Investigating the impact of usability on software architecture through scenarios: a case study on web systems

Tamer Rafla*, Pierre N. Robillard† and Michel Desmarais‡
Software Engineering Research Lab, Ecole Polytechnique de Montreal
P.O. Box 6079, Station Centre-Ville, Montreal, QC, H3C 3A7, Canada

Abstract

Usability has primarily been served by separating the user interface from the remainder of the application. However, several researchers have recently determined that there is a direct relationship between architectural decisions and usability requirements. This leads us to conclude that more attention should be devoted to usability driven architectural analysis methods. We present a case study that consists in adapting an existing software architecture analysis method (SAAM) for the purpose of deriving the interdependencies between architectural characteristics and usability requirements. More specifically, we investigate the impact of implementing usability requirement changes on the architecture. Potential design solutions that accommodate the corresponding usability mechanisms into the web software architecture are presented, along with their rational and the process by which they are obtained. We conclude by recommending how usability issues can be dealt with proactively during the design of the architecture and present the need to integrate those usability requirements in a software engineering process.

Keywords: Software architecture, Usability, Scenarios, Web applications, SAAM

1. Introduction

Humans play an active and essential role in the operation of interactive software and the user interface (UI) has become an essential part of many software systems (Hix and Hartson, 1993). The architectural answer to building a UI is based on the principles of modularity and separation. This
separation made it easier to make modifications to the UI and usability seemed to be primarily a property of the presentation of information. It was not considered in software architecture design, due to the widespread assumption amongst software engineers that it has to do only with the visible part of the system. However, this view is now challenged (Bass and John, 2003). A more recent perspective on the UI is that the software architecture needs to be explicitly designed so that the final system can satisfy usability requirements. Even if the presentation and the functionality are well designed, the usability of a system could be greatly compromised if the underlying architecture does not support user concerns. In addition, it often proves hard to make the necessary changes to a system to improve its usability. A reason for this is that many of the required usability driven changes involve changes to the system that cannot be easily accommodated by the software architecture (Rafla et al. 2004).

We investigate the link between usability requirements and software architecture in a case study of a web-based application. The purpose of this paper is twofold: 1) adapt a scenario-based approach to analyze a software architecture for usability; 2) determine how usability requirements can be supported and study their impact on the architecture by proposing web-specific design solutions.

We begin by discussing the scenario-based method that was used to analyse the web architecture for usability. Web-specific design solutions are then proposed to determine how usability requirements are supported. Their impact on the architecture is investigated and the results of the late implementation of usability on the architecture are analysed. We close by discussing how usability issues can be dealt with proactively during the design of the architecture and provide topics of future work.

2 SAAM: Software Architecture Analysis Method

The Software Architecture Analysis Method (SAAM) (Kazman et al., 1994; Kazman et al., 1996; Clements, 2001) was among the first to address the assessment of software architectures using scenarios. It has traditionally been applied to the development of related software qualities such as maintainability (Haggander and Bengtsson, 1999; Bengtsson and Bosch, 2000; Dobrica and Niemela, 2002). Since very little architectural assessment techniques for usability exist, can the SAAM be also considered for usability? This avenue is hence explored after a short description of the method. SAAM, (Fig. 1) consists of the following four steps:
• **Scenario development:** The first step is to illustrate the kind of activities the system must support and the kinds of anticipated changes that will be made to it. Since usability is often defined in a very abstract fashion and is not well interpreted by software designers, scenarios are a good way of synthesizing individual interpretation of software quality into a common view, therefore they can make usability requirements more specific.

• **Architecture description:** There is no architectural description of the system and the product is thoroughly studied and an architectural diagram is developed. This activity is recommended to be carried out in parallel with the previous activity in an iterative mode. The final version of the architecture description together with the scenarios serves as the input for the subsequent activities of the method.

• **Individual scenario evaluation:** SAAM evaluates a scenario by investigating which architectural elements are affected by that scenario. The impact of the modifications associated with each scenario is estimated by listing the components that are affected.

• **Overall evaluation:** Finally, after scenario evaluation, the results are interpreted to draw conclusions concerning the software architecture and to assess the impact of the usability scenarios on the web architecture.

---

![Diagram](image)

Fig. 1. Steps of the SAAM (adapted from Kazman et al., 1994)
In the next sections, the usability driven analysis of the architecture is conducted by elucidating the four phases of SAAM.

