

Оглавление

1. Введение	4
2. Предпроектное исследование	6
2.1. Постановка задачи	7
3. Концептуальный этап проектирования	9
3.1. Выбор общесистемной методологии проектирования	9
3.2. Диаграмма пакетов	10
3.3. Выделение системы из среды	11
4. Формирование технического задания	13
4.1. Введение.....	13
4.2. Общие сведения.....	13
4.3. Назначение разработки.....	13
4.4. Требования к программе или программному изделию	13
4.5. Стадии и этапы разработки	15
4.6. Порядок контроля и приемки.....	15
5. Технический этап проектирования.....	16
5.1. Разработка топологии системы.....	16
5.2. Разработка диаграммы развертывания	17
6. Рабочий этап проектирование	19
6.1. Архитектура скрипта автоматической сборки, тестирования и развертывания.....	19
6.2. Диаграмма активности системы автоматизированной сборки, тестирования и развертывания	21
7. Апробирование разработанной системы для модельных условий	23
8. Заключение	24
Список литературы	25
Приложение 1. Код maven скрипта для подготовки архива	26
Приложение 2. Инструкция по сборке с помощью maven	30

1. Введение

Имитационное моделирование (ИМ) на ЭВМ находит широкое применение при исследовании и управлении сложными дискретными системами (СДС) и процессами, в них протекающими. К таким системам можно отнести экономические и производственные объекты, транспортные системы (морские порты, аэропорты) и комплексы перекачки нефти и газа, программное обеспечение сложных систем управления и вычислительные сети, а также многие другие.

Широкое использование ИМ объясняется тем, что размерность решаемых задач и неформализуемость сложных систем не позволяют использовать строгие методы оптимизации. Эти классы задач определяются тем, что при их решении необходимо одновременно учитывать факторы неопределенности, динамическую взаимную обусловленность текущих решений и последующих событий, комплексную взаимозависимость между управляемыми переменными исследуемой системы, а часто и строго дискретную и четко определенную последовательность интервалов времени. Указанные особенности свойственны всем сложным системам.

Проведение имитационного эксперимента позволяет:

1. Сделать выводы о поведении СДС и ее особенностях:
 - без ее построения, если это проектируемая система
 - без вмешательства в ее функционирование, если это действующая система, проведение экспериментов над которой или слишком дорого, или небезопасно
 - без ее разрушения, если цель эксперимента состоит в определении пределов воздействия на систему
2. Синтезировать и исследовать стратегии управления
3. Прогнозировать и планировать функционирование системы в будущем
4. Обучать и тренировать управленческий персонал и т.д.

Разработка интеллектуальной среды имитационного моделирования РДО выполнена в Московском Государственном Техническом Университете (МГТУ им.Н.Э. Баумана) на кафедре "Компьютерные системы автоматизации производства". Причинами ее проведения и создания РДО явились требования универсальности ИМ относительно классов моделируемых систем и процессов, легкости модификации моделей,

моделирования сложных систем управления совместно с управляемым объектом (включая использование ИМ в управлении в реальном масштабе времени) и ряд других, сформировавшихся у разработчиков при выполнении работ, связанных с системным анализом и организационным управлением сложными системами различной природы.

В настоящий момент РДО обретает второе рождение под именем Rao X. Благодаря переносу исходного кода с C++ на Java с использованием библиотеки Xtext, достигнут качественный рост производительности и удобства использования среды. Вместе с переходом на новый язык программирования необходимо заново разработать систему автоматизированной сборки, тестирования и развертывания Rao X.

2. Предпроектное исследование

Система Rao X используется в проведении лабораторных работ по курсу Моделирование технологических и производственных процессов, поэтому предполагается, что студентам должно быть легко скачать систему для домашней подготовки.

Для выпуска новой версии необходимо выполнить следующую инструкцию:

- Установить Java 8

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

If output is different than (except version numbers):

- Скачать Eclipse IDE for Java and DSL Developers

```
cd ~/Downloads
gunzip -c eclipse-dsl-luna-SR2-linux-gtk-x86_64.tar.gz | tar xvf -
cd eclipse
./eclipse
```

- Выкачать репозиторий raox

```
ssh-add ~/.ssh/github.openssh.private.key
git clone git@github.com:aurusov/raox.git
```

Или скачать изменения

```
git fetch
```

- Загрузить проект в Eclipse

```
File > Import > General > Existing Projects into Workspace > Select root
directory > /home/USERNAME/git/raox > Finish
```

- Запустить генерацию артефактов Xtext

```
ru.bmstu.rk9.rao/src/ru.bmstu.rk9.rao/Rao.xtext > Run As > Generate Xtext Artifacts > Proceed
```

- Дождаться компиляции проекта
- Выполнить экспорт/собрать .jar артефакты проекта(плагины)

```
File > Export > Plug-in Development > Deployable plug-ins nad fragments > Next >
Select All > Finish
```

- Положить .плагины в папку eclipse/dropins и сделать архив
- Выложить подготовленный архив в открытый доступ

Эту процедуру необходимо выполнять всякий раз, когда надо подготовить релиз. К слову в Rao X было внесено более 1250 изменений, решено 115 различных задач и выпущено 20 номерных релизов.

Поэтому было решено разработать систему автоматизированной сборки, тестирования и развертывания.

2.1. Постановка задачи

Проектирование любой системы начинается с выявления проблемы, для которой она создается. Под проблемой понимается несовпадение характеристик состояния систем, существующей и желаемой.

В результате предпроектного исследования была выявлена необходимость в разработке системы автоматизированной сборки, тестирования и развертывания Rao X, которая предназначена для того чтобы автоматизировать сборку артефактов Rao X, подготовку архива с Eclipse и выкладывание полученного архива на сайт.

