

The Mashware Challenge: Bridging the Gap Between Web Development and Software Engineering

Tommi Mikkonen
Tampere University of Technology
Korkeakoulunkatu 1, FI-33720 Tampere, Finland
tommi.mikkonen@tut.fi

Antero Taivalsaari
Nokia Research Center
Visiokatu 1, FI-33720 Tampere, Finland
antero.taivalsaari@nokia.com

ABSTRACT

The software industry is currently experiencing a paradigm shift towards web-based software. Although the Web was not originally designed to be a software platform, it is rapidly becoming *the* platform for all the end-user software. In this position paper we argue that (1) development practices for web applications are still far from the maturity levels of traditional software engineering, (2) web development will evolve towards “mashware” – mashup software that leverages source code and software components downloaded dynamically from all over the world, (3) there is still an impedance mismatch between web development and software engineering, (4) the trend towards mashware will exacerbate the gap between web development and software engineering, and (5) research is needed in several areas, including modularity and security, to ensure that the academic world does not get left behind from the fundamental changes that are impacting in the software industry.

Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques – *modules and interfaces, software libraries*. D.2.10 [Software Engineering]: Design – *methodologies*. D.2.11 [Software Engineering]: Software Architectures – *data abstraction, information hiding, languages*.

General Terms

Design, Experimentation, Languages.

Keywords

Software engineering, web engineering, mashups, mashware.

1. INTRODUCTION

The software industry is currently experiencing a paradigm shift towards web-based software. Applications that were previously written for specific operating systems or CPUs are now written for the Web, to be used from a web browser from anywhere on the planet. Although the Web was not originally designed to be a

software platform, it is rapidly becoming *the* platform of choice for all the end-user software. For instance, the majority of new business applications, e.g., travel expense reporting systems, banking and stock trading systems, are now web-based.

In recent years, significant progress has been made in turning web engineering into a real engineering field; for a comprehensive overview, see [6]. However, we argue that the development practices for web applications are still far from the maturity levels of traditional software development. In fact, as summarized in earlier papers [13, 14, 15], there is still an *impedance mismatch* between web-based software development and software engineering.

We believe that the evolution of web technologies will eventually lead us to *mashware* – mashup software that leverages source code and software components that are downloaded dynamically from all over the world. Such software can dramatically improve the productivity of software development, allowing massive reuse of software components across the planet. However, without new software architectures, methodologies and systematic approaches towards the development of such software, the gap between web development and software engineering will only grow wider. To avoid this, research is needed in several areas related to web development, including security, modularity and legal aspects, as well as software engineering methodologies to foster the development of mashware in systematic fashion.

The rest of this paper is structured as follows. Section 2 provides a brief overview of the evolution of the Web from the viewpoint of software development. Section 3 discusses the trend towards mashware – software as a worldwide mashup. Section 4 focuses on the impedance mismatch between web development and software engineering. Section 5 outlines a number of interesting problem areas in which future research is required. We finish the paper with a call for action for academic researchers in Section 6.

2. EVOLUTION OF THE WEB AS A SOFTWARE PLATFORM

The World Wide Web has undergone a number of evolutionary phases. Initially, web pages were simple textual documents with limited user interaction capabilities based on hyperlinks. Soon, graphics support and form-based data entry were added. Gradually, with the introduction of DHTML – the combination of HTML, Cascading Style Sheets (CSS), the JavaScript language [4], and the Document Object Model (DOM) – it became possible to create interactive web pages with built-in support for advanced graphics and animation. Numerous plug-in components – such as Flash, RealPlayer and Shockwave – were then introduced to make

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

FoSER 2010, November 7-8, 2010, Santa Fe, New Mexico, USA.

Copyright 2010 ACM 978-1-4503-0427-6/10/11...\$10.00.

it possible to build web pages with visually rich, interactive multimedia content.

In general, the evolution of the Web has advanced from simple, “classic” web pages with text and static images to animated multimedia pages with plug-ins to Rich Internet Applications. Below we provide a brief summary of the evolution of the Web as a software platform (Figure 1).

