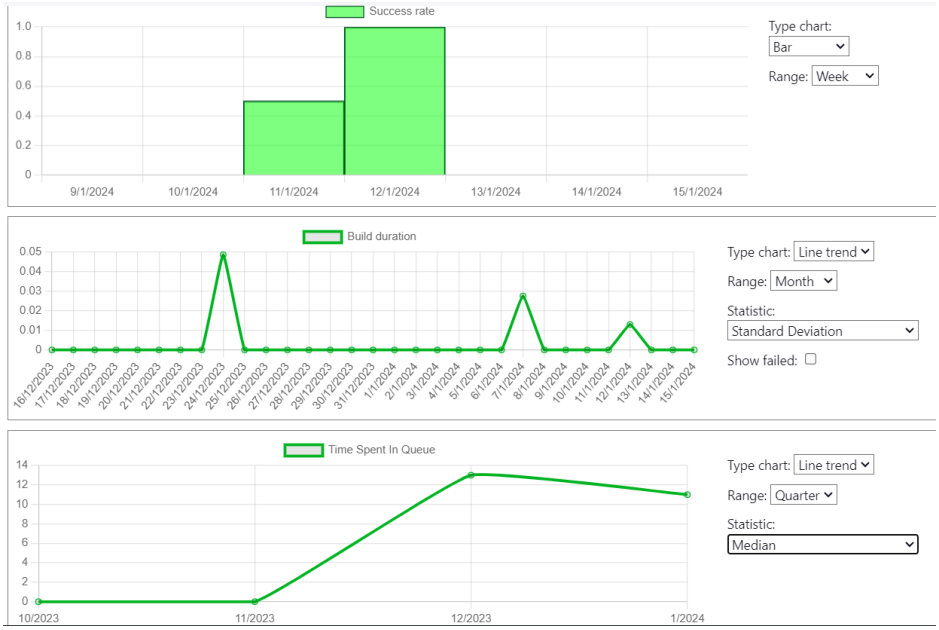


Рис.3.1. Интерфейс плагина Jenkins

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

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

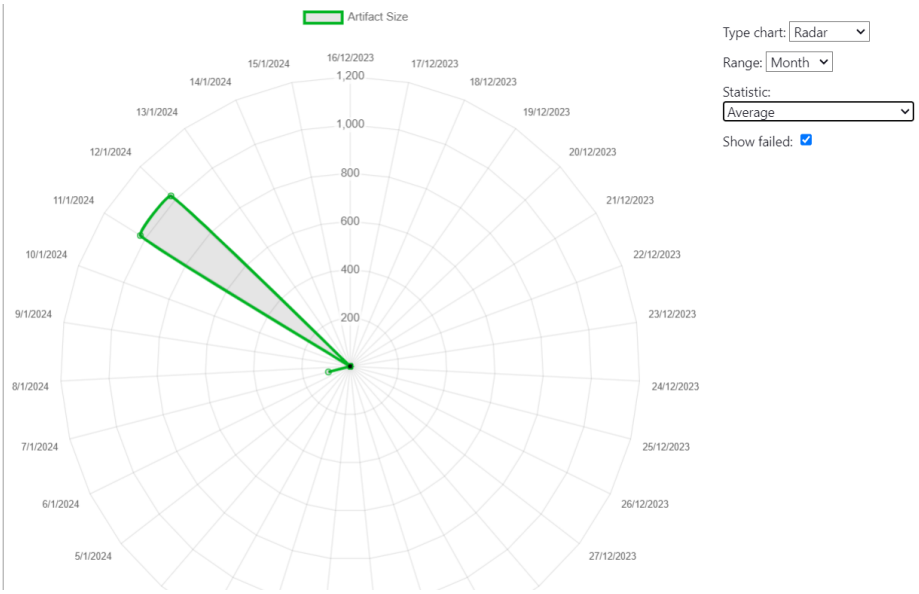


Рис.3.2. Интерфейс графика Radar для AS

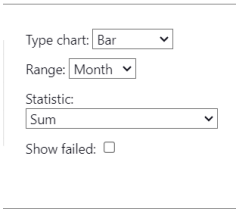


Рис.3.3. Интерфейс элементов управления

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 статистического показателя, а также включение в графики данных об упавших сборках, при выборе соответствующих чекбоксов.

