Use of Web Application Frameworks in the Development of Small Applications

Irena Petrijevcanin Vuksanovic, Bojan Sudarevic

National and University Library in Zagreb
Zagreb, Croatia
ivuksanovic@nsk.hr, bsudarevic@nsk.hr

Abstract - With the emergence of modern software development methodologies and related web application frameworks, many developers started using them because of their advantages, such as faster development, enhanced security, availability of useful and standardized libraries, simpler organization of work in development teams and clearer structure of code thanks to the strict conventions and use of design patterns that encourage separation of domain logic, user interface and data processing model. However, because of perceived disadvantages of frameworks - complexity and overhead of framework code, learning curve, possible undetected security vulnerabilities, etc. - most developers choose to use them only for development of large and complex applications, while they develop small applications from scratch. In this paper we compare development process of two versions of the same application – first developed in pure PHP, and second developed using CodeIgniter web application framework. Based on results of the comparison, we argue that, contrary to the common practice, use of web application frameworks is justified in development of small applications.

I. INTRODUCTION

Over the last two decades, many trends in the software development have emerged, among which the most prominent are:

- A growing number of applications, including complex applications such as office suites, are developed as web applications ("software as a service" model).
- Users expect access to the applications in early (alpha and beta) phases of development.
- Users expect new features to be added constantly and often.

As a result of these trends, development cycles have shortened, and the developers are under increasing pressure, since they have to satisfy conflicting interests: new version (with new features) should be released frequently, but at the same time application must remain stable and well structured. In some cases, the result is the prolonged beta phase, since the developers can not guarantee the stability of the application (example of this is Gmail, Google's webmail service that was more than five years in the beta phase).

As a solution for these problems, new software development methodologies that enable more dynamic organization of development process have emerged. On the technical side, the implementation of these methodologies is facilitated through modern web application frameworks.

However, these new solutions are usually used in large projects, while small applications are still often developed in a mostly non-structured manner.

II. SOFTWARE DEVELOPMENT METHODOLOGIES

The most prominent among the modern software development methodologies are Rapid application development and Extreme programming.

A. Rapid application development

Rapid application development is a set of practices and methods tailored toward [1]:

- Reduction of development schedules.
- Reduction of perceived development schedules by making progress more visible.
- Reduction of schedule volatility, thus reducing the chance of a runaway project.

Its main characteristic is minimal planning in favor of rapid prototyping. The planning and prototyping phases are interlaced with writing the software itself. This method enables more dynamic development process and faster achieving of basic software functionality, which is expanded with new features through iterative repeating of all phases.

B. Agile development and Extreme programming

Agile development is a collection of programming methodologies that follow principles defined in the Manifesto for Agile Software Development [2]:

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change following a plan.

Extreme programming, the most popular agile methodology, is geared toward eliminating requirements, design, and testing phases as well as the formal documents that go with them. Analysis, design, coding, testing, and even deployment phases should occur with rapid frequency [3].

Emphasis is also on independence of the team members that, after initially meeting at beginning of each phase, set their own individual goals and interact informally and only when needed.

III. WEB APPLICATION FRAMEWORKS

Software framework is an abstraction layer that provides software libraries that offer solutions for most common programming problems, with the goal of eliminating repetitive operations.

Web application frameworks are specifically geared toward development of web applications and services. They facilitate the use of the aforementioned software development methodologies.

A. Evolution of web development

In the early 1990s, most web pages were static HTML documents (with the exception of CGI applications that were usually written in Perl). In the mid-1990s new languages, such as ASP, ColdFusion and PHP, were developed specifically for use in the web. Shortly after that, first libraries aimed at solving common tasks specific for the web development (e.g. generating HTML) were created. Collections of these libraries can be considered as early Web-application frameworks.

In 2004, Ruby on Rails framework was released. Ruby on Rails is considered the most prominent web application framework of the latest generation [4]. It is written in, and works on top of, the Ruby language. Characteristics of Rails are “don’t repeat yourself” principle, “convention over configuration” concept and use of several architectural patterns, such as Model-View-Controller and Active Record. After Ruby on Rails, similar frameworks written in other languages followed, such as Django (Python), Catalyst (Perl), ASP.NET MVC (.NET languages) and Zend (PHP).