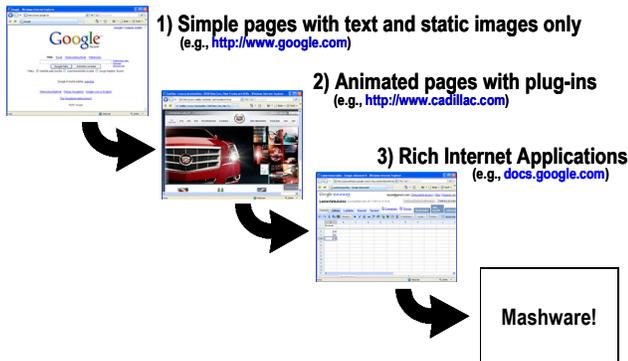


Figure 1. Evolution of the Web as a Software Platform

In the first phase, web pages were truly *pages*, i.e., page-structured documents that contained primarily text with interspersed images, without animation or any interactive content. Navigation between pages was based on simple hyperlinks, and a new web page was loaded from the web server each time the user clicked on a link. There was no need for asynchronous network communication between the browser and the web server. Some pages were presented as *forms*, with simple textual fields and the possibility to use basic widgets such as buttons and radio buttons.

In the second phase, web pages became increasingly interactive, with animated graphics and plug-in components that allowed richer, more interactive content to be displayed. This phase coincided with the commercial takeoff of the Web, when companies realized that they could create commercially valuable web sites by displaying advertisements or by selling merchandise or services over the Web. Navigation was no longer based solely on links, and communication between the browser and the server became increasingly advanced. The JavaScript scripting language, introduced in Netscape Navigator version 2.0B in December 1995, made it possible to build animated, interactive content more easily. The use of plug-in components such as Flash, Quicktime, RealPlayer and Shockwave spread rapidly, allowing advanced animations, movie clips and audio tracks to be inserted in web pages. In this phase, the Web started moving in directions that were unforeseen by its designers, with web sites behaving more like multimedia presentations rather than static pages. Content mashups and web site cross-linking became increasingly popular.

Recently, the Web has evolved towards a platform for desktop-style web applications, also known as Rich Internet Applications. Such technologies make it possible to build web sites that behave much like desktop applications, for example, by allowing web pages to be updated asynchronously one user interface element at a time, rather than requiring the entire page to be updated each time something changes. These systems often eschew link-based navigation and utilize direct manipulation techniques familiar from desktop-style applications instead.

The three phases discussed above are by no means mutually exclusive. Rather, web pages representing all three phases coexist on the Web today. The majority of commercial web pages today represent the second phase. However, the trend towards web applications is becoming increasingly common, with new web application development technologies and systems being introduced frequently.

3. TOWARDS MASHWARE: WEB APPLICATIONS AS MASHUPS

In web terminology, a *mashup* is a web site that combines content from multiple web sites into an integrated experience. Mashups are content aggregates that leverage the power of the Web to support worldwide sharing of content that conventionally would not have been easily accessible or reusable in different contexts or from different locations. In principle, the content used in mashups can be anything as long as it can be meaningfully combined with other information available on the Web. The key requirement is that the content must be available in a format that can be reused easily in other contexts. Various tools for mashup development have been introduced in recent years, including Google Mashup Editor (<http://code.google.com/gme>), IBM Mashup Center (<http://www.ibm.com/software/info/mashup-center/>), Intel Mash Maker (<http://mashmaker.intel.com/>), Microsoft Popfly (<http://popfly.com/>), and Yahoo! Pipes (<http://pipes.yahoo.com/>). A summary of these tools is available in [13].

In the absence of security restrictions that prevent the downloading of executable code to the browser from different domains, the content in mashups could be executable code as well. In fact, we believe that the next logical step in the evolution of the Web as a software platform is *mashware*. By this we refer to a generalized form of mashup-based software development, in which applications can be composed by dynamically combining code and other content originating from web sites from all over the world. For instance, the user interface widgets of an application might be downloaded from one site, storage features from another site, the localization capabilities from a third site, and so on, based on the availability of best components for each purpose. We refer to such a model for application development as *mashware* – software as a worldwide mashup. The general idea is depicted in Figure 2.

In Figure 2, we assume that the developer is building a new web-based application to visualize stock market information. The application consists of a main application – downloaded from the developer’s own web server– that will dynamically download the other necessary components from other web sites. These components include: (1) widget library used for presenting the user interface of the application, (2) stock graph visualization library for creating stock graphs, (3) stock quote / market data service interface available from a third site, and (4) localization (L10N) components for customizing the market data and the language for a specific country. All these components are downloaded from different web servers and used dynamically without compilation, static linking, or explicit installation.

