, Том 2025, № ?, 2025, стр. 1–15.

АНАЛИЗ ДАННЫХ???

Разработка плагина Jenkins для визуализации статистики по результатам сборки программы

А. Д. Кухто, В. А. Пархоменко,

Санкт-Петербургский политехнический университет Петра Великого, Санкт-Петербург, Россия

Поступила в редколлегию 31.07.2025 г.

Аннотация—В работе представлен плагин Jenkins для визуализации статистики по процессу сборки программы. Осуществлен сравнительный анализа аналогичных решений, выбор функционала, проектирование архитектуры плагина. Представлена реализация плагина с открытым исходным кодом. Для визуализации и обработки метрик сборок используются различные статистические показатели, типы диаграмм, а также анализ данных для прогнозирования значения метрик. Разработана методика создания наборов данных для тестирования плагинов визуализации метрик. Проведена иллюстрация работы плагина на реальном популярном проекте с открытым исходным кодом frontend-maven-plugin. Для создания массивов данных написан отдельный скрипт, которым сгенерировано 12 версий проекта на 12 месяцев года (для генерации сборок на год). Представлены результаты анализа проекта.

КЛЮЧЕВЫЕ СЛОВА: визуализация, статистика, Jenkins, CI/CD, плагин, сборка, метрики сборки.

1. ВВЕДЕНИЕ

Для упрощения процесса разработки программного обеспечения широко применяются практики DevOps, одной из которых является непрерывная интеграция, сборка и доставка (Continuous Integration, Continuous Delivery, далее — CI/CD). 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.

Одним из ведущих средств CI/CD является TeamCity компании Jetbrains. Главными недостатками TeamCity является то, что это платное решение, лицензия обходится ИТ компания достаточно дорого, а также недостатком является то, что компания Jetbrains покинула ИТ рынок России. 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 пользуется популярностью у разработчиков при локальной разработке решений с открытым исходным кодом.

На данный момент в Jenkins нет плагина, который полностью заменяет модуль визуализации статистики из TeamCity (далее — Statistics). Разрабатываемый плагин 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.

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

Фокусировка данной работы на изучение сборок в процессе CI/CD оправдана тем, что эта комбинация повышает качество кода. Так, например, в [1], в результате изучения более 12 тыс. репозиториев выяснено, что внедрение CI/CD подхода повышает количество выявленных проблем с проектами с одновременным увеличением скорости комитов.

Дальнейшая часть работы организована следующим образом. В разделе 2 представлен обзор источников и похожих плагинов в Jenkins в рамках визуализации статистики сборок, а в разделе 3 — спецификация задачи. В разделе 4 описана разработка плагина для Jenkins, а в разделе 5 — приложение для тестирования, включая методику создания наборов данных. В разделе 6 приводятся результаты работы плагина на примере популярного приложения frontend-maven-plugin.

2. ИССЛЕДОВАНИЕ АНАЛОГОВ

2.1. Обзор источников

Обзор исследований в области CI/CD проводился, в основном, в DBLP. По ключевым словам "CI/CD"найдено 114 работ. Далее опишем наиболее релевантные работы, посвященные исследованиям разных метрик сборок.

Одной из основных метрик является «воспроизводимость» сборки, которая заключается в устойчиво воспроизводимом формировании побитово идентичного бинарного исполняемого файла из исходного кода, что важно в том числе для безопасности кода. Авторы [2] выявили связь между качеством кода и воспроизводимостью сборки. В [3] исследованы 11,528 и 597,066 сборок Arch Linux и Debian пакетов, соответственно. На этих пакетах авторы выявили связь статуса воспроизводимости сборок с тремя внешними факторами. Построена таксономия из 16 причин невоспроизводимости. Показано, что статистически значимой причиной является архитектура аппаратного обеспечения. Также, с меньшим эффектом, показана зависимость от версий и зависимостей проекта.

В препринте [4] сформирован набор данных с проектами на Java. Он содержит 12,283 невоспроизводимых артефактов, для каждого из которых имеется ссылка на проект в Maven Central. Авторы создали таксономию причин невоспроизводимости, основываясь на ошибках в манифесте сборки, SBOM, файловой системе, JVM байткоде, свойствах версионирования и временных метках. Другим важным результатом работы является создания алгоритма по исправлению причин, в результате чего авторам удалось исправить 26.89% сборок в основном за счет исправления метаданных. В [5] CI/CD подход адаптирован для поиска воспроизводимых сборок как в качестве самостоятельного метода повышения безопасности программ, так и в рамках программно-аппаратных решений. Однако мы не нашли в данной работе достаточных подробностей для адаптации техники в Jenkins.

