АО «НИЦ СПб ЭТУ»

Утвержден
СМК 7.3.2-06ПР-ЛУ

ДОКУМЕНТИРОВАННАЯ ПРОЦЕДУРА

Разработка программного обеспечения
Порядок выполнения

СМК 7.3.2-06ПР

Подпись и дата	
Инв. № дубл.	
Взам. инв. №	
Подпись и дата	
Инв. № подл.	

Введена в действие приказом от 15.10.2024 № 101/1-О
Всего листов (страниц) 37
Экземпляр №

Содержание

1 Область применения

Настоящая документированная процедура устанавливает порядок выполнения каждого процесса разработки программного обеспечения.

Настоящая документированная процедура направлена на выполнение требований ГОСТ Р ИСО 9001-2015 (8.3), ГОСТ РВ 0015-002-2020 (8.3), ОСТ 134-1028-2012 изм. 2 (8.3) и развивает положения документа СМК 4.2.2-01РК-2006.

Настоящая документированная процедура распространяется на порядок разработки программного обеспечения, разрабатываемого акционерным обществом «Научно-инженерный центр Санкт-Петербургского электротехнического университета» (далее – АО «НИЦ СПб ЭТУ»).

Настоящая документированная процедура обязательна для применения во всех подразделениях АО «НИЦ СПб ЭТУ» при выполнении работ по контрактам (договорам) на выполнение опытно-конструкторской работы (составной части опытно-конструкторской работы), научно-исследовательской работы (составной части научно-исследовательской работы), в проектах по выполнению работ и оказанию услуг.

2 Нормативные ссылки

В настоящей документированной процедуре использованы нормативные ссылки на следующие стандарты и документы системы менеджмента качества:

ГОСТ 19.101-2024 Виды программ и программных документов

ГОСТ 28806-90 Качество программных средств. Термины и определения

ГОСТ Р 2.005-2023 Единая система конструкторской документации. Термины и определения

ГОСТ Р 56136-2014 Управление жизненным циклом продукции военного назначения. Термины и определения

ГОСТ Р 56862-2016 Система управления жизненным циклом. Разработка концепции изделия и технологий. Термины и определения

ГОСТ Р 56939-2024 Разработка безопасного программного обеспечения. Общие требования

ГОСТ Р 57100-2016 Системная и программная инженерия. Описание архитектуры

ГОСТ Р ИСО 9001-2015 Системы менеджмента качества. Требования

ГОСТ РВ 0015-002-2020 Система разработки и постановки на производство военной техники. Системы менеджмента качества. Требования

ОСТ 134-1028-2012 изм. 2 Ракетно-космическая техника. Требования к системам менеджмента качества предприятий, участвующих в создании, производстве и эксплуатации изделий

СМК 4.2.2-01РК-2006 Руководство по качеству

СМК 4.2.3-01ПР Управление документацией системы менеджмента качества

СМК 7.3.0-05ПР Разработка программной документации. Порядок выполнения

СМК 7.3.1-02ПР Управление проектной деятельностью

СМК 7.3.2-01РИ Разработка программного кода. Требования к формату

СМК 7.3.2-02ПР Определение и анализ требований к изделию

СМК 7.3.2-04РИ Разработка программных средств. Требования к разработке, содержанию и оформлению программных средств, подлежащих сертификации

СМК 7.3.5-02ПР Тестирование программного обеспечения. Порядок выполнения

СМК 7.3.2-10ПР Проектирование архитектуры программного обеспечения

СМК 7.3.3-01РИ Центральный репозиторий. Регламент работы

СМК 7.3.4-06РИ Разработка программного кода. Порядок проведения инспекции кода

СМК 7.5.3-03РИ Именование дистрибутивов. Требования к формату

П р и м е ч а н и е – При пользовании настоящим стандартом целесообразно проверить действие ссылочных стандартов (классификаторов) в информационной системе общего пользования – на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет или по ежегодному информационному указателю «Национальные стандарты», который опубликован по состоянию на 1 января текущего года, и по выпускам ежемесячного информационного указателя «Национальные стандарты» за текущий год. Если на документ дана недатированная ссылка, то следует использовать документ, действующий на текущий момент, с учетом всех внесенных в него изменений. Если заменен ссылочный документ, на который дана датированная ссылка, то следует использовать указанную версию этого документа. Если после принятия настоящего документа в ссылочный документ, на который дана датированная ссылка, внесено изменение, затрагивающее положение, на которое дана ссылка, то это положение применяется без учета данного изменения. Если ссылочный документ отменен без замены, то положение, в котором дана ссылка на него, применяется в части, не затрагивающей эту ссылку.

3 Термины и определения

В настоящей документированной процедуре применены следующие термины с соответствующими определениями:

3.1 **артефакт:** По СМК 7.3.3-01РИ (статья 3.2).

3.2 **архитектура (системы):** Основные понятия или свойства системы в окружающей среде, воплощенной в ее элементах, отношениях и конкретных принципах ее проекта и развития [ГОСТ Р 57100, статья 3.2].

3.3 **группа управления проектом:** По СМК 7.3.1-02РИ (статья 3.15).

3.4 **диаграмма потоков данных:** Визуальное представление перемещения данных в пределах процесса или системы.

3.5 **дистрибутив:** По СМК 7.3.0-05ПР (статья 3.1).

3.6 **жизненный цикл:** Совокупность явлений и процессов, повторяющаяся с периодичностью, определяемой временем существования типовой конструкции изделия от ее замысла до утилизации или конкретного экземпляра изделия от момента завершения его производства до утилизации [ГОСТ Р 56136, статья 3.16].

3.7 **заинтересованные лица:** Группа людей, которая задействована в процессе разработки программного изделия, может влиять на процесс или результат разработки, или на которую влияет процесс или результат разработки (заказчик, пользователи, эксперты и консультанты, проектная команда, соисполнители).

3.8 **изделие:** Предмет или набор предметов производства, подлежащих изготовлению в организации по конструкторской документации [ГОСТ Р 2.005, статья 1].

3.9 **итерация:** Законченный цикл разработки, приводящий к выпуску конечного программного продукта или некоторой его сокращенной версии, которая расширяется от итерации к итерации, чтобы, в конце концов, стать законченным изделием.

3.10 **программа:** Совокупность команд и данных, обеспечивающая выполнение заданной последовательности действий средствами вычислительной техники [ГОСТ 19.101, статья 3.1].

3.11 **программная документация:** Документы, содержащие сведения, необходимые для разработки, изготовления, сопровождения и эксплуатации программы [ГОСТ 19.101, статья 4.1].

3.12 **программное изделие:** Программный комплекс и/или программный компонент (библиотека), набор программных комплексов и/или программных компонентов (библиотек) с описанной функциональностью, прошедший испытания, состоящий из дистрибутива программного изделия и соответствующего комплекта программной документации.

3.13 **программное обеспечение:** Совокупность программ, позволяющая организовать решение задач на персональной электронно-вычислительной машине.

3.14 **программный комплекс:** Программа, состоящая из двух или более программных компонентов и/или программных комплексов, выполняющих взаимосвязанные функции [ГОСТ 19.101, статья 3.3].

3.15 **программный компонент:** Программа, рассматриваемая как единое целое, выполняющая законченную функцию [ГОСТ 19.101, статья 3.3].

3.16 **программный продукт:** Программное средство, предназначенное для поставки, передачи, продажи пользователю [ГОСТ 28806, статья 3].

3.17 **проект:** По СМК 7.3.1-02РИ (статья 3.57).

3.18 **разработка:** Стадия жизненного цикла изделия, на которой выполняются проектирование конструкции изделия, изготовление и испытания опытных образцов, технологическая подготовка производства [ГОСТ Р 56136, статья 3.21].

3.19 **системный аналитик проекта:** По СМК 7.3.1-02ПР.

3.20 **системный архитектор проекта:** По СМК 7.3.1-02ПР.

3.21 **тестирование:** Процесс, в ходе которого изделие (система, комплекс, программный комплекс и т. д.) подвергается проверке в контролируемых условиях, результаты регистрируются (фиксируются), и производится оценка какого-либо аспекта изделия или его части с целью подтверждения соответствия предъявляемым (заранее определённым) требованиям и выявления имеющихся дефектов (несоответствий, ошибок).

3.22 **уязвимость программы:** Недостаток программы, который может быть использован для реализации угроз безопасности информации (уязвимость программы может быть результатом ее разработки без учета требований по обеспечению безопасности информации или результатом наличия ошибок проектирования или реализации) [ГОСТ Р 56939, статья 3.19].

3.23 **хранилище данных:** Источник, приемник или промежуточное хранилище данных внутри моделируемой системы.

3.24 **эксплуатационные документы:** Документы, содержащие сведения для обеспечения функционирования и эксплуатации программы [ГОСТ 19.101, статья 4.2].

3.25 **docker-контейнер:** Среда выполнения со всеми необходимыми компонентами, такими как код, зависимости и библиотеки, которые необходимы для запуска кода программы в среде контейнеризации.

3.26 **docker-образ:** Отдельный исполняемый файл, используемый для создания контейнера.

3.27 **OVAL-описание:** Открытый язык описания и оценки уязвимостей.

3.28 **SVN:** Централизованная система управления версиями.

4 Сокращения

В настоящей документированной процедуре приняты следующие сокращения:

АРМ – автоматизированное рабочее место;
БД – база данных;
ИБП – источник бесперебойного питания;
ОС – операционная система;
ОС СН – операционная система специального назначения;
ПД – программная документация;
ПО – программное обеспечение;
ПЭВМ – персональная электронно-вычислительная машина;
СМК – система менеджмента качества;
ФСТЭК – Федеральная служба по техническому и экспортному контролю;
AD – Active Directory;
CI – Continious Integration;
CD – Continious Delivery;
CVE – Common Vulnerabilities and Exposures;
DSL – Domain Specific Language;
SCAP – Security Content Automation Protocol.

5 Ответственность

Начальник департамента программного обеспечения несет ответственность за своевременное и качественное выполнение работ подчиненных подразделений, участвующих в разработке ПО.

Начальник департамента управления требованиями и бизнес-анализа несет ответственность за своевременное и качественное выполнение работ подчиненных подразделений, участвующих в разработке ПО.

Главный системный архитектор несет ответственность за соблюдение требований настоящей документированной процедуры в целом.

Системный архитектор проекта несет ответственность за соблюдение требований настоящей документированной процедуры.

Системный аналитик проекта несет ответственность за соблюдение требований настоящей документированной процедуры в части своевременной разработки, фиксации в соответствующих местах хранения артефактов архитектуры, аналитики и разработки ПО.

Ответственность должностных лиц, участвующих в разработке ПО, приведена в 6.1.3.

6 Разработка программного обеспечения

6.1 Общие положения

6.1.1 Принципы итеративной разработки

6.1.1.1 В качестве основной модели процесса разработки используется итеративный процесс разработки программного продукта. Жизненный цикл программного продукта в выбранной модели разработки состоит из серии относительно коротких итераций.

Итерация включает, как правило, задачи:

– планирование работ;

– анализ;

– проектирование;

– реализация (разработка);

– тестирование;

– оценка достигнутых результатов.

Соотношение этих задач может существенно меняться в зависимости от приоритетов фазы разработки. Таким образом, соотношение различных задач в итерации лежит в основе формирования фазы разработки путем объединения группы итераций. Каждая фаза разработки, таким образом, имеет свои специфические цели в жизненном цикле программного продукта и считается выполненной, когда эти цели достигнуты.

Каждый цикл итерации начинается с планирования того, что должно быть выполнено. Все итерации, кроме итераций начальной (исследовательской) фазы и первых итераций фазы технического проекта, завершаются созданием функционирующей версии разрабатываемой системы. В конце цикла итерации также проводится оценка выполненных работ и того, были ли достигнуты цели.

