



**«Московский государственный технический университет  
имени Н.Э. Баумана»**

**(МГТУ им. Н.Э. Баумана)**

---

ФАКУЛЬТЕТ \_\_\_\_\_

КАФЕДРА \_\_\_\_\_

---

## **РАСЧЁТНО - ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту на тему:**

---

---

---

---

---

---

---

---

Студент \_\_\_\_\_ (Подпись, дата) \_\_\_\_\_ (И.О.Фамилия)

Руководитель курсового проекта \_\_\_\_\_ (Подпись, дата) \_\_\_\_\_ (И.О.Фамилия)

Москва, 20\_\_

Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»  
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой \_\_\_\_\_  
(Индекс)

(И.О.Фамилия)

« \_\_\_\_ » \_\_\_\_\_ 20 \_\_\_\_ г.

## З А Д А Н И Е

### на выполнение курсового проекта

по дисциплине \_\_\_\_\_

(Тема курсового проекта)

Студент \_\_\_\_\_  
(Фамилия, инициалы, индекс группы)

График выполнения проекта: 25% к \_\_\_\_ нед., 50% к \_\_\_\_ нед., 75% к \_\_\_\_ нед., 100% к \_\_\_\_ нед.

#### 1. Техническое задание

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

#### 2. Оформление курсового проекта

2.1. Расчетно-пояснительная записка на \_\_\_\_ листах формата А4.

2.2. Перечень графического материала (плакаты, схемы, чертежи и т.п.) \_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

Дата выдачи задания « \_\_\_\_ » \_\_\_\_\_ 20 \_\_\_\_ г.

Руководитель курсового проекта

\_\_\_\_\_  
(Подпись, дата)

\_\_\_\_\_  
(И.О.Фамилия)

Студент

\_\_\_\_\_  
(Подпись, дата)

\_\_\_\_\_  
(И.О.Фамилия)

#### Примечание:

1. Задание оформляется в двух экземплярах; один выдаётся студенту, второй хранится на кафедре.

## Оглавление

|  |    |
|--|----|
| Введение .....   | 3  |
| 1. Предпроектное исследование .....                                    | 4  |
| 1.1. Тестирование .....  | 4  |
| 1.2. Исходное состояние программной части проекта РДО .....            | 4  |
| 1.3. Основные положения языка РДО. ....                                | 5  |
| 1.4. Постановка задачи .....   | 7  |
| 2. Разработка технического задания на систему .....                    | 8  |
| 2.1. Основания для разработки.....                                     | 8  |
| 2.2. Назначение разработки .....                                       | 8  |
| 2.3. Характеристики объекта автоматизации. ....                        | 8  |
| 2.4. Требования к программе или программному изделию.....              | 8  |
| 2.4.1. Требования к системе. ....                                      | 8  |
| 2.4.2    Требования к надежности .....                                 | 8  |
| 2.4.3    Условия эксплуатации .....                                    | 9  |
| 2.4.4    Требования к составу и параметрам технических средств.....    | 9  |
| 2.4.5    Требования к информационной и программной совместимости ..... | 9  |
| 2.4.6    Требования к маркировке и упаковке.....                       | 9  |
| 2.4.7    Требования к транспортированию и хранению .....               | 9  |
| 2.5. Требования к программной документации .....                       | 9  |
| 2.6. Стадии и этапы разработки .....                                   | 9  |
| 2.7. Порядок контроля и приемки .....                                  | 9  |
| 3. Концептуальный этап проектирования .....                            | 10 |
| 3.1. Метод непрерывной интеграции.....                                 | 10 |
| 3.2. Диаграмма пакетов тестов РДО .....                                | 11 |
| 3.3. Разработка системы системного тестирования.....                   | 12 |
| 4. Техническое проектирование .....                                    | 13 |

|   |    |
|---|----|
| 4.1. Проектирование RDO-Console и RDO-Console-Test.....                   | 13 |
| 5. Рабочее проектирование .....   | 15 |
| 5.1. Разработка диаграммы состояний RDO-Console-Test .....                | 15 |
| 5.2. Разработка диаграммы состояний системы системного тестирования ..... | 16 |
| 6. Результаты.....  | 17 |
| 6.1. Новая система тестирования .....                                     | 17 |
| 6.2. Сервер непрерывной интеграции.....                                   | 17 |
| 6.3. Выполнение блочных и компонентных тестов .....                       | 18 |
| 6.4. Выполнение системных тестов.....                                     | 19 |
| Заключение.....   | 20 |
| Список использованных источников.....                                     | 21 |
| Приложение А. Исходный код программы системного тестирования .....        | 22 |
| Приложение Б. Документация по развертыванию сервера. ....                 | 26 |

## **Введение**

При создании программного обеспечения разработчики сталкиваются с проблемой повышения качества и надёжности своих программ. Для решения этой задачи могут служить различные методы: качественное проектирование, выбор современного языка программирования высокого уровня, надёжные средства разработки и отладки, тестирования и многое другое. Тестирование является важной частью разработки надёжного программного обеспечения.

До недавнего времени при разработке системы дискретного имитационного моделирования тестирование было неполноценным и не использовалось должным образом. В этой работе я расскажу о разработке системы тестирования для РДО и о том, как эта система работает сейчас.

## 1. Предпроектное исследование.

### 1.1. Тестирование

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:

- *Блочным тестированием* называют тестирование полного класса, метода или небольшого приложения, написанного одним программистом или группой, выполняемое отдельно от прочих частей системы.
- *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.

### 1.2. Исходное состояние программной части проекта РДО

Система дискретного имитационного моделирования РДО написана на языке программирования C++. Система состоит из восьми библиотек, нескольких приложений, множества файлов. Ее исходные тексты насчитывают порядка 300 тысяч строк исходного кода.

Для тестирования всех компонентов существует порядка десяти блочных тестов. Такое небольшое количество тестов не способно обеспечить надежную и стабильную работу системы моделирования. Тесты всей системы с сборе на момент начала разработки системы тестирования отсутствовали.

