# Analyzing duplication on code generated by Scaffolding frameworks for Graphical user interfaces

André M. Andrade
Crateús Campus
Fed. Univ. of Ceará
andre@crateus.ufc.br

Rodrigo A. Vilar
Exact Sciences Department
Fed. Univ. of Paraíba
rodrigovilar@dce.ufpb.br

Anderson A. Lima, Hyggo Almeida, Angelo Perkusich
Embedded Systems and Pervasive Comp. Lab.
Fed. Univ. of Campina Grande
anderson.lima,hyggo,perkusic@embedded.ufcg.edu.br

## Abstract

*Scaffolding is an approach used by some modern web frameworks in order to generate an initial version of applications code based on domain model meta data. Since this temporary code should be customized by programmers to implement real systems, its quality metrics are important aspects. In this paper, a methodology is proposed and applied in order to relate domain model size and a quality metric — amount of duplicated code — focusing on Graphical user interface implementation. Results show that code duplication grows at least linearly with the growth of the number of entities in domain model. There are also some scenarios where quadratic proportions were found. These observations suggest that, for large domain models, code quality and its evolution would be affected when scaffolding frameworks are used.*

***Keywords*** - Code generation, Scaffolding frameworks, Code duplication, Meta data

## 1 Introduction

Automatic generation is an approach that has been widely used on modern web frameworks, e.g. Ruby on Rails, Grails, Yeoman and Spring Roo[1]. These frameworks implement a technique, called Scaffolding [13], with algorithms and templates that generate application initial code and configuration, based on domain model meta data. After generation, developers evolve and maintain source code in order to implement real applications. In doing so, programmers would expect that Scaffolding frameworks produce code with good quality metrics, to ease their work. This means that code smells, like duplication and coupling, should be avoided by scaffolding algorithms.

In this paper, an experiment was designed, performed and analyzed to measure duplication on code produced by two popular web frameworks, Ruby on Rails and Spring Roo, specifically for Graphical User Interfaces (GUI), which represents a great amount of development effort on Enterprise Applications [11, 10]. This study is part of a research project aiming at understanding and enhancing meta data usage on Enterprise Application development. In this case, the proposed experiment design is an innovative approach to analyze how duplication behaves on generated code as software complexity grows.

The results suggest that, in spite of giving advantages for project initial steps like configuration and ramp-up, using scaffolding frameworks can lead to problems on code evolution and maintainability, when applications have complex domain models. Linear and quadratic proportions between domain model size and duplicated code amount were found within this experiment scope, indicating that code quality in applications with vast domain models would degrade.

In Section 2, Scaffolding frameworks and Duplicated Code detection are rationalized. Section 3 details Experiment design for this study, whose results were analyzed in Section 4. Finally, conclusions and future work proposals are presented in the last Section.

## 2 Background

This section gives an overview of Scaffolding frameworks and duplicated code detectors.

[1]rubyonrails.org, grails.org, yeoman.io, projects.spring.io/spring-roo

DOI: 10.18293/SEKE2017-164

### Scaffolding Frameworks

Several modern frameworks generate code automatically based on domain model features, such as, entities, properties, relationships and other meta data [7]. As its name suggests, code generated by Scaffolding frameworks is not designed to be definitive. It is only a first functionality draft that should be polished in order to produce the desired functionality.

Among several frameworks that use Scaffolding, such as, Spring Roo, Play Framework, Apache Tapestry, ASP.NET Dynamic Data, Ruby on Rails, CakePHP, Django and Yeoman, two technologies have been chosen for this work, in order to analyze code duplication on scaffolded applications: Ruby on Rails (RoR) and Spring Roo (Roo). The key factors to choose these frameworks were: relevance due to industrial and worldwide adoption, according to the BuildWith's Framework Usage Statistics [2]; coverage of different language types (static and dynamically typed languages); and availability of code duplication detection tools, in order to operate this experiment.

Ruby on Rails (RoR) allows users to build features quickly using the `generate scaffold` command to generate the application code [6]. Despite being very seductive because of facility and quickness, "the complexity and sheer amount of code in the scaffolding can be utterly overwhelming to a beginning Rails developer" [6]. Generated code consists of Ruby language files (.rb) for back-end and Embedded Ruby files (.erb) for front-end view.

