Выпускная квалификационная работа на тему «Разработка модуля системы CSIA для мониторинга описания работ по разработке ПО в системах контроля версий и управления проектами»

Исходные данные к работе

1. ГОСТ 34.602.89 «Техническое задание на создание автоматизированной системы»
2. СМК-О-СМГТУ-36-20 «Выпускная квалификационная работа: структура, содержание, общие правила выполнения и оформления»
3. Документация к системе RadixWare компании ООО «Компас Плюс»
4. Структура системы контроля версий «Subversion»
5. Структура системы контроля версий «GitTea»
6. Структура системы управления проектами «Jira»

Перечень вопросов, подлежащих разработке в выпускной квалификационной работе:

1. Анализ требований, предъявляемых к разрабатываемой системе
2. Изучение систем контроля версий и систем управления проектами
3. Разработка проектного решения модуля на базе системы «CSIA»
4. Разработка модуля на базе системы «CSIA»

Графическая часть: подготовка электронной презентации выпускной квалификационной работы средствами ...

Слайд 1 - ...

...

**Реферат**

Тема выпускной квалификационной работы: «Разработка модуля системы CSIA для мониторинга описания работ по разработке ПО в системах контроля версий и управления проектами».

Ключевые слова: СИСТЕМА КОНТОРЯ ВЕРСИЙ, СИСТЕМА УПРАВЛЕНИЯ ПРОЕКТАМИ, ...

Пояснительная записка содержит n страниц без приложения, n иллюстраций, n информационных источников.

Объектом исследования в данной выпускной квалификационной работе является процесс разработки программного обеспечения. Предмет исследования – описание работы в системах контроля версий и управления проектами.

Цель данной работы – разработка модуля системы CSIA для мониторинга описания работ по разработке ПО в системах контроля версий и управления проектами.

В пояснительной записке изучены особенности работы систем контроля версий и управления проектами. Рассмотрены проектные решения по разработке модуля на базе системы CSIA. ...

Отзыв

Содержание

[Введение 8](#_Toc1)

[1 Особенности разработки программных продуктов в ООО «Компас плюс» 9](#_Toc2)

[1.1 Общие характеристики коммерческих продуктов компании 9](#_Toc3)

[1.1.1 Общая архитектура продуктов 9](#_Toc4)

[1.1.2 Хостовые приложения 12](#_Toc5)

[1.1.3 Front-end приложения 13](#_Toc6)

[1.2 Системы используемые при разработке ПО 14](#_Toc7)

[1.2.1 Система контроля версий Subversion 14](#_Toc8)

[1.2.2 Система контроля версий Git 16](#_Toc9)

[1.2.3 Система управления проектами Jira 18](#_Toc10)

[1.3 Нормативы описания разработок 21](#_Toc11)

[1.3.1 Формат описания разработок в системах контроля версий 21](#_Toc12)

[1.3.2 Регламенты описания задач в системах управления проектами 22](#_Toc13)

[1.4 Контроль качества продукта 22](#_Toc14)

[1.4.1 Тестирование 23](#_Toc15)

[1.4.2 Документация 23](#_Toc16)

[1.4 Выводы к первой главе 23](#_Toc17)

[2 Проектные решения разработки модуля системы csia для мониторинга описания работ по разработке по в системах контроля версий и управления проектами 24](#_Toc18)

[2.1 Общие сведения о система CSIA 24](#_Toc19)

[2.1.1 Платформа RadixWare 24](#_Toc20)

[2.1.2 Модули системы CSIA 24](#_Toc21)

[2.2 Требования к разрабатываемому модулю 24](#_Toc22)

[2.3 Организация взаимодействия с системами контроля версий 24](#_Toc23)

[2.3.1 Взаимодействие с Subverion 24](#_Toc24)

[2.3.2 Взаимодействие с GitTea 24](#_Toc25)

[2.4 Организация взаимодействия с системой управления проектами Jira 24](#_Toc26)

[2.5 Мониторинг нарушений в проекте 24](#_Toc27)

