Why is RE for web-based software development easier?

Didar Zowghi[1], Vincenzo Gervasi[2]

[1] Faculty of Information Technology, University of Technology, Sydney,
P O Box 123 Broadway, NSW 2007
Australia
didar@it.uts.edu.au
[2] Dipartimento di Informatica, University of Pisa
c.so Italia, 40 – I-56125 Pisa
Italy
gervasi@di.unipi.it

Abstract. In this paper we identify the fundamental differences between web-based and conventional software engineering. We express our views about why we think requirements engineering for web-based application development should be easier than requirements engineering for conventional software development. We will examine the consequences of holding such a position and propose a more effective process model for web-based application development. Given that we hold such view, we believe that we can do a better job of requirements engineering than is currently being practiced in web-based application development. We conclude by showing why we believe this is a promising area for investing research efforts.

1 Introduction

Although the development process for web-based applications has many similarities with conventional software development, it also has a number of unique characteristics that make it fundamentally different from traditional software. These characteristics are both technical (such as the reliance on a highly componentized overall architecture) and organisational (such as variations in client understanding of the system capabilities at different stages of development). Although these differences are not generally well understood in the research literature, a discussion of some of these issues can be found in [2,3,5,7].

This research highlights the need for a clearer understanding of the specification process. In particular, Web systems are more likely to be built in a more evolutionary manner than is typical of conventional software and IT systems, and hence their specifications should be shaped accordingly. Developers of web-based systems typically do not distinguish between the requirements elicitation and analysis, and the system design, nearly as clearly as for conventional development. Indeed, many developers see it as important to utilise the early design stages to elicit more detailed client feedback on requirements.

One of the major problems we face in web-based development is the *independent evolution of heterogenous components* involved such as server, middleware, client side libraries, network tier and the like. We have to identify a set of *controlled* components such as applications, server software, possibly server ISP, and a set of *independent* components such as client ISP, operating system and browser, transport layer, etc.

The crucial question is then: How do we model a web-based application? As software developers, we only model a small part of the big picture. Still, what we model has to fit in the larger scheme of things.

There are essentially two viewpoints in web-based applications. One is that of the end-user, which focuses on the *user experience.* The client's web browser and operating system, Internet service provider, and network hardware often influence end-user experience more than the server-side application itself. The second viewpoint is that of the problem owner, who wants to provide a service for the end user.

The developer of a web application has only control on a few artefacts, namely the server application software and, in certain cases, limited choice of web server and operating system. In other words, as developers we only have control over the last hub of the network of heterogenous components that make up the entire infrastructure (from application software to a client's web browser, see Figure 1) of a web-based system. Design choices are thus confined to the few components the developer controls.

On the other hand, user requirements are based on the user experience of the whole system. When we combine the user requirements and the design requirements, we end up with a great deal of constraints on the system requirements. Since we are very restricted and constrained in the choice of architecture and the way we have to design the application, it is more sensible to start the software development process with *design* and let the design drive the requirements gathering process. In other words, it makes more sense to state the more restrictive requirements (those on the architecture) first, and allow the more flexible requirements (those pertaining to the user experience) to adapt to the design. This strategy prevents the expression of overly optimistic requirements on the user experience side, on which a developer could consume precious resources only to discover that such requirements are infeasible given the architectural constraints imposed on a web-based application.

2 Essential parts of a model for web applications

There are three essential components that make up any web-based system:

1. Server side components: Server application code, web server, network layer, operating system, server hardware, and connection to the network.
2. Client side components: Client application code, web browser, network layer, operating system, and connection to the network.
3. Transport infrastructure: Server Internet Service Provider, inter-ISP network hardware and software, client Internet Service Provider.

Any of the above components can evolve independently. Typically the developer of a web-based application has only full control over the application code and maybe some limited control over the web server and operating system on the server side (e.g. a developer can opt not to upgrade the operating system). This factor compounds on the natural heterogeneity of the web environment (in which clients' network access can vary widely in method, speed, and reliability as can their hardware, operating system, or browser – not to mention different revisions of both hardware and software components). The end result is that, even if certain requirements are known to be satisfied at the inception of a new web service, the independent evolution of all the different components will again establish a completely heterogenous environment in a short time.

