**Софийски Университет**

**„Св. Климент Охридски”**

Софтуерни Технологии 2010

*Проект на тема*

*„АвтоматизираноТестване с Ruby on Rails”*

Компютърни Науки

Трети Курс

Иван Стоилов

Мартин Минчев

Сава Сертов

Tодор Петров

Съдържание

[Увод в тестването 3](#_Toc264928358)

[Защо тестването е важно 3](#_Toc264928359)

[„Ръчно” срещу „Автоматизирано” тестване 3](#_Toc264928360)

[Ruby on Rails (RoR) 3](#_Toc264928361)

[Кратко представяне на RoR 3](#_Toc264928362)

[Архитектура на приложенията създадени с RoR 4](#_Toc264928363)

[Модел (Model) 4](#_Toc264928364)

[Изглед (View) 4](#_Toc264928365)

[Контролер (Controller) 4](#_Toc264928366)

[Увод в тестването с RoR 5](#_Toc264928367)

[Процес на тестване 5](#_Toc264928368)

[Генериране на скелет на приложение 5](#_Toc264928369)

[Тестов режим на работа 6](#_Toc264928370)

[Реализация на тестването с RoR 6](#_Toc264928371)

# Увод в тестването

## Защо тестването е важно

Разработването на софтуер е сложна интелектуална дейност. Ето защо грешките, допускани при разработката на програми, са неизбежни, независимо от опита, който имат проектантите и програмистите. Не съществува методология на програмиране, която да гарантира, че създадените чрез нея програми са безгрешни. Чрез тестването се прави опит да се въведе стил на изграждане на софтуер, при който както отделният програмист, така и целия екип допускат по-малко грешки. Презумпцията, че всяка новосъздадена програма съдържа грешки налага щателна проверка преди да се пристъпи към използването й.

Тестването е специална процедура, с изпълнението на която се цели откриване на грешки в програмите. То е едно от най-ефикасните неща, които могат да се направят за да се подобри качеството на софтуера. Честото тестването на софтуера е от изключителна важност, защото помага за ранното откриване на грешки в кода, което спестява много бъдещи разходи и усилия.

## „Ръчно” срещу „Автоматизирано” тестване

Практиката показва, че повечето разработчици тестват своите приложения. Проблемът е, че някои от тях тестват ръчно, а не автоматизирано. Недостатъкът на ръчното тестване е, че при всяка промяна по софтуера, тестващият трябва да повтаря едни и същи стари тестове, за да е сигурен, че новата добавка не поврежда старата функционалност. При една такава монотонна и повтаряща се дейност, възможността за човешка грешка или пропуск става много голяма. Когато големината на софтуера нараства, тази вероятност се увеличава още повече. Имено за това е препоръчително тестването да става по автоматизиран начин, тоест тестовете да се изпълняват от програма.

# Ruby on Rails (RoR)

## Кратко представяне на RoR

Ruby on Rails (често съкращавано като Rails или RoR) е популярна платформа за разработка на уеб приложения, написана изцяло на програмния език Ruby. Характерно за RoR е, че дава на разработчиците изключителна бързина и гъвкавост при изграждането на уеб приложения. Мотото на RoR е „Web development that doesn’t hurt!”.

## Архитектура на приложенията създадени с RoR

RoR налага архитектурата Model-View-Controller (съкратено MVC) върху всички приложения създавани чрез него. Благодарение на това бизнес логиката, данните и потребителският интерфейс в тях са добре изолирани.

### http://betterexplained.com/wp-content/uploads/rails/mvc-rails.png

### Модел (Model)

Моделите олицетворява данните, с които работи приложението и правилата за достъп до тях. За нуждите на RoR, моделите основно се използват за да управлява взаимодействието на приложението с база от данни. В повечето случаи един модел отговаря на една таблица в базата данни.

### Изглед (View)

Изгледите представляват потребителския интерфейс на приложението. В RoR, изгледите са основно HTML файлове с внедрен Ruby код, в които изпълнението на Ruby е свързано с изобразяване на данните. Главната задача на изгледите е да са посредник на информация между потребителя и останалата час от приложението.

### Контролер (Controller)

Контролерите представляват връзката между изгледите, моделите и потребителя. Работата им се състои в това да получават заявки от потребителя, да извличат данни от моделите, да ги обработват ако е необходимо и накрая да изпращат данните на изгледите.

