Требования к внешним командам разработки для совместного участия в проектах ПАО “ВымпелКом”



# Лист изменений

Дата

Версия

Описание

Автор

25/03/2021

0.1

Первичное описание требований (ориентация на

Java)

Тихомиров К..

# Общие требования к производственному процессу

1. Команда разработки представляет собой автономную команду, включающую в себя следующие минимальные роли:

Роль Зона ответственности

Владелец продукта обеспечивает единую точку входа в команду

Разработчик реализация бизнес-логики

Тестировщик интеграционные тесты

автоматизированное тестирование

DevOps-инженер процессы сборки и поставки кода

Оптимальный состав команды 6-7 человек. Команда разработки не должна быть более 10 человек.

Команда должна работать по производственному циклу с применением коротких итераций (1-3 недели разработки), включающих в себя демонстрации промежуточных результатов всем заинтересованным участникам продукта.

Рекомендуется реализовывать производственный цикл по методологии Agile, с использованием фреймворков SCRUM или Kanban

# Применяемые технологии, библиотеки и средства и архитектурные подходы

Разработка на основе микросервисов.

Микросервисы представляют из себя программные компоненты, реализующую одну бизнес-функцию (или “сильно-связанный” набор функций) и предоставляют свой API в виде HTTP/JSON (REST/RESTful) интерфейса.

Микросервисы могут быть реализованы с использованием различных языков программирования, но должны удовлетворять заявленным требованиям.

## Требования при реализации бизнес-логики на Java

* с использованием Java 11 / JAX-RS v2.0 / Jersey без привязки к конкретным серверам приложений (например, Java EE) или фреймворкам (например, Spring).
* Допустимо использовать библиотеки Apache Common Libraries, Google Guice/Guava и другие по дополнительным соглашениям.
* Не сохраняют сеансовую информацию между вызовами (являются stateless).
* Работают внутри легковесного контейнера Web-приложений Eclipse Jetty 11
* Рекомендуется легковесная замена для ORM - JOOQ - для работы с РСУБД

## Требования при реализации бизнес-логики на .Net

* Использование фреймворка .NET5 или новее
* Работа приложения внутри контейнеров Docker

## Оформление REST API

см. [Beeline REST API Conventions](https://confluence.veon.com/display/BRITEA/Beeline+REST+API+Conventions)

Допустимо использовать еще какие-либо дополнительные библиотеки по предварительному согласованию с представителями ПАО “ВымпелКом”.

Также предварительного согласования требует особые требования к окружению (операционные системы, применение JNI и др).

Типовыми ОС являются:

* OS Linux (OEL/CentOS/Ubuntu)
* OS Windows 2016 R2 и выше

## UI-интерфейсы для Web

Портал реализуется с использованием:

Семантический HTML5 / SCSS / JavaScript

Angular 11+ или React в виде основного фреймворка, по согласованию со специалистами Вымпелком.

Дизайн-системы Beeline с обязательным использование токенов Требования по поддержке браузеров:

Google Chrome 88+ (ориентируемся на актуальную версию в пределах одного квартала)

Mozilla Firefox 85+ (ориентируемся на актуальную версию в пределах одного квартала) IE 11 - Microsoft Edge

Приложение реализуется на базе дизайн-макетов представленных от UI/UX-дизайнера.

Макеты реализуются:

на базе дизайн-системы Beeline (Figma)

## Типы хранилищ данных для реализации

РСУБД:

PostgreSQL 12.1 или выше, MS SQL Server 2017 или выше с использованием ORM Dapper

in-memory: Aerospike Hazelcast колоночная СУБД ClickHouse хранение метрик и логов:

ElasticSearch

InfluxDB

Prometheus

Использование хранимых процедур для стандартных операций CRUD (Create, Read, Update, Delete) должно быть сведено к минимуму если нет дополнительных требований массовой обработки данных. Это требуется для снижения трудозатрат по созданию и миграции между различными РСУБД (PostgreSQL, Oracle).

