РЕФЕРАТ

Выпускная квалификационная работа: 61 страница, 11 рисунков, 5 таблиц, 22 источника.

Ключевые слова: ВКР, сервис, deploy-бот, мессенджер, TeamCity, CI/CD, развертывание(деплой), DevOps.

Объектом исследования является ООО «Свик».

Предметом исследования является процесс автоматизированного развёртывания программных решений в рамках CI/CD.

Целью выпускной квалификационной работы является разработка сервиса для управления сборкой и развертыванием программного обеспечения.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Проанализировать текущее состояние процессов развертывания и выявить существующие недостатки.
2. Выполнить обзор существующих решений в области автоматизации CI/CD.
3. Определить функциональные и нефункциональные требования к разрабатываемому программному средству.
4. Разработать архитектуру и проектные решения программного продукта.
5. Реализовать программное средство deploy-бота, обеспечивающего интеграцию TeamCity и мессенджера Slack.
6. Провести оценку экономической эффективности внедрения разработанного решения.

Исходные данные, средства и методы, используемые при выполнении проекта:

– документация API TeamCity и Slack API;

– методы системного анализа бизнес-процессов и функционального моделирования (BPMN, UML);

– средства разработки программного обеспечения: Python, Bash, библиотеки работы с API, системы контейнеризации Docker; система контроля версий Git;

– инструменты анализа трудоёмкости и расчёта экономической эффективности проектов.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Разработка программного обеспечения в современных условиях невозможна без применения подходов, направленных на ускорение и повышение надёжности процессов поставки кода. Одним из таких подходов является DevOps-практика, в рамках которой особую роль играет внедрение CI/CD (Continuous Integration / Continuous Delivery) – непрерывной интеграции и доставки программного продукта.

«CI/CD стал неотъемлемой частью современной разработки ПО и позволяет организациям быстро и надежно поставлять новое программное обеспечение на рынок» [11].

Однако несмотря на широкое распространение CI/CD-инструментов, таких как TeamCity, Jenkins, GitLab CI и другие, многие команды всё ещё сталкиваются с рядом ограничений: неинтуитивные интерфейсы, высокая техническая сложность запуска сборок, недостаточная прозрачность и отсутствие гибкого управления развёртыванием на различных окружениях (dev, staging, production и др.). Это порождает ошибки, задержки и усложняет командную работу.

Актуальность темы обусловлена необходимостью повышения эффективности процессов доставки программного обеспечения путём автоматизации взаимодействия с CI/CD-системами через привычные интерфейсы – мессенджер Slack. Такое решение позволяет оперативно управлять процессами развертывания, получать нотификации о ходе сборок, инициировать откаты и прочие действия без необходимости доступа к интерфейсу TeamCity.

Объектом исследования является ООО «Свик».

Предметом исследования является процесс автоматизированного развёртывания программных решений в рамках CI/CD.

Целью выпускной квалификационной работы является разработка сервиса для управления сборкой и развертыванием программного обеспечения.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Проанализировать текущее состояние процессов развертывания и выявить существующие недостатки.
2. Выполнить обзор существующих решений в области автоматизации CI/CD.
3. Определить функциональные и нефункциональные требования к разрабатываемому программному средству.
4. Разработать архитектуру и проектные решения программного продукта.
5. Реализовать программное средство сервис, обеспечивающего интеграцию TeamCity и мессенджера Slack.
6. Провести оценку экономической эффективности внедрения разработанного решения.

Исходные данные, средства и методы, используемые при выполнении проекта:

– документация API TeamCity и Slack API;
– методы системного анализа бизнес-процессов и функционального моделирования (BPMN, UML);
– средства разработки программного обеспечения: Python, Bash, библиотеки работы с API, системы контейнеризации Docker; система контроля версий Git;
– инструменты анализа трудоёмкости и расчёта экономической эффективности проектов.

Практическая значимость разработанного решения заключается в повышении скорости и надёжности процессов развертывания программных

решений, снижении трудозатрат сотрудников, минимизации ошибок при деплое и обеспечении высокой мобильности управления процессами CI/CD.

1 Аналитическая часть

1.1 Технико-экономическая характеристика предметной области

Компания ООО «СВИК», адрес: Алтайский край., г. Барнаул, ул. Шумакова, д. 23А офис 408 зарегистрирована 15.09.2005. Организации присвоены ИНН 2225071847.

Это современная компания, деятельность которой сосредоточена в области разработки, сопровождения и внедрения программных продуктов, а также управления цифровыми данными и ИТ-инфраструктурой. Предприятие активно использует современные технологии DevOps, CI/CD, микросервисную архитектуру, применяет контейнеризацию, облачные решения и собственные системы аналитики.

На рисунке 1 представлена организационно-штатная структура ООО «Свик».

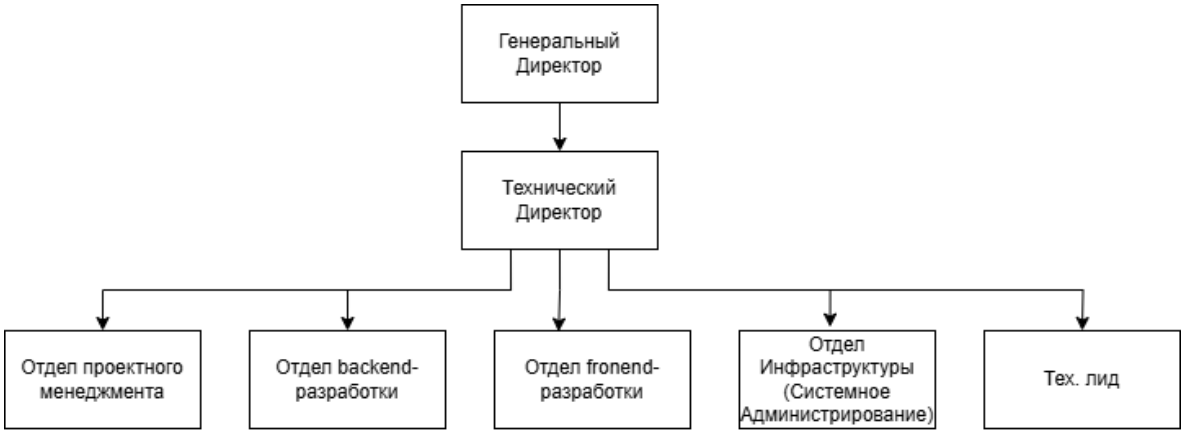

Рисунок 1 – Организационно-штатная структура ООО «Свик»

Организационно компания имеет следующую иерархию:

Генеральный директор – высшее руководство организации. Отвечает за общее стратегическое управление, принятие ключевых решений, взаимодействие с заказчиками и инвесторами.

Технический директор (CTO) – осуществляет контроль над техническим развитием компании, курирует разработку, инфраструктуру и проектную деятельность.

В подчинении у технического директора находятся четыре ключевых отдела:

1. Отдел Backend-разработки. «Сотрудники которого занимаются созданием серверной логики, проектированием архитектуры программных продуктов, реализацией API и интеграцией с внешними системами. Backend-разработчики взаимодействуют с базами данных, реализуют бизнес-логику приложений и обеспечивают надёжность и масштабируемость сервисов» [17].

Руководитель backend-разработки – принимает архитектурные решения, осуществляет code review, руководит процессами сотрудников в отделе.

Backend-разработчики (middle, junior) – выполняют текущие задачи по реализации и поддержке функционала.

2. Отдел Frontend-разработки. «Отвечает за клиентскую часть приложений и пользовательский интерфейс. Разработчики этого направления обеспечивают удобную и быструю работу интерфейса, адаптивную верстку и взаимодействие с серверной частью через API. Они активно работают с современными JavaScript-фреймворками и следят за соблюдением принципов UX/UI» [17].

3. Отдел проектного менеджмента. «Роль аналитиков – сбор требований, проектирование решений и сопровождение проекта на всех этапах. Основная задача сотрудников отдела – выявление и формализация требований заказчика, моделирование бизнес-процессов, составление технических заданий и контроль сроков реализации проектов. Они обеспечивают коммуникацию между клиентом и техническими специалистами, а также сопровождают проекты на всех этапах их жизненного цикла» [17].

4. Отдел инфраструктуры (Системное администрирование) «Ключевой отдел, обеспечивающий работу серверной инфраструктуры, CI/CD и автоматизации» [11]. Ключевую роль в обеспечении бесперебойной работы всей цифровой инфраструктуры играет отдел инфраструктуры, в состав которого входят DevOps-инженеры и системные администраторы. «DevOps-инженеры занимаются автоматизацией процессов разработки, сборки и доставки программного обеспечения, сопровождают CI/CD-системы, развертывают и конфигурируют виртуальные среды» [13].

Системные администраторы отвечают за настройку серверов, сетей, безопасность, резервное копирование и общее администрирование ИТ-среды компании. В рамках практики особое внимание уделялось взаимодействию с этим подразделением, поскольку именно здесь реализуются задачи, связанные с установкой и сопровождением TeamCity.

5. Технический лидер (Tech Lead). Не входит в отдельный отдел, но играет критически важную роль. Он выступает как связующее звено между командами разработки, аналитиками и DevOps-инженерами. «Координирует взаимодействие между командами, принимает архитектурные решения, проводит ревью кода и оказывает менторскую поддержку разработчикам. Техлид обеспечивает соблюдение единых технических стандартов и способствует повышению эффективности командной работы» [11].

Категории и численность работающих. В компании работают специалисты различного профиля: разработчики программного обеспечения (frontend и backend), системные администраторы, DevOps-инженеры, аналитики и проектные менеджеры. Средняя численность сотрудников составляет около 100 человек.

Виды, номенклатура и объемы продукции или услуг. Компания предоставляет услуги по разработке корпоративных и клиентских программных решений, автоматизации бизнес-процессов, настройке и сопровождению CI/CD-систем, консалтингу в области цифровой

трансформации. Объём работ определяется контрактами с заказчиками и включает как разработку новых систем, так и поддержку существующих.

«Этапы подготовки услуг включают сбор и анализ требований заказчика, проектирование архитектуры решений, разработку и тестирование программного обеспечения, автоматизацию процессов развертывания, развёртывание решений в продуктивных средах и последующее техническое сопровождение. Методология проектирования ИС соответствует рекомендациям по структурному проектированию» [17].

Виды и количество материальных ресурсов и оборудования. Для обеспечения деятельности компании используются серверные мощности для размещения систем CI/CD и программных продуктов, сетевое оборудование, лицензированное программное обеспечение, рабочие станции сотрудников, а также системы резервного копирования и мониторинга.

Материальные потоки включают закупку серверного и сетевого оборудования, лицензий на программные продукты. Финансовые потоки состоят из поступлений за выполненные проекты и расходов на поддержание инфраструктуры. Информационные потоки включают в себя исходные коды программ, документацию, скрипты автоматизации и отчёты систем мониторинга. Описание информационных потоков и их обработки строится на теоретических основах, изложенных в [15].

Положение на рынке ООО «СВИК» занимает стабильные позиции на региональном рынке информационных технологий. Компания успешно конкурирует с местными ИТ-компаниями и федеральными аутсорсинговыми организациями благодаря качеству предоставляемых услуг, применению современных технологий и индивидуальному подходу к клиентам. Сфера влияния компании охватывает международные проекты, активно развиваются направления в области DevOps и CI/CD.

Сильные и слабые стороны, основные тенденции развития предприятия и отрасли. «К сильным сторонам компании относятся высокий уровень технической компетентности сотрудников, широкое использование

современных технологий (Docker, Kubernetes, микросервисы, облачные решения), гибкость в подходе к решению задач клиентов» [11]. Среди слабых сторон можно выделить ограниченные кадровые ресурсы при необходимости быстрого масштабирования проектов и зависимость от квалификации отдельных специалистов. Основными тенденциями развития отрасли являются дальнейшее распространение DevOps-практик, автоматизация процессов разработки, переход к облачным инфраструктурам и рост интереса к микросервисным архитектурам.

Подробное описание функций и их важности в процессе деятельности объекта. «Отдел инфраструктуры компании выполняет функции по обеспечению бесперебойной работы CI/CD-процессов», разработке и поддержке средств автоматизации развертывания, мониторингу состояния инфраструктуры и обеспечению резервного копирования данных. Деятельность отдела критически важна для стабильной работы всех разработанных решений [13].

