

## РАЗРАБОТКА ПЛАГИНА JENKINS ДЛЯ ВИЗУАЛИЗАЦИИ СТАТИСТИКИ ПО ПРОЦЕССУ СБОРКИ ПРОГРАММЫ

© 2024 г. А. Д. Кухто, В. А. Пархоменко, А. В. Щукин

*\*Санкт-Петербургский политехнический университет Петра Великого Институт компьютерных наук и кибербезопасности*

*195251 Санкт-Петербург, Политехническая, ул., д. 29, 3-й учебный корпус, каб. 308*

*E-mail: kuhto.ad@edu.spbstu.ru, vladimir.parkhomenko@spbstu.ru, Alexander.Schukin@spbstu.ru*

Поступила в редакцию 10.07.2012

В настоящий момент российские ИТ-компании для реализации процессов разработки и тестирования отдают предпочтения отечественным инструментам или инструментам с открытым исходным кодом. Одним из таких инструментов является CI платформа Jenkins. В своем функционале Jenkins уступает коммерческим продуктам, таким как TeamCity. Одним из таких недостатков является отсутствие полноценного плагина для визуализации истории изменения процесса сборки программы. Целью работы является разработка плагина Jenkins для визуализации статистики по процессу сборки программы. Объект исследования — средства для сборки приложений в инструментах совместного использования. Предмет исследования — визуализация статистики по процессу сборки программы. Основными методами проведения работы являются методы сравнительного анализа аналогичных решений и методы объектно-ориентированного программирования. Перед началом разработки были выбраны инструменты и изучен процесс разработки плагинов в Jenkins, а также рассмотрены аналогичные плагины. После анализа предметной области было проведено проектирование архитектуры плагина, а также функциональная модель системы. В результате работы разработан прототип плагина для визуализации статистики по процессу сборки программы Jenkins. Для визуализации и обработки метрик сборок применялись статистические показатели, разные типы диаграмм, а также анализ данных для прогнозирования значения метрик. Проведена апробация и тестирование разработки на реальном проекте с открытым исходным кодом в системе Jenkins. Область применения разработанного плагина - промышленная разработка программных продуктов, которые собираются и тестируются с использованием CI/CD инструмента Jenkins. Плагин рекомендуется использовать для оценки эффективности CI/CD процесса, а также принятия решений по улучшению качества кода и тестирования продукта на основе полученных данных. КЛЮЧЕВЫЕ СЛОВА: ВИЗУАЛИЗАЦИЯ, СТАТИСТИКА, JENKINS, ПЛАГИН, СБОРКА, МЕТРИКИ СБОРКИ.

### 1. Введение

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

Для упрощения процесса разработки программного обеспечения в настоящий момент широко применяются практики DevOps, одной

из которых является непрерывная интеграция, сборка и доставка (Continuous Integration, Continuous Delivery, далее — CI/CD) [1]. Существует множество средств CI/CD, которые применяются в промышленной разработке: TeamCity, Jenkins, Gitlab CI и другие.

Под *сборкой программного продукта* будем подразумевать процесс объединения отдельных файлов и компонентов программы в единый исполняемый файл или пакет, который включает в

себя компиляцию, связывание модулей, оптимизацию и другие операции, необходимые для создания готового к выполнению приложения [2].

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 [3].

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

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

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

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

аналогов.

**Объект исследования** — средства для сборки приложений в инструментах совместного использования.

**Предмет исследования** — визуализация статистики по процессу сборки программы.

**Цель** — разработать плагин Jenkins для визуализации статистики по процессу сборки программы.

Задачи:

1. Изучить инструменты сборки приложений.
2. Изучить особенности CI/CD, Jenkins, работу и характеристики сборок Jenkins.
3. Описать метрики и статистики, которые могут собираться по результатам работы сборки Jenkins.
4. Изучить методы разработки плагинов Jenkins.
5. Провести проектирование плагина и описать архитектуру.
6. Реализовать плагин.
7. Провести тестирование и апробацию плагина.

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

## 2. АНАЛИЗ СРЕДСТВ СБОРКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

### 2.1. Инструменты 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.

Gradle — система автоматизации сборки, которая также часто используется для Java разработки.