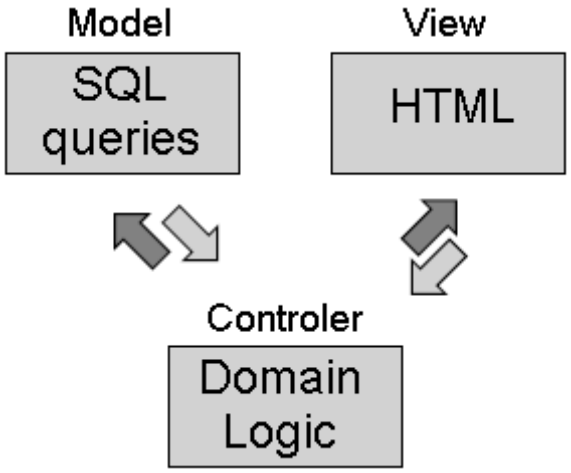

Figure 1. Model-View-Controller web application

B. “Don’t repeat yourself “ principle

“Don’t repeat yourself” principle is stated as "every piece of knowledge in a system should be expressed in just one place [5]. It is aimed at reducing repetition in software code, test plans, the build system, database schemas and documentation. This principle is implemented in frameworks through libraries aimed at solving the most common tasks, such as data validation, session and cookie management, file uploading, user authorization and authentication, etc.

C. “Convention over configuration” concept

Frameworks that follow "convention over configuration" concept are enforcing defaults in most aspects of application, e.g. class, method, variable, constant and database table names, file structure, coding style, etc.

Compliance with established conventions simplifies software development and code maintenance, especially when working in teams, as all team members follow the same rules. On the other hand, too strict enforcement of conventions threatens the flexibility of applications.

D. Architectural patterns

Architectural patterns are general reusable solutions to commonly occurring problems in software design. They offer well-established solutions to architectural problems, help to document the architectural design decisions and facilitate communication between developers through a common vocabulary [6].

Model-View-Controller pattern (Fig. 1) promotes separation of domain logic (controller), user interface (view) and data processing (model), as opposed to mixing HTML, SQL queries and domain logic in the source code (Fig. 2). In modern web application frameworks this pattern is usually implemented through folder structure. View files are responsible for showing data to the users of application. No programming logic or database queries can be run here, though data access may occur in these files. They are structured as HTML files and usually use a template language to present dynamic data, passed from the controller. Model files are responsible for fetching, modifying, inserting, and removing data from the database. Controller files calls and fetches data from the

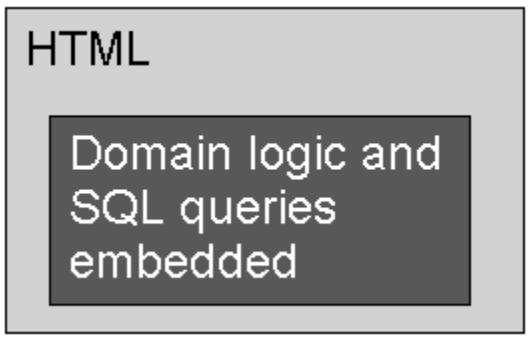

Figure 2. Non-structured web application

models, loads the data and passes it to the views, and sends the results to the user [7].

Second architectural pattern used in most modern web application frameworks is Active Record, used for accessing data in relational database. Table is wrapped into a class and an object instance is tied to a row in the table. Active Record class usually has methods that implement SELECT, INSERT, DELETE and UPDATE statements [9].

In most frameworks Active Record class supports all major databases, which enable easy change of database without modifications in queries, usually by changing one setting in the configuration file.

E. *Advantages and disadvantages of web application frameworks*

When starting a development of a web application, developers and project managers make certain decisions on the development process. One of those decisions is whether to use web application frameworks or to develop the application from scratch. Key advantages of web application frameworks include [8]:

- A complete environment for Web site development, interoperability, security, and maintenance so that developers do not have to build customized systems from the ground up every time they launch a new site.
- Standards, consistency, and predictability.
- Software components or building-blocks so that developers can share and reuse code.
- A model or standard architecture that allows easy visualization of how the entire system works.
- Reusable and thoroughly tested code in the libraries, classes and functions.
- Well-structured code using architectural patterns.

Key disadvantages of web application frameworks include:

- Complexity and overhead of framework code, in some situations visibly reduces application performance and creates greater burden for the underlying hardware.
- Security vulnerabilities in framework code affects applications built using it.
- High learning curve.
- Strict conventions hinder the application flexibility and developer's creativity.

Advantages clearly outweigh disadvantages when application that is being developed is large and complex and developed by a team. On the other hand, if the application is small and simple and developed by one developer or a small team, common perception is that the benefits that frameworks bring is not sufficient to justify their use.

Although such reasoning takes into account both advantages and disadvantages of frameworks, it obviously gives greater importance to disadvantages, as if advantages simply do not apply in the case of small applications. It also fails to consider factors such as maintenance and future development and possible growth of the application and intangible factors such as personal satisfaction of developers.