Организационная структура, распределение функций между элементами. Отдел инфраструктуры входит в состав технического подразделения и подчиняется техническому директору. «Функции внутри отдела распределяются между DevOps-инженерами, системными администраторами и специалистами по мониторингу. Каждый сотрудник отвечает за определённые компоненты инфраструктуры: серверные платформы, автоматизацию процессов развертывания, настройку систем резервного копирования и мониторинга.»

Категории и численность работающих и их информационные потребности. В отделе инфраструктуры работают DevOps-инженеры и системные администраторы. Средняя численность подразделения составляет 5–10 человек. Основные информационные потребности сотрудников включают доступ к системам контроля версий, CI/CD-серверам, средствам мониторинга и конфигурационным базам данных.

Виды и количество материальных ресурсов и оборудования Отдел инфраструктуры использует серверы для размещения сервисов TeamCity и систем мониторинга, рабочие станции для специалистов, сетевое оборудование для организации внутренней инфраструктуры, системы резервного копирования и хранения данных.

Уровень автоматизации в отделе высокий. «Основные процессы сборки, тестирования, развёртывания и мониторинга программных решений автоматизированы с использованием современных CI/CD-инструментов, контейнеризации и скриптов автоматизации» [12].

«Материальные потоки подразделения включают закупку серверного оборудования. Финансовые потоки – это расходы на поддержку ИТ-инфраструктуры. Информационные потоки состоят из данных о сборках программного обеспечения, отчётов систем мониторинга и данных резервного копирования» [21].

1.2 Анализ функционирования объекта исследования

В рамках проведения обследования деятельности объекта исследования был проведён анализ документооборота, связанного с процессами автоматизации развертывания программных решений. Для сбора информации использовались методы интервьюирования сотрудников отдела инфраструктуры и анализа внутренней документации.

На объекте функционируют как входные, так и выходные документы, относящиеся к процессу управления развертыванием.

В качестве входных документов используются заявки на развертывание новых версий приложений, предоставляемые разработчиками или менеджерами проектов. Данные заявки направляются в отдел

инфраструктуры с использованием корпоративных каналов связи – электронной почты или мессенджер Slack.

В процессе исполнения заявки DevOps-инженеры формируют скрипты запуска развертывания, предназначенные для автоматизации передачи кода на целевые окружения. Скрипты создаются с использованием текстовых редакторов или специализированных интегрированных сред разработки и сохраняются в репозиториях контроля версий Gitea [5].

Результатом выполнения процессов развертывания является «лог выполнения операций, формируемый средствами системы CI/CD TeamCity» [1]. Логи содержат информацию о ходе и результатах развертывания и доступны сотрудникам для анализа успешности выполнения процедур.

«CI/CD (Continuous Integration and Continuous Delivery) – это набор практик и методологий в сфере разработки программного обеспечения, направленных на автоматизацию и улучшение процессов разработки, тестирования и доставки приложений. Основная цель CI/CD – обеспечить непрерывное интегрирование изменений в код (Continuous Integration) и непрерывное развертывание приложений (Continuous Delivery), что способствует более быстрой, надежной и эффективной разработке и поставке программного обеспечения» [11].

Обобщённая информация по документообороту представлена в таблице 1.

«Для анализа и проектирования программных систем используют язык моделирования, включающий процедуры для решения вопросов моделирования предметной области и требований» [22].

Таким языком является «UML (Unified Modeling Language – унифицированный язык моделирования), разработанный компанией Rational Software Corporation для унификации лучших свойств, которыми обладали более ранние методы нотации» [22].

В существующей («как есть») практике процесс управления развертыванием осуществляется вручную, с использованием электронной

почты или мессенджеров Slack для подачи заявок на деплой. Это создаёт значительную зависимость от человеческого фактора, увеличивает время реакции и снижает воспроизводимость процесса [6].

Таблица 1 – Сводный состав документации на предприятии

№ п/п	Наименование документа	Кто готовит (источник)	Получатель (адресат)	Формат хранения	Инструмент (ТС и ПО для создания)	Количество экземпляров в год	Трудоёмкость подготовки одного документа, мин	Методы защиты
	Входные документы							
1	Заявка на развертывание новой версии ПО	Разработчик / Менеджер проекта	DevOps-инженер	Электронная форма (мессенджер, почта)	Slack, Email	120	10 мин	Ограничение доступа к каналам связи
	Выходные документы							
1	Скрипт запуска процесса развертывания	DevOps-инженер	Система TeamCity	Файл скрипта (.sh,)	IDE / Текстовый редактор	100	20 мин	Контроль версий (Gitea)
2	Лог выполнения процесса деплоя	Система TeamCity (автоматически)	Разработчик, DevOps-инженер	Электронный файл	Автоматическая генерация	500	0 мин (автоматически)	Ограничение доступа через учетные записи

Процесс функционирования в текущем состоянии «как есть» включает следующие основные этапы:

1. Инициирование заявки на развертывание. Инициатор (разработчик или менеджер проекта) формирует заявку на выполнение развертывания новой версии программного продукта.

2. Заявка направляется DevOps-инженеру через корпоративную почту или мессенджер.

3. Анализ заявки и подготовка скрипта развертывания. DevOps-инженер вручную проверяет полноту и корректность заявки.

4. Подготавливает скрипт запуска или параметры сборки в TeamCity на основе внутренних шаблонов.

5. Инициация процесса развертывания в TeamCity. Через веб-интерфейс TeamCity вручную настраивается задача и запускается процесс сборки/деплоя [1].

6. Мониторинг выполнения развертывания. Инженер следит за прогрессом через интерфейс TeamCity, анализирует логи в случае сбоев. «CD включает в себя непрерывный мониторинг состояния приложения в продакшн. Метрики и логи собираются и анализируются в реальном времени, что позволяет оперативно реагировать на любые проблемы и принимать меры по их устранению. При возникновении ошибок процесс останавливается и требует вмешательства» [18].

7. Фиксация результатов выполнения. Результаты выполнения сохраняются в системе. Информация о статусе выполнения передаётся заинтересованным сторонам вручную (по запросу).

8. Реагирование на ошибки. В случае неуспешного деплоя инженер повторно анализирует ошибки, вносит правки в скрипт и инициирует запуск заново.

«Для описания, текущего состояния процессов «как есть» в рамках анализа функционирования объекта исследования была построена диаграмма вариантов использования (use case)», отражающая ключевые этапы ручного CI/CD в организации [22].

«Диаграммы вариантов использования отображают общие особенности функционирования моделируемой системы без рассмотрения ее внутренней структуры. Диаграммы вариантов использования представляют также действующих лиц (actor), инициирующих варианты использования системы» [22].

Диаграмма показывает участие двух ролей – инициатора заявки (разработчика или менеджера проекта) и исполнителя (DevOps-инженера). Инициатор вручную формирует заявку на развертывание программного

продукта и отправляет её через мессенджер или корпоративную почту. DevOps-инженер анализирует заявку, проверяет её полноту, вручную подготавливает параметры и скрипт сборки, инициирует запуск в TeamCity, следит за выполнением, анализирует логи и вручную информирует инициатора о результате.

Модель на рисунке 2 наглядно демонстрирует, что все ключевые действия выполняются вручную и требуют участия инженера, что приводит к высокой трудозатратности и сниженной скорости реакции. «Каждый этап процесса сопровождается потенциальными рисками: неполные заявки, задержки в коммуникации, человеческие ошибки при настройке или передаче параметров, отсутствие централизованного контроля и автоматизированной обратной связи» [21].

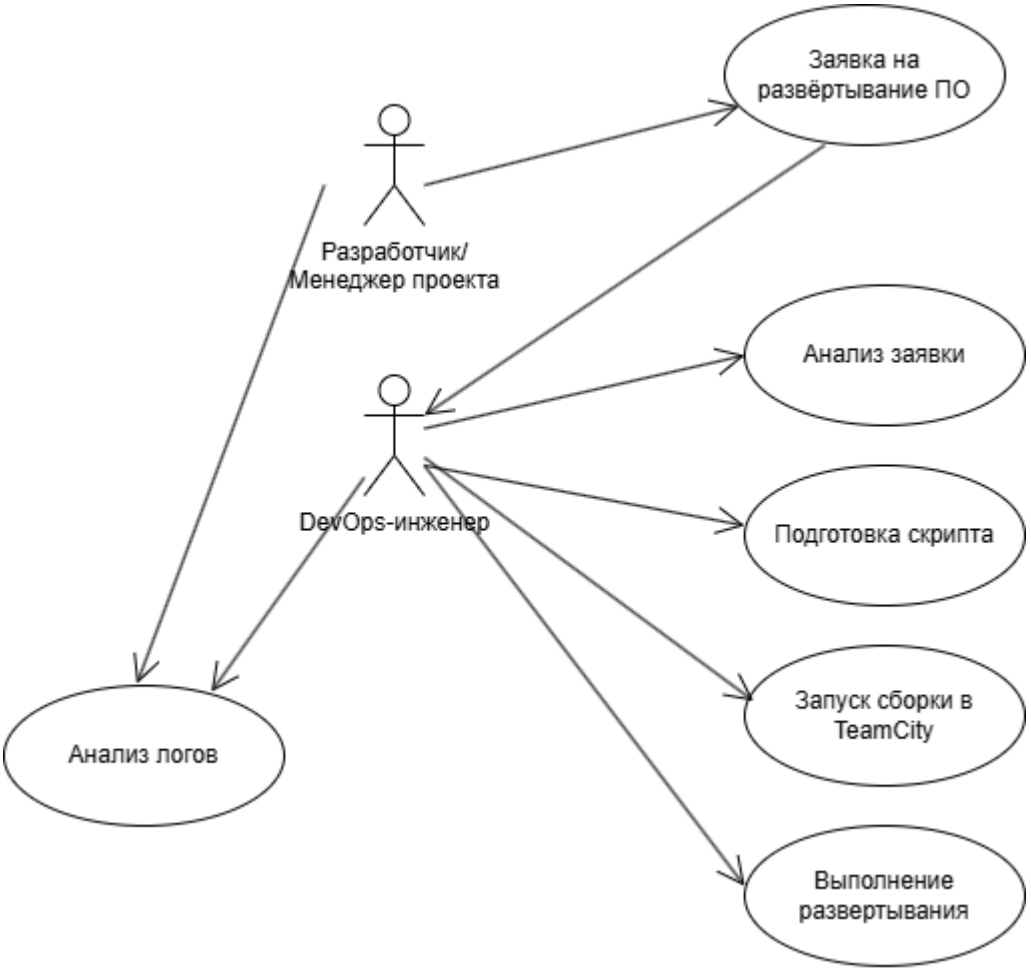

Рисунок 2 – Диаграмма UseCase «Как есть»

В результате анализа были выявлены основные недостатки текущей схемы:

– высокая трудоёмкость обработки заявок, достигающая 10–30 минут на одну операцию;

– недостаточная оперативность, особенно при росте количества задач;

– отсутствие автоматизированного сбора управленческих показателей;

– дублирование информационных потоков при передаче параметров;

– слабая формализация статусов и отсутствующая система уведомлений;

– неполнота и противоречивость передаваемых данных;

– отсутствие централизованного аудита и истории исполнения заявок.

На основании выявленных проблем предложено основное направление совершенствования – разработка и внедрение программного средства Deploy-бота. Данный бот должен обеспечить автоматизацию процесса: от приёма заявки в мессенджере (Slack) до автоматической валидации параметров, запуска сборки через API TeamCity и последующего выполнения деплоя в целевом окружении. Такое решение позволит [19]:

– снизить ручные трудозатраты и ускорить цикл развертывания;

– устранить ошибки, вызванные человеческим фактором;

– обеспечить прозрачность процессов и доступ к централизованным логам;

– предоставить инициатору заявки автоматическую обратную связь и статус исполнения;

– собрать статистику выполнения и упростить аудит.

Таким образом, внедрение автоматизированного решения на базе Deploy-бота позволит устранить выявленные узкие места и перейти к современной, устойчивой и масштабируемой архитектуре CI/CD с минимальным участием человека.

1.3 Определение цели и задач проектирования информационной системы

Целью разработки информационной системы (ИС) является автоматизация процессов управления сборкой и развертыванием программных решений в рамках CI/CD-конвейера с использованием TeamCity. Основное внимание уделяется устранению выявленных недостатков существующей практики: высокой трудоёмкости, низкой оперативности, отсутствию централизованного контроля, слабой прослеживаемости и дублированию данных.