3 Determining the proper set of scenarios

In this section, step 1 of the method is performed. Bass and his team at the Software Engineering Institute identified a collection of architecturally significant scenarios that expressed a usability issue (Bass et al., 2001). Those scenarios are common to many interactive systems and are not related to the domain functionality of any system. The ones that will inevitably increase the usability of the system that serves as the case study for this work are extracted and their respective usability benefits are illustrated. Those benefits encompass indicators of efficiency, problem solving and learnability (effectiveness), and customer satisfaction (Nielsen, 1993; Schneiderman, 1998; ISO 1998b) (Table 1). Those scenarios are applicable to web applications and can be considered for other web systems that suffer from the same usability issues identified here. However, we do not claim that this set of usability scenarios covers all the usability problems identified in internet applications.

- **Checking for correctness**: Users may make errors they do not notice. Depending on the context, error correction should be enforced directly or suggested through system prompts. Mistakes made entering credit card billing information, for example, can require costly manual follow-ups. The checking for correctness scenario will be validated if and only if a mechanism that allows for all entered fields to be validated is incorporated.

- **Retrieving forgotten passwords**: Users may forget their password and retrieving it may be difficult or may cause lapses in security. Systems should provide alternative strategies to grant users access. It becomes tedious to contact the site administrator every time the password is forgotten. Since this is not always accomplished in a timely fashion, users should be able retrieve it on their own without requiring the help of anyone. Thus, retrieving forgotten passwords is another scenario worth considering.

- **Modifying interfaces**: Poorly designed interfaces increase user errors. After looking at the system from a user’s point of view, it became evident that it is lacking several important aesthetic issues that need to be considered to make the site more usable. Good hypermedia design practices should be used, to develop applications that are easy to use, provide friendly navigational spaces, and seamlessly integrate the underlying transactional behaviour.
• **Providing good help:** It doesn’t matter how intuitive an application is; if it is rich in functionality, then the user will inevitably need some kind of assistance. And as web applications become more complex, providing help is becoming inexorably important. Since the system is not equipped with a help system and seeing that some fields are hard to comprehend, providing good help is another scenario that should be taken into consideration.

• **Supporting international use:** The increasing diversity of the Internet has created a tremendous number of multilingual resources on the Web. Users may want to configure an application to communicate in their language or according to the norms of their culture. Systems should easily be configurable for deployment in multiple languages.

<table>
<thead>
<tr>
<th>Usability Aspects</th>
<th>Effectiveness</th>
<th>Efficiency</th>
<th>Satisfaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Providing good help</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Modifying interfaces</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Checking for correctness</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Retrieving forgotten passwords</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Supporting International use</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 1. Linking usability aspects to scenarios (adapted from Bass and John, 2003)

4 **Describing the architecture**

4.1.1 **Replan: The meeting management system**

Implemented using the Unified Process for EDUcation (UPEDU) (Robillard et al., 2003), Replan is a web-based meeting management system aimed at organizers of meetings where the
number and geographic dispersion make scheduling difficult. This system would allow meeting coordinators to send availability requests to a set of individuals, so that each of them can specify personal availability periods. The set of availability periods would then graphically be represented using a calendar tool to enable a coordinator to visualize the relevant information at a glance, making the scheduling decisions easier.

4.1.2 Replan’s architecture

The Java Enterprise Edition (J2EE™) web architecture (Inderjeet et al., 2002) consists of web browsers, a web server and a network connection (Fig. 2 and 3). Browsers request web pages from the web server which distributes JavaServer Pages (JSP) of formatted information to clients. This request is made over a network connection and uses the HTTP protocol.

In this implementation of the architecture, the JSP files not only contains the HTML code necessary to produce the UI, but are also responsible for processing the user’s requests in order to perform the necessary business logic. Hence, the JSP contains a significant amount of embedded Java code responsible for storing the session and linking JSP pages together. The JSP component is in charge of the request processing and the creation of any objects used by the JSP, as well as deciding, depending on the user’s actions, which JSP page to forward the request to. Functionalities of this component include the validation of request parameters with JavaScript, the authentication of the user and the post of user forms by connecting to the Database through the Java component, before dispatching the request to another JSP for presentation of the response. The Java component of the system is interacting with an Oracle9i Database. The Tomcat web server combines the components according to configuration information that specifies the connection between these components. In fact, interdependent JSPs were grouped in modules since the system is modeled from a high-level point of view and the relationship between those JSPs is not of an interest to us. The application control is decentralized, because the current JSP page being displayed determines the next page to display. The page flows are embedded on the links between the pages.