[2.5.1 Структурная модель 24](#_Toc28)

[2.5.2 Алгоритм поиска нарушений 24](#_Toc29)

[2.6 Генерация отчета 24](#_Toc30)

[2.6.1 Работа со статистикой нарушений 24](#_Toc31)

[2.6.2 Алгоритм генерации отчета 24](#_Toc32)

[2.7 Выводы ко второй главе 24](#_Toc33)

[3 Результаты опытной эксплуатации модуля системы csia для мониторинга описания работ по разработке по в системах контроля версий и управления проектами 26](#_Toc34)

[3.1 Работа с разработанными сущностями 26](#_Toc35)

[3.1.1 Сущность проекта 26](#_Toc36)

[3.1.2 Сущность репозитория 26](#_Toc37)

[3.1.3 Сущности системы управления проектами 26](#_Toc38)

[3.2 Создание и настройка задачи для мониторинга 26](#_Toc39)

[3.2.1 Задача Task.CheckProjectViolations 26](#_Toc40)

[3.2.2 Настройка расписания 26](#_Toc41)

[3.3 Пример сгенерированного отчёта 26](#_Toc42)

[3.4 Выводы к третьей главе 26](#_Toc43)

[Заключение 27](#_Toc44)

[Список используемых источников 28](#_Toc45)

[Приложение А 29](#_Toc46)

# Введение

В настоящее время программное обеспечение ...

# **1 Особенности разработки программных продуктов в ООО «Компас плюс»**

## **1.1 Общие характеристики коммерческих продуктов компании**

### 1.1.1 Общая архитектура продуктов

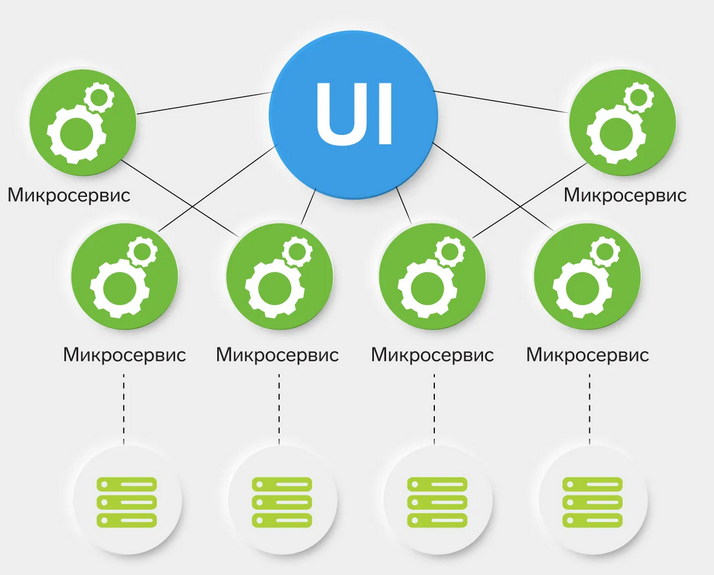
Специфика коммерческих продуктов компании предполагает большой набор критериев, которым должны соответствовать разрабатываемые информационные системы. Из основных выделяются следующие критерии:

1. Отказоустойчивость – системы, работающие в реальном времени и выполняющие услуги по денежному переводу, что является одной из основных задач продуктов компании, должны работать с минимальным количеством сбоев. Клиент должен быть уверен, что его перевод совершится, либо все связанные с переводом сущности вернутся в исходное состояние, иными словами должна обеспечиваться транзакционная целостность. Помимо этого, выход из строя одного из компонентов системы не должен приводить к неработоспособности всей системы в целом;
2. Масштабируемость – информационная система должная обладать свойствами горизонтального масштабирования. Это необходимо для распределения высокой нагрузки на систему. Сервисы системы, физически, могут находиться на различных вычислительных устройствах и использовать максимальное количество доступных им ресурсов, что увеличивает скорость выполнения запроса;
3. Эффективное выполнение задач – скорость выполнения запроса очень важна. Это улучшает пользовательский опыт взаимодействия с системой и позволяет обрабатывать большее количество запросов в единицу времени. Очевидно, что любая задача может быть выполнена на любом современном языке программирования. Однако это вовсе не означает что такая задача будет выполнена эффективно. Поэтому существует необходимость создания продукта, состоящего из ряда систем, написанных на языках программирования, которые наиболее эффективно выполняют ту или иную задачу;
4. Поддержка кодовой базы – современные программные продукты имеют весьма большую кодовую базу, которая легко может содержать несколько сотен тысяч строк кода. Поддержка таких продуктов достаточно сложная задача. Для упрощения процесса поддержки необходимо разделять части продукта друг от друга для того, чтобы иметь четкое представление о том, что выполняет каждый элемент разрабатываемой системы.