«Использование методологий в интересах организации преследует простую и вполне понятную цель: организовать процесс создания информационной системы и обеспечить управление этим процессом таким образом, чтобы созданная ИС гарантированно отвечала потребностям организации, а также соответствовала предъявляемым к ней требованиям. Таким образом, любая методология проектирования ИС учитывает те же аспекты, на которых основана работа с любыми проектами: качество ИС, сроки создания ИС и бюджет на создание ИС» [18].

«Чтобы создание ИС стало эффективным с точки зрения качества, сроков и бюджета, используются исчерпывающие описания различных процессов на протяжении всего жизненного цикла информационной системы, в результате чего создание становится более простым, удобным и выгодным. Кроме того, методология проектирования корпоративных ИС включает в себя возможности для сопровождения, модификации и наращивания ИС, а также позволяют «вписать» создаваемую ИС в уже имеющуюся ИТ-инфраструктуру» [18].

При проектировании архитектуры системы были учтены основные требования к корпоративным ИС, изложенные в [14].

Цель разработки может быть представлена в виде двух подцелей:

Улучшение экономических показателей функционирования процесса развертывания ПО:

– сокращение времени обработки одной заявки на развертывание;

– уменьшение количества сбоев и ошибок при деплое за счёт снижения влияния человеческого фактора;

– снижение нагрузки на DevOps-инженеров и высвобождение их ресурсов для выполнения более сложных и креативных задач;

– сокращение простоев в процессе внедрения новых версий программного обеспечения.

Повышение качества обработки информации:

– повышение скорости обработки заявок и информирования участников процесса;

– повышение достоверности и полноты передаваемых параметров за счёт использования структурированной формы;

– обеспечение прослеживаемости и журналирования всех этапов обработки заявок;

– интеграция с TeamCity для прямого запуска процессов без участия человека;

– формирование единого хранилища информации по развертываниям для последующего анализа и отчётности.

Проектируемая система – Deploy-бот для автоматизации CI/CD-процессов через TeamCity. «С точки зрения классификации, ИС относится к информационно-управляющим системам» [17], «реализующим функции оперативного и частично аналитического управления с элементами автоматизированного документооборота и сервисного взаимодействия между участниками CI/CD-процесса» [16].

Состав задач отображен в таблица 2

Состав автоматизируемых процессов:

1. Подача заявки на развертывание;
2. Валидация параметров заявки;

3. Автоматическая генерация и запуск скрипта развертывания в TeamCity;
4. Уведомление участников о ходе выполнения и результате;
5. Формирование логов и архива заявок;

Таблица 2 – Состав задач в каждом процессе

Процесс	Задачи
Регистрация заявки	Прием данных от пользователя, проверка корректности, сохранение в базе
Обработка заявки	Верификация параметров, проверка готовности окружения
Запуск деплоя	«Формирование параметров запуска, инициализация процесса в TeamCity» [1]
Контроль выполнения	Мониторинг статуса, сбор логов, обработка ошибок
Уведомление	Отправка результатов инициатору заявки и другим заинтересованным сторонам
Хранение информации	Формирование истории выполненных действий, отчётность

Таким образом, проектируемая ИС направлена на повышение эффективности CI/CD-процессов за счёт максимальной автоматизации, стандартизации и прозрачности этапов развертывания ПО, снижая зависимость от ручных действий и минимизирую вероятность ошибок.

1.4 Обзор и анализ существующих разработок, выбор технологии проектирования

«На данный момент процесс развертывания программных решений с использованием CI/CD в большинстве компаний, включая рассматриваемую организацию, преимущественно осуществляется вручную или с помощью частично автоматизированных инструментов, таких как TeamCity, Jenkins и GitLab CI» [11]. В таблице3 представлен анализ систем.

Эти решения предоставляют широкий спектр возможностей для автоматизации сборки, тестирования и деплоя, однако их эксплуатация часто требует непосредственного участия DevOps-инженеров и не исключает влияние человеческого фактора.

Проведённый анализ показывает, что ни одно из стандартных решений, такие как Jenkins, GitLab CI, Octopus Deploy и TeamCity не обеспечивает

комплексной автоматизации, особенно в части упрощения взаимодействия между участниками процесса развертывания через привычные коммуникационные платформы. В связи с этим проектируется собственный сервис как надстройка над TeamCity, так как в организации ООО «Свик» уже приобретена лицензия данного ПО.

Таблица 3 – Анализ существующих разработок

Решение	Описание	Интеграция с мессенджерами	Преимущества и не достатки
Jenkins[4]	Популярная open-source CI/CD платформа с широкими возможностями кастомизации и плагинами.	Есть через API	«Требует настройки скриптов и участия DevOps, меньше удобства из коробки.» [4]
GitLab CI [2]	Встроенная CI система в экосистеме GitLab с поддержкой вебхуков и бот-интеграций.	Есть через API	«Интеграция требует кастомной разработки, ограничена экосистемой GitLab.» [2]
TeamCity	Коммерческая CI/CD платформа от JetBrains с мощной автоматизацией сборок, тестирований и развертываний.	Есть через API	«Из коробки поддерживает интеграцию с Slack, и email (Только уведомления), гибкая настройка уведомлений и управления процессами через REST API и Kotlin DSL; поддержка Docker, Kubernetes и популярных VCS; высокая стабильность и масштабируемость;» [1]
Octopus Deploy [3]	Специализированный инструмент для автоматизации развертывания с визуальным управлением релизами.	Есть через API	«Требует лицензии и дополнительной инфраструктуры; интеграция с мессенджерами ограничена, не покрывает весь цикл CI/CD.» [3]

Deploy-bot – серивис ориентированный на удобную подачу и исполнение заявок через интерфейс Slack, что позволяет централизовать управление развертываниями без необходимости постоянного привлечения специалистов.

Таким образом, обосновано создание оригинальной информационной системы, адаптированной под конкретные требования организации и существующую инфраструктуру.

1.5 Выбор и обоснование проектных решений

В данном подразделе проводится обоснованный выбор проектных решений по основным обеспечивающим подсистемам информационной системы Deploy-бота, предназначенной для автоматизации процессов управления сборкой и развертывание программного обеспечения с интеграцией в TeamCity. Эти решения сформированы на основе анализа предметной области, выявленных ранее недостатков и требований к функциональности, безопасности, надёжности и производительности разрабатываемой системы.

«Технологическое обеспечение определяет способ организации процессов ввода, передачи, обработки и выдачи информации в рамках CI/CD.» [11] В текущем ручном подходе к управлению деплоем наблюдаются проблемы в виде отсутствия формализованной подачи заявок, постоянного участия DevOps-инженеров в повторяющихся рутинных операциях, а также большого количества ошибок, связанных с человеческим фактором. В проектируемой системе технологический процесс полностью автоматизируется: заявки подаются через Slack с проверкой всех параметров на этапе ввода, после чего автоматически запускаются сборки и развёртывания через TeamCity API. Такой подход исключает типовые ошибки, ускоряет цикл поставки и делает процесс управляемым и воспроизводимым.

Программное обеспечение разрабатывается с применением сразу нескольких языков программирования, каждый из которых выполняет свою функциональную роль. Основная логика бота реализована на языке Python [9], который обеспечивает работу с SlackAPI [6].

Как указывается на официальном сайте языка, «Go предлагает встроенные средства конкурентного программирования, что делает его

удобным выбором для разработки высоконагруженных систем», позволяющий эффективно обрабатывать несколько заявок одновременно и работать с TeamCity API [1] на низком уровне [7].

Скрипты деплоя, исполняемые на сервере, написаны на Bash, что даёт «гибкость при работе в Unix-среде и удобство встраивания в пайплайны» [1]. Такое распределение задач между языками позволяет достичь баланса между читаемостью, производительностью и удобством сопровождения. В качестве СУБД используется «PostgreSQL, как надёжное и масштабируемое решение, а взаимодействие между компонентами осуществляется по протоколу HTTPS с использованием JSON-формата данных» [20].

Информационное обеспечение системы включает как внемашинные, так и внутримашинные элементы. Внемашинную часть составляют классификаторы окружений (dev, test, prod), справочник доступных проектов и шаблоны параметров деплоя. Внутримашинная часть – это централизованная база данных PostgreSQL, в которой хранится информация о пользователях, история заявок, статусы выполнения, параметры запусков и логи операций. Данные структурированы таким образом, чтобы обеспечить их логическую связанность, минимизировать дублирование и упростить формирование отчётов. Архитектура базы – клиент-серверная, с разграничением прав доступа и возможностью резервного копирования. «Клиент-серверные (двухзвенные) системы значительно снижают нагрузку на сеть, так как клиент общается с данными через специализированного посредника сервер БД, который размещается на машине с базой данными» [20]. Все формы взаимодействия пользователя с системой (например, подача заявки) реализуются через Slack с контролем полноты и корректности вводимых данных.

Техническое обеспечение предполагает развёртывание всех компонентов системы на серверной инфраструктуре, соответствующей современным требованиям по производительности и отказоустойчивости. Минимальные характеристики сервера: 8 физических ядер, 32 ГБ

оперативной памяти, SSD-накопитель от 200 ГБ, с возможностью горизонтального масштабирования. «Это обусловлено необходимостью одновременной обработки множества запросов, хранения логов и постоянного взаимодействия с CI/CD-сервером TeamCity» [1]. Предпочтительно использовать облачные платформы (например, Yandex Cloud) либо локальные серверы предприятия при наличии соответствующего SLA. Сеть должна обеспечивать стабильное соединение по защищённому протоколу HTTPS, а при удалённом доступе – VPN-соединение.

Математическое обеспечение разрабатываемой системы ориентировано на поддержку алгоритмов постановки задач в очередь, распределения заявок по приоритетам и автоматического повторного запуска деплоя при возникновении ошибок. «Также используются методы логической обработки логов TeamCity и определения успешности операций по ключевым меткам. Основываясь на принципах систем массового обслуживания, система автоматически распределяет заявки по времени и окружениям, оптимизируя ресурсную нагрузку и исключая конфликты при параллельных развёртываниях» [1].

Лингвистическое обеспечение включает языки, форматы и протоколы, обеспечивающие взаимодействие между компонентами системы и пользователем. «В пользовательском интерфейсе применяются простые команды на английском языке с возможностью локализации. Внутреннее взаимодействие между сервисами реализовано через REST API, основанное на JSON.» Программная логика написана на Python и Go [7], тогда как конфигурационные скрипты и исполняемые блоки – на Bash [10]. Такая структура обеспечивает простоту сопровождения, расширяемость и модульность архитектуры.

«Эргономическое обеспечение направлено на создание удобного, минималистичного и интуитивно понятного интерфейса» [17].

Slack-бот взаимодействует с пользователем пошагово, позволяя без технических знаний подать корректную заявку. Статусы задач, ошибки и

результаты выполнения деплоя подаются в текстовой и визуально маркированной. Таким образом, разработчики и менеджеры могут использовать систему с ПК или мобильного устройства, не заходя в TeamCity, что повышает доступность и снижает барьер вхождения.

Обеспечение информационной безопасности реализуется на нескольких уровнях. «На уровне аутентификации предусмотрена идентификация пользователя по Slack ID» с привязкой к ролям в системе. Все данные передаются по защищённому протоколу HTTPS, логируются действия пользователя, обращения к API и результаты операций. Конфиденциальные параметры деплоя (токены, ключи, секреты) хранятся в зашифрованном виде. Доступ к базе данных и компонентам системы разграничен по ролям [6].

Таким образом, принятые проектные решения по всем обеспечивающим подсистемам направлены на создание современной, гибкой и надёжной информационной системы, способной обеспечить автоматизацию процессов CI/CD с высокой степенью прозрачности, защищённости и удобства для конечного пользователя. Использование современного технологического стека и модульной архитектуры гарантирует возможность масштабирования, поддержки и дальнейшего развития проекта.

2 Проектная часть

2.1 Разработка функционального обеспечения

В аналитической части выпускной квалификационной работы выявлено, что существующий процесс «как есть» развертывания новых версий программного обеспечения осуществляется преимущественно вручную и характеризуется значительным количеством неформализованных операций, выполняемых DevOps-инженерами. Такие этапы, как прием и верификация заявок, инициирование сборки в CI-среде, мониторинг выполнения и информирование инициатора, реализуются вручную, что приводит к существенным временным затратам, повышенному риску возникновения ошибок, недостаточной прозрачности и ограниченным возможностям контроля над процессом в целом. Данные проблемы коррелируют с типичными ограничениями традиционных ручных практик CI/CD, что подтверждается в ряде научных и практических источников.