Spring Roo is a domain-driven development framework [8] that uses Java, Spring, AspectJ and Maven[2], combining Java annotations with shell commands to generate all application layers. Roo uses Java on back-end and Java Server Pages XML compliant files (.jspx) on front-end view pages.

### Duplicated Code Detectors

Duplication is a strong evidence of high coupled and poor reusable source code. According to Fowler, code duplication represents a conceptual coupling [4], because maintenance on copied code must be done in all code copies. Lee, Barta and Juliff also observed that high coupling and code duplication increase effort for system maintenance [9], including on enterprise applications.

There are several techniques to detect duplicated code in software projects: text-based, token-based, metric-based, AST-based (Abstract Syntax Trees), PDG-based (Program Dependence Graphs) [1]. Two of those techniques were used in this research: AST-based that parses program code into an abstract syntax tree, divides it into sub trees and marks common sub trees as code clones; and text-based that compares every line of code as a string and marks code clones based on the similarity between the text fragments [1].

For Ruby on Rails, three duplicated code detection tools were found: Towelie, Flay and Simian[3]. Flay, which is based on AST, presented better results and is able to process both .rb and erb. files. Therefore Flay was chosen to analyze Ruby on Rails code.

For Spring Roo applications, PMD-CPD[4], which uses String matching, was chosen, because it is one of the most popular static code analysis tool for Java projects. It is able to detect code duplication in Java and JSP files. Since, PMD-CPD does not work with AspectJ files, all aspects code was moved to Java classes, in order to check its duplication. This operation is common for Spring Roo and is performed through a simple command.

## 3. Experiment Design

In order to analyze duplication on code generated by Scaffolding frameworks, several experiment units have been designed. Essentially, this experiment controls two input variables — domain model characteristics and technology (Ruby on Rails or Spring Roo) — and generates values for one output variable: amount of duplicated code.

### Domain Model Scenarios

To control domain model input variable, six scenarios groups were prepared. Table 1 details domain models structure used by each scenario group, specifying quantity of entities and properties. Besides that, scenarios groups explore another domain models features such as property names and types. The most complex scenario is composed by six entities, whose size is similar to small, but useful, systems like: a project management application containing projects, workers, activities, time appointment and artifacts; or a purchase control system containing costumers, vendors, products, sale and sale item.

Scenario groups use a variable that ranges from 2 to 6 (except on scenario F that varies from 1 to 6). In doing so, each scenario group produces several scenario instances as inputs for the experimental units. Therefore, this experiment has 31 scenarios instances that were evaluated using Ruby on Rails and Spring Roo.

In a scenario group, the influence of one domain model feature over code duplication can be tested. In scenario group A, for instance, the number of entities (without properties) varies from 2 to 6. So, code generated by scaffolding frameworks can be analyzed for each scenario instance,

[2]spring.io, eclipse.org/aspectj, maven.apache.org

[3]github.com/gilesbowkett/towelie, github.com/seattlerb/flay, www.harukizaemon.com/simian/

[4]pmd.sourceforge.net/pmd-4.3.0/cpd.html

and code duplication amount can be related to number of entities. In scenario group F, this same procedure can be repeated to relate duplication amount to number of properties in two entities.

| $G$ | **Description** | **Variable meaning** |
|---|---|---|
| **A** | N Entities without properties | N:Number of entities |
| **B** | N Entities with 1 distinct property | N:Number of entities |
| **C** | N Entities with 1 property with same name | N:Number of entities |
| **D** | N Entities with 1 property with same type | N:Number of entities |
| **E** | N Entities with 1 property with same name and type | N:Number of entities |
| **F** | 2 Entities with M distinct properties | M:Number of properties |

**Table 1. Experiment scenario groups**

**Experiment execution**

For each experiment unit, which received as input a scenario instance ($si$) and a scaffolding framework ($sf$), the following steps were executed: run $sf$ shell commands to generate code concerting $si$; measure code duplication (with Flay for RoR and CPD for Roo); and register the amount of duplicated code.

It was required some customization in order to tune duplicated code detection. Flay and CPD define minimum threshold values in order to take into account duplicated code blocks. There are default values ($MassThreshold = 18$ on Flay[5] and $DuplicateChunkSize = 100$ on CPD[6]), which can be customized according to user needs.