The only way a developer has to reduce the risks stemming from heterogeneity is to strictly adhere to published and endorsed standards (e.g., Internet RFCs [6] like the definition of HTTP). Strict adherence, in turn, limits the design freedom. On the other hand, the developer may want to provide services that are not easily delivered through standard interfaces and protocols (e.g., streaming video, teleconferencing with a shared whiteboard, etc.). In such cases, the developer may have to use proprietary solutions (e.g., more efficient or more feature-rich), at the price of increasing risks. Readers may have direct experience of how the process of keeping a number of browser's plug-ins up to date with evolving browsers and operating systems can become complex and cumbersome.

In a web-based design, we have thus a situation in which design freedom and risks stemming from independent evolution of components are inversely proportional. This is a consequence of the fact that we have to satisfy requirements that involve all the chain of connected components while only having control over application code and limited control over the operating system and server.

To alleviate this problem, we propose that web-based development should start with high level design, taking into account all the constraints imposed by the components that we don't have control on. This has implications on how to specify non-functional requirements but also for functional requirements.

3 Example

Suppose we are specifying requirements for a web-based system that provides geographical data such as route planning or finding the shortest paths between various destinations. If we start with requirements gathering we may find the following requirement:

The System shall provide the shortest path between two points to the user.

We can implement this requirement in two ways:

1. Get a request from the user to the server, compute the shortest path on server, and send the answer to the user. This option reduces network traffic, but increases latency.
2. Send the whole geographical database for the area of interest to the client, and have the request handled locally on the client side. Cache the database on the client in case of multiple requests. This option gives shorter response time, at the price of increased network traffic for a single request.

If we don’t take the architectural constraints into account, both are equally acceptable solutions, but once we start making assumptions on the network speed, browser capacity etc., one of them will be preferred over the other. So, it seems to be preferable to start considering the architectural requirements and let those drive the elicitation and capture of user requirements. In our example the second solution, while apparently preferable in terms of user experience, actually implies a number of requirements on uncontrolled components of the web application (e.g., it assumes that the user’s browser will be able to execute our code, that storage is available on the client side, etc.). Due to heterogeneity (users using different browser) and independent evolution (a new version of the browser may prove incompatible with our client-side code), the second solution is thus more risky. If a risk reduction strategy is followed, only the first solution is applicable, and we should take that into account while performing requirements analysis.

4 A Proposal

One approach to the development process for web applications would be to start with the identification of the set of independent components and a set of dependent component within the web-based application. The dependent components are likely to be those involved in the high level architecture. This entails that software development has to start with high-level design (or architectural design). Conceding that the software development is fundamentally evolutionary and adopting a spiral model of software life cycle [1,4], we thus propose the “Design-First-Spiral-Model (DFSM) for web-application development. Figure 2 is an adaptation of the original spiral model emphasizing the starting point being design rather than requirements elicitation. In DFSM, architectural constraints imposed by the nature of the web medium are taken into account *before* analysing the user requirements.

These additional constraints actually *reduce* the space of acceptable designs, and consequently the requirements analysis steps can be more focused. Furthermore, this restriction in focus can assist in resolving conflicts during requirements elicitation, because of the inherent priority imposed by design constraints. We believe that requirements engineering in this context is easier than in the general case, since fewer decisions have to be taken and fewer options are open to the developer. Therefore, each analysis step will either provide a valid refinement, or establish the very impossibility of satisfying the user needs, with less work and in a shorter time than in an open process.