We wish to verify if the incorporation of usability requirements would be possible with different implementations of the web architecture. Two architectures were selected for the following reasons:

- To ensure the validity and the correctness of the results.
To show that the proposed solutions are possible with different implementations of the architecture.

The other excluded projects suffered major drawbacks such as failure to meet the requirements. Some features were nonexistent and the system was not fully operational.

Table 2 compares the two selected architectures, comparing the number of Java and JSP files since they are the essence of the JSP web architecture. The number of tables created in the database is also illustrated. Team A has 29 Java classes counting for 160 KB and 15 JSP files for 50 KB. The business logic represents 76% of the overall structure of the system while the interface represents only 24%. This first system was properly implemented since business logic is well encapsulated in the Java classes and the JSP were only used to embed some HTML code, in other words for the UI. However, Team B has more JSP files than Java classes. It has 7 Java classes for a total size of 74 KB, while 24 JSP files totaled 200 KB. The business logic represents only 26% of the overall structure of the system while the view represents 74%. Some business logic is incorporated in the JSP component rather than in the Java and the guidelines of the web architecture were not properly followed. Fig. 2 and 3 illustrate the architecture of the two selected systems.

Table 2: Structure of two versions of Replan

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>JSP</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number</td>
<td>Size (KB)</td>
<td>Number</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Team A</td>
<td>29</td>
<td>160</td>
<td>15</td>
</tr>
<tr>
<td>Team B</td>
<td>7</td>
<td>74</td>
<td>24</td>
</tr>
</tbody>
</table>
In this section, step 3 is exemplified. Design solutions to satisfy the chosen usability scenarios are presented. The magnitude of the impact is represented as follows:

- **No impact**: A small number of web components is modified for the two selected architectures and the development effort is low. The required changes are categorized as architecturally independent and still can be implemented once the system has been designed.

- **Minor impact**: The number of modified architectural components is different however the scenarios necessitate more or less the same implementation effort. However; it is beneficial to consider those usability requirements at earlier stages of development.

**Fig. 2. Team A’s architecture**

**Fig. 3. Team B’s architecture**

5 **Scenario evaluation**

In this section, step 3 is exemplified. Design solutions to satisfy the chosen usability scenarios are presented. The magnitude of the impact is represented as follows:

- **No impact**: A small number of web components is modified for the two selected architectures and the development effort is low. The required changes are categorized as architecturally independent and still can be implemented once the system has been designed.

- **Minor impact**: The number of modified architectural components is different however the scenarios necessitate more or less the same implementation effort. However; it is beneficial to consider those usability requirements at earlier stages of development.
Major impact: Major components are reworked and the required changes necessitate a substantial development effort. This effort is entirely dependent on the implementation of the architecture. Such modifications should be considered at the design stage due to the considerable effort they entail.

5.1 Checking for correctness

With server-side validation, the processing of the form fields is performed after they are submitted to the server. Using JavaBeans (Inderjeet et al., 2002), the designer needs to create a “validation bean” that will perform secure server-side validation on the entered data. Implementing a validation bean requires several steps. First, a data bean that has fields identical to the fields of the form must be designed. Second, the validation rules for these fields must be devised. These two steps are implemented in the Java component of the application. For example, if the validation of the user’s form is required then a validation bean with its respective validation rules needs to be added to the User.java class. Finally, all JSP files must be redesigned to work effectively with every validation bean that was introduced. They are slightly modified to incorporate the few lines of code responsible of initializing the bean and directing the request to the Java classes for validation. This approach separates the business logic (Java component) from the presentation (JSP component).