В первой исследовательской фазе основное внимание уделяется задачам анализа. В итерациях второй фазы – техническом проекте – внимание уделяется проектированию и опробованию ключевых проектных решений. В третьей фазе – фазе разработки конструкторской документации или конструирования наиболее велика доля задач разработки и тестирования. Последняя фаза – передача в эксплуатацию – в наибольшей степени решает задачи обучения и технической поддержки.

Преимущества выбранной модели разработки заключаются в следующих аспектах:

– в итерационном процессе все итерации, составляющие фазу разработки, нацелены на последовательное осмысление стоящих проблем, наращивание эффективности решений и снижение риска потенциальных ошибок в проекте;

– при таком подходе можно гибко учитывать новые требования или производить тактические изменения в целях;

– такой подход позволяет выявлять проблемы и разрешать их на самых ранних этапах разработки, что связано с меньшими затратами.

Итерационный процесс предполагает разработку, реализацию и тестирование архитектуры на самых ранних этапах разработки. Такой подход позволяет устранять самые опасные риски, связанные с архитектурой, на ранних стадиях разработки, и избегать существенных переработок программного продукта на поздних стадиях.

6.1.1.2 В зависимости от этапа и специфики проекта применение итеративного подхода может иметь различные особенности в рамках проектной команды и, в основном, укладывается в два основных подхода.

Первый подход – явное выделение итераций:

– выбирается длина итерации (например, две недели);

– оценивается мощность группы разработки в человеко-днях на итерацию;

– назначается ежеитерационное совещание-планёрка, в ходе которой происходит планирование будущей итерации;

– составляется и постоянно поддерживается в актуальном состоянии список задач (бэклог), который формируется на основе требований заказчика, но может включать и вспомогательные задачи. Задачи должны быть упорядочены по приоритету. Приоритет

определяет заказчик или его представитель. Трудоёмкость задач в бэклоге должна быть оценена. Для оценки трудоемкости задач применяются различные подходы, например, покер планирования (англ. Planning Poker, а также англ. Scrum poker);

– при планировании итерации происходит выбор наиболее приоритетных задач из бэклога с учётом мощности группы разработки. После чего происходит распределение задач между разработчиками, которые затем приступают к их выполнению;

– в ходе реализации задач команда разработки проводит периодические короткие встречи (например, ежедневные) с целью оставаться в едином информационном поле выполняемого проекта и обсуждать следующие вопросы:

1) прогресс/статус выполнения задач за время с прошлой встречи;

2) проблемы при реализации и способы их возможного решения;

3) иные организационные и технические вопросы, влияющие на выполнение задач;

– на следующем совещании-планёрке происходит оценка успешности итерации, контроль и анализ её результатов. Задачи, которые были выполнены, удаляются из бэклога, которые не были выполнены, остаются с переоценкой трудоёмкости.

Второй подход – контроль потока задач:

– контролируется объём задач, который команда может вести единовременно;

– задачи берутся из очереди по мере необходимости при наличии ресурса;

– в ходе реализации задач команда разработки проводит периодические короткие встречи (например, ежедневные) с целью оставаться в едином информационном поле выполняемого проекта и обсуждать следующие вопросы:

1) прогресс/статус выполнения задач за время с прошлой встречи;

2) проблемы при реализации и способы их возможного решения;

3) иные организационные и технические вопросы, влияющие на выполнение задач.

6.1.1.3 В зависимости от преобладающих типов работ в различных фазах жизненного цикла разработки программного продукта команда разработки представляет конечные или/и промежуточные результаты следующим образом:

– результаты технического проектирования выносятся на представление и защиту в рамках архитектурного комитета;

– промежуточные результаты реализации требований в ходе разработки представляются членам группы управления проектом и другим заинтересованным лицам в рамках организованного показа (демо);

– результаты выполнения конкретных задач в ходе итераций представляются членам команды на планёрках/статусах и т. п.

6.1.2 Жизненный цикл

6.1.2.1 В процессе разработки программного продукта выделяются этапы, характеризующие глобальное состояние программного продукта и определяющие основные процессы, обеспечивающие успешный переход на следующий этап. С учётом итеративного принципа разработки программного продукта, стоит учитывать, что этап характеризует не столько жёсткий набор работ, характерных только для определённого этапа, сколько приоритеты различных фаз разработки, которые могут присутствовать в каждом этапе жизненного цикла.

Например, на этапе проектирования большую часть времени занимает именно процесс проектирования программного продукта и меньше процесс разработки/реализации, обычно относящийся к прототипированию. В то время как на этапе разработки больше занимает непосредственно написание кода, нежели уточнение или проработка низкоуровневых архитектурных решений.

6.1.2.2 Выделяются следующие этапы жизненного цикла программного продукта:

– формирование облика системы (решаемые задачи, основные характеристики и верхнеуровневая структура);

– анализ требований и проектирование;

– реализация:

1) разработка программного кода;
2) конфигурирование механизмов непрерывной интеграции;
3) разработки инструкций и механизмов для развертывания ПО;
4) тестирование;
5) документирование;

– сертификация:

1) взаимодействие с сертифицирующей лабораторией;
2) устранение выявленных замечаний;

– внедрение (развёртывание):

1) изготовление дистрибутивов и эксплуатационной документации;
2) установка и настройка на оборудовании для эксплуатации;
3) обучение пользователей;

– сопровождение:

1) получение обратной связи от пользователей в ходе эксплуатации;
2) формирование перечня будущих доработок;
3) устранение ошибок и уязвимостей.

6.1.2.3 В ходе развития программного продукта этапы жизненного цикла могут повторяться неоднократно в зависимости от наличия потребности доработок программного продукта.

6.1.3 Роли сотрудников

Разработка ПО – сложный технический и интеллектуальный процесс, требующий задействования специалистов различного профиля и квалификации. В процессе разработки выделяются несколько основных ролей (таблица 1), которые могут выполнять/совмещать сотрудники различных должностей без обязательного взаимооднозначного соответствия.

Таблица 1

Роль	Сфера ответственности	Комментарии
Руководитель проекта	Управление планом расходов проекта. Высокоуровневое планирование и контроль. Обеспечение коммуникаций и решение организационных вопросов с заинтересованными лицами (вне проектной команды)	–

Главный конструктор проекта	Аппаратное исполнение изделия в целом. Разработка конструкторской документации. Изготовление и сборка изделия.	–

Продолжение таблицы 1

Роль	Сфера ответственности	Комментарии
	Организация проведения испытаний. Размещение и подключение аппаратных средств на объекте эксплуатации. Планирование этапов разработки	
Системный аналитик	Разработка модели предметной области. Выявление и формализация требований. Координация работ аналитиков ПО. Формализация автоматизируемого процесса. Планирование аналитической проработки	–
Системный архитектор	Техническое руководство разработкой изделия в части ПО. Разработка архитектуры системного уровня. Разработка и согласование протоколов взаимодействия. Выдача технических требований к оборудованию. Планирование разработки ПО	Обычно является ответственным за репозитории протоколов взаимодействия
Алгоритмист	Разработка алгоритмов обработки информации. Роль эксперта предметной области	–
Архитектор	Разработка архитектуры подсистемы. Координация работ разработчиков, тестировщиков, технических писателей, инженеров конфигураций в части разработки подсистемы ПО. Разработка материалов в ПД	Обычно является ответственным за репозиторий ПО. Координация работ кроме планирования и контроля предполагает: – техническую поддержку перечисленных ролей; – определение требований к квалификации исполнителя и выбор исполнителя из команды; – повышение квалификации исполнителя, при необходимости; – техническую декомпозицию задач; – определение порядка выполнения задач с точки зрения технической зависимости;

Окончание таблицы 1

Роль	Сфера ответственности	Комментарии
		– выпуск версий проектов и оповещение проектной команды о данном событии; – подготовку checklist для тестирования; – и другие
Аналитик	Разработка постановок задач по выявленным требованиям. Представление разработок заказчику. Проверка реализации функций. Обучение работы с ПО	–
Разработчик	Программная реализация задач. Разработка материалов в ПД	–
Тестировщик	Проверка корректности реализации функциональных и нефункциональных требований	Может выделяться роль системного тестировщика в случае необходимости организации интеграционного тестирования
Инженер конфигураций	Сборка программных продуктов. Настройка процесса CI/CD. Развёртывание ПО	–
Технический писатель	Оформление предоставленных материалов. Разработка ПД. Проведение ПД и ПО по подразделениям	–
Инженер эксплуатации	Развёртывание и настройка программного изделия на объекте эксплуатации. Сервисные работы на объекте эксплуатации	В том числе роль могут выполнять: – системный архитектор; – системный аналитик; – инженер конфигураций; – программный архитектор; – тестировщик

6.2 Анализ требований

Анализ требований приведен в документе СМК 7.3.2-02ПР.

6.3 Проектирование архитектуры

Проектирование архитектуры ПО приведено в документе СМК 7.3.2-10ПР.

6.4 Разработка программного обеспечения

6.4.1 Хранение исходных кодов

Хранение исходных кодов ПО приведено в документе СМК 7.3.3-01РИ.

6.4.2 Порядок обзора кода

Порядок обзора кода ПО приведен в документе СМК 7.3.4-06РИ.

6.4.3 Порядок оформления кода

Порядок оформления кода ПО приведен в документе СМК 7.3.2-01РИ.

6.5 Тестирование программного обеспечения

Описание процесса тестирования ПО приведено в СМК 7.3.5-02ПР.

6.6 Документирование программного обеспечения

Описание процесса документирования ПО приведено в СМК 7.3.0-05ПР.

6.7 Конфигурация системы CI/CD

6.7.1 Принципы непрерывной интеграции

6.7.1.1 Современный процесс разработки ПО предполагает наличие в АО «НИЦ СПб ЭТУ» системы непрерывной интеграции (CI – Continious Integration) (далее – CI), берущей на себя ответственность за:

– периодическую сборку ПО (часто по коммиту в репозитории);
– тестирование (модульное, интеграционное);
– анализ исходного кода;
– контроль зависимостей;
– хранение результирующих артефактов;
– сборку пакетов под целевую платформу;
– выпуск новых версий ПО с дополнительными проверками;
– анализ уязвимостей.

С целью обеспечения высокого уровня качества разработки используются следующие принципы и развиваются направления автоматизации:

– принцип «Infrastructure as Code»;
– выбор базовой платформы CI – Jenkins;
– интеграция базовой платформы CI с иными системами, применяемыми в разработке:

1) система контроля версий;
2) система статического анализа;
3) система развёртывания проектов;
4) система управления проектами;
5) и другие системы;

– выбор оптимального для проекта механизма сборки и развёртывания на основе инструментов:

1) контейнеризации;
2) виртуализации;
3) непосредственного развёртывания на ПЭВМ;

– поддержка семантического версионирования;

– поддержка автоматизированного тестирования приложений с интерфейсом пользователя (Sikuli, AssertJ, Selenium);

– организация хранения артефактов сборки в репозиториях различных форматов и стандартов (FTP, Nexus);

– автоматизация подготовки ПО к сертификации.

Обеспечением всего спектра работ в части CI занимается подразделение непрерывной интеграции с привлечением по необходимости сотрудников иных подразделений. Подразделение непрерывной интеграции обеспечивает выполнение как работ в интересах конкретных проектов, так и выполнение ежегодно актуализируемого плана стратегического развития CI.

