A Course on Web Programming

Tuomas Turto and Tommi Mikkonen

Department of Software Systems
Tampere University of Technology
PL 553, 33101 Tampere, Finland
{tuomas.turto, tommi.mikkonen}@tut.fi

Abstract. We faced the problem of deciding what is important when revising the contents of a Web Programming course. Torn between balancing industrial expectations and our vision of Web development based on research conducted at our University, we formed a set of topics that were first used during the Spring 2010. In this paper we report on the driving forces for the changes, how we structured the content and an outline of the course.

1 Introduction

Considering the current prevalence of different Web applications, for many students the *de facto* platform for new applications is the browser. Although encouraging from a motivational point of view, this is also somewhat unfortunate since web programming is a difficult topic that requires certain maturity with respect to software engineering. Paradoxically the situation is worsened by the availability of good tools and frameworks. The tools and frameworks make it relatively easy to get started and apply them to solve problems that fit nicely into the tools' problem domain. On the other hand, in order to understand and utilize the Web as a programming environment, a multitude of skills are required.

The required skills range from understanding dynamic programming languages to replication in distributed systems. These skills and the hands-on attitude required by the often substandard libraries are usually only found in the top part of a class. However, precisely these skills are favored by the industry. On the other hand, investing a lot of effort in technologies does not serve the academic aims of the course. From an academic perspective we should study the solid foundation of web engineering and critically investigate the weak points.

A course on web programming, in one form or another, has been lectured at the Tampere University of Technology since the late 1990s. From time to time larger changes were made in its contents but for a few years the course has been given in the same form. For the academic year 2010, however, changes were made that, hopefully, better enable to students to understand how the Web works and how it should be programmed. At the time of the writing, the first instance of the new course has just been completed. In this paper we report our motivation for the changes, our reasoning behind the content on the course and lessons learned from the first implementation.

2 Motivation and Expectations

There was no single issue that required the contents of the web programming course to be reviewed in full, instead it could be seen that a number of smaller issues started to appear. After they had been accumulating for a while, it seemed sensible to try to tackle most of them at the same time. The origins of the issues can be divided into developments in the field of web engineering and the practical applications, the skills required by the industry from our graduates and our own purely academic view of the future of web development. We review each of these in the following.

2.1 Recent developments

Due to the industrial significance of the Web, much of the innovative work done on web engineering and especially its practical applications is done outside the academia. We can divide the output from industry broadly into three categories

- standard-setting applications
- novel technologies and standards
- innovative solutions using existing techniques.

Each of these has a two-fold effect. Consider the standard-setting applications such as Google's GMail or Facebook. Firstly this affects the view students taking the course on web engineering have. Students take the course with the expectation that after the course they should have some understanding on how the standard-settings applications work – in an ideal case they should themselves be equipped with the tools to implement them. On the other hand the teaching staff should themselves have a clear grasp on how to implement such systems in order to identify the most important techniques used.

The same is true for the novel technologies and standards that easily fly under the radar but are, from an academic point of view, suddenly adopted by a range of significant users. Take, for example, the OpenID[4] and OAuth[3] protocols. The first one is used for single sign-on authentication and the second can be used to authorize resource sharing between web service providers. Both of these standards emerged out of vendor specific systems and were soon adopted by a wide audience. Although practical applications by their nature, the significance of these systems for general web ecosystem is so large that they have to be taken into account.

Innovative solutions using existing techniques might be considered as an euphemism for a kludge or a hack by some developers. Take, for example, the Bayeux protocol for implementing the transportation of asynchronous messages with low latency between the server and the client. The practical applications for such long-polling systems to enable asynchronous communication for use in, say, chat systems are evident. They go, however, against the basic principles of the HTTP protocol itself. Nevertheless, even in these cases we felt that we need to account for such systems. For good or bad, they are an inherit part of many practical systems built today.