Одной из архитектур, хорошо подходящих под данные критерии является микросервисная архитектура.

Микросервисная архитектура – архитектура программного продукта, основанная на создании небольших, автономных, совместно работающих сервисах [1].

На рисунке 1 представлен пример микросервисной архитектуры. На нем видно, что существует система, предоставляющая графический пользовательский интерфейс (UI). Выполняя какой-либо запрос пользователя обращение не поступает только к одной системе, а в зависимости от поставленной задачи используется определенный микросервис. В зависимости от необходимости, микросервисы могут обращаться друг к другу, либо, например, иметь подключение к базе данных.



*Рисунок 1 – Пример микросервисной архитектуры*

Выход из строя какого-либо элемента в этой сети не приводит к прекращению работы всего продукта, так как каждый микросервис является независимой системой, работоспособность которой не влияет на все остальные элементы. Выход из строя микросервиса может означать лишь прекращение работы определенного функционала, но никак не всего продукта.

Масштабируемость может достигается с помощью клонирования микросервиса, что приводит к распределению нагрузки и повышению скорости работы всего продукта.

Каждый микросервис может быть написан на любом языке программирования с использованием любой технологии, подходящей для конкретной задачи которая ставится перед микросервисом. Это, как уже было сказано, также благоприятно сказывается на скорости работы продукта. Помимо этого, такой подход позволяет поделить разработку продукта между несколькими, независимыми друг от друга командами, что в свою очередь повышает скорость разработки продукта, а также его качество, так как решаемые микросервисом задачи ограничены и относятся к одной области, что позволяет сфокусироваться только на их решении.

При таком подходе, деление продукта на компоненты, имеющие ограниченную область применения, происходит само собой. Каждый микросервис проектируется и разрабатывается с ограниченной функциональностью и не может заменять другой микросервис. Это позволяет разработчику достаточно быстро разобраться в структуре работы данного сервиса и внести необходимые изменения.

Таким образом, микросервисная архитектура является наиболее подходящим архитектурным паттерном для разработки высоконагруженых программных продуктов и успешно используется на протяжении нескольких лет разработки в компании «Компас плюс».

### ***1.1.2 Хостовые приложения***

Основой любого коммерческого продукта, разрабатываемого в компании «Компас плюс», является хостовое приложение.

Хостовое приложение – программное обеспечение, выполняющее основные функции разрабатываемого продукта.

В зависимости от проекта, хостовое приложение может иметь различную функциональность, архитектуру и окружение. В данном случае, под окружением понимается ряд дополнительных систем, разработанных, как в компании, так и вне её, которые в совокупности с хостовым приложением и формируют готовый программный продукт.

В качестве основы хостового приложения используется одна из двух систем. TranzWare и RadixWare.

TranzWare – это система, использующая объектно-ориентированный подход при разработке приложения. Немного про TW.

Со временем, на смену TranzWare пришла система RadixWare. Немного про Radix.

### 1.1.3 Front-end приложения

Помимо хостового приложения, проект может состоять из ряда дополнительных систем, называемыми Front-end приложениями.