6.7.1.2 Основные средства, применяемые в системе CI, следующие:

– Jenkins – ядро системы CI:

1) отработка всего автоматизированного процесса сборки (далее – pipeline);
2) взаимодействие со всеми составными частями системы CI;
3) хранение и представление журналов сборки ПО;
4) управление сборкой ПО;

– GitLab – система хранения исходных кодов:

1) хранение исходного кода в отдельных репозиториях и группах проектов;
2) отображение состояния проверки сборки и тестирования каждого коммита;
3) обеспечение возможности обзора кода разработчиками, отличными от автора кода;

– Nexus репозиторий – хранилище версионированных артефактов этапа разработки:

1) хранение Python-артефактов (wheel) в репозитории PyPI-формата;
2) хранение JavaScript-артефактов (npm) в репозитории npm-формата;
3) хранение Java-артефактов (jar/war) в репозитории maven-формата;

– Nexus Docker-registry – хранилище docker-образов;

– FTP-сервер – репозиторий пакетов целевых платформ (deb/rpm):

1) хранение deb-артефактов в репозиториях apt-формата;
2) хранение rpm-артефактов в репозиториях yum-формата;

– SonarQube – средство статического анализа кода:

1) анализ покрытия кода модульными тестами;
2) анализ ошибок в коде;
3) анализ уязвимостей для некоторых языков программирования;
4) анализ дублирования кода между проектами и в рамках одного проекта;
5) анализ статических метрик поддерживаемости кода;

– Doxygen-сервер – хранилище генерированной документации по проектам в виде Web-страниц по каждому программному проекту;

– JIRA – система ведения проектов и задач (каждый коммит привязан к задаче);

– DefectDojo – инструменты DevSecOps, ASPM (управление безопасностью приложений и уязвимостями).

DefectDojo организует:

1) отслеживание уязвимостей;
2) дедупликацию найденных уязвимостей;
3) отчетность.

6.7.2 Покоммитная сборка

6.7.2.1 Ежедневный процесс разработки ПО – команда разработки реализует различную функциональность и выкладывает (коммитит) в систему контроля версий исходного кода, используемую в АО «НИЦ СПб ЭТУ» (SVN). Ежедневный процесс разработки ПО покрывает всё, что происходит до выпуска очередной версии ПО и основной целью автоматизации является поддержание ПО в корректном состоянии с точки зрения тестирования и нахождения проблем разработки на ранних этапах в автоматизированном режиме.

6.7.2.2 Диаграмма потоков данных основного процесса разработки, покрытого средствами CI приведена в приложении А (рисунок А.1). Все основные стадии процесса разработки, покрытого средствами CI, выполняются либо в Docker-контейнерах в случае ОС СН «Astra Linux» и ОС Windows Server 2016, либо на виртуальных машинах, управляемых средством виртуализации Proxmox, в случае ОС Windows 7 и ОС Windows 10 (то есть ОС, контейнеризация для которых крайне затруднена).

6.7.2.3 Все фазы процесса покоммитной сборки выполняются в специально разворачиваемой на время сборки среде без артефактов предыдущих сборок, в том числе в случае виртуализации на ProxMox производится откат заданного snapshot средствами proxmox-jenkins-plugin, использующим ProxMox REST API управления виртуальными машинами.

6.7.2.4 Описание основных средств, применяемых в CI, приведено в 6.7.1.

6.7.3 Выпуск версии программного продукта

6.7.3.1 Диаграмма процесса выпуска новой версии программного продукта в рамках сборочной системы Jenkins приведена в приложении А (рисунок А.2). Сценарий выпуска версии реализован посредством общей программной библиотеки разработчика, выполненной в соответствии с архитектурой Jenkins Shared Library.

6.7.3.2 Процесс выпуска версий состоит из нескольких этапов:

– создание тега в репозитории проекта (для различных стеков технологий процесс может отличаться, описан в базе знаний на страницах соответствующих стеков) – процесс выпуска релиза для Python проектов;

– после этого в GitLab срабатывает webhook для Jenkins, на котором, в свою очередь, и запускается непосредственный процесс сборки новой версии;

– все этапы сборки новой версии запускаются в docker-контейнерах, на основе специально подготовленных docker-образов;

– основными этапами сборки в рамках Jenkins являются:

1) клонирование репозитория проекта;
2) установка сборочных зависимостей и подготовка окружения;
3) сборка;
4) тестирование;
5) генерация и публикация документации;
6) сборка пакета и его выкладывание в репозитории;

– выпуск версии в системе управления проектами (ручная операция).

6.7.3.3 В состав сборочной инфраструктуры выпуска версии входят такие сервисы как:

– GitLab – для хранения исходных кодов проекта;

– Nexus – включает в себя Docker registry для хранения образов, репозитории для хранения зависимостей, а так же артефактов сборок, например: jar, pov, tar, war;

– FTP – в качестве репозитория для систем назначения (astra, mcbc, windows), а так же для хранения результатов сборок в виде пакетов;

– Doxygen – для публикации документации проектов;

– Jenkins – ядро системы CI.

6.7.3.4 Более подробное описание всех средств, применяемых в CI, приведено в 6.7.1.

6.7.4 Версионирование программных продуктов

Порядок версионирования программного продукта приведен в документе СМК 7.5.3-03РИ.

6.7.5 Продуктовая сборка

6.7.5.1 Цель продуктовой сборки – формирование дистрибутива с:

– набором исходных кодов;

– генерированным описанием кода;

– сборочными инструментами для прохождения сборки и сертификации;

– эталоном собранного ПО.

6.7.5.2 Диаграмма потоков данных процесса продуктовой сборки, покрытого средствами CI, представлена в приложении А (рисунок А.3). В основе сборочной системы:

– библиотека для Jenkins, написанная на Groovy, и выполняющая функцию декларативного описания сценария продуктовой сборки, реализованная на основе применения Domain Specific Language (DSL). Позволяет упрощенным способом описать правила подготовки исходных кодов и расположения артефактов в эталоне для поставки заказчику;

– продуктовый сборщик в виде модуля, написанного на Python, который обеспечивает унифицированную сборку как в сборочной среде Jenkins, так и в отрыве от неё.

6.7.5.3 В состав исходных кодов продуктовой сборки входит как минимум Jenkinsfile – сценарий с декларативным описанием процесса выпуска версии, а также любые другие дополнительные файлы, специфичные для ПО. Как и в случае покоммитной сборки, код продуктов хранится в системе контроля версий в отдельном проекте с именем в виде обозначения программного изделия.

6.7.5.4 Запуск процесса продуктовой сборки осуществляется с помощью отправки HTTP-запроса в Jenkins после коммита в ветку или создания тега в репозитории программного продукта. Для обеспечения чистоты сборочной среды и в зависимости от платформы назначения весь процесс сборки программного продукта осуществляется в docker-контейнерах в случае ОС СН «Astra Linux» и ОС Windows Server 2016, либо на виртуальных машинах, управляемых средством виртуализации Proxmox в случае ОС Windows 7, ОС Windows 10 и ОС МСВС.

6.7.5.5 Для хранения образов docker-контейнеров используется Nexus Docker Registry. Перед процессом сборки происходит подготовка сборочного окружения с установкой всех необходимых зависимостей.

В состав сборочной инфраструктуры выпуска версии входят такие сервисы как:

– Nexus – в качестве хранилища пакетов и их исходников, например, для JS, Java или Python;

– FTP – в качестве хранилища результатов продуктовой сборки, а так же в качестве хранилища репозиториев для систем назначения.

Более подробное описание всех средств, применяемых в CI, приведено в 6.7.1.

6.7.5.6 В библиотеке Jenkins предусмотрен механизм проверки повторяемости продуктовой сборки путем сравнения контрольных сумм двух наборов результатов продуктовой сборки. Так же для каждой продуктовой сборки осуществляется сбор дополнительной описательной информации, такой как: количество строк кода и списка используемых языков программирования.

Ниже представлен пример описания сборки, позволяющей получить как готовый набор исходных кодов, так и эталон с проверкой повторяемости сборки:

```
@Library('product@2.18.0') _
…
String productName = '00000-01'
…
createProduct(productName, 'mcbc_5ch01-07') {
   buildTools {
   packagesToDownload = ["build-system-${version}", 'gcc6-c++', …]
  }

  cppSource {
   downloadAside {
    archive {name = 'libuuid'; version = '1.6.2'; revision = '14'}
    …
   }
   download {
    …
    archive {name = 'launch-parameters-control'; version = '2.1.0' }
   }
  }

buildProduct(platform: 'mcbc_5ch01-07', product: productName)
```

6.7.6 Сертификация программного обеспечения

Порядок подготовки ПО к сертификации приведен в документе СМК 7.3.2-04РИ.

6.7.7 Поиск уязвимостей

6.7.7.1 Уязвимости могут быть вызваны ошибками в коде, неправильной конфигурацией системы или некорректным использованием библиотек и компонентов.

Основные причины наличия уязвимостей:

– код может содержать опасные конструкции;

– может использоваться старая версия зависимостей с публично известными уязвимостями (Common Vulnerabilities and Exposures (CVE));

– небезопасная настройка по умолчанию (сервисы могут быть настроены так, что легко поддаются атакам).

Исходя из различных источников наличия уязвимостей, выделяются несколько классов средств, применяемых для обеспечения безопасности разработки ПО.

6.7.7.2 Анализ зависимостей.

Анализ уязвимых deb/rpm-пакетов (OpenSCAP).

Устанавливаемый deb/rpm-пакет ПО часто требует как зависимостей целевой ОС, которые могут содержать уязвимости, так и дополнительные зависимости, поставляемые в составе эталона ПО.

Для анализа наличия уязвимостей в зависимых deb/rpm-пакетах применяется специализированное средство OpenSCAP – открытая платформа для автоматизированного аудита безопасности и оценки соответствия стандартам. В частности, является средством для поиска уязвимостей системы по версиям пакетов. Разработано на основе стандартов проекта SCAP (Security Content Automation Protocol). OpenSCAP помогает пользователям управлять безопасностью IT-инфраструктуры, автоматизируя процессы проверки и выявления уязвимостей.

Основные возможности OpenSCAP:

– проверка соответствия стандартам безопасности;

– оценка уязвимостей;

– анализ конфигураций;

– применимость для всех стеков технологий, производящих при сборке rpm/deb-пакеты.

Инструмент OpenSCAP интегрирован в систему управления CI (Jenkins) посредством общей программной библиотеки разработчика. ФСТЭК содержит актуализируемую БД уязвимостей в формате применяемого средства и система управления CI также отвечает за ежедневное обновление БД уязвимостей с выкладыванием на локальные ресурсы АО «НИЦ СПб ЭТУ». Результаты анализа уязвимостей доступны при сборке проекта в системе управления CI для каждого проекта в виде HTML-отчёта, приведённого на рисунке 1.

Название пакета	Описание проблемы	Связанные ссылки
info-transfer-apkodm=3.1.0	Astra Linux -- уязвимость в apr-util (2023-0303SE17MD)	2023-0303SE17MD VULN-20230217.2 VULN-20230301.2 BDU:2024-02969 CVE-2022-25147
info-transfer-skppad=3.1.0	Astra Linux -- уязвимость в apr-util (2023-0303SE17MD)	2023-0303SE17MD VULN-20230217.2 VULN-20230301.2 BDU:2024-02969 CVE-2022-25147

Рисунок 1

OpenScap-образ, созданный на основе минимального образа для текущей платформы с установленным OpenSCAP, позволяет получить список уязвимостей для текущей платформы.

Ответственность компонентов:

– Jenkins:

1) Pipeline обновления БД уязвимостей – с Gitlab загружается и запускается скрипт выкачивания БД OVAL-описаний. После на openscap-образе запускается openscap-утилита проверки. Результат проверки в виде базовых уязвимостей платформы выкладывается на ftp вместе с БД уязвимостей;

2) шаг проверки уязвимостей – выполняется на openscap-образе после получения deb-пакетов с jenkins-образа. С ftp выкачиваются БД уязвимостей и базовые уязвимости платформы, а с Gitlab выгружается скрипт сравнения уязвимостей. Каждый deb-пакет устанавливается, запускается openscap-утилита. Полученный результат в виде текущих уязвимостей платформы вместе с базовыми уязвимостями платформы передаются в скрипт сравнения уязвимостей, что позволяет получить список уязвимостей для каждого пакета;

– GitLab – хранит скрипт выкачивания БД OVAL-описаний и скрипт сравнения уязвимостей;

– FTP – используется для хранения базовых уязвимостей платформы и БД уязвимостей.