Разрабатываемая система также даст возможность тестировать Rao X. Тестирование – самая популярная методика повышения качества, подкрепленная многими исследованиями и богатым опытом разработки коммерческих приложений. Существует множество видов тестирования: одни обычно выполняют сами разработчики, другие – специализированные группы по контролю качества программного обеспечения. Стив Макконнелл выделяет следующие виды тестирования:

- **Тестирование компонента** – это тестирование класса, пакета, небольшого приложения или другого элемента системы, разработанного несколькими программистами или группами, выполняемое в изоляции от остальных частей системы.
- **Интеграционное тестирование** – это совместное выполнение двух или более классов, пакетов, компонентов или подсистем, созданных несколькими программистами или группами. Этот вид тестирования обычно начинают проводить, как только созданы два класса, которые можно протестировать, и продолжают до завершения работы над системой.
- **Регрессивным тестированием** называют повторное выполнение тестов, направленное на обнаружение дефектов в программе, уже прошедшей набор тестов.
- **Тестирование системы** – это выполнение ПО в его окончательной конфигурации, интегрированного с другими программными и аппаратными системами. Предметом тестирования в этом случае является безопасность, производительность, утечка ресурсов, проблемы синхронизации и прочие аспекты, которые невозможно протестировать на более низких уровнях интеграции.

На данном этапе развития Rao X, наиболее важным является «тестирование компонента» или так называемые Unit тесты. Окончательно получим требования к разрабатываемой системе

Автоматизированное выкачивание исходного кода

Автоматизированная компиляция и экспорт .jar артефактов

Автоматизированное тестирование компонентов (Unit тестирование)

Автоматизированная подготовка архива, а именно подстановка плагинов в папку eclipse/dropins и последующая архивация

Автоматизированное развертывание артефактов в репозитории

Автоматизированное развертывание архива на сайте

3. Концептуальный этап проектирования

3.1. Выбор общесистемной методологии проектирования

Задача, поставленная на этапе предпроектного обеспечения, может быть решена на основе следующих концепций:

- Модульность
- Объектная ориентированность

Модульность — это свойство системы, связанное с возможностью ее декомпозиции на ряд внутренне связанных между собой модулей. Применительно к конструированию технических систем модульность — принцип, согласно которому функционально связанные части группируются в законченные узлы — модули. В свою очередь модульность в программировании — принцип, согласно которому программное средство (ПС) разделяется на отдельные именованные сущности, называемые модулями. Модульность часто является средством упрощения задачи проектирования ПС и распределения процесса разработки ПС между группами разработчиков. При разбиении ПС на модули для каждого модуля указывается реализуемая им функциональность, а также связи с другими модулями.

Объектно-ориентированное программирование (ООП) — парадигма программирования, в которой основными концепциями являются понятия объектов и классов. Объект — это сущность, которой можно посылать сообщения, и которая может на них реагировать, используя свои данные. Объект — это экземпляр класса. Данные объекта скрыты от остальной программы. Скрытие данных называется инкапсуляцией.

Наличие инкапсуляции достаточно для объектности языка программирования, но ещё не означает его объектной ориентированности — для этого требуется наличие наследования.

Но даже наличие инкапсуляции и наследования не делает язык программирования в полной мере объектным с точки зрения ООП. Основные преимущества ООП проявляются только в том случае, когда в языке программирования реализован полиморфизм; то есть возможность объектов с одинаковой спецификацией иметь различную реализацию.

Выбранная модель проектирования, позволила производить разработку поэтапно и разделить разработку плагина и подсистемы, необходимой для его (и других плагинов) загрузки.

Как итог, необходимо разработать модуль реализующий интерфейс, необходимый для загрузки и который является связующим звеном загружаемого расширения и системы в целом. Интерфейс был разработан вместе с подсистемой загрузки плагинов.

Интерфейс определяет границу взаимодействия между классами или компонентами, специфицируя определенную абстракцию, которую осуществляет реализующая сторона. В отличие от концепции интерфейсов во многих других областях, интерфейс в ООП является строго формализованным элементом объектно-ориентированного языка и в качестве семантической конструкции широко используется кодом программы.

3.2. Диаграмма пакетов

На Рис. 1 представлена упрощенная диаграмма пакетов системы Rao X, на которой указаны основные зависимости от сторонних библиотек.

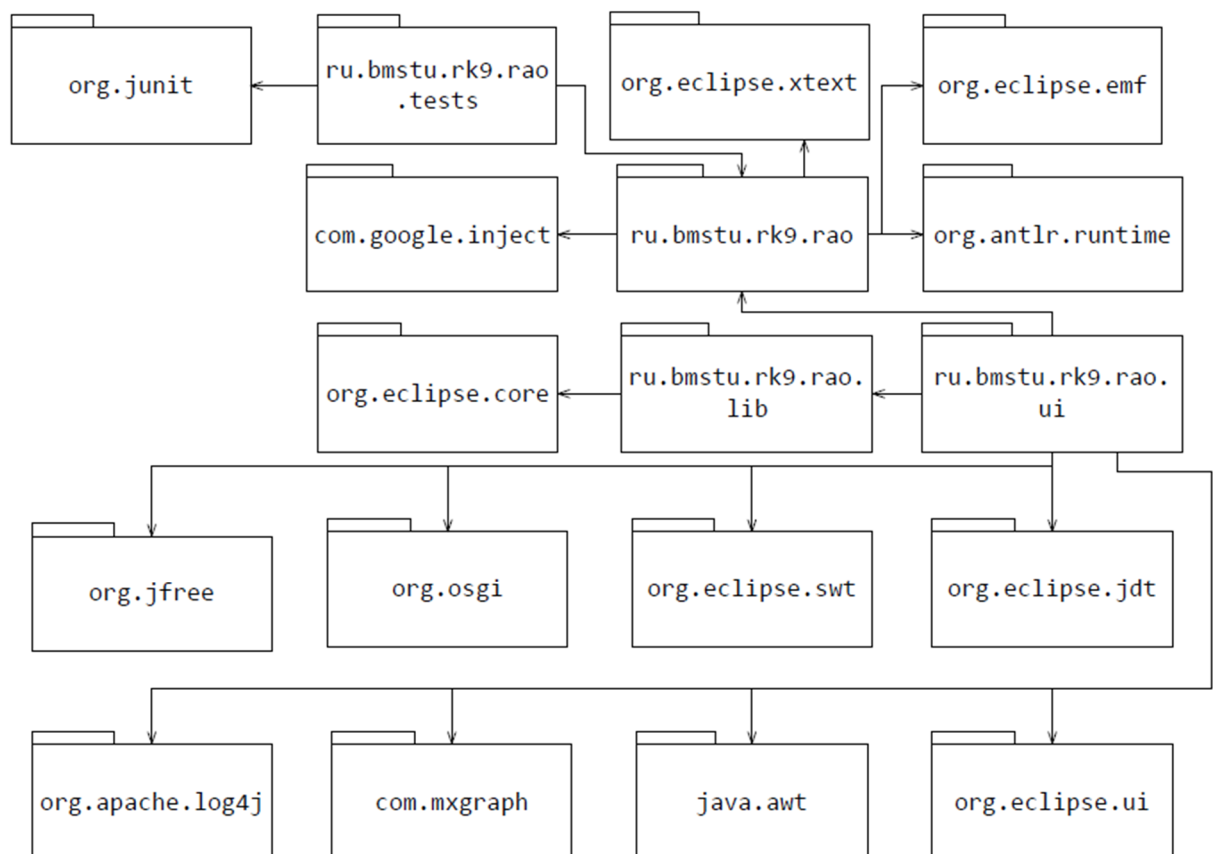


Рис. 1

ru.bmstu.rk9.rao – Пакет, содержащий модули, необходимые для работы реализации языка РДО