Для проектов на JavaScript для управления зависимостями и сборками может использоваться npm в связке с Webpack.

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

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

Инструменты CI/CD позволяют 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.

## 2.2. Анализ средств CI/CD

CI/CD — это технология автоматизации тестирования и доставки/развертывания готового приложения заказчику [4]. 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 [5]:

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

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

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.

TeamCity - это сервер CI от компании JetBrains, 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, 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, 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, который связывает автоматизированные сборки, тесты и выпуски в единый рабочий процесс.

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

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

Если сравнивать описанные выше средства, то TeamCity обладает самой мощной встроенной функциональностью, а также достаточно большим количеством интеграций и плагинов [6], 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 количеством интеграций и плагинов, которые постоянно пишутся, что позволяет устранить основной его недостаток по наличию встроенных функций.

### 2.3. Анализ существующих плагинов Jenkins по визуализации статистики процесса сборки

Поскольку для работы со сборками приложений был выбран Jenkins, то разработка плагина будет производиться в этой системе. В любой системе с большим количеством приложений, сборок и тестов будет удобно производить мониторинг и визуализацию информации по статистике работы сборок во времени.

Под статистикой работы сборок будем понимать следующие статистические характеристики собираемых метрик (продолжительность исполнения сборки, время проведенное в очереди сборкой, размеры полученных по итогам сборки артефактов): среднее арифметическое, мода, медиана, размах, среднеквадратическое отклонение, среднеквадратическое отклонение несмещенное, дисперсия.

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

- визуализация метрики *Success Rate* (далее — SR) - процент успешности сборок, который будет показывать сколько сборок завершилось успешно;
- визуализация метрики *Build Duration* (далее — 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* (далее — TQ) - время проведенное в очереди сборок, в том числе среднее время, вычисляемое аналогично Build Duration;
- визуализация метрики *Test Count* (далее — TC) - количество выполненных тестов в

сборке, в том числе количество выполненных тестов в упавших сборках, если таковые успели выполниться;

- визуализация метрики *Artifacts Size* (далее — AS) - размер созданных во время сборки артефактов, в том числе средний размер за определенный интервал времени, а также учет артефактов, которые успели создаться в сборках до падения.

Сначала рассмотрим уже разработанные плагины визуализации и их недостатки и преимущества в сравнении с разрабатываемым решением. Результаты сравнения приведены в табл.1.

Build Monitor Plugin - плагин, который обеспечивает наглядное представление статуса выбранных заданий Jenkins.

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.

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 сборки, а также наличие прогнозирования метрик следующей сборки. Также данные плагины не предлагают динамическое изменение графиков по мере изменения временного интервала или установления фильтров.

#### 2.4. Требования к разработке

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

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

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

1. Должна собираться информация о требуемой метрике у всех сборок.
2. Производиться фильтрация сборок т.е. должны отбираться только сборки за последний месяц (в том числе упавшие, если был выбран данный чекбокс).
3. Полученные сборки должны группироваться по дням т.е. на итоговом графике должно быть 30/31 точка или столбца.
4. Если необходимо производиться вычисление статистической обработки среди всех сгруппированных за день метрик сборок.
5. 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.

По метрикам BD, AS, TQ должна быть возможность выбрать статистический показатель, в соответствии с которым должна производиться обработка итоговых значений. Возможные показатели перечислены в разделе 2.3.

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

Также было принято решения добавить анализ данных, чтобы делать предположение, о том какими метриками будет обладать следующая запущенная сборка, при вычислении данного значения должно быть рассчитаны веса каждой сборки/сборок по графику за определенный

Таблица 1.

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

период, и если сборка была собрана, например, месяц назад - она должна иметь меньший вес, чем сборка, собранная вчера.

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

### 2.5. Выводы

По всем описанным выше разделам можно прийти к выводу, что данный плагин актуален для ИТ-компаний, которые ранее отдавали предпочтение многофункциональному инструменту TeamCity, в котором уже были все необходимые для работы функции, особенно это актуально для компаний в РФ, но также может понадобиться и другим компаниям, которые приняли решение отказаться от TeamCity в пользу Jenkins из-за больших денежных затрат на лицензию. Также будет реализован 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.