В [6] проведен анализ между успешностью сборки и такими социально-психологическими показателями поведения разработчиков как вежливость и эмоции, отражаемые в комментариях. На изучаемых наборах данных TravisTorrent и GitHubTorrent выяснено, что нет статистически значимой связи между вежливостью и успешностью сборки. В то же время злость имеет статистически значимое влияние на неуспешность сборок. Такие эмоции, грусть, любовь и радость не оказали значимого влияния в исследовании.

В [7] изучено 924 616 сборок из 588 проектов GitHub, связанных с Travis CI. В этой и более раннем исследовании [8] показано, что долгое время сборки приводит к ее прерываниям. В связи с этим авторы предлагают новую метрику «производительности» сборок. На исследуемом наборе данных показано, что с течением времени разработчики склонны жертвовать одним из показателей, когда невозможно достичь одновременно минимизации времени сборок, пре-

рываний и увеличения производительности. Обе работы предлагают техники для сокращения времени сборок.

2.2. Похожие плагины Jenkins

Основным критерием для сравнения средств визуализации информации о сборках является поддержка метрик оценок сборок. Наиболее популярными метриками являются следующие:

- *Success Rate* (далее — SR) — процент успешности сборок;
- *Build Duration* (далее — BD) — время выполнения сборок;
- *Time Spent in queue* (далее — TQ) — время проведенное в очереди сборок;
- *Test Count* (далее — TC) — количество выполненных тестов в сборке;
- *Artifacts Size* (далее — AS) — размер созданных во время сборок артефактов.

В Jenkins существует ряд плагинов, похожих на Statistics. Результаты их сравнения приведены в таблице 1, где также отмечена часть функционала разрабатываемого решения.

Таблица 1. Сравнительный анализ плагинов Jenkins

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	-	+	-	+

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.

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

3. ПОСТАНОВКА ЗАДАЧИ

В предыдущем разделе мы частично специфицировали требования к разрабатываемому плагину. Поскольку он является аналогом модуля Statistics, который выбран эталоном в силу признания на рынке, то функционал должен также покрывать функционал этого модуля. Рассмотрим совокупность требований подробнее.

1. Должна производиться визуализация указанных в предыдущем разделе метрик сборок с помощью графиков и диаграмм (столбчатые, линейные тренды, круговые).
2. Для указанных метрик должны визуализироваться следующие статистические показатели:
   - среднее арифметическое;
   - мода;
   - медиана;
   - размах;
   - среднеквадратическое отклонение;
   - среднеквадратическое отклонение несмещенное;
   - дисперсия.
3. Для метрики BD 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.
4. Для метрики TQ должно также рассчитываться среднее время, вычисляемое аналогично Build Duration.
5. Для метрики TC должно также рассчитываться количество выполненных тестов в упавших сборках, если таковые успели выполниться.
6. Для метрики AS должен также рассчитываться средний размер за определенный интервал времени, а также учет артефактов, которые успели создаться в сборках до падения.
7. 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. В частности, если 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 должна быть 28/31 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.
8. 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.
9. По метрикам BD, AS, TQ должна быть возможность выбрать статистический показатель, в соответствии с которым должна производить обработка итоговых значений.
10. Составление прогноза, 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.

3.1. Спецификация вариантов использования

Диаграмма вариантов использования, показывающая функционал плагина отображена на рис. 1. На данной диаграмме основное внимание уделяется процессу визуализации статистики метрик сборок. Основное действующее лицо одно — пользователь системы, который запускает сборки и работает в CI системе. Им может выступать любой участник команды, который задействован в разработке, тестировании, доставке и внедрению приложения.