ru.bmstu.rk9.rao.ui – Пакет, содержащий модули, отвечающие за пользовательский интерфейс

ru.bmstu.rk9.rao.lib – Пакет, содержащий модули, необходимые для работы имитационных моделей

ru.bmstu.rk9.rao.tests – Пакет, содержащий модули, необходимые для тестирования

Остальные представленные на рисунке диаграммы – сторонние, это необходимо учитывать при разработке системы автосборки. И реализовать следующие зависимости:

org.junit

com.google.inject

org.antlr.runtime

org.apache.log4j

org.eclipse.emf

org.eclipse.xtext

org.eclipse.core

java.awt

org.eclipse.jdt

org.osgi

org.jfree

com.mxgraph

org.eclipse.ui

org.eclipse.swt

3.3. Выделение системы из среды

Рассмотрим какими ресурсами потенциально обладает система автоматизированной сборки, тестирования и развертывания. На Рис. 2 изображена топология располагаемых ресурсов.

Исходный код хранится на github - крупнейший¹ веб-сервис для хостинга IT-проектов и их совместной разработки. Основан на системе контроля версий Git.

Собранные архивы с eclipse и плагинами размещается на кафедральном сервере РК9 и доступен для скачки через официальный сайт Rao X(raox.ru).

¹ GitHub Dominates the Forges (<https://github.com/blog/865-github-dominates-the-forges>)

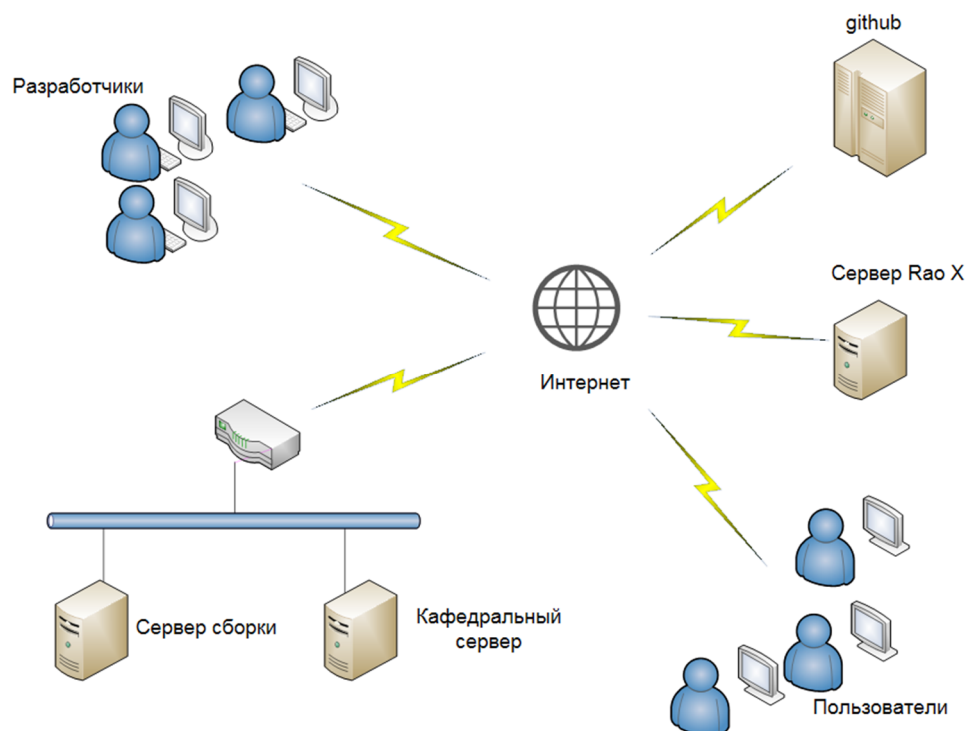


Рис. 2

Кроме того в распоряжении имеется сервер сборки с установленным на нем Jenkins – инструмент непрерывной интеграции, написанный на Java. Этот сервер объединен в локальную сеть с кафедральным сервером и может быть использован для автоматизированной сборки.

4. Формирование технического задания

4.1. Введение

Программный комплекс Rao X предназначен для разработки и отладки имитационных моделей на языке РДО.

Система автоматизированной сборки, тестирования и развертывания предназначена для облегчения выхода релиза, а так же повышения качества программного продукта вследствие добавления тестирования.

4.2. Общие сведения

- Полное наименование темы разработки: «Разработки системы автоматизированной сборки, тестирования и развёртывания»
- Заказчик: Кафедра «Компьютерные системы автоматизации производства " МГТУ им. Н.Э.Баумана»
- Разработчик: студент кафедры «Компьютерные системы автоматизации производства» Чернов А.О.
- Основание для разработки: Задание на курсовой проект
- Плановые сроки начала работы: 5 сентября 2015г.
- Плановые сроки окончания работы по созданию системы: 25 декабря 2015г.

4.3. Назначение разработки

Функциональным назначением объекта разработки является предоставление возможности подготавливать готовый архив с программным обеспечением Rao X, доступный для скачивания на сайте gaox.ru в автоматизированном режиме

4.4. Требования к программе или программному изделию

Требования к функциональным характеристикам:

Система должна реализовывать:

- Автоматическое выкачивание исходного кода
- Автоматическая компиляция и экспорт .jar артефактов
- Автоматическое тестирование компонентов (Unit тестирование)
- Автоматическая подготовка архива, а именно подстановка плагинов в папку eclipse/dropins и последующая архивация
- Автоматическое развертывание артефактов в репозитории

- Автоматическое развертывание архива на сайте

Требования к надежности:

Основное требование к надежности направлено на поддержание в исправном и работоспособном серверов, которые относятся к системе автоматизированной сборки, тестирования и развертывания Rao X.