## Типовые интеграции

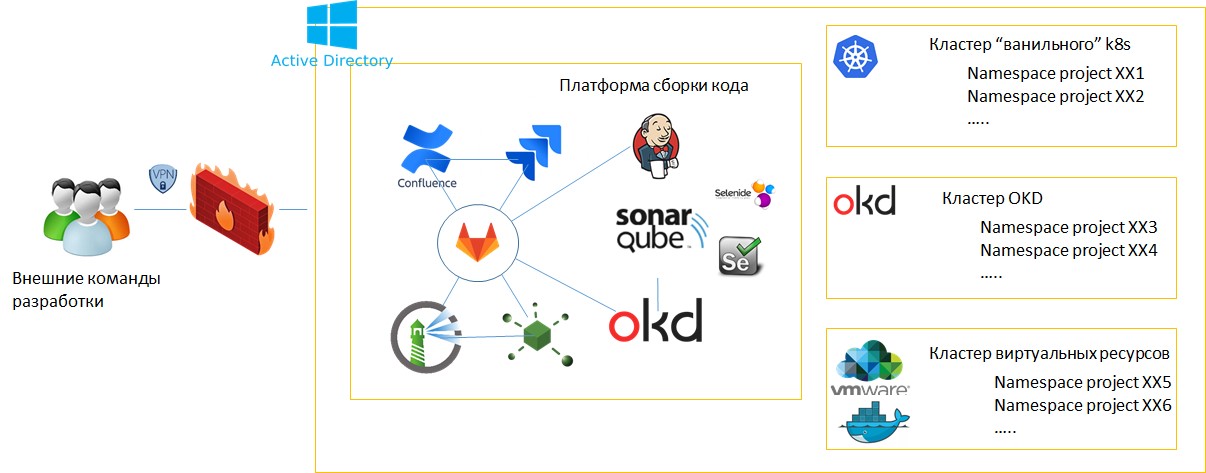
Для интеграционного слоя, чтобы не смешивать базовую логику и взаимодействие с внешними системами, предлагается использовать Apache Camel.

### Обязательные требования

Продуктовая команда, при сдаче проекта, обязана предоставить артефакта с указанием всех используемых компонентов, их версий и ссылок на лицензии.

# Инфраструктурная поддержка производственных процессов

Разработка программного обеспечения в ПАО "ВымпелКом" придерживается DevOps-методологии, а производственные процессы поддерживаются следующими основными системами:



* Системы хранения и управления исходным кодом (VCS)- обеспечение хранения исходного кода, истории изменений и ведения параллельных веток.
* Системы автосборки - сборка программных компонентов и их регистрация в корпоративных репозиториях.
* Системы хранения артефактов - обеспечение хранения артефактов сборки код и контейнеров.
* Система оценки качества исходного кода - автоматизированный статистический анализ кода, а также оценка покрытием кода Unit-тестами (Code Coverage)

## Практика Инфраструктура как код (Infrastructure as a code) и применение DevOps методологий

Команды разработки ПАО "ВымпелКом" и аутсорс-команды, в процессе разработки, должны придерживаются практики Infrastructure as a code (IaaC), позволяющей управлять инфраструктурой в виде кода и применять различные практики и инструменты из разработки, такие как версионирование, тестирование, ревью, CI/CD и т.д..

Применение практики IaaC позволяет управлять виртуальными машинами на программном уровне, что исключает необходимость ручной настройки и обновлений для отдельных компонентов оборудования. Все описание инфраструктуры должно храниться в целевом репозитории GitLab, связанным с разрабатываемым продуктом.

В качестве системы управления конфигурациями (software configuration management, SCM) в компании ПАО "ВымпелКом" наиболее широко применяется Ansible. Рекомендуем использовать следующие разделы:

Inventories - раздел для хранения и описания окружения продукта может содержать:

Хосты и группы;

Переменные хоста;

Групповые переменные;

Группы групп и групповые переменные и т.п.

Roles - раздел для хранения описания набора файлов, задач, шаблонов, переменных и обработчиков, которые вместе служат для настройки конкретного сервиса или службы.

### Структура каталога ролей

Пример структуры проекта:

site.yml webservers.yml fooservers.yml roles/ common/ tasks/ handlers/ files/ templates/ vars/ defaults/ meta/ webservers/ tasks/ defaults/ meta/

При использовании каждый каталог должен содержать main.yml - файл, содержащий соответствующий контент: tasks - содержит основной список задач, которые должна выполнить роль;

handlers - содержит обработчики, которые могут использоваться этой ролью или даже где-либо вне этой роли; defaults- переменные по умолчанию для роли; vars- специальные переменные для роли. files - файлы, которые можно развернуть используя данную роль; templates - содержит шаблоны, которые можно развернуть через эту роль. meta- определяет некоторые метаданные для этой роли.