В рамках проектной части была разработана целевая модель «как должно быть», основанная на реинжиниринге существующих бизнес-процессов с целью устранения выявленных недостатков и повышения эффективности. Ключевым элементом «как должно быть» модели является внедрение автоматизированного сервиса – Deploy-бота, предназначенного для приёма команд на развертывание через корпоративный мессенджер Slack, их обработки и последующего запуска соответствующих процессов в CI/CD-инфраструктуре без необходимости непосредственного участия DevOps-инженера.

Для формализации функциональной архитектуры проектируемой системы была построена диаграмма вариантов использования (Use Case), демонстрирующая взаимодействие между двумя основными субъектами: инициатором заявки (разработчиком или менеджером) и исполнителем

(Deploy-ботом). «Моделирование выполнено с применением языка UML, что обеспечивает универсальность представления и пригодность модели для последующей реализации» [22].

В «как должно быть» модели реализованы следующие принципиальные изменения по сравнению с текущим состоянием:

Формализация процесса подачи заявки. Пользователь инициирует развертывание посредством чат-бота в Slack [6], используя стандартизированную форму, при этом бот осуществляет предварительную валидацию параметров на стороне клиента, обеспечивая корректность вводимых данных.

«Автоматизация взаимодействия с TeamCity». Deploy-бот формирует и отправляет запросы к API TeamCity для запуска процессов сборки, используя предопределённые шаблоны параметров, что исключает необходимость ручного вмешательства [1].

Разделение процессов CI и CD:

1. «Continuous Integration (CI). Этот аспект CI/CD подразумевает автоматизацию процесса интеграции кода разработчиков в общий кодовый репозиторий. Каждое изменение в коде автоматически интегрируется с существующим кодом, и после этой интеграции проводятся автоматические тесты. Это позволяет обнаруживать и устранять конфликты и ошибки на ранних этапах разработки, обеспечивая стабильность и качество кода» [11].

2. «Continuous Delivery (CD). Этот аспект CI/CD связан с автоматизированным процессом развертывания приложений. При наличии CD каждое успешное изменение в коде, прошедшее CI, может быть автоматически развернуто в тестовое или staging-окружение, где оно проходит дополнительное тестирование. Это позволяет гарантировать, что приложение всегда готово к поставке в продуктивную среду, и сокращает время между завершением разработки и выпуском новой версии» [11].

Сборка проекта осуществляется средствами TeamCity, а последующее развертывание в целевых окружениях реализуется отдельным модулем,

разработанным на языке Go, который посредством GoSSH управляет серверами напрямую, обеспечивая гибкость и масштабируемость.

Сбор и анализ логов. По завершении развертывания бот осуществляет сбор и анализ логов, предоставляя инициатору, детализированный отчёт о результатах выполнения.

Снижение зависимости от DevOps-инженеров. «Роль специалистов DevOps сводится к сопровождению инфраструктуры и совершенствованию скриптов, в то время как повседневные операции по развертыванию полностью автоматизированы» [11].

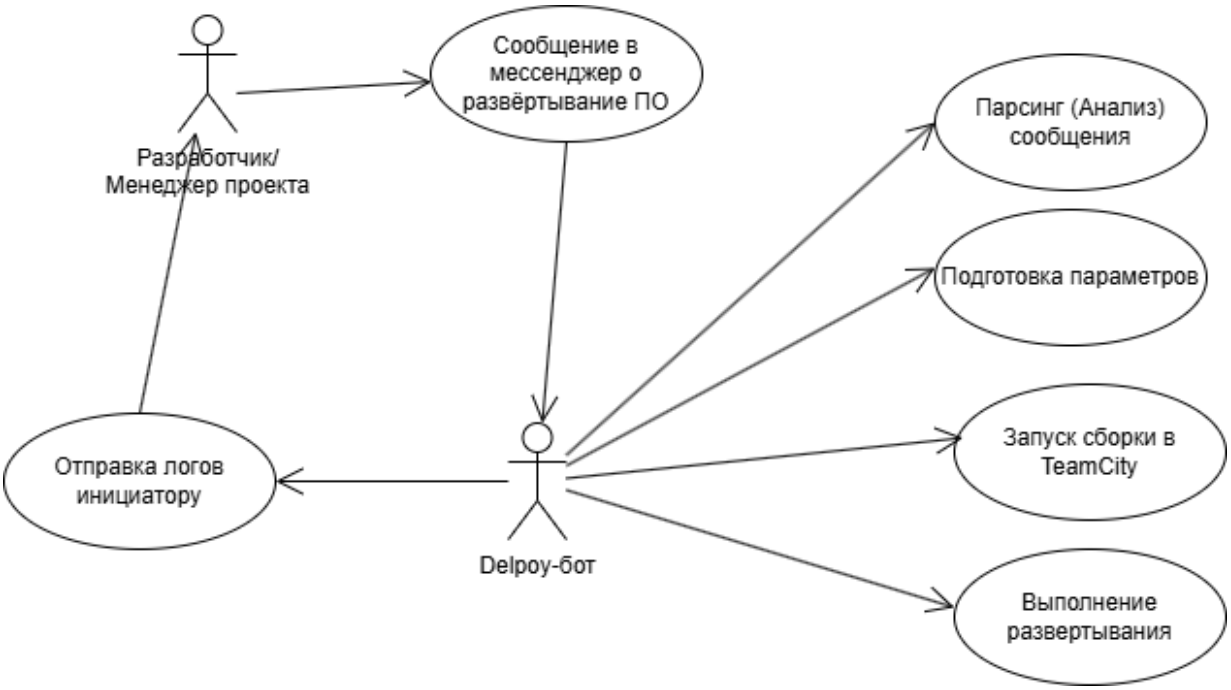

Рисунок 3 – Диаграмма UseCase «Как должно быть»

Диаграмма Use Case отражает модель взаимодействия в целевом состоянии на рисунке №3:

1. Инициатор (разработчик или менеджер) взаимодействует с Deploy-ботом через интерфейс Slack.
2. Deploy-бот выполняет валидацию входных данных.
3. Бот инициирует сборку посредством обращения к TeamCity API.

4. По завершении CI-пайплайна бот запускает процесс деплоя на серверах через Go-модуль с использованием библиотеки GOSSH [8].

5. После успешного завершения развертывания бот передаёт инициатору статус выполнения и соответствующие логи выполнения развертывания.

Таким образом, предложенное функциональное обеспечение обеспечивает комплексную автоматизацию ключевых этапов CI/CD, что «способствует снижению временных затрат, устранению дублирования данных и минимизации влияния человеческого фактора, а также повышает управляемость и прозрачность процессов» [11].

В результате процесс развертывания становится более оперативным, предсказуемым и удобным как для технических специалистов, так и для менеджеров проектов, что соответствует современным требованиям к эффективному управлению жизненным циклом программного обеспечения.

2.2 Разработка информационного обеспечения

2.2.1 Используемые классификаторы и системы кодирования

В информационной системе Deploy-бот используются классификаторы и кодовые обозначения, обеспечивающие структурированное и однозначное представление входной и справочной информации. Их основное назначение – стандартизация параметров деплоя и обеспечение корректной маршрутизации заявок на развертывание программных решений. Система классификаторов также упрощает автоматическую валидацию и обработку данных, поступающих через интерфейс Slack-бота.

В таблице 4 ниже представлена структура кодовых обозначений объектов и характеристика используемых классификаторов:

Таблица 4 – Состав задач в каждом процессе

№	Наименование кодируемого множества объектов	Значность кода	Система кодирования	Система классификации	Вид классификатора
1	Окружения (dev, stage, prod)	3	Серийная	Иерархическая	Локальный
2	Проекты	До 10 символов	Комбинированная	Отсутствует	Внутрисистемный справочник
3	Пользователи (Slack ID)	9 цифр	Порядковая	Отсутствует	Внутрисистемный
4	Статусы заявок	1 слово	Серийная	Иерархическая	Внутрисистемный
5	Типы сообщений/уведомлений	1–2 символа	Порядковая	Фасетная	Внутрисистемный

«Каждое значение в классификаторе уникально и может использоваться системой как ключ для доступа к параметрам шаблонов, правам пользователей, идентификации релизов и другим бизнес-операциям. Использование локальных и внутренних классификаторов обусловлено спецификой архитектуры системы, а также необходимостью динамического обновления данных в ходе эксплуатации» [17].

2.2.2 Характеристика нормативно-справочной и входной оперативной информации

Нормативно-справочная информация проектируемой системы включает постоянные справочники, необходимые для корректной валидации входных данных и формирования заявок на развертывание. Основными справочниками являются: справочник проектов, справочник окружений, справочник пользователей с ролями, справочник параметров деплоя. Эти данные администрируются через специальный интерфейс или напрямую через БД и используются на этапе подачи заявки в Slack-боте. Все справочники реализованы в виде отдельных таблиц в СУБД PostgreSQL, а

при заполнении полей в форме Slack -бота они подгружаются автоматически, обеспечивая пользователю только допустимые варианты.

Входной документ в данной системе представляет собой форму заявки на развертывание программного продукта, заполняемую пользователем через Slack-бот.

Таблица 5 – deployment_requests:

Поле	Идентификатор	Шаблон	Тип данных	Примечание
id	ID	UUID	UUID	Первичный ключ
user_id	UID	[0-9]+	INT	Внешний ключ на users
project_id	PID	[A-Z_]+	VARCHAR(20)	Внешний ключ на projects
environment_id	EID	[a-z]+	VARCHAR(10)	Внешний ключ на environments
version	VER	v1.2.3 / #456	VARCHAR(15)	Номер сборки или тег
status	STATUS	queued/running/...	VARCHAR(20)	Текущий статус заявки
created_at	T_CREATED	datetime	TIMESTAMP	Дата и время подачи заявки
finished_at	T_FINISHED	datetime/null	TIMESTAMP	Время завершения (если есть)

Все поля сопровождаются интерактивными подсказками, валидацией формата и автоматическим автозаполнением на основе данных из справочников. Источником данных выступает сам пользователь – разработчик или менеджер, авторизованный в системе. После заполнения все данные из формы поступают в таблицу №5 deployment_requests в базе данных.

Форма содержит следующие основные показатели:

- проект (выбирается из справочника проектов),
- версия (вводится вручную или выбирается из доступных билдов),
- окружение (выбирается из справочника окружений),
- комментарий (необязательный параметр),
- Slack ID пользователя (определяется автоматически).

Характеристики хранения:

1. Средняя длина записи: ~220 байт;

2. Количество записей: 10–30 в сутки;

3. Способ обращения: выборочный, по индексу status и created_at;

4. Организация хранения: централизованная PostgreSQL БД;

5. Длительность хранения: от 6 месяцев до 1 года.

Справочник environments содержит перечень доступных окружений (dev, stage, prod) и их технические характеристики. Справочник users – идентификаторы, роли, Slack. Эти таблицы обновляются вручную по мере необходимости. Частота актуализации – раз в месяц или при изменении оргструктуры. Объём актуализации – не более 10%.

Элементы СУБД PostgreSQL:

– Первичные и внешние ключи;

– Ограничения уникальности;

– Триггеры: автоматическое заполнение поля finished_at при завершении заявки;

– Валидация: ограничения на поля comment (длина ≤ 512 символов) и status (список допустимых значений).

2.2.3 Характеристика результатной информации

Результирующая информация, формируемая в процессе работы Deploy-бота, предназначена для оперативного контроля, анализа и информирования участников CI/CD-процесса. «Она обеспечивает прозрачность выполнения операций по развертыванию программных продуктов, а также повышает управляемость технической инфраструктуры» [13].

В отличие от классических ИС, результатная информация в системе Deploy-бота не представляется в виде печатных отчётов или сложных экранных форм. Она передаётся в интерактивной форме напрямую в мессенджере Slack в виде сообщений о статусе заявки на развертывание. Такие сообщения являются основным результатным документом и используются для оперативного управления и мониторинга.

Основные виды результатной информации включают:

1 Статус выполнения заявки (успешно, ошибка, выполняется);

2 Сводка по сборке (идентификатор билда, окружение, длительность);

3 Форматированный лог выполнения (ключевые этапы процесса, ошибки);

4 Временные метки начала и завершения процесса;

5 Служебный ID и ссылка на сборку в TeamCity.