## 3. ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПЛАГИНА

### 3.1. Модель системы

Диаграмма вариантов использования, показывающая функционал плагина отображена на рис.1. На данной диаграмме основное внимание также уделяется процессу визуализации статистики метрик сборки. 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 archimate отображена на рис.2. Основное внимание на диаграмме уделяется визуализации статистики сборки, поскольку это изначально является целью разработки. Также там будут отражены дополнительные функции такие как фильтрация, и вычитывание статистических метрик.

### 3.2. Архитектура Jenkins

Перед объяснением построения архитектуры плагинов Jenkins, необходимо привести описание архитектуры Jenkins, где будет отображено место разрабатываемых плагинов в CI системе. Установленные плагины Jenkins-CI, а также локальные сценарии и приложения выполняются на сервере Jenkins-CI и предоставляют расширяемый набор функций управления и обработки данных [7].

Архитектура плагинов использует точки расширения, которые, предоставляют разработчикам плагинов возможности реализации для расширения функциональности системы Jenkins [8]. Точки расширения автоматически обнаруживаются Jenkins во время загрузки системы.

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

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

Разработка будет выполняться в объектно-ориентированной парадигме, т.е. приложение будет разбито на классы, будет применяться наследование, полиморфизм и инкапсуляция. Все классы, которые будут разработаны для плагина отображены на рис.3.

При рассмотрении диаграммы необходимо отметить, что два класса являются встроенными

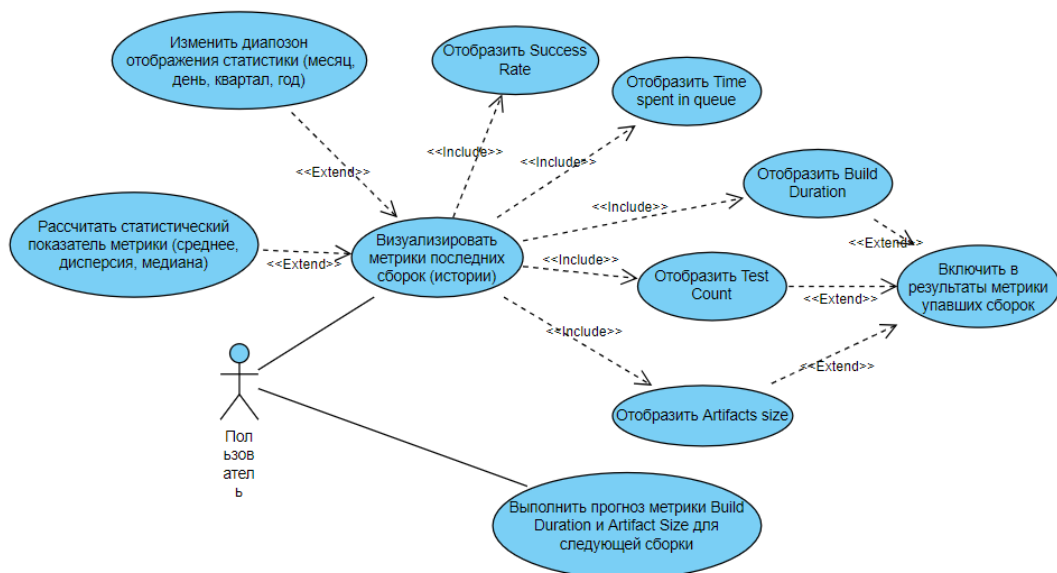


Рис. 1.

