**Описание процессов, обеспечивающих поддержание жизненного цикла программного обеспечения**

**«Интеллектуальная система оцифровки и анализа личности человека cvcode»**

**Оглавление**

[Аннотация 3](#_Toc184750475)

[Термины и сокращения 3](#_Toc184750476)

[Перечень сокращений 3](#_Toc184750477)

[1. Процессы жизненного цикла программного обеспечения 4](#_Toc184750478)

[1.1. Общие сведения 4](#_Toc184750479)

[1.2. Процессы внедрения программных средств 4](#_Toc184750480)

[1.2.3. Процессы проектирования программных средств 8](#_Toc184750481)

[1.2.4. Процесс конструирования программных средств 10](#_Toc184750482)

[1.2.5. Процесс комплексирования программных средств 11](#_Toc184750483)

[1.2.6. Процесс квалификационного тестирования программных средств 13](#_Toc184750484)

[1.3. Процесс поддержки программных средств 15](#_Toc184750485)

[1.3.1. Процесс управления документацией программных средств 15](#_Toc184750486)

[1.3.2. Процесс управления конфигурацией программных средств 17](#_Toc184750487)

[1.3.3. Процесс обеспечения гарантий качества программных средств 19](#_Toc184750488)

[1.3.4. Процесс верификации программных средств 21](#_Toc184750489)

[1.3.5. Процесс валидации программных средств 23](#_Toc184750490)

[1.3.6. Процесс ревизии программных средств 26](#_Toc184750491)

[1.3.7. Процесс аудита программных средств 28](#_Toc184750492)

[1.3.8. Процесс решения проблем в программных средствах 30](#_Toc184750493)

[2. Порядок технической поддержки программного обеспечения 33](#_Toc184750494)

[2.1. Общие сведения 33](#_Toc184750495)

[2.2. Техническая поддержка первого уровня 35](#_Toc184750496)

[2.3. Техническая поддержка второго уровня 37](#_Toc184750497)

[2.4. Принципы организации технической поддержки 39](#_Toc184750498)

[3. Устранение неисправностей, выявленных в ходе эксплуатации программного обеспечения 41](#_Toc184750499)

[4. Совершенствование программного обеспечения 44](#_Toc184750500)

[5. Информация о персонале 46](#_Toc184750501)

[Термины и сокращения 3](#_Toc181726866)

[Перечень сокращений 3](#_Toc181726867)

[1. Процессы жизненного цикла программного обеспечения 3](#_Toc181726868)

[1.1. Общие сведения 3](#_Toc181726869)

[1.2. Процессы внедрения программных средств 3](#_Toc181726870)

[1.2.1. Основной процесс внедрения 3](#_Toc181726871)

[1.2.2. Процесс анализа требований к программным средствам 3](#_Toc181726872)

[1.2.3. Процессы проектирования программных средств 3](#_Toc181726873)

[1.2.4. Процесс конструирования программных средств 3](#_Toc181726874)

[1.2.5. Процесс комплексирования программных средств 3](#_Toc181726875)

[1.2.6. Процесс квалификационного тестирования программных средств 3](#_Toc181726876)

[1.3. Процесс поддержки программных средств 3](#_Toc181726877)

[1.3.1. Процесс управления документацией программных средств 3](#_Toc181726878)

[1.3.2. Процесс управления конфигурацией программных средств 3](#_Toc181726879)

[1.3.3. Процесс обеспечения гарантий качества программных средств 3](#_Toc181726880)

[1.3.4. Процесс верификации программных средств 3](#_Toc181726881)

[1.3.5. Процесс валидации программных средств 3](#_Toc181726882)

[1.3.6. Процесс ревизии программных средств 3](#_Toc181726883)

[1.3.7. Процесс аудита программных средств 4](#_Toc181726884)

[1.3.8. Процесс решения проблем в программных средствах 4](#_Toc181726885)

[2. Порядок технической поддержки программного обеспечения 4](#_Toc181726886)

[2.1. Общие сведения 4](#_Toc181726887)

[2.2. Техническая поддержка первого уровня 4](#_Toc181726888)

[2.3. Техническая поддержка второго уровня 4](#_Toc181726889)

[2.4. Принципы организации технической поддержки 4](#_Toc181726890)

[3. Устранение неисправностей, выявленных в ходе эксплуатации программного обеспечения 4](#_Toc181726891)

[4. Совершенствование программного обеспечения 4](#_Toc181726892)

[5. Информация о персонале 4](#_Toc181726893)

# **Аннотация**

Данный документ содержит:

* описание процессов, обеспечивающих поддержание жизненного цикла программного обеспечения;
* устранение неисправностей, выявленных в ходе эксплуатации программного обеспечения;
* информацию о персонале, необходимом для обеспечения такой поддержки.

**Интеллектуальная система оцифровки и анализа личности человека cvcode (далее – CVCODE) — интеллектуальная система, которая использует искусственный интеллект для распознавания черт характера и уровня интеллекта пользователей по данным активности в социальной сети ВК. Платформа интегрирована с ВК (VK.com), и после регистрации анализирует страницу пользователя с помощью двух натренированных нейронных сетей, предоставляя информацию о развитости интеллекта, и о характере по модели OCEAN. На основе тестов, таких как: трудовая мотивация, умственные способности, тип командной роли и социальный интеллект – сервис позволяет полноценно и всесторонне оценить личность пользователя. Сервис включает тренажеры для развития памяти, внимания и социального интеллекта, а также предоставляет индивидуальные планы развития для пользователей. CVCODE применяется пользователями для получения первичных данных о себе, своих сильных и слабых сторонах. Интерфейс помогает отрефлексировать собственные качества интеллекта и личности. Платформа бесплатна для всех пользователей, а доступ к результатам тестирования предоставляется только с личного согласия пользователей.**

Данный документ предназначен для администраторов CVCODE.

# Термины и сокращения

|  |  |
| --- | --- |
| Термин | Расшифровка |
| Программное обеспечение, Платформа | CVCODE, https://cvcode.ru |
| Разработчик | ООО «Социальный код» |
| Клиент | физическое лицо, акцептировавшее лицензионный договор Разработчика и получившее доступ к Платформе |

# Перечень сокращений

|  |  |
| --- | --- |
| Сокращение | Расшифровка |
| ОС | Операционная система |
| ИИ | Искусственный интеллект |

# **1. Процессы жизненного цикла программного обеспечения**

## **Общие сведения**

CVCODE — интеллектуальная платформа, использующая искусственный интеллект для анализа личности и интеллекта на основе данных из социальной сети ВКонтакте. Жизненный цикл программного обеспечения (ЖЦПО) CVCODE организован в соответствии с требованиями ГОСТ Р ИСО/МЭК 12207-2010 и включает процессы разработки, внедрения, сопровождения и совершенствования системы.

Основные цели процессов ЖЦПО:

* Обеспечение качества и надежности программного обеспечения.
* Оптимизация работы платформы для пользователей.
* Постоянное обновление функциональности для соответствия современным требованиям.

## **1.2. Процессы внедрения программных средств**

#### 1.2.1. Основной процесс внедрения

Основной процесс внедрения CVCODE организован таким образом, чтобы гарантировать успешное развертывание системы, ее адаптацию к условиям эксплуатации и обучение пользователей.

#### **Цели процесса:**

1. Подготовить инфраструктуру для работы платформы.
2. Настроить и протестировать систему в реальной среде эксплуатации.
3. Обучить пользователей работе с системой.
4. Обеспечить эксплуатацию платформы в установленный срок.

#### **Основные этапы:**

1. **Инициация процесса внедрения:**
	* Создание проектной группы из представителей заказчика и команды CVCODE.
	* Проведение анализа требований заказчика.
	* Определение ключевых этапов внедрения, сроков и ответственных лиц.
2. **Подготовительный этап:**
	* Оценка технической инфраструктуры заказчика:
		+ Серверные мощности.
		+ Сети передачи данных.
		+ Рабочие места пользователей.
	* Установка необходимых программных средств (операционных систем, систем управления базами данных, вспомогательных утилит).
3. **Установка программного обеспечения:**
	* Развертывание серверной версии CVCODE для юридических лиц или подключение к облачной версии для физических лиц.
	* Настройка API для взаимодействия с инфраструктурой заказчика.
	* Интеграция с внешними системами (например, корпоративными системами или кадровыми базами данных).
4. **Настройка системы:**
	* Адаптация платформы под потребности заказчика (например, разработка кастомизированных модулей или подключение специфической базы знаний).
	* Настройка пользовательских ролей и уровней доступа.
5. **Тестирование и проверка:**
	* Функциональное тестирование платформы:
		+ Проверка корректности работы основных модулей (анализ данных, тесты, тренажеры).
	* Нагрузочное тестирование для оценки производительности.
	* Исправление обнаруженных дефектов.
6. **Обучение пользователей:**
	* Проведение обучающих сессий для сотрудников заказчика.
	* Подготовка руководств и методических материалов.
	* Онлайн-поддержка в формате вебинаров или FAQ.
7. **Промышленный запуск:**
	* Перевод системы в рабочий режим.
	* Мониторинг работы платформы в первые 2-4 недели эксплуатации.
	* Оперативное устранение возникающих проблем.

#### **Результаты процесса:**

* Успешное развертывание и настройка платформы.
* Обученные пользователи, способные эффективно работать с системой.
* Полностью функционирующая платформа, интегрированная с инфраструктурой заказчика.
* Документация, включающая описание процесса внедрения, конфигурацию системы и инструкции для пользователей.

#### **Пример для CVCODE:**

Для образовательного учреждения, использующего CVCODE для оценки студентов, процесс внедрения включал:

* Настройку сервера на базе Ubuntu 22.04.
* Интеграцию платформы с внутренними базами данных учащихся.
* Разработку модулей для анализа soft skills и формирования рекомендаций для студентов.
* Обучение преподавателей работе с платформой и предоставление методических материалов.

Такой подход обеспечил успешную эксплуатацию системы и высокую степень удовлетворенности пользователей.

#### 1.2.2. Процесс анализа требований к программным средствам

Процесс анализа требований к программным средствам CVCODE направлен на выявление, документирование и согласование требований, необходимых для разработки и успешного внедрения системы.