Роль результатной информации:

1 Оперативное управление – позволяет DevOps-инженерам и разработчикам получать мгновенную обратную связь по действиям;

2 Отчётность – используется при разборе инцидентов или проверке истории изменений;

3 Обратная связь – улучшает взаимодействие между командой и системой.

«Формат сообщений оптимизирован под мессенджер: кратко, структурировано, без избыточных сведений. Все ключевые показатели представлены наглядно и доступны для восприятия даже в мобильном интерфейсе» [14].

Такая информация, хоть и не хранится постоянно в БД, может сохраняться во временные файлы или передаваться во внешнюю систему логирования, если архитектура проекта это предусматривает.

Таким образом, результатная информация в системе играет ключевую роль для участников CI/CD, обеспечивая высокую прозрачность, управляемость и своевременное принятие решений.

2.2.4 Информационная модель и ее описание

Проектируемая сервис – Deploy-бот для автоматизации CI/CD – базируется на логике взаимодействия между сущностями предметной

области: пользователями, заявками на деплой, окружениями и статусами выполнения. Хотя система не использует полноценную СУБД, как это бывает в классических ИС, построение логической (инфологической) и физической модели данных остаётся важным этапом проектирования, особенно в части поддержки кэширования, сериализации и интеграции с внешними API (TeamCity, Slack).

«Инфологическое проектирование базы данных отражает сущности предметной области и связи между ними, не учитывая физическую реализацию» [20].

Разрабатываемый сервис – Deploy-бот для автоматизации CI/CD – не использует полноценную систему управления базами данных (СУБД) в классическом понимании. Вместо этого данные обрабатываются на лету: в оперативной памяти, через сериализацию, кэширование или временное хранение в виде структур. Тем не менее, «логическая модель данных остаётся важной частью проектирования, так как позволяет формализовать внутренние сущности, их свойства и взаимосвязи» [17], а также подготовить архитектуру к потенциальному масштабированию или внедрению СУБД в будущем [20].

Логическая модель отражает основные объекты предметной области, с которыми взаимодействует Deploy-бот. К ним относятся:

User – пользователь (разработчик или менеджер), инициирующий деплой через Slack.

DeploymentRequest – заявка на выполнение развертывания, содержащая параметры запуска, окружение и другую необходимую информацию.

Environment – среда, в которую производится деплой (например, dev, stage, prod).

Status – состояние выполнения заявки (например, pending, running, success, failed).

Log – текстовые записи, отражающие этапы выполнения процесса.

Взаимосвязи между сущностями. Один пользователь может создавать множество заявок (User → DeploymentRequest).

Каждая заявка относится к одному окружению и имеет один статус (DeploymentRequest → Environment, DeploymentRequest → Status).

Каждая заявка может содержать несколько логов (DeploymentRequest → Log). «Логическая модель оформляется в виде диаграммы «сущность-связь» (ER-диаграммы), что обеспечивает наглядное представление структуры данных и их взаимосвязей» [21] на рисунке 4.

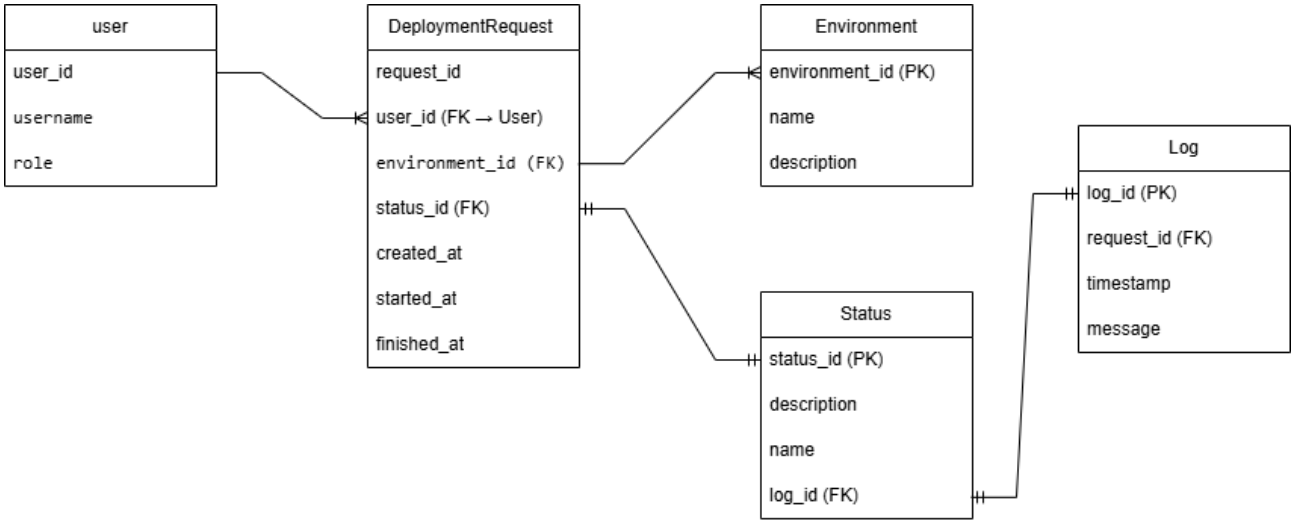

Рисунок 4 – Логическая схема данных

Физическая модель данных в текущей версии проекта отсутствует в виде постоянного хранилища, однако её структура может быть реализована с использованием реляционной СУБД, например, PostgreSQL, в случае необходимости расширения проекта. При этом предполагается использование следующих таблиц:

– users – идентификаторы Slack-пользователей, их имена и роли;

– environments – список окружений и их описания;

– deployment_requests – заявка с параметрами, автором, целевым окружением, временными метками и статусом;

– logs – привязанные к заявке записи с метками времени и описанием этапов выполнения.

Даже в отсутствие СУБД, такое логическое проектирование обеспечивает:

– структурированность данных;

– возможность упрощённого рефакторинга кода;

– базу для внедрения полноценной БД в будущем;

– поддержку механизмов сериализации, фильтрации и логирования.

Таким образом, логическая модель данных обеспечивает необходимую формализацию ключевых сущностей проекта и готовность к масштабированию. Она может быть реализована средствами ORM или JSON-хранилищ в зависимости от выбранного подхода при развитии проекта.

2.3 Разработка программного обеспечения

2.3.1 Структурная схема функций управления и обработки данных

Разработка структурной схемы функций управления и обработки данных в системе автоматизации CI/CD-процессов через Deploy-бот направлена на логическую декомпозицию всех операций, реализуемых информационной системой, и формирование сценариев взаимодействия пользователя с программным обеспечением. В данной системе функции делятся на две группы: служебные и основные.

К служебным функциям относятся операции, обеспечивающие поддержку и эксплуатацию системы. «Это валидация входных данных (например, проверка формата команды в чате), проверка прав доступа пользователя, протоколирование действий, а также инициализация

конфигурации системы при старте» [14]. Также сюда можно отнести операции по загрузке справочной информации об окружениях и настройках TeamCity, формирование логов и журналов активности.

«Основные функции – это выполнение ключевых бизнес-операций, ради которых и разрабатывается система» [14]. В случае Deploy-бота к ним относятся: приём заявки от пользователя на развертывание новой версии, валидация и маршрутизация команды, инициация процесса билда и деплоя через TeamCity API, отслеживание статуса выполнения, отправка результатов пользователю, отображение статуса ранее отправленных заявок, а также формирование отчетов или логов по завершению процесса. Все эти действия представляют собой узлы дерева функций с определенной иерархией, начиная от высокоуровневого запроса пользователя и заканчивая низкоуровневыми API-вызовами и ответами.

В рамках пользовательского взаимодействия реализован сценарий диалога в формате меню и команд, реализуемых через чат-бот в Slack. Базовый сценарий включает следующие кадры (этапы):

1. Пользователь вводит команду /deploy и указывает имя ветки, которую нужно развернуть на северах, если пользователь не указал никакого окружения (dev, stage, prod), то деплой происходит на окружение dev.

2. Бот получает команду, проверяет корректность, далее бот отправляет запрос в API TeamCity для запуска сборки, при это уведомляет пользователя реакцией на сообщения, что запустился деплой.

3. После завершения сборки в TeamCity, бот инициирует развертывание и информирует пользователя о ходе выполнения.

4. По завершении процесса бот присылает статус (успешно, ошибка) и лог файл на рисунках 5 и 6.

Команда /help даёт справку по возможным действиям.

Список команд бота развертывания:

- /deploy abort: запустить git merge --abort в локальном каталоге проекта

– /deploy abort_front: запустить git merge --abort в локальном каталоге проекта frontend
  – /deploy <branch> : развернуть backend;
  – /deploy <branch> -f : развернуть frontend;
  – /deploy <branch> -f2 : развернуть frontend (версия SPA);
– /deploy <branch> -g: развернуть WS-сервис;
– /deploy -p master - деплой master на production;
– /deploy -sb master - деплой master на sandbox;
– /deploy help: вывести сообщение HELP.

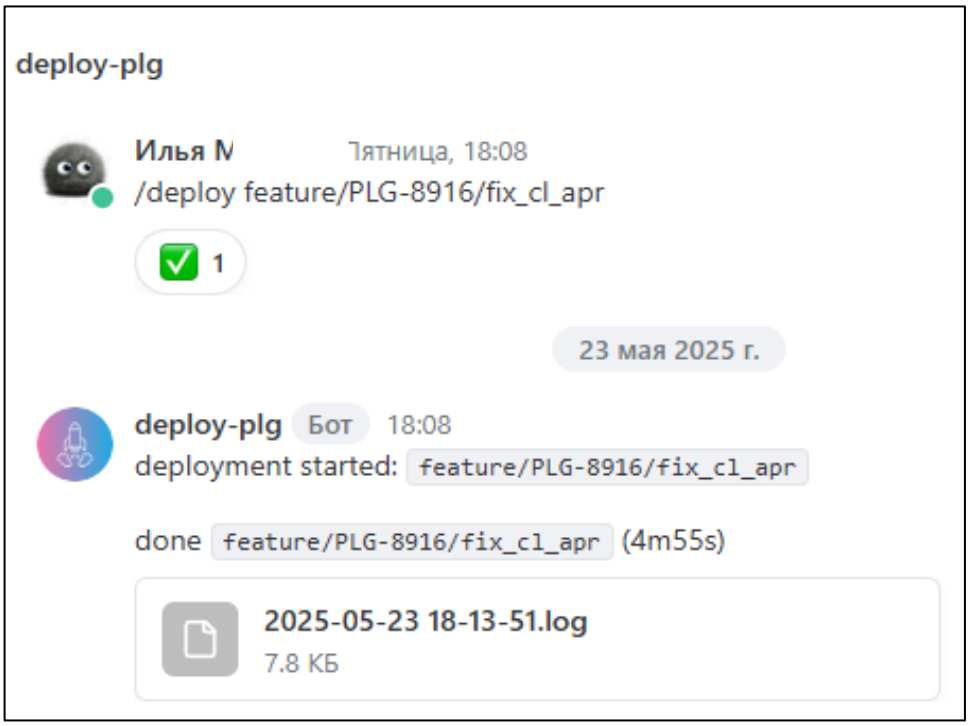

Рисунок 5 – Успешный статус деплоя

«Все этапы диалога связаны между собой через маршрутизатор логики взаимодействия и включают функции контроля ошибок и поддержки пользователя. В каждой точке сценария предусмотрены обработки исключений, повторные запросы, подсказки и подтверждения, что значительно повышает удобство использования системы» [15].

Таким образом, структурная схема функций Deploy-бота демонстрирует модульную и расширяемую архитектуру, где каждая

операция изолирована и может быть доработана или расширена без влияния на остальную часть системы. Это соответствует современным требованиям к надёжному CI/CD-инструменту и позволяет адаптировать бота под конкретные задачи и инфраструктуру компании.