This approach of providing form validation entails the modification of the same number of components and files for both architectures (Fig 4 and 5). Surprisingly, after detailed analysis of both architectures, it became evident that the JSP contains some embedded Java code responsible for storing all objects and also transferring the control to another JSP in charge of validation. Unfortunately, business logic should be encapsulated in the Java component while the presentation in the JSP component. This means that all the inserted Java code in the JSP should be moved to the Java component and the JSP should contain only HTML code responsible for displaying the form. However, this method involves a significant development effort in both architectures since a large number of components are reworked. The impact of this scenario on the web architecture is classified as intermediate.
5.2 Retrieving forgotten passwords

The Web has moved beyond purely open content available to all. It is now commonly used to provide information that ought to be restricted in some way to members. One common method of restricting access is to ask users to enter a username and a password. This simple combination can be a source of annoyance and frustration to users because the user’s password can be forgotten. Adding a link in the login page that will redirect you to a lost password page where you can enter your username is the simplest solution. Once the request is submitted, the password is going to be mailed to the user. This is fairly simple to implement since the username is basically the email of the user according to the specifications of the system and the modifications are reasonably straightforward to implement (Fig. 6). A few lines of Java code were added in the User class (Java component) to select the user’s password from the Database. Also, a LostPassword.JSP page, which is linked to the Login.JSP page, is created to mail that password to the user. That newly created JSP is included in the Main module with the Login.JSP. The changes required to instigate this scenario requires the developer to rework the JSP and the Java components (Fig. 6). Fortunately, these adjustments seem to be rather simple since they don’t require a significant programming effort and can be added at any stage of development. This solution works for both architectures. The Java component is slightly modified for both teams and a new interface has been
added to the User Module in the JSP component. The JSP architecture can easily accommodate the mechanism to retrieve forgotten passwords. This scenario doesn’t have an impact on the system’s architecture since it requires the same development effort for both teams and it is independent of the overall implementation of the system.

5.3 Supporting international use

In today’s competitive world, many businesses are becoming global in order to reach the widest possible target audience. This is a problem, because web applications developed by those organizations must be suitable for use by customers who use and understand different languages. Therefore, there is an increasing need to develop and design cost effective web solutions capable of delivering content in multiple languages. To be considered “Multilingual” an application has to support the following elemental categories of features:

- **Data representation**: support all the characters set encodings that any end users may use
- **Data display**: display strings, currency, number, date formats weights and measures, etc., in a format an end user will understand
• **Data input:** must understand the format of the data being submitted to the application. For example, a French speaking user will submit the date “9/11/04” for November 9th 2004 (dd/mm/year), while an American English speaking user would submit the same date as 11/09/04 (mm/dd/year). Knowing the format of submitted data is essential to data integrity.

The internationalization of data-intensive web applications can be achieved by replicating the system. This means that it is necessary to replicate the database, translate the HTML code embedded in the JSP files that produces the web pages, and modify the Java code to manipulate the newly introduced set of encodings. The required translation language is assumed to be alphabetically-based and languages that use symbols are not considered (e.g. Chinese, Japanese, etc.)

### 5.3.1 Database extension and queries rewriting

The main characteristic of a data-intensive web application is the interaction with database servers, which provide dynamically the data to be published in the requested pages. The internationalization process requires the modification of the information stored in the database, thus an extension of the database schema is necessary in order to support such translations.

- For each table $T$, representing an entity type with attributes subject to translation, a new table $T_{\text{trans}}$ has to be created with $T_{\text{trans}} = \{ PK_T, \text{language}, \{ A_1, \ldots, A_n \} \}$

- For each table $R$, representing a relationship type with attributes subject to translation, a new table $R_{\text{trans}}(PK_R, \text{language}, A_1, \ldots, A_n)$ has to be created.

($PK_T$ is the set of attributes representing the primary key, and $\{ A_1, \ldots, A_n \}$ the set of new attributes)

Rewriting the queries is indispensable in order to include the modified attributes in the result of each query. The characteristics of this rewriting are as follows:

- Assuming that the system needs to be used by English speaking users. This language is obtained by applying a custom rule to the preference language list coming from the HTTP request. French is denoted as being the master language and English the requested language.