#### **Цели процесса:**

1. Определить функциональные и нефункциональные требования к платформе.
2. Обеспечить прослеживаемость требований на всех этапах разработки.
3. Установить приоритеты и согласовать их с заинтересованными сторонами.

#### **Основные этапы:**

1. **Сбор требований:**
	* Проведение интервью с клиентами и пользователями.
	* Анализ текущих бизнес-процессов заказчика.
	* Исследование технической инфраструктуры и специфики организации.
2. **Классификация требований:**
	* **Функциональные требования**:
		+ Возможность анализа данных пользователей ВК.
		+ Проведение тестов и тренажеров.
		+ Формирование индивидуальных планов развития.
	* **Нефункциональные требования**:
		+ Высокая производительность (поддержка нагрузки до 1 млн пользователей в год).
		+ Интеграция с API сторонних систем.
		+ Соответствие стандартам безопасности данных.
3. **Документирование требований:**
	* Составление спецификации требований (Software Requirements Specification, SRS).
	* Использование стандартных шаблонов для структурирования данных:
		+ Описание сценариев использования (use cases).
		+ Диаграммы процессов (например, BPMN).
4. **Анализ требований:**
	* Проверка требований на полноту, непротиворечивость и реализуемость.
	* Оценка влияния требований на архитектуру системы.
	* Учет рисков и ограничений.
5. **Приоритизация требований:**
	* Распределение по категориям:
		+ **Критические**: обязательные для выполнения базовых функций платформы.
		+ **Важные**: улучшающие пользовательский опыт.
		+ **Желательные**: повышающие ценность продукта.
	* Совместная работа с заказчиком для согласования приоритетов.
6. **Согласование и утверждение требований:**
	* Проведение презентаций для заинтересованных сторон.
	* Получение обратной связи и внесение правок.
	* Утверждение спецификации и формирование базовой линии требований.
7. **Обновление и управление требованиями:**
	* Регулярный пересмотр требований в зависимости от изменяющихся условий.
	* Внесение изменений с учетом влияния на стоимость, сроки и технические аспекты.

#### **Результаты процесса:**

* Полный и согласованный документ требований.
* Описание сценариев использования платформы.
* Базовая линия требований для последующих этапов разработки.

#### **Пример для CVCODE:**

Во время внедрения CVCODE для крупной корпорации был выполнен следующий процесс анализа требований:

1. Проведены интервью с отделами HR и IT.
2. Выявлены функциональные требования:
	* Интеграция с корпоративной системой учета кадров.
	* Формирование команд на основе анализа soft skills.
3. Определены нефункциональные требования:
	* Поддержка одновременного доступа 500 пользователей.
	* Защищенность данных в соответствии с ФЗ-152.
4. Утвержден документ, включающий 120 требований, классифицированных по приоритетам.

Этот подход позволил обеспечить соответствие платформы бизнес-целям заказчика и добиться успешного запуска.

### **1.2.3. Процессы проектирования программных средств**

Процессы проектирования программных средств CVCODE направлены на создание архитектуры системы, определение её компонентов и взаимодействий, а также подготовку детального проекта для реализации требований.

Цели процесса:

1. Разработать архитектуру системы, обеспечивающую выполнение всех требований.
2. Обеспечить масштабируемость, надежность и безопасность платформы.
3. Согласовать проект с ключевыми заинтересованными сторонами.

Основные этапы:

1. Архитектурное проектирование:
	* Определение структуры системы:
		+ Разделение платформы на модули (например, модуль анализа данных, модуль тестирования, дашборд).
		+ Разработка общей архитектуры системы с учетом распределенной среды.
	* Проектирование интерфейсов:
		+ Определение API для взаимодействия с внешними и внутренними системами.
		+ Создание схем обмена данными между модулями.
	* Выбор технологий:
		+ Базы данных: PostgreSQL для хранения пользовательских данных.
		+ Backend: Python/Django для серверной части.
		+ Frontend: React.js для пользовательского интерфейса.
	* Создание архитектурной документации:
		+ Диаграммы компонентов.
		+ Схемы взаимодействий (UML, Sequence diagrams).
2. Детальное проектирование:
	* Проектирование модулей:
		+ Описание функциональности каждого компонента.
		+ Определение взаимосвязей между модулями.
	* Проектирование базы данных:
		+ Определение структуры хранения данных (например, информация о пользователях, результатах тестов, логах активности).
		+ Нормализация данных для оптимизации работы.
	* Документирование:
		+ Подготовка технических спецификаций для разработчиков.
3. Обеспечение совместимости и прослеживаемости:
	* Установление связей между требованиями и проектными решениями.
	* Проверка соответствия архитектуры заявленным требованиям.
4. Валидация проектных решений:
	* Проведение ревизий архитектуры с участием заинтересованных сторон.
	* Моделирование ключевых сценариев использования платформы.

Результаты процесса:

* Документированная архитектура системы (архитектурный проект).
* Техническая спецификация для разработки модулей.
* Утвержденный проект базы данных.
* Сценарии взаимодействия компонентов.

Пример для CVCODE:

При проектировании CVCODE для образовательного учреждения была реализована следующая архитектура:

1. Основные модули:
	* Модуль анализа активности пользователей ВК.
	* Модуль тестирования (психологические тесты, тесты интеллекта).
	* Дашборд для представления результатов анализа.
	* API для интеграции с кадровыми системами.
2. Архитектурные решения:
	* Распределенная архитектура с использованием микросервисов.
	* База данных PostgreSQL с высокой степенью нормализации.
3. Совместимость:
	* Интеграция с внешними системами через REST API.
	* Соответствие требованиям безопасности (использование HTTPS и шифрования данных).

Проектирование завершилось созданием полной технической документации, которая была использована на этапе разработки. Это обеспечило ясность задач для всех членов команды и успешное выполнение проекта.

### **1.2.4. Процесс конструирования программных средств**

Процесс конструирования программных средств CVCODE включает в себя разработку исходного кода, его тестирование, сборку и подготовку программного обеспечения к интеграции с другими модулями системы.

#### **Цели процесса:**

1. Реализовать проектные решения, заложенные на этапе проектирования.
2. Обеспечить качество и функциональность программного кода.
3. Подготовить ПО к тестированию и интеграции.

#### **Основные этапы:**

1. **Разработка программного обеспечения:**
	* **Создание исходного кода:**
		+ Разработка функциональных модулей в соответствии с утвержденным проектом.
		+ Реализация алгоритмов анализа данных, выполнения тестов и обработки результатов.
		+ Создание интерфейсов API для взаимодействия с внешними системами.
	* **Применение стандартов кодирования:**
		+ Использование PEP 8 (для Python).
		+ Применение инструментов статического анализа кода (например, Flake8, SonarQube).
2. **Юнит-тестирование:**
	* Написание и выполнение модульных тестов для проверки каждого компонента.
	* Использование фреймворков для тестирования (например, PyTest, JUnit).
	* Регистрация и исправление ошибок, выявленных на этапе тестирования.
3. **Сборка системы:**
	* Конфигурация инструментов автоматической сборки (например, Jenkins, GitLab CI).
	* Подготовка контейнеров для развертывания (Docker):
		+ Создание и настройка Docker Compose для запуска микросервисов.
		+ Тестирование контейнеров в локальной среде.
	* Сборка бинарных или исполняемых файлов, если это требуется.
4. **Документирование кода:**
	* Добавление комментариев и документации в код (например, docstrings в Python).
	* Создание README-файлов для модулей.
	* Обновление технической документации для разработчиков.
5. **Обеспечение совместимости:**
	* Проверка совместимости кода с ранее разработанными модулями.
	* Подготовка к интеграционному тестированию.
6. **Внедрение системы контроля версий:**
	* Разделение кода по веткам разработки (например, main, feature, bugfix).
	* Управление версиями с использованием Git.

#### **Результаты процесса:**

* Исходный код, соответствующий проектной документации и стандартам качества.
* Тестируемая и рабочая версия системы.
* Документированные исходные файлы и инструкции по сборке.

#### **Пример для CVCODE:**

В процессе конструирования платформы CVCODE для крупной компании были выполнены следующие шаги:

1. Разработка модуля анализа данных пользователей ВК:
	* Использование Python и библиотеки Scikit-learn для обработки данных.
	* Реализация алгоритмов классификации по модели OCEAN.
2. Юнит-тестирование:
	* Проверка правильности вычислений в модуле анализа данных.
	* Тестирование API запросов на корректность ответов.
3. Контейнеризация:
	* Настройка Docker Compose для запуска модулей API, базы данных PostgreSQL и брокера сообщений Redis.
4. Подготовка документации:
	* Описание интерфейсов API для разработчиков.
	* Добавление инструкций по локальному развертыванию системы.

Этот процесс обеспечил выпуск стабильной версии платформы, готовой к дальнейшему тестированию и интеграции.

### **1.2.5. Процесс комплексирования программных средств**

Процесс комплексирования программных средств CVCODE заключается в интеграции всех компонентов системы, проверке их совместной работы и подготовке системы к квалификационному тестированию.

#### **Цели процесса:**

1. Интегрировать разработанные модули в единую систему.
2. Обеспечить корректное взаимодействие компонентов.
3. Подготовить систему к эксплуатационному тестированию.

#### **Основные этапы:**

1. **Подготовка к комплексированию:**
	* Разработка стратегии комплексирования:
		+ Определение порядка интеграции модулей (например, сначала API, затем клиентская часть).
		+ Определение критических точек взаимодействия (например, взаимодействие модуля анализа данных с базой данных).
	* Настройка инфраструктуры:
		+ Развёртывание тестовой среды на базе контейнеров (Docker Compose).
		+ Конфигурация виртуальных сетей для связи между модулями.
2. **Интеграция компонентов:**
	* Подключение модулей один за другим в соответствии с утверждённым порядком.
	* Настройка и тестирование взаимодействия между модулями:
		+ Обмен данными между API и модулем анализа.
		+ Подключение базы данных PostgreSQL к сервисам API.
	* Проверка взаимодействия пользовательского интерфейса с серверной частью.
3. **Тестирование интеграции:**
	* **Функциональное тестирование:**
		+ Проверка корректности выполнения основных сценариев (например, регистрация пользователя, выполнение теста, получение результатов).
	* **Нагрузочное тестирование:**
		+ Оценка производительности системы при подключении большого числа пользователей.
	* **Регрессионное тестирование:**
		+ Проверка, что ранее исправленные ошибки не возобновились после интеграции новых компонентов.
	* Регистрация и устранение обнаруженных дефектов.
4. **Документирование результатов:**
	* Создание отчётов о тестировании и интеграции.
	* Обновление технической документации для разработчиков и администраторов.
5. **Подготовка к эксплуатации:**
	* Настройка автоматизированного развертывания системы (например, через Ansible или Terraform).
	* Финальная проверка соответствия системы требованиям.