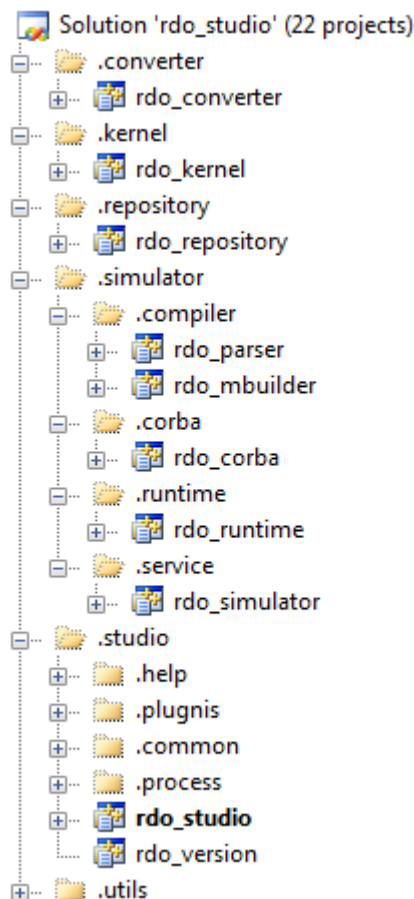


Рис 1.1. Дерево проекта rdo\_studio в IDE Microsoft Visual Studio Windows

1. rdo\_utils – библиотека вспомогательных функций, для работы с файлами, временем, умными указателями.
2. rdo\_kernel – ядро системы моделирования, отвечает за обмен сообщениями между компонентами системы.
3. rdo\_repository – подсистема для открытия, закрытия моделей, сохранения результатов моделирования.
4. rdo\_parser – компилятор современного синтаксиса РДО.
5. rdo\_mbuilder – компилятор графических моделей.
6. rdo\_convertor – преобразователь старых моделей в новые.
7. rdo\_runtime – симулятор, виртуальная машина системы моделирования.
8. rdo\_simulator – обобщенная система для моделирования : объединяет работу Parser, Runtime.
9. rdo\_studio – графическая версия системы моделирования РДО.

### 1.3. Основные положения языка РДО.

В основе системы РДО – «Ресурсы, Действия, Операции» – лежат следующие положения:

- Все элементы сложной дискретной системы (СДС) представлены как ресурсы, описываемые некоторыми параметрами.
- Состояние ресурса определяется вектором значений всех его параметров; состояние СДС – значением всех параметров всех ресурсов.
- Процесс, протекающий в СДС, описывается как последовательность целенаправленных действий и нерегулярных событий, изменяющих определенным

образом состояния ресурсов; действия ограничены во времени двумя событиями: событиями начала и конца.

- Нерегулярные события описывают изменение состояния СДС, непредсказуемые в рамках продукционной модели системы (влияние внешних по отношению к СДС факторов либо факторов, внутренних по отношению к ресурсам СДС). Моменты наступления нерегулярных событий случайны.

- Действия описываются операциями, которые представляют собой модифицированные продукционные правила, учитывающие временные связи. Операция описывает предусловия, которым должно удовлетворять состояние участвующих в операции ресурсов, и правила изменения ресурсов в начале и конце соответствующего действия.

- При выполнении работ, связанных с созданием и использованием ИМ в среде РДО, пользователь оперирует следующими основными понятиями:

- **Модель** - совокупность объектов РДО-языка, описывающих какой-то реальный объект, собираемые в процессе имитации показатели, кадры анимации и графические элементы, используемые при анимации, результаты трассировки.

- **Прогон** - это единая неделимая точка имитационного эксперимента. Он характеризуется совокупностью объектов, представляющих собой исходные данные и результаты, полученные при запуске имитатора с этими исходными данными.

**Проект** - один или более прогонов, объединенных какой-либо общей целью. Например, это может быть совокупность прогонов, которые направлены на исследование одного конкретного объекта или выполнение одного контракта на имитационные исследования по одному или нескольким объектам.

**Объект** - совокупность информации, предназначенной для определенных целей и имеющая смысл для имитационной программы. Состав объектов обусловлен РДО-методом, определяющим парадигму представления СДС на языке РДО.

Объектами исходных данных являются:

- типы ресурсов (с расширением .trp);
- ресурсы (с расширением .rss);
- образцы операций (с расширением .pat);



- операции (с расширением .org);
- точки принятия решений (с расширением .dpt);
- константы, функции и последовательности (с расширением .fun);
- кадры анимации (с расширением .frm);
- требуемая статистика (с расширением .pmd);
- прогон (с расширением .smr).

Объекты, создаваемые РДО-имитатором при выполнении прогона:

- результаты (с расширением .pmv);
- трассировка (с расширением .trc).

#### **1.4. Постановка задачи**

Требуется разработать удобную и надежную систему тестирования для системы дискретного имитационного моделирования РДО.

## **2. Разработка технического задания на систему**

### **2.1. Основания для разработки**

Задание на курсовой проект.

### **2.2. Назначение разработки**

Цель разработки системы: повышение степени уверенности в правильности работы РДО, сокращение времени обнаружения ошибок.

Основная цель данного курсового проекта – разработать систему тестирования РДО, т.е. написать необходимое программное обеспечение для регулярной сборки и проверки результатов работы компонент системы моделирования и всей системы в целом.

### **2.3. Характеристики объекта автоматизации.**

РДО – язык дискретного имитационного моделирования, включающий на данный момент подход сканирования активностей, процессный и событийный.

### **2.4. Требования к программе или программному изделию**

#### **2.4.1. Требования к системе.**

Система тестирования должна по расписанию и/или по требованию в автоматическом режиме:

1. подготовить среду тестирования (обновить/выкачать код системы и тестов)
2. скомпилировать из исходных кодов систему моделирования РДО и ее тесты
3. запустить на выполнение модульные тесты
4. проверить результаты модульных тестов
5. запустить на выполнение интегральные тесты (на подготовленных моделях)
6. проверить результаты интегральных тестов
7. отправить отчет о тестировании разработчику, инициировавшему тестирование, и его руководителю

#### **2.4.2 Требования к надежности**

Основное требование к надежности направлено на поддержание в исправном и работоспособном состоянии ЭВМ, на которой происходит использование программного комплекса Jenkins.

### **2.4.3 Условия эксплуатации**

Аппаратные средства должны эксплуатироваться в помещениях с выделенной розеточной электросетью 220В ±10%, 50 Гц с защитным заземлением.