2.2 Industry Expectations

With the recent developments, also the demands the industry places on new hires have changed. As outlined previously, the industry churns out novel solutions to new problems in the domain of web engineering at an increasing pace. The current expectation is that either the graduate knows about the techniques or is equipped with suitable skills and with a solid understanding about the fundamentals of web engineering so that there are no obstacles to learning the new techniques.

This situation is in a striking contrast with respect to how things appeared to be a few years ago. Then it seemed to be enough if an introductory course on web programming taught how to utilize J2EE in the context of Web engineering. Now there is no such homogeneity. Naturally familiarity with different Java-based solutions is widely sought after in job advertisements but with the increasing diversity of specific techniques employees look for, choosing a single technology base for a course is not as straightforward as it used to be.

Moreover, the principles and techniques of Web have spread outside the Web proper. Take for example the job advertisements that look for "– background in Web Runtime (WRT) or Webkit development" combined with "– Symbian, Linux, Maemo or Windows platform –". Many also prefer some amount of JavaScript fluency that was nowhere to be seen few years ago. In our opinion this reflects the increasing significance of client-side programming on the Web and on devices, such as mobile phones, that utilize the Web development model.

2.3 Academic Motivation

In addition to industry's expectations and as a response to the recent developments we feel that each group discussing a course on Web programming at a university should have a vision on the future of web development. It not only helps to integrate some recent research into the course but hopefully also provides a means to navigate around too much industrial pressure and focus on things that the faculty feels are important – if just including them in passing during lectures.

As an example, in our case the academic movation for developing the curriculum for a course on Web engineering can be divided into two parts, both being under active research at our own institute. These are

- Web as a distributed application platform
- Web techniques outside the Web, e.g. on mobile devices.

The first one reflects our vision of the browser transforming itself into a target for new applications. In a sense this has already happened with web-based applications, but we believe that limits continue to be stretched, as was the case with the Lively Kernel[5], for example. For a Web curriculum this view means that we must take JavaScript seriously and provide even coverage between programming the client and the server.

Using Web techniques outside the Web is gaining momentum, as could be also seen from the industry's requirements. In practice this means using lightweight dynamic languages at the client and integrating the client and the resources at the server using HTTP. Client might expose additional resources for the scripting language, which often is JavaScript, but the basic development environment is familiar from the Web. For the course contents this means that general principles should be favored over specific implementation techniques.

2.4 Department's View

Also the department plays a role in forming the requirements for the course. They do it by specifying the minimum requirements for passing the course. For our course the department has decided on the following[6]

> After completing the course OHJ-5101 Web Programming the student can explain the behavior of a modern Web program from the view of
> – HTTP protocol
> – the server, and
> – the client
> During the course the student familiarizes himself with programming both the server and the client in the context of Web. After the course the student has the required know-how for developing new simple applications for the Web environment. Moreover, he can apply the skills in practice. Student can give examples of the most common problems faced while developing Web applications. In addition, the student can name typical security issues in Web applications.

There are three things worth emphasizing. The first thing is the fact that the previous quote from the Course Catalogue describes the minimum requirements. The second item is the requirement for the ability to actually apply the skills in a real setting reflecting the industry's expectations. The third one is the note on security issues. It is interesting that while others seem to take the security issues for granted, the department's requirements emphasize them.

It might seem somewhat disturbing that the department's requirements do not include anything that we might take to be academic in the sense that they would not be directly applicable but useful in some wider sense – as a foundation for future work. However, it must be noticed that the text above is meant for students who are selecting which courses to take. Often they are interested in the skills they acquire during a course – skills that might make them employable later. Moreover, the quote reflects the minimum requirements required for passing the course. There is a lot of room to cover interesting topics outside minimum requirements.