#### **Результаты процесса:**

* Интегрированная система, готовая к квалификационному тестированию.
* Документация, описывающая порядок интеграции и конфигурации.
* Полный отчёт о результатах тестирования взаимодействий.

#### **Пример для CVCODE:**

В ходе комплексирования платформы CVCODE были выполнены следующие задачи:

1. **Интеграция модулей:**
	* Модуль анализа данных пользователей был подключён к API и базе данных.
	* Модуль тренажеров интегрирован с пользовательским интерфейсом для отображения результатов.
2. **Тестирование:**
	* Проведено нагрузочное тестирование на 10 000 одновременных пользователей.
	* Проверено корректное взаимодействие между API и базой данных PostgreSQL при выполнении сложных запросов.
3. **Документирование:**
	* Описаны сценарии взаимодействий между компонентами.
	* Подготовлена инструкция для развёртывания комплексированной версии системы.

Этот процесс обеспечил готовность CVCODE к переходу на этап квалификационного тестирования и развертывания у пользователей.

### **1.2.6. Процесс квалификационного тестирования программных средств**

Квалификационное тестирование программных средств CVCODE направлено на проверку соответствия интегрированной системы заявленным требованиям, критериям качества и готовности к эксплуатации.

#### **Цели процесса:**

1. Подтвердить соответствие платформы требованиям спецификации.
2. Обеспечить высокую производительность и стабильность системы.
3. Подготовить программное обеспечение к эксплуатации без критических ошибок.

#### **Основные этапы:**

1. **Планирование квалификационного тестирования:**
	* Разработка плана тестирования:
		+ Определение целей и задач тестирования.
		+ Составление списка тестовых сценариев, охватывающих все основные функции системы.
		+ Определение критериев успеха для каждого теста.
	* Подготовка тестовой среды:
		+ Настройка тестовой инфраструктуры (серверы, базы данных, сети).
		+ Загрузка тестовых данных.
2. **Проведение тестирования:**
	* **Функциональное тестирование:**
		+ Проверка выполнения всех ключевых функций (например, анализ данных, выполнение тестов, предоставление рекомендаций).
	* **Нагрузочное тестирование:**
		+ Проверка устойчивости системы при пиковой нагрузке.
		+ Оценка времени отклика и производительности при различных уровнях нагрузки.
	* **Интеграционное тестирование:**
		+ Проверка взаимодействия всех модулей (например, API, базы данных, пользовательского интерфейса).
	* **Регрессионное тестирование:**
		+ Проверка, что ранее исправленные ошибки не появились снова после внедрения изменений.
	* **Безопасность:**
		+ Тестирование защиты данных пользователей в соответствии с требованиями ФЗ-152 (например, шифрование, авторизация, защита от SQL-инъекций).
3. **Регистрация результатов:**
	* Фиксация результатов каждого теста:
		+ Список успешно выполненных тестов.
		+ Выявленные дефекты и их классификация по степени критичности.
	* Документирование проблем и передача их на исправление.
4. **Повторное тестирование:**
	* Проверка устранённых дефектов.
	* Проведение тестов повторно для подтверждения стабильности системы.
5. **Составление итогового отчёта:**
	* Описание результатов тестирования, включая выводы о готовности платформы.
	* Список нерешённых проблем (при наличии) и их влияние на систему.

#### **Результаты процесса:**

* Полный отчёт о квалификационном тестировании, включающий:
	+ Перечень протестированных функций.
	+ Данные о производительности и стабильности системы.
	+ Выявленные и исправленные дефекты.
* Готовая к эксплуатации версия платформы.

#### **Пример для CVCODE:**

В процессе квалификационного тестирования платформы CVCODE были проведены:

1. **Функциональные тесты:**
	* Проверка точности анализа данных по модели OCEAN.
	* Тестирование функций тренажёров и генерации рекомендаций.
2. **Нагрузочные тесты:**
	* Оценка работы системы при 50 000 одновременных запросах.
	* Проверка времени отклика при нагрузке в 90% от максимальной.
3. **Безопасность:**
	* Проведено тестирование шифрования персональных данных.
	* Проверка защиты от внешних атак (SQL-инъекции, XSS).
4. **Итоговый отчёт:**
	* Успешно завершено 95% тестов.
	* Оставшиеся 5% касались малокритичных функций, которые были доработаны перед промышленным запуском.

Этот процесс подтвердил готовность системы CVCODE к эксплуатации, обеспечив её соответствие всем заявленным требованиям.

## **1.3. Процесс поддержки программных средств**

### **1.3.1. Процесс управления документацией программных средств**

Управление документацией программных средств CVCODE включает разработку, обновление, хранение и предоставление доступа к документам, необходимым для всех этапов жизненного цикла системы. Это обеспечивает точное описание функциональности, процессов и процедур работы платформы.

#### **Цели процесса:**

1. Гарантировать доступность актуальной документации для пользователей и разработчиков.
2. Упростить обучение и сопровождение системы.
3. Обеспечить соответствие документации нормативным требованиям.

#### **Основные этапы:**

1. **Планирование и классификация документации:**
	* Определение перечня необходимых документов:
		+ Техническая документация (описание архитектуры, спецификация API, требования).
		+ Пользовательская документация (руководства для пользователей, инструкции по установке и эксплуатации).
		+ Эксплуатационная документация (описание процессов обслуживания, обновлений и восстановления).
	* Разработка структуры и шаблонов для каждого типа документации.
2. **Разработка документации:**
	* Создание новых документов на основе требований и проектных решений.
	* Описание ключевых функций платформы:
		+ Процесс анализа данных.
		+ Работа с дашбордом.
		+ Проведение тестов и использование тренажеров.
	* Описание процедур:
		+ Установка системы (локальная и облачная версии).
		+ Подключение API для юридических лиц.
3. **Обновление и актуализация документации:**
	* Регулярное обновление документов при внесении изменений в платформу.
	* Поддержка версионности документации:
		+ Введение номеров версий.
		+ Сохранение изменений в системе контроля версий (например, Git).
4. **Распределение документации:**
	* Хранение документации в централизованной системе (например, Confluence или SharePoint).
	* Предоставление пользователям и разработчикам доступа к актуальной версии документов.
	* Размещение пользовательской документации на официальном сайте <https://cvcode.ru>.
5. **Проверка и валидация документации:**
	* Регулярный аудит документов на соответствие текущему состоянию системы.
	* Сбор обратной связи от пользователей и разработчиков для улучшения содержания.
6. **Управление архивом документации:**
	* Сохранение архивных версий документации для возможного восстановления данных.
	* Организация доступа к устаревшим версиям по запросу.

#### **Результаты процесса:**

* Актуальная и структурированная документация, доступная для всех заинтересованных сторон.
* Обеспечение простоты в обучении пользователей и разработчиков.
* Соответствие документации требованиям ГОСТ и внутренних стандартов компании.

#### **Пример для CVCODE:**

1. **Пользовательская документация:**
	* Разработано руководство пользователя с подробным описанием работы с дашбордом и тестами.
	* Инструкция по подключению CVCODE через API для корпоративных клиентов.
2. **Техническая документация:**
	* Описание архитектуры системы с диаграммами взаимодействий.
	* Спецификация REST API с примерами запросов и ответов.
3. **Распределение:**
	* Руководства и инструкции размещены в открытом доступе на https://cvcode.ru/help.
	* Техническая документация доступна в системе управления знаниями компании для разработчиков.

Процесс управления документацией помогает поддерживать прозрачность работы CVCODE, упрощает интеграцию для новых клиентов и обеспечивает соответствие нормативным стандартам.

### **1.3.2. Процесс управления конфигурацией программных средств**

Управление конфигурацией программных средств CVCODE включает идентификацию, контроль, отслеживание изменений и учет всех компонентов системы на протяжении её жизненного цикла. Этот процесс позволяет обеспечивать согласованность версий, стабильность работы и упрощает внесение обновлений.

Цели процесса:

1. Гарантировать целостность и согласованность конфигурации системы.
2. Обеспечить контроль изменений для всех компонентов ПО.
3. Сохранить полную историю версий и изменений.

Основные этапы:

1. Идентификация конфигурации:
	* Определение составных частей системы:
		+ Модули платформы (например, модуль анализа данных, API, база данных).
		+ Компоненты, используемые в инфраструктуре (например, серверы, контейнеры Docker).
	* Назначение уникальных идентификаторов для каждого элемента конфигурации.
	* Создание базовой линии конфигурации (baseline), включающей текущую стабильную версию всех компонентов.
2. Контроль изменений:
	* Внедрение системы управления версиями (например, Git):
		+ Использование веток для разработки, тестирования и выпуска (например, main, develop, feature).
		+ Регистрация изменений с помощью комментариев к коммитам.
	* Утверждение изменений:
		+ Рассмотрение и одобрение изменений через pull requests.
		+ Оценка влияния изменений на систему.
3. Отслеживание состояния конфигурации:
	* Поддержка актуальной информации о статусе всех компонентов.
	* Использование инструментов для управления конфигурацией (например, Ansible, Terraform):
		+ Автоматизация настройки серверов и развертывания компонентов.
		+ Управление зависимостями между компонентами.
4. Ведение документации конфигурации:
	* Регистрация изменений в конфигурации в соответствующей документации.
	* Обновление инструкций по развертыванию и настройке системы.
5. Аудит и проверка конфигурации:
	* Регулярная проверка системы на соответствие утверждённой конфигурации.
	* Сравнение текущего состояния системы с базовой линией.
	* Исправление несоответствий (если необходимо).
6. Архивация устаревших конфигураций:
	* Сохранение предыдущих версий конфигураций в архиве для возможного восстановления.
	* Учет зависимостей между версиями компонентов.

Результаты процесса:

* Чётко определённая и зафиксированная конфигурация системы.
* Упрощённое управление изменениями.
* Снижение рисков, связанных с несовместимостью версий.