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

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 взаимодей-

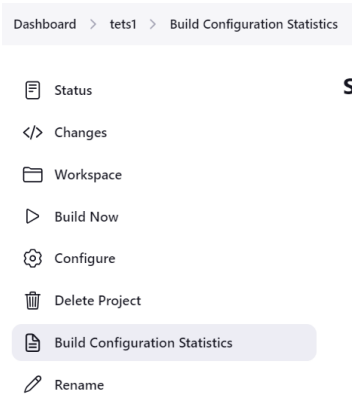


Рис.3.4. Интерфейс элементов управления

ствии 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 [8]. В процессе CI/CD производится непрерывное тестирование разработанного кода, а также тестирование разработанного приложения. Сам плагин является частью CI/CD процессе, но также требует тестирования корректной работы своей функциональности, тестирование разработанного кода, а также тестирования на соответствие исходным 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 [5].

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

В данном случае тест-кейсы были описаны с помощью техники тест-дизайна Матрица трассировки. Если обратиться к определению, то матрица трассировки — двумерная таблица, содержащая соответствие функциональных требований и подготовленных тестовых сценариев [4], а на пересечении столбцов и строк ставится метка, о том, что данное требование покрывается данным тест-кейсом. В случае данной работы из соображений удобства и оптимизации тестовой документации, было принято решение модифицировать матрицу трассировки и 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 [6] 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 [34].

### *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 [36], т.е. тестирование происходит на основе знания исходного кода, таким образом, юнит тестирование методом белого ящика поможет нам достаточно широко покрыть все модули

Таблица 4.1

Составленные тест-кейсы

| № | Описание | Ожидание |
|---|---|---|
| 1 | Проверка выбранного периода на графике SR (месяц) | Количество дней соответствует прошлому месяцу, на каждый день отображены корректные значения процента успешности сборок |
| 2 | Проверка выбранного периода на графике BD (неделя) с учетом выбора настройки среднего значения и упавших сборок | Количество дней 7 в соответствии со всеми днями недели, на каждый день отображены корректные значения среднего времени продолжительности сборок, учтены упавшие сборки |
| 3 | Проверка выбранного периода на графике TQ (квартал) с учетом выбора настройки среднего значения | Количество кварталов 4 в соответствии с кварталами года, на каждый квартал отображены средние значения проведенного в очереди времени сборок |
| 4 | Проверка выбранного периода на графике TC (год) с учетом выбора настройки упавших сборок | Количество месяцев 12 соответствует прошедшему году, на каждый месяц отображены корректные значения количества тестов, с учетом тестов выполненных на упавших сборках |
| 5 | Проверка выбранного периода на графике AS (день) | Количество часов 24 соответствует всем часам прошедшего дня, на каждый час отображены корректные значения размера итоговых артефактов полученного по результатам задания |
| 6 | Проверка корректного отображения 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 |

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 [35]. 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 [30] 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. Пример отредактированной сборки с датой в прошлом указан на рисунке 4.1.

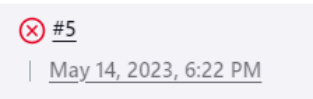


Рис.4.1. Сборка с датой в прошлом

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

Запуски были сгенерированы с разной датой начала:


Рис.4.2. Сгенерированные сборки

– 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. Такая сборка отображена на рисунке 4.3.


Рис.4.3. Длинная упавшая сборка

А также сборки, с созданными артефактами, часть из которых в статусе успешного выполнения, с короткой продолжительностью, а часть из которых завершены падением с большой продолжительностью выполнения. Были определены разные файлы-артефакты для генерации с размером от 70 байт до 1 Кб. Пере-

нос созданных файлов в артефакты выполнялся с помощью конфигурационного разделы в сборке Post-build Actions, в котором выплавлялась команда Archive the artifacts. Сборка с артефактом отображена на рисунке 4.4.


Рис.4.4. Короткая успешная сборка с артефактом

Часть сборок выполнялось посредством созданного класса в плагине BuildConfigurationStatisticsBuilder, который добавлял еще один вариант запуска шагов сборки Build Steps. Этот класс выполнял вывод информации о 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 4.5.

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

Рис.4.5. Консоль шага с разработанным Builder

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

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

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 [21]. 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.