### **2.4.4 Требования к составу и параметрам технических средств**

Программный продукт должен работать на компьютерах со следующими характеристиками:

- объем ОЗУ не менее 512 Мб;
- объем жесткого диска не менее 10 Гб;
- микропроцессор с тактовой частотой не менее 800 МГц;
- java 1.6

### **2.4.5 Требования к информационной и программной совместимости**

Данная система должна работать под управлением различных дистрибутивов GNU/Linux.

### **2.4.6 Требования к маркировке и упаковке**

Не предъявляются.

### **2.4.7 Требования к транспортированию и хранению**

Не предъявляются.

## **2.5. Требования к программной документации**

Необходимо разработать документацию по системе тестирования в виде инструкции для разработчиков РДО и инструкции для администратора системы.

## **2.6. Стадии и этапы разработки**

1. Предпроектное исследование.
2. Изучить методологию TDD и Continuous Integration.
3. Изучить существующие решения для организации тестирования.
4. Концептуальный этап проектирования.
5. Разработать алгоритм добавления новой функциональности в систему РДО.
6. Технический этап проектирования.
7. Рабочий этап проектирования.

## **2.7. Порядок контроля и приемки**

Контроль и приемка системы тестирования должны осуществляться на тестовом примере.

### 3. Концептуальный этап проектирования

#### 3.1. Метод непрерывной интеграции

По наблюдениям Ф. Брукса, тестирование занимает половину или больше половины времени разработки проекта.

Успешным методом, зарекомендовавшим себя для повышения удобства и сокращения времени тестирования, является метод *Continuous Integration* (метод непрерывной интеграции). Это процесс, при котором автоматизированным способом выкачиваются исходные тексты программ, происходит их сборка, выполнение тестов и извещение разработчиков о результатах тестирования. Подобная практика себя успешно применяется во многих современных компаниях по разработке программного обеспечения. Как правило, сборки проводятся ночью, чтобы результаты тестирования были готовы к началу рабочего дня. Как и любой другой метод, непрерывная интеграция обладает достоинствами и недостатками.

Достоинства:

- немедленный прогон блочных тестов для свежих изменений
- постоянное наличие текущей стабильной версии вместе с продуктами сборки — для тестирования, демонстрации и т. п.

Недостатки:

1. затраты на поддержку работы непрерывной интеграции, обучение программистов
2. необходимость в выделенном сервере под нужды непрерывной интеграции

Для развертывания системы непрерывной интеграции существуют готовые решения. Командой разработчиков РДО был выбран программный продукт Jenkins (Дженкинс). Дженкинс – это инструмент непрерывной интеграции, написанный на языке Java. Распространяется по лицензии MIT и бесплатно доступен для скачивания с <http://jenkins-ci.org> для любого использования без ограничений. Основным компонентом тестирования, с которым работает среда, является задача – это своего рода проект, для которого конфигурируются настройки, расписание запусков, действия системы тестирования на результат сборки. Также система обладает богатым функционалом, настройкой прав доступа и многим другим. Работа с системой осуществляется через браузер.

### 3.2. Диаграмма пакетов тестов РДО

Для проверки механизма моделирования был выбран набор моделей, которые будут использоваться для системного тестирования.

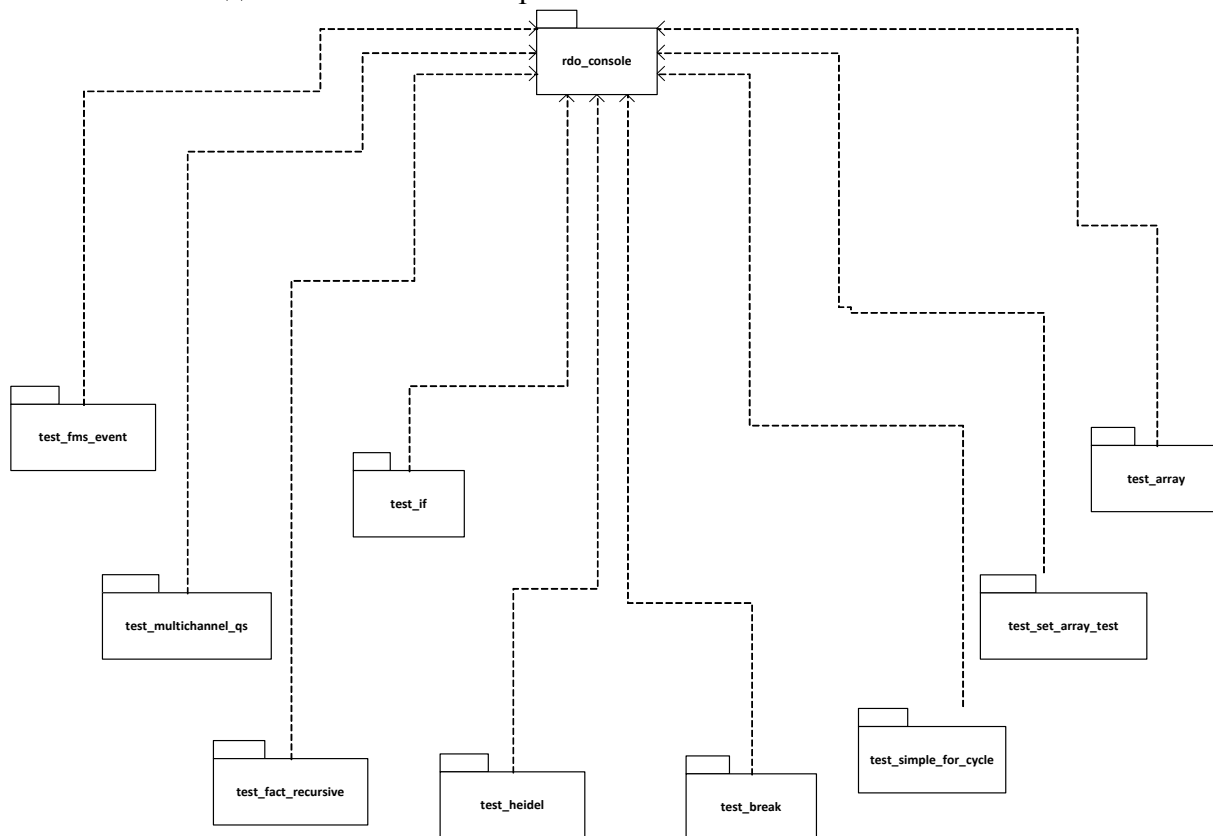


Рис 3.1. Диаграмма системных тестов РДО.