Процесс применения OpenSCAP подробно представлен диаграммой развёртывания на рисунке 2.

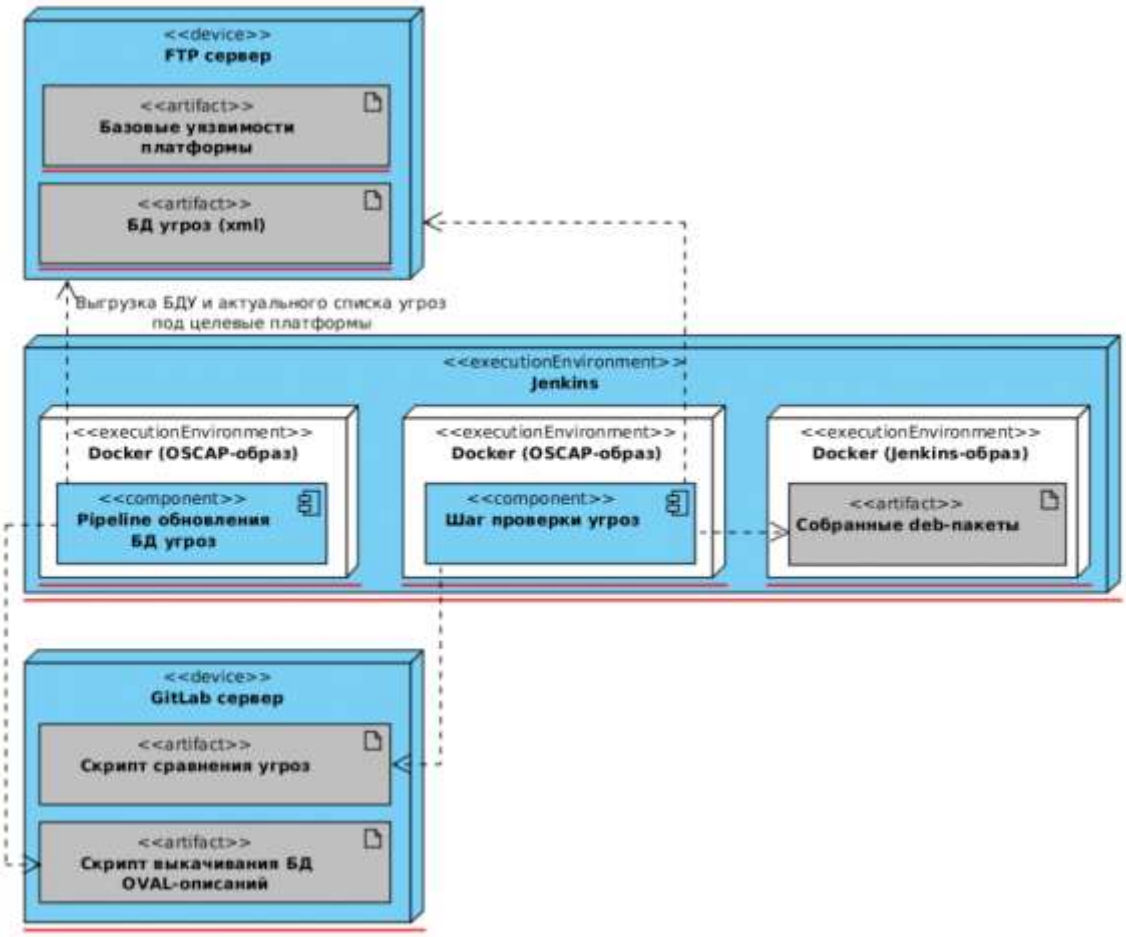

Рисунок 2

Анализ уязвимых библиотек.

Некоторые стеки технологий используют собственную систему разрешения зависимостей при сборке без задействования репозиториев deb/rpm-пакетов. Это требует

наличия дополнительных средств анализа, проверяющих зависимости в файлах специального формата в зависимости от используемой системы сборки:

– Python-проекты – requirements.txt;

– Java-проекты – pom.xml;

– JavaScript-проекты – packages.json.

Применяемые средства анализа, интегрированные в pipeline сборки проектов (через общую программную библиотеку разработчика) соответствующих стеков технологий:

– Trivy – сканер уязвимостей в образах контейнеров и файловых системах. Кроме того, Trivy может находить ошибки в файлах конфигурации, жёстко запрограммированные конфиденциальные данные, использование несовместимых лицензий в проекте;

– Grype – сканер уязвимостей в образах контейнеров и файловых системах. Определяет уязвимости в пакетах ОС, пакетах для конкретных языков и зависимостях приложений.

В общем виде архитектура применения средств анализа представлена на рисунке 3.

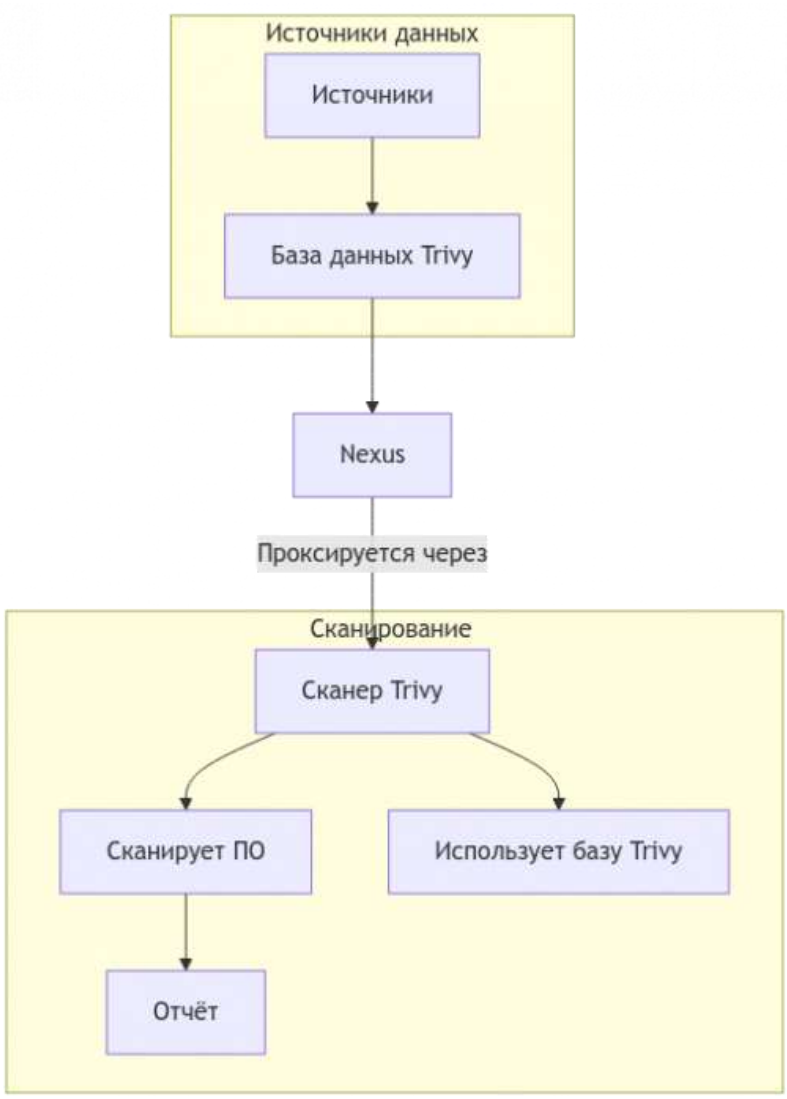

Рисунок 3

Основными особенностями архитектуры применения являются:

– использование нескольких источников при формировании БД уязвимостей указанных инструментов;

– отдельная процедура выкачивания общей БД уязвимостей в локальный репозиторий Nexus.

Результаты поиска уязвимостей по всем стекам технологий/проектам выкладываются посредством системы CI в систему отслеживания уязвимостей DefectDojo.

Система DefectDojo позволяет:

– агрегировать результаты анализа по всем проектам АО «НИЦ СПб ЭТУ»;

– распределять найденные уязвимости по веткам/тегам;

– отслеживать динамику уязвимостей;

– проводить совместную работу над устранением недостатков;

– планировать сканирование приложений;

– помечать уязвимости, присутствующие в банке данных угроз ФСТЭК (при помощи специально разработанного расширения).

DefectDojo – комплексное решение для управления уязвимостями, которое позволяет командам разработчиков ПО своевременно и эффективно выявлять и устранять уязвимости в системе безопасности.

На рисунке 4 приведён пример интерфейса DefectDojo.

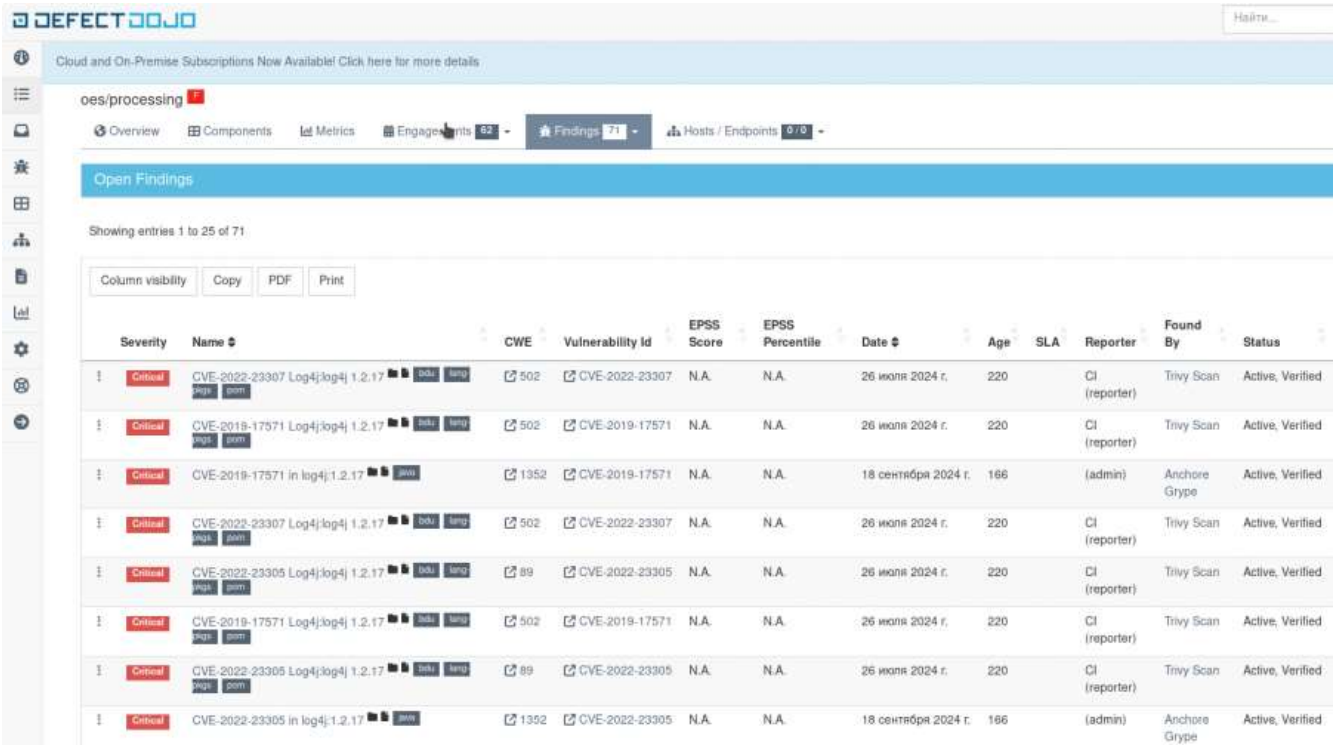

Рисунок 4

6.7.7.3 Статический анализ исходного кода.