Условия эксплуатации:

- Эксплуатация должна производиться на оборудовании, отвечающем требованиями к составу и параметрам технических средств, и с применением программных средств, отвечающим требованиям к программной совместимости
- Аппаратные средства должны эксплуатироваться в помещениях с выделенной розеточной электросетью 220В $\pm 10\%$, 50 Гц с защитным заземлением

Требования к составу и параметрам технических средств:

Программный продукт должен работать на компьютерах со следующими характеристиками:

- объем ОЗУ не менее 1024 Мб
- микропроцессор с тактовой частотой не менее 1600 МГц
- требуемое свободное место на жестком диске – 40 Гб

Требования к информационной и программной совместимости:

- операционная система Windows Server 2003 и старше или Ubuntu 15.10 и старше²
- наличие в операционной системе ПО для сборки maven и работы с системой контроля версий git

Требования к маркировке и упаковке:

Не предъявляются.

Требования к транспортированию и хранению:

Не предъявляются.

² Microsoft, Windows являются зарегистрированными торговыми марками или торговыми марками Microsoft Corporation (в США и/или других странах).

Ubuntu является зарегистрированной торговой маркой Canonical Ltd.

Названия реальных компаний и продуктов, упомянутых в данной пояснительной записке, могут быть торговыми марками соответствующих владельцев.

4.5. Стадии и этапы разработки

Разработка должна быть проведена в три стадии:

- техническое задание
- технический и рабочий проекты
- внедрение

На стадии «Техническое задание» должен быть выполнен этап разработки и согласования настоящего технического задания.

На стадии «Технический и рабочий проект» должны быть выполнены перечисленные ниже этапы работ:

- разработка системы
- разработка методики тестирования
- разработка программной документации
- испытания системы

4.6. Порядок контроля и приемки

Контроль и приемка работоспособности системы автоматизированной сборки, тестирования и развертывания должны осуществляться в процессе проверки функциональности (апробирования) системы в целом, а также в процессе проверки функциональности (апробирования) полученной в результате его работы системы имитационного моделирования Rao X путем многократных тестов в соответствии с требованиями к функциональным характеристикам системы.

5. Технический этап проектирования

Как уже было сказано ранее на этапе концептуального проектирования необходимо разработать систему автоматизированной сборки, тестирования и развертывания на имеющихся ресурсах. В рамках технического этапа проектирования была разработана топология разрабатываемой системы, а также диаграмма развертывания, уточняющая функции элементов системы.

5.1. Разработка топологии системы

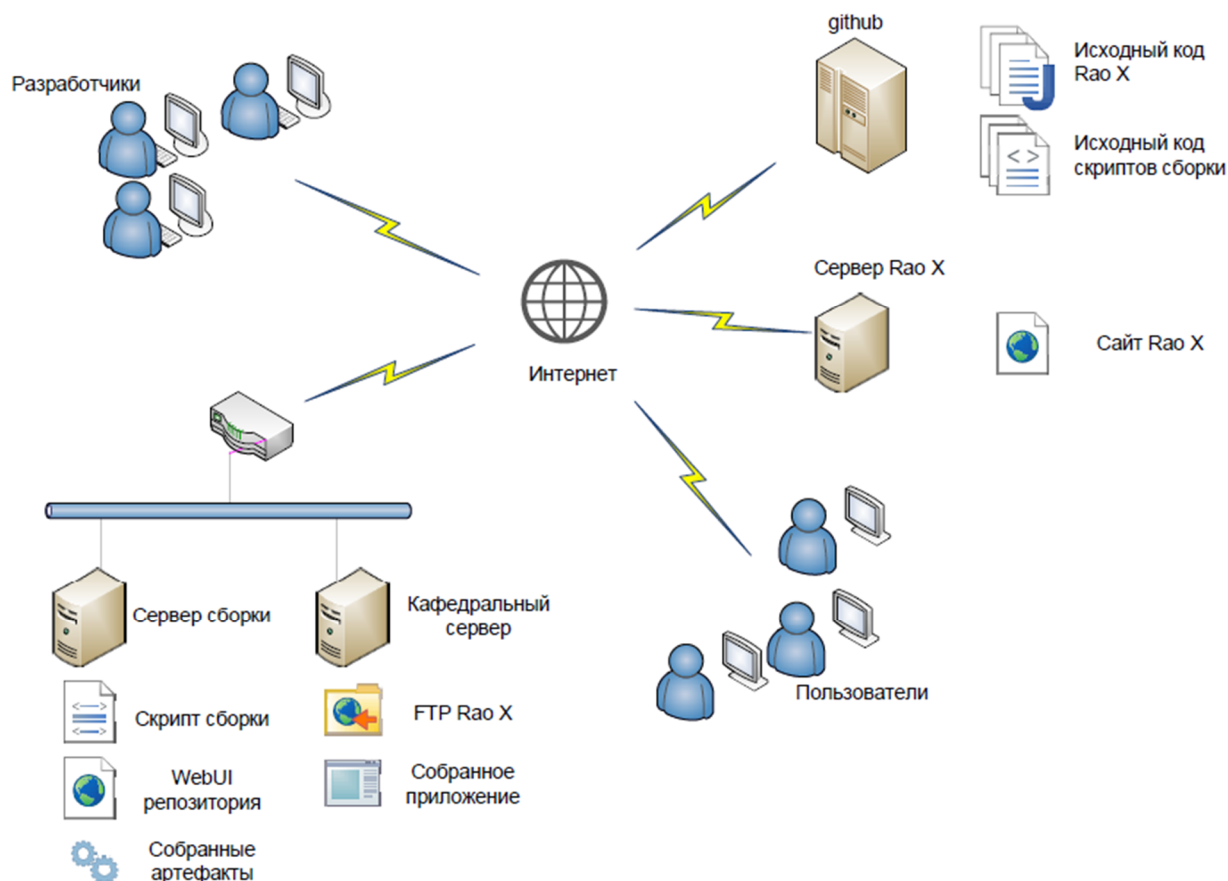


Рис. 3

На Рис. 3 изображена уточнённая топология разрабатываемой системы.