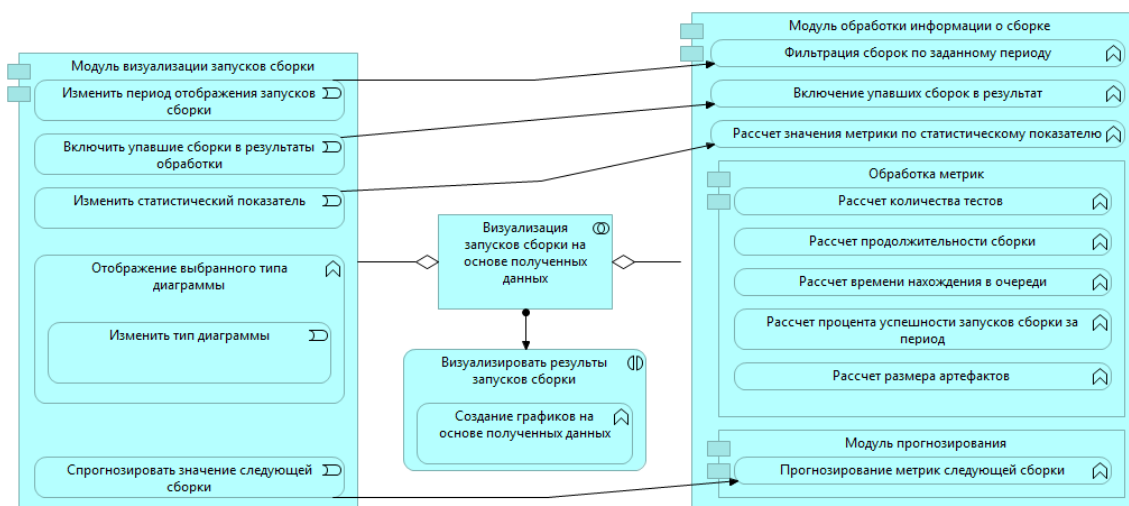


Рис. 2.

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.

