An Agile Approach for Web Systems Engineering

Vítor Estêvão Silva Souza
Computer Science Department,
Federal University of Espírito Santo
Av. Fernando Ferrari, S/N
Goiabeiras – Vitória, ES
+55 (27) 3335-2134
vitorsouza@gmail.com

Ricardo de Almeida Falbo
Computer Science Department,
Federal University of Espírito Santo
Av. Fernando Ferrari, S/N
Goiabeiras – Vitória, ES
+55 (27) 3335-2167
falbo@inf.ufes.br

ABSTRACT
In the last few years, Web applications have evolved from static hypertext documents to complex information systems. This evolution leads to the necessity of methodologies specifically designed for development of Web-based systems, focusing on agility in the process. This paper presents an agile approach for the development of Web applications that applies the concept of agile modeling, adopts a standard software architecture and is heavily based on frameworks, speeding up system analysis, design and implementation.

Categories and Subject Descriptors
D.2.2 [**Sofware Engineering**]: Design Tools and Techniques – *Object-oriented design methods.*

General Terms
Documentation, Design.

Keywords
Web Engineering, Agile Development, Web Application, Frameworks, Rapid Development, Java.

1. INTRODUCTION
In the beginnings of the World Wide Web, the software infrastructure behind it supported static Web pages only, i.e., pure hypertexts organized in files that were delivered to browser clients upon requests handled by a Web server. From 1993 on, with the emergence of CGI (Common Gateway Interface) and Web programming languages such as PHP (94), ASP (95) and Java Servlets (96) / JSP (99), these Web servers became much more powerful, allowing programmers to develop software to run on them.

Soon enough, enterprise systems, such as online stores (B2C – Business-to-Consumer), or supply chain management (B2B – Business-to-Business), were being developed for the Web, taking advantage of their remote nature and ease of deployment. Any computer connected to the Internet with a browser installed could use a Web-based system and usually there is no need for software installation on the client side.

WebMedia'05, December 5–7, 2005, Poços de Caldas, Minas Gerais, Brazil.

First generation Web applications (WebApps) were usually developed in an ad hoc manner, with no concern for Software Engineering principles. However, to develop increasingly complex WebApps nowadays, the adoption of an engineering approach is imperative. Web Engineering (WebE) borrows many conventional Software Engineering's fundamental concepts and principles. In addition, it incorporates specialized process models, software engineering methods adapted to the characteristics of this kind of application, and a set of enabling technologies [1].

WebE is relatively new and, thus, a broad and fertile field for research. For example, technologies for codifying WebApps are evolving very quickly. Several frameworks are becoming available, especially if we consider popular technologies, such as Java, .NET and PHP. Once learned, these frameworks make the development much faster and more productive. This makes them very attractive, especially because agility is very important for the development of WebApps [1].

In this context, the Software Engineering Lab (LabES) of the Federal University of Espírito Santo (UFES) is working on an approach for rapid development of Java WebApps, based on Agile Modeling [2] and the existence of many free and open-source Java frameworks. This paper presents the proposed approach (section 2) and what we're planning for future work (section 3).

2. RAPID DEVELOPMENT OF WEBAPPS USING JAVA AND FRAMEWORKS
As with conventional software engineering, the WebE process starts with the identification of the business needs, followed by project planning. Next, requirements are detailed and modeled taking into account the analysis and design perspective. Then the application is built using tools and methods specialized for the Web. Finally, the system is tested and delivered to end-users [1]. Four activities of this generic process framework deserve more attention in the context of the approach presented in this paper:

- Requirement Specification: a document defining the project scope and presenting the use case diagrams and their descriptions is written, documenting all system functionalities;
- Analysis: to better understand and describe the business domain, structural and behavioral models are built based on the requirements;

- Design: once the implementation platform is chosen, system architecture must be defined. In the approach presented here, a standard architecture is given. For each package of this architecture, a class diagram should be developed. Also an user interface design activity should be performed, considering Web-specific aspects, such as HTML pages and the classes that integrate the current system with the frameworks used;
- Implementation: design models are transformed into source code using frameworks in order to reduce the coding effort.

These four activities form a core process framework that should be applied in an iterative fashion, allowing for user feedback and requirements and system evolution.

Requirements specification and analysis do not take the implementation platform into account, and, thus, are very similar to the corresponding activities in the conventional software engineering. However, since agility is desired, the principles of Agile Modeling should be followed, among them:

- Model with a purpose: the developer must have an specific goal in mind before creating a model;
- Use multiple models: each model should present a different aspect of the system and only those models that provide value to their audience should be built;
- Travel light: as work proceeds, keep only those models that will provide long-term value;
- Content is more important than representation: modeling should impart information to its intended audience.
- Know the models and the tools you use to create them.

Design, on the other hand, is heavily dependent on the implementation platform. During design the system architecture must be defined. The approach proposed here defends the use of a standard software architecture for WebApps, shown in figure 1. Moreover, for developing the models required to design those packages in detail we advocate the usage WAE, a Web Application Extension of UML, proposed in [3].