## Хранение исходного кода

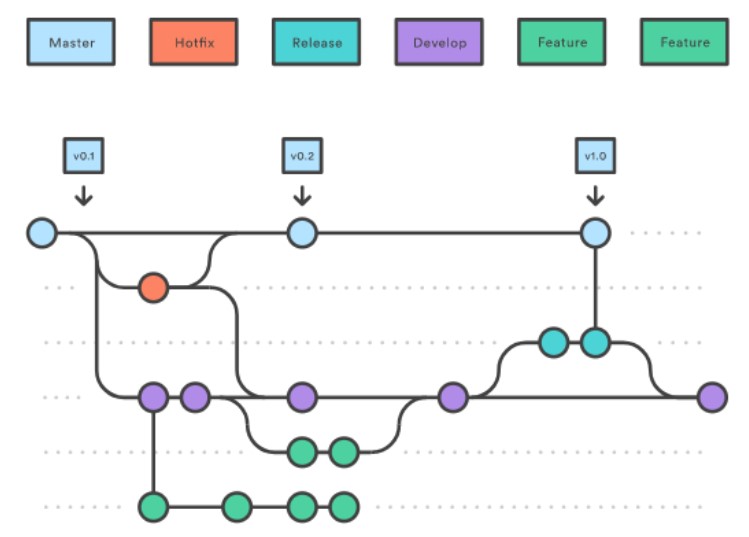
Для исходного кода и управления исходниками в компании используются несколько VCS систем:

GitLab - разработка коммерческих проектов на Java, Ruby, Scala, C++ -based технологиях.

Microsoft TFS - интегрированная среда для разработки коммерческих проектов на Microsoft/.NET-based технологиях (не целевое решение).

## Модель ветвления и версионирования для GitLab

Модель ветвления Gitflow



Ветка master является веткой, где ведутся последние разработки. В рамках нее производится разработка и merging из других веток (feature branches). Для нее настраивается проект в Jenkins, который в постоянном режиме собирает master builds, запускает Unit/Integration Tests.

Для данной ветки определяются ответственные за прием (merge) изменений и ограничивается доступ для остальных разработчиков.

Ветка develop служит в качестве интеграционной ветки для разработки новой функциональности и хранения полной истории проекта

Каждая новая функциональность должна разрабатываться в отдельной ветке, которую можно отправлять в центральный репозиторий для создания резервной копии/для совместной работы команды. Ветки функций (feature branches), создаются на основе develop. После завершения работ над функцией создается Pull Request (PR) (в некоторых системах называется еще как Merge Request), который содержит краткое описание проделанной работы и осуществляется запрос на внесение изменение (merge) в ветку develop. Новый код не должен отправляться напрямую в master.

Используется подход для работы описанный как Github Flow. После завершения работ над данной функцией и закрытия PR созданную ветку нужно удалять. Детальный процесс рассмотрения PR каждая команда определяет самостоятельно, но необходимо чтобы в командах была роль, которая отвечает за ветку.

При окончании разработки задекларированной функциональности для нового, от ветки develop создается ветка release. Создание данной ветки означает начало следующего цикла релиза, в ходе которой новая функциональность уже не добавляется, а производится только отладка багов, создание документации и решение других задач, связанных с релизом. После успешного окончания тестирования релиз-кандидата, ветка release сливается в master, и ей присваивается тег с версией. Кроме этого, она должна быть также слита обратно в ветку develop, в которой с момента создания ветки релиза могли добавляться изменения с момента создания ветки релиза.

Ветки hotfix используются для быстрого внесения исправлений в рабочую версию кода. Это единственная ветка, которая должна быть создана непосредственно от master. Как только исправление завершено, ветка hotfix должна быть объединена как с master, так и с develop (или с веткой текущего релиза), а master должен быть помечен обновленным номером версии.

В рамках ветки используются tags для отметки release или milestone builds (v1.0, v1.1-SNAPSHOT,v1.1-M1-SNAPSHOT, и т.д.). Соответственно процесс выпуска нового релиза или milestone системы является определение для него соответствующего тега без создания отдельного branch и для них определены соответствующие Jenkins Jobs.