Сервер сборки – отвечает за автоматизированную сборку Rao X. На этой же машине физически располагается публичный репозиторий собранных артефактов, необходимый для сторонних разработчиков плагинов к системе Rao X, а так же позволяющий функционировать системе развертывания ли использовать один и тот же скрипт на любых машинах даже без подключения по локальной сети.

Кафедральный сервер – отвечает за доступ в интернет. На этой машине работает nginx – веб-сервер и прокси-сервер. Так же этот компьютер используется в качестве FTP-сервера, на котором хранятся готовые архивы с Rao X, которые доступны для скачивания через сайт **raox.ru**, который функционирует на отдельном **сервере Rao X**. Такая топология в купе с хранением до 5 версий программы, дает необходимое сохранение работоспособности системы при отказе по любым причинам любой подсистемы или её части.

5.2. Разработка диаграммы развертывания

На **Ошибка! Источник ссылки не найден.** изображена диаграмма развертывания системы автоматизированной сборки, тестирования и развёртывания Rao X, которая в том числе уточняет топологию

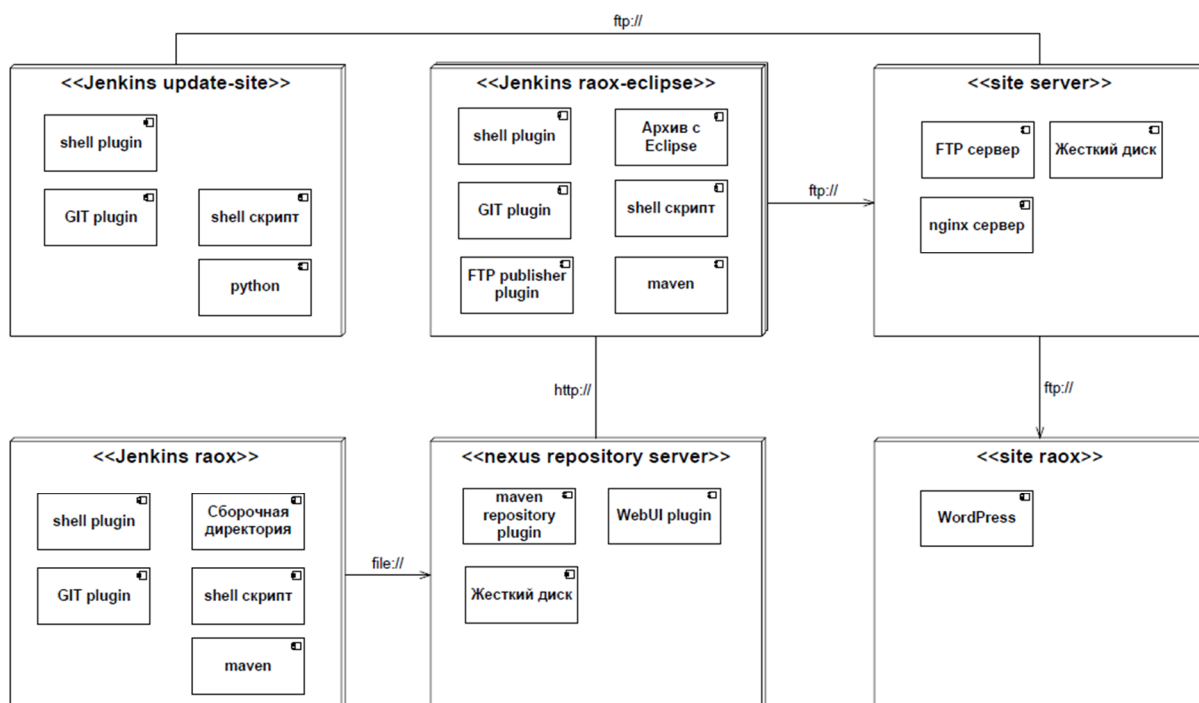


Рис. 4

Как уже было сказано ране на сервере сборки настроена система непрерывной интеграции Jenkins. В этой системе настроено 3 отдельные задачи:

Jenkins raox – отвечает за сборку и тестирование .jar артефактов с помощью maven и развертывание их на **nexus repository server**.

nexus repository server – публичный сервер-репозиторий Sonatype Nexus. Так как для сборки используется maven, необходимо использовать

соответствующий плагин, для хранения .jar артефактов, развёртываемых с помощью maven.

Jenkins raox-eclipse – отвечает за подготовку и сжатие архива с eclipse и плагинами, которые выкачиваются из репозитория. Готовый архив развёртывается на FTP-сервере, размещенном на компьютере **site-server**

Jenkins update-site – отвечает за запуск скрипта, который обновляет интернет страницу на сервере **site raox** в соответствие со списком файлов на **site-server**.