Front-end приложения – программные продукты, отвечающие за взаимодействие пользователя с системой. Данные продукты являются прослойкой между пользователем и хостовой системой. Они, с помощью пользовательского интерфейса, позволяют пользователю передавать информацию к хосту и выполнять различные операции.

В качестве основы взаимодействия с пользователем может выступать как веб-страница (в качестве примера таких сервисов могут выступать Merchat portal, HPP, Internet bank), так и мобильное приложение (Mobicash, Pocket bank).

Однако нередки случаи, когда системы, которые непосредственно предоставляют взаимодействие с пользователем являются лишь частью от общего числа Front-end приложений. К Front-end приложениям можно также отнести и вспомогательные системы для кэширования данных, системы хранения статических ресурсов, системы маршрутизации запросов с возможностью конвертации протокола взаимодействия. Все эти системы могут быть написаны с помощью различных технологий. Можно выделить основные технологии и языки программирования, используемые при разработке Front-end приложений:

1. NodeJS – асинхронная среда выполнения JavaScript кода, предназначенная для создания масштабируемых сетевых приложений;
2. React – декларативная JavaScript библиотека для создания пользовательских интерфейсов;
3. Android – программный стек на базе Linux с открытым исходным кодом, созданный для широкого спектра устройств и форм-факторов;
4. Swift – язык программирования, предназначенный для создания приложений, работающих на операционной система iOS;
5. Spring – фреймворк для языка программирования Java, предназначенный для разработки веб-сервисов.

## 1.2 Системы используемые при разработке ПО

### 1.2.1 Система контроля версий Subversion

В настоящее время, разрабатываемые системы имеют огромную сложность и объемы разработки. Проектирование и реализация такой системы одним человеком просто невозможна, так как это займет большое количество времени и сильно снизит качество конечного продукта. Поэтому над разработкой систем большой сложности, зачастую, трудится команда из нескольких разработчиков.

В свою очередь, разработку в команде трудно представить без какой-либо системы контроля версий. Это обуславливается тем, что все разработчики должны иметь доступ к исходному коду. Должно осуществляться логирование, версионирование и хранение всех внесенных изменений. Одной из таких систем контроля версий является Subversion.

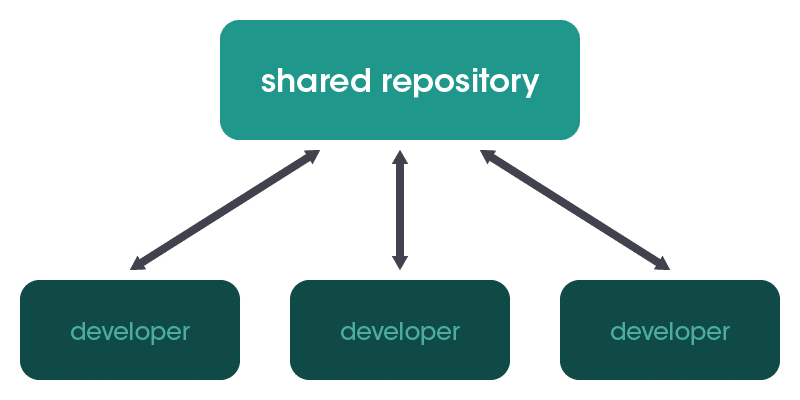
Subverison (SVN) – система контроля версий с открытым исходным кодом. Она позволяет организовать управление каталогами и файлами, а также изменениями, внесенными с течением времени. Это даёт возможность восстановить ранние версии данных и изучить историю всех изменений [2].

В 2004 году, данная система пришла как альтернатива существующей на тот момент CVS. Изначально, SVN создавался для того, чтобы исправить основные недостатки CVS, вместе с тем, сохранив высокую совместимость между этими системами.