For various reasons the construction of this kind of software is not yet possible. However, this kind of an approach would make it possible for application developers and software engineers to collaborate on an immensely large scale, allowing unparalleled

sharing and reuse of software, data, layout and visualization information, or any other content across the planet. Applications would be truly *web* applications, consisting of components that are loaded dynamically from those web sites that provide the most applicable components for each purpose. If such massive-scale reuse were possible, the productivity of software development could potentially be improved dramatically. The Web could be the enabler that will finally make large-scale worldwide software reuse a reality rather than just a perpetual dream.

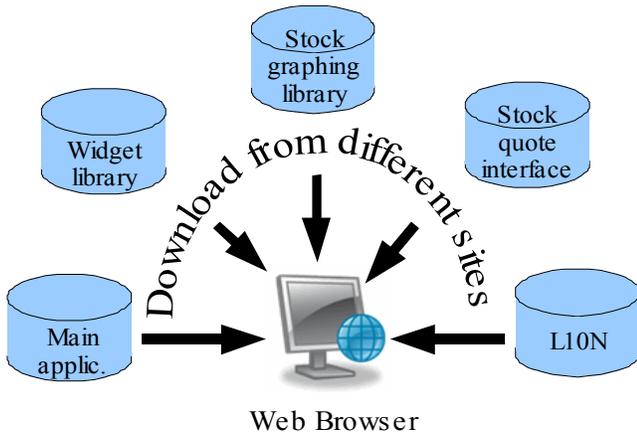


Figure 2. Software as a Worldwide Mashup

4. IMPEDANCE MISMATCH

As long as the primary purpose of web development was the creation of web sites consisting of documents, pages and forms, there was little reason to apply software engineering principles to web development. The principles and practices for web development evolved rather independently of the principles and practices for software engineering. There is a considerably body of research in the field of *web engineering* to turn web development into a systematic engineering field. For an introduction to web engineering, refer to [3, 6, 12]. ICWE (International Conference on Web Engineering) conference proceedings are also a good starting point.

Web Development	Conventional SW Development
- Documents	- Applications
- Page / form oriented interaction	- Direct manipulation
- Managed graphics, static layout	- Directly drawn, dynamic graphics
- Instant worldwide deployment	- Conventional deployment
- Source code and text favored	- Binary representations favored
- Development based mostly on conventions and "folklore"	- Development based on established engineering principles
- Informal development practices	- More formal development
- Target environment not designed for applications	- Target environment specifically intended for applications
- Tool-driven development approach	- A wide variety of development approaches available

Table 1. Impedance Mismatch

There is still an *impedance mismatch* between web development and conventional software development (Table 1). This mismatch arises from the fact that the Web was not originally designed to be a software platform. The Web was originally built around the

notion of documents that have a relatively static layout and use page- or form-oriented interaction. Mechanisms and APIs for supporting features that are common in conventional desktop applications, such as interactive graphics and direct object manipulation, were largely missing from the web browser.

Even today, the primary way to manipulate the graphical objects on the screen in a web-based application is to programmatically tweak the attributes in the web page's Document Object Model (DOM) tree from JavaScript code. The DOM is effectively a large global data structure that – compared to most desktop operating system graphics interfaces – is manipulated at a rather low level of abstraction. The other APIs offered by the standard web browser environment are also limited. This has led to the introduction of various JavaScript libraries that aim at raising the level of abstraction and facilitating web application development in general. Such libraries include Dojo (<http://dojotoolkit.org/>), jQuery (<http://jquery.com/>), Mootools (<http://mootools.net/>), Prototype (<http://www.prototypejs.org/>), Scriptaculous (<http://script.aculo.us/>), Sencha (<http://www.sencha.com/>) and numerous others.

Unlike conventional binary applications, web applications are generally deployed in textual form, using representations such as HTML, XML, CSS and JavaScript source code. The ongoing shift to dynamic representations and languages has a significant impact on development, integration testing and deployment practices [8]. Furthermore, since most of the components in web applications are downloaded dynamically, static checking is generally very difficult, placing special emphasis on tools that ensure that all the necessary components and functions are in place. The use of tools is important also because the web browser is an exceptionally forgiving execution environment, not displaying error messages or halting program execution until absolutely necessary.

In spite of these and many other differences and challenges, we believe that the trend towards web-based software will continue and even strengthen in the future. Conventional binary software simply cannot compete in a web-driven economy in which worldwide software distribution is effectively free and in which new software versions can be completed and released – without compilation, linking, installation and/or rebooting – in a matter of seconds or minutes rather than months or years. All this will dramatically change the development, deployment and use of software – implying significant changes for the software industry and for software engineering research, too.