6. Рабочий этап проектирование

6.1. Архитектура скрипта автоматической сборки, тестирования и развертывания

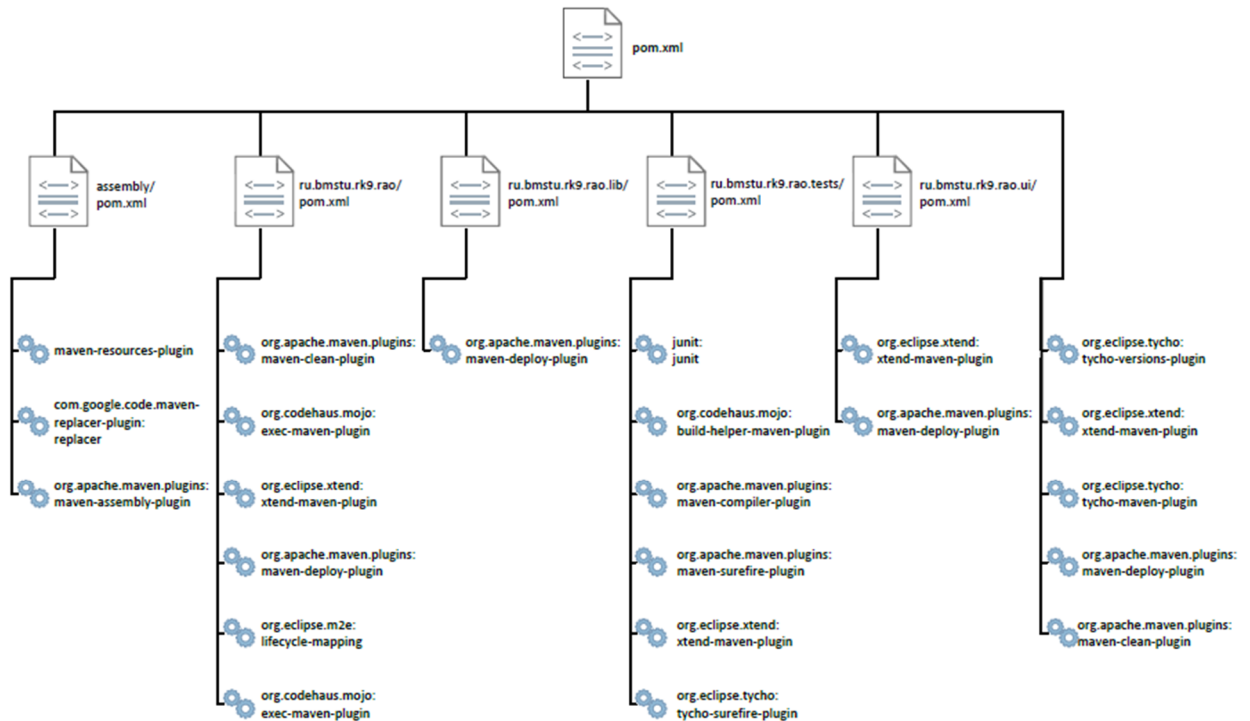


Рис. 5

На Рис. 5 представлена архитектура скрипта автосборки. Список используемых в maven сборке плагинов:

- `maven-resources-plugin`
- `com.google.code.mavenreplacer-plugin:replacer`
- `org.apache.maven.plugins:maven-assembly-plugin`
- `org.apache.maven.plugins:maven-clean-plugin`
- `org.codehaus.mojo:exec-maven-plugin`
- `org.eclipse.xtend:xtend-maven-plugin`
- `org.eclipse.m2e:lifecycle-mapping`
- `junit:junit`
- `org.codehaus.mojo:build-helper-maven-plugin`
- `org.apache.maven.plugins:maven-surefire-plugin`
- `org.apache.maven.plugins:maven-compiler-plugin`
- `org.eclipse.tycho:tycho-surefire-plugin`
- `org.apache.maven.plugins:maven-deploy-plugin`
- `org.eclipse.tycho:tycho-versions-plugin`

- `org.eclipse.tycho:tycho-maven-plugin`

Листинг скрипта для подготовки, сжатия и развертывания архива с eclipse приведен в Приложение 1.

6.2. Диаграмма активности системы автоматизированной сборки, тестирования и развертывания

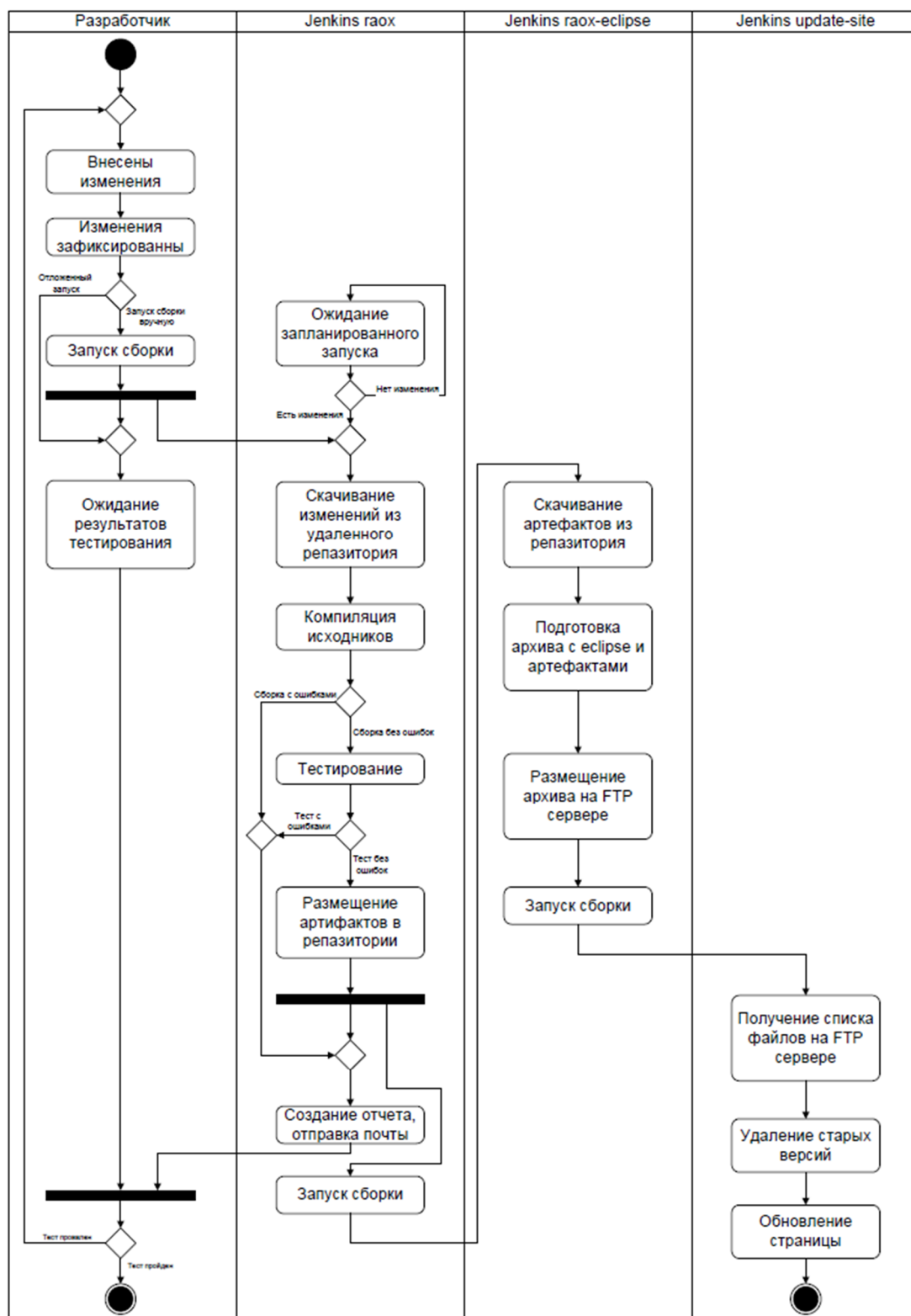


Рис. 6

На Рис. 6 изображена диаграмма активности, которая подробно показывает работу системы, описанной на этапе рабочего и технического проектирования.