IV. EXAMPLE APPLICATION: ISVU2ALEPH

A. *Aleph implementation project*

Within the project of implementation of the new library management software, National and University Library in Zagreb and the libraries of 24 faculties of the University of Zagreb and 14 institutes, have moved to the Aleph Integrated Library System. Successful implementation depended in large part on the transfer of data and settings from the existing systems into the new one. While bibliographic data from most of the libraries was transferred relatively straightforwardly, the transfer of administrative data (including patron records) represented a greater challenge [10] since it is not standardized and varied greatly between existing systems.

In the case of the faculty libraries of the University of Zagreb, most of the patron (student) data was imported into Aleph from the Higher Education Information System (ISVU), network oriented modular system for data processing and interaction within the higher education system [11]. National and University Library has assumed the role of coordinator of the Aleph implementation process and, accordingly, the obligations related to, among others, training of staff, monitoring of the system, maintaining a support system for the users, analyzing data loads, ensuring data security, collecting and tracking of bibliographic data for conversion, as well as processing and importing patron data from ISVU to Aleph.

List of all students enrolled in the current academic year at the University of Zagreb is provided by ISVU to National and University Library as a text file, in which each line represents a record/student. Each record contains 13 delimiter-separated values, including student's name, surname, postal address, e-mail address, name and code of the faculty, the date when student status expires, and several ID numbers. Conversion of ISVU data into the format suitable for import into Aleph database, as well as the first phase of the import itself, was done by the application developed by National and University Library, ISVU2Aleph.

B. *ISVU2Aleph features*

Following features are supported in ISVU2Aleph:

- User authorization of the National and University Library staff member who perform data import.
- Loading the ISVU data into the application.
- Saving data into ISVU2Aleph database for later processing.
- Conversion to UTF-8.

- Detecting and fixing potential errors (e.g. broken lines).
- Editing data.
- Generating or manually adding and saving to database data that is not present in ISVU (other patron personal data, and data specific for the libraries and Aleph) but is necessary for the functioning of the library.
- Export of data from ISVU2Aleph database to XML file.
- Uploading XML file to Aleph server, where it can be imported into Aleph database through Aleph's Patron Loader Interface utility (PLIF).

C. *Technologies used in development of ISVU2Aleph*

ISVU2Aleph is a web application, written in the PHP language.

It was decided to develop ISVU2Aleph as a web application because of easy deployment – web browser and Internet connection are the only requirements and nothing has to be installed on the user's computer. PHP – scripting language originally designed to create dynamic web content – has been chosen mainly because of its flexibility and simplicity, although other advantages exists, such as support for all major operating systems and databases [12].

As a database, SQLite was used in the beginning, but was later replaced with MySQL. ISVU2Aleph is installed on the web server running Debian GNU/Linux operating system and Apache HTTP Server.

Finally, in the development of version 2.0, CodeIgniter, modern web application framework, written in PHP, was used. CodeIgniter supports Model-View-Controller architectural pattern, provides Active Record database abstraction layer with support for all major relational database systems, follows "Don't repeat yourself" principle by offering numerous useful classes and helpers and promotes "Convention over configuration" concept by offering (but not enforcing) set of default configurations.

D. *Development process*

Development of ISVU2Aleph was done during the summer of 2010. Two versions were developed in that period.

In June 2010 functionality and workflow of the application was defined by the manager of the Project team for implementation of Aleph. Version 1.0 was developed in June, in 19 hours of coding over three days, and was written in pure PHP, without the use of a framework, by a single developer. It used SQLite database as a backend. It included all the aforementioned features, with the exception of uploading XML file to Aleph server. Coding style was purely procedural, with PHP code and SQL queries embedded in the HTML code (the structure is identical to the one shown in Fig. 2).

Successful test of importing data from ISVU to ISVU2Aleph, and then exporting to XML file were performed in July 2010. XML file was then manually uploaded to Aleph server and imported into Aleph patron database using PLIF without errors.

After the test, it was decided to further simplify the procedure by enabling uploading of XML files through ISVU2Aleph. It was also decided to make a switch to MySQL database, because of its robustness and flexibility in comparison with SQLite.

Version 2.0 was developed in August 2010 by the same developer, in 11 hours of coding during one day. It was written in PHP, using CodeIgniter framework and purely object-oriented coding style. It was successfully tested in late August, and subsequently used for first production import in which data on 64842 students – patrons of the faculty libraries of the University of Zagreb – was imported into Aleph. Thanks to the consistent use of framework's capabilities and strict adherence to its conventions, application's architecture is identical to the one shown in Fig. 1.

E. *Comparison of the development processes*

It should be noted that any attempt to compare development processes of versions 1.0 and 2.0 is influenced by the fact that functionality was mostly known during development of version 2.0. While comparison would be much more relevant if two versions were developed in parallel, some – at least hypothetical – comparisons can be made.