Пример для CVCODE:

1. Идентификация:
	* Назначены версии для основных компонентов:
		+ Модуль анализа данных — v1.5.0.
		+ База данных PostgreSQL — v12.3.
		+ API — v2.0.1.
2. Контроль изменений:
	* Внесение изменений через Git с использованием веток для новых функций (feature branches).
	* Проведение код-ревью перед слиянием изменений в основную ветку.
3. Автоматизация:
	* Настроен Ansible для автоматического развертывания обновлений на тестовых серверах.
4. Документирование:
	* Обновлены инструкции по настройке контейнеров Docker для локальной разработки.

Процесс управления конфигурацией позволяет поддерживать стабильность и упрощает работу с системой, обеспечивая согласованность компонентов и их готовность к обновлениям.

### **1.3.3. Процесс обеспечения гарантий качества программных средств**

Процесс обеспечения гарантий качества программных средств (Quality Assurance, QA) CVCODE направлен на достижение высокого уровня соответствия системы заявленным требованиям, стандартам разработки и ожиданиям пользователей. Он включает регулярный мониторинг, тестирование и аудит процессов разработки и эксплуатации.

Цели процесса:

1. Обеспечить соответствие системы требованиям спецификации.
2. Минимизировать количество ошибок на всех этапах жизненного цикла.
3. Снизить риски, связанные с эксплуатацией системы.

Основные этапы:

1. Планирование обеспечения качества:
	* Разработка плана QA, включающего:
		+ Методы и подходы к проверке качества.
		+ Ключевые метрики (например, количество дефектов, время на исправление, уровень покрытия тестами).
		+ Ответственных лиц на каждом этапе.
	* Определение стандартов и регламентов:
		+ Применение международных стандартов (например, ISO 25010).
		+ Внедрение лучших практик разработки и тестирования.
2. Контроль процесса разработки:
	* Проведение код-ревью:
		+ Проверка соответствия стандартам кодирования.
		+ Выявление потенциальных ошибок на раннем этапе.
	* Инструменты статического анализа кода (например, SonarQube, Flake8).
	* Управление изменениями:
		+ Утверждение изменений через систему контроля версий (например, pull requests в Git).
3. Тестирование программного обеспечения:
	* Юнит-тестирование: проверка отдельных компонентов системы.
	* Интеграционное тестирование: оценка взаимодействия между модулями.
	* Системное тестирование: проверка системы в целом.
	* Регрессионное тестирование: предотвращение возврата ранее исправленных дефектов.
	* Нагрузочное тестирование: проверка производительности системы.
4. Мониторинг и аудит качества:
	* Регулярный аудит процессов разработки и тестирования.
	* Анализ дефектов, выявленных на этапах тестирования и эксплуатации.
	* Сбор и анализ обратной связи от пользователей.
5. Управление дефектами:
	* Регистрация выявленных проблем в системе трекинга задач (например, HULY).
	* Приоритизация дефектов:
		+ Критические (требующие немедленного исправления).
		+ Средние и низкие (планируемые для исправления в следующих релизах).
	* Контроль исправления и повторное тестирование.
6. Отчётность и документирование:
	* Подготовка отчётов о выполненных тестах, выявленных дефектах и их исправлениях.
	* Документирование ключевых метрик (например, среднее время на исправление ошибок, количество релизов без дефектов).

Результаты процесса:

* Программное обеспечение, соответствующее заявленным требованиям и ожиданиям пользователей.
* Стабильность работы платформы в эксплуатационных условиях.
* Снижение рисков возникновения критических ошибок.

Пример для CVCODE:

1. Планирование качества:
	* Разработан план QA с использованием метрик: уровень покрытия тестами — 90%, среднее время на исправление ошибок — 2 дня.
2. Контроль разработки:
	* Проведён анализ кода с использованием SonarQube, выявлены и устранены 15 потенциальных уязвимостей.
3. Тестирование:
	* Успешно выполнено нагрузочное тестирование на 10 000 одновременных пользователей.
	* Проведено регрессионное тестирование после добавления новых функций.
4. Управление дефектами:
	* Зарегистрировано 12 дефектов в HULY, из которых 10 исправлены до следующего релиза.
5. Отчётность:
	* Подготовлен отчёт о качестве релиза, где указано, что все критические дефекты устранены.

Процесс обеспечения качества позволяет платформе CVCODE оставаться надёжным и высокопроизводительным инструментом, отвечающим запросам пользователей и стандартам индустрии.

### **1.3.4. Процесс верификации программных средств**

Процесс верификации программных средств CVCODE направлен на проверку соответствия разработанного программного обеспечения требованиям спецификации и проектной документации. Верификация гарантирует правильность реализации функций и подтверждает, что все элементы системы работают в соответствии с проектными решениями.

Цели процесса:

1. Подтвердить соответствие программного обеспечения технической спецификации.
2. Обнаружить и устранить дефекты на этапе разработки.
3. Обеспечить корректность функционирования каждого компонента и их взаимодействия.

Основные этапы:

1. Планирование верификации:
	* Составление плана верификации, включающего:
		+ Перечень элементов, подлежащих проверке.
		+ Методы и инструменты верификации.
		+ Критерии успеха для каждого этапа.
	* Определение ответственных за выполнение задач верификации.
2. Проведение верификации:
	* Анализ проектной документации:
		+ Проверка соответствия архитектурных и технических решений требованиям спецификации.
		+ Анализ диаграмм, моделей данных и API-документации.
	* Тестирование компонентов:
		+ Юнит-тестирование:
			- Проверка корректности работы отдельных модулей.
			- Оценка соответствия функций их описанию в проектной документации.
		+ Интеграционное тестирование:
			- Проверка взаимодействия модулей между собой.
			- Выявление несоответствий в обмене данными.
	* Анализ исходного кода:
		+ Использование статического анализа кода (например, SonarQube, Flake8) для выявления ошибок и уязвимостей.
		+ Проверка качества кода на соответствие стандартам разработки.
3. Документирование результатов:
	* Фиксация обнаруженных несоответствий, их причин и последствий.
	* Создание отчётов о выполнении тестов с указанием успешно пройденных этапов.
4. Повторная верификация:
	* Исправление выявленных дефектов.
	* Проведение повторной проверки для подтверждения устранения несоответствий.
5. Заключение по результатам верификации:
	* Подготовка итогового отчёта о результатах верификации.
	* Согласование отчёта с заинтересованными сторонами.

Методы верификации:

* Ручной анализ:
	+ Проверка проектной документации и спецификаций.
* Автоматизированное тестирование:
	+ Юнит-тесты, интеграционные тесты, статический анализ кода.
* Инспекции:
	+ Совместный анализ кода и проектных решений командой разработчиков и тестировщиков.

Результаты процесса:

* Подтверждение соответствия программного обеспечения спецификации и проектным решениям.
* Выявленные и устранённые несоответствия в реализации функций.
* Полный отчёт о верификации, включающий список проверенных компонентов и описание проведённых тестов.

Пример для CVCODE:

1. Планирование:
	* Определены критерии верификации:
		+ Соответствие анализа данных пользователей модели OCEAN.
		+ Корректная работа API для передачи данных в кадровую систему.
2. Проверка:
	* Проведено юнит-тестирование модуля анализа данных, все тесты успешно выполнены.
	* Интеграционное тестирование выявило проблему в передаче данных между API и базой данных, которая была устранена.
	* Статический анализ кода показал 5 несоответствий стандартам, все исправлены.
3. Документирование:
	* Создан отчёт с результатами верификации, включающий перечень исправленных дефектов и результаты повторной проверки.
4. Заключение:
	* По итогам верификации система признана готовой для перехода на этап квалификационного тестирования.

Процесс верификации является важным шагом для обеспечения качества и надёжности платформы CVCODE, минимизируя риски и гарантируя корректность работы.

### **1.3.5. Процесс валидации программных средств**

Процесс валидации программных средств CVCODE направлен на подтверждение того, что программное обеспечение соответствует реальным потребностям пользователей и бизнес-целям заказчика. В отличие от верификации, валидация оценивает систему в условиях её реального использования.

#### **Цели процесса:**

1. Убедиться, что программное обеспечение соответствует заявленным функциональным и нефункциональным требованиям.
2. Подтвердить, что система решает задачи пользователей и приносит ожидаемую ценность.
3. Проверить готовность программного обеспечения к эксплуатации.

#### **Основные этапы:**

1. **Планирование валидации:**
	* Определение ключевых целей валидации, включая:
		+ Проверку выполнения основных сценариев использования (например, регистрация, анализ данных, предоставление рекомендаций).
		+ Оценку пользовательского опыта (UX).
	* Разработка тестовых сценариев, приближённых к реальным условиям эксплуатации.
	* Согласование критериев успеха с заказчиком.
2. **Подготовка к валидации:**
	* Создание тестовой среды, максимально приближённой к рабочей:
		+ Настройка серверов, API и базы данных.
		+ Загрузка тестовых данных, имитирующих реальные данные пользователей.
	* Обучение участников тестирования, если это требуется (например, сотрудников заказчика).
3. **Проведение валидации:**
	* **Функциональное тестирование:**
		+ Проверка выполнения основных функций (например, анализ данных пользователей ВК, проведение тестов).
	* **Пользовательское тестирование:**
		+ Вовлечение реальных пользователей для оценки удобства использования системы.
		+ Сбор обратной связи о работе платформы.
	* **Нагрузочное тестирование:**
		+ Проверка производительности при типичной и пиковой нагрузке.
	* **Анализ соответствия требованиям:**
		+ Сравнение работы системы с требованиями, указанными в спецификации.
4. **Регистрация и анализ результатов:**
	* Документирование выявленных проблем, несоответствий и их влияния на пользовательский опыт.
	* Классификация проблем по степени критичности.
5. **Исправление выявленных проблем:**
	* Внесение изменений в программное обеспечение для устранения несоответствий.
	* Повторная валидация после исправлений.
6. **Заключение по результатам валидации:**
	* Подготовка итогового отчёта, включающего:
		+ Список проверенных функций.
		+ Результаты пользовательского тестирования.
		+ Выявленные и исправленные дефекты.

