# Web Forms and XML Processing: Some Quality Factors of Process and Product

Mário Amado Alves[2]
Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa
maa@di.fct.unl.pt

## Abstract

"The web is bad; really bad."—observed Jakob Nielsen three years ago about *The Web Usage Paradox* [1]. The true *paradox* today is that *science and technology* institutions have bad sites! Programmes devoted to the *information society* itself have bad sites!! Hopefully this international conference will make a difference. At least show Nielsen and myself are not just "fools on the hill" and, hopefully, help management make wiser decisions regarding web development strategies and, correlatively, better technical staff selection. The current paper contributes to this goal by exposing a method for the development of complex web services which embodies a number of quality assurance items and has passed the test of real web service deployment, featuring user authentication, multiple forms, recorded data, and automated page creation. The method and the case are described with incursions into selected technical details. Quality factors are explicitly or implicitly associated with each described item.

## 1. Web Quality Manifesto

The web is bad; really bad. [1]

And it is getting worse. Already three years have passed since web usage specialist Jakob Nielsen's article [1] has appeared, and still his observations are right on the mark: "90% of all commercial websites are overly difficult to use due to *bloated* page design that takes forever to download, internally focused design that *hypes* products without giving real info, *obscure site structures*, *lack of navigation support*, narrative writing style optimised for print, not for the way users read online, etc." ([1] abridged, original emphasis maintained).

In the current paper I add a couple of items to this list.

Why is it even worse, today? Well, for one, *institutional* sites are bad. In fact, the true *paradox* today is that *science and technology* institutions have bad sites! Research programmes devoted to the *information society* itself have bad sites!! The 2000 Olympics site was bad. Oracle's site is bad. I submit a Law of Inverse Quality: *the greater the institution, worse the site.*

I hope this international conference—in particular panel PN1.2 entitled *Qualidade nos Sistemas de Informação da Administração Pública: o Início duma Cruzada* (*Quality in Public Administration Information Systems: the Start of a Quest*), in regard to institutional sites—will make a difference. At least it will show Nielsen and myself are not simply "fools on the hill"—not anymore. And, hopefully, it will help educated management make wiser decisions regarding web development strategies—and, correlatively, better technical staff selection.

The current paper contributes to this honourable goal by exposing a method for the development of complex web services which embodies a number of quality assurance items and has passed the test of real service case deployment.

Is the Web really worse today, rather than three years ago? Yes, definitely. Here is a *recent* (2000) observation from the same author of [1]: "If you are going to go and buy something on a new website, you will fail. If you go to a new website, you will not be able to use it." (http://www.wired.com/news/business/0,1367,40155,00.html)

Web quality is a twofold problem: technical and social.

**Technical.** A veritable plethora of techniques and methods exists today to develop web services. Judging from the results, most of them are *bad*. The current paper presents a method that emphasises some quality factors of process and product. These factors are explicitly or implicitly associated with each described technical item. Bottom line: it is a *good* method. It is proven. The rest of this paper will deal with the technical aspect only.

**Social.** The social problem is to convince people to use good methods and techniques. To be *quality-aware*. People like Jakob Nielsen [1] is trying to pass the message for some years now. The message is simple: *User: demand web quality. Web service provider:*

[2] My research is supported by the *Fundação para a Ciência e Tecnologia*, vd. Acknowledgements.

*provide quality (or else die).* But seemingly the word is not getting through. Users are not demanding. Perhaps they simply do not know the Web could be much better. Perhaps they simply do not want to: one way for a site to be better is to be simpler; perhaps most users prefer complicated, slow sites. This social aspect is not addressed further in the current paper.

## 2. Quality Factors Overview

> HTTP, CGI are the GOTOs of the 1990s. [2]

We present a method for the development of complex web services. The method was tested with a service case featuring:

- user authentication
- multiple forms
- recorded data
- automated page creation

The method emphasises quality at two stages: development and execution (meaning runtime execution of the service). It does this by scoring high on quality factors of process and product respectively; mostly of product, but high scores here are justified by process factors implicit in the method, as illustrated in Table 1.

| Factor | Justification |
|---|---|
| *Correctness* | High traceability: rich error messages. Operationalised completeness checks. |
| *Reliability* | All errors handled. Standard technology. Simple page design. |
| *Maintainability* | Good choice of programming language (Ada). Separation of HTML code and service logic. |

Table 1. Product quality high scores justified

The product also scores high on *efficiency*, *usability*, *portability*, and *interoperability*. It scores less on *testability* (test data must be prepared for each case, and it is not operationalised), and *integrity* (no access control tool). These scores and their justification are further supported by the items detailed in the rest of the paper.