3 Surveying the Landscape

Based on the motivation and expectations, we next survey the concepts, techniques and technologies that in our opinion should be a part of a modern web

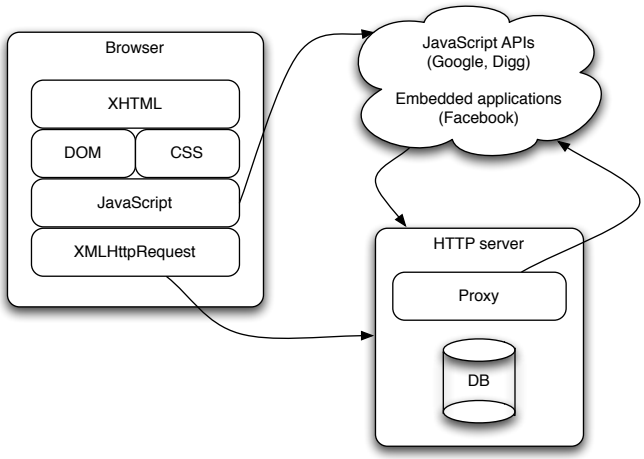

Fig. 1. Division of Techniques

programming course table of contents. Depending on outside constraints, such as the length and difficulty of the course, not necessarily all aspects can be covered during a single course, though.

The crude division between different topics is presented in the Figure 1. We divide the material between programming the browser, programming the server, communication in web applications and being part of the Web ecosystem. Some issues, such as security, affect multiple parts and have to be discussed either after other material has been covered or in parts.

3.1 Browser

When revising the contents for the web programming course, the single most significant weakness in the material concerned programming the browser. Although the material was only a couple years old, the browser was neglected almost completely. Today the browser itself is the target of many interesting non-trivial applications and in principle one can write applications solely for the browser, even without any extra communication between the server[5].

The relevance of the browser programming is only increasing as modern specifications include aspects that make it resemble even more a traditional application platform. These enhancements include, for example, 3D bindings for JavaScript and HTML5 features such as web sockets. Nevertheless, at the moment the key elements that are required are (X)HTML, CSS, DOM bindings and JavaScript. Comparing to traditional courses on Software Engineering, in the Web environment these form the assembler – or system – level programming approach. Understanding these key techniques is mandatory in order to appreciate the challenges that programming the browser entails.

The first three can be discussed as they are and should not present any significant difficulties. Although for some students the concept of declarativeness that creeps up from creating effect by changing the abstract internal form of the page might be a bit unfamiliar, the APIs are themselves are quite straightforward. Pedagogically the best route might be to show how a selected client-side JavaScript library, such as jQuery, abstracts these assembler level concepts into a coherent whole. The workings of a client-side UI library, such as jQuery UI, can be explained in a similar fashion.

JavaScript, on the other hand, is a significantly more difficult subject. At an institution, such as ours, where students start with statically typed traditional languages such as C++ or Java, the whole dynamic nature of JavaScript might be unfamiliar. At universities where dynamic languages, such as Python, are used during undergraduate courses it might not be such an issue. Nevertheless, JavaScript is a language of its own and most important difficulties arise in

- understanding the prototypal nature of the language
- applying good Software Engineering practices also with JavaScript
- tool support

Combined with the inherit dynamism of the language, the prototypal nature of JavaScript makes it probably different from any other language the students have accustomed to. JavaScript resembles more Self with substantial functional influences than it does, say, Python. Although interesting in itself, this also makes introducing JavaScript difficult as many of the libraries tend to emulate class-based object system which in principle is foreign to JavaScript. A choice must be made whether to present JavaScript as a programming language that stands on its own or as a scripting language that can be made to resemble one with a class system.

As the language itself can be a bit foreign, the emphasis must be put on the use of proper software engineering principles. The problem is, however, that it is not quite clear what these should be in case of the browser and JavaScript. The language itself lacks proper support for modules and namespace handling. These are often emulated by some convention in libraries and must be explained. Compounded with varying tool support, developing includes challenges not present in traditional systems programming. These must be taken into account when discussing developing larger JavaScript applications[1].

3.2 Server

The use of programs at the server-side to generate HTML documents on the fly represents the traditional form of web applications. As such this traditional form of serving dynamic content must be discussed. However, the discussion should focus on the first principles of web. In essence this consists of the HTTP request and response, how requests are processed by the web server, the protocol by which the server transforms control over to the web application, and, to some extent, the role of the database.