К основным отличиям SVN от CSV можно отнести следующее:

1. Контроль изменений каталогов. CSV имеет возможность отслеживания только файлов, в то время как SVN имеет свою собственную «виртуальную» файловую систему, что позволяет этой системе отслеживать изменения каталогов;
2. Настоящая история версий. CVS позволяет контролировать только изменения в файлах. Изменений названий и копирование файлов не поддерживается. SVN же дает возможность копирования, удаления и переименование как каталога, так и файла с возможностью сохранения истории;
3. Атомарная фиксация изменений. Ряд изменений имеет возможность попасть в хранилище либо целиком, либо не попасть туда вовсе. Это предотвращает наличие ошибок вида частичного внесения изменений в хранилище;
4. Выбор средства доступа к хранилищу по сети. SVN имеет абстракцию над доступом к хранилищу. Такой подход дает возможность реализации с помощью различных сетевых механизмов доступа. Можно организовать доступ как по протоколу HTTP, так и туннелировать передаваемые данные через SSH;
5. Единый способ работы с данными. Поиск различий между файлами выполняется с помощью специального бинарного алгоритма. Данный алгоритм работает одинаково как с бинарными, так и с текстовыми данными;
6. Эффективная работа с ветками и метками. Создание ветки (копии проекта с дополнительными изменениями) не должно сопровождаться полным копированием проекта и пропорциональному увеличению занимаемого на диске пространства. Subversion создает ветки с помощью механизма, похожего на механизм создания жестких ссылок в файловых системах. Благодаря этому сокращается используемое дисковое пространство и увеличивается скорость создания такой ветки [2].

В основе данной системы контроля версий лежит принцип централизованной модели (рисунок 2). Этот принцип означает наличие единственной «главной копии». Локально хранятся только те файлы, которые были изменены разработчиком. Когда разработчик вносит изменения в репозиторий это приводит к ограничению доступа для других членов команды. Этот контролируемый подход к рабочему процессу разработки снижает риск возникновения конфликтов и привлекает из-за его безопасного и централизованного характера и четкого процесса.



*Рисунок 2 – структура организации Subversion*

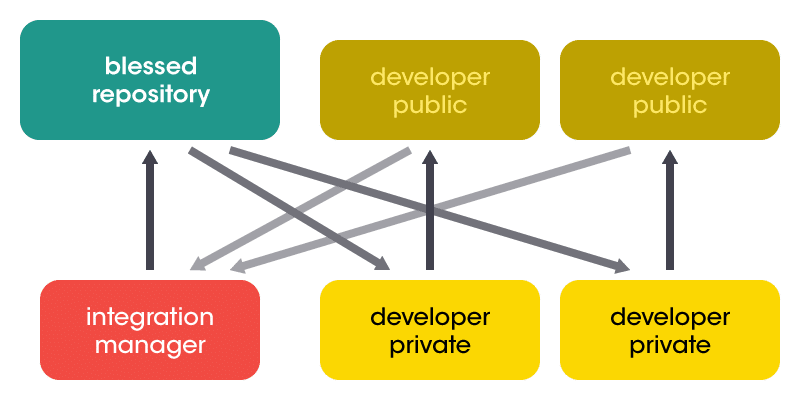
Все вышеперечисленные параметры привели к тому, что все хостовые приложения, разрабатываемые в компании «Компас плюс» используют именно эту систему контроля версий. Более того, технологические платформы, с помощью которых и ведется разработка хостовых систем имеют полную поддержку данной системы контроля версий.

### 1.2.2 Система контроля версий Git

Помимо Subversion и CSV существуют и другие системы контроля версий. Одна из таких систем является Git.

Git – распределенная система контроля версий. В таких системах каждый разработчик проекта имеет возможность создания локального репозитория, который является полной копией центрального хранилища. Благодаря такому подходу каждый разработчик может работать над проектом автономно на своем компьютере. В тот момент, когда изменения будут готовы, разработчик может отправить их в общее хранилище и объединить.