Значительное число ошибок может быть найдено при помощи статического анализа с применением различных средств.

Для указанных языков программирования применяются следующие средства статического анализа:

– C++:

1) cpplint;
2) clazy (для Qt);
3) cppcheck;
4) профиль SonarQube (CXX) – cpp comunity plugin;

– Python:

1) набор плагинов flake8;
2) публикация сообщений линтера в комментариях к MR (reviewdog);
3) профиль SonarQube;

– JavaScript:

1) Eslint;
2) публикация сообщений линтера в комментариях к MR (reviewdog);
3) профиль SonarQube;

– Java – профиль SonarQube.

Результаты статического анализа кода по всем стекам технологий/проектам выкладываются посредством системы CI в систему анализа программного кода SonarQube.

Система SonarQube позволяет:

– агрегировать результаты анализа по всем проектам АО «НИЦ СПб ЭТУ»;

– централизованно актуализировать профили качества;

– распределять найденные ошибки по веткам/тегам;

– отслеживать динамику ошибок;

– определять потенциальные небезопасные участки кода, непокрытые достаточным числом тестов.

На рисунке 5 приведён пример интерфейса SonarQube.

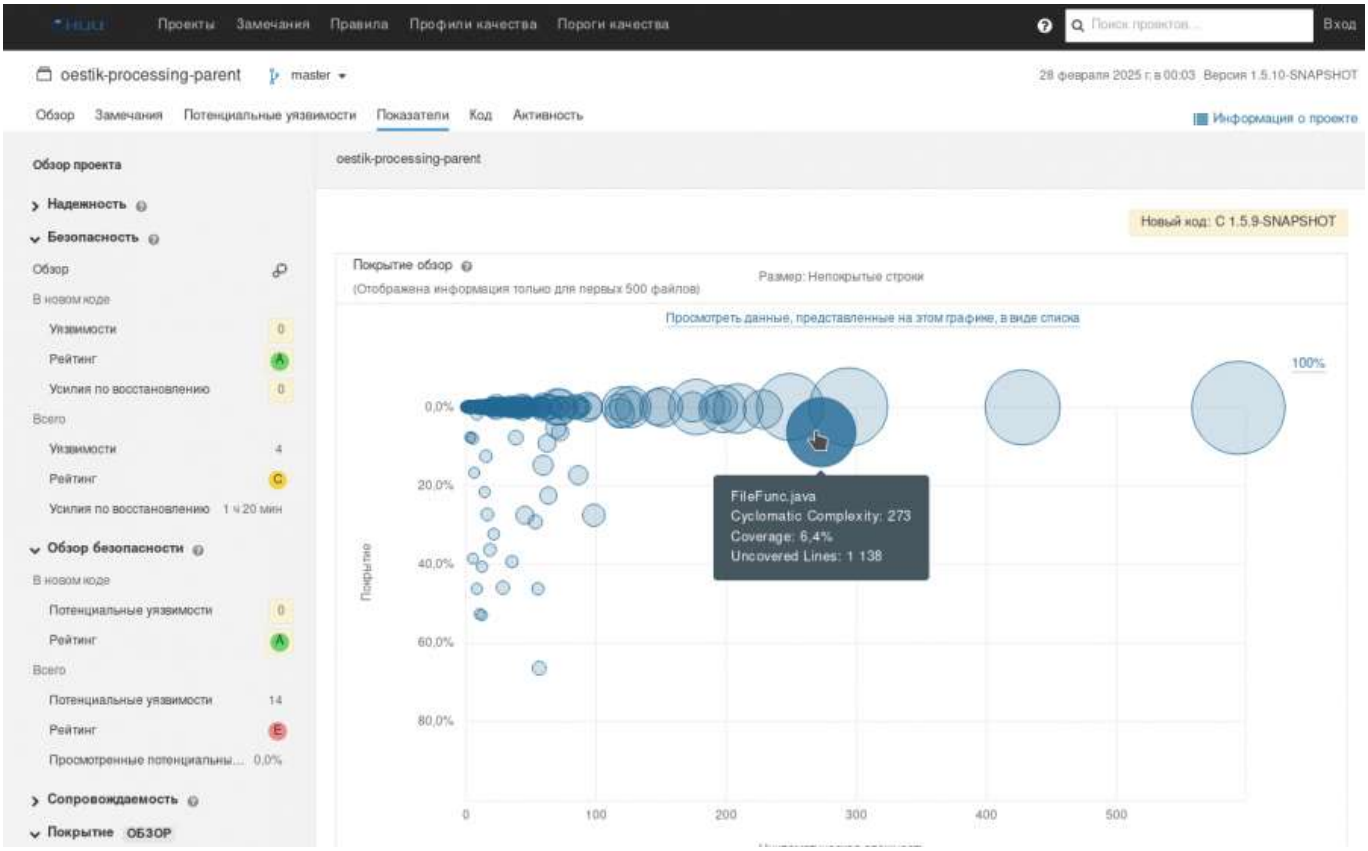

Рисунок 5

6.7.7.4 Динамический анализ исходного кода.

С целью выявления сложных ошибок программирования применяется динамический анализ исходного кода, выполняющийся на этапе автоматизированного модульного

тестирования под платформой разработчика в системе CI и предваряемый сборкой программного продукта с задействованием специализированных средств.

Проверка осуществляется на шаге тестирования и приводит к падению соответствующих тестов.

Например, для анализа C++ проектов задействованы:

– address sanitizer из состава gcc – выявление проблем доступа к памяти;

– undefined behavior sanitizer из состава gcc – выявление неопределённого поведения.

6.8 Инфраструктура для поддержания разработки программного обеспечения

6.8.1 Контроль доступа

6.8.1.1 Система авторизации.

Домен используется как единая система авторизации для всех корпоративных сервисов. Это позволяет централизованно управлять учетными записями сотрудников и доступом к ресурсам. В данный момент используется Active Directory (AD).

Принципы работы:

– централизованная учетная запись:

1) каждому сотруднику создается уникальная учетная запись обычно в формате «[первые буквы имени].[фамилия]» (например, «a.titov» для Анатолия Титова);

2) учетная запись используется для доступа к корпоративным сервисам через настройку самих сервисов;

– методы авторизации:

1) LDAP – прямая интеграция сервисов (например, система управления проектами JIRA) для проверки учетных данных через каталог домена;

2) SSO (Single Sign-On) – единая авторизация для доступа к нескольким сервисам без повторного ввода пароля;

3) синхронизация учетных данных – обновление баз пользователей сервисов (например, система мониторинга Zabbix) домена с использованием автоматизированных скриптов;

– централизованная авторизация при доступе к сервисам:

1) авторизация в сервисах осуществляется через интеграцию с доменом;

2) доступные ресурсы и разрешения определяются на основе членства в группах.

Стандарты:

– все новые учетные записи и группы создаются в домене согласно шаблону именования;

– политики паролей и других методов безопасности регулируются централизованной политикой домена, настроенной в соответствии с рекомендациями руководящих документов.

6.8.1.2 Разграничение прав пользователей.

Доступ к корпоративным ресурсам определяется через группы пользователей, зарегистрированных в домене. Это обеспечивает централизованное управление доступом и позволяет гибко адаптировать права под текущие задачи.

Принципы работы:

– централизованное создание/удаление групп: пользователи объединяются в группы на основе их роли и/или принадлежности к проекту. Примеры групп:

1) NIC\secret-project-developers – разработчики, работающие над проектом secret-project;

2) NIC\dpo – сотрудники департамента программного обеспечения;

3) NIC\grafana-admins – сотрудники с правами администратора для сервиса Graphana;

– связи между системами: настраиваются через доменные группы. Например, для каждого репозитория исходного кода может быть задана специальная переменная «mailgroup», которая указывает на доменную группу. Пользователи автоматически добавляются в эту группу при настройке доступа и актуализируются при изменении состава разработчиков в автоматизированном режиме. К группе репозиториев привязывается доменная группа, агрегирующая все вложенные доменные группы репозиториев. Пример:

1) группа репозиториев java-libs (группа java-libs-developers);

2) репозиторий amqp-client (группа java-libs-amqp-client-developers);

– назначение прав:

1) каждому ресурсу назначаются права доступа на уровне группы;

2) доступ предоставляется согласно принципам минимально необходимого доступа (Least Privilege);

– поддержка и актуализация:

1) состав групп регулярно пересматривается и актуализируется;

2) удаление прав производится после увольнения сотрудника или по заявке руководителя;

– разграничение доступа:

1) пользователи должны иметь доступ только к тем ресурсам, которые необходимы для выполнения их задач;

2) настройки доступа производятся через доменные группы. Для каждого сервиса предусмотрены привилегии, привязанные к группе. Пример: в системе управления CI производится гибкая настройка прав доступа на основе доменных групп, используя Project-Based Security. Это позволяет разграничивать доступ, основываясь на принадлежности пользователя к определенной группе. На рисунке 6 приведён пример организации матричного доступа.

Стандарты:

– группы создаются с учетом функциональных и проектных требований посредством оформления заявок через систему управления проектами;

– доступ пользователя организуется посредством оформления заявок через систему управления проектами. Право доступа к документации проекта и\или репозиторию выдается в соответствии с заявкой, которую должен утвердить ответственный. Пример групп ресурсов с выделенными ответственными:

1) repo:doc&common – заявки на репозитории doc или common (svn);

2) repo:source – заявки на репозитории исходного кода (git);

3) repo:ftp – заявки по доступу к разработанным программным продуктам (ftp);

– аудит прав осуществляется отделом вычислительных комплексов и систем. Несанкционированные изменения или нарушения политики доступа фиксируются и анализируются.

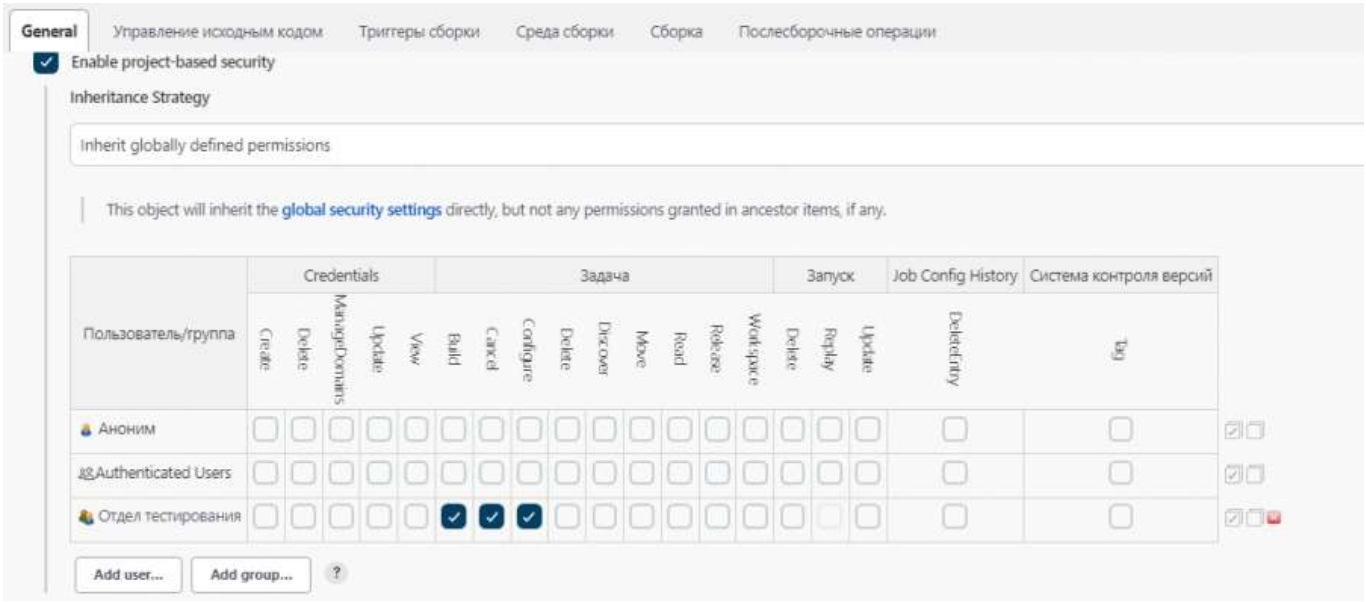

Рисунок 6

6.8.1.3 Централизованное хранилище секретов.