The method comprises selected and created "open source" software tools and components: package **CGI** by David Wheeler (modified version included in [3]), package **XML_Parser** by the author [3], and GNAT by GNU, NYU and ACT (vd. **adapower.com**).

We use the word *safety* as a synonym of *reliability*, and we use the words *method* and *safety* in a wide sense, viz. with *method* ranging from architecture to coding, and *safety* including effectiveness and efficiency both in development (*cost safety*) and execution.

In this paper the method is presented with examples from the real development case, and with incursions into the detail of selected aspects.

The method is continually evolving, due to both external technological change and internal planned increments. Some of these planned increments are also exposed in this paper, as a means of obtaining feedback from the software engineering community. This trait in particular puts the method on the top level of the CMM (Capability Maturity Model, vd. **http://www.sei.cmu.edu**). Other well known software process references associated with the current method are *vanilla frameworks*, *extreme programming*, *futurist programming* (vd. Internet). The precise form of these associations is left implicit in the paper.

## 1. The Case

The most recent application of the method was in the implementation of an official inquiry to schools via Internet. This was in Portugal, in the year 2000. The purpose of the inquiry was to evaluate a recent reform in school administration. The inquirer, and my client, was *CEESCOLA*[3], a state-funded research centre in education—henceforth simply the *Centre*.

The inquirees were 350 secondary schools and school groups randomly chosen out of a nation-wide universe of 1472 such entities.

The service was required to be accessible only by the selected schools, so these were previously given, via surface mail, private access elements (identifier and password). A time window for answering the inquiry was fixed, and the system was required to make the answers available to the Centre as soon as they were submitted.

The inquiry itself took the form of a number of long and complex questionnaires: each questionnaire had hundreds of questions, and the answer to certain questions determines the existence of other questions or their domain of possible answers.

Note that this case is very similar to electronic commerce services in complexity and safety issues.

## 2. The Method

The top-level features of the method are:

- HTML
- CGI
- separation of HTML code (documents) and service logic (program)
- HTML extended internally
- documents prepared through XML transformations
- both the service logic and the transformations written in Ada
- session state maintained in the served pages
- a single meta-HTML unit
- a single service procedure

[3] Centro de Estudos da Escola = Center for School Studies, Faculty of Psychology and Education Sciences of the University of Lisbon.

The separation of HTML code and service logic is a crucial design premise. Our rationale for this converges for the most part with that described in [2]. In order to attain separation, the stored pages are written in a slightly extended HTML, call it meta-HTML, which is transformed by the service, upon each request, into the served pages in standard HTML.

Also, minimalized HTML was preferred as a basis for meta-HTML, because minimalized HTML is more readable by humans than its non-minimalized counterpart or XHTML—and the ultimate reviewers of the meta-document are human.

Now, HTML, minimalized HTML, XHTML, and the designed meta-HTML are all subsumed by a slightly relaxed XML, notably one not requiring pairing end tags. This may seem nonsensical to XML formalist eyes and sound of heresy to XML purist ears, but in practice such a "dirty" version of XML is very convenient. With a *robust* XML processor one can easily control that one dirty aspect of not requiring pairing end tags. Package **XML_Parser** has such a robustness feature. The gains include:

- a single processing component for all "dirty" XML instances (HTML, minimalized HTML, meta-HTML, XHTML)
- increased readability of the input units
- an easy path to proper XML representations (not taken, but the current trend from HTML towards XML in the Web was a concern)

So, in this paper, we take the liberty of calling simply XML to all that—and hence the pervasive use of the term and inclusion of XML tools in the method.

XML processing happens at two stages: data preparation and service execution.

**Data preparation.** The questionnaires are created by client staff using WYSIWYG editors like Microsoft FrontPage and Word. Then these items are transformed into the final, static meta-HTML items. The major part of this transformation is automated, by means of Ada procedures utilising package **XML_Parser**. The transformation consists of:

- rectify the messy HTML emitted by Microsoft tools
- rectify and insert control elements and attributes
- structure the items into identified groups

Because the necessary ad hoc transformation programs are small (c. 1k lines), and the compiler is fast and easily installable on any site, Ada can also be used here, instead of the usual unsafe scripting languages.

**Service execution.** The pages are not served directly: they have a number of markers that must be replaced by the definitive values. This is done at run-time by the main service procedure, again utilising **XML_Parser**. The rest of this section focuses on this.