В основе системы Git, как уже было сказано лежит распределенный подход (рисунок 3). Это означает, что все разработчики имеют одинаковый доступ ко всем файлам в хранилище и каждый из них может склонировать хранимый проект с исходным кодом и всеми метаданными. Работа с локальным репозиторием приводит к тому, что, за исключением внесения изменений, все остальные действия могут быть выполнены быстро, поскольку они затрагивают только файлы на локальном диске разработчика, а не на удаленном сервере, как это происходит в SVN. Это позволяет нескольким членам команды получать доступ к одному и тому же файлу и вносить изменения по мере необходимости, но это удобство сопряжено с риском конфликтов слияния Git. Несмотря на это, разработчики по всему миру понимают ценность распределенной модели, поскольку она упрощает совместное использование кода и облегчает удобную совместную работу с большими группами.



*Рисунок 3 – структура работы Git*

К основным отличиям Git от Subversion можно отнести следующее:

1. Автономная работа. SVN требует, чтобы участники были подключены к главному серверу хранилища, что существенно исключает работу в автономном режиме. Git же, наоборот, не требует постоянного подключения к серверу. Единожды склонировав хранилище можно полноценно работать, иногда, синхронизируясь с основным сервером, загружая в него свои изменения и сохраняя к себе изменения других разработчиков.
2. Разрешение конфликтов. Git также превосходит SVN, когда дело доходит до объединения и разрешения конфликтов. Git был разработан для работы с открытым исходным кодом, где многочисленные участники могут работать над одними и теми же частями кодовой базы. Для обеспечения такого типа совместной работы Git создала надежную систему разрешения конфликтов слияния, которая делает процесс более плавным и управляемым
3. Отказоустойчивость. Распределенная модель контроля версий Git помогает снизить вероятность потери основного хранилища. Поскольку участники клонируют основной репозиторий, риск полной потери основного репозитория значительно снижается. С другой стороны, централизованная модель управления версиями SVN создает потенциал для единой точки отказа, если что-то случится с основным репозиторием.

Такая структура работы отлично подходит для Front-end разработки. Поэтому многие приложения, относящиеся к данной категории используют именно Git из-за его гибкости, работы с ветками и возможностями слияния изменений.

### 1.2.3 Система управления проектами Jira

Для эффективной работы в команде требуется придерживаться четких правил и методологий разработки. Это важно, так как подход к разработке программного обеспечения напрямую связан с успешностью разрабатываемого продукта, ведь без правильного подобранной методологии достижение стабильности в работе продукта, безопасности и устойчивости функциональных особенностей практически невозможно.

Методология разработки ПО – это система, определяющая порядок выполнения задач, методы оценки и контроля. Модели разработки ПО выбирают, исходя из направления проекта, его бюджета, сроков реализации конечного продукта. Подходы разработки ПО отличаются друг от друга тем, как этапы жизненного цикла программного обеспечения взаимосвязаны между собой внутри цикла разработки.

Зачастую, разработка программного обеспечения строится из следующих этапов:

1. Определение стратегии. Необходимо выяснить требования заказчика, оценить затраты на реализацию, подсчитать бюджет, определить возможности выполнения работ на взаимных для клиента условиях.
2. Анализ. На данном этапе происходит исследование функций, атрибутов и связей, определенных на стадии определения стратегии.
3. Проектирование. Выполняется построение модели разрабатываемой системы, определяются основные сущности и архитектура проекта.
4. Реализация. Этап непосредственной разработки продукта на основе спроектированной модели.
5. Тестирование. Процесс проверки разработанной системы на наличие ошибок.
6. Внедрение. Процедура передачи готовой системы заказчику. На данном этапе могут устраняться раннее необнаруженные ошибки.
7. Использование и техническая поддержка. После завершения разработки клиент нуждается в технической поддержке. Разработчик, по договоренности с клиентом, предоставляет поддержку на временной или постоянной основе.

Одной из самых распространенных и гибких методологий разработки ПО является Agile.

Agile – гибкая система разработки, в которой большие задачи разбиваются на небольшие этапы. После каждого этапа команда выдает готовую часть продукта на тестирование и оценку.