After running experimental units with $MassThreshold = 18$ for Ruby on Rails and Flay, several relevant duplicated blocks were not detected. Therefore, for this experiment, threshold values were decreased ($MassThreshold = 4$ on Flay and $DuplicateChunkSize = 40$ on CPD), in an effort to maximize code duplication detection.

Measurement units for Flay and CPD vary due to its distinct clone detection approaches. Flay measure similar ASTs by its *mass*. CPD compares repeated string *tokens*. This difference is not a problem because this experiment goal is not to compare RoR and Roo quality. In both cases, the objective is to relate code duplication (either by mass or by token) with domain model size.

[5]docs.codeclimate.com/docs/duplication

[6]maven.apache.org/plugins/maven-pmd-plugin/cpd-mojo.html

## 4 Result Analysis

In order to organize data results from experiment, several aforesaid factors were used: technology (RoR or Roo); scenarios groups; number of entities (N); and number of properties by entity (M). There is also another determinant to group duplicated code amount: layer. It is because both Ruby on Rails and Spring Roo use the MVC pattern [5] and, since this work focuses on GUI, view and controller layers should be considered. However each layer needs to be evaluated separately because its code uses distinct languages.

| $G$ | **Variation** | | **Duplicated Code amount** | | | |
|---|---|---|---|---|---|---|
| | N | M | RoR | | Roo | |
| | | | view | contr | view | contr |
| A | 2 | 0 | 1006 | 308 | 3010 | 3311 |
| | 3 | 0 | 1956 | 462 | 3010 | 5237 |
| | 4 | 0 | 3206 | 616 | 3010 | 7005 |
| | 5 | 0 | 4756 | 770 | 3010 | 8773 |
| | 6 | 0 | 6606 | 924 | 3010 | 10830 |
| B | 2 | 1 | 1314 | 312 | 8273 | 3311 |
| | 3 | 1 | 2580 | 468 | 10448 | 5237 |
| | 4 | 1 | 4254 | 624 | 12495 | 7005 |
| | 5 | 1 | 6336 | 780 | 14796 | 8773 |
| | 6 | 1 | 8826 | 936 | 16843 | 10830 |
| C | 2 | 1 | 1314 | 312 | 8273 | 3311 |
| | 3 | 1 | 2580 | 468 | 10448 | 5237 |
| | 4 | 1 | 4254 | 624 | 12495 | 7005 |
| | 5 | 1 | 6336 | 780 | 14796 | 8773 |
| | 6 | 1 | 8826 | 936 | 16843 | 10830 |
| D | 2 | 1 | 1314 | 312 | 8273 | 3311 |
| | 3 | 1 | 2580 | 468 | 10448 | 5237 |
| | 4 | 1 | 4254 | 624 | 12495 | 7005 |
| | 5 | 1 | 6336 | 780 | 14796 | 8773 |
| | 6 | 1 | 8826 | 936 | 16843 | 10830 |
| E | 2 | 1 | 1314 | 312 | 8273 | 3311 |
| | 3 | 1 | 2580 | 468 | 10448 | 5237 |
| | 4 | 1 | 4254 | 624 | 12495 | 7005 |
| | 5 | 1 | 6336 | 780 | 14796 | 8773 |
| | 6 | 1 | 8826 | 936 | 16843 | 10830 |
| F | 2 | 1 | 1314 | 312 | 8273 | 3311 |
| | 2 | 2 | 1382 | 314 | 8775 | 3311 |
| | 2 | 3 | 1450 | 316 | 9747 | 3311 |
| | 2 | 4 | 1518 | 318 | 11516 | 3311 |
| | 2 | 5 | 1586 | 320 | 13974 | 3311 |
| | 2 | 6 | 1654 | 322 | 16970 | 3311 |