В ходе разработке системы тестирования необходимо обновить существующие блочные тесты для того, чтобы они могли, выполняться в автоматическом режиме системой непрерывной интеграции.

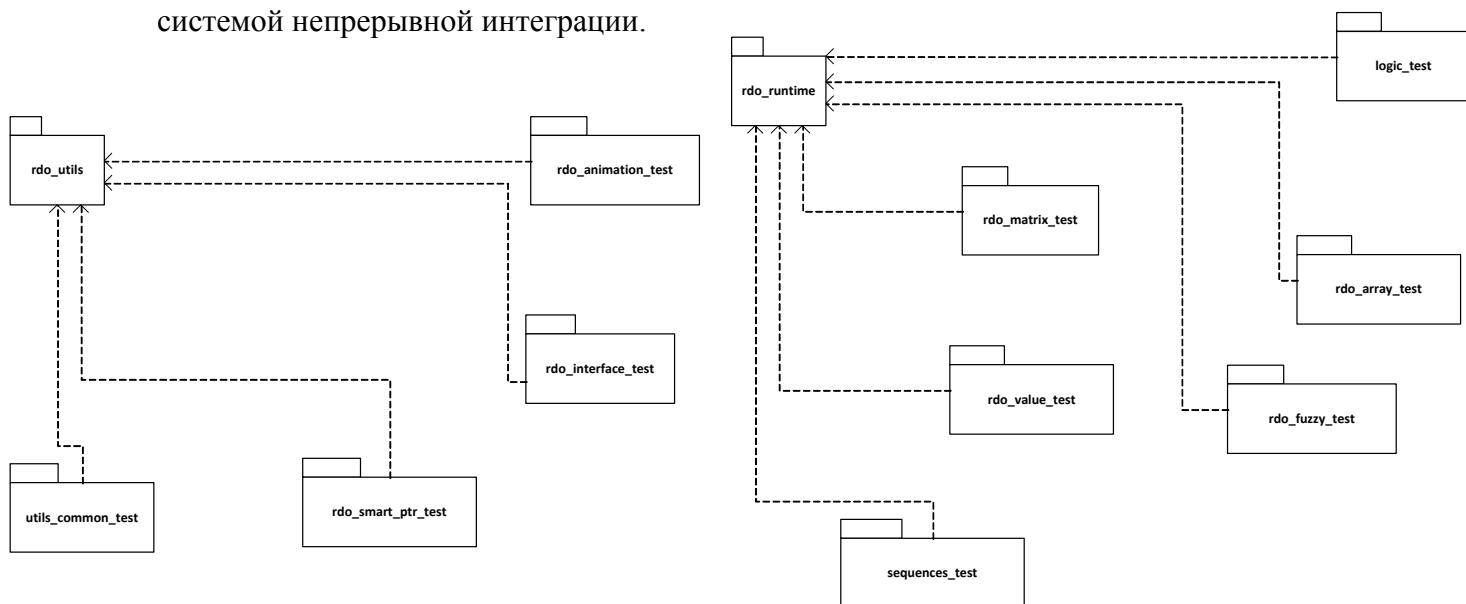


Рис 3.2. Диаграмма пакетов блочных тестов РДО.

### 3.3. Разработка системы системного тестирования

Система системного тестирования должна найти файлы тестовых моделей в заданном каталоге, выполнять моделирование на тестовых моделях, сравнивать результаты трассировки и результатов с эталонами. Информация о системных тестах храниться в .rtestx. Типичный файл .rtestx выглядит следующим образом:

```
<?xml version="1.0"?>
<test>
  <model>heidel.rdox</model>
  <target>CONSOLE</target>
  <exit_code>0</exit_code>
  <trace>heidel_etalon.trc</trace>
  <result>heidel_etalon.pmv</result>
</test>
```

XML документ состоит из следующих блоков:

- <model></model> – относительный путь к модели
- <target></target> – тип теста
- <exit\_code></exit\_code> – код завершения системы моделирования
- <trace></trace> – относительный путь к эталонному файлу трассировки
- <result></result> – относительный путь к файлу результатов

Для удобства редактирования и просмотра файлы .rtestx необходимо располагать как можно ближе к тестируемой модели, лучше всего в этой же папке.