К основным принципам данной методологии относят:

1. Наивысшим приоритетом является удовлетворение потребностей клиентов за счет своевременной и непрерывной поставки ценного программного обеспечения;
2. Меняющиеся требования не является критичным фактором даже на поздних стадиях разработки. Гибкие процессы используются для конкурентного преимущества;
3. Частая доставка работающего программного обеспечения. От пары недель до пары месяцев, отдавая предпочтение более коротким временным рамкам;
4. Самый эффективный и действенный метод передачи информации команде разработчиков и внутри нее – беседа лицом к лицу;
5. Менеджеры и разработчики должны ежедневно работать вместе на протяжении всего проекта;
6. Работающее программное обеспечение является основным мерилом прогресса [4].

Принимая на вооружение данную методологию необходим инструмент, который будет соответствовать данным принципам и использоваться для организации работ. Одним из таких инструментов является система управления проектами Jira. Эта система является agile-инструментом управления проектами, который позволяет командам планировать, отслеживать, выпускать и поддерживать программное обеспечение. Это единый достоверный источник информации для всего жизненного цикла разработки, благодаря которому автономные команды получают доступ к контексту и могут быстро двигаться вперед, не теряя из виду более масштабную бизнес-цель. ...

## 1.3 Нормативы описания разработок

## 1.3.1 Формат описания разработок в системах контроля версий

Процесс внесения изменений в системы контроля версий сопровождается сообщением, в котором разработчик оставляет небольшой комментарий, описывающий внесенные изменения. По-умолчанию формат таких сообщений не регламентируется системами контроля версий, что позволяет разработчику оставить любой текст в качестве сообщения. Однако во время разработки может возникнуть проблема отслеживания изменений. Взглянув на текст сообщения не всегда возможно понять какие именно изменения были внесены. Разработчик попросту может написать неинформативный текст или вовсе оставить сообщение пустым. Во избежание данной ситуации в компании «Компас плюс» существует регламент описания изменений в системах контроля версий.

Рассматриваемый регламент описывает следующий формат сообщения:

[‘Тип изменения’] (‘Номер задачи’) ‘Название компонентов’. ‘Краткое описание’ <’Номер версии’> (‘Метка подтверждения’)

‘Дополнительное описание’

Приведем пример правильно оформленного текста описания изменения:

[#] (TXPG-2154) Order,Token. Исправлена ошибка оплаты с помощью карты <4.0.23> (QAD)

Была изменена логика работы метода execute класса Order.Payment. Добавлено вспомогательное поле tempSurcharge в классе Token.

Как можно наблюдать, данный формат обладает рядом параметров. Описание каждого из параметров представлено ниже:

1. Тип изменения. Описывает то, какой характер носит изменение. Может содержать одно из определенных в регламенте значений. Перечисление ...;
2. Номер задачи. Номер определенной задачи, в рамках которой велась работа по внесению описываемых изменений. Сам номер генерируется автоматически системой управления проектами Jira во время создания задачи;
3. Название компонентов. Имена компонентов, в которых были внесены изменения. Особенно актуально данное поле во время описания изменений в хостовых приложениях, разрабатываемых на платформе RadixWare, так как структура проекта обязательно содержит модули, определяемые во время проектировки приложения;
4. Краткое описание. Небольшой текст, вкратце описывающий внесенное изменение;
5. Номер версии. Необязательный параметр, которое заполняется при внесении изменений более чем в одну версию. Хранит номер самой старой версии, в котору вошло изменение (например, изменение вошло в версии 4.0.23, 4.0.24 и 4.0.25, тогда данный параметр будет иметь значение 4.0.23 т.к. это самая старая версия);
6. Метка подтверждения. Необязательный параметр. Имеет лишь одно значение ‘QAD’. Наличие данной метки означает, что внесение изменений согласовано с отделом тестирования;
7. Дополнительное описание. Необязательный параметр. Более развернутое описание вносимых изменений.

Как можно наблюдать, данный формат дает исчерпывающее представление о вносимом изменении. Данный формат является обязательным, каждое сообщение к изменению в системах контроля версий должно был описано именно в нём.