# Увод в тестването с RoR

Автоматизираното тестването е заложено в основите на RoR. То е силно препоръчително и е почти неотлъчна част от всяко приложение, разработено с RoR. Основната идеята на тестването е проста: написва се код, който проверява дали уеб приложението работи според очакванията. Веднъж написани, тестовете могат да се изпълняват многократно, всеки път когато има нужда от тях. По този начин процесът се автоматизира.

В RoR присъстват три основни вида тестове, като всеки от тях е насочен към различна част от дадено приложение.

* Unit тестове – използват се за тестване на моделите
* Функционални тестове – използват се за тестване на функционалността на контролерите и изгледите
* Интеграционни тестове – използват се за тестване на взаимодействието между отделните контролери
* Тестове на производителността – използват се за да се тества производителността на приложението

# Процес на тестване

## Генериране на скелет на приложение

При създаване на скелет на ново приложение, RoR автоматично създава директория - *test*. Нейното предназначение е да съхранява всички файлове и директории, свързани с автоматизираното тестване. Тя има следното съдържание:

* test
* fixtures - съдържа файлове с тестови данни
* functional - съдържа файлове с Функционални тестове
* integration - съдържа файлове с интеграционни тестове
* performance – съдържа файлове с тестове на производителността
* unit - съдържа файлове с unit тестове
* test\_helper.rb – съдържа настройки на тестовете

## Тестов режим на работа

Всяко приложение разработено с RoR имa 3 основни режима, в които може да работи – *production*, *development* и *test*.

Режимът „*test*” е предназначен специално за тестване. Този режим на работа изисква специална база данни, различна от основната за приложението. Причината за това е, че често при тестване се откриват грешки и ходът на програмата не е изцяло предвидим, което води до голям риск от заличаване на данни. Именно затова се използва друга база от данни, в която няма ценни данни.

# Реализация на тестването с RoR

Модулите са широко използвани в езика за програмиране Ruby. Те много приличат на класове с изключение на две неща: не могат да се създават обекти от тип даден модул и модулите не могат да бъдат наследявани. Модулите се ползват за две основни неща. Първото е, за да се групират методи, класове, константи и др. на едно място. По този начин те се капсулират и изолират, с което се предотвратяват евентуални противоречия между тях и останалия код. Второто е, за да се реализира вид множествено наследяване, което по принцип не е заложено в езика. То се получава, когато клас включи в дефиницията си даден модул. Тогава всички елементи на модула стават част от класа.

Сега ще се спрем по-подробно на класовете и модулите, който имат връзка с тестването в RoR. Тестовете в RoR се създават като самостоятелни класове, който наследяват определени класове от ядрото на RoR. По този начин те се сдобиват с допълнителна функционалност, която е от голяма полза за изграждането им.

* Unit тестовете наследяват класа TestCase, който се намира в модула ActiveSupport
* Функционалните тестовете наследяват класа TestCase, който се намира в модула ActionController
* Интеграционните тестовете наследяват класа IntegrationTest, който се намира в модула ActionController
* Тестовете на производителността наследяват класа PerformanceTest, който се намира в модула ActionController

От своя страна класовете *ActionController::TestCase*, *ActionController::IntegrationTest*, *ActionController::PerformanceTest* наследяват класа *ActiveSupport::TestCase*. Той пък включва в себе си няколко модула и е наследник на класа *Test::Unit::TestCase*. По този начин можем да проследим цялата йерархия на класовете и модулите, свързани с тестването в RoR. На следващата диаграма е илюстрирана една част от тази йерархия.



Заслужава си да споменем модула *Test::Unit*. Той представлява платформа за unit тестване, създадена за езика Ruby. В последствие, с развитието на RoR, този модул се е превърнал в основа, върху която да стъпят тестовете и тестването в новата платформа. Класът *Test::Unit::TestCase* има обединяваща роля за *Test::Unit*. В него се събира цялата функционалност на модула. Една от най-важните функционалности, които той включва е тази на модула *Test::Unit::Assertions*. *Test::Unit::TestCase* се наследява от класа *ActiveSupport::TestCase*, който има централна роля в йерархията на класовете и от когото всички видове тестове в RoR наследяват функционалност по един или друг начин.