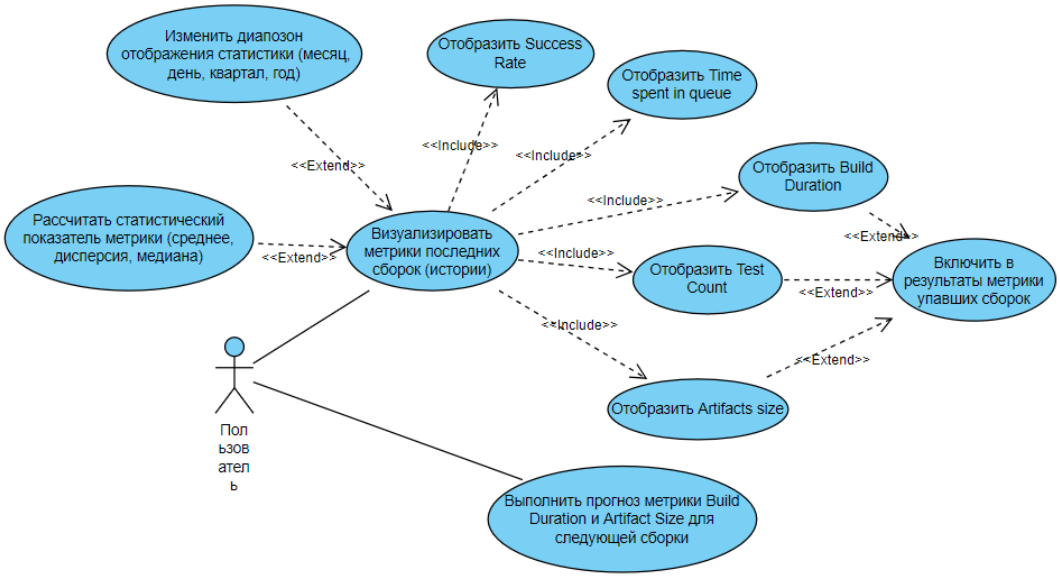

Рис. 1. Спецификация вариантов использования разрабатываемого плагина

4. РАЗРАБОТКА ПЛАГИНА JENKINS

После того, как определены основные требования к разрабатываемому плагину, приступим к функциональному моделированию плагина и выбору основного стека технологий, которые обсуждаются в подразделах 4.1 и 4.2, соответственно. Далее в подразделах 4.3, 4.4 и 4.5 обсуждается общая клиент-серверная архитектура, структура классов и реализация плагина, соответственно.

4.1. Проектирование функциональной модели плагина

Функциональная модель плагина в нотации archimate отображена на рис. 2. Основное внимание на диаграмме уделяется визуализации статистики сборок, поскольку это изначально является целью разработки. Также на диаграмме отражены дополнительные функции такие как фильтрация, и высчитывание статистик метрик.

4.2. Выбор средств реализации

Поскольку Jenkins написан на Java, то почти все плагины написаны на этом языке, кроме небольшого числа на Groovy. Однако мы решили не использовать данный язык. Java использован для программирования ядра плагина и бизнес-логики. Для реализации графических компонентов, графиков и диаграмм применен JavaScript. Помимо прочего, для стилизации

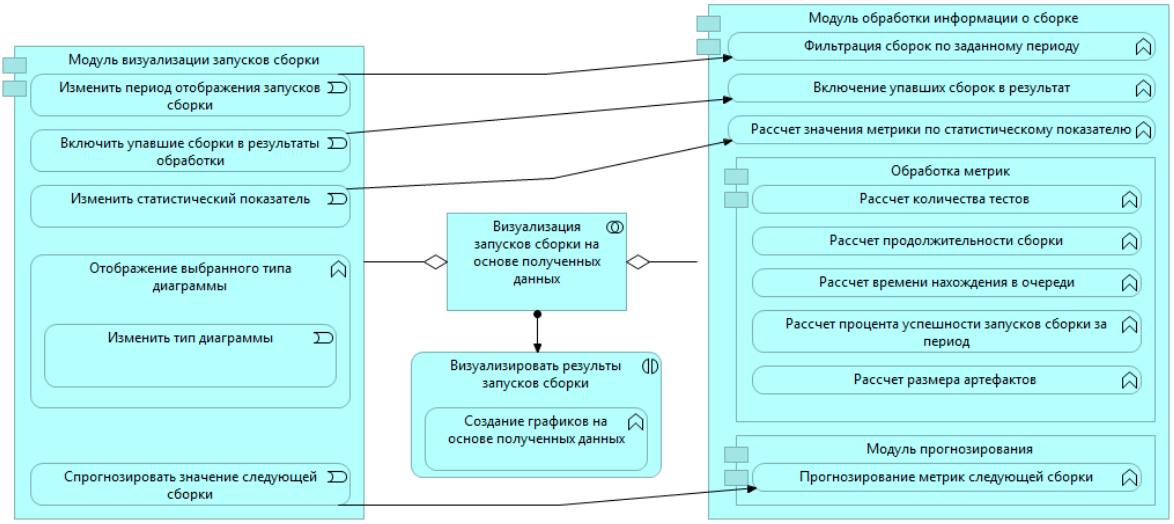

Рис. 2. Процесс визуализации сборок

компонентов веб-интерфейса использован язык каскадных таблиц стилей CSS, который позволяет настроить удобное отображение и позиционирование элементов на странице плагина Jenkins.