**Table 2. Experiment results**

On Table 2, where resulting data is shown, each line defines a scenario instance, and columns represent, respectively, scenario group ($G$), number of entities ($N$), number of properties ($M$) and amount of duplicated code for: Ruby

| G | Ruby on Rails and Flay | | | | Spring Roo and CPD | | | |
|---|---|---|---|---|---|---|---|---|
| | View | | Controller | | View | | Controller | |
| | Model | $R^2$ | Model | $R^2$ | Model | $R^2$ | Model | $R^2$ |
| A | $150N^2 + 200N + 6$ | 1 | $154N$ | 1 | 3010 | 1 | $1857.4N - 398.4$ | 0.999 |
| B | $204N^2 + 246N + 6$ | 1 | $156N$ | 1 | $2148.8N + 3975.8$ | 0.999 | $1857.4N - 398.4$ | 0.999 |
| C | $204N^2 + 246N + 6$ | 1 | $156N$ | 1 | $2148.8N + 3975.8$ | 0.999 | $1857.4N - 398.4$ | 0.999 |
| D | $204N^2 + 246N + 6$ | 1 | $156N$ | 1 | $2148.8N + 3975.8$ | 0.999 | $1857.4N - 398.4$ | 0.999 |
| E | $204N^2 + 246N + 6$ | 1 | $156N$ | 1 | $2148.8N + 3975.8$ | 0.999 | $1857.4N - 398.4$ | 0.999 |
| F | $68M + 1246$ | 1 | $2M + 310$ | 1 | $328.8M^2 - 563.1M + 8526.4$ | 0.999 | 3311 | 1 |

**Table 3. Polynomials models found for each scenario group, framework and layer**

on Rails (RoR) view and controller; and Spring Roo (Roo) view and controller.

This experiment executed 62 experimental units, which were designed after combining all possible input variables: 31 domain model scenarios (explained in Section 3); and 2 technologies (Ruby on Rails and Spring Roo).

### Data synthesis

Some function models were studied in an attempt to express, based on experiment results, a relation between amount of duplicated code and domain model complexity, that is number of entities ($N$) and properties ($M$). Therefore, polynomial regression model was used to find the best mathematical function composed of an $n$-degree polynomial that fits the relation between $CD$ and $M$ or $N$, such as:

$$CD \approx f(N) \text{ or } CD \approx f(M)$$

where function $f$ is a polynomial with degree $n$.

Polynomial model reliability is analyzed from residuals that define a coefficient of determination ($R^2$) [3], which reflects the correlation between the resulting samples from $CD$ and the values of $M$ and $N$. The closer this number is to 1, the better is the polynomial model.

$CD$ function models have the following structure:

$$CD_{g,s,l}(N) = f(N) \text{ or } CD_{g,s,l}(M) = f(M)$$

where $g$ is scenario group (A, B, C, D, E or F); $s$ is scaffolding framework (RoR or Roo); $l$ is layer (Controller or View); $N$ is number of entities; $M$ is number of properties by entity; $f(N)$ and $f(M)$ are $n$-degree polynomials. The code duplication detection tool – Flay or CPD – was not considered as a parameter since they do not vary in the same framework.

For each executed combination of framework, scenario group and layer, the polynomial that best fits to $CD$ samples was chosen, according to $R^2$ value, and having lower degree. Table 3 shows the polynomials chosen for each scenario group. Considering same combination of framework and layer, the B, C, D and E scenarios showed the same polynomial model, therefore, their graphs are represented in the same line on the Table 4, which represents graphically the experiment results.

### Interpretation

Based on the polynomial models found and their related graphs, some observations can be defined, within this experiment scope:

1. Code duplication in Ruby on Rails view layer varies in quadratic proportion to number of entities and in linear proportion to number of properties, with perfect models found;
2. Code duplication in Ruby on Rails controller layer varies in linear proportion to both number of entities and number of properties, with perfect models found;
3. Code duplication in Spring Roo view layer is constant for any number of entities without properties (scenario group A), with a perfect model found;
4. Code duplication in Spring Roo view layer vary in linear proportion to number of entities (with properties) and in quadratic proportion to number of properties, without perfect models found;
5. Code duplication in Spring Roo controller layer is in linear proportion to number of entities, without perfect models found;
6. Code duplication in Spring Roo controller layer is constant for any number of properties in two entities (scenario group F), with perfect a model found;
7. For Ruby on Rails all models found have $R^2 = 1$ i.e., polynomials have high quality to represent experiment scenarios and to possible predict code duplication for greater domain models;
8. For Spring Roo all non constant models have $R^2 = 0.999$. In fact, several polynomials would represent this relation with $R^2 = 0.999$ and the one with lowest