Для моделирования ситуации просмотра статистики по метрикам сборки в условиях разных версий продукта, когда вносятся значительные изменения в код, что влечет за собой увеличния, в возможно и уменьшения (в случаи оптимизации) времени сборки продукта, необходимо откатиться к более ранним коммитам. Поскольку для апробации используется открытый проект, то данные манипуляции с исходным кодом возможно выполнить. Для того чтобы откатиться к предыдущим версиям, были проделаны следующие действия:

A. Сделать fork проекта в личный репозиторий.
B. Найти подходящий коммит, в котором были выполнены значительные изменения (более 500 строк измененного кода).
C. Откатиться до найденного коммита.

D. Создание новой ветки на основе коммита, до которого произошел откат.

E. Отправка ветки в личный удаленный репозиторий на GitHub.

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

Для отката к более ранним версия был написан скрипт на языке Python, представленный в приложении П7.

Далее при генерации сборок было произведено по 2 запуска на каждую версию продукта, время в каждой сборке было отредактировано на каждый месяц за прошедший год. Перед выполнением каждого запуска был отредактирован параметр, по которому определяется из какой ветки берется исходный код продукта.

—

Для проверки подсчета метрики TC, был добавлен второй шаг *mvn clean test*, который запускает юнит тесты, а также в разделе Post-build Actions добавлено действие Publish JUnit test result report, которое по результатам прогона теста формирует отчет JUnit из xml лога с результатами, созданного после выполнения всех тестов.

—

Сгенерированные на этом проекте сборки отображены на рисунке 4.6.

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

В 8 месяце в изменениях кода была повышена версия npm и node, что оптимизировало время выполнения сборки, которое уменьшилось с 19.3 до 18.3 секунд. А версия в 11 месяце, вероятно была не протестировано перед загрузкой в главную ветку, поскольку результат из 3 запусков сборки закончился падением. Что также видно на графике SR, т.е. можно сделать вывод что в коммите был дефект, а не оптимизация, которая ускорила время сборки в несколько раз.