In our opinion it is easy to neglect the role of HTTP as the fundamental building block. Note that we discuss HTTP under the topic of server programming instead of communication. A solid understanding of the HTTP protocol right from the start equips the students to later on make objective comparisons, say, between a RESTful and SOA architectures. Furthermore, in order to understand more complex scenarios with proxies and gateways, it is necessary to appreciate why the client-server model was chosen and how the communication between these two is configured through request and response headers.

In practice some framework is always used to implement server-side programs. However, understanding the division of work between the web server and the framework is mandatory. Moreover, students should know the details of how communication between the two is achieved. To this end, a comparison of different techniques, say FastCGI and mod_python, gives the students an overview why we have distinct application servers and serve static content from elsewhere.

Of the basic building blocks for a server-side program, in our discussions we have found that opinions over what should be covered about databases vary the most. Many web programming courses require a database course as a mandatory prerequisite. As such, some argue that students with some background in SQL and with a working knowledge of databases do not need other instruction than that of how different frameworks implement the ORM layer. Others, however, would expect even an introductory course on web programming to include somewhat more discussion on persistence layer. They justify this by arguing that often database is the first bottleneck in real-life applications.

Arguably the only reason not to discuss databases in more depth would be the either to have a distinct course for database scalability or, as is more likely, because of time limitations. Discussion on databases and the new proposed alternatives, such as the key-value stores, would bring cutting edge work into the classroom and make it possible to discuss solutions aimed at scalability, such as memcached, that might be difficult to integrate into the course without an added discussion on key-value stores.

Finally, everything above is about inclusion. However, it is also important to make explicit what we exclude. In our reasoning, the large WS-* stack of web services does not belong to an introductory course on web programming. Some argue that it should not even belong to an advanced course. This is a matter of taste but in our discussions we felt that traditional big web services are too bulky for us to go through them in class. Moreover, the APIs published by major providers tend to be more or less RESTful.

3.3 Communication and the Web Ecosystem

Discussion of communication focuses on the client-server model – its benefits and limitations. In practice this means discussing how the massive scalability of the Web has been achieved, but at the same time noting how the model restricts the communication to be always client initiated, at least in principle.

The most important practical issue with respect to communication is the behavior of XMLHttpRequest in client-side JavaScript. XMLHttpRequest is also

the natural place to discuss the same-origin requirements for network access, and also with regard to DOM and cookies. This is a natural starting point for discussion on web security.

Security can be discussed in conjunction with communication. For the obvious reasons, all of the attacks are based on communication patterns of different kinds. The only difference is that in the attacks, the communication is formed with the intent of generating undesired behavior. Looking at the current environment, the most significant security issues relate to different injection attacks, including the cross-site scripting vulnerabilities, and to cross-site request forgery. At least these should be covered in any introductory course. Understanding why they work also strengthens the general understanding of the underpinnings of the Web.

Depending on the time available, we envisage that the discussion of communication methods should also include mashups and remote resource loading for JavaScript. In addition to being timely, mashups also demonstrate the power of Web as an ecosystem for applications.

4 Implementation

Based on the motivation presented, we have implemented an introductory course on web programming that includes some parts of the ideal course structure presented in the previous section. In this section we describe the course contents, present how it was implemented during Spring 2010, and conclude with some final remarks.

4.1 The Course

The size of the course OHJ-5101 Web Programming is 4 ECTS units. Each ECTS unit corresponds to approximately 27 hours of work performed. In calendar time the course lasts 14 weeks and divided into two halves. The course includes 24 hours of lectures given two hours at a time. In addition to this the students have to implement a mandatory project work in pairs. If no weekly exercises are arranged, it can be seen from the sizing that in this case the mandatory project can be of substantial size.

Our division of material is depicted in Table 1. The table shows how during the first half the server side programming was discussed and how in the second half the emphasis was on the client-side programming and security issues. As can be seen from the content, we had to exclude much of the interesting material that we deemed fit for a introductory course. Moreover, the table shows the realized lecture schedule. Thus we have the odd ends lecture when previous material took more time than expected.