#### **Методы валидации:**

* **Тестирование на реальных данных:** использование реальных или приближённых к ним данных.
* **Сценарное тестирование:** выполнение тестов по заранее разработанным сценариям.
* **Пользовательское тестирование:** вовлечение конечных пользователей для оценки системы.
* **Демонстрация работы системы заказчику.**

#### **Результаты процесса:**

* Подтверждение соответствия системы потребностям пользователей.
* Описание выявленных проблем и пути их устранения.
* Полный отчёт о готовности системы к эксплуатации.

#### **Пример для CVCODE:**

1. **Планирование:**
	* Разработаны сценарии валидации:
		+ Регистрация пользователя через ВК.
		+ Проведение теста на определение мотивации.
		+ Анализ данных и предоставление рекомендаций.
2. **Проведение валидации:**
	* Проведено тестирование на реальных данных 100 пользователей.
	* Получена обратная связь от сотрудников HR-отдела, использующих CVCODE для оценки кандидатов.
3. **Анализ и исправления:**
	* Выявлены и исправлены несоответствия в модуле формирования рекомендаций.
	* Усовершенствован интерфейс на основе отзывов пользователей.
4. **Заключение:**
	* Подготовлен итоговый отчёт, подтверждающий готовность системы к запуску.

Процесс валидации гарантирует, что CVCODE соответствует ожиданиям пользователей, решает их задачи и готов к внедрению в рабочую среду.

### **1.3.6. Процесс ревизии программных средств**

Процесс ревизии программных средств CVCODE направлен на анализ качества разработки и эксплуатации системы, соответствие стандартам и выявление областей для улучшения. Ревизия фокусируется на проверке процессов, документации, архитектуры и соответствия установленным требованиям.

Цели процесса:

1. Провести всесторонний анализ программного обеспечения и сопутствующих процессов.
2. Выявить несоответствия стандартам и нормативам.
3. Рекомендовать улучшения для повышения качества разработки и эксплуатации.

Основные этапы:

1. Подготовка к ревизии:
	* Определение цели и объёма ревизии:
		+ Проверка соответствия архитектуры требованиям.
		+ Анализ качества кода и документации.
	* Формирование ревизионной группы:
		+ Эксперты по разработке, тестированию и эксплуатации.
	* Подготовка плана ревизии:
		+ Список объектов проверки (модули системы, базы данных, API, документация).
		+ Методы и критерии оценки.
2. Сбор данных:
	* Анализ проектной и эксплуатационной документации:
		+ Сравнение фактического состояния системы с утверждёнными спецификациями.
	* Анализ исходного кода:
		+ Использование инструментов статического анализа (например, SonarQube, Flake8) для выявления дефектов и уязвимостей.
	* Сбор информации о процессе эксплуатации:
		+ Логи работы системы.
		+ Отчёты о сбоях и инцидентах.
3. Оценка процессов:
	* Проверка соответствия процессов разработки стандартам (например, ГОСТ Р ИСО/МЭК 12207-2010).
	* Анализ управления версиями и конфигурацией.
	* Оценка процесса тестирования:
		+ Уровень покрытия тестами.
		+ Качество автоматизированного и ручного тестирования.
4. Выявление несоответствий:
	* Составление перечня обнаруженных проблем:
		+ Недочёты в документации.
		+ Ошибки в реализации функций.
		+ Несоответствия требованиям безопасности.
	* Классификация несоответствий по степени критичности:
		+ Критические (требуют немедленного исправления).
		+ Средние и низкие (планируются для исправления в будущем).
5. Рекомендации по улучшению:
	* Разработка рекомендаций по устранению выявленных проблем.
	* Определение областей для повышения эффективности процессов.
6. Документирование результатов:
	* Подготовка отчёта о ревизии:
		+ Объекты проверки.
		+ Обнаруженные несоответствия.
		+ Рекомендации по улучшению.
	* Передача отчёта заинтересованным сторонам.
7. Мониторинг выполнения рекомендаций:
	* Контроль внедрения предложенных улучшений.
	* Повторная ревизия (при необходимости).

Методы ревизии:

* Инспекция кода: анализ качества кода вручную и с помощью автоматизированных инструментов.
* Сравнительный анализ: сопоставление фактического состояния системы с документацией и требованиями.
* Интервью: опрос разработчиков, тестировщиков и эксплуатационного персонала.

Результаты процесса:

* Полный отчёт о ревизии, включающий выявленные несоответствия и рекомендации по их устранению.
* План мероприятий по улучшению качества системы.
* Обновлённая документация.

Пример для CVCODE:

1. Цели ревизии:
	* Проверка архитектуры системы после внедрения новых модулей.
	* Анализ качества документации для API.
2. Проведение ревизии:
	* Выявлено 5 критических и 10 средних проблем:
		+ Недостатки в защите API.
		+ Отсутствие некоторых разделов в документации.
	* Проведён анализ кода с помощью SonarQube, выявлены потенциальные уязвимости.
3. Рекомендации:
	* Обновить методы авторизации в API.
	* Дополнить документацию описанием новых функций.
4. Результаты:
	* Подготовлен отчёт с планом исправлений, внедрение которого начато через неделю после завершения ревизии.

Процесс ревизии обеспечивает стабильность и соответствие программного обеспечения CVCODE установленным стандартам, помогает выявить слабые места и повышает качество системы.

### **1.3.7. Процесс аудита программных средств**

Процесс аудита программных средств CVCODE направлен на независимую оценку соответствия системы заявленным требованиям, стандартам и нормативам. Аудит проводится для подтверждения качества и эффективности процессов разработки, сопровождения и эксплуатации программного обеспечения.

Цели процесса:

1. Независимая проверка соответствия системы требованиям спецификации и нормативам.
2. Оценка эффективности процессов разработки и сопровождения.
3. Выявление проблем и предоставление рекомендаций по их устранению.

Основные этапы:

1. Планирование аудита:
	* Определение целей аудита:
		+ Проверка выполнения требований ГОСТ Р ИСО/МЭК 12207-2010.
		+ Анализ соответствия внутренних процессов разработке и эксплуатации ПО.
	* Формирование группы аудиторов:
		+ Независимые специалисты (внутренние или внешние).
	* Разработка программы аудита:
		+ Перечень объектов проверки (документация, код, процессы).
		+ Методы и критерии оценки.
2. Сбор данных:
	* Анализ документации:
		+ Проверка спецификаций, проектной документации, руководств пользователей.
	* Изучение отчетов о разработке и сопровождении:
		+ Логи работы системы.
		+ Отчёты об исправлении дефектов.
	* Интервью с ключевыми участниками процесса:
		+ Разработчики, тестировщики, операторы сопровождения.
3. Проведение аудита:
	* Проверка процессов разработки:
		+ Анализ системы управления версиями и изменений (Git).
		+ Оценка уровня автоматизации тестирования.
	* Проверка процессов эксплуатации:
		+ Анализ логов работы системы и зарегистрированных инцидентов.
		+ Оценка оперативности реакции на инциденты.
	* Проверка качества документации:
		+ Соответствие документации фактическому состоянию системы.
		+ Актуальность и полнота руководств и инструкций.
	* Проверка безопасности:
		+ Анализ соответствия стандартам информационной безопасности (например, шифрование данных, защита от атак).
4. Анализ результатов:
	* Выявление несоответствий:
		+ Нарушения стандартов.
		+ Проблемы в процессах или реализации ПО.
	* Классификация несоответствий по уровню критичности:
		+ Критические: угрожают стабильности или безопасности системы.
		+ Средние и низкие: планируются для устранения в будущем.
5. Формирование отчёта:
	* Подготовка итогового документа:
		+ Перечень проверенных объектов.
		+ Список выявленных проблем.
		+ Рекомендации по устранению недостатков.
	* Представление отчёта заинтересованным сторонам.
6. Мониторинг выполнения рекомендаций:
	* Контроль устранения выявленных несоответствий.
	* Проведение повторного аудита (при необходимости).

Методы аудита:

* Инспекция документации и процессов:
	+ Проверка регламентов разработки, тестирования и эксплуатации.
* Инструментальный аудит:
	+ Использование автоматизированных средств для анализа кода (например, SonarQube).
* Опрос и наблюдение:
	+ Интервью с разработчиками, тестировщиками и пользователями.

Результаты процесса:

* Итоговый отчёт о соответствии программного обеспечения стандартам и требованиям.
* Рекомендации по улучшению процессов и устранению проблем.
* Обновление документации и внутренних регламентов.

Пример для CVCODE:

1. Планирование:
	* Цели аудита:
		+ Проверка соответствия API стандартам безопасности.
		+ Анализ процессов исправления дефектов.
	* Определены критерии:
		+ Время реакции на инциденты не должно превышать 2 часов.
2. Проведение:
	* Анализ логов системы показал 98% успешных запросов к API.
	* Выявлены недостатки в документации безопасности, требующие доработки.
3. Результаты:
	* Подготовлен отчёт с рекомендациями:
		+ Усовершенствовать процесс ведения документации.
		+ Ускорить обработку низкоприоритетных инцидентов.
4. Реализация:
	* Через месяц проведён повторный аудит, подтвердивший устранение выявленных несоответствий.

Процесс аудита является неотъемлемой частью обеспечения качества CVCODE, предоставляя объективную оценку состояния системы и помогая улучшить её эффективность и безопасность.

### **1.3.8. Процесс решения проблем в программных средствах**

Процесс решения проблем в программных средствах CVCODE направлен на оперативное выявление, анализ, устранение дефектов и минимизацию их влияния на пользователей. Этот процесс обеспечивает устойчивую работу системы и быстрое восстановление её функциональности в случае возникновения сбоев.

#### **Цели процесса:**

1. Оперативно устранять проблемы, влияющие на работоспособность системы.
2. Предотвращать повторное возникновение дефектов.
3. Минимизировать воздействие проблем на пользователей и бизнес-процессы.