Верстка страниц осуществлена 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.

В качестве инструмента сборки проекта выбран Maven, поскольку абсолютное большинство разработанных плагинов для Jenkins использует Maven. Данное средство предоставляет удобные архетипы для начала разработки плагинов, что делает использование того же Gradle не рациональным.

Для работы с графиками в Jenkins и Java 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 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 — библиотека легких, автономных математических и статистических компонентов, решающая наиболее распространенные проблемы, связанные со статистикой. В экспериментальных целях решено ограничить метод для прогнозирования метриками BD и AS для следующей сборки (одного месяца), при этом используется линейная регрессия с весовыми коэффициентами.

4.3. Проектирование архитектуры плагина

На рис. 3 представлена диаграмма с архитектурой плагина.

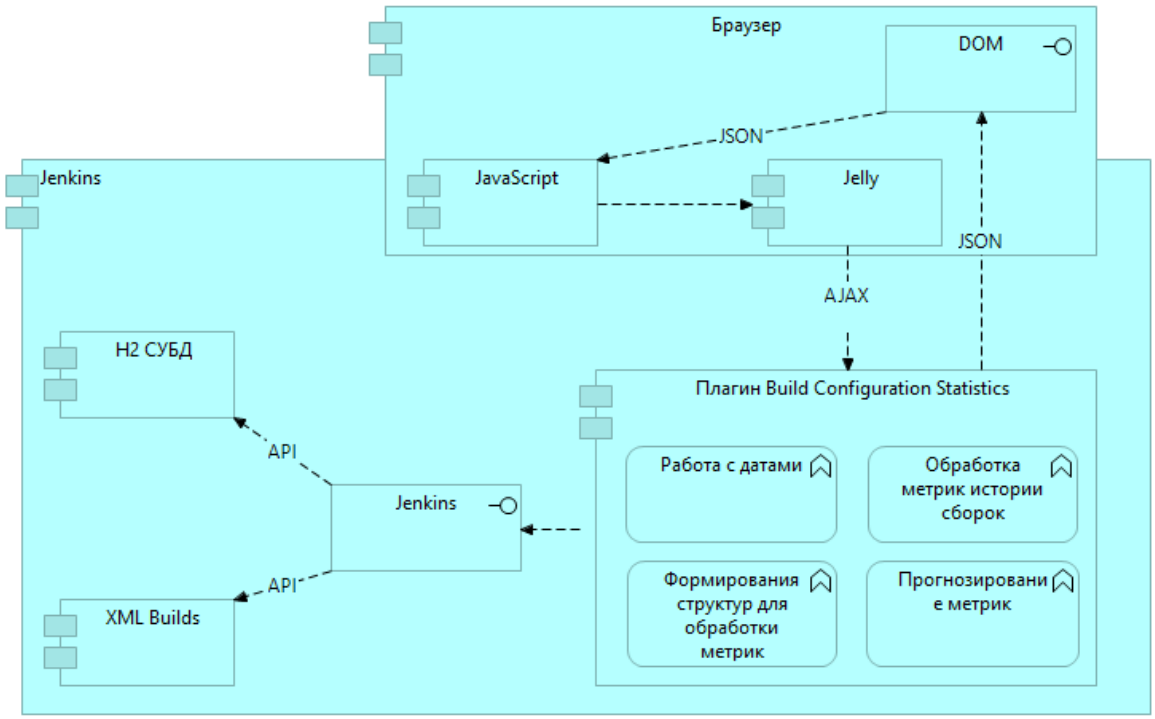

Рис. 3. Архитектура плагина

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.

4.4. Проектирование классов плагина

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

Все классы плагина отображены на рис. 4. 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. Оба перечисленных класса встроены в Jenkins.