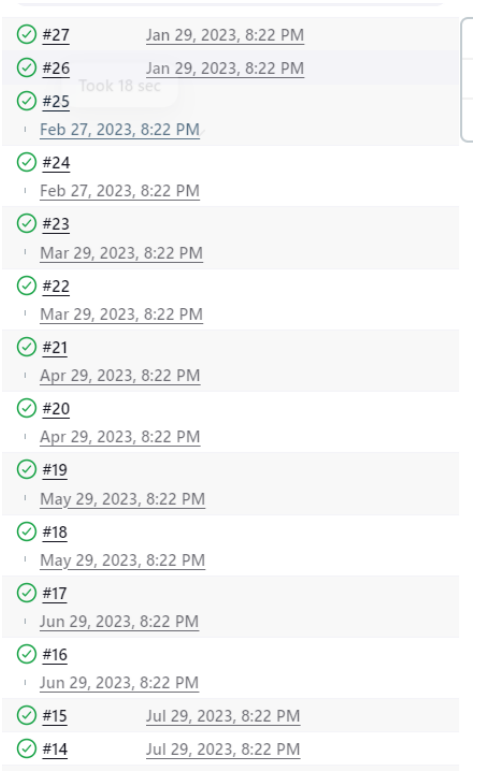


Рис.4.6. Сгенерированные сборки реального проекта

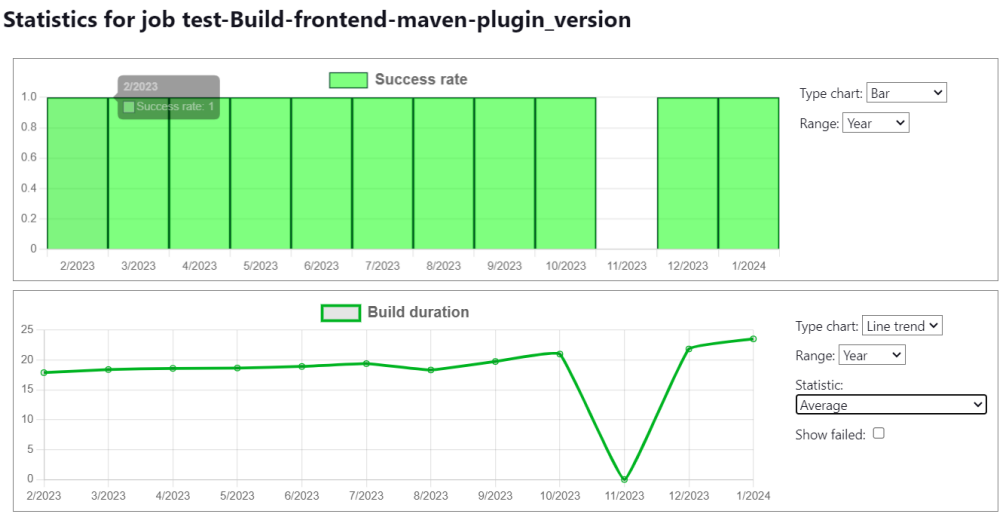


Рис.4.7. Результаты апробации на реальном проекте SR, BD

На рисунке 4.8 можно увидеть с помощью радарной диаграммы общий размер, сгенерированных логов-артефактов за последний год. Видно, что с каждым месяцем размер артефактов увеличивался, что также можно объяснить увеличением исходного кода продукта, также наглядна видно разницу между первым и последним месяцем года, а также то что при отображении упавших сборок видно, то что генерировались артефакты, хоть и с небольшим размером.

По описанным выше результатам визуализации можно прийти к выводу, что плагин полезен тем, что отображает изменение различных метрик запусков сборки

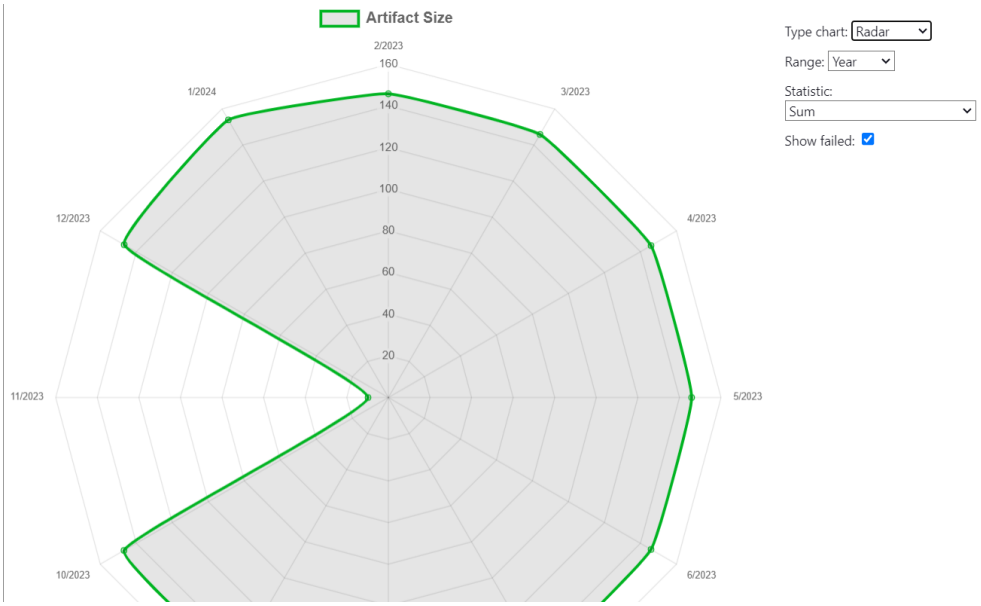


Рис.4.8. Результаты апробации на реальном проекте AS

с течением времени, т.е. можно увидеть тенденцию изменений в созданных сборках Jenkins в течении цикла разработки за нужный период времени и если метрики в какой момент изменили свои значения, можно определить, что это за момент (или период) и проанализировать как изменения в сборке, тестах или коде могли повлиять на это.

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

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

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

### *4.2.3. Оценка результатов работы*

Для того чтобы оценить результаты работы, будет проведено сравнение функционала, который присутствует в аналогичных решениях: плагинах, сравнительный

Таблица 4.2

Количественные результаты работы

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

анализ, которых проводился в разделе 1.5 и модуль Statistics, реализованный в средстве CI TeamCity, который и послужил причиной для создания аналогичного модуля в Jenkins.

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

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

Все результаты сравнения с аналогичными решениями приведены в таблице 4.2.

Если сравнивать по количеству визуализируемых метрик, то видно, что все 5 метрик, которые были реализованы в TeamCity, удалось реализовать и в разработанном плагине, аналогичные плагины Jenkins визуализируют определенные из перечисленных в разделе 1.5 метрик, но их количество меньше 5.