#### **Основные этапы:**

1. **Регистрация проблемы:**
	* Выявление проблемы:
		+ Мониторинг системы (автоматический или ручной).
		+ Обратная связь от пользователей.
	* Регистрация проблемы в системе управления задачами (например, HULY):
		+ Указание описания проблемы, условий её возникновения, и степени влияния.
	* Назначение приоритета:
		+ **Критический:** проблема влияет на ключевую функциональность или безопасность.
		+ **Средний:** проблема вызывает частичные неудобства.
		+ **Низкий:** незначительные дефекты, не влияющие на использование системы.
2. **Анализ проблемы:**
	* Сбор данных:
		+ Логи системы.
		+ Сценарии использования, при которых возникает проблема.
	* Определение причин:
		+ Анализ кода, запросов к API, базы данных.
		+ Использование инструментов диагностики (например, ELK Stack, Sentry).
	* Категоризация:
		+ Программная ошибка (bug).
		+ Конфигурационная проблема.
		+ Неправильное использование системы.
3. **Разработка решения:**
	* Выбор подхода:
		+ Немедленное исправление (для критических проблем).
		+ Плановое исправление в следующем релизе (для проблем с низким приоритетом).
	* Разработка и тестирование исправления:
		+ Написание патча или обновления.
		+ Юнит- и регрессионное тестирование для подтверждения работоспособности исправления.
	* Обновление системы:
		+ Развертывание исправления в тестовой среде.
		+ Выпуск обновления для пользователей (при необходимости).
4. **Валидация исправления:**
	* Проверка работоспособности исправления:
		+ Убедиться, что проблема устранена и система работает корректно.
	* Анализ побочных эффектов исправления:
		+ Проверка, что внесённые изменения не привели к новым проблемам.
5. **Документирование проблемы:**
	* Описание обнаруженной проблемы, её причины и метода устранения.
	* Обновление базы знаний для предотвращения повторного возникновения.
6. **Коммуникация с пользователями:**
	* Информирование пользователей о проблеме (при необходимости).
	* Объяснение причин, текущего статуса и ожидаемого времени исправления.
	* Уведомление пользователей об успешном устранении проблемы.
7. **Превентивные меры:**
	* Анализ проблемы для предотвращения её повторного возникновения.
	* Внедрение улучшений в процессы разработки и тестирования.

#### **Результаты процесса:**

* Устранённые дефекты и восстановленная работоспособность системы.
* Обновлённая база знаний для разработчиков и сопровождения.
* Увеличение уровня удовлетворённости пользователей.

#### **Пример для CVCODE:**

1. **Регистрация проблемы:**
	* Пользователи сообщили о сбое в модуле анализа данных.
	* Проблема зарегистрирована как критическая в системе HULY.
2. **Анализ:**
	* Выявлено, что сбой вызван некорректной обработкой входных данных.
	* Логи указали на сбой в модуле интеграции с базой данных.
3. **Решение:**
	* Разработан патч для обработки некорректных данных.
	* Исправление протестировано в тестовой среде.
4. **Валидация:**
	* Исправление проверено в боевой среде, проблема не воспроизводится.
5. **Документирование:**
	* Описание проблемы и исправления добавлено в базу знаний.
6. **Превентивные меры:**
	* Добавлены дополнительные проверки входных данных на этапе анализа.

Этот процесс обеспечивает непрерывность работы платформы CVCODE, минимизирует влияние дефектов на пользователей и поддерживает высокий уровень доверия к системе.

# **2. Порядок технической поддержки программного обеспечения**

## **2.1. Общие сведения**

Техническая поддержка программного обеспечения CVCODE направлена на обеспечение стабильной работы системы, быстрое реагирование на возникающие проблемы и помощь пользователям в использовании платформы. Поддержка включает выполнение работ по мониторингу, диагностике, устранению дефектов, обновлению и консультированию пользователей.

Основные цели технической поддержки:

1. Гарантировать стабильность работы платформы в любых эксплуатационных условиях.
2. Обеспечить оперативное устранение сбоев и неисправностей.
3. Оказывать пользователям квалифицированную помощь и консультации.
4. Поддерживать актуальность программного обеспечения через регулярные обновления.

Объекты технической поддержки:

1. Программные модули системы:
	* Анализ данных.
	* Модуль тестирования.
	* Пользовательский интерфейс.
2. Интеграционные решения:
	* API для взаимодействия с внешними системами.
	* Взаимодействие с базой данных.
3. Инфраструктура:
	* Серверы и сетевые компоненты.
	* Средства резервного копирования.
4. Пользователи:
	* Физические лица (индивидуальные пользователи платформы).
	* Юридические лица (корпоративные клиенты).

Уровни технической поддержки:

1. Первый уровень (первичная поддержка):
	* Ответы на вопросы пользователей о работе системы.
	* Решение типовых проблем (например, помощь с авторизацией или доступом к данным).
	* Перенаправление более сложных запросов на второй уровень.
2. Второй уровень (техническая поддержка):
	* Диагностика и устранение технических проблем, связанных с функционированием системы.
	* Консультации пользователей по вопросам настройки и интеграции.
	* Мониторинг производительности системы.
3. Третий уровень (экспертная поддержка):
	* Решение сложных технических вопросов, требующих глубоких знаний системы.
	* Устранение проблем в исходном коде.
	* Разработка обновлений и улучшений.

Основные принципы технической поддержки:

1. Доступность:
	* Пользователи могут обратиться за помощью через электронную почту, чат или телефон.
	* Поддержка доступна в рабочие часы, для корпоративных клиентов возможно предоставление 24/7 поддержки.
2. Оперативность:
	* Время реакции на запросы определяется уровнем критичности:
		+ Критические проблемы: немедленное реагирование.
		+ Некритические: обработка в течение согласованного времени.
3. Качество:
	* Каждый запрос фиксируется в системе тикетов.
	* Проводится контроль удовлетворённости пользователей.
4. Прозрачность:
	* Пользователи получают информацию о статусе своего запроса.
	* Отчёты о выполнении работ предоставляются корпоративным клиентам.

Пример для CVCODE:

1. Сценарий для физического лица:
	* Пользователь обратился с проблемой доступа к результатам тестирования.
	* Проблема решена специалистом первого уровня путём восстановления данных через систему администратора.
2. Сценарий для корпоративного клиента:
	* У клиента возникли задержки в обновлении базы данных.
	* Запрос передан на второй уровень, где специалисты устранили сбой в соединении с сервером.
	* Клиенту предоставлен отчёт с описанием проблемы и рекомендациями по её предотвращению.

Этот порядок технической поддержки позволяет обеспечивать высокую надёжность платформы CVCODE, оперативно реагировать на потребности пользователей и поддерживать их удовлетворённость работой системы.

## **2.2. Техническая поддержка первого уровня**

Техническая поддержка первого уровня (First Line Support) является первой точкой взаимодействия пользователей с командой поддержки CVCODE. Её задача — оперативно обрабатывать запросы, предоставлять базовую помощь и направлять более сложные проблемы на второй или третий уровень.

Цели поддержки первого уровня:

1. Предоставлять быстрые и понятные ответы на часто задаваемые вопросы.
2. Устранить простые технические проблемы, не требующие глубокого анализа.
3. Направлять сложные запросы к специалистам более высокого уровня.

Функции технической поддержки первого уровня:

1. Регистрация запросов:
	* Приём обращений через:
		+ Онлайн-чат.
		+ Электронную почту.
		+ Телефонные звонки.
	* Внесение информации о запросе в систему управления заявками (например, HULY, Zendesk).
	* Классификация запроса по типу:
		+ Проблема (например, сбой системы).
		+ Вопрос (например, консультация по использованию функционала).
		+ Предложение (например, идея для улучшения платформы).
2. Решение типовых запросов:
	* Помощь с авторизацией:
		+ Восстановление пароля.
		+ Проверка правильности входа через аккаунт ВКонтакте.
	* Консультации по использованию функционала:
		+ Проведение тестов.
		+ Использование тренажёров.
	* Восстановление доступа:
		+ Обновление данных пользователя в системе.
		+ Проверка статуса подключения API для корпоративных клиентов.
3. Направление запросов на следующий уровень:
	* Оценка сложности и критичности проблемы.
	* Передача запросов на второй уровень при необходимости:
		+ Проблемы с производительностью.
		+ Технические сбои, требующие анализа логов.
4. Информирование пользователей:
	* Предоставление информации о статусе заявки.
	* Оповещение о сроках решения проблемы.
5. Документирование и отчётность:
	* Регистрация всех действий, выполненных для решения запроса.
	* Подготовка отчётов для анализа производительности первой линии поддержки.

Методы и инструменты поддержки первого уровня:

1. Системы управления запросами:
	* Использование тикет-систем (например, Zendesk, Freshdesk) для регистрации и обработки заявок.
2. База знаний:
	* Шаблоны ответов на часто задаваемые вопросы.
	* Руководства по типовым проблемам.
3. Скрипты взаимодействия:
	* Стандартизированные ответы для обеспечения профессионального общения.

Время реагирования:

* Критические запросы: обработка в течение 15 минут.
* Некритические запросы: обработка в течение 1 часа.

Результаты работы первого уровня:

1. Оперативное решение простых запросов пользователей.
2. Снижение нагрузки на второй и третий уровни поддержки.
3. Высокий уровень удовлетворённости пользователей.

Пример работы первого уровня для CVCODE:

1. Запрос: Пользователь не может войти в систему через ВКонтакте.
	* Решение: Специалист проверил состояние подключения API ВКонтакте, обновил данные пользователя и предоставил инструкцию по повторной авторизации. Проблема решена за 10 минут.
2. Запрос: Вопрос о том, как пройти тест на социальный интеллект.
	* Решение: Пользователю предоставлена ссылка на инструкцию и краткое описание шагов. Запрос обработан в чате за 5 минут.
3. Запрос: Зависание системы при загрузке тренажёра.
	* Решение: Запрос зарегистрирован и передан на второй уровень для анализа производительности.

Поддержка первого уровня в CVCODE позволяет эффективно обрабатывать типовые запросы, оставляя ресурсы второй и третьей линии для более сложных технических задач.

## **2.3. Техническая поддержка второго уровня**

Техническая поддержка второго уровня (Second Line Support) предоставляет углубленный анализ и решение технических проблем, которые не удалось устранить на первом уровне. Специалисты второго уровня обладают более глубокими знаниями о внутренней структуре платформы CVCODE и её технических особенностях.

Цели поддержки второго уровня:

1. Диагностировать и устранять сложные технические проблемы.
2. Поддерживать и улучшать работоспособность системы.
3. Взаимодействовать с третьим уровнем поддержки для решения наиболее критических проблем.