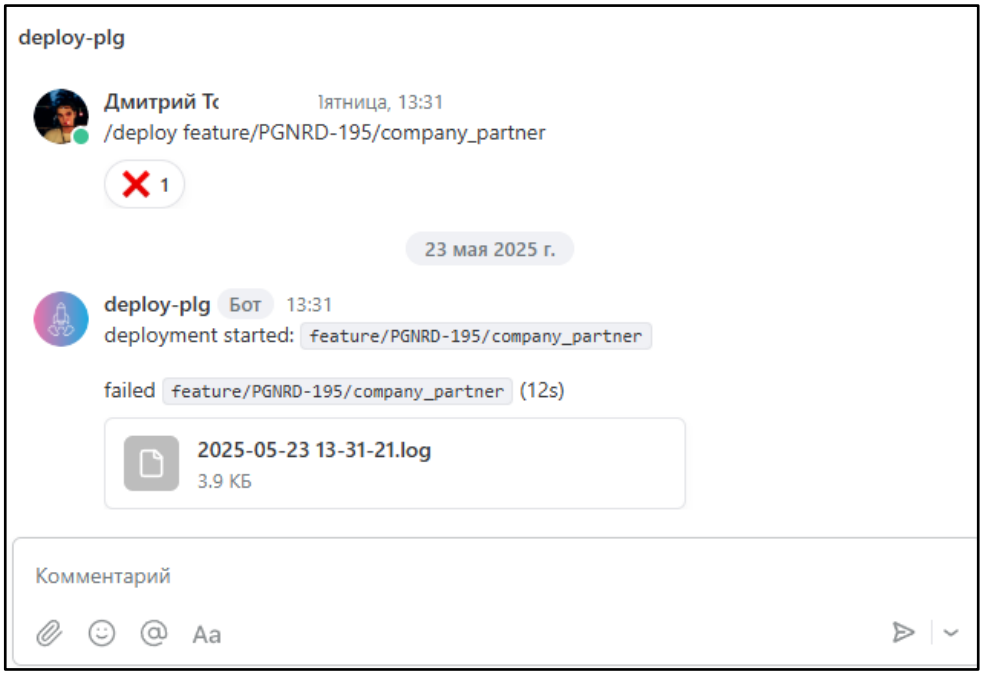

Рисунок 6 – Статус ошибки деплоя

2.3.2 Описание программных модулей

В основе архитектуры программного обеспечения лежит система Deploy-бот, обеспечивающая управление процессами развертывания (CI/CD) с интеграцией в TeamCity, мессенджеры Slack и внешние системы. Система имеет модульную структуру, включающую компоненты на нескольких языках программирования: Bash, Python и Go [12].

Общий состав архитектуры:

1. Deploy-бот – основное программное средство, выступающее посредником между пользователями и системой CI/CD.

2. Интерфейс взаимодействия – чат-бот, интегрированный в мессенджер Slack. Пользовательский ввод реализуется через команды [6].

3. «TeamCity – система, обеспечивающая непрерывную интеграцию и доставку программных продуктов, включающая богатый набор возможностей для автоматизации процессов сборки» [1].

4. PostgreSQL – используется как основная СУБД TeamCity, обеспечивает хранение метаданных о проектах, билдах, пользователях.

5. «Gitea – внутренняя система контроля версий, в которой размещаются исходные коды проектов» [5].

2.3.3 Схема взаимосвязи программных модулей и информационных файлов

Deploy-бот представляет собой распределённую систему, состоящую из нескольких программных компонентов, каждый из которых отвечает за выполнение определённого этапа процесса CI/CD. В рамках архитектуры бота обеспечивается чёткое взаимодействие между модулями, а также обмен данными между программными компонентами и внешними системами, такими как Slack, TeamCity и серверами развертывания.

Модуль взаимодействия с пользователем реализован на языке Python и использует библиотеку slack_sdk. Он принимает команды от пользователей через мессенджер Slack, проверяет корректность структуры входящих сообщений, а затем передаёт их далее по цепочке.

Парсинг-компонент, написанный на языке Go, принимает обработанное сообщение от Python-модуля, извлекает ключевые параметры (название проекта, ветку, окружение и др.) и подготавливает их к передаче в TeamCity и модуль деплоя.

Bash-обработчик выполняет роль промежуточного звена для CI-процесса: он взаимодействует с API TeamCity, формирует запрос на сборку и отслеживает её состояние. По завершении сборки, при успешном результате, инициируется выполнение деплой-операций.

Deploy-модуль, реализованный также на Go с использованием библиотеки gossh, подключается к удалённым серверам и выполняет инструкции деплоя (копирование артефактов, рестарт сервисов, миграции и т.д.). Логи выполнения операций сохраняются в лог-файл.

Все промежуточные и итоговые данные – включая параметры деплоя, статусы, пути к артефактам и журнал логов – передаются между модулями либо в виде JSON-сообщений, либо через файловые интерфейсы и REST API. Информация логирования организована в формате структурированных логов (.log) и хранится централизованно на выделенном сервере.

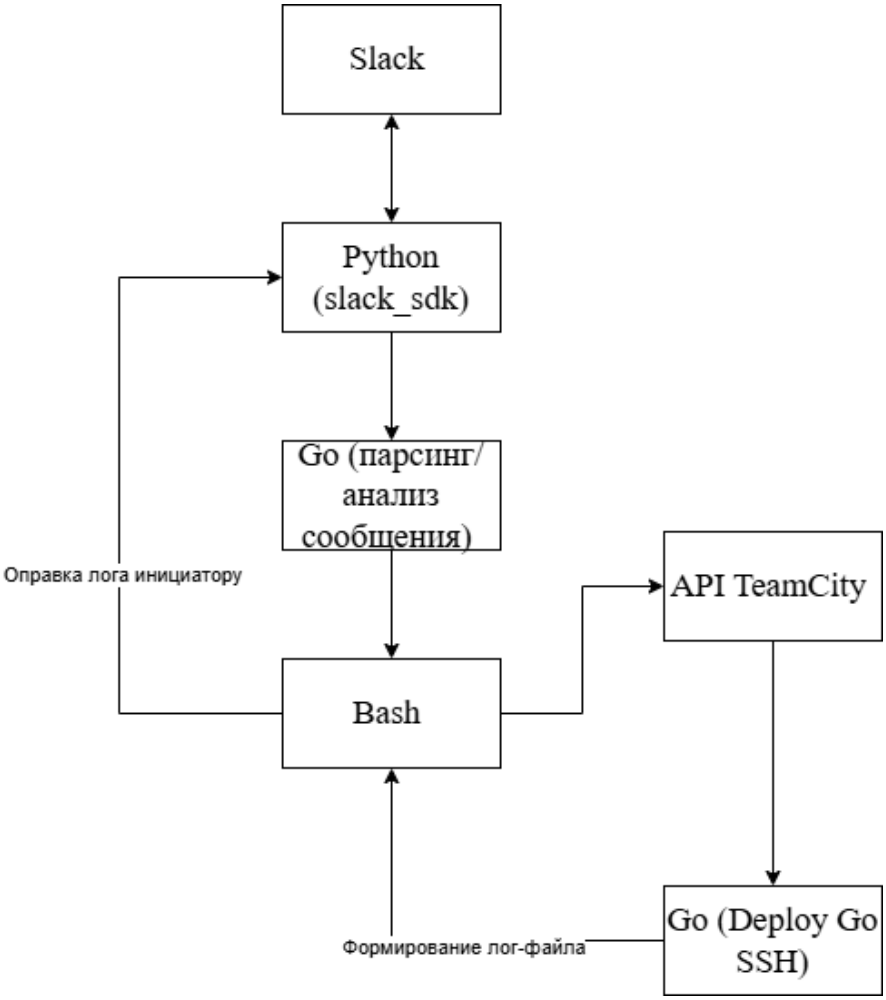

Рисунок 7 – Схема взаимосвязи программных модулей

Итоговая схема демонстрирует следующие взаимосвязи на рисунке №7:

Slack → Python (slack_sdk) → Go (парсинг) → Bash → API TeamCity → Go (Deploy via SSH) → Серверы

Лог-файлы → Python-модуль → Slack (отправка статуса пользователю)

Такая архитектура обеспечивает изоляцию ответственности, масштабируемость компонентов и устойчивость к сбоям отдельных модулей.

Таким образом, архитектура системы представляет собой чётко скоординированную структуру из компонентов, взаимодействующих по цепочке. Deploy-бот выступает в роли посредника между пользователем и TeamCity, организуя удобный, контролируемый и автоматизированный процесс развертывания, а логирование и интеграция с Gitea обеспечивают полноту информации и воспроизводимость операций.

2.3.4 Компоненты пользовательского интерфейса

Пользовательский интерфейс разрабатываемой информационной системы представляет собой интеграцию с мессенджером Slack, через которые осуществляется всё взаимодействие пользователя с системой Deploy-бота. Такой подход выбран осознанно: он снижает порог входа для конечного пользователя, исключает необходимость установки дополнительных программ и полностью соответствует современным трендам в DevOps-инфраструктуре. Мессенджеры служат графическим и логическим интерфейсом, через который пользователи подают команды, получают результаты, статусы и логи процессов CI/CD.

«При проектировании интерфейса особое внимание было уделено эргономике, поскольку эффективность взаимодействия человека и системы напрямую влияет на надёжность выполнения операций и скорость принятия решений.» [15].

«Интерфейс строится на принципах минимализма и последовательности. Отображаются только необходимые пользователю элементы: команды, уведомления и ссылки. Визуальное наполнение

интерфейса строго структурировано: важные элементы (например, успешный результат деплоя, ошибки или предупреждения) выделяются с помощью цветовой кодировки, эмодзи и маркеров, что повышает восприятие информации в условиях высокой загруженности специалистов» [19].

Каждое действие пользователя сопровождается ответной реакцией системы: подтверждение принятия команды, подсказка или результат на рисунках 8-10. Примеры диалоговых сценариев, /deploy – обращение к боту, является ключевым словом.

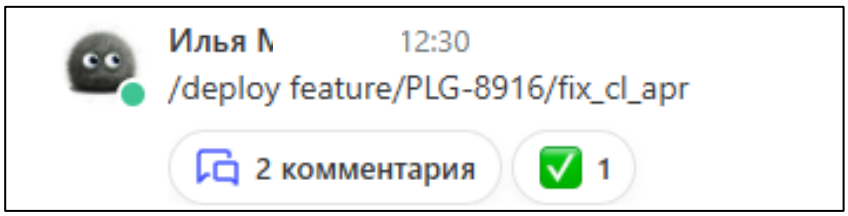

Рисунок 8 – Обращение к Deploy-bot

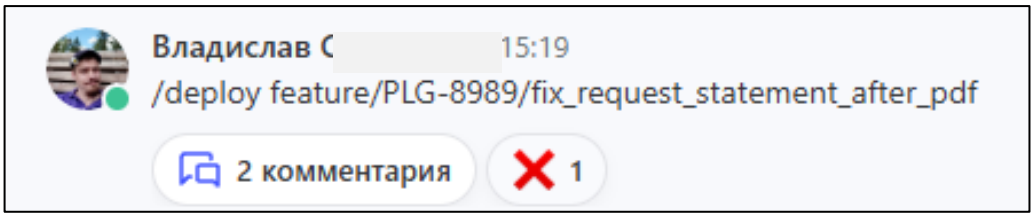

Рисунок 9 – Статус сборки «Ошибка»