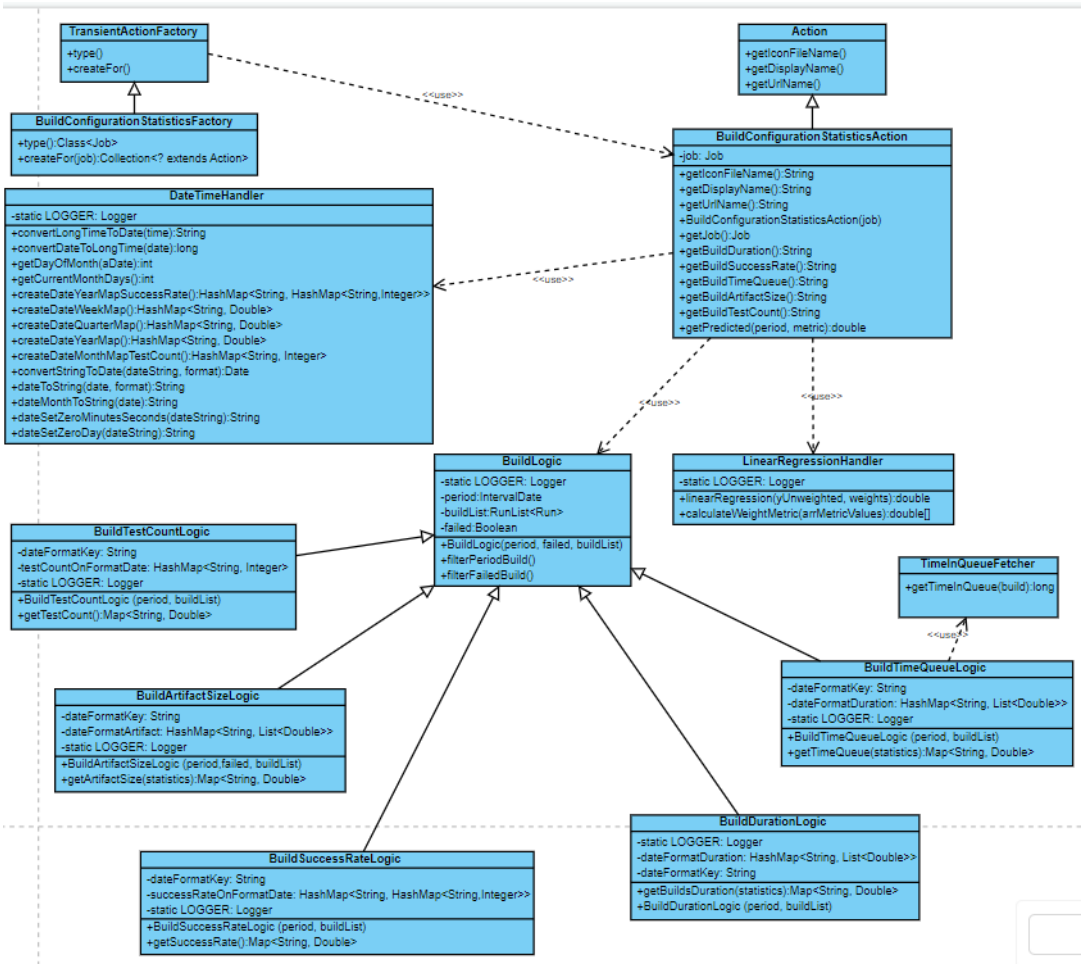

Рис. 4. Диаграмма классов плагина

Основная часть остальных классов требуется для работы с определенной метрикой статистики выполнения сборок Jenkins, что следует из их названия. Также разработан дополнительный класс *DateTimeHandler*, который позволяет 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.

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.

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 сборок 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.

4.5. Реализация и тестирование плагина

Спроектированный плагин реализован, функционал серверной части протестирован с помощью модульных тестов на Junit и Mockito. 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().

Для проверки пользовательской части приложения использован Selenium web driver и язык программирования Python. WebDriver управляет браузером, как это делает пользователь, с ис-

пользованием сервера Selenium. Код тестов расположен в отдельном проекте в классе TestCase, в котором 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).

Плагин размещен в открытом репозитории по ссылке [9]. Интерфейс страницы плагина с графиками в системе Jenkins на странице задания показан на рис. 5.