### 4.3. Выводы

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

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

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

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

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

# ЗАКЛЮЧЕНИЕ

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

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

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

По итогу реализации объем кода проекта составляет 2889 строк, из которых:

- 1533 строк - Java код на сервере Jenkins;
- 755 строк - Jelly и JS на клиентской части;
- 411 строк - Java unit тесты;
- 188 строк - Python код UI тестов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 5 сборщиков модулей для приложений Node.js. — URL: https://tproger.ru/articles/5-razlichnyh-instrumentov-dlya-obedineniya-prilozhenij-node-js (дата обращения: 20.11.2023).

2. Как использовать GitLab в условиях санкций? — URL: https://habr.com/ru/companies/ruvds/articles/715010/ (дата обращения: 20.11.2023).

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

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

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

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

7. *Соколова А.* Подходит CI/CD вашему бизнесу: плюсы и минусы конвейеров // Cloud Networks. — 2021. — URL: https://cloudnetworks.ru/analitika/podhodit-ci-cd-vashemu-biznesu-plyusy-i-minusy-konvejerov/ (дата обращения: 20.11.2023).

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

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

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

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

12. Jenkins-CI, an Open-Source Continuous Integration System, as a Scientific Data and Image-Processing Platform / I. Moutsatsos [и др.] // Journal of Biomolecular Screening. — 2016. — Т. 22. — С. 1087057116679993. — DOI 10.1177/1087057116679993.

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

14. Build Monitor View. — URL: https://plugins.jenkins.io/build-monitor-plugin/ (visited on 20.11.2023).

15. Build Time Blame. — URL: https://plugins.jenkins.io/build-time-blame/ (visited on 20.11.2023).

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

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

18. 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).

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

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

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

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

23. global-build-stats. — URL: https://plugins.jenkins.io/global-build-stats/ (visited on 20.11.2023).

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

25. Java Docs. — URL: https://docs.oracle.com/en/java/ (visited on 02.10.2023).

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

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

28. Jenkins Documentation. — URL: https://www.jenkins.io/doc/ (visited on 02.10.2023).

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

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

31. *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).

32. TeamCity Docs. — URL: https://www.jetbrains.com/help/teamcity/teamcity-documentation.html (visited on 02.10.2023).

33. 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).