5. INTERESTING RESEARCH AREAS

There are numerous interesting research areas related to bridging the gap between web development and software engineering, especially in view of the transition towards mashware. In our earlier papers, we have categorized and divided the areas as follows [7, 15]:

- (1) software engineering principle violations,
- (2) usability and user interaction issues,
- (3) networking and security issues,
- (4) browser inoperability and incompatibility issues,
- (5) development style and testing issues,
- (6) deployment model changes, and
- (7) performance issues.

Moreover, we have investigated the software engineering principle violations from a number of different angles including modularity, consistency, simplicity, elegance, reusability, and portability [7, 15]. For the purposes of this short position paper, we want to briefly highlight two topics – modularity and security – that hinder the transition towards mashware.

5.1 Lack of Modularity

Modularity is a form of abstraction that allows systems to be composed of distinct parts, independently of implementation details. A module is generally defined to be a self-contained component of a system that has a well-defined interface to the other components. A *well-defined interface* separates the specification of a component from its implementation details, supporting *separation of concerns* and allowing the implementation to be changed without impacting the external use of the component. *Information hiding* is a principle that goes hand in hand with modularity and well-defined interfaces. These modularity principles were introduced in the 1970s by numerous researchers (see, e.g., [9, 10, 11]).

In principle, the absence of static bindings or binaries on the Web would not necessarily have to imply the lack of modularity, well-defined interfaces or information hiding. Unfortunately, today's web sites are rather unmodular “white boxes”, or worse yet, “glass boxes”, with their implementation details openly exposed to the outside world. The glass box approach is beneficial in the sense that it opens up the content of a web site to be leveraged by thousands of other sites and developers. However, it also makes such leveraged web sites extremely brittle, for there is no widely established way for a web site to publish its intended external interface separately from its implementation. Specification languages such as WSDL [17], WADL [16] and WebML [2] have been proposed and studied extensively, but none of them has reached widespread use in the industry yet.

As summarized in a previous article [14], there is a *major contradiction* on the Web today with respect to modularity: Mashup developers – or more generally web application developers – are usually unknown to each other. When developers reuse content from other sites, they often do not know the developers of those other sites in person. In such a setting, modularity is critical, for it is nearly impossible to build well-behaved, maintainable systems unless all the elements of reuse have well-defined interfaces with proper information hiding. This simply isn't the case in web development today.

5.2 Security Issues

The history of the web browser as a document viewing environment – as opposed to an application platform – is apparent when analyzing the limitations that web browsers have in the area of networking and security. Many of these limitations date back to conventions that were established early on in the design and historical evolution of the web browser. Some of the limitations are “folklore” and have never been formally documented or standardized. The key problems in this area include [7, 15]:

Same Origin policy restrictions. The philosophy behind the same origin policy is simple: it is not safe to trust content loaded from arbitrary web sites. When a document containing a script is downloaded from a certain web site, the script is allowed to

access resources only from the same web site (origin) but not from other sites. The same origin policy makes it difficult to build (and deploy) mashups or other web applications that combine content (e.g., news, weather data, stock quotes, traffic statistics) and code from multiple web sites. In reality, different ways to circumvent the problem exist, many of which result in superfluous complexity and potential security problems of their own.

No namespace isolation to safeguard applications from cross-site scripting. A well-known security problem area in web application development today is *cross-site scripting* (XSS). Cross-site scripting is a type of security vulnerability that arises from the fact that today's browsers and the JavaScript language do not provide namespace isolation to safeguard scripts or other active content loaded from one site from the scripts or active content loaded from other sites. A good summary of the issues is provided in [5].

Limited access to host platform APIs. Web applications are usually run in a sandbox that places various restrictions on which resources and host platform capabilities the web browser can access. The sandbox security limitations of the web browser make it difficult to build web applications that utilize local resources or other host platform capabilities.

The key point arising from these and other limitations is that there is a need for a *more fine-grained security model* for web applications. On the Web today, applications are second-class citizens that are on the mercy of the classic, “one-size-fits-all” sandbox security model of the browser. Decisions about security are determined primarily by the origin from which the application is loaded, and not by the specific needs of the application.

6. CALL FOR ACTION

The World Wide Web is the most powerful medium for sharing information in the history of humankind. Therefore, it is not surprising that the use of the Web has spread into many new areas outside its original intended use. Because of its impressive power, it is easy to overlook the deficiencies of the Web in areas that are beyond its original design. In many ways, the use of the Web as a software platform resembles the fairy tale about emperor's new clothes [1]: the deficiencies are so blatantly obvious that they should be visible to everybody. Yet the Web is so powerful that people are willing to look the other way.