7. Апробирование разработанной системы для модельных условий

Апробирование разработанной системы осуществлялось при помощи многократного тестирования функционала. Выявленные в процессе тестирования ошибки и недочеты были исправлены на этапе рабочего проектирования. На этом же были разработаны первые Unit тесты системы. В будущем развитие этой системы, а именно добавлении регрессионных и интеграционных тестов, а также повышение покрытия Unit тестов позволит повысить качество системы и облегчит поддержание ее в рабочем.

На листе результатов представлены снимки работающей системы, а именно работающая и настоящая система непрерывной интеграции Jenkins, со всеми описанными задачами, а также консоль одной из отладочных сборок; вкладка «скачать» сайта gaoh.ru, с доступными для скачивания архивами с системой Rao X для ОС windows x64, x86 и linux x64, x86; веб интерфейс публичного репозитория с .jar артефактами системы Rao X, а так же страничка с документацией по сборке с помощью разработанной системы.

8. Заключение

В рамках данного курсового проекта были получены следующие результаты:

1. Проведено предпроектное исследование системы имитационного моделирования Rao X и доступных, для системы автоматизированной сборки, тестирования и развёртывания, ресурсов
2. На этапе концептуального проектирования системы был сделан выбор общесистемной методологии проектирования, с помощью диаграммы пакетов нотации UML определены зависимости, определена топология системы. Сформулировано техническое задание
3. На этапе технического проектирования выбраны необходимые технические средства, уточнена топология системы. Разработана диаграмма развёртывания разрабатываемой системы.
4. На этапе рабочего проектирования написан программный код скрипта для реализации спроектированных ранее алгоритмов работы. Архитектура maven сборки показана на соответствующей схеме. Разработана диаграмма активности системы. Настроены сервер сборки и все необходимые задачи, публичный репозиторий, FTP-сервер и прокси-сервер
5. Результаты апробирования позволяют сделать вывод об адекватной работе разрабатываемой системы
6. Была разработана документация по функционалу системы, являющейся руководством по его использованию на сервере или персональном компьютере

Список литературы

1. Емельянов В.В., Ясиновский С.И. Имитационное моделирование систем, язык и среда РДО. М.: МГТУ им. Н. Э. Баумана, 2009. -583с.
2. Мартин Р. Чистый код. Создание, анализ и рефакторинг / пер. с англ. Е. Матвеев – СПб.: Питер, 2010. – 464 стр.
3. Макконнелл С. Совершенный код. Мастер класс – М.: Издательско-торговый дом «Русская Редакция» ; СПб.: Питер, 2005. – 896 стр.: ил.
4. Repository Management with Nexus [<http://books.sonatype.com/nexus-book/3.0/reference/index.html>]
5. Программирование на Java. Java Maven. [<http://javaxblog.ru/article/java-maven-1/>]

Приложение 1. Код maven скрипта для подготовки архива

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>ru.bmstu.rk9.rao</groupId>
  <artifactId>deploy</artifactId>
  <version>1.0.0</version>

  <name>Package archive with eclipse</name>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <set_rao_perspective-token>-product</set_rao_perspective-token>
    <set_rao_perspective-value>
      -perspective${line.separator}
      ru.bmstu.rk9.rao.ui.perspective${line.separator}
      -product
    </set_rao_perspective-value>
    <target-os-property></target-os-property>
    <target-bitness-property></target-bitness-property>
    <raox-version></raox-version>
    <eclipse-content-relative-path></eclipse-content-relative-path>
  </properties>

  <profiles>
    <profile>
      <id>mac-os</id>
      <activation>
        <property>
          <name>target-os</name>
          <value>mac</value>
        </property>
      </activation>
      <properties>
        <eclipse-content-relative-path>Contents/Eclipse</eclipse-content-
relative-path>
      </properties>
    </profile>
    <profile>
      <id>set-bitness</id>
      <activation>
        <property>
          <name>target-bitness</name>
        </property>
      </activation>
      <properties>
        <target-bitness-property>-${target-bitness}</target-bitness-property>
      </properties>
    </profile>
    <profile>
      <id>set-os</id>
      <activation>
        <property>
          <name>target-os</name>
        </property>
      </activation>
      <properties>
```

```

    <target-os-property>-${target-os}</target-os-property>
  </properties>
</profile>
<profile>
  <id>set-raox-version</id>
  <build>
    <plugins>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>versions-maven-plugin</artifactId>
        <version>2.2</version>
        <configuration>
          <properties>
            <property>
              <name>raox-version</name>
              <dependencies>
                <dependency>
                  <groupId>ru.bmstu.rk9.rao</groupId>
                  <artifactId>ru.bmstu.rk9.rao.lib</artifactId>
                </dependency>
              </dependencies>
            </property>
          </properties>
          <generateBackupPoms>false</generateBackupPoms>
        </configuration>
        <executions>
          <execution>
            <phase>initialize</phase>
            <goals>
              <goal>update-properties</goal>
            </goals>

            </execution>
          </executions>
        </plugin>
      </plugins>
    </build>
  </profile>
<profile>
  <id>package-as-eclipse-dropins</id>
  <activation>
    <property>
      <name>eclipse-archive-root-path</name>
    </property>
  </activation>
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-resources-plugin</artifactId>
        <version>2.7</version>
        <executions>
          <execution>
            <id>copy-eclipse-ini</id>
            <phase>prepare-package</phase>
            <goals>
              <goal>copy-resources</goal>
            </goals>
            <configuration>
              <overwrite>true</overwrite>
              <outputDirectory>${basedir}/target</outputDirectory>
              <resources>
                <resource>

```