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

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

36. 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

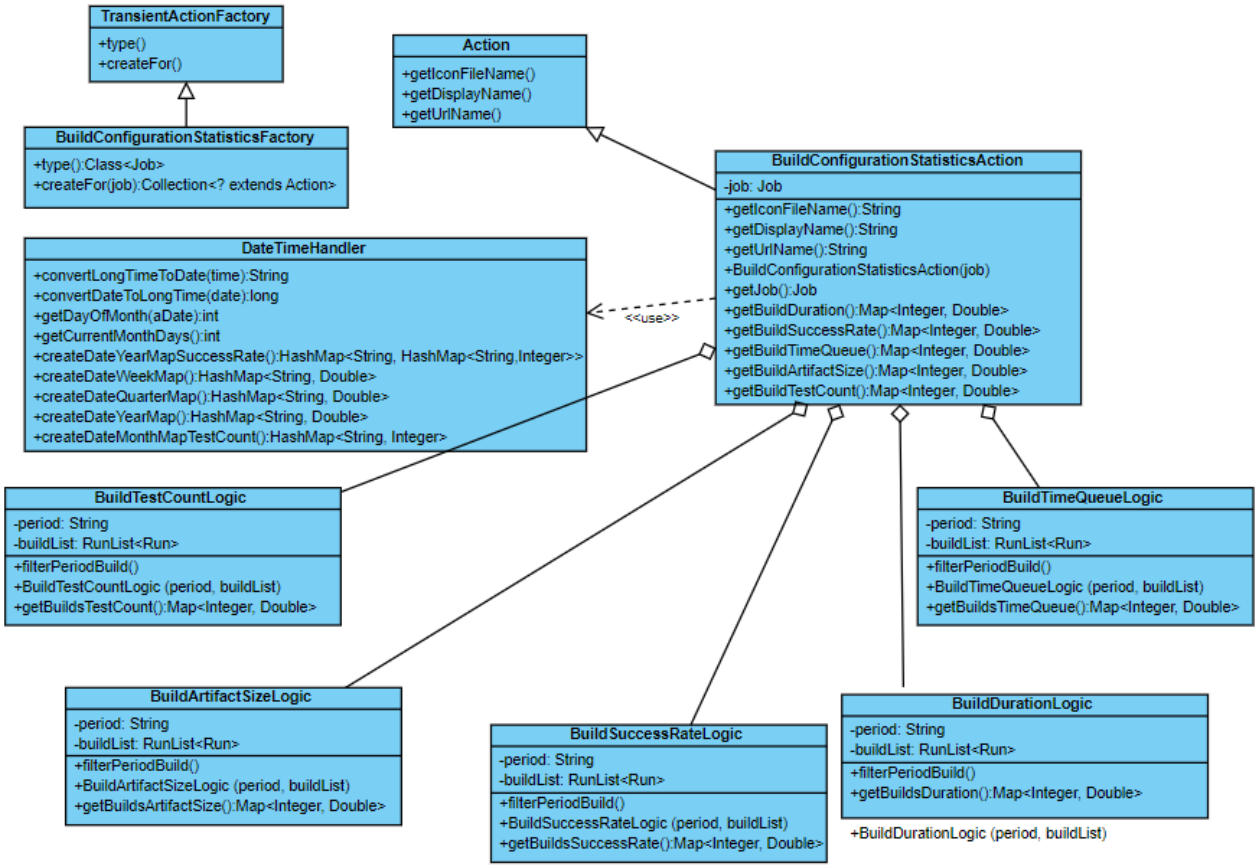


Рис.П1.1. Диаграмма классов плагина

**Приложение 2**

# Idef0

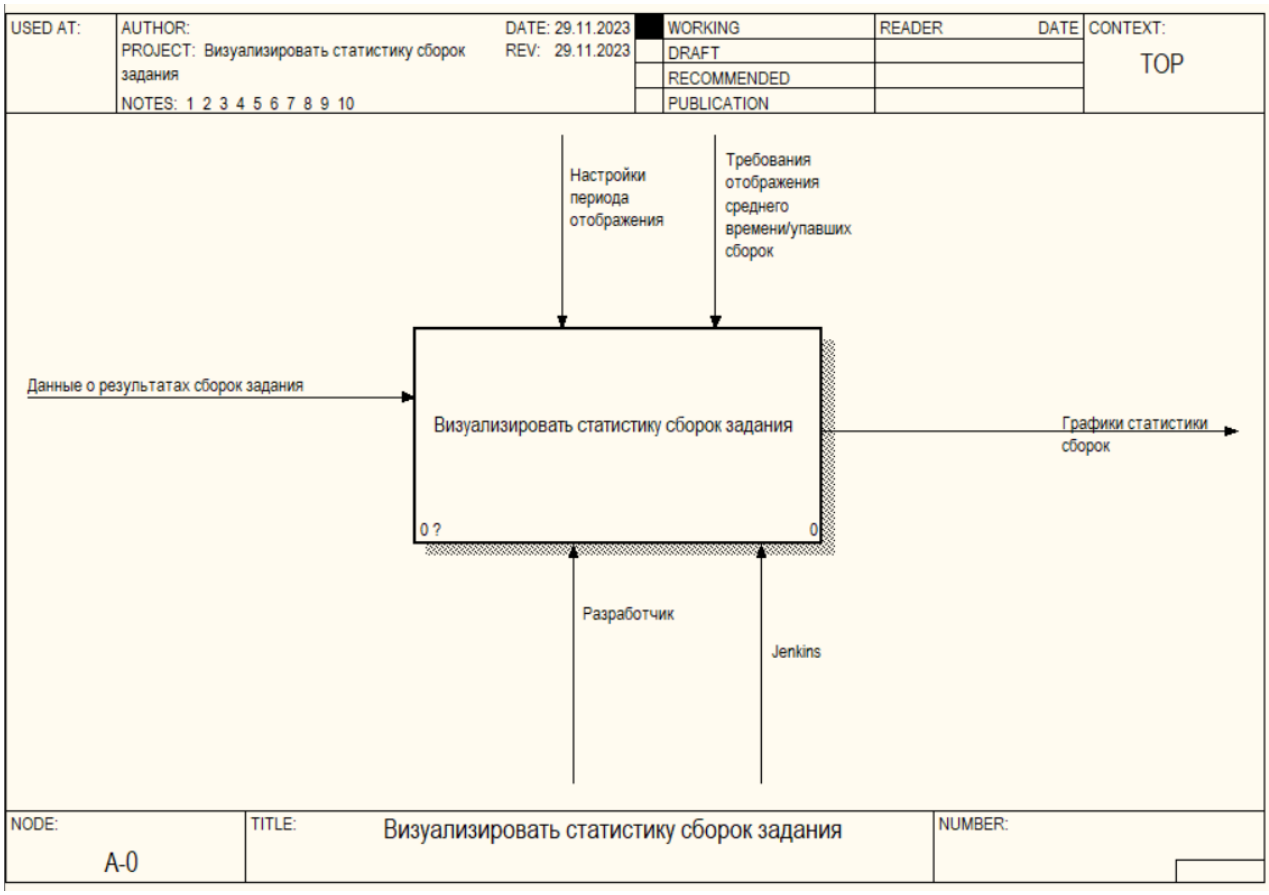


Рис.П2.1. Процесс визуализации сборок