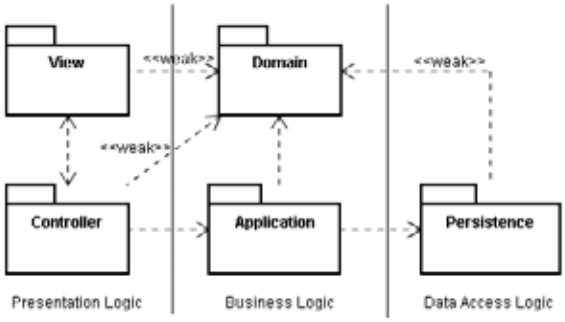

Figure 1. The proposed standard architecture for WebApps.

The presentation logic layer contains the classes that are responsible for the user interface, in this case Web pages (View package), and the classes that control the interaction (Controller package). The controller classes receive user input from the view, call system functions (Application package), and return the control to the view to display the results.

It is important to highlight that the dependency between controller and application is unidirectional, keeping the business logic layer independent of the presentation layer. The dependencies identified with the <<weak>> stereotype denote loose coupling between the packages. View and Controller, for instance, use domain objects (Domain package) only to display its data or to pass them around as parameters.

The business logic layer is also composed by two packages: Application and Domain. Domain classes represent the business domain concepts identified and modeled in class diagrams during requirement analysis and refined during design. Typically, these classes are very simple, resembling mere data structures most of the time (and thus commonly called “dumb objects”). The “intelligence” that is missing in these objects is set to the application classes, which maps to code what was defined as use cases in the requirement specification phase. An application class typically implements one or more use cases, and its methods implements the various flows of events described by the corresponding use cases. In this way, it is up to the application classes to create, retrieve, update and delete domain objects from the persistent media, according to the use case descriptions.

Finally, the data access logic layer is responsible for storing persistent objects in long-term duration media, such as databases. Given that the commonest scenario nowadays is the use of relational databases, we advocate the use of an object/relational (O/R) mapping framework along with the DAO (Data Access Object) design pattern [4]. Many of the existing O/R frameworks not only provide transparent persistence for objects configured by the developer, but they also generate the database schema, relieving us from the necessity of building relational models for the database logical and physical design.

The DAO pattern adds an extra abstraction layer, decoupling the data access logic layer from the persistence technology, allowing developers to change to another O/R framework, if needed. DAO classes are modeled during design stage, only making explicit the queries that can be made to retrieve objects from the database, since all DAO objects have, naturally, the ability to store and delete objects.

It is still possible to improve the interaction among the design components discussed previously using Dependency Injection [5] and Aspect Oriented Programming (AOP) [6]. With the former, dependencies among objects are declared in configuration files and a container is responsible for instantiating objects as they are needed and automatically link their dependencies. The latter allows us to separate cross-cutting concerns into aspects, which are written in one single place instead of being spread all over the source code. For both concepts there are frameworks available to support their adoption.

Finally, concerning the implementation phase, there is a great number of frameworks for WebApps development. They allow developers to focus on writing code for business logic instead of writing infrastructure code (sometimes referred to as “plumbing” code). In a case study following the proposed approach, we are developing a “lite” version of an existing

Cooperative Learning Environment, called AmCorA [7]. In this project, the following frameworks are being used:

- For the Object/Relational mapping framework, Hibernate was chosen and is used together with DAO classes to persist objects;
- The controller component is implemented with the support of WebWork, which also provides other features for validation of form fields, automatic type conversion, internationalization, and so on;
- In the view component, FreeMarker was chosen to build Web pages from templates. Using it Web designers can work in parallel with programmers, each focusing on their expertise;
- Also in the view component, a framework for the decoration of Web pages, SiteMesh, is used in order to keep a consistent layout through all pages;
- Covering all layers, Spring Framework performs dependency injection to integrate the different packages and also provides other services, such as automatic transaction management, using AOP;
- Integrated to Spring, the framework Acegi Security performs authentication and authorization services for components of both controller and application packages.

The frameworks above compose one of the many possible combinations of frameworks, since for each role there is more than one framework available. We intend to apply the proposed approach on various combinations of frameworks in order to prevent it from becoming over-fitted for this one.

3. CONCLUSIONS AND FUTURE WORK

From the case studies conducted using the proposed approach, we can conclude that the creation of agile software processes for WebApps development using frameworks is promising. The directives drawn for future work in this line of research include:

- Development of many web applications to refine the proposed approach;
- Use of several frameworks and evaluation of their impact in the proposed approach, generalizing it in some aspects or creating specific proposals for specific frameworks;
- In particular, we plan to study and evaluate the Java standards for the Java EE platform (JSF and EJB) in their new versions to be officially released soon and already available for testing.

4. REFERENCES

[1] R.S. Pressman, Software Engineering: A Practitioner's Approach, 6th edition, Mc Graw Hill, 2005.

[2] S. Ambler, "What Is Agile Modeling (AM)?", 2002, http://www.agilemodeling.com/index.htm.

[3] J. Conallen. Building Web Applications with UML. 2nd edition, Addison-Wesley, 2002.

[4] Core J2EE Patterns – Data Access Object. http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html.

[5] Inversion of Control Containers and the Dependency Injection Pattern. http://www.martinfowler.com/articles/injection.html.

[6] G. Kiczales, et al., Aspect-Oriented Programming. Proceedings of the European Conference on Object-Oriented Programming, 1997.

[7] Netto, H.V., Menezes, C.S., Pessoa, J.M. AmCorA: uma Experiência com Construção e Uso de Ambientes Virtuais no Ensino Superior, XIV Simpósio Brasileiro de Informática na Educação, Rio de Janeiro, 2003 (in Portuguese)