#### 4.1. Проектирование RDO-Console и RDO-Console-Test

## 4.1. Проектирование RDO-Console и RDO-Console-Test

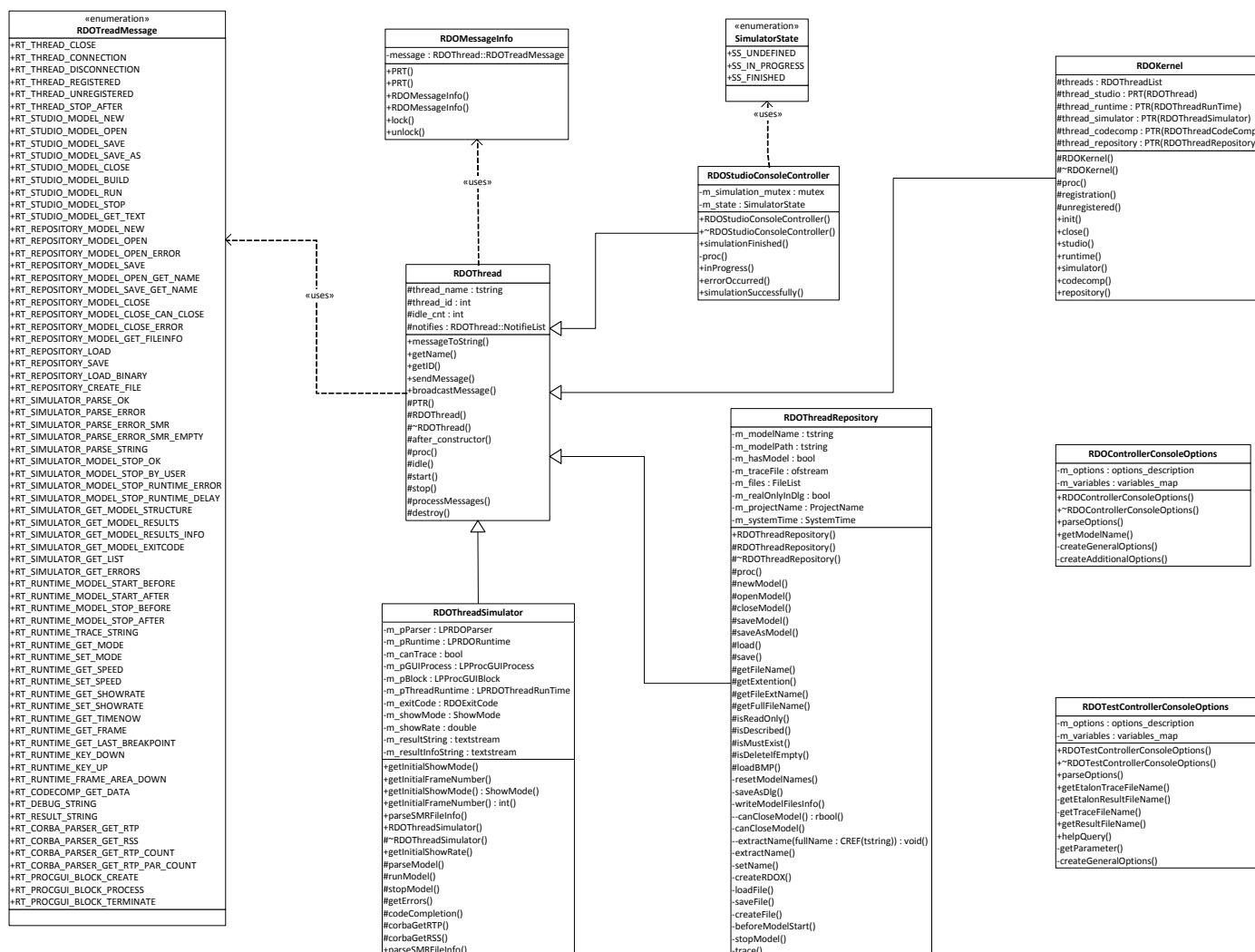


Рис 4.1. Диаграмма классов RDO-Studio-Console

Приложение RDO-Studio-Console содержит классы:

- RDOKernel – класс ядра системы моделирования
- RDOThreadSimulator – класс системы моделирования
- RDOThreadRepository – класс системы открытия, сохранения моделей и результатов
- RDOStudioConsoleController – клиентский класс управления системой моделирования

- `RDOControllerConsoleOptions` – вспомогательный класс обработки параметров командной строки

К консольной версии системы моделирования предъявляются требования:

1. Возвращение кода возврата выполнения.

К тестирующей программе предъявляются требования:

1. Сравнение файлов результатов и трассировок с учетом внутренней структуры этих файлов.
2. Возвращение программой кода сравнения.



## 5. Рабочее проектирование

### 5.1. Разработка диаграммы состояний RDO-Console-Test

Диаграммы состояний (state machine diagrams) – это известная технология описания поведения системы. В том или ином виде диаграммы состояний существуют с 1960 года, и на заре объектно-ориентированного программирования они применялись для представления поведения системы.



Рис 5.1. Диаграмма состояний RDO-Console-Test

Диаграмма позволяет понять последовательность выполнения действий, происходящих в процессе работы программы. В ходе работы программа разбирает входные данные, проверяет наличие моделей, выполняет сравнение и возвращает результат.

## 5.2. Разработка диаграммы состояний системы системного тестирования

Для тестирования всей системы дискретного имитационного моделирования в целом разработана программа на языке python.