- For each attribute $A$ subject to translation, $A$ is replaced with two attributes French-$A$ and English-$A$ containing its value in French and English respectively.
Extending the database and rewriting the queries would solve the data display, data representation and data input issues. Only an additional table for each entity or a relationship type object to translation is added. The worst case situation, according to Aykin (1999), given a database containing $m$ tables with attributes subject to translation, the number of new tables that have to be added is equal to $2m$. Thus, all the database tables should be duplicated, and the queries nested in the Java component should be reworked for both systems (Fig.7)

5.3.2 Translating the web pages

Rewriting the queries has the aim to extract from the database the requested data together with their translation in the desired language. All JSP files are thoroughly examined and all keywords are translated. Considering that English represents the requested language, all attributes should be available in that language. The required modification does have a major impact on the architecture of Replan. Different implementation effort would be necessary in order to incorporate the supporting international use scenario in both teams since their JSP, Java and Database components are different and the development effort is reasonably high (Table 2).

5.4 Modifying the interface

Usability is no longer a luxury for web development. As the popularity and acceptance of web-based systems increases, the issues related to usability and UI design increased in frequency. While people may grow comfortable and assured with one of the most obvious issues of concern – that of security – they may find it difficult to feel assured at some web applications because of their lack of standards and practices associated with the field of usability. And by applying Nielsen’s usability heuristics (2000; 2002), the daunting task of improving features on a data-intensive web application can be approached. Using this set of heuristics as touchstones to the evaluation, it became clear that the system suffers from usability issues such as the absence of a title or a header that describes screen contents, and an inconsistent icon design scheme and stylistic treatment across the system. The design of the JSP files that contain the HTML fragments that make up the UI should minimize the user’s memory load by keeping the same navigation bar and page layout.
A flexible, minimalist aesthetic design eases the users’ eyes and allows them to concentrate on the task at hand without distraction in the interface design (odd colours, etc.). The use of a linear progression easily encourages a narrative flow as well as giving the user a feeling control over the process. Matching the system and real world expectations means that when the task is “completing a form”, the flow in completing that form should be as smooth and problem-free as possible. This scenario can be satisfied by looking at the JSP files that suffer from the usability issues found by applying Nielsen’s usability heuristics, and then reworking those files. Team A has 15 JSP files while Team B has 24 (Table 2). Therefore the probability of Team A suffering from major usability issues in their interfaces is lower than Team B’s. Also, the effort required to modify the JSP component is entirely different for both teams. This scenario has a minor impact on the architecture.
since it requires the rework of only one component of the architecture. The objective here was not to list all the violated heuristics but to merely state that they can be corrected without affecting in a severe way the architecture of both applications.

### 5.5 Providing good help

Users don’t tend to read printed documents before using a Web application, so online help needs to be easy to use and allow quick location of the necessary information. A help system should be well organized and should present users with a clear path to the information that they need either through context-sensitive help or by providing intuitive and advanced search capabilities. Since most web applications are simply several pages linked together, if the user has a question or requires help, a simple hyperlink to FAQ does the trick.

There are many different types of Internet web application help systems. By far, the most popular method is to simply create one page with every help item for every page on the website. Basically, everything is on a page. Users click on the hyperlink and they receive everything they could ever want about the site. The problem is that they simply had a quick question and though the help option would generate a quick and simple answer, yet they entered a world where they must scroll through massive amounts of data in hopes of finding the answer to the now forgotten answer. The other more advanced method is a full-blown help application. Once the help icon is clicked, the user dives into an application for the help system. This type of application is more in common with online books. This kind of help application is more thorough because it covers the entire application, including other aspects such as customization or installation of client components.

A method that lies between the two in functionality is proposed. This is an easy design solution where each page on the site has its own separate help page invoked by calling the `Help.JSP` file which is going to be added to the JSP component. Every web page would include this simple hyperlink (Fig. 8). This newly introduced page will, once invoked, extract the referring page name that called the help system and store it in a variable. That variable will then lookup the corresponding help file in HTML format in a newly created table in the database that contains an index on the calling page’s name field. Once retrieved, the help text will be inserted into the JSP and will be presented to the browser. A back button, containing the link to the referring page should also be added to return the browser to the original page that started this entire help journey. The database will contain one row in a table for every page. Each row has all the required
text to be displayed for the application. Also, if the help text changes, only an update to the table row for the specific page’s entry is required. There are no pages to change and it’s a clean and simple method to integrate a help system into the system. This solution was sufficient because the Tomcat web server can support server variables (such as REFERER_URL) that will notify the target page of the name of the sending JSP document. The implementation of this scenario is dependent on the size of the JSP component, more precisely the number of JSP files. According to Table 2, Team A has 15 JSPs with while Team B has 24. The greater is the number of interfaces (JSP files), the bigger is the help system since every interface should have its own help content. The addition of a help system has a minor impact on the architecture of Replan. The effort needed to incorporate this scenario is entirely dependent on the actual implementation of the JSP component and the Database. This entails doing nothing to the web application since the help system is completely self-contained.