This position paper is a call for action for software researchers. So far, web engineering and software engineering have evolved as separate fields. Software engineering researchers have paid little attention to the evolution of HTML, CSS, JavaScript and web technologies in general. Conversely, researchers in the web engineering field have largely ignored the established principles and over four decades of research in the software engineering field. As the transition towards web-based software progresses, this will have to change.

It is time to forget the origins of the browser as a document viewing environment and to start treating the Web as a real, full-fledged application platform – one whose capabilities will eventually far exceed those of the earlier software platforms. Even though the conceptual and technological foundations of the Web are somewhat shaky, with a concentrated effort most of the problems can be fixed. Transforming the Web into a first-class

software platform should be one of the most important research topics for academic researchers in this decade and beyond.

7. REFERENCES

- [1] Andersen, H.C. 1837. *Emperor's New Clothes*. Houghton Mifflin, 2004 (the original story published in 1837).
- [2] Ceri, S., Fraternali, P., and Bongio, A. 2000. Web Modeling Language (WebML): a Modeling Language for Designing Web Sites. In *Proc. 9th International Conference on World Wide Web* (Amsterdam, Holland, May 15-19, 2000), ACM. URL: <http://www9.org/w9cdrom/start.html>.
- [3] Deshpande, Y., and Hansen, S. 2001. Web Engineering: Creating a Discipline Among Disciplines. *IEEE Multimedia* 8, 2 (April-June 2001), 82-87.
- [4] Flanagan, D. 2006. *JavaScript: The Definitive Guide*. O'Reilly Media.
- [5] Jim, T., Swamy, N., and Hicks, M. 2007. Defeating Script Injection Attacks with Browser-Enforced Embedded Policies. In *Proc. 16th International Conference on World Wide Web* (Banff, Canada, May 8-12, 2007), ACM, 601-610.
- [6] Kappel, G., Pröll, B., Reich, S., and Retschitzegger, W. (eds) 2006. *Web Engineering: The Discipline of Systematic Development of Web Applications*. John Wiley and Sons.
- [7] Mikkonen, T., and Taivalsaari, A. 2008. Web Applications: Spaghetti Code for the 21st Century. In *Proc. 6th International Conference on Software Engineering Research, Management and Applications* (Prague, Czech Republic, August 20-22, 2008), IEEE Computer Society, 319-328.
- [8] Paulson, L.D. 2007. Developers Shift to Dynamic Programming Languages. *IEEE Computer*, February 2007, 12-15.
- [9] Parnas, D.L. 1972. A Technique for Software Module Specification with Examples. *Communications of the ACM* 15, 5 (May 1972), 330-336.
- [10] Parnas, D.L. 1972. On the Criteria to be Used in Decomposing Systems into Modules. *Communications of the ACM* 15, 12 (December 1972), 1053-1058.
- [11] Parnas, D.L., Clements, P.C., and Weiss, D.M. 1983. Enhancing Reusability with Information Hiding. In *Proc. ITT Workshop on Reusability in Programming* (Newport, Rhode Island, September 7-9, 1983), 240-247.
- [12] Rossi, G., Pastor, O., Schwabe, D., and Olsina, L. (eds) 2008. *Web Engineering: Modeling and Implementing Web Applications*. Human-Computer Interaction Series, Springer, London.
- [13] Taivalsaari, A. 2009. *Mashware: the Future of Web Applications*. Sun Labs Technical Report TR-2009-181, February 2009.
- [14] Taivalsaari, A., and Mikkonen, T. 2008. Mashups and Modularity: Towards Secure and Reusable Web Applications. In *Proc. 1st International Workshop on Social Software Engineering* (L'Aquila, Italy, September 16, 2008).
- [15] Taivalsaari, A., Mikkonen, T., Ingalls, D., and Palacz, K. 2008. Web Browser as an Application Platform: The Lively Kernel Experience. In *Proc. 34th Euromicro Conference on Software Engineering and Advanced Applications* (Parma, Italy, September 3-5, 2008), IEEE Computer Society, 293-302.
- [16] World Wide Web Consortium (W3C) 2009. *Web Application Description Language (WADL)*. W3C Member Submission (Aug 31, 2009). URL: <http://www.w3.org/Submission/wadl/>.
- [17] World Wide Web Consortium (W3C) 2007. *Web Services Description Language (WSDL) Version 2.0, Part 1: Core Language*. W3C Recommendation (June 26, 2007). URL: <http://www.w3.org/TR/2007/REC-wsdl20-20070626/>.