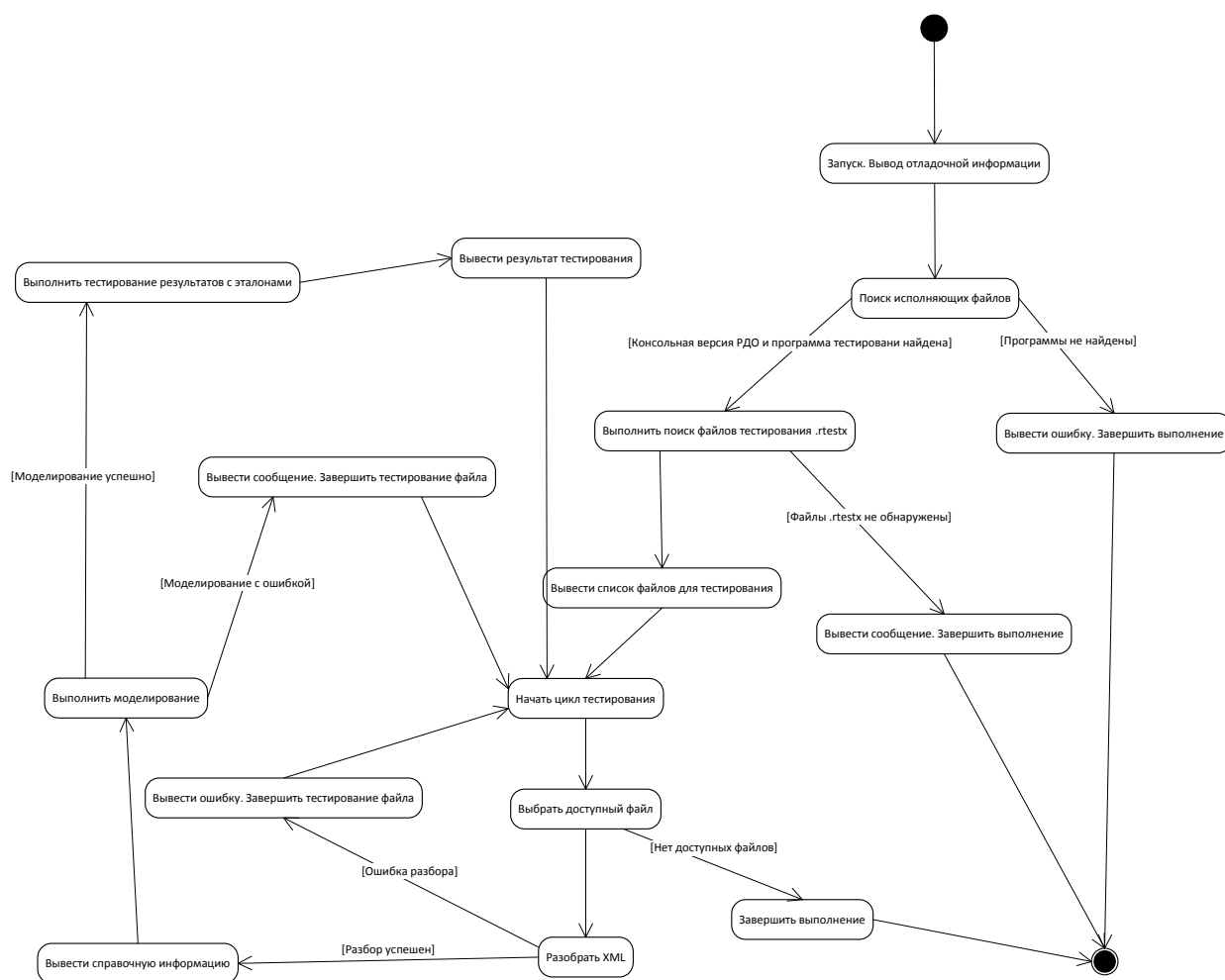


Рис 5.2. Диаграмма состояний системы системного тестирования

Программа выполняет поиск тестовых моделей в каталоге проекта (поиск файлов .rtestx). Выполняет XML разбор. Выполняет моделирование и сравнение результатов моделирования, выводит статистику и результат системного тестирования.

## 6. Результаты

### 6.1. Новая система тестирования

В ходе работы над проектом была разработана и реализована система тестирования с использованием современных средств и методик. Система состоит из блочных компонентных тестов исходного кода на языке C++ с использованием библиотеки BOOST UNIT TEST и средства сборки CMake/CTest. Для проверки работоспособности всей системы разработаны RDO-Console-Test и тестирующая программа на python. Также подготовлен первый набор системных тестов. Настроен и запущен сервер с системой непрерывной интеграции с использованием Jenkins.

### 6.2. Сервер непрерывной интеграции

Был настроен сервер с ОС Ubuntu Linux 12.04 LTS. На него было установлено все необходимое программное обеспечение и система Jenkins.

Каждому из разработчиков RDO была создана своя задача / проект в Jenkins. Система предоставляет статистику по успехам и неудачам выполнения тестов и другую справочную информацию.



| S | W | Name ↓                                    | Крайний успех          | Крайняя неудача | Крайняя продолжительность |
|---|---|---|------------------------|-----------------|---------------------------|
|   |   | <a href="#">rdo-console-trunk</a>         | 8 часов 53 минут (#39) | Неизвестно      | 4 минуты 29 секунд        |
|   |   | <a href="#">rdo-studio-fuzzylogic</a>     | 8 часов 53 минут (#22) | Неизвестно      | 2 минуты 18 секунд        |
|   |   | <a href="#">rdo-studio-multithreading</a> | 8 часов 53 минут (#21) | Неизвестно      | 2 минуты 17 секунд        |
|   |   | <a href="#">rdo-studio-oop</a>            | 8 часов 53 минут (#21) | Неизвестно      | 2 минуты 18 секунд        |
|   |   | <a href="#">rdo-studio-processqui</a>     | 8 часов 53 минут (#23) | Неизвестно      | 4 минуты 54 секунд        |
|   |   | <a href="#">rdo-studio-trunk</a>          | 12 часов (#27)         | Неизвестно      | 1 минута 26 секунд        |
|   |   | <a href="#">rdo-studio-undefparam</a>     | 8 часов 53 минут (#19) | Неизвестно      | 3 минуты 49 секунд        |

Иконка: [S](#) [M](#) [L](#)


[Легенда](#) [RSS для всех](#) [RSS для неудачных](#) [RSS для последних сборок](#)

Рисунок 6.1. Успешные сборки веток проекта в Jenkins.

Удобный доступ через WEB интерфейс позволяет работать системой тестирования с различных компьютеров и операционных систем.

### 6.3. Выполнение блочных и компонентных тестов

Для разработки блочных тестов используется библиотека Boost Unit Test, содержащая компоненты для удобного написания автоматических тестов. В качестве средства сборки используются CMake/CTest позволяющие по команде `make test` запустить выполнение тестирующих подпрограмм.



```
[ 97%] Built target test_rdo_sequences
[ 99%] Built target rdo
[100%] Built target rdo_test
+ make test
Running tests...
Test project /home/jenkins/jobs/rdo-console-trunk/workspace/build
  Start 1: test_rdo_utils
1/10 Test #1: test_rdo_utils ..... Passed    0.07 sec
  Start 2: test_rdo_utils_smart_prt
2/10 Test #2: test_rdo_utils_smart_prt ..... Passed    0.01 sec
  Start 3: test_rdo_utils_interface
3/10 Test #3: test_rdo_utils_interface ..... Passed    0.01 sec
  Start 4: test_rdo_animation
4/10 Test #4: test_rdo_animation ..... Passed    0.02 sec
  Start 5: test_rdo_runtime_logic
5/10 Test #5: test_rdo_runtime_logic ..... Passed    0.01 sec
  Start 6: test_rdo_runtime_array
6/10 Test #6: test_rdo_runtime_array ..... Passed    0.02 sec
  Start 7: test_rdo_runtime_matrix
7/10 Test #7: test_rdo_runtime_matrix ..... Passed    0.02 sec
  Start 8: test_rdo_runtime_value
8/10 Test #8: test_rdo_runtime_value ..... Passed    0.02 sec
  Start 9: test_rdo_runtime_fuzzy
9/10 Test #9: test_rdo_runtime_fuzzy ..... Passed    0.02 sec
  Start 10: test_rdo_sequences
10/10 Test #10: test_rdo_sequences ..... Passed   10.25 sec

100% tests passed, 0 tests failed out of 10

Total Test time (real) = 10.47 sec
Finished: SUCCESS
```

Рисунок 6.2. Успешное прохождение компиляции и выполнение блочных автотестов.