Fig. 8. Help system (Team B)
6 Overall evaluation

6.1.1 Summary

If the required development effort is low and is the same for both Team A and Team B, then we conclude that the required modifications are architecturally insensitive since the two teams have distinct architectures. The retrieving forgotten passwords scenario seems to be independent of the architecture, and the proposed solution would necessitate the same effort for both systems. The Java component will be modified to retrieve the user’s password from the database, and a lost password page (added to the Main module) is created. Fortunately, these modifications will leave the architecture unchanged, minimizing the impact on the architecture.

If the envisioned usability modifications require a relatively small number of architectural components to be reworked, then the changes have a minor impact on the architecture. In this case, the required changes necessitate a considerably small development effort. The modifying interfaces scenario necessitates the listing of all the interfaces that are lacking important usability guidelines established by Nielsen, and correcting them. The greater the number of JSP files, the longer it will take to thoroughly examine those files and find usability problems. Since both teams have different implementations of the JSP component, they would require dissimilar implementation efforts. In fact, if Nielsen’s guidelines were properly followed during the architectural design phase, then the effort to having to go through all the interfaces could have been avoided. Also, for the providing good help scenario, a new table containing the help information has to be created. It requires solely the modification of the database, but no major components need to be modified. All JSP pages should be redesigned to include a link to the new Help page. The greater the number of JSP files, the larger is the new help database table. Therefore, both of these scenarios entail a minor impact on the architecture given that it requires roughly the same development effort for both teams and does not involve the revision of major components.

The supporting international use scenario implicates primarily the modification of the database. Every table in the database is subject to replication since data input and data display in an appropriate language should be ensured. The greater the number of tables, the greater is the effort to validate that scenario. Team A has five tables while Team B has 11. Thus, the effort required to develop the database is considerably greater for the second team. Unfortunately, extending the database does not denote that the scenario is satisfied as the JSP and Java components need to be revised as well. All interfaces should be translated. Generating a large number of JSP files would
require a significant effort to translate the HTML fragments embedded in those files. Also, the database queries need to be rewritten to manipulate the newly introduced parameters in the database. In conclusion, the supporting international use scenario has a major impact on the architecture of Replan since it requires a substantial effort to integrate it in two noticeable different architectures and all the components of the architecture are modified. Table 3 lists the chosen scenarios and their impact on the web architecture.

Table 3: Results of scenario evaluation

<table>
<thead>
<tr>
<th>Usability Scenario</th>
<th>Architectural impact</th>
<th>Files affected</th>
<th>Components modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checking Correctness</td>
<td>Intermediate</td>
<td>• 4 JSPs that contain forms&lt;br&gt;• 3 Java classes</td>
<td>JSP and Java</td>
</tr>
<tr>
<td>Retrieving Forgotten Passwords</td>
<td>No</td>
<td>• 1 JSP file (Help.jsp)&lt;br&gt;• 1 java class (User.java)</td>
<td>JSP and Java</td>
</tr>
<tr>
<td>Providing Good help</td>
<td>Minor</td>
<td>• All JSP files&lt;br&gt;• Database tables</td>
<td>JSP and Database</td>
</tr>
<tr>
<td>Modifying Interfaces</td>
<td>Minor</td>
<td>• All JSP files suffering from usability issues</td>
<td>JSP</td>
</tr>
<tr>
<td>Supporting international use</td>
<td>Major</td>
<td>• All JSP files&lt;br&gt;• All Java classes containing SQL queries&lt;br&gt;• Database tables</td>
<td>JSP, Java and Database</td>
</tr>
</tbody>
</table>