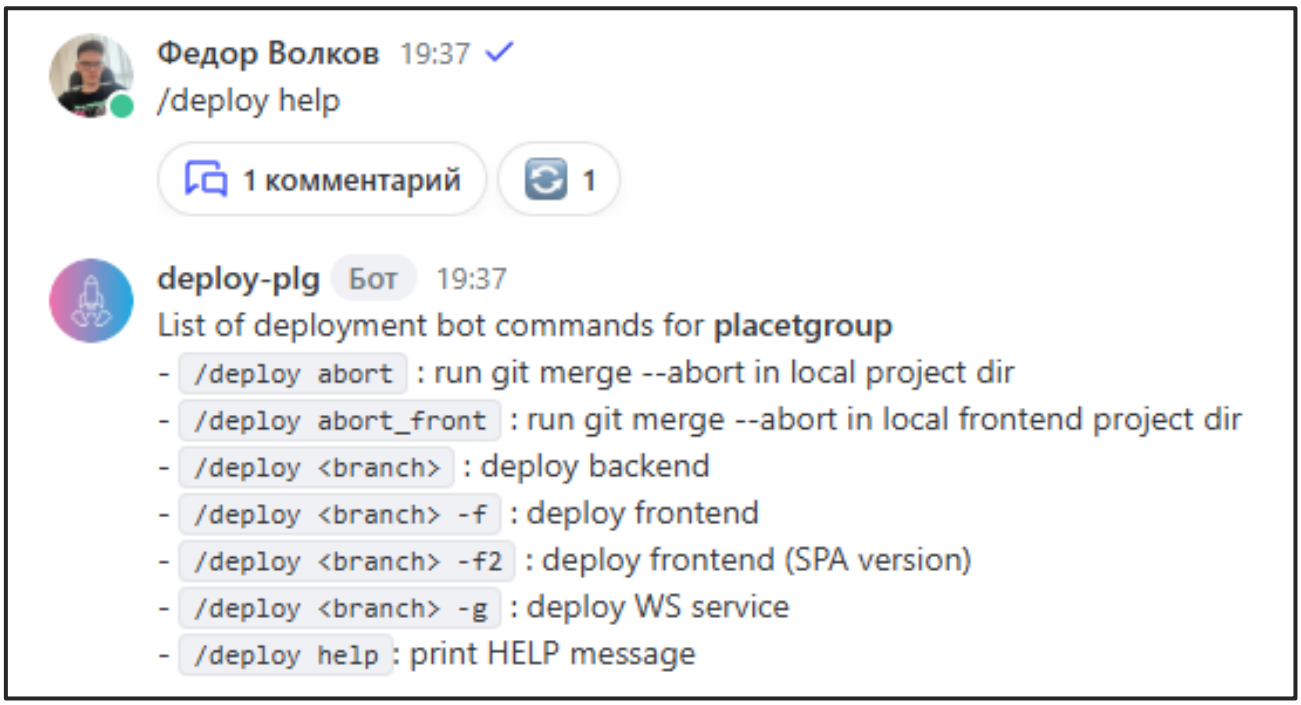

Рисунок 10 – Вывод команды HELP

Все элементы интерфейса спроектированы в соответствии с рекомендациями Slack API: стандартизированная раскладка, минимизация ручного ввода, быстрый отклик, чёткая структура сообщений.

«Стандарты оформления интерфейса включают:

– использование шрифта с высоким уровнем читаемости;

– контрастная цветовая палитра (зеленый – успех, красный – ошибка, синий – информация);

– соблюдение визуальных иерархий: основное сообщение, подпояснение, интерактивные элементы;

– единообразное расположение кнопок и повторяющихся блоков (например, блок статуса сборки);

– поддержка мультиязычности и локализация сообщений» [6].

В рамках взаимодействия реализована также система подсказок: при некорректной команде Deploy-бот предлагает список доступных команд или указывает, на каком этапе был допущен сбой. Такой подход минимизирует ошибки пользователя и повышает общую надёжность CI/CD процессов.

Таким образом, интерфейс Deploy-бота ориентирован на краткость, ясность, быструю навигацию и интуитивное взаимодействие, что особенно

критично в условиях оперативного управления развёртыванием на различных окружениях. Проектирование интерфейса и взаимодействия с пользователем базировалось на принципах разработки веб-приложений, изложенных в [11].

2.4 Компьютерно-сетевое обеспечение

2.4.1 Выбор размера сети и её структуры

Проектируемая сеть предназначена для поддержки инфраструктуры автоматизированного CI/CD, включающей Deploy-бота, TeamCity, Gitea, и команды DevOps. Сеть охватывает одно офисное здание с серверной и рабочими зонами.

«Сетевая топология реализована по принципу «звезда» с использованием управляемого коммутатора уровня L3. Для изоляции трафика применяются VLAN, разделяющие серверный, пользовательский и административный сегменты. Такая структура обеспечивает безопасность, управляемость и масштабируемость сети» [17].

В качестве основного шлюза используется маршрутизатор MikroTik, настроенный на балансировку и отказоустойчивость с двумя интернет-провайдерами. Это позволяет обеспечить бесперебойный доступ к внешним сервисам, webhook-интеграциям и удалённой работе через VPN.

2.4.2 Выбор сетевого оборудования

Для функционирования CI/CD-инфраструктуры выбрано следующее оборудование: один сервер под TeamCity (в Docker), один сервер под PostgreSQL, а также выделенный шлюз MikroTik для организации интернет-доступа и VPN.

Рабочие станции разработчиков подключены через управляемые коммутаторы уровня L2. Связь между этажами осуществляется через витую пару категории 6. Общая протяжённость линий составляет около 200 метров. Используются гигабитные коммутаторы и роутеры с возможностью управления и сегментирования трафика (VLAN).

2.4.3 Выбор конфигурации сети

Конфигурация сети реализована по иерархической модели с топологией «звезда». Центральный маршрутизатор MikroTik обеспечивает маршрутизацию и балансировку между двумя провайдерами. «Внутри сети настроены VLAN для разделения доступа между сегментами: сервера, разработчики, администрация. Визуализация топологии выполнена в графическом редакторе с отображением серверов, рабочих станций, сегментов и шлюза. Конфигурация обеспечивает отказоустойчивость и масштабируемость».

2.4.4 Выбор сетевого программного обеспечения

Для поддержки работы проектируемой CI/CD-инфраструктуры используется современное и надёжное сетевое программное обеспечение. Основу маршрутизации, балансировки нагрузки между двумя интернет-провайдерами, а также настройки VPN-туннелей составляет RouterOS, установленная на шлюзе MikroTik. Данный инструмент обеспечивает стабильное управление трафиком, резервирование каналов связи и организацию безопасного доступа к сети извне.

«Контейнеризация и управление сервисами (TeamCity, Deploy-бот, вспомогательные утилиты) реализуются с помощью Docker, что позволяет легко масштабировать систему и изолировать модули» [12].

В качестве основной системы управления базами данных используется PostgreSQL, развёрнутая в контейнере.

«Мониторинг – неотъемлемая часть DevOps-подхода, обеспечивающая надежность и производительность вашей системы. Он помогает быстро реагировать на проблемы и улучшать качество вашего приложения.» [11]

«Инструментарий DevOps является критической частью методологии, предоставляя средства для автоматизации, мониторинга и управления процессами разработки и доставки ПО.» [11]

«Основные категории инструментов DevOps включают системы контроля версий, средства автоматизации CI/CD, инструменты мониторинга и управления контейнерами» [11].

«Для обеспечения мониторинга состояния серверов, сетевого оборудования и критичных сервисов используется Zabbix. Эта система позволяет отслеживать загрузку ресурсов, доступность хостов, своевременно оповещать о сбоях и предоставляет удобную визуализацию через веб-интерфейс» [12]. Агент Zabbix установлен на всех ключевых узлах инфраструктуры.

Безопасный удалённый доступ обеспечивается за счёт настроек VPN (на базе WireGuard или OpenVPN) и стандартных инструментов администрирования по SSH. Все выбранные программные средства совместимы между собой, имеют активную поддержку, широкое распространение и подходят для эксплуатации в небольших и средних корпоративных сетях.

3 Оценка эффективности внедрения информационной системы

3.1 Общие положения

Эффективность информационной системы (ИС) отражает степень её пригодности для достижения поставленных целей в условиях конкретной предметной области. На этапе разработки и внедрения крайне важно не только создать функционально завершённую систему, но и оценить её влияние на деятельность предприятия или подразделения. Такой подход позволяет определить, насколько оправданы затраты на разработку и внедрение ИС, а также выявить возможные направления её дальнейшего совершенствования.

«Эффективность ИС можно рассматривать в двух аспектах. В более широком смысле – как способность системы оказывать положительное влияние на принятие управленческих решений и достижение целей организации. В более узком – как способность ИС удовлетворять информационные потребности пользователей с минимальными затратами ресурсов, обеспечивая необходимую полноту, достоверность и своевременность информации» [18].

«Экономическая эффективность ИС определяется её способностью снижать издержки и повышать производительность труда» [18].

На практике эффективность ИС проявляется через совокупность её качеств, таких как:

- технологичность и простота внедрения;
- удобство эксплуатации и поддержки;
- способность снижать трудозатраты и ускорять процессы;
- обеспечение надежности и безопасности данных;
- экономическая обоснованность внедрения.

«Система считается эффективной, если она обеспечивает рациональное использование ресурсов предприятия, улучшает производственные или управленческие процессы, способствует уменьшению ошибок и увеличивает производительность труда. Немаловажным является также соответствие ИС требованиям пользователей и её устойчивость к отказам при эксплуатации» [21].

К числу основных качественных характеристик ИС относятся:

1. Надежность – способность системы функционировать без сбоев в течение заданного периода времени;
2. Безопасность – защита информации от несанкционированного доступа и сохранение её целостности;
3. Достоверность – точность и правильность обработки данных;
4. «Экономичность – соответствие затрат на разработку и эксплуатацию достигнутому результату» [21].

Таким образом, оценка эффективности ИС – это неотъемлемая часть её проектирования и внедрения, позволяющая подтвердить целесообразность и пользу от использования системы в конкретной организационной среде.

3.2 Показатели эффективности

В рамках данной выпускной квалификационной работы, направленной на разработку Deploy-бота для автоматизации CI/CD процессов через TeamCity, оценка эффективности внедряемой информационной системы базируется на специфике автоматизированных задач и требований к современным DevOps-процессам.

Показатели отражают, насколько система отвечает целям и задачам предприятия в области управления разработкой и внедрением ПО.

1. «Оперативность выполнения операций – время от подачи команды до завершения деплоя» [13]. Внедрение Deploy-бота сокращает это

время с нескольких минут или часов (в зависимости от ручного согласования и запусков) до нескольких секунд.

2. «Полнота формирования результатной информации» [13] – Deploy-бот формирует сообщения о статусе выполнения (успешно/ошибка), ссылки на логи, метки сборок и версии, что обеспечивает полноту информационной поддержки принятия решений.

3. Точность обработки запросов – за счёт строгих форматов команд и централизованной валидации параметров снижается риск ошибок и повторных запусков.

Показатели технической эффективности отражают совершенство архитектуры и технологий системы.

1. «Надежность – система устойчива к сбоям: в случае ошибки происходит корректная регистрация и информирование пользователя. Автоматизация исключает человеческий фактор» [17].

2. «Масштабируемость – архитектура позволяет расширять функциональность за счёт добавления новых сценариев деплоя и поддержки новых проектов без изменения базовой логики» [17].

3. Интероперабельность – система интегрирована с Gitea (репозиторий исходного кода), TeamCity, мессенджером, что свидетельствует о высоком уровне совместимости.

Показатели эксплуатационной эффективности оценивают удобство и производительность в ежедневной работе:

1 Снижение трудоёмкости – количество ручных действий при CI/CD уменьшилось до одной команды в мессенджере. Экономия времени на одного релиза может достигать 15–30 минут.

2 Повышение производительности труда – DevOps-инженеры могут сосредоточиться на сложных задачах, а не рутине.

3 Простота и удобство интерфейса – чат-интерфейс не требует специального обучения и доступен с любых устройств.

4 «Эргономичность – взаимодействие построено по принципу «одна команда – один результат», что минимизирует когнитивную нагрузку на пользователя» [16].

Данная группа показывает финансовую обоснованность внедрения:

1 Снижение эксплуатационных затрат – уменьшение времени на релиз прямо связано с сокращением ФОТ на выполнение одних и тех же задач.

2 Экономия трудозатрат – если ежедневно выполняется 5–10 сборок вручную, то автоматизация экономит до 3–5 часов в неделю одного специалиста.

3 Срок окупаемости – при условии затрат на разработку (N руб.) и экономии в месяц (M руб.), система окупается в течение 2–3 месяцев, что соответствует высоким требованиям ROI.

4 «Коэффициент эффективности – может быть рассчитан как отношение годовой экономии к затратам на внедрение; в нашем случае он значительно превышает минимально допустимый норматив» [16] (например,> 0.33).

3.3 Расчет экономической эффективности

Одним из ключевых этапов при внедрении информационной системы является оценка её экономической эффективности. Это необходимо для обоснования целесообразности проекта и демонстрации его потенциальной пользы для организации. Расчёты производятся путём сравнения показателей базового (до автоматизации) и проектного (с внедрением разработанной ИС) вариантов. На основании анализа рассчитываются: трудоёмкость, стоимостные затраты, годовой экономический эффект, срок окупаемости и коэффициент эффективности.