Используется для безопасного хранения и управления секретами, такими как: пароли, ключи API, токены доступа и конфиденциальные данные. Хранилище секретов обеспечивает централизованное управление доступом и минимизацию рисков утечки. Доступ осуществляется как пользователями, так и сервисами АО «НИЦ СПб ЭТУ» в автоматизированном режиме. В данный момент используется HashiCorp Vault.

Принципы работы:

– секреты и их динамика:

1) секреты хранятся в зашифрованном виде и предоставляются только авторизованным пользователям и сервисам;

2) хранилище секретов поддерживает как статические секреты (например, пароли), так и динамически создаваемые (например, временные учетные данные для доступа к базе данных);

3) при изменении секретов поддерживается хранение истории их изменения;

– доступ к секретам:

1) доступ предоставляется на основе политики, которая определяется ролями (Role-Based Access Control, RBAC);

2) каждая роль определяет, какие секреты доступны и какие действия можно выполнять (чтение, создание, удаление и т. д.);

– интеграция с сервисами:

1) секреты из хранилища могут использоваться для автоматизации процессов в систему управления CI, развертывания приложений, подключения к базам данных и т. д.;

2) во время работы сервисов с хранилищем в самом сервисе хранится только ID секрета, а не сам секрет;

3) пример интеграции с системой управления CI через Jenkinsfile:

```
def secrets = [
    [
      path: 'jenkins_kv/gitlab',
       secretValues: [
```

```
            [envVar: 'GIT_TOKEN', vaultKey: 'download-product-gitfiles'],
            ]
          ]
        ]
...
Utils.pipeline.withVault(secrets) {
 Utils.pipeline.sh("git clone -b master http://oauth2:$GIT_TOKEN@ git.nic.etu/repository-name.git ../target_dir")
}
```

Стандарты:

– хранение секретов:

1) секреты группируются в пространстве имен (namespaces) в соответствии с проектами, отделами или сервисами;

2) секреты находятся в зашифрованном виде как в состоянии покоя, так и во время транзита;

– политики доступа:

1) политики разрабатываются индивидуально для каждого пространства имён, разграничением прав занимается администратор хранилища;

2) ведётся аудит доступа к секретам;

– шифрование данных:

1) все данные, хранящиеся в хранилище, зашифрованы с использованием современных криптографических алгоритмов;

2) даже в случае физической или логической компрометации хранилища (например, его кражи), расшифровать данные невозможно без доступа к мастер-ключу.

6.8.1.4 Управление сертификатами.

В АО «НИЦ СПб ЭТУ» используется центр сертификации для выпуска самоподписанных сертификатов. Сертификаты обеспечивают безопасное соединение для внутренних сервисов.

Принципы работы:

– центр сертификации:

1) является доверенным источником для создания, управления и обновления сертификатов;

2) выпущенные сертификаты используются исключительно внутри АО «НИЦ СПб ЭТУ» и не доверяются публичными браузерами или внешними сервисами;

3) сертификаты выдаются для сервисов и их доменов (например, proxmox-jenkins.nic.etu);

– интеграция и использование:

1) внутренние сервисы могут использовать сертификаты, выданные внутренним центром сертификации, если требуется;

2) внешние сервисы используют сертификаты, выпущенные публичными центрами сертификации, если требуется;

3) для использования сертификата пользователь должен добавить открытый ключ на ПЭВМ.

Пример сертификата приведен на рисунке 7.

Поле	Значение
Country Name (2 letter code) [AU]	RU
State or Province Name (full name) [Some-State]:	Saint Petersburg
Locality Name (eg, city) []:	Saint Petersburg
Organization Name (eg, company) [Internet Widgits Pty Ltd]:	NIC SPb ETU
Organizational Unit Name (eg, section) []:	Continuous Integration Group
Common Name (e.g. server FQDN or YOUR name) []:	proxmox-jenkins.nic.etu
Email Address []:	d.stepanishev@nicetu.spb.ru

Рисунок 7

Стандарты:

– процесс выпуска сертификатов – выпуск сертификатов осуществляется на основе внутренних требований АО «НИЦ СПб ЭТУ» по заявкам в системе управления проектами;

– хранение и ротация:

1) выпущенные сертификаты и ключи хранятся в защищенном хранилище;

2) срок действия сертификатов ограничен;

– безопасность:

1) закрытые ключи никогда не передаются вне защищенных хранилищ;

2) все сертификаты и ключи имеют строгие политики доступа, основанные на группах пользователей и ролях.

6.8.2 Резервное копирование

6.8.2.1 Резервное копирование является критически важным процессом для обеспечения сохранности данных и минимизации простоев в случае сбоев. В целях обеспечения резервирования проводится классификация сервиса и выбор для него оптимально подходящего вида и средства резервного копирования.

6.8.2.2 Файловое резервирование.

Инструменты – Borg-backup, UrBackup.

Тип данных – отдельные файлы и каталоги.

Инструменты предназначены для работы с изменяемыми наборами файлов.

Borg-backup – использует дедупликацию и сжатие, что делает его эффективным для хранения данных, где обновляются только части файлов.

Резервируемые сервисы:

– сервисы мониторинга (Grafana.nic.etu/Prometheus/Loki);

– сервис автоматизированного развёртывания (Portainer);

– сервис централизованного поиска текстовой информации (search.nic.etu/elasticsearch).

Основные этапы создания резервной копии:

– инициализация репозитория для хранения резервных копий;

– определение данных и конфигураций для резервирования;

– запуск инкрементного копирования со сжатием и опциональным шифрованием.

UrBackup – обеспечивает прозрачное создание резервных копий пользовательских данных на сервере с возможностью гибкой настройки частоты и глубины копирования. Имеет графический интерфейс.

Резервируемые сервисы:

– сервер хранения собранных программных продуктов и документов (FTP+ SVN);

– сервер хранения исходных кодов (GitLab);

– сервер управления репозиториями (Sonatype Nexus).

Основные этапы создания резервной копии:

– установка клиента на устройстве и настройка серверного подключения;

– определение каталогов для резервирования;

– автоматическое создание резервных копий по расписанию или вручную.

6.8.2.3 Резервирование образов дисков.

Инструмент – Clonezilla.

Тип данных – полные образы дисков или разделов.

Clonezilla – копирует весь диск или раздел, включая загрузочные области, таблицы разделов и все данные.

Полезно для создания резервной копии ОС, развертывания стандартных настроек на несколько устройств или восстановления после катастрофических сбоев.

Резервируемый сервис – образы ОС для разных сервисов CI инфраструктуры (например, docker серверы).

Основные этапы создания резервной копии:

– запуск процесса создания образа диска или раздела;

– сохранение образа в сетевое хранилище;

– проверка целостности созданного образа.

6.8.2.4 Резервирование виртуальных машин.

Инструмент – система снимков виртуальных машин в Proxmox VE.

Тип данных – снимки виртуальных машин.

Система снимков виртуальных машин в Proxmox VE подходит для виртуализированных сред, где важно сохранять текущее состояние всей виртуальной машины (операционная система, приложения, данные).

Снимки виртуальной машины позволяют быстро откатиться к предыдущему состоянию без длительных операций восстановления. В Proxmox VE система снимков (snapshots) предоставляет возможность фиксировать текущее состояние виртуальной машины или контейнера в определенный момент времени. Это особенно полезно для таких систем, как Jenkins, которые требуют высокой стабильности и могут подвергаться изменениям, например, при обновлении, внесении конфигурационных изменений или добавлении плагинов.

Резервируемые сервисы:

– база знаний (Wiki);

– хранилище генерированной документации (Doxygen);

– система управления CI (Jenkins).

Основные этапы создания резервной копии:

– выбор виртуальной машины или контейнера для резервирования;

– настройка расписания резервных копий (снимков);

– сохранение данных в локальном хранилище.

6.8.3 Система мониторинга

6.8.3.1 Мониторинг инфраструктуры АО «НИЦ СПб ЭТУ» обеспечивает:

– контроль за состоянием оборудования и сервисов;

– оперативное реагирование на нештатные ситуации;

– обоснованное проведение модернизации оборудования с возможным перераспределением производительных мощностей;

– исследование сложных корреляций событий в системе с целью установления первопричины сбоя;

– контроль заданных требований к уровню надёжности сервисов АО «НИЦ СПб ЭТУ».

Мониторинг инфраструктуры подразделяется на два класса с, возможно, различными применяемыми инструментами:

– мониторинг оборудования;

– мониторинг сервисов.

6.8.3.2 Мониторинг оборудования.

Инструмент – Zabbix.

Функции мониторинга:

– мониторинг доступности узлов сети по протоколу ICMP;

– мониторинг оборудования по протоколу SNMP;

– мониторинг ПЭВМ посредством установленного агента;

– представление результатов мониторинга на карте сети с физическим расположением узлов и в виде таблиц/графиков;

– хранение истории значений параметров мониторинга;

– уведомление о нештатных событиях посредством принятого средства обмена мгновенными сообщениями (Rocket.Chat-канал zabbix-alert-ci, опционально Telegram).

Классы ресурсов.

Оборудование:

– сервера – Proxmox, SonarQube, Nexus, Docker и др;

– сетевое оборудование – коммутаторы, маршрутизаторы, межсетевые экраны;

– источники бесперебойного питания.

Сервисы:

– отслеживание доступности API и состояния Web-сервисов;

– состояние и размер системы управления базами данных (Postgres);

– система управления CI (Jenkins) в части достаточности ресурсов узлов сборки.

На рисунке 8 представлен мониторинг числа контейнеров, необходимых для сборки проектов с течением времени в рамках суток.

Рисунок 8

Архитектура.

Система мониторинга охватывает серверы, сети, приложения и прочие ресурсы и предоставляет средства для сбора, анализа, визуализации и уведомления о метриках и событиях.

Особенности архитектуры:

– единая платформа – всё интегрировано в одном решении, включая сбор данных, анализ и уведомления;

– шаблоны оборудования – множество готовых шаблонов мониторинга различного оборудования в открытом доступе с возможностью разработки собственных;

– инструменты автоматического обнаружения устройств;

– управление событиями – встроенная система корреляции событий.

На рисунке 9 представлена диаграмма потоков данных мониторинга.

Представление результатов.

Результаты представлены следующим образом:

– таблицы;

– графики;

– панели – показывают текущие проблемы, отсортированные по приоритетам;

– карты сетей – отображают текущее состояние оборудования и последние предупреждения.