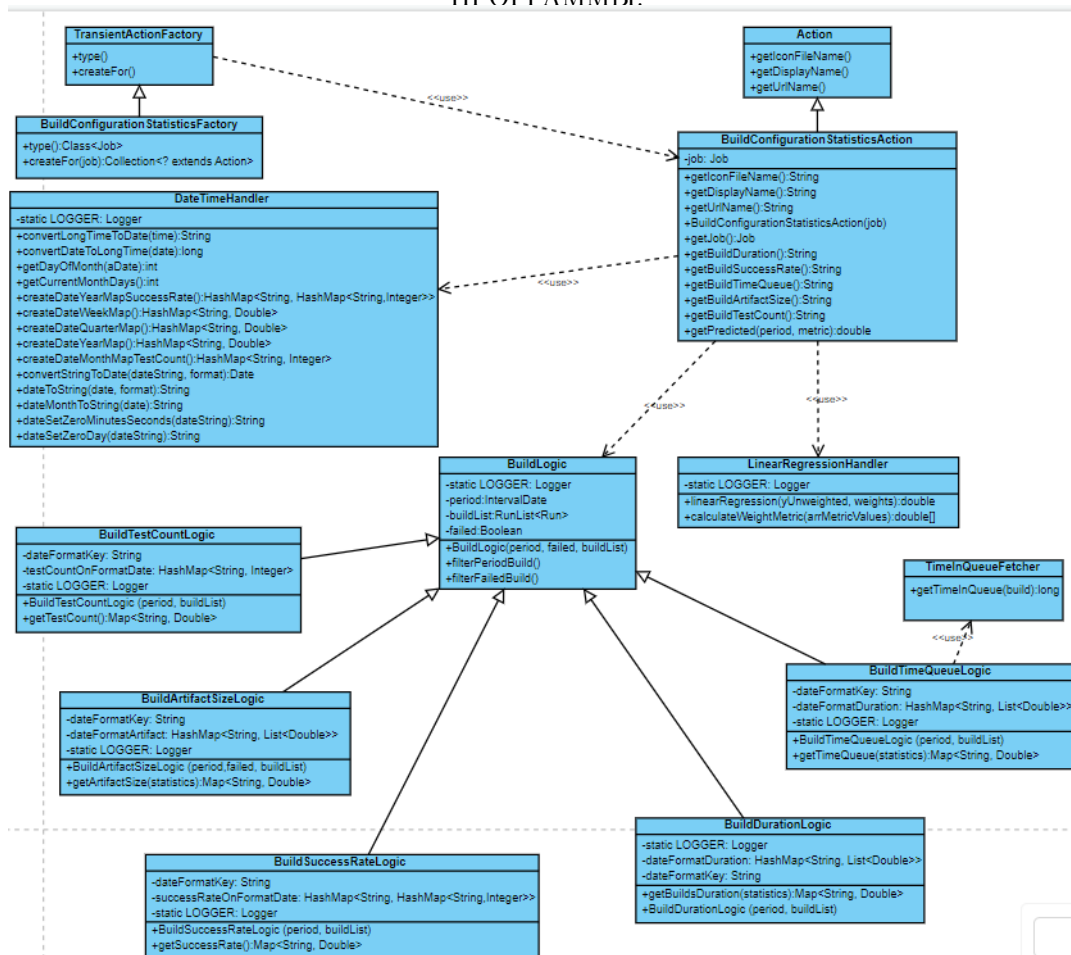


Рис. 3.

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.

### 3.3. 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 резуль-

татом). 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 рис.4 представлено изображение архитектуры плагина.

### 3.4. 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. Недостатком такого выбора является то, что абсолютное большинство плагинов написано на чистом Java, а значит сообщества и поддержка при разработке на Java будет значительно большей. Также в сравнении с Groovy, Java обладает большей производительностью [9], статической типизацией и подходит для разработки приложений в парадигме ООП.

Java будет использоваться для программирования ядра плагина и бизнес-логики. Также для программирования графических компонентов, графиков и диаграмм будет использоваться язык программирования JavaScript. Основное предназначение JS - 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, 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. В Jelly можно вызывать функции Java, использовать такие синтаксические конструкции,

как циклы, условия и переменные, также он позволяет легко обратиться к объектам в Java.

### 3.5. Инструменты сборки

В качестве инструмента сборки проекта был выбран Maven, который можно использовать для создания и управления любым проектом на основе Java.

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.

### 3.6. 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.

Для удобства разработки и взаимодействия клиентской части с серверной необходимо, чтобы обработанные данные из Jenkins в браузер передавались в формате JSON. Для обеспечения преобразования объектов и структур Java в JSON была использована библиотека Gson от компании google.

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

Для того чтобы протестировать разработанный плагин, будут написаны unit тесты. Поскольку разработка плагина будет вестись на языке Java, то для написания тестов будет использована Java инструмент. В качестве такого инструмента был выбран Junit.

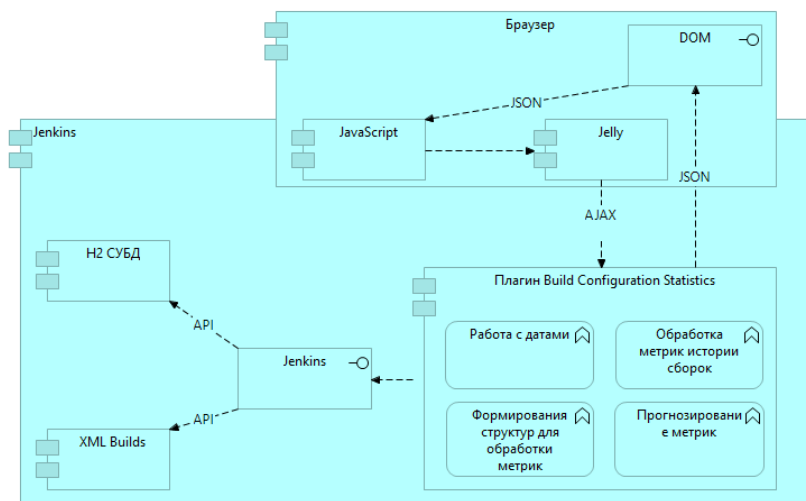


Рис. 4.

Поскольку функционал плагина использует обращение к структурам Jenkins (Job, Run), для тестирования необходимо использовать заглушки этих структур. Для работы с заглушками была выбрана библиотека *mockito*.

### 3.7. Выводы

В данном разделе 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.

## 4. РЕАЛИЗАЦИЯ ПРОТОТИПА ПЛАГИНА

### 4.1. Описание разработанных классов

В процессе написания кода плагина были запрограммированы классы в соответствии с диаграммой классов из раздела 3.