For instance, change of the underlying database from SQLite to MySQL would be reduced to one minute needed to modify four settings in the configuration file, if the application has been developed using CodeIgniter from the beginning, thanks to CodeIgniter's Active Record Class (not counting time spent on programming the database, since that time is identical for both methods). On the other hand, the same change in the version written without using framework (performed later, solely for the purposes of this paper) took two hours of developer's time.

Unit testing is also an area of development where web application frameworks offer significant advantage. Software unit testing, defined by the IEEE as a process that includes the performance of test planning, the acquisition of a test set, and the measurement of a test unit against its requirements [13], is facilitated in CodeIgniter by the Unit Testing Class. In the version programmed without the use of CodeIgniter, unit testing was performed manually (although it should be noted that it could have been done by using specialized unit testing framework, such as PHPUnit).

F. *Code maintenance and future development*

Probably the greatest advantages of the version developed using CodeIgniter framework are in the area of code maintenance and future development of the application, as it has already been shown in the database switch example.

Code maintenance is considered by most developers as complex, technically difficult, "dirty" and tedious work [14], especially if the code being maintained has been written by another developer.

Use of CodeIgniter in development of ISVU2Aleph reduces time spent on code maintenance and future development in many ways:

- In the likely scenario in which the development will once be continued by another developer, new developer will spend significantly less time studying existing code of the version written in CodeIgniter than he would spend studying the code of the version 1.0, thanks to the Model-View-Controller architecture and other coding conventions.
- Future improvements that should otherwise be programmed manually (such as protection measures against future hacking attack techniques and yet unknown vulnerabilities) may be implemented in existing CodeIgniter methods and functions in future versions of the framework, and may only require upgrade of the framework to a newer version.
- Thanks to the numerous libraries and helpers included in CodeIgniter, same functionality is usually achieved with less code, so it is more easily maintainable. The same is true for features that will be added in the future.

V. CONCLUSION

Web application frameworks offer numerous technical and organizational advantages (e.g. faster development and cleaner application structure) over classical development methods. Also, programming using web frameworks is more comfortable for developers, since they do not have to deal with many common programming tasks.

Regardless of the above advantages, the development of small applications is usually done without frameworks, due to their perceived disadvantages.

However, we believe that the aforementioned advantages are equally applicable in the development of small applications, especially if the fact that the future development of the application is usually not possible to predict is taken into account.

REFERENCES

[1] S. McConnell, "Rapid development: Taming wild software schedules". Remond, Microsoft Press, 1996.

[2] Kent Beck et al., "Manifesto for agile software development", 2001. URL: http://www.agilemanifesto.org, accessed February 7th 2011.

[3] J. Shore and S. Warden, "The art of agile development". Sebastopol, O'Reilly Media, 2007.

[4] B. Askins and A. Green, "A Rails/Django comparison," in 2006 Open Source Developers' Conference (December 2006). Melbourne, 2006, URL: http://osdcpapers.cgpublisher.com/product/pub.84/prod.29/m.1/fid=174194/Askins%2CGreen-6934-RailsVsDjango.pdf (accessed February 13th 2011).

[5] S. Ruby, D. Thomas and D. Heinemeier Hansson, "Agile web development with Rails". Raleigh, The Pragmatic Programmers, 2008.

[6] P. Avgeriou and U. Zdun, "Architectural Patterns Revisited - A Pattern Language," in Proceedings of 10th European Conference on Pattern Languages of Programs (July 2005). Irsee, pp 1-39.

[7] J. Argudo Blanco and D. Upton, "CodeIgniter 1.7". Birmingham, Packt Publishing, 2009.

[8] T. J. Shelford and G. A. Remillard, "Real web project management: Case studies and best practices from the trenches," Boston, Addison-Wesley Professional, 2002.

[9] M. Fowler, "Patterns of Enterprise Application Architecture". Reading, Addison-Wesley Professional, 2003.

[10] I. Petrijevcanin Vuksanovic and B. Sudarevic, "Migration process and data modeling in National and University Library in creating ILS," in Proceedings of ELMAR-2010 (September 2010). Zadar, 2010, pp. 155–158.

[11] M. Baranović, M. Borčić, D. Hunjet, V. Kalafatić, D. Kranjčec, J. Mesarić and B. Peh, "Informacijski sustav visokih učilišta". Zagreb, Ministarstvo znanosti i tehnologije Republike Hrvatske, 2003.

[12] R. Lerdorf, K. Tatroe and P. MacIntyre, "Programming PHP". Sebastopol, O'Reilly Media, 2006.

[13] "IEEE Standard for Software Unit Testing", IEEE, 1986, URL: http://www.cs.tut.fi/kurssit/OHJ-3500/dokumentit/IEEE-STD-PDF/std1008-1987.pdf (accessed March 26th 2011).

[14] R. L. Glass, "Software Conflict 2.0: The Art and Science of Software Engineering", Atlanta, developer.* Books, 2006.