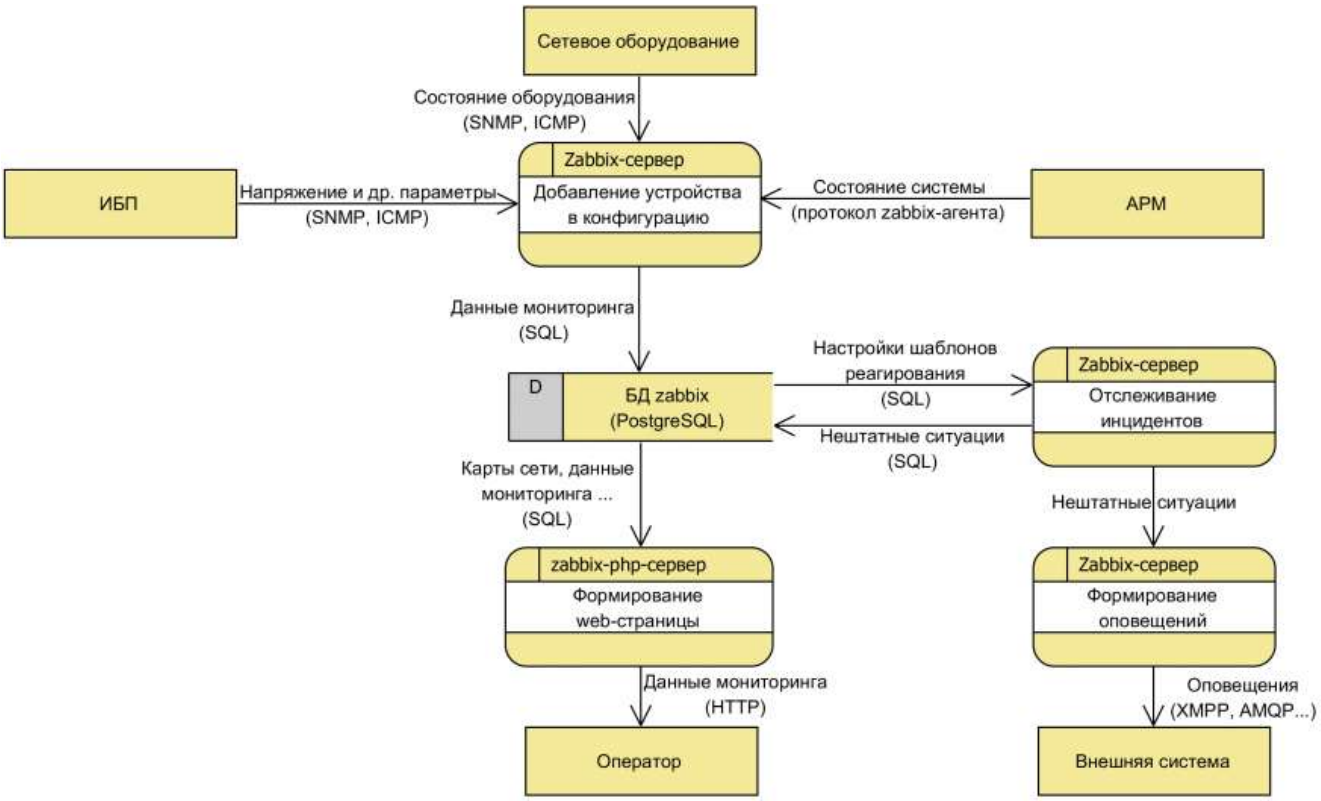

Рисунок 9

На рисунках 11, 10 представлены примеры карты и панели мониторинга соответственно.

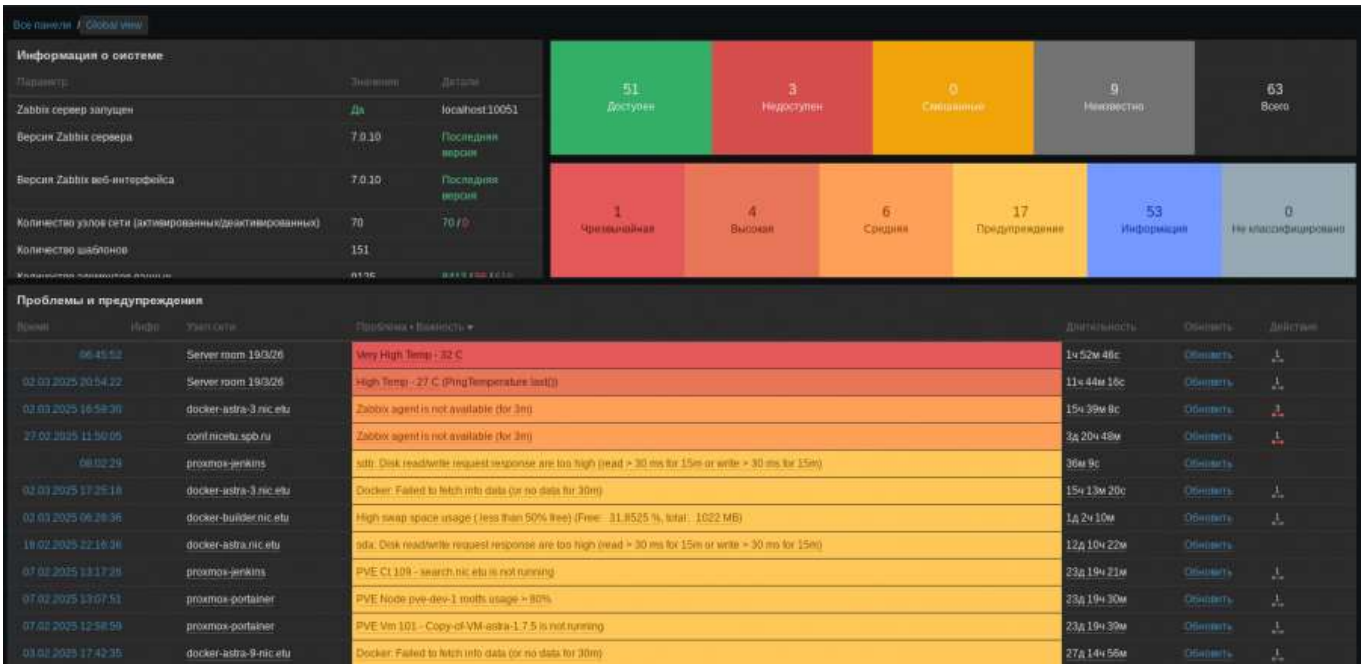

Рисунок 10

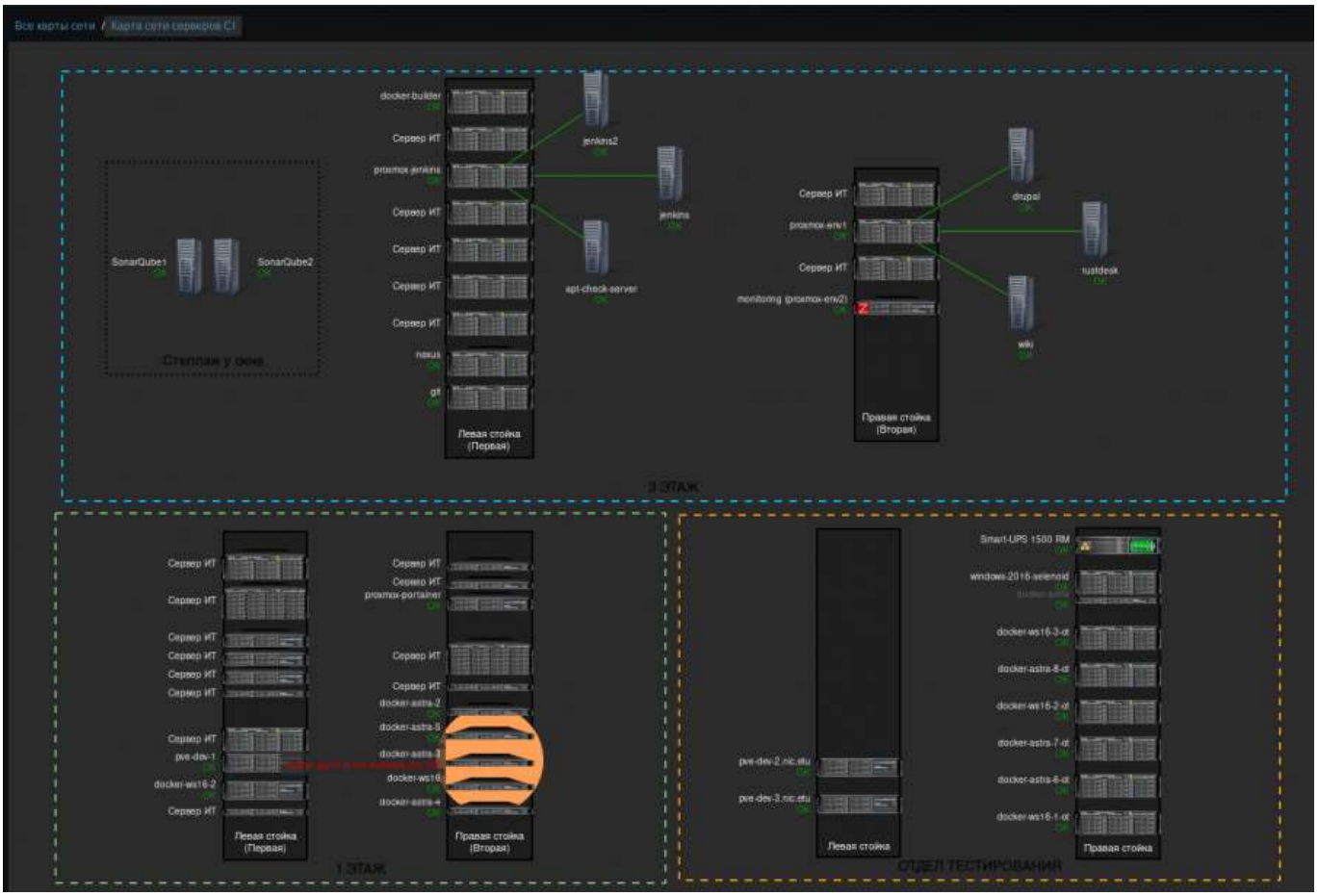

Рисунок 11

Данные с течением времени прореживаются в соответствии с настройками хранения с целью с одной стороны уменьшить объём хранимой информации на длительном промежутке времени, с другой – обеспечить возможность анализа исторических данных за достаточно большой промежуток времени.

6.8.3.3 Мониторинг сервисов.

Инструмент – Grafana + Loki + Promethus.

Функции мониторинга:

– мониторинг состояния сервисов/служб;

– сбор и организация поиска по журналам работы сервисов;

– хранение истории значений параметров мониторинга.

Классы ресурсов.

Контейнеры:

– мониторинг состояния и метрик контейнеров на нодах Docker;

– мониторинг контейнеров, управляемых Portainer.

Сервисы:

– Jenkins – статусы по задачам (Jobs), активные контейнеры;

– Nexus – доступность и производительность;

– SonarQube – метрики состояния.

Сторонние данные:

– скрипты CI инфраструктуры, например, для проверки актуальности версий приложений в CI;

– импортированные данные из системы мониторинга оборудования (Zabbix) (при необходимости).

Архитектура.

Система состоит из трех основных компонентов:

– Prometheus – система мониторинга и оповещений, оптимизированная для сбора метрик временных рядов;

– Grafana – платформа визуализации данных, которая поддерживает подключение множества источников данных;

– Loki – агрегатор журналов, интегрированный с Grafana, оптимизированный для хранения метрик и логов в едином формате.

Особенности архитектуры:

– большая часть современных сервисов умеет поддерживать работу с prometheus и предоставляет endpoint для сбора метрик (нет необходимости дополнительных агентов);

– существует большое количество экспортеров для приложений, которые не поддерживают выдачу метрик;

– динамические панели с настраиваемыми параметрами;

– журналы и метрики вместе – Loki и Grafana упрощают анализ метрик и логов в одном интерфейсе;

– поддержка языков запросов – PromQL и LogQL дают гибкость в запросах данных.

Представление результатов.

Результаты представлены следующим образом:

– таблицы;

– графики – временные ряды, гистограммы, тепловые карты;

– панели;

– журналы.