## 6.4. Выполнение системных тестов

Для выполнения системных тестов разработана тестирующая программа на языке python. Ее исходный код приведен в приложении А. В ходе выполнения системных тестов выводится лог сообщений следующего вида:

```
STARTED SCRIPT : trunk/scripts/python/test.py

DEBUG INFO

Find test project files : ['./trunk/app/rdo_console/test/fms_event/fms_event.rtestx',
                           './trunk/app/rdo_console/test/multichannel_qs/multichannel_qs.rtestx', './trunk/app/rdo_console/test/simple_qs/array.rtestx',
                           './trunk/examples/heidel/heidel.rtestx']
Find RDO executables   : ('./build/rdo', './build/rdo_test')

STARTED TEST CYCLE
-----
Project      : ./trunk/app/rdo_console/test/fms_event/fms_event.rtestx
Model file   : fms_event.rdox
Target       : CONSOLE
Exit code    : 0
Trace file   : fms_event_etalon.trc
Result file  : fms_event_etalon.pmv
SIMYLATION EXIT CODE : 0
CHECK EXIT CODE      : OK
-----
Project      : ./trunk/app/rdo_console/test/multichannel_qs/multichannel_qs.rtestx
Model file   : multichannel_qs.rdox
Target       : CONSOLE
Exit code    : 0
Trace file   : multichannel_qs_etalon.trc
Result file  : multichannel_qs_etalon.pmv
SIMYLATION EXIT CODE : 0
CHECK EXIT CODE      : OK
-----
Project      : ./trunk/app/rdo_console/test/simple_qs/array.rtestx
Model file   : simple_qs.rdox
Target       : CONSOLE
Exit code    : 0
Trace file   : simple_qs_etalon.trc
Result file  : simple_qs_etalon.pmv
SIMYLATION EXIT CODE : 0
CHECK EXIT CODE      : OK
-----
Project      : ./trunk/examples/heidel/heidel.rtestx
Model file   : heidel.rdox
Target       : CONSOLE
Exit code    : 0
Trace file   : heidel_etalon.trc
Result file  : heidel_etalon.pmv
SIMYLATION EXIT CODE : 0
CHECK EXIT CODE      : OK
-----
Finished: SUCCESS
```

Рисунок 6.3. Успешное прохождение системных тестов.

## **Заключение**

В результате проведенной работы была разработана система тестирования для системы дискретного имитационного моделирования РДО. Настроен и введен в эксплуатацию сервер непрерывной интеграции. Повышена степень доверия разработчиков к системе. Сокращен цикл разработки.

## Список использованных источников

1. Бьерн Страуструп. Язык моделирования C++. Специальное издание. Пер. с англ. – М.: ООО «Бином-пресс», 2007 г. – 1104 с.
2. Солтер Н.А., Клепер С.Дж. C++ для профессионалов. М.: Диалектика, 2006 г. , пер. с 2005, 907 с.
3. Макконнелл С. Совершенный код. Мастер класс – М.: Издательско-торговый дом «Русская Редакция» ; СПб.: Питер, 2005. – 896 стр.: ил.
4. Брукс Ф. Мифический человеко-месяц или как создаются программные системы. - Символ-Плюс, 2006, 304 с.
5. Фаулер М. Основы UML, 3-е издание. : СПб: Символ-Плюс, 2002 г. 185 с.
6. Ларман, Крэг. Применение UML и шаблонов проектирования. 2-е издание. : Пер. с англ. — М. : Издательский дом "Вильямс", 2004. — 624 с.
7. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению. ГОСТ 19.201-78.
8. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. ГОСТ 19.701-90. Условные обозначения и правила выполнения.
9. <http://www.boost.org/> - сайт библиотеки Boost
10. <http://www.cmake.org/> - сайт системы сборки CMake/CTest
11. <http://jenkins-ci.org/> - сайт системы непрерывной интеграции Jenkins

## Приложение А. Исходный код программы системного тестирования

```
#####
# Copyright (c) 2012 Evgeny Proydakov <lord.tiran@gmail.com>
#####

import os
import sys
import xml.dom.minidom

#####
#                                     constant
#####

directory          = '.'
model_directory    = '.'
test_expansion     = '.rtestx'
trace_expansion    = '.trc'
result_expansion   = '.pmv'
build_dir_substr   = 'build'
rdo_ex_substr      = 'rdo'
rdo_test_ex_substr = 'rdo_test'

null_file = 'null_temp_file'

if sys.platform == 'win32':
    rdo_ex_substr      = 'rdo.exe'
    rdo_test_ex_substr = 'rdo_test.exe'

dividing_line = '-----'
dividing_line = '-----'

TARGET_CONSOLE    = 'CONSOLE'
TAGRET_CONVERTER  = 'CONVERTOR'
TARGET_GUI        = 'GUI'

EXIT_CODE_TERMINATION_NORMAL = 0
EXIT_CODE_TERMINATION_ERROR  = 1

# global exit code variable
G_EXIT_CODE = EXIT_CODE_TERMINATION_NORMAL

#####
#                                     functions
#####

def get_files_list(dir):
    dirs = []
    nfile = []
    files = []
    for dirname, dirnames, filenames in os.walk(dir):
        dirs.append(dirname)
        for subdirname in dirnames:
            dirs.append(os.path.join(dirname, subdirname))
        for filename in filenames:
            files.append(os.path.join(dirname, filename))
    return files

def get_test_files(dir):
```



```

    files = get_files_list(dir)
    nfile = filter(lambda x: x.endswith(test_expansion), files)
    return nfile

def get_executables(dir):
    # get rdo_studio_console executable and test_rdo_studio_console
    executable

    files = get_files_list(dir)
    rdo_ex = filter(lambda x: x.endswith(rdo_ex_substr), files)[0]
    rdo_test_ex = filter(lambda x: x.endswith(rdo_test_ex_substr), files)[0]

    return rdo_ex, rdo_test_ex

def get_text_from_node_list(nodelist):
    rc = []
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            rc.append(node.data)
    return ''.join(rc)

#####
#                                     main code
#####

print "STARTED SCRIPT :", sys.argv[0]

# search rdo and rdo_test executables
executables = get_executables(directory)

rdo_ex      = executables[0]
rdo_test_ex = executables[1]

if not os.path.exists(rdo_ex) or not os.path.exists(rdo_test_ex):
    print 'Build app not found. Critical error !!!'
    exit(EXIT_CODE_TERMINATION_ERROR)

# search .rtestx files
files = get_test_files(model_directory)
files.sort()

print '\nDEBUG INFO\n'

print 'Find RDO executables      : ', executables, '\n'
print 'Find test project files : ', files, '\n'

# parse xml and start tests
print 'STARTED TEST CYCLE'

for task in files:
    print dividing_line

    dom = xml.dom.minidom.parse(task)

    dirname = os.path.dirname(task) + '/'

    model_name_with_ex =
get_text_from_node_list(dom.getElementsByTagName('model')[0].childNodes)