Если для компонентов существует интеграционный стенд тестирования или предполагаются промежуточные релизы для ручного тестирования, то master некоторые билды помечаются как milestone builds соответствующим тегом. В версии компонентов добавляется суффикс M[nn]. Итого полная версия компонента в ветке milestone представляет собой vN[.K]-M[nn]-SNAPSHOT.

Примеры: v1.0-M1-SNAPSHOT, v2.0.1-M2-SNAPSHOT. Периодичность и необходимость обновления цифр nn в суффиксе M[nn] определяется самой командой.

## Сборка компонентов

Любой выпускаемый программный компонент обязан собираться корпоративной системой автосборки. Компонент должен собираться автоматически после внесения любых изменений в репозитории исходного кода.

Сборка подразумевает собой:

* Скачивание исходного кода из репозитория исходного кода.
* Генерацию всех необходимых артефактов для компиляции.
* Компиляцию исходного кода и сгенерированных файлов в бинарную сборку:
* JAR/WAR (для Java приложений).
* ZIP - для скриптов РСУБД.
* Возможны и другие результируюшие типы файлов для специализированных типов проектов.
* Присвоение номера сборки (build number).
* Регистрацию собранного бинарного артефакта в общем репозитории.

Для сборки компонентов рекомендуется использовать:

* Maven для бизнес-логики написанной на Java, C#, Ruby, Scala,
* Gradle для бизнес-логики написанной на Java, Grovy, Scala,
* CMake для бизнес-логики написанной на C++,
* Может также использоваться Gulp - для сборки компонентов на базе JavaScript.

Сборка должна осуществляться без требований использования IDE, специальных утилит или проприетарных плагинов.

## Хранение артефактов проекта

### Sonatype Nexus

Используется для хранения бинарных артефактов продукта и кеширования артефактов, загруженных из интернета.

Поддерживаемые артефакты :

Java, Maven (jar);

Python (pip);

Ruby (gem);

NPM;

Bower; Yum (rpm);

gitlfs;

Raw;

Apt (deb);

Go;

Nuget.

Harbor

Используется для хранения и управления образами Docker-контейнеров в безопасном окружении.

## Database as a Code

Скрипты установка баз данных, внесения изменений в структуру или миграции данных должны:

храниться в корпоративной VCS системе (subrepository продукта); применяться в конвейере поставки CI/CD; проверяться на тестовых средах и соответствовать концепции идемпотентности.

Рекомендуемые инструменты для автоматизации поставки и рефакторинга баз данных - Flyway, Liquibase.

# Тестирование и управление качеством создаваемых компонент

## Continuous Quality as a Code

Команды разработки ПАО "ВымпелКом" и аутсорс-команды, в процессе разработки, должны придерживаются практики непрерывного контроля качества разрабатываемого кода (постоянное измерение, улучшение и автоматизации проверки качества кода).

Создание разработчиками модульных тестов (unit-тестрование) и их выполнение в процессе сборки кода является обязательным требованием компании.

В качестве системы проведения статического анализа качества кода в компании используется SonarQube (<https://docs.sonarqube.org/latest/>).

Правила контроля и критерии измерения качества программного кода в проекте устанавливаются по предварительному согласованию с представителями ПАО “ВымпелКом”.

## Обеспечение качества продукта и автоматизация тестирования

Для проведения тестирования в команде должен быть выделен член команды, проводящий требуемые виды тестирования:

Ручное функциональное тестирование проводится в процессе реализации функциональности в рамках спринта разработки.

Автоматизированное интеграционное тестирование с точки зрения “серого ящика” (gray box). Как минимум на уровне smoke-тестирования базовых функций системы:

* Для Web написание тестов на базе Selenium + JUnit/TestNG
* Для бизнес-логики Java Рекомендуется использовать фреймворки:

JUnit

TestNG

Rest-Assured

Cucumber

Allure

Автоматизированное регрессионное тестирование при выполнении сборки релиза.

Исходный код компонентов проходит автоматизированное тестирование в ПАО "ВымпелКом" на предмет соответствия качеству, стилевому оформлению (статистический анализ) и полноты покрытия кода тестами (запуск unit-тестов и анализ выполнения).