**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.

**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.

**IntervalDate.** Перечисляемый тип для удобства работы с датами-периодами.

**Statistics.** Перечисляемый тип для удобства работы с показателями статистики.

**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.

**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.

**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.

**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.

**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.

**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 сборки за определенный период.

**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.

**LinearRegressionHandler.** Класс для прогнозирования метрик следующей сборки с помощью линейной регрессии с весовыми коэффициентами.

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

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.

#### 4.2. Результаты разработки плагина

При разработке плагина надо было учитывать, что требуется отображать все графики на одной странице задания друг под другом, поскольку при выборе одного периода, например, месяца, будет получена сводная информация по каждой сборке или нескольких сборок запущенных в один день. Графики отображаются посредством перехода на соответствующую ссылку, оставляя при этом пользователя в том же задании (странице с результатами последних сборок). Интерфейс страницы плагина с графиками в системе Jenkins на странице задания показан на рис.5.

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

#### 4.3. Выводы

В разделе 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.

### 5. ТЕСТИРОВАНИЕ И АПРОБАЦИЯ ПЛАГИНА В JENKINS

#### 5.1. Методы тестирования

Тестирование программного обеспечения — обширное понятие, которое включает планирование, проектирование и, собственно, выполнение тестов [10]. 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

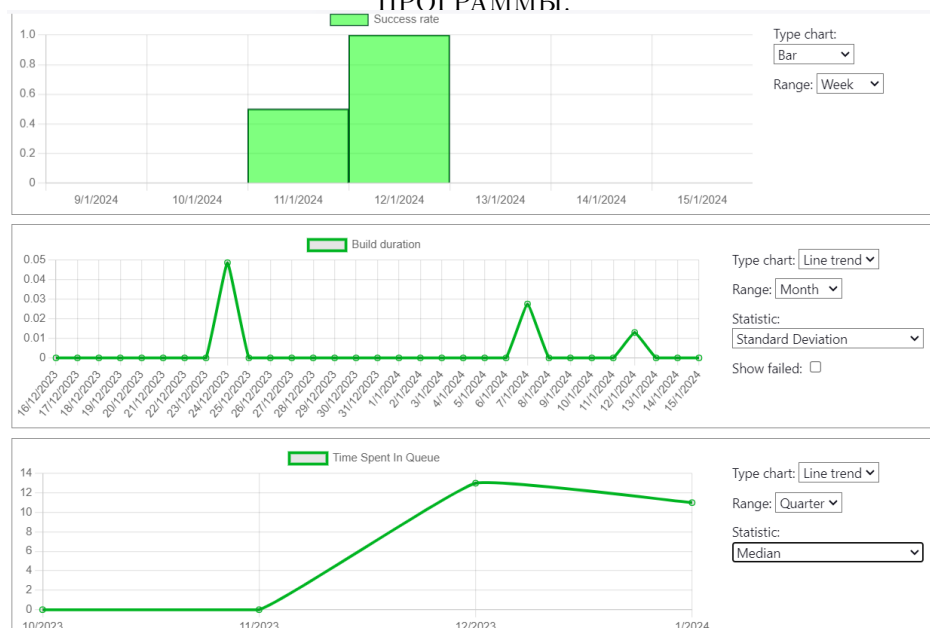


Рис. 5.

тестирования на соответствие исходным требованиям, которые были предъявлены к разработке в разделе 2.

Существует множество методов тестирования и техник тест дизайна, в процессе анализ функциональных требования были отобраны те, которые наиболее релевантны 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.

**Unit тестирование.** 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 сборки - `testGetTimeInQueue()`.

**BDD тестирование.** Помимо юнит тестов, были написаны автоматизированные тесты с использованием методологии Behavior-driven development (BDD) или разработки через поведение. В соответствии с BDD тесты следует писать на естественном, удобочитаемом языке, ориентированном на поведение приложения [11].

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

Тесты написанные с BDD подходом, проверяют работоспособность основного функционала плагина, а именно получения обработочной информации по метрикам сборки, в соответствии с

указанными настройками/фильтрами пользователя.

**UI тестирование.** Также для автоматизации тестирования UI части плагина, был применен Selenium web driver и язык программирования python. 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)`.

## 5.2. Апробация плагина

Апробация плагина будет проводится в системе CI Jenkins для которой и был разработан плагин визуализации. Для того чтобы про-

вести 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 [12] `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.