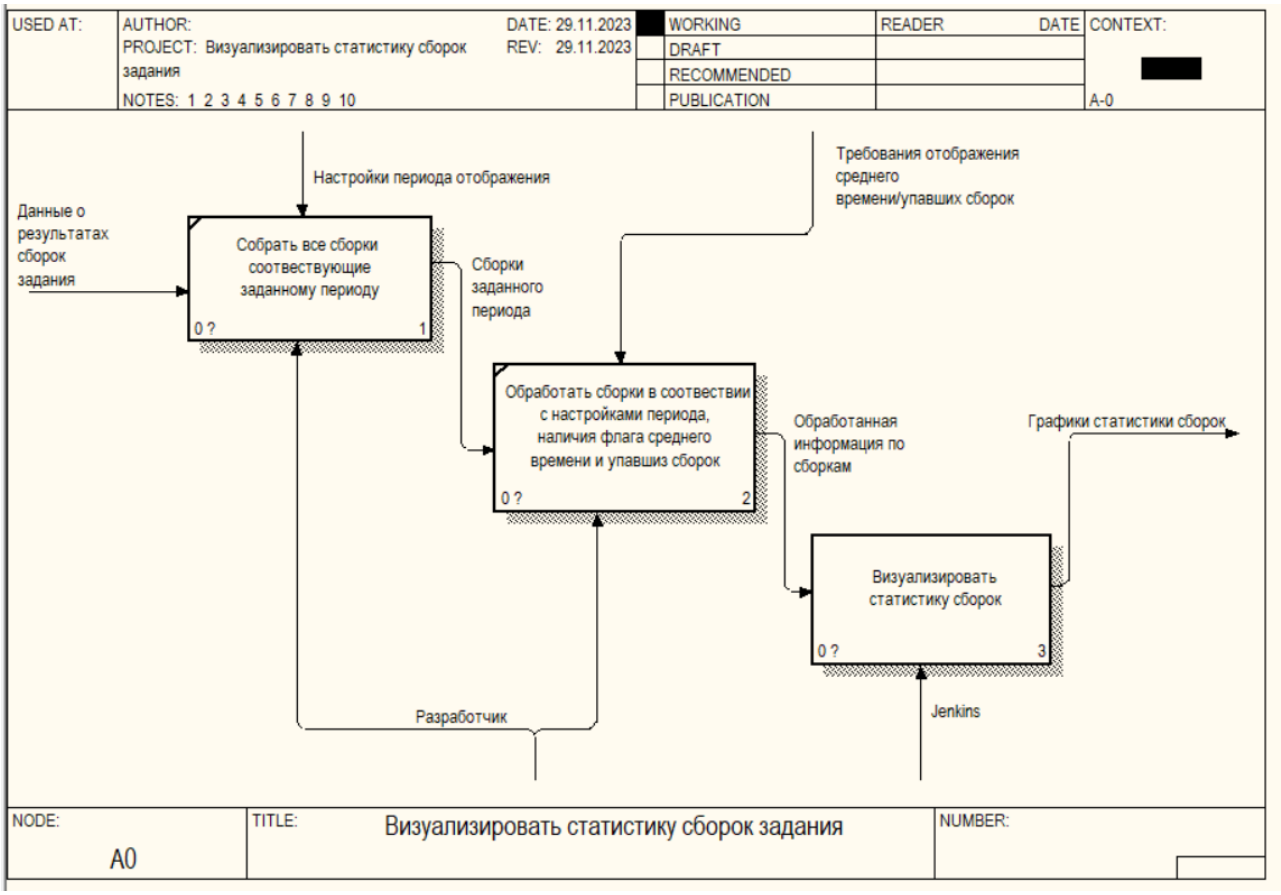


Рис.П2.2. Детализация процесса визуализации

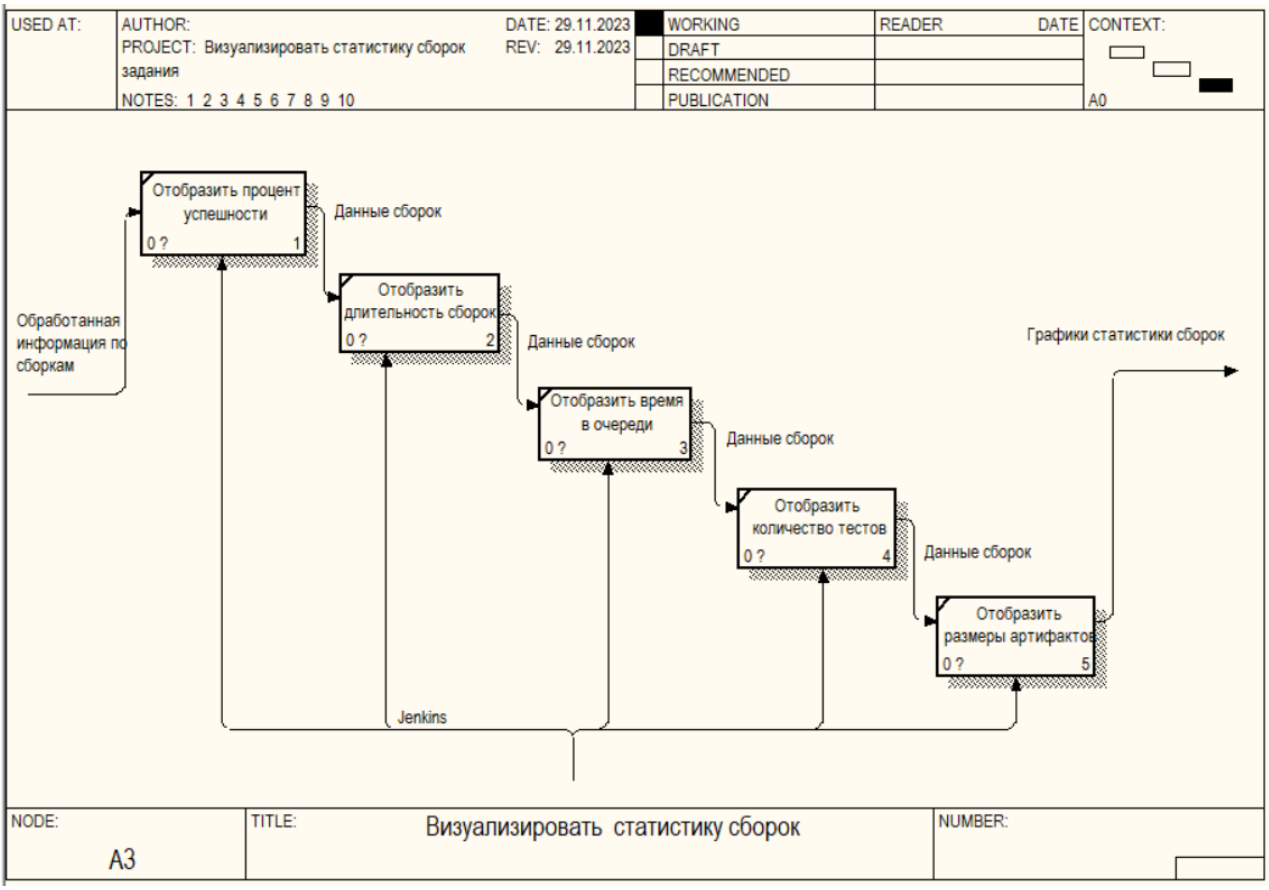


Рис.П2.3. Процесс построения графиков