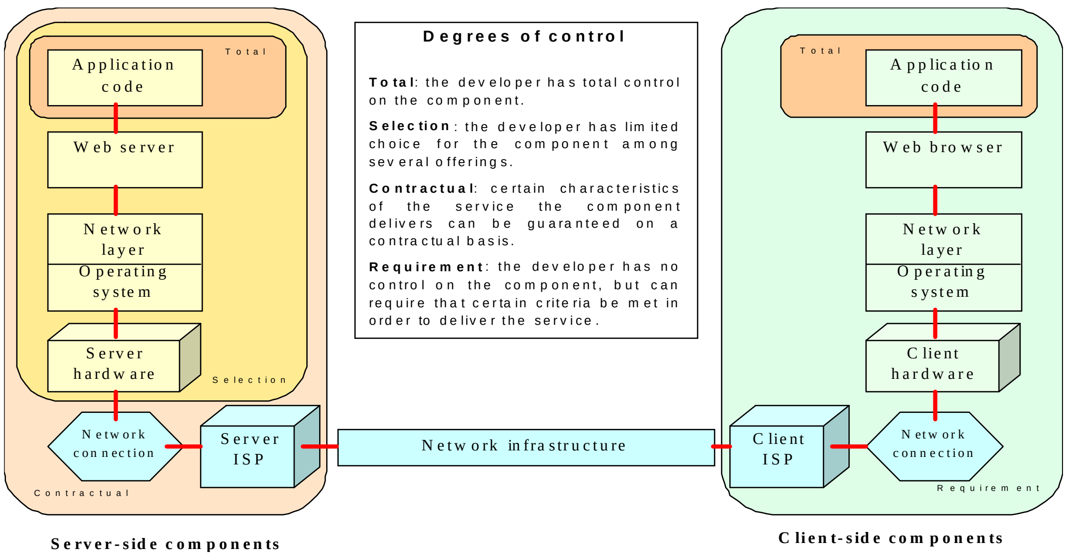
5 Conclusion

In this paper we have discussed how RE for web-based application imposes additional constraints on the system to be built. This is due to two factors: 1) the inherent client-server architecture and 2) the independent evolution of *heterogenous components*. In order to reduce associated risks while satisfying the constraints, the developers have to comply with standards. The consequence is less freedom in design. Since architectural design of web applications is more constrained than the “user experience” requirements, we believe that it is more advantageous to start the development process with high level design and adapt user requirements to design constraints. We therefore propose the *design-first-spiral-model* as a possible solution in improving web-based application development.

References

1. Boehm, B., A Spiral Model of Software Development and Enhancement, in *Software Engineering Project Management*, R.H. Thayer, ed., IEEE Computer Society Press, 1988, 128-142.
2. Burdman, J., 1999, *Collaborative Web Development,* Addison-Wesley.
3. Conallen, J., *Building Web Applications with UML,* Addison-Wesley, ISBN: 0201615770.
4. Dorfman M., System and Software Requirements Engineering, in *System and Software Requirements Engineering*, R.H. Thayer and M. Dorfman, eds., IEEE Computer Society Press, 1990.
5. Haggard, M., *Survival Guide to Web Site Development,* Microsoft Press, ISBN 1-57231-851-1.
6. The Internet Engineering Task Force. URL: http://www.ietf.org.
7. Reifer, D, Web Development: Estimating Quick-to-Market Software, IEEE Software November-December 2000, 57-64.

Figures:



Degrees of control

Total: the developer has total control on the component.

Selection: the developer has limited choice for the component among several offerings.

Contractual: certain characteristics of the service the component delivers can be guaranteed on a contractual basis.

Requirement: the developer has no control on the component, but can require that certain criteria be met in order to deliver the service.

Figure 1. Some of the components in a web-based application, and the degree of control the developer has on them.

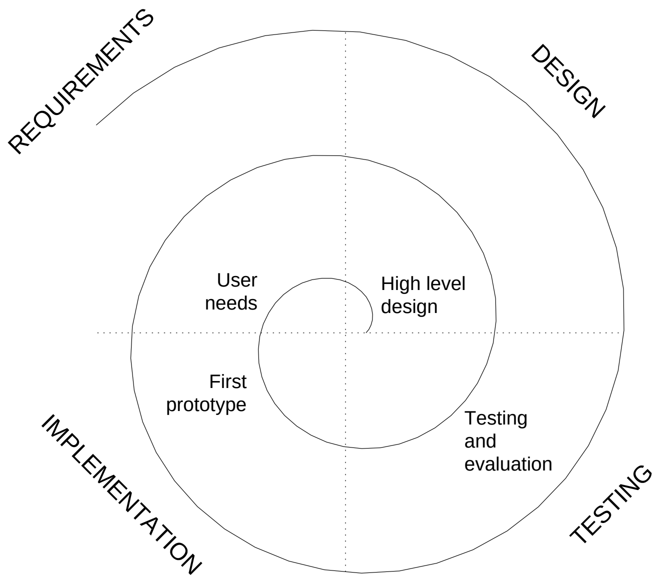


Figure 2. The Design-First Spiral Model for web applications.