## 1.3.2 Регламенты описания задач в системах управления проектами

Текст

## 1.4 Контроль качества продукта

Как было указано раннее, процесс разработки программного обеспечения сопровождается такими этапами как тестирование и поддержка. В компании «Компас плюс» для этих процессов существуют специальные отделы название отделов. Работа данных отделов строится, в том числе, на анализе изменений, которые были внесены разработчиком. Сотрудник каждого из отделов имеет доступ к задаче, в рамках которой были внесены изменения. На основе этого и строится его работа.

Получив полную информацию об изменении сотрудник отдела тестирования может приступить к его проверке. Для него важно то как внесенное изменение влияет на поведение какого-либо функционала системы и какие версии продукта это затрагивает. Из задачи сотрудник отдела тестирования, может определить, например, правильные параметры запроса к системе, понять как она должна отреагировать на то или иное действие.

В случае несоответствия ожидаемого результата с полученным, тестировщик вправе создать задачу об ошибке, однако не всегда неожиданное поведение является ошибкой. В такой ситуации неожиданное поведение является лишь результатом неполноценного описания изменения разработчиком, поэтому очень важно грамотно описывать изменения как в системе контроля версий, так и в соответствующей задаче.

Работа сотрудника отдела документации схожа с работой, описанной выше. Очень важную информацию, в данном случае, несет тип внесенного изменения, так как не все то, что было внесено в разрабатываемый продукт подлежит документации. Это относится к внутренним изменениям в системе, которые не влияют на функциональность, а лишь оптимизируют работу системы. Для тестировщика в любом случае возникает необходимость проверки функционала на корректность работы, однако для документатора это означает что данное изменение не подлежит описанию.

Также, во время передачи новой версии программного продукта, клиенту передается и список изменений, который строится на основе сообщений в системах контроля версий. В данном случае, как и в случае с документацией, важную роль играет тип внесенного изменения. Не все типы изменений передаются конечному клиенту. Список передаваемых типов изменений описан ниже: ...

### 1.4.1 Тестирование

Текст

### 1.4.2 Документация

Текст

## 1.4 Выводы к первой главе

Текст

# 2 Проектные решения разработки модуля системы csia для мониторинга описания работ по разработке по в системах контроля версий и управления проектами

## 2.1 Общие сведения о система CSIA

### 2.1.1 Платформа RadixWare

Текст

### 2.1.2 Модули системы CSIA

Текст

## 2.2 Требования к разрабатываемому модулю

Текст

## 2.3 Организация взаимодействия с системами контроля версий

## 2.3.1 Взаимодействие с Subverion

Текст

## 2.3.2 Взаимодействие с GitTea

Текст

## 2.4 Организация взаимодействия с системой управления проектами Jira

Текст

## 2.5 Мониторинг нарушений в проекте

### 2.5.1 Структурная модель

Текст

### 2.5.2 Алгоритм поиска нарушений

Текст

## 2.6 Генерация отчета

### 2.6.1 Работа со статистикой нарушений

Текст

### 2.6.2 Алгоритм генерации отчета

Текст

## 2.7 Выводы ко второй главе

Текст

# 3 Результаты опытной эксплуатации модуля системы csia для мониторинга описания работ по разработке по в системах контроля версий и управления проектами

## 3.1 Работа с разработанными сущностями

### 3.1.1 Сущность проекта

Текст

### 3.1.2 Сущность репозитория

Текст

### 3.1.3 Сущности системы управления проектами

Текст

## 3.2 Создание и настройка задачи для мониторинга

### 3.2.1 Задача Task.CheckProjectViolations

Текст

### 3.2.2 Настройка расписания

## 3.3 Пример сгенерированного отчёта

Текст

## 3.4 Выводы к третьей главе

Текст

# Заключение

Текст

# Список используемых источников

1 - <https://vk.com/doc7608079_449808456?hash=TrY4ho4utnq5LpCaxFpuamI4W8paaZPTR1sjBm5taBz&dl=49oI2tsG9r5KEMwqABnsociXSnqZQzVK3UstJ5VR2U4>

# Приложение А