```

        <directory>${eclipse-archive-root-path}/${eclipse-content-
relative-path}</directory>
        <includes>
            <include>eclipse.ini</include>
        </includes>
    </resource>
</resources>
</configuration>
</execution>
</executions>
</plugin>
<plugin>
    <groupId>com.google.code.maven-replacer-plugin</groupId>
    <artifactId>replacer</artifactId>
    <version>1.5.3</version>
    <executions>
        <execution>
            <phase>prepare-package</phase>
            <goals>
                <goal>replace</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <ignoreMissingFile>true</ignoreMissingFile>
        <file>${basedir}/target/eclipse.ini</file>
        <replacements>
            <replacement>
                <token>${set_rao_perspective-token}</token>
                <value>${set_rao_perspective-value}</value>
            </replacement>
        </replacements>
    </configuration>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-assembly-plugin</artifactId>
    <version>2.5.5</version>
    <configuration>
        <descriptor>eclipse-dropins.xml</descriptor>
        <finalName>raox</finalName>
    </configuration>
    <executions>
        <execution>
            <phase>package</phase>
            <goals>
                <goal>single</goal>
            </goals>
        </execution>
    </executions>
</plugin>
</plugins>
</build>
</profile>
</profiles>

<repositories>
    <repository>
        <id>eclipse</id>
        <layout>p2</layout>
        <url>http://download.eclipse.org/releases/mars/201506241002/</url>
    </repository>
    <repository>

```

```

    <id>raox-repo</id>
    <url>http://raox.rk9.bmstu.ru/nexus/content/repositories/raox-m2/</url>
  </repository>
</repositories>

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-dependency-plugin</artifactId>
      <version>2.10</version>
      <executions>
        <execution>
          <id>copy</id>
          <phase>package</phase>
          <goals>
            <goal>copy</goal>
          </goals>
          <configuration>
            <artifactItems>
              <artifactItem>
                <groupId>ru.bmstu.rk9.rao</groupId>
                <artifactId>ru.bmstu.rk9.rao</artifactId>
                <version>LATEST</version>
                <outputDirectory>${basedir}/target/plugins</outputDirectory>
              </artifactItem>
              <artifactItem>
                <groupId>ru.bmstu.rk9.rao</groupId>
                <artifactId>ru.bmstu.rk9.rao.lib</artifactId>
                <version>LATEST</version>
                <outputDirectory>${basedir}/target/plugins</outputDirectory>
              </artifactItem>
              <artifactItem>
                <groupId>ru.bmstu.rk9.rao</groupId>
                <artifactId>ru.bmstu.rk9.rao.ui</artifactId>
                <version>LATEST</version>
                <outputDirectory>${basedir}/target/plugins</outputDirectory>
              </artifactItem>
              <artifactItem>
                <groupId>ru.bmstu.rk9.raox.plugins.game5</groupId>
                <artifactId>ru.bmstu.rk9.raox.plugins.game5</artifactId>
                <version>LATEST</version>
                <outputDirectory>${basedir}/target/plugins</outputDirectory>
              </artifactItem>
            </artifactItems>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
</project>

```

Приложение 2. Инструкция по сборке с помощью maven

Тест инструкции доступен в репозитории с исходниками по ссылке <https://github.com/aurusov/raox/blob/release/current/build-with-maven.md>

Сборка через maven

Общие положения

```
[export MAVEN_OPTS="-Xmx512M"]  
[mvn initialize -N -Pset-git-version]  
mvn [clean] package [-Declipse-path=<path-to-eclipse-root>] [-Dtarget-os=linux|win|mac] [-Dtarget-bitness=x64|x86]
```

- `export MAVEN_OPTS="-Xmx512M"` - Устанавливает максимальное количество выделяемой памяти. Команда необходима, если сборка выдает ошибку `java heap space`. Под Windows `set MAVEN_OPTS="-Xmx512M"` или через настройки переменных среды.
- `mvn initialize -N -Pset-git-version` - Устанавливает версию бинарников на основе гита. Обязательная команда, если бинарники передаются пользователям.
 - `-N` или `--non-recursive` - Запускает только родительский `pom.xml`. Без этого флага версия поменяется, но сборка завершится с ошибкой, что критично для сборки на Дженинсе.
 - `-Pset-git-version` - Определяет и устанавливает версию на основе `git describe --tags`
- `mvn [clean] package` - Запускает очистку (`clean`) и следом за ней сборку (`package`) проекта, как если бы они запускались отдельно

```
[mvn clean]  
mvn package
```

`mvn clean` - Удаляет бинарники и содержимое папок `target`.

`mvn package` - Запускает сборку. Результаты сборки

- архив с плагинами `assembly\target\rao-<version>-plugins.zip`. Например
 - `rao-2.8.0.12-gea41589-plugins.zip` для не релизной сборки
 - `rao-2.8.0-plugins.zip` для релизной
 - папка с плагинами
 - `assembly\target\plugins\ru.bmstu.rk9.rao.lib-<version>.jar`
 - `assembly\target\plugins\ru.bmstu.rk9.rao.ui-<version>.jar`
 - `assembly\target\plugins\ru.bmstu.rk9.rao-<version>.jar`
 - `mvn clean package [-Declipse-path=<path-to-eclipse-root>] [-Dtarget-os=linux|win|mac] [-Dtarget-bitness=x64|x86]`
 - Запускает сборку, результатом которой является готовый к развертыванию архив, содержащий `eclipse` и плагины из пункта выше, подставленные в папку `dropins`.
- Параметр `eclipse-path` указывает путь до корневой папки `eclipse`. Значение `<path-to-eclipse-root>` задается

абсолютным путем или относительно папки `assembly`. Если путь содержит пробелы, то кавычки ставятся вокруг параметра целиком, т.е.

```
mvn clean package "-Declipse-path=C:\path with spaces\eclipse"
```

Параметры `target-os` и `target-bitness` задают операционную систему и битность целевой платформы соответственно. Влияют на название получаемого архива: `rao-<version>-<target-os>-<target-platform>.zip`. Имеют смысл только для сборки с параметром `-Declipse-path`, обязательны для сборки на Дженкинсе.

Автосборка на Дженкинсе

Только джарники

```
export MAVEN_OPTS="-Xmx512M"
mvn initialize -N -Pset-git-version
mvn clean package
```

Архив с Эклипсом

```
export MAVEN_OPTS="-Xmx512M"
mvn initialize -N -Pset-git-version
mvn clean package -Declipse-path=../../../../../eclipses/eclipse-dsl-mars-1-linux-gtk-x64 -Dtarget-os=linux -Dtarget-bit
```