6.2 Discussion

The scenario retrieving forgotten password that covered only one indicator of the effectiveness aspect of usability (accommodating mistakes) (Table 1) has no impact on the architecture. The providing good help scenario which covers both indicators of the efficiency aspect of usability (supporting problem-solving and learnability) and the modifying interfaces scenario which covers accelerating error-free portion and accommodating mistakes aspects of effectiveness, necessitate minor modifications to the architecture. The checking for correctness scenario covers four usability
indicators, encompassing both effectiveness and efficiency aspects and has an intermediate impact on the architecture. The internationalization scenario supports all three aspects (effectiveness, efficiency and satisfaction) and has a major impact on the architecture. In addition, this scenario is the only one that supports increasing user’s confidence and comfort. Hence, there is a relationship, as shown in Fig. 11, between the number of usability aspects of a scenario and the impact it will have on the architecture. The greater the number of usability aspects, the greater is the impact on the architecture. This work can be useful to a design team to evaluate and design a web architecture. It would enable them to select and implement at later stages of development the scenarios that can easily be incorporated into the architecture (the ones that cover a low number of usability aspects). It would also allow them to consider during the elaboration of the architecture, scenarios that require the overall structure of the system to be reworked. Those scenarios are said to be ‘complex’ since they cover many cognitive aspects, such as increasing individual effectiveness, reducing the impact of mistakes and the increase of the comfort of the user.

Also, most of the chosen usability scenarios can be integrated into both existing systems without incurring major changes to the architectures except for one, the supporting international use scenario. It is the only scenario that requires all web components, as well as the database, to be reworked. It does not entail the same implementation effort for both systems. Having a greater number of Database tables means that the effort for update and maintenance is higher, since every instance in that database has to be maintained. Therefore, the supporting international use has an impact on another quality attribute, maintainability. Regrettably, this last scenario is not the only one that has an impact on maintainability. If the separation of concerns is not established, as it should be, then the checking for correctness scenario has also an impact on that quality attribute. Problems can arise when applications contain a mixture of data access code, business logic, and presentation code. Such applications are difficult to maintain because the interdependencies between all of the components can cause strong ripple effects whenever a change is made at any given location. The web components are difficult to reuse as they depend on many other classes and require maintenance in multiple places.
6.3 Recommendations

Recent studies confirm that a significant part of the maintenance costs of software systems is spent on usability issues (Li and Henry, 1993). These high costs are largely due to the usability requirements which will not be unveiled until the software has been implemented or deployed. This analysis shows that the usability scenarios that entail a substantial architectural impact (checking for correctness and supporting international use) have an effect on the maintainability of a web application. The system architecture was not properly designed to cope with the incorporation of late usability changes. We firmly believe that the architectural impact of the chosen usability scenarios could have been less serious if the architecture was different. We discuss in the following section how the scenarios that entail a significant impact on the architecture, namely the checking for correctness and the supporting international use scenarios, can be instigated and their architectural impact minimized.

In any web-based application, the validation of the data can be checked in two ways. The first method is the approach that was discussed previously, validation on the server side while the
second is to perform validation on the client side, before anything is submitted to the server. This is usually done with JavaScript (JS) running on the client’s browser. A few JS functions that handle common validation tasks are created. Activated by clicking the form’s “submit” button, they test field content for proper formatting. If they detect a problem, the form submission is aborted and an error message is displayed. Client-side validation has numerous advantages for the server and the user. It lightens the load on the server and allows the data to be validated before it is sent for processing. Since validation takes place on the client machine, there’s no delay from the client contacting a remote server. Besides the user getting a quicker response, server processing power is conserved. The implementation of this approach in the architecture would require transferring the control responsible for validation to an external component responsible for regrouping all JavaScript validation files. Fig. 9 illustrates the new proposed architecture.

The main component that needs to be reworked in order to satisfy the supporting international use scenario is the database as it is closely coupled to the database queries in the Java component. If this scenario was introduced during the architectural design phase, would the design of the database have been different? The software architecture is the artefact that embodies the earliest design decisions and if this scenario was introduced during the architectural description phase, then we definitely believe that the system would have been designed in a way that would accommodate that scenario. Both teams would eventually have spent more time on the design of their database since it has an effect on their development effort of the Java component. Hence, this scenario should have been related with decisions taken early in software development process. UPEDU®
lists a software architecture description as an outcome of the elaboration phase, while user-centered development processes such as ISO 13407 do not even mention software architectures. And since architectural decisions can preclude the delivery of a usable system, we strongly believe that the inclusion of architecturally sensitive usability scenarios in UPEDU® could help us design more usable systems.