Функции технической поддержки второго уровня:

1. Диагностика и анализ:
	* Углублённый анализ запросов, переданных с первого уровня:
		+ Изучение логов системы для выявления причин сбоев.
		+ Проведение диагностики соединений, API и баз данных.
	* Использование инструментов для мониторинга производительности (например, ELK Stack, Prometheus).
2. Решение технических проблем:
	* Исправление ошибок конфигурации:
		+ Настройка серверов, соединений API и базы данных.
	* Устранение проблем с производительностью:
		+ Оптимизация запросов к базе данных.
		+ Анализ и устранение «узких мест» в работе системы.
	* Решение проблем интеграции:
		+ Настройка взаимодействия с внешними системами.
3. Тестирование решений:
	* Проверка работоспособности исправлений в тестовой среде.
	* Убедиться, что исправления не вызывают побочных эффектов.
4. Эскалация на третий уровень:
	* Передача запросов на третий уровень поддержки в случае:
		+ Обнаружения критических дефектов в исходном коде.
		+ Необходимости доработки или переработки модулей.
5. Взаимодействие с клиентами:
	* Предоставление информации о статусе выполнения задач.
	* Консультации по вопросам настройки и интеграции.
6. Документирование:
	* Регистрация всех действий по диагностике и устранению проблем.
	* Обновление базы знаний с описанием решения проблем.

Методы и инструменты второго уровня поддержки:

1. Мониторинг и диагностика:
	* ELK Stack для анализа логов.
	* Prometheus для мониторинга производительности.
2. Системы управления конфигурацией:
	* Ansible, Terraform для автоматической настройки окружений.
3. Системы отслеживания проблем:
	* HULY, Redmine для регистрации запросов и отслеживания их статуса.

Время реагирования:

* Критические проблемы: решение в течение 4 часов.
* Некритические проблемы: решение в течение 1 рабочего дня.

Результаты работы второго уровня:

1. Решение сложных технических проблем, влияющих на работоспособность системы.
2. Минимизация влияния инцидентов на пользователей.
3. Повышение стабильности и производительности системы.

Пример работы второго уровня для CVCODE:

1. Запрос: API платформы возвращает ошибки при обработке данных.
	* Диагностика: Специалисты второго уровня обнаружили, что проблема вызвана превышением лимита запросов в базе данных.
	* Решение: Оптимизированы SQL-запросы, уменьшена нагрузка на сервер.
	* Результат: Проблема устранена в течение 3 часов.
2. Запрос: Задержка в обновлении данных пользователя.
	* Диагностика: Логи показали сбой в работе брокера сообщений (Redis).
	* Решение: Перезапуск службы Redis и обновление её конфигурации.
	* Результат: Данные обновляются корректно, проблема не воспроизводится.
3. Запрос: Некорректная работа интеграции с корпоративной кадровой системой.
	* Диагностика: Выявлено несоответствие форматов данных.
	* Решение: Настроена конвертация данных перед отправкой.
	* Результат: Проблема устранена, интеграция работает корректно.

Техническая поддержка второго уровня обеспечивает профессиональный анализ и устранение сложных проблем, сохраняя стабильность и производительность платформы CVCODE. Она служит связующим звеном между первым уровнем поддержки и глубокой технической проработкой на третьем уровне.

## **2.4. Принципы организации технической поддержки**

Принципы организации технической поддержки программного обеспечения CVCODE обеспечивают стабильную, качественную и оперативную помощь пользователям, гарантируя эффективность работы платформы и высокий уровень удовлетворённости клиентов.

#### **Основные принципы:**

1. **Доступность:**
	* Техническая поддержка должна быть доступна для пользователей через удобные каналы:
		+ Электронная почта.
		+ Онлайн-чат.
		+ Телефонная линия.
	* Возможность обращения в поддержку 24/7 для корпоративных клиентов (при наличии соответствующего соглашения).
2. **Оперативность:**
	* Быстрая реакция на запросы в зависимости от их приоритета:
		+ Критические инциденты — немедленная обработка.
		+ Некритические запросы — обработка в течение согласованного времени.
	* Чёткое распределение времени реакции и решения для каждого уровня поддержки.
3. **Прозрачность:**
	* Пользователи должны быть осведомлены о статусе своих запросов:
		+ Уведомления об изменении статуса заявки (зарегистрировано, в работе, завершено).
		+ Возможность отслеживания прогресса через систему управления запросами.
	* Предоставление отчётов о выполнении работ для корпоративных клиентов.
4. **Профессионализм:**
	* Поддержка должна предоставляться квалифицированными специалистами, обладающими глубокими знаниями платформы.
	* Регулярное обучение сотрудников для повышения их компетенций:
		+ Новые технологии.
		+ Обновления функционала платформы.
5. **Инкрементальная эскалация:**
	* Плавный переход запросов между уровнями поддержки:
		+ Первый уровень решает типовые запросы и проблемы.
		+ Второй уровень занимается техническими неисправностями.
		+ Третий уровень отвечает за сложные случаи и исправление кода.
6. **Качество:**
	* Постоянный контроль качества предоставляемой поддержки:
		+ Анализ времени решения проблем.
		+ Оценка обратной связи от пользователей.
	* Минимизация повторяющихся ошибок и проблем.
7. **Персонализированный подход:**
	* Индивидуальный подход к пользователям:
		+ Учет истории обращений для персонализации решений.
		+ Адаптация решений под конкретные условия и требования клиентов.
8. **Предупреждение проблем:**
	* Превентивные действия для минимизации вероятности возникновения инцидентов:
		+ Регулярный мониторинг системы.
		+ Проведение профилактических работ и обновлений.

#### **Реализация принципов:**

1. **Системы и инструменты:**
	* Использование системы управления заявками (например, HULY, Zendesk) для отслеживания запросов.
	* Мониторинг производительности и состояния системы через инструменты (например, Prometheus, ELK Stack).
2. **Процессы:**
	* Чётко определённые регламенты для каждого уровня поддержки.
	* Использование шаблонов для типовых запросов и проблем.
3. **Обратная связь:**
	* Регулярный сбор отзывов от пользователей.
	* Проведение анализа удовлетворённости клиентов и внедрение улучшений на основе полученных данных.
4. **Интеграция с разработкой:**
	* Техническая поддержка тесно связана с командой разработчиков:
		+ Передача сложных запросов и предложений по улучшению функционала.
		+ Совместное тестирование сложных исправлений.

#### **Пример для CVCODE:**

1. **Доступность:**
	* Физические лица могут обратиться через форму обратной связи на сайте.
	* Корпоративные клиенты имеют выделенную линию для связи 24/7.
2. **Оперативность:**
	* Среднее время ответа на запросы первого уровня составляет 10 минут.
	* Все критические запросы решаются в течение 4 часов.
3. **Прозрачность:**
	* Пользователи получают уведомления об изменении статуса заявки на почту.
	* Корпоративные клиенты имеют доступ к порталу для мониторинга запросов.
4. **Профессионализм:**
	* Специалисты второго уровня прошли сертификацию по работе с API и интеграциями.

Принципы организации технической поддержки CVCODE гарантируют, что пользователи могут эффективно работать с платформой, оперативно получать помощь и оставаться удовлетворёнными качеством предоставляемых услуг.

# **3. Устранение неисправностей, выявленных в ходе эксплуатации программного обеспечения**

Процесс устранения неисправностей в ходе эксплуатации программного обеспечения CVCODE направлен на оперативное выявление, анализ и исправление проблем для обеспечения стабильной работы системы и минимизации её влияния на пользователей.

#### **Цели процесса:**

1. Быстрое выявление и устранение дефектов, возникающих в процессе эксплуатации.
2. Минимизация сбоев и простоев системы.
3. Предотвращение повторного возникновения проблем.

#### **Основные этапы устранения неисправностей:**

1. **Выявление проблемы:**
	* Регистрация инцидента пользователем или автоматизированной системой мониторинга:
		+ Логирование ошибок в системе.
		+ Жалобы или обращения от пользователей.
	* Классификация проблемы:
		+ **Критическая:** нарушение работы ключевых функций системы.
		+ **Средняя:** частичное нарушение работы платформы.
		+ **Низкая:** незначительные дефекты, не влияющие на функциональность.
2. **Анализ проблемы:**
	* Сбор данных о проблеме:
		+ Логи системы (например, с помощью ELK Stack).
		+ Описание условий воспроизведения проблемы.
		+ История изменений или обновлений, предшествующих возникновению неисправности.
	* Определение причины:
		+ Ошибка в коде.
		+ Проблемы конфигурации.
		+ Некорректное использование системы пользователями.
3. **Разработка и тестирование решения:**
	* Разработка исправления:
		+ Внесение изменений в код или конфигурацию.
		+ Обновление зависимостей (если требуется).
	* Тестирование исправлений:
		+ Юнит-тестирование для проверки работоспособности исправления.
		+ Регрессионное тестирование для предотвращения появления новых дефектов.
	* Тестирование в тестовой среде для оценки влияния изменений.
4. **Развёртывание исправления:**
	* Подготовка обновления:
		+ Создание патча или новой версии системы.
		+ Обновление документации (если необходимо).
	* Развёртывание исправления:
		+ Установка исправления на рабочую среду.
		+ Мониторинг работы системы после обновления.
5. **Документирование проблемы и решения:**
	* Описание проблемы, её причины и метода исправления.
	* Обновление базы знаний для предотвращения повторного возникновения проблемы.
	* Подготовка отчёта для заинтересованных сторон.
6. **Обратная связь и мониторинг:**
	* Информирование пользователя о статусе решения проблемы.
	* Мониторинг исправленного функционала для предотвращения рецидивов.

#### **Роли и ответственность:**

1. **Специалисты первой линии поддержки:**
	* Принимают запросы от пользователей.
	* Передают информацию о проблемах на второй уровень.
2. **Специалисты второй линии поддержки:**
	* Диагностируют проблему и устраняют неисправности, связанные с конфигурацией и инфраструктурой.
3. **Разработчики (третий уровень поддержки):**
	* Вносят изменения в код для устранения сложных дефектов.
	* Разрабатывают и тестируют исправления.

#### **Методы и инструменты:**

* **Инструменты мониторинга:** Prometheus, ELK Stack для анализа производительности и логов.
* **Системы управления задачами:** HULY для регистрации и отслеживания дефектов.
* **Среды тестирования:** Docker для проверки исправлений в изолированной среде.
* **Системы контроля версий:** Git для управления изменениями в коде.