Сильной стороной средства является качественное интерактивное представление графиков/диаграмм. На рисунке 12 представлен пример графического представления.

Рисунок 12

7 Управление документом

Управление настоящей документированной процедурой осуществляется в соответствии с документом СМК 4.2.3-01ПР.

Ответственным за актуализацию настоящей документированной процедуры является начальник департамента программного обеспечения.

Приложение А
(обязательное)
Диаграммы потоков данных

А.1 Диаграмма потоков данных покоммитной сборки приведена на рисунке А.1.

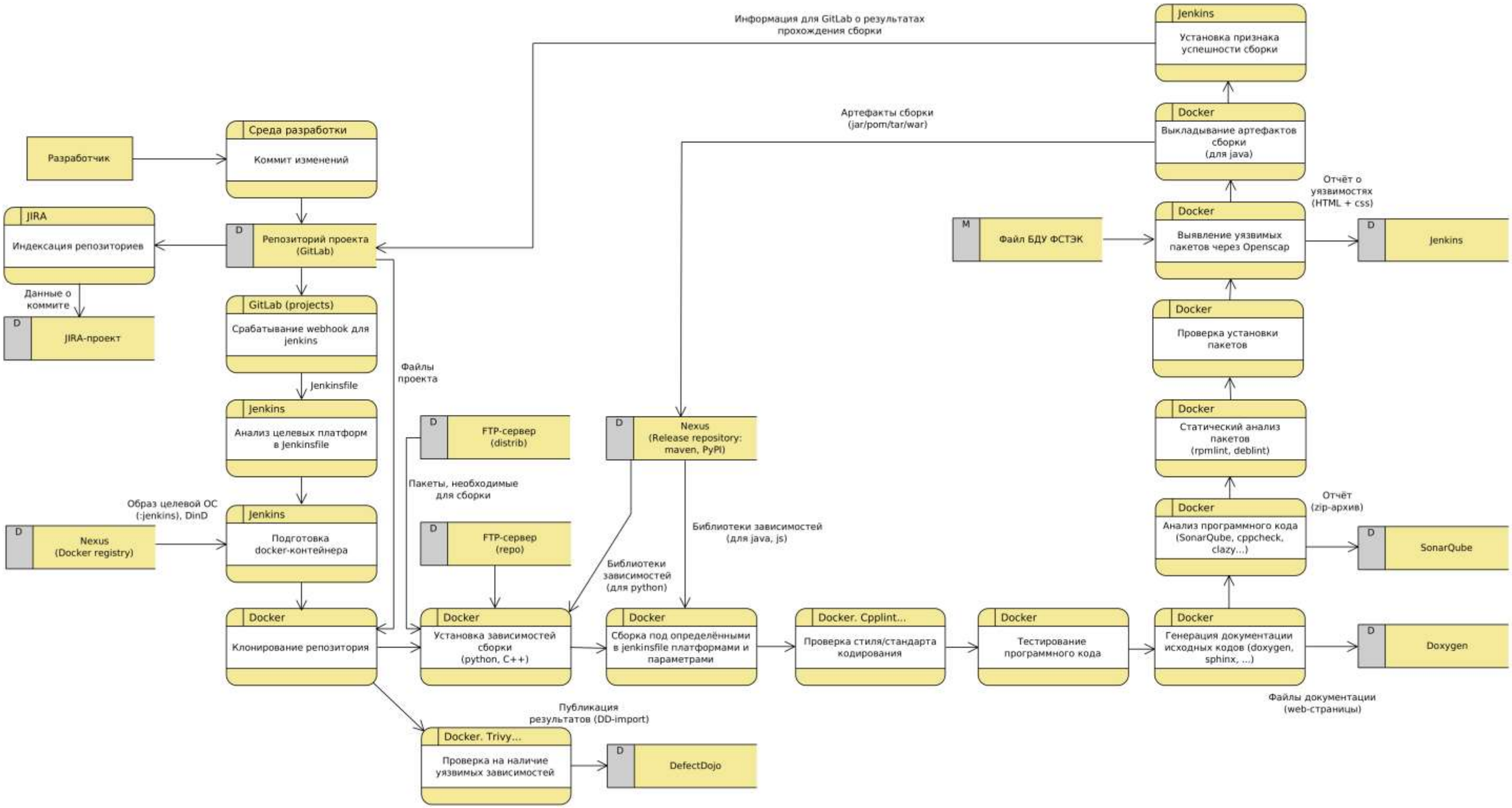

Рисунок А.1

А.2 Диаграмма потоков данных выпуска версии приведена на рисунке А.2.

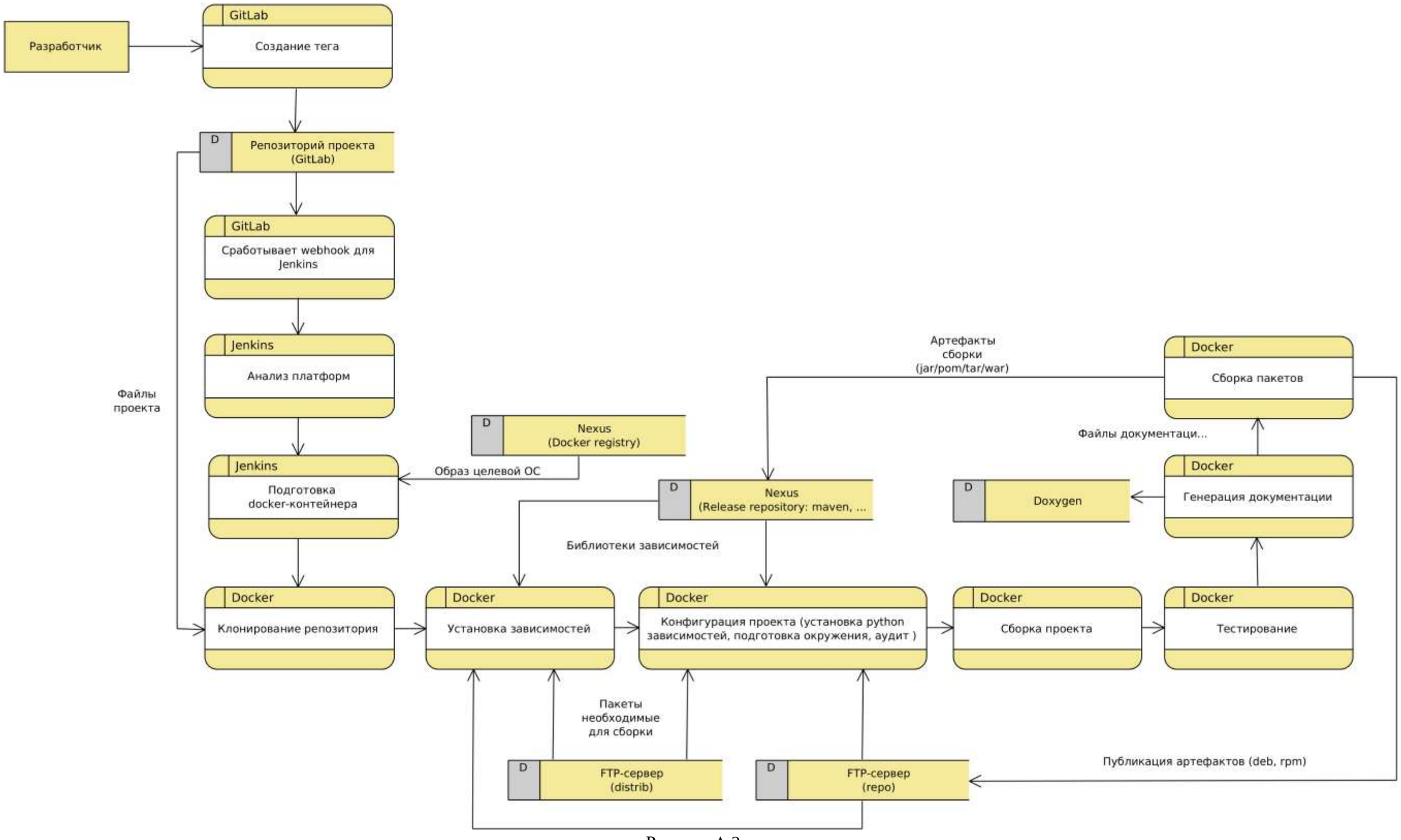

Рисунок А.2

А.3 Диаграмма потоков данных продуктовой сборки приведена на рисунке А.3.

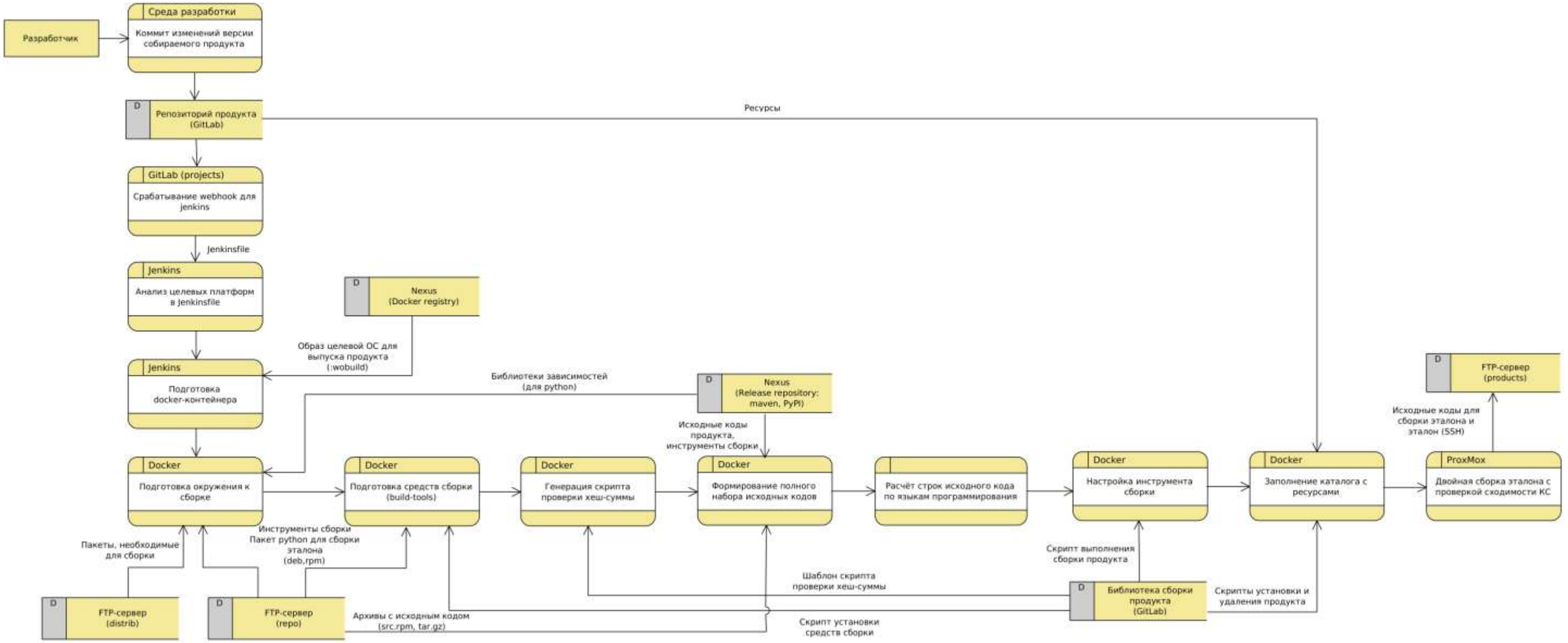

Рисунок А.3

Лист регистрации изменений									
Изм.	Номера листов (страниц)				Всего листов (страниц) в документе	Номер документа	Входящий номер сопроводительного документа и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					