Анализ базового и проектного вариантов. В существующей системе (базовом варианте) процесс CI/CD осуществляется вручную DevOps-инженером. Каждый деплой занимает в среднем 15 минут. В день выполняется около 5 деплоев, а число рабочих дней в месяце принимается равным 20. Таким образом, месячные трудозатраты составляют:

$$T\text{мес} = 15\text{мин} \times 5 \times 20 = 1500\text{мин} = 25\text{ч} \quad (1)$$

Учитывая, что в году 12 месяцев, годовая трудоёмкость:

$$T_0 = 25\text{ч} \times 12 = 300\text{ч} \quad (2)$$

Почасовая ставка DevOps-инженера, исходя из оклада 120 000 руб. в месяц и 160 рабочих часов, составляет:

$$\mathrm{R} = \frac{120000}{160} = 750\text{руб/ч} \quad (3)$$

Внедрение Deploy-бота автоматизирует около 80% операций CI/CD. Таким образом, при сохранении 20% ручных действий трудоёмкость снижается до:

$$\mathrm{T}_j = 300\text{ч} \times 0.2 = 60\text{ч} \quad (4)$$

Экономия по трудозатратам составляет:

$$\Delta\mathrm{T} = \mathrm{T}_0 - \mathrm{T}_j = 300 - 60 = 240\text{ч} \quad (5)$$

Относительные показатели эффективности по трудозатратам:

$$\mathrm{K}_m = \frac{\Delta\mathrm{T}}{\mathrm{T}_0} = \frac{240}{300} = 0.8 \quad (80\% \text{ снижение}) \quad (6)$$

$$\mathrm{I}_\mathrm{m} = \frac{\mathrm{T}_0}{\mathrm{T}_j} = \frac{300}{60} = 5$$

(7)

Эти значения указывают на значительное улучшение трудоёмкости, в 5 раз меньше времени затрачивается после внедрения системы.

Расчёт стоимостных эксплуатационных затрат. Эксплуатационные затраты по базовому варианту определяются как произведение трудоёмкости и ставки:

$$C_0 = T_0 \times R = 300 \times 750 = 225000\,\text{руб}$$

(8)

После внедрения системы:

$$Cj = Tj \times R = 60 \times 750 = 45000\,\text{руб}$$

(10)

Соответственно, годовая экономия:

$$\Delta C = C_0 - C_j = 225000 - 45000 = 180000\text{р}$$

(11)

Относительный коэффициент экономии:

$$K_\text{с} = \frac{C0}{\Delta C} = \frac{225000}{180000} = 0.8$$

(12)

Таким образом, внедрение системы позволяет сократить годовые эксплуатационные расходы на 80%.

Экономический эффект. Капитальные вложения в разработку и внедрение Deploy-бота составили:

$$Kj = 60000\,\text{руб}$$

(13)

Дополнительные ежегодные расходы на сопровождение составляют:

$$C_{\text{сопров}} = 20000\,\text{руб}$$

(14)

Базовый вариант затрат на ИС отсутствует:

$$K_0 = 0$$

(15)

Расчёт приведённых затрат по базовому варианту:

$$Z_0 = C_0 + E_n \times K_0 = 225000 + 0.15 \times 0 = 225000$$

(16)

Приведённые затраты по проектному варианту:

$$Zj = Cj + En \times Kj = 45\,000 + 0.15 \times 60\,000 = 45\,000 + 9\,000 = 54\,000$$

(17)

Годовой экономический эффект от внедрения:

$$E = Z_0 - Z_j = 225000 - 54000 = 171000\text{руб}$$

(18)

Для оценки финансовой целесообразности рассчитываются два ключевых показателя:

$$Tок = \frac{\Delta C}{\Delta K} = \frac{180000}{60000} = 0.33\ \text{года}\ (\approx 4\ \text{мес.})$$

(19)

$$Kэ = \frac{1}{T_{ок}} = 10.33 \approx 3.03$$

(20)

Нормативный коэффициент эффективности E_n принят равным 0.15. Поскольку $K_э > E_n$, можно заключить, что внедрение разработанной системы является высокоэффективным.

Проведённые расчёты подтверждают, что внедрение сервиса в CI/CD-инфраструктуру позволяет значительно оптимизировать процесс развертывания. Снижение трудозатрат на 80% и уменьшение эксплуатационных расходов на 180000 руб. в год делает проект экономически целесообразным. Срок окупаемости составляет менее 4 месяцев, а коэффициент эффективности более чем в 20 раз превышает нормативный.

Таким образом, система обеспечивает не только повышение производительности, но и ощутимую экономическую выгоду, оправдывая все затраты на её разработку и внедрение.

ЗАКЛЮЧЕНИЕ

В условиях активного роста цифровой инфраструктуры предприятий особое значение приобретает автоматизация процессов сборки и развертывания программного обеспечения. Традиционные подходы к CI/CD, основанные на ручных действиях DevOps-инженеров, приводят к повышенным трудозатратам, задержкам в релизах и повышенному риску ошибок. В связи с этим возникла необходимость разработки специализированного решения для автоматизации CI/CD-процессов.

Объектом исследования является ООО «Свик».

Предметом исследования является процесс автоматизированного развёртывания программных решений в рамках CI/CD.

Целью выпускной квалификационной работы является разработка сервиса для управления сборкой и развертыванием программного обеспечения.

Для достижения поставленной цели были решены следующие задачи:

1. Проведен анализ текущего состояния процессов развертывания и выявлены существующие недостатки.
2. Выполнен обзор современных решений в области автоматизации CI/CD.
3. Определены функциональные и нефункциональные требования к разрабатываемому программному продукту.
4. Разработана архитектура и проектные решения программного средства.
5. Реализован сервис, обеспечивающий интеграцию TeamCity с корпоративным мессенджером Slack.
6. Проведена оценка экономической эффективности внедрения разработанного решения.

Практическая значимость проекта подтверждена проведённым расчетом экономического показателя окупаемости, который свидетельствует о целесообразности и выгодности применения разработанного сервиса в условиях предприятия.

Таким образом, выполненная работа вносит вклад в повышение эффективности процессов CI/CD за счёт автоматизации ключевых этапов развертывания, снижая трудозатраты, минимизируя ошибки и обеспечивая прозрачность и управляемость процессов. Реализация сервиса способствует ускорению выпуска программных продуктов и повышению качества их сопровождения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Teamcity Jetbrains [Электронный ресурс]: статья – Режим доступа: URL: https://www.jetbrains.com/ru-ru/teamcity/– Загл. с экрана.

2. GitLab: The DevSecOps Platform [Электронный ресурс]: статья – Режим доступа к статье: URL: https://about.gitlab.com/ – Загл. с экрана.

3. Octopus Deploy [Электронный ресурс]: официальный сайт – Режим доступа: URL: https://octopus.com/ – Загл. с экрана.

4. Jenkins User Documentation [Электронный ресурс]: статья – Режим доступа: URL: https://www.jenkins.io/doc/ – Загл. с экрана.

5. Gitea: Git with a cup of tea [Электронный ресурс]: официальный сайт – Режим доступа: URL: https://gitea.com/ – Загл. с экрана.

6. Slack API Documentation [Электронный ресурс]: документация – Режим доступа: URL: https://api.slack.com/ – Загл. с экрана.

7. Go Programming Language [Электронный ресурс]: официальный сайт – Режим доступа: URL: https://go.dev/ – Загл. с экрана.

8. GoSSH [Электронный ресурс]: библиотека для работы с SSH на Go – Режим доступа: URL: https://pkg.go.dev/github.com/gliderlabs/ssh – Загл. с экрана.

9. Python Programming Language [Электронный ресурс]: официальный сайт – Режим доступа: URL: https://python.org/ – Загл. с экрана.

10. Bash Reference Manual [Электронный ресурс]: документация – Режим доступа: URL: https://www.gnu.org/software/bash/manual/bash.html – Загл. с экрана.

11. Баланов А. Н. DevOps: интеграция и автоматизация: учебное пособие для вузов / А. Н. Баланов. – Санкт-Петербург: Лань, 2024. – 240 с. – Текст: непосредственный.

12. Гифт, Ной. Python и DevOps: ключ к автоматизации Linux / Ной Гифт, Кеннеди Берман, Альфредо Деза, Григ Георгиу. – Санкт-Петербург:

Питер, 2024. – 544 с.: ил. – (Серия «Бестселлеры O'Reilly»). – ISBN 978-5-4461-2929-4.

13. Кристи,Уилсон. Грокаем Continuous Delivery / Уилсон Кристи. – Санкт-Петербург: Питер, 2024. – 400 с.: ил. – (Серия «Библиотека программиста»). – ISBN 978-5-4461-2372-8.

14. Астапчук, В. А. Корпоративные информационные системы: требования при проектировании: учебное пособие для вузов / В. А. Астапчук, П. В. Терещенко. – 3-е изд., перераб. и доп. – Москва: Издательство Юрайт, 2024. – 175 с. – (Высшее образование). – ISBN 978-5-534-16715-3. – Текст: электронный // Образовательная платформа Юрайт [сайт]. – URL: https://urait.ru/bcode/531569. – Загл. с экрана.

15. Волкова, В. Н. Теория информационных процессов и систем: учебник и практикум для вузов / В. Н. Волкова. – 2-е изд., перераб. и доп. – Москва: Издательство Юрайт, 2024. – 432 с. – (Высшее образование). – ISBN 978-5-534-05621-1. – Текст: электронный // Образовательная платформа Юрайт [сайт]. – URL: https://urait.ru/bcode/536108. – Загл. с экрана.

16. Воронова, И. В. Проектирование: учебное пособие для вузов / И. В. Воронова. – 2-е изд. – Москва: Издательство Юрайт, 2024. – 167 с. – (Высшее образование). – ISBN 978-5-534-14420-8. – Текст: электронный // Образовательная платформа Юрайт [сайт]. – URL: https://urait.ru/bcode/520046 – Загл. с экрана.

17. Грекул, В. И. Проектирование информационных систем: учебник и практикум для вузов / В. И. Грекул, Н. Л. Коровкина, Г. А. Левочкина. – 2-е изд., перераб. и доп. – Москва: Издательство Юрайт, 2025. – 404 с. – (Высшее образование). – ISBN 978-5-534-19505-7. – Текст: электронный // Образовательная платформа Юрайт [сайт]. – URL: https://urait.ru/bcode/ – Загл. с экрана.

18. Зараменских, Е. П. Управление жизненным циклом информационных систем: учебник и практикум для вузов / Е. П. Зараменских. – 2-е изд. – Москва: Издательство Юрайт, 2024. – 497 с. –

(Высшее образование). – ISBN 978-5-534-14023-1. – Текст: электронный // Образовательная платформа Юрайт [сайт]. – URL: https://www.urait.ru/bcode/536966 – Загл. с экрана

19. Полуэктова, Н. Р. Разработка веб-приложений: учебное пособие для вузов / Н. Р. Полуэктова. – 2-е изд. – Москва: Издательство Юрайт, 2024. – 204 с. – (Высшее образование). – ISBN 978-5-534-18645-1. – Текст: электронный // Образовательная платформа Юрайт [сайт]. – URL: https://www.urait.ru/bcode/545238 – Загл. с экрана

20. Илюшечкин, В. М. Основы использования и проектирования баз данных: учебник для вузов / В. М. Илюшечкин. – Москва: Издательство Юрайт, 2024. – 213 с. – (Высшее образование). – ISBN 978-5-534-03617-6. – Текст: электронный // Образовательная платформа Юрайт [сайт]. – URL: https://urait.ru/bcode/535450 (дата обращения: 09.01.2025). – Загл. с экрана.

21. Информационные системы в экономике: учебник для вузов / В. Н. Волкова, В. Н. Юрьев, С. В. Широкова, А. В. Логинова; под редакцией В. Н. Волковой, В. Н. Юрьева. – Москва: Издательство Юрайт, 2024. – 402 с. – (Высшее образование). – ISBN 978-5-9916-1358-3. – Текст: электронный // Образовательная платформа Юрайт [сайт]. – URL: https://urait.ru/bcode/536689 – Загл. с экрана.

22. Черткова, Е. А. Программная инженерия. Визуальное моделирование программных систем: учебник для вузов / Е. А. Черткова. – 3-е изд., перераб. и доп. – Москва: Издательство Юрайт, 2025. –146 с. – (Высшее образование). – ISBN 978-5-534-18197-5. – Текст: электронный // Образовательная платформа Юрайт [сайт]. – URL: https://www.urait.ru/bcode/562413 – Загл. с экрана.