# Документирование

Разработчики при создании программных интерфейсов обязаны документировать на русском и/или английском языке (в зависимости от продукта и внешних требований):

1. Все публичные интерфейсы и методы для языка Java документируются с помощью JavaDoc
2. WSDL или XSD модели - документируются все операции, все типы и элементы в самих файлах.
3. REST интерфейсы описываются с применением OpenAPI Specification (Swagger RESTful API Documentation Specification) - [http://swagger.io /specification/](http://swagger.io/specification/).
4. Таблицы, колонки в РСУБД - используются средства SQL РСУБД для хранения текстовых описаний.

# Диагностика и конфигурация компонентов

## Журналирование событий

Все программные компоненты обязаны писать диагностические события в файл.

События необходимо разделять по уровню критичности. Системный журнал событий должен писаться на одном языке - английском.

В журналах запрещено использовать:

* Просторечивые слова
* Слова из сленга или ненормативную лексику
* Не допускается вывод ошибок, сообщений в стандартный вывод (в случаях, если это не консольная утилита).
* Не допускается вывод влогах учётных данных и токенов аутентификации (шифруйте или маскируйте конфиденциальные данные).

Рекомендации по выбор уровней записи при журналировании:

* ERROR - возникновение непоправимой ошибки, требующей немедленной реакции. Индикатор того, что система может работать некорректно.

На этом уровне могут могут быть предоставлены ошибки: недоступности БД, определения сервисов, невозможности корректной инициализации

* WARN - возникновение ситуации, при которой работа может быть продолжена, но есть ограничения. Индикатор того, что была обнаружена проблема, для которой предусмотрена программная обработка. на этом уровне могут журналироваться ошибки: запуск системы в Read-only или Single-mode, недоступность системы и использование кэшированных данных, неуспешная попытка аутентификации или авторизации и т.п.
* INFO - фиксация важного системного события или шага исполняющегося бизнес-процесса. Также может быть использовано для фиксации важных изменений в состоянии сервисов и приложений.
* DEBUG - детальная информация для разработчиков. Сообщения на этом уровне будут отключены на продуктивной среде и доступны на Development или Integration.

Используйте временные метки.

Временные метки позволяют понять последовательность событий, они необходимы для отладки и анализа. При фиксировании времени рекомендуется использовать наиболее подробные значения (например, на уровне милли- или микросекунд). Временные метки должны стоят в начале сообщения в формате yyyy-mm-dd HH:mm:ss. Всегда указывайте часовой пояс, если не используете на сервере время по умолчанию (UTC).

Используйте в сообщении отслеживаемый идентификатор или токен сообщения.

Присваивайте уникальный уникальный идентификатор события при попадании в систему. Старайтесь идентификатор передавать на другие этапы обработки, для отслеживания движение события по системе, это полезно для отладки, конкурентной обработки и операций, связанных с данными. В идеале, в системе должен быть уникальный идентификатор события в рамках всех модулей и сервисов.

Указывайте соответствие идентификаторов в точках перехода.

При передаче события из одной системы в другую, идентификаторы меняются в соответствии с принятым в другой системе соглашением. В таких точках перехода необходимо отдельным сообщением явно указывать соответствие старого и нового идентификатора, с добавлением необходимого контекста.

Предоставляйте достаточный контекст для ошибок и сбоев

Предоставляйте в случае сбоя максимально полную информацию о возникновении ошибки (желателен технологический и бизнес-контекст). Если запрос или сообщение не удалось обработать, то в дополнение к ошибке очень полезно журналировать и тело сбойного запроса.

Фреймворки для журналирования:

JavaScript/TypeScript: Log4js / pino Java: Log4j

## Конфигурация приложений

1. Программные компоненты, настраиваемые из файла или из БД, при отсутствии конфигурационных файлов или параметров (опциональных или обязательных), неправильности их задания должны корректно об этом сообщать администраторам системы (писать в журнал, показывать на консоли или в Web-интерфейсах). Необходимо исключить падения с системными исключениями без возможности понять истинную причину в данных случаях ("популярный NPE").
2. Компоненты требующие файлы настроек, должны поставляться с примерами настройки и указанными параметрами по умолчанию. Файл настроек должен автоматически быть создан при отсутствии его в положенном месте.