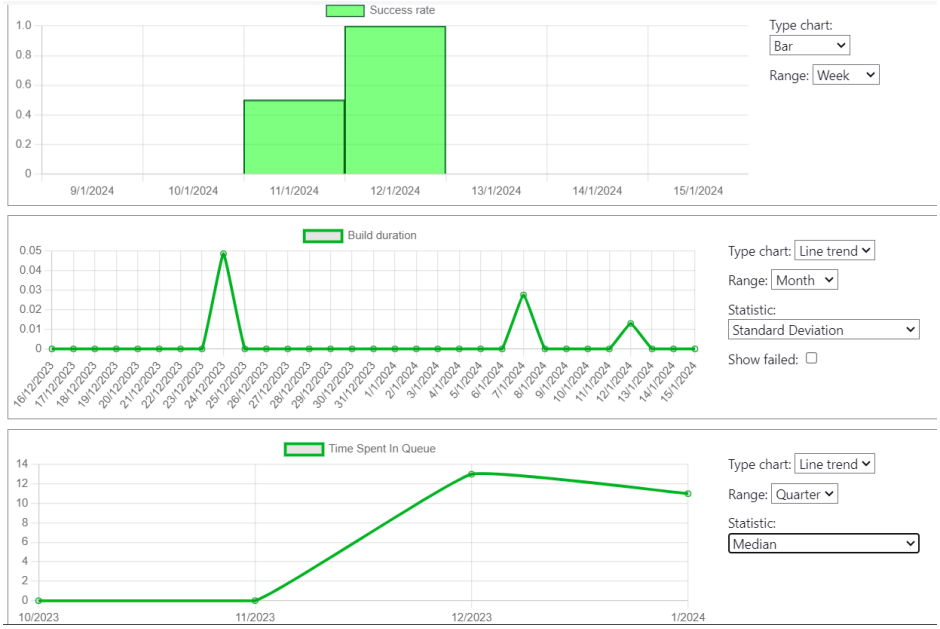

Рис. 5. Интерфейс плагина Jenkins

5. ПРИЛОЖЕНИЕ ДЛЯ ТЕСТИРОВАНИЯ И НАБОРЫ ДАННЫХ

В иллюстративных целях проведено тестирование на стороннем проекте. В качестве такого проекта выбран frontend-maven-plugin [10]. На момент подачи статьи у проекта свыше 880 форков на GitHub, а также более 4,3 тысяч звезд, из чего следует, что его разработка была полезна для ИТ сообщества и активно используется разработчиками.

Это плагин, который загружает/устанавливает Node и NPM локально для проекта разработчика, запускает npm install, а затем любую комбинацию Bower, Grunt, Gulp, Jspm, Karma или Webpack и может работать в Windows, OS X и Linux [10]. Этот продукт используется для:

– разделения фронтенд и бекенд сборок, сводя количество взаимодействия между ними до минимума;
– использования Node.js и его библиотеки в процессе сборки без глобальной установки Node или NPM;
– проверки, что запущенные версии Node и NPM одинаковы в каждом окружении сборки.

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.

Далее рассмотрим процесс создания набора данных, основываясь на изучаемой программе.

5.1. Создание набора данных о метриках сборок проекта

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 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, совершены шаги следующей методики:

1. Создание копии проекта в личный репозиторий.
2. Поиск подходящего коммит, в котором были выполнены значительные изменения (более 500 строк измененного кода).
3. Откат версии проекта 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 шаги методики начиная с коммита, до которого был произведен откат. Алгоритм повторен 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 также размещенный в репозитории [9].

Далее при генерации сборок 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 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 отредактированы по месяцам года (одна версия — один месяц), а не по реальным датам коммитов в репозитории. Рекомендуется смотреть на этап жизненного цикла продукта и проводить визуализацию по кварталам или месяцам.

Сгенерированные на этом проекте сборки отображены на рис. 6.

6. РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТОВ

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

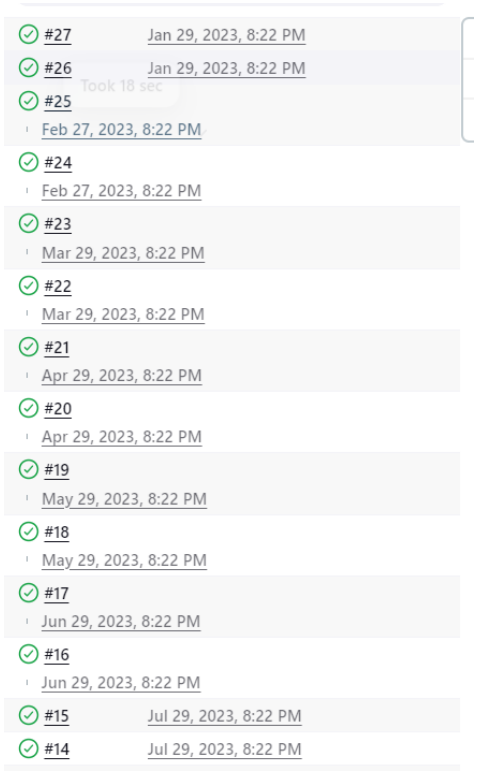

Рис. 6. Фрагмент сгенерированных сборок реального проекта

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 протестирована перед загрузкой в главную ветку, поскольку результат из 3 запусков сборки закончился падением. Это 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 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 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