#### **Результаты процесса:**

1. Устранённые дефекты и восстановленная функциональность системы.
2. Обновлённая база знаний с описанием исправлений.
3. Увеличение стабильности и надёжности работы платформы.

#### **Пример для CVCODE:**

1. **Выявление проблемы:**
	* Пользователь сообщил, что модуль анализа данных выдаёт ошибку при обработке некоторых профилей.
	* Проблема зарегистрирована в HULY как инцидент с высоким приоритетом.
2. **Анализ:**
	* Логи системы показали, что ошибка возникает из-за некорректной обработки данных с отсутствующими полями.
	* Причина — дефект в алгоритме обработки входных данных.
3. **Разработка решения:**
	* В код добавлены проверки на наличие всех необходимых полей.
	* Исправление протестировано в тестовой среде.
4. **Развёртывание:**
	* Исправление установлено в рабочей среде в течение 4 часов.
	* Проведён мониторинг работы модуля, проблема не воспроизводится.
5. **Документирование:**
	* Описание проблемы и решения добавлено в базу знаний.
	* Обновлена инструкция по тестированию аналогичных сценариев.

Устранение неисправностей в CVCODE обеспечивает непрерывную работу платформы, минимизирует негативное влияние на пользователей и поддерживает высокий уровень доверия к системе.

# **Совершенствование программного обеспечения**

Совершенствование программного обеспечения CVCODE направлено на систематическое обновление и улучшение платформы для удовлетворения растущих требований пользователей, внедрение новых функций, повышение производительности и обеспечения соответствия современным стандартам.

#### **Цели процесса:**

1. Постоянное улучшение функциональности программного обеспечения.
2. Обеспечение соответствия современным технологическим и нормативным требованиям.
3. Повышение производительности, удобства использования и безопасности платформы.

#### **Основные этапы совершенствования:**

1. **Сбор предложений и обратной связи:**
	* Мониторинг отзывов пользователей:
		+ Анализ обращений в техническую поддержку.
		+ Опросы и анкетирование пользователей.
	* Инициативы команды разработчиков:
		+ Предложения по оптимизации архитектуры.
		+ Идеи по внедрению новых технологий.
	* Анализ рыночных трендов:
		+ Изучение конкурентных решений.
		+ Адаптация платформы под новые бизнес-требования.
2. **Анализ и планирование:**
	* Оценка целесообразности изменений:
		+ Анализ выгоды для пользователей и затрат на внедрение.
		+ Оценка возможных рисков.
	* Приоритизация улучшений:
		+ Критические обновления (например, устранение уязвимостей).
		+ Развитие существующих функций.
		+ Внедрение новых модулей и технологий.
	* Разработка плана обновлений:
		+ Определение этапов, сроков и ресурсов для реализации.
3. **Разработка и тестирование улучшений:**
	* Реализация изменений в коде:
		+ Добавление новых функций и модулей.
		+ Оптимизация существующего функционала.
	* Тестирование обновлений:
		+ Юнит- и интеграционные тесты.
		+ Пользовательское тестирование новых функций.
	* Проверка на совместимость с существующими компонентами платформы.
4. **Развёртывание обновлений:**
	* Разграничение этапов:
		+ Тестовая среда (предварительная проверка).
		+ Постепенное внедрение для групп пользователей (A/B-тестирование).
		+ Полное развёртывание на рабочей среде.
	* Обеспечение резервного восстановления:
		+ Создание бэкапов перед установкой обновлений.
		+ Возможность отката изменений при выявлении проблем.
5. **Мониторинг и сбор отзывов:**
	* Анализ работы обновлённой системы:
		+ Логи производительности.
		+ Обратная связь пользователей.
	* Внесение корректировок:
		+ Быстрое реагирование на проблемы, выявленные после обновлений.
6. **Документирование и обучение:**
	* Обновление пользовательской и технической документации:
		+ Руководства по использованию новых функций.
		+ Изменения в API для интеграций.
	* Проведение тренингов для пользователей и сотрудников.

#### **Методы и инструменты:**

* **Системы управления проектами:** HULY для планирования и контроля задач по совершенствованию.
* **Средства тестирования:** Selenium, PyTest для автоматизации тестов.
* **Мониторинг:** Prometheus, Grafana для анализа производительности после обновлений.
* **Коммуникация с пользователями:** Чат-боты, формы обратной связи, уведомления о нововведениях.

#### **Результаты процесса:**

1. Расширение функциональности и улучшение пользовательского опыта.
2. Увеличение производительности и стабильности системы.
3. Повышение конкурентоспособности платформы на рынке.

#### **Пример совершенствования программного обеспечения для CVCODE:**

1. **Сбор предложений:**
	* Пользователи корпоративного сегмента предложили интеграцию с системами управления обучением (LMS).
	* Команда разработчиков выявила необходимость оптимизации модуля анализа данных.
2. **Анализ и планирование:**
	* Принято решение о разработке API для интеграции с LMS.
	* Запланировано обновление алгоритмов анализа для повышения точности.
3. **Разработка и тестирование:**
	* Разработан API для передачи результатов тестов в LMS.
	* Проведено A/B-тестирование нового алгоритма анализа.
4. **Развёртывание:**
	* API развернут для тестовой группы клиентов.
	* Постепенное внедрение нового алгоритма для всех пользователей.
5. **Мониторинг:**
	* Отмечено снижение времени обработки данных на 25%.
	* Пользователи LMS высоко оценили новую интеграцию.

Совершенствование программного обеспечения CVCODE — это непрерывный процесс, направленный на удовлетворение потребностей пользователей и поддержку платформы на уровне современных технологических и функциональных стандартов.

# **5.** **Информация о персонале**

Для эффективного выполнения задач по разработке, поддержке и совершенствованию программного обеспечения CVCODE привлекается высококвалифицированный персонал с чётко распределёнными ролями и обязанностями. Команда формируется с учётом требований каждого этапа жизненного цикла программного обеспечения.

#### **Основные категории персонала:**

1. **Разработчики:**
	* Разрабатывают функциональные модули, тестируют и оптимизируют программное обеспечение.
	* Занимаются реализацией новых функций и исправлением дефектов.
	* Внедряют стандарты кодирования и следят за качеством исходного кода.
2. **Тестировщики:**
	* Проверяют работоспособность всех компонентов системы.
	* Разрабатывают сценарии тестирования (автоматизированного и ручного).
	* Проводят нагрузочные и регрессионные тесты.
3. **Системные администраторы:**
	* Обеспечивают бесперебойную работу серверов и сетей.
	* Настраивают инфраструктуру для развертывания и эксплуатации ПО.
	* Мониторят производительность системы и проводят профилактическое обслуживание.
4. **Менеджеры проектов:**
	* Координируют работу команды.
	* Планируют задачи, сроки и ресурсы для реализации проектов.
	* Контролируют выполнение требований заказчиков и внутренние процессы.
5. **Инженеры по поддержке:**
	* Оказывают помощь пользователям (первый и второй уровни поддержки).
	* Устраняют проблемы, возникающие в процессе эксплуатации.
	* Консультируют клиентов по функционалу платформы.
6. **Эксперты по данным и искусственному интеллекту:**
	* Разрабатывают алгоритмы анализа данных и нейронные сети.
	* Обучают модели на пользовательских данных.
	* Проводят исследования для повышения точности и эффективности алгоритмов.
7. **Специалисты по информационной безопасности:**
	* Обеспечивают защиту данных пользователей.
	* Разрабатывают и внедряют политики безопасности.
	* Контролируют соответствие системы требованиям законодательства (например, ФЗ-152).
8. **Технические писатели:**
	* Создают и обновляют документацию для пользователей и разработчиков.
	* Обеспечивают доступность и понятность руководств по использованию платформы.
	* Ведут базу знаний для сотрудников.

#### **Требования к персоналу:**

1. **Квалификация:**
	* Высшее образование в области IT, математики или инженерии.
	* Знание современных технологий разработки, тестирования и эксплуатации ПО.
	* Опыт работы на аналогичных проектах.
2. **Компетенции:**
	* Разработчики: владение языками программирования (Python, JavaScript), знание фреймворков (Django, React).
	* Тестировщики: умение работать с инструментами автоматизации тестирования (Selenium, PyTest).
	* Системные администраторы: опыт работы с контейнеризацией (Docker, Kubernetes), настройкой CI/CD.
	* Эксперты по данным: навыки работы с ML-библиотеками (Scikit-learn, TensorFlow).
3. **Навыки работы в команде:**
	* Эффективная коммуникация.
	* Способность к совместному решению сложных задач.
	* Ориентированность на результат.

#### **Обязанности персонала по этапам жизненного цикла:**

1. **Разработка:**
	* Разработчики и тестировщики работают над реализацией новых функций.
	* Эксперты по данным внедряют и тестируют алгоритмы анализа.
2. **Техническая поддержка:**
	* Инженеры по поддержке обрабатывают запросы пользователей.
	* Системные администраторы устраняют сбои и оптимизируют производительность.
3. **Совершенствование:**
	* Менеджеры проектов организуют внедрение улучшений.
	* Разработчики и эксперты по данным реализуют новые модули.
4. **Безопасность:**
	* Специалисты по информационной безопасности проводят аудит системы.
	* Разрабатывают рекомендации по защите данных.

#### **Пример распределения обязанностей в CVCODE:**

1. **Разработка нового модуля анализа данных:**
	* Разработчики создают интерфейсы API.
	* Эксперты по данным внедряют алгоритмы классификации.
	* Тестировщики проверяют точность анализа на тестовых данных.
2. **Устранение проблемы в интеграции:**
	* Инженеры по поддержке фиксируют инцидент.
	* Системные администраторы анализируют сбои в соединении с базой данных.
	* Разработчики устраняют ошибку в конфигурации.
3. **Обновление системы:**
	* Менеджеры проектов составляют план обновления.
	* Разработчики и тестировщики готовят релиз.
	* Системные администраторы развёртывают обновление в рабочей среде.

Информация о персонале и распределение обязанностей помогают эффективно организовать процессы разработки и поддержки платформы CVCODE, обеспечивая её стабильность, функциональность и соответствие потребностям пользователей.