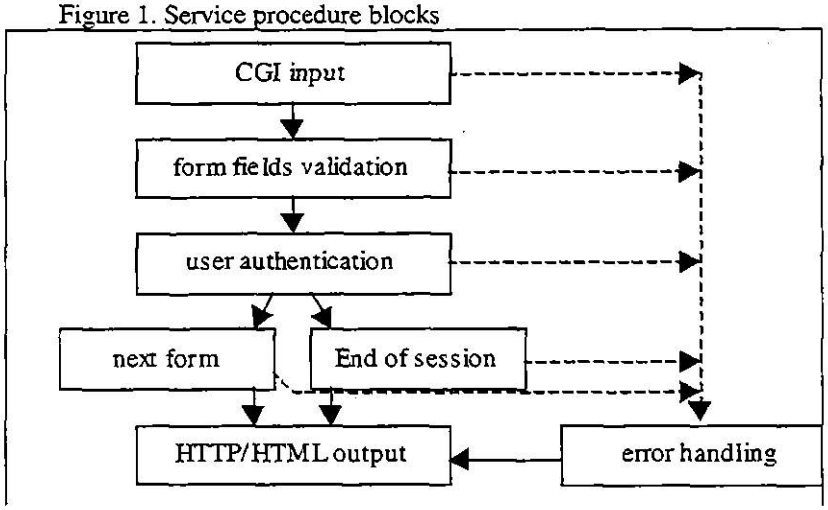
Input values from one form are relayed onto the next as hidden input elements. This provides for:

- data communication between session points, or forms—this implements sessions
- general tests on input values to be run on any session point—this increases safety

All input values are relayed, so careful naming of input elements is required (in order to avoid collision). The localisation of all forms in a single meta-unit promotes this.

The method evidently relies on the usual external pieces of web technology: an HTTP/CGI server and web browsers. The service was deployed with the Apache server running on a Linux system. Some problems were felt here, notably an access security hole: the service internal database files, in order to be accessible by the main procedure, had to be configured in such a way that they were also accessible by all local Linux system users! This problem is perhaps corrigible with the proper Apache settings; but this server's documentation is hardly comprehensible.

Figure 1. Service procedure blocks




### 4.1 The service procedure

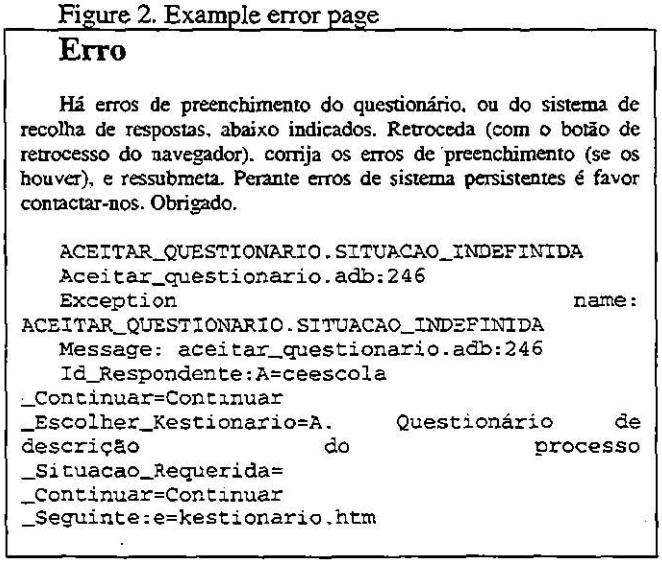
The service procedure is a non-reactive program, i.e. it terminates, a usual CGI procedures are. It is designed as the sequence of blocks sketched in Figure 1.

The computation is data-driven by form input values, meta-HTML markers, and system database files (users, passwords, etc.) The form input values and the files are totally case-dependent, so we focus on the meta-HTML markers, and dedicate the next section to them.

The exception handling is crucial. All errors are captured in a report page served to the user with a wealth of useful information, including instructions for error recovery, illustrated in Figure 2.[4] This happens

[4] The original data in Portuguese are shown in the figures because they have formal indentifiers in Portuguese (sometimes in English, e.g. when they emanate from the compiler), and we wanted to ensure referenctial consistency between all data items shown in this paper and at its presentations.

Figure 2. Example error page



**Erro**

Há erros de preenchimento do questionário, ou do sistema de recolha de respostas, abaixo indicados. Retroceda (com o botão de retrocesso do navegador), corrija os erros de preenchimento (se os houver), e ressubmeta. Perante erros de sistema persistentes é favor contactar-nos. Obrigado.

```
   ACEITAR_QUESTIONARIO.SITUACAO_INDEFINIDA
   Aceitar_questionario.adb:246
   Exception                              name:
ACEITAR_QUESTIONARIO.SITUACAO_INDEFINIDA
   Message: aceitar_questionario.adb:246
   Id_Respondente:A=ceescola
_Continuar=Continuar
_Escolher_Kestionario=A.    Questionário    de
descrição            do            processo
_Situacao_Requerida=
_Continuar=Continuar
_Seguinte:e=kestionario.htm
```

even during development and testing, facilitating these tasks greatly.