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.

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.

В ходе апробации плагина были сгенерированы сборки, которые отображены на рис.6.

Запуски были сгенерированы с разной датой начала:

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

Рис. 6.

- 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.

А также сборки, с созданными артефактами, часть из которых в статусе успешного выполнения, с короткой продолжительностью, а часть

из которых завершены падением с большой продолжительностью выполнения. Были определены разные файлы-артефакты для генерации с размером от 70 байт до 1 Кб. Перенос созданных файлов в артефакты выполнялся с помощью конфигурационного раздела в сборке Post-build Actions, в котором выплавлялась команда Archive the artifacts.

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

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 "tempbuild111111111111111111"1>tempbuild.txt*. А для генерация файлов в 1Кб команда *fsutil file createnew tem.txt 1048576*.

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 [13]. Этот продукт используется для:

- разделения фронтенд и бекенд сборок, сводя

количество взаимодействия между ними до минимума;

- использовать Node.js и его библиотеки в процессе сборки без глобальной установки Node/NPM;
- позволяет убедиться, что запущенные версии Node и NPM одинаковы в каждом окружении сборки.

У проекта 865 форков 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 увеличения, в возможно и уменьшения (в случае 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:

1. Сделать fork проекта в личный репозиторий.
2. Найти подходящий коммит, в котором были выполнены значительные изменения (более 500 строк измененного кода).
3. Откатиться до найденного коммита.
4. Создание новой ветки на основе коммита, до которого произошел откат.
5. Отправка ветки в личный удаленный репозиторий на GitHub.

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

Для отката к более ранним версиям был написан скрипт на языке Python.

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

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

Результаты работы плагина на описанном выше проекте отображены на рис.7-8. На рис.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 *prn* и *node*, что оптимизировало время выполнения сборки, которое уменьшилось с 19.3 до 18.3 секунд. А версия в 11 месяце, вероятно была не протестирована перед загрузкой в главную ветку, поскольку результат из 3 запусков сборки закончился падением. Что также видно на графике SR, т.е. можно сделать вывод что в коммите был дефект, а не оптимизация, которая ускорила время сборки в несколько раз.

На рис.8 можно увидеть с помощью радар-

Statistics for job test-Build-frontend-maven-plugin\_version



Рис. 7.

ной диаграммы общий размер, сгенерированных логов-артефактов за последний год. Видно, что с каждым месяцем размер артефактов увеличивался, что также можно объяснить увеличением исходного кода продукта, также наглядна видна разница 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 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.

Для того чтобы оценить результаты работы, будет проведено сравнение функционала, который присутствует в аналогичных решениях: плагинах, сравнительный анализ, которых проводился в разделе 2.3 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 аналогич-

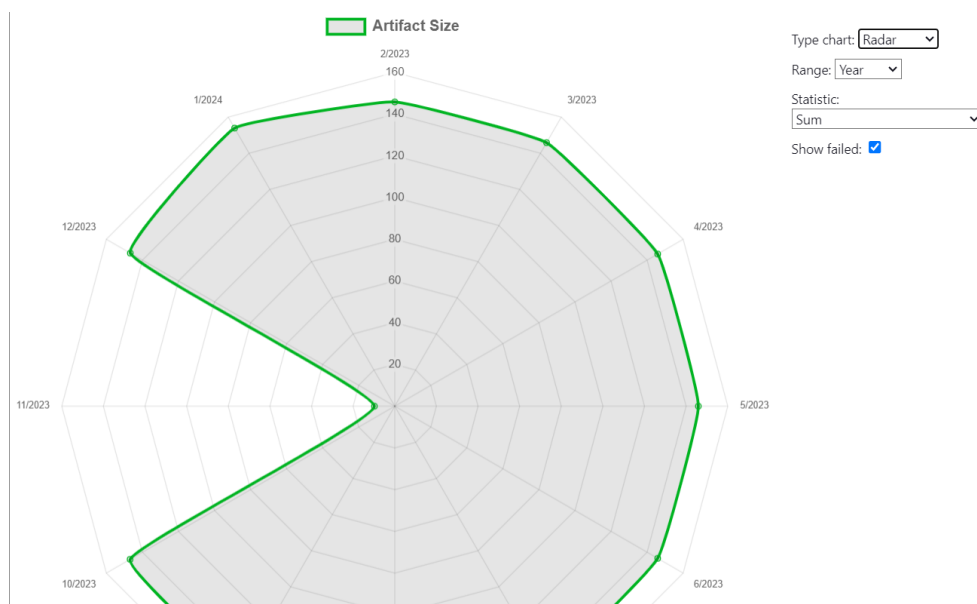


Рис. 8.