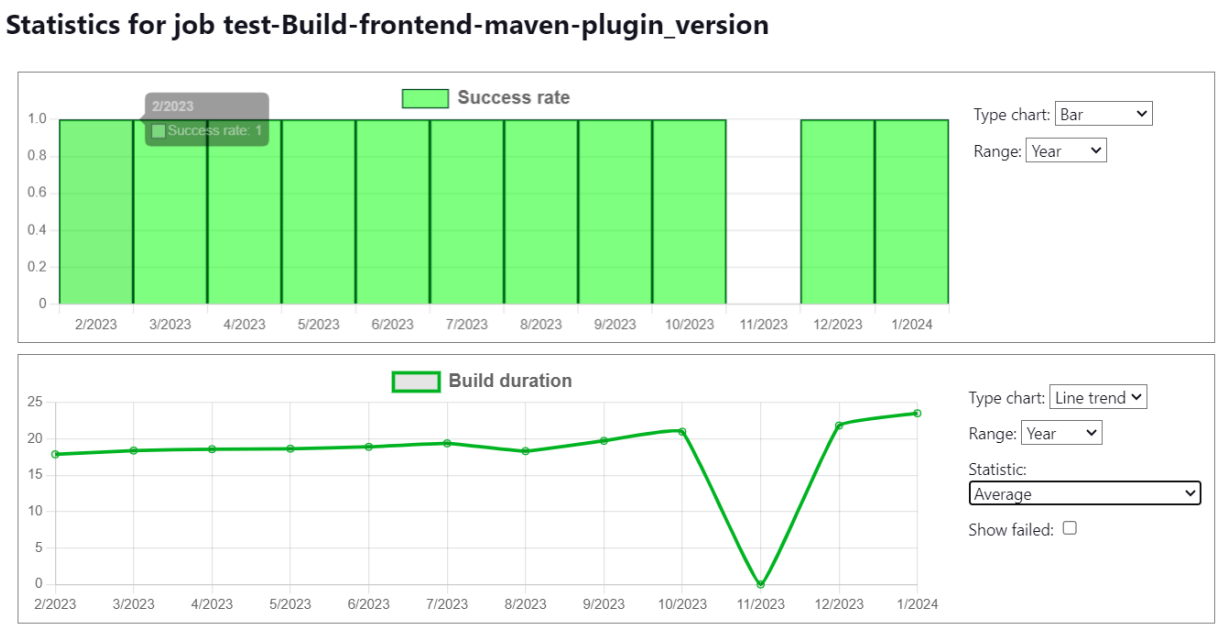

Рис. 7. Результаты апробации на реальном проекте, метрики SR, BD

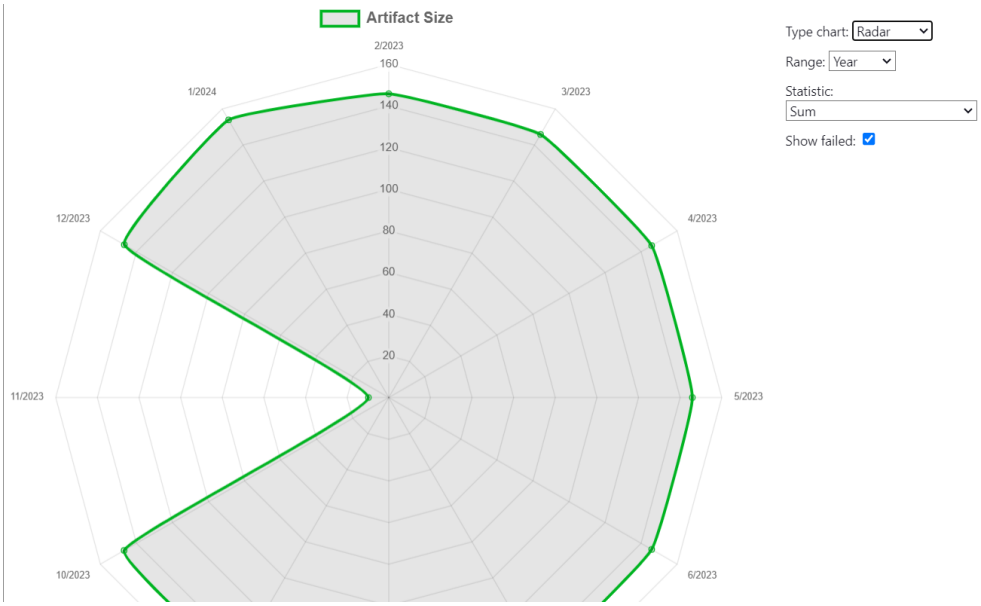

Рис. 8. Результаты апробации на реальном проекте, метрика AS

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, проведено сравнение функционала, который присутствует в аналогичных решениях: плагинах, сравнительный анализ, которых проводился в подразделе 4.2 и модуль Statistics, реализованный в средстве 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. Статистики отображаются 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. Результаты сравнения с аналогичными решениями приведены в табл.2.

Таблица 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

Если сравнивать по количеству визуализируемых метрик, то видно, что все 5 метрик, которые реализованы в TeamCity, удалось реализовать и в разработанном плагине, аналогичные плагины Jenkins визуализируют определенные из перечисленных в подразделе 4.2 метрик, но их количество меньше 5.

ЗАКЛЮЧЕНИЕ

В результате проведенной работы разработан прототип плагина для визуализации статистики сборок Jenkins, Проанализирована предметная область, проведен сравнительный анализ аналогичных решений. После выбора средств и инструментов 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. Проведено тестирование серверной и пользовательской частей плагина. Для иллюстрации работоспособности проведено тестирование на реальном популярном проекте frontend-maven-plugin.

Разработана методика создания наборов данных для тестирования плагинов визуализации метрик. Для поддержки автоматизации выполнения ее шагов написано 2 Python скрипта. Подготовлены данные по 24 запускам сборок на 12 разных версий продукта. Осуществлен подробный анализ динамики анализа сборок frontend-maven-plugin. В частности, версия проекта в 11 месяце, скорее всего, не была протестирована перед загрузкой в главную ветку, поскольку результат из 3 запусков сборки закончился падением, что видно на графиках SR, BD и AS.