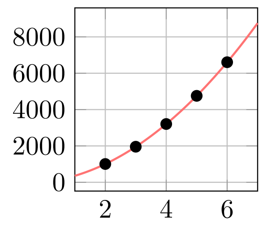
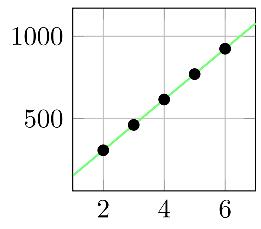
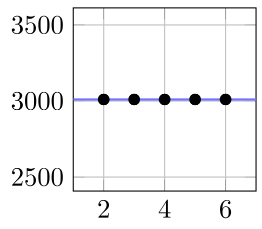
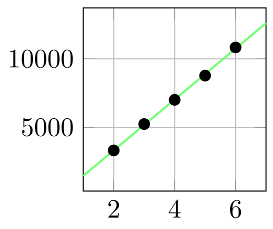
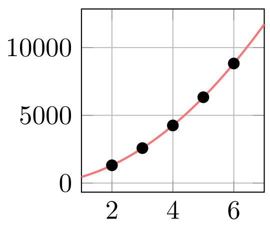
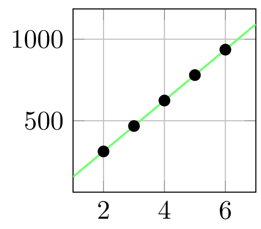
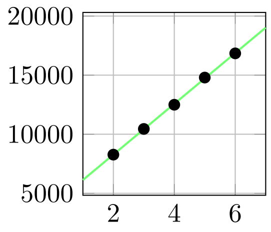
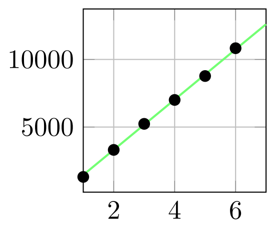
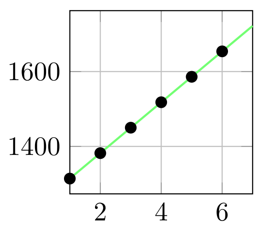
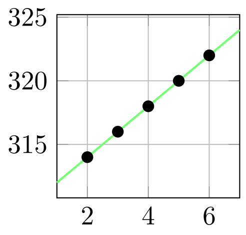
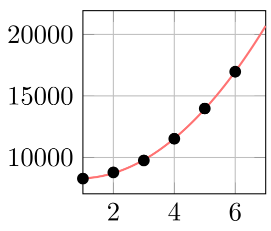
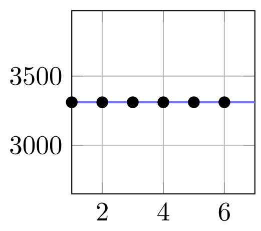
| $G$ | **RoR – View** | **RoR – Controller** | **Roo – View** | **Roo - Controller** |
|---|---|---|---|---|
| A |  |  |  |  |
| B-E |  |  |  |  |
| F |  |  |  |  |

**Table 4. Polynomials plots for all scenario groups**

degree was chosen. In spite on being very good models, they can contains errors and should be validated in other experiments with greater number of samples.

Ideal frameworks should generate code without duplication or, at least, with a constant amount of duplicated code. In this experiment, two observations define constant relations, however these observations are improbable to occur on real systems. Observation 3 represents entities without properties, which is a strange scenario for real projects, because every entity should have properties. Observation 6 takes place on systems with only two entities. In spite of being very small and improbable systems, this result is useful because it indicates that number of properties does not influence code duplication in Spring Roo controllers.

After combining all observations, this experiment shows that code duplication grows in linear or quadratic proportion to number of entities or properties, for all scenarios, considering view and controller layers together.

These relations predicts that, for two completely different scaffolding frameworks, generated code is duplicated in a significant amount. At least, code duplication grows in a linear proportion to domain model size.

Models with $R^2 = 0.999$ have several polynomial models to represent code duplications on units of this experiment. However, in this work, it is not possible to define the most appropriate function. Therefore, replications of this experiment could be made to obtain clearer conclusions for scenarios with N and M greater than 6.

## Threats to validity

It's necessary to consider the risk of the many generalizations found here: from small domain models to large domains; from two scaffolding frameworks to all scaffolding frameworks; and from two duplicated code detection tools to all detection tools.