ных решений. Все результаты сравнения с аналогичными решениями приведены в табл.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 2.3 метрик, но их количество меньше 5.

### 5.3. Выводы

По описанным в разделе результат, можно прийти к выводу, что тестирование проведено успешно. 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.

Визуализация метрик со статистическими показателями была проверена на различных типах диаграмм/графиков: столбчатая диаграмма, линейный тренд, радарная диаграмма.

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

Работоспособность плагина проверена на проекте открытым с исходным кодом frontend-maven-plugin. В ходе апробации были получено

24 запуска сборок на 12 разных версий продукта, в том числе 2 упавших запуска на одной из версий. Также было написано 2 python скрипта для формирования окружения для апробации.

Плагин рекомендуется использовать для:

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

## 6. ЗАКЛЮЧЕНИЕ

Таблица 2.

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

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 реализации 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 форка).

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

По итогу реализации объем кода проекта составляет ~3220 строк, из которых:

- ~1680 строк - Java код на сервере Jenkins;
- ~780 строк - Jelly и JS на клиентской части;
- ~260 строк - Java unit тесты;
- ~210 строк - Java bdd тесты;
- ~180 строк - Python код UI тестов;
- ~85 строк - python скрипты для апробации.

#### СПИСОК ЛИТЕРАТУРЫ

1. Zubin Irani. 5 common pitfalls of ci/cd—and how to avoid them. *Cloud Networks*, 2017.
2. Что такое сборка в программировании. <https://uchet-jkh.ru/i/cto-takoe-sborka-v-programmirovanii>. [Accessed 20-11-2023].
3. Учимся создавать и настраивать jenkins jobs. <https://habr.com/ru/companies/slurm/articles/742504/>. [Accessed 20-11-2023].
4. Ci/cd. <https://blog.skillfactory.ru/glossary/ci-cd>. [Accessed 20-11-2023].
5. Александра Соколова. Подходит ci/cd вашему бизнесу: плюсы и минусы конвейеров. *Cloud Networks*, 2021.
6. Ci/cd tools comparison: Jenkins, teamcity, bamboo, travis ci, and more. <https://www.altexsoft.com/blog/cicd-tools-comparison/>. [Accessed 20-11-2023].
7. Ioannis Moutsatsos, Imtiaz Hossain, Claudia Agarinis, Fred Harbinski, Yann Abraham, Luc Dobler, Xian Zhang, Christopher Wilson, Jeremy Jenkins, Nicholas Holway, John Tallarico, and Christian Parker. Jenkins-ci, an open-source continuous integration system, as a scientific data and image-processing platform. *Journal of Biomolecular Screening*, 22:1087057116679993, 11 2016.
8. The architecture of jenkins plugins. <https://subscription.packtpub.com/book/programming/9781784390891/10/ch10lv11sec62/the-architecture-of-jenkins-plugins>. [Accessed 20-11-2023].
9. Victoria Puzhevich. Groovy vs java: Detailed comparison and tips on the language choice, 2020.
10. Тестирование ПО: суть профессии, требования и заработная плата. [https://habr.com/ru/companies/habr\\_career/articles/517812/](https://habr.com/ru/companies/habr_career/articles/517812/). [Accessed 22-12-2023].
11. Quick guide to bddmockito. <https://www.baeldung.com/bdd-mockito>. [Accessed 22-12-2023].
12. Abhishek Pathare. Tutorial: Developing complex plugins for jenkins, 2022.
13. frontend-maven-plugin. <https://github.com/eirslett/frontend-maven-plugin>. [Accessed 25-12-2023].