Разработанный плагин для CI/CD платформы Jenkins и скрипты для поддержки методики создания наборов данных размещены в открытом доступе по ссылке [9]. Таким образом, разработанный плагин позволяет поддерживать всесторонний анализ сборок проектов в процессе непрерывной интеграции, превышающий по функциональности аналоги, включая Statistics Teamcity.

СПИСОК ЛИТЕРАТУРЫ

1. Fairbanks J. , Tharigonda A., Eisty N. U., Analyzing the Effects of CI/CD on Open Source Repositories in GitHub and GitLab, IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA), 2023, pp. 176-181.
2. Butler S., Gamalielsson J., Lundell B. et al., On business adoption and use of reproducible builds for open and closed source software, Software Qual. J., 2023, vol. 31, pp. 687–719.
3. Bajaj R., Fernandes E., Adams B. et al., Unreproducible builds: time to fix, causes, and correlation with external ecosystem factors, Empir. Software Eng., 2024, vol. 29, art. num. 11.
4. Sharma A., Baudry B., Monperrus M., Canonicalization for Unreproducible Builds in Java. CoRR abs/2504.21679, 2025.
5. Marcela S. M., Kimes C., Auditing the CI/CD Platform: Reproducible Builds vs. Hardware-Attested Build Environments, Which is Right for You? In Proceedings of the 2024 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED '24), ACM, 2024, pp. 43–44.
6. Ortu M. et.al., Angry-builds: an empirical study of affect metrics and builds success on github ecosystem, In Proceedings of the 19th International Conference on Agile Software Development: Companion (XP '18), ACM, 2018, art. 35, pp. 1–2.
7. Ghaleb T. A., Hassan S., Zou Y., Studying the Interplay Between the Durations and Breakages of Continuous Integration Builds, IEEE Transactions on Software Engineering, 2023, vol. 49, no. 4, pp. 2476-2497
8. Ghaleb T.A., da Costa D.A., Zou Y., An empirical study of the long duration of continuous integration builds, Empir Software Eng, 2019, vol. 24, pp. 2102–2139.
9. Kuhto A.D, Parkhomenko V.A., A plugin for build analysis in Jenkins, `https://github.com/Software-Analysis-Community/buildConfigurationStatistics`
10. Frontend maven plugin, `https://github.com/eirslett/frontend-maven-plugin`

Paper Title

I. I. Ivanov , P. P. Petrov, S. S. Sidorov

Abstract.

KEYWORDS: Keyword 1, keyword 2.