7 Conclusion and future work

In the software and usability engineering communities, little work has been presented on how to integrate usability in the design of software architectures. We contribute to this ongoing research by discussing how the SAAM, a scenario-based architecture analysis technique can be adapted to assist software designers in designing software architectures that support usability. Our application of SAAM consists of four major steps: (1) determining the set of usability scenarios, (2) describing the software architecture, (3) scenario evaluation and (4) overall evaluation and discussion.

The first step of the method consisted in extracting from the work of Bass et al. the scenarios that are applicable to web applications along with their respective usability benefits. Potential web-based design solutions were then proposed to highlight the impact of those scenarios on the web architecture once instigated. The proposed solutions have been obtained by means of inductive process that guarantees that these solutions are possible, albeit not necessarily the only one. After analyzing two distinct architectures of the same system, our study shows that four of the six chosen scenarios could be implemented into an existing system without incurring major architectural changes. The usability scenario that covered one usability benefit (namely retrieving forgotten passwords) has no impact on the architecture, while the ones that covered two benefits (such as the modifying interfaces and providing good help) have a minor impact. In fact, the scenarios that exposed a great number of usability aspects (checking for correctness and supporting international use) have a major impact on the architecture. There is therefore a relationship between usability scenarios and their architectural impact. And, besides having a serious impact on the architecture, these previous scenarios seem to have an effect on the maintainability of the system if they were to be implemented at later stages of development. This effect could have been avoided if the usability requirements were defined and considered in the architectural description phase. Both teams constructed their architectures without paying sufficient attention to usability requirements and whether it can easily accommodate late modifications. This is why relating system’s usability to
architectural decisions taken during the design phase is crucial, and the architecture should be well designed to incorporate usability requirements. Software development processes that evolved from human-centred design or human-computer interaction, like the ISO standard 13407 do not define software architectures and therefore the usability scenarios that have an impact on the architecture could not really be implemented. On the other hand, since UPEDU® is being defined as architecture-centric, the next step of this research would be to explore how it can accommodate user concerns and also investigate how the architecturally-sensitive usability aspects can be integrated in the process.

Acknowledgments
We are grateful to the students that participated in the project course at the winter 2002 semester and we thank them for their collaboration in providing the details of their work. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) under grants A0141 and 121923-03.

References
Aykin, N., 1999. Internationalization and Localization of Web Sites. Proc. of the 8th International Conference on Human-Computer Interaction (HCI), Munich, Germany, pp. 1218-1222
Inderjeet, I., Stearns, B., Johnson M., 2002. Designing Enterprise Applications with the J2EE Platform, Addison-Wesley, CO.

Tamer Rafla is a M.A.Sc. student in the Software Engineering Research Lab at Ecole Polytechnique de Montreal. He has held several positions in the telecommunication and software industries after receiving his B.A.Sc from Concordia University. His research interests include usability and cognitive aspects in software engineering, requirements engineering, software architecture and software processes. He is an academic member of the IEEE, Quebec Order of Engineers, and a founding member of the WWISA.

Pierre N. Robillard is a full professor in the Department of Computer Engineering at the Ecole Polytechnique de Montreal where he heads the Software Engineering Research Lab. His research interests include cognitive aspects in software engineering, software collaborative work, software measurement, software engineering education and software processes. He received his B.Sc. degree from the University of Montreal in 1969, an M.A.Sc. degree in computer communications from the University of Toronto in 1973 and a Ph.D. degree in electrical engineering from University of Laval in 1977. He recently co-authored the book “Software Engineering Process with the UPEDU”. He is a licensed professional engineer in the province of Quebec, Canada and member of the IEEE, ACM, EACE and CIPS.

Michel Desmarais is an associate professor at the Computer Engineering Department of Ecole Polytechnique de Montreal since 2002. His field of expertise is in the domains of Human-Computer Interactions and Artificial Intelligence. He has over ten years of experience in software project management. He was principal researcher of the Human-Computer Interaction group at the Computer Research Institute of Montreal between 1990 and 1998, where he directed a research program in HCI and computer assisted learning, and was involved in a number of research projects in close collaboration with private corporations. From 1998 to 2002, he directed the Web services department in a private company and directed a number of R&D Web-based software projects. He is the author of a book, ten scientific journal publications and over 20 book chapters and conferences papers.