From a practical point of view, we took the Django framework, written in Python, as the example framework. Django served a two-fold purpose. Firstly, it was used as an example implementation. Consequently, we were able to discuss handling of state, for example, in the context of an actual implementation and

Lecture 1: Introduction – Web as a distributed system[2] – The HTTP protocol	Lecture 7: JavaScript – JavaScript: The Good Parts[1] – What makes JavaScript different
Lecture 2: From HTTP to Programming – Dynamic resources – Web server and programs – Examples	Lecture 8: JavaScript in the Browser – Browser as host environment – HTML, CSS and the DOM interfaces
Lecture 3: Server-side Frameworks – Model-View-Controller – Django – ORMs and the Model Layer	Lecture 9: Visiting Lecture – Lecture given by a practicing web engineer
Lecture 4: Web and State – Concluded frameworks – Statelessness of HTTP – Cookies – Effect of State on Server	Lecture 10: Communication – XMLHttpRequest – Abstracting communication – Same-Origin Policy
Lecture 5: Authentication – Protocol support (Basic, Digest) – Modern authentication (OpenID) – Framework support	Lecture 11: Security – CSRF and Login CSRF – Injection attacks (SQL, XSS) – Session Fixation
Lecture 6: Odd ends – Authorization (OAuth) – Basics of REST	Lecture 12: Wider Web – Mashups – Being Part of Web

Table 1. Course Outline

browse through the code. This, we believe, made it more concrete for students. Secondly, we needed a framework for the course project for the students to use. As we discussed during lectures how Django implements the different parts required by a web application, students could apply this knowledge during the course project.

The course project was a simple CD database. Ignoring the switch from J2EE to Django, the only significant difference between the course project in the new version of the course compared to the old one was the emphasis on programmable API. In all the previous courses the application was meant to be used by humans utilizing a normal web browser. In the new course, however, in addition to the traditional browser interface a RESTful API was present. The API was used as an example of an interface callable from client-side JavaScript using XMLHttpRequest in addition to being an example of a public interface that might be offered by some real service.

4.2 Lessons Learned

The most important lesson learned was that giving a good introduction to web engineering takes time. The twelve two-hour lectures are not enough to cover in suitable depth all the required aspects of an introductory course. During the next academic year we will have four additional weeks of calendar time which corresponds to eight lecture hours. This will helps us get further, or at least to discuss the issues at necessary depth, but the only reasonable solution seems to be to divide the course into two parts. This, however, would again entail a discussion how to divide the material.

In retrospect, we also had to spend too much time on Django issues, such as rendering templates, just for the sake of the course project. In the future, as time is the limiting factor, we plan to move all these mundane details to weekly exercises and discuss Django only during lectures when the framework's implementation of some issues helps to clarify how an abstract web programming concept is realized in an actual implementation. Otherwise the switch from J2EE to Django seems to have been a success. Documentation is easily available and it is easy to get started.

In our opinion, having a large enough course project is a necessity. As the problem domain does not contain any significant theoretical hurdles but instead most of the problems are very much oriented towards engineers, showing real working code is the best proof of understanding. This also helps us to convince the industry of our approach – they get graduates who can produce real working software.

5 Conclusions

We have presented a motivation, a landscape and an outline for what we believe should form an introductory course in web engineering. During our first implementation round, we have found the biggest problem to be time available for the course. Time allowing, even more emphasis should be put on the client-side programming.

References

1. Crockford, D.: JavaScript: The Good Parts. O'Reilly (2008)
2. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
3. Hammer-Lahav, E.: The OAuth 1.0 Protocol. IETF Internet-Draft (August 2010)
4. OpenID Authentication 2.0 - Final, http://openid.net/specs/openid-authentication-2_0.html
5. Taivalsaari, A., Mikkonen, T., Ingalls, D., Palacz, K.: Web Browser as an Application Platform: The Lively Kernel Experience. Tech. Rep. SMIL TR-2008-175, Sun Labs (January 2008)
6. Tampere University of Technology Course Catalog (2010-2011)