Due to time and resource restrictions, only small domain models were analyzed. However a great effort was made in this work to create an analysis methodology that can be easily reproduced. In doing so, the observations found here can be tested for larger domains and with other technologies.

In spite of using few scaffolding frameworks, the chosen frameworks use different approaches, language types and clone detection techniques. These differences enhance results strength.

There is also a lack of scenarios that increase both entities and properties quantity. In doing so, the impact of number of entities and properties combined was not tested in this work.

**Packaging**

All experiment code and scripts are available on two open source repositories, one for Ruby on Rails[7] and the other for Spring Roo[8]. Each scenario group is organized into separated branches and there is a commit for each experiment unit.

These repositories contains a README file on master branch that explain how to reproduce and expand this experiment.

## 5 Conclusion

This paper presents an approach that is, as far as we know, innovative to relate domain model complexity and duplication on code generated by scaffolding frameworks.

For domain models with up to six entities and properties, duplicated code results could be generalized into polynomials, which lead to two main conclusions: (i) code duplication increases, at least, at linear proportion to domain model size, on GUI generated by two scaffolding frameworks (Ruby on Rails and Spring Roo); and (ii) for models that fitted on experiment results with high quality, it could be expected that the same models would work for greater domain models.

The consequence of conclusion (i) is that duplicated code on applications generated by scaffolding framework can increase to a severe amount when it has complex domain models, and this increase would inhibit code evolution.

Conclusion (ii) can be tested in bigger scenarios and, if it is confirmed, the polynomials found by this work would be used to estimate code duplication, on code generated by Ruby on Rails and Spring Roo, for applications with complex domain models.This estimation would also help software architects to choose the scaffolding framework that will be used in development projects.

As future works, we suggest to replicate this experiment with other Scaffolding frameworks, such as, Grails (Groovy), django (Python), CakePHP, ASP.NET; and with bigger domain models.

Besides that, for frameworks that easily expose its templates, such as, Ruby on Rails, asymptotic computational complexity analysis on code generation algorithms could be compared to our experiment results, in order to confirm or deny them.

In order to analyze the most popular GUI frameworks approaches, identify its pros and cons related to software maintenance, and propose better solutions, this research is the second part of a bigger work that had first results published at 2015 by our coauthor Rodrigo Vilar. Vilar [12] already showed good results by using some rendering patterns to provide reuse on GUI code. As next step, we intend to present a framework proposal and a comparative analysis with popular GUI frameworks considering the relation between complexity, productivity and maintainability.

[7] github.com/Andersoonalves/Rails_Duplicate_Code_Analysis

[8] github.com/rodrigovilar/RooCodeDuplication

## References

[1] Y. L. Aritra Ghosh. An empirical study of a hybrid code clone detection approach on java byte code. *GSTF Journal on Computing (JoC); Singapore*, 5(2):34–45, 2017.

[2] BuildWith. Framework usage statistics: Statistics for websites using framework technologies. *https://trends.builtwith.com/framework*, 2017.

[3] G. C. R. Douglas C. Montgomery. *Applied Statistics & Probability for Engineers*. Wiley, 3 edition, 2002.

[4] M. Fowler. Reducing coupling. *IEEE Software*, (4):102–104, 2001.

[5] M. Fowler. *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[6] M. Hartl. *Ruby on Rails Tutorial: Learn Web Development with Rails*. Addison-Wesley Professional, 2016.

[7] J. Herrington. *Code generation in action*. Manning Publications Co., 2003.

[8] S. M. Josh Long. *Getting Started with Roo. Rapid Application Development for Java and Spring*. O'Reilly, 2011.

[9] M. Lee, B.-Z. Barta, and P. Juliff. *Software quality and productivity: theory, practice, education and training*. Springer, 2013.

[10] G. Meixner, G. Calvary, and J. Coutaz. Introduction to model-based user interfaces - w3c working group note 07 january 2014.

[11] B. A. Myers and M. B. Rosson. Survey on user interface programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 195–202. ACM, 1992.

[12] D. O. Rodrigo Vilar and H. Almeida. Rendering patterns for enterprise applications. In *Proceedings of the 20th European Conference on Pattern Languages of Programs*, EuroPLoP '15, New York, NY, USA, 2015. ACM.

[13] D. Thomas and D. Hansson. *Agile Web Development with Rails*. Pragmatic Bookshelf, 2006.