```

```

    target =
get_text_from_node_list(dom.getElementsByTagName('target')[0].childNodes)
    exit_code =
get_text_from_node_list(dom.getElementsByTagName('exit_code')[0].childNodes)
    etalon_trace_name =
get_text_from_node_list(dom.getElementsByTagName('trace')[0].childNodes)
    etalon_result_name =
get_text_from_node_list(dom.getElementsByTagName('result')[0].childNodes)

    print 'Project      :', task
    print 'Model file   :', model_name_with_ex
    print 'Target       :', target
    print 'Exit code    :', exit_code
    print 'Trace file    :', etalon_trace_name
    print 'Result file   :', etalon_result_name
    print ''

    model = dirname + model_name_with_ex
    etalon_trace = dirname + etalon_trace_name
    etalon_result =  dirname + etalon_result_name

    model_name = model_name_with_ex.partition('.')[0]

    simulation_trace = dirname + model_name + trace_expansion
    simulation_result = dirname + model_name + result_expansion

    if target == TARGET_CONSOLE:
        simulation_code = os.system(rdo_ex + ' -i ' + model
                                   + ' >> ' + null_file)

        print "SIMYLATION EXIT CODE :", simulation_code

        simulation_exit_code_string = ''
        if cmp(simulation_code, exit_code):
            simulation_exit_code_string = "OK"
        else:
            simulation_exit_code_string = "ERROR"

        print "CHECK SIM EXIT CODE  :", simulation_exit_code_string

        check_exit_code_string = ''
        if simulation_code == EXIT_CODE_TERMINATION_NORMAL:
            test_code = os.system(rdo_test_ex + ' -T ' + etalon_trace
                                  + ' -R ' + etalon_result
                                  + ' -t ' + simulation_trace
                                  + ' -r ' + simulation_result
                                  + ' >> ' + null_file)

            if not test_code == EXIT_CODE_TERMINATION_NORMAL:
                G_EXIT_CODE = EXIT_CODE_TERMINATION_ERROR
                check_exit_code_string = 'ERROR'
            else:
                check_exit_code_string = 'OK'

        print "TEST EXIT CODE      :", test_code
        print "CHECK TEST CODE       :", check_exit_code_string

    else:
        print 'INVALID TARGET'

# remove temp file
#os.remove(simulation_trace)

```

```
        #os.remove(simulation_result)

print

os.remove(null_file)

sys.exit(G_EXIT_CODE)
```

## Приложение Б. Документация по развертыванию сервера.

1. Установить Kubuntu Linux 12.04 LTS или выше. Рекомендуется использовать версии только с увеличенным циклом поддержки. (PS - дженкинс много места занимает - выделите под него раздел в 30ГБ не меньше)
2. Установить ssh nginx vim emacs python для работы с сервером и все что написано в файле doc/configuring.environment.debian, находящемся в корне РДО
3. Установить jenkins с <http://jenkins-ci.org/>
4. Настроить проксирование nginx как в документации на сайте

<https://wiki.jenkins-ci.org/display/JENKINS/Installing+Jenkins+on+Ubuntu>

5. Сделать так чтобы jenkins считал своей базовой директорией /home/jenkins
6. Скопировать всю папку /home/jenkins где все настроено - пользователи и текущая история билдов. - если сервер жив и все цело или у вас есть backup
7. Наслаждаться ...

-----  
**\* Если скопировать не откуда. \***

Убедитесь что верно выполнили пункт 5 !

Заходим на сервер через веб интерфейс с указанием IP адреса. Создаем первого пользователя в окошке старта.

### **Настроить Jenkins -> конфигурирование системы ->**

УБЕДИТЬСЯ ЧТО !

|                     |               |
|---------------------|---------------|
| Домашняя директория | /home/jenkins |
|---------------------|---------------|

**Сообщение системы - Не забываем тестировать свой код !!!**

**Количество сборщиков - 9+**

**Задержка перед сборкой - 5**

**Область защиты (realm) - jenkins's own user database**

**Авторизация включаем - матричное распределение прав.**

**Jenkins URL <http://rdo.rk9.bmstu.ru:81/>**

Уведомление почтой - сервер SMTP 192.168.0.5

Sender E-mail Address - [rdo@rk9.bmstu.ru](mailto:rdo@rk9.bmstu.ru)

## Настроить Jenkins -> Управление плагинами

установить [Subversion Plugin](#)

## Настроить Jenkins -> Manage Users

Добавить пользователей по вкусу с паролями

---

Далее будем создавать сборки

### Новая задача

пишем имя - my\_build\_name

выбираем тип - **создать задачу со свободной конфигурацией**

Нажимаем - **ОК**

ДАЛЕЕ

Сколько дней хранить результаты сборки - 3

**Управление исходным кодом** URL репозитория вводим

Первый раз SVN попросит пароль

**Триггеры сборки** - Опрашивать SCM об изменениях - 0 7 \* \* \*

**Выполнить команду shell**

```
pwd
ls -a
if [ -d build ]; then echo "build directory has already existed"; else mkdir build; fi
cd build
cmake -D BOOST_ROOT=/home/evgeny/software/boost-install ../trunk
make
make test
cd ..
```

```
#cd $(ls | grep -v build)
python trunk/scripts/python/test.py
```

**некоторые пути могут отличаться - их нужно поправить например директория буста или путь к скрипту test.py**

Уведомление по почте - тут по вкусу

Нажимаем - сохранить

-----

Теперь все должно работать. Выходим в главное меню и пробуем запускать сборку.