## 3. Meta-HTML

This section describes the meta-HTML used in the example case. Other HTML extensions are possible. In fact this possibility is a major plus of the method: it provides applicability to a wide range of possible web services, through case-by-case adaptation of the meta-documentary language. It can even go beyond XML eventually, but that is another story.

### 3.1 Input field types

The names of the form/input fields are extended with a type suffix of the form `:t`, where `t` is a single letter as described in Table 2.

| | description |
|---|---|
| | integer |
| | alphanumeric |
| | subject to special verification |

Table 2. Type suffixes

The upper case versions of `t` (`I`, `A`, `E`) additionally require a non-null value. The set is easily extended with more basic types, e.g. float and date. Type `e` (from the Portuguese word *especial*) requires a case-by-case treatment in the main procedure. The relevant section in the procedure is structured as a **case** construct: any `e` type value falling back to the **others** case raises a **System_Error** (or something similar). This together with the proper test data set increases safety in the development stage.

### 3.2 Conditional inclusion

Meta-element `if` provides conditional selection of parts of the meta-document to be included in the served page. The selected part is the enclosed content of this element. This is similar to the C preprocessor directive `#if`. The condition is expressed in the element attributes

$$Name_1 = |Value_{1.1}| \ldots |Value_{1.m_1}|$$

...

$$Name_n = |Value_{n.1}| \ldots |Value_{n.m_n}|$$

which contain references to form/input element names and values. The set of attributes is a conjunction of disjunctions. The (positive) Boolean value of the set determines inclusion of the element content. Figure 3 shows an excerpt of the example meta-document with heavy use of conditional inclusion, and Figure 4 shows the corresponding HTML result for a particular session.

Note the pervasive use of `E` suffixes in the meta-text: this was very helpful in assuring completeness of treatment of all cases—and therefore the correctness of the service.

### 3.3 Session control

A special hidden input element named `_Seguinte:e` (Portuguese for *next*) specifies the next meta-HTML unit to be processed. This in non-trivial at the start of the session, when moving from an authentication form to the main set.

Also, the absence of this element may be used to signal to the main procedure that the session is in its final step, usually submission of the combined data of all forms.

A small number (circa five) of other special elements were found necessary to control very specific aspects of the service. It was technically easy to implement them in the same vein, notably with the CGI and XML processing resources already available.

## 4. The tools and components

To see the next transactional "transfer" happen, ignore the XML (and SOAP) hype and *watch for actual* XML implementations.
(Mike Radow, in [3])

A modified version of package **CGI** by David Wheeler served well as the CGI component. The modifications, done by myself, included:

- Elimination of auxiliary overloading which caused ambiguity problems to the GNAT compiler. I suspect GNAT's complaints were legitimate, language-wise; perhaps Wheeler used another, non-validated, compiler; or the problem was not detected until my use of the package.
- Redesign of the output format of procedure **Put_Variables**.

The modified version is now in [3]. Further modifications are planned and described there.

Package **XML_Parser** by myself, also in [3], was used to transform the HTML emitted by the non-technical staff into extended HTML and then into the served HTML pages. Although **XML_Parser** served well as the (extended) HTML component of the current project case, it has severe limitations with respect to XML proper, noticeable in its documentation; it has also some design drawbacks, viz. the finite state device is entangled with the rest of the code.

To overcome these limitations, I have already developed a new XML processing package, **XML_Automaton**. This package properly encapsulates the finite state device. A new XML parser package, **XML_Parser_2**, will use **XML_Automaton** as its engine, in order to produce a more localised interpretation of the XML input. **XML_Parser_2** is designed after **XML_Parser** with respect to the (internal) treatment of XML element containment, and I am trying to make the expression of this containment generic, probably with an array of packages drawing on **XML_Parser_2**, each dedicated to a certain expression: an Ada linked list, Prolog facts, a DOM structure (Document Object Model, vd. **w3.org**), etc.

A rather specific but interesting point is the character-by-character vs. chunking way of processing XML input. XML elements may span over more than one text line. In chunk-based parsers, the chunk is normally the line. These parsers, especially if also based on character string pattern matching libraries, have a real problem here. **XML_Automaton** does not.

**XML_Parser_2** design includes an unbounded array of stacks. Currently I am choosing between two bases for the implementation of this structure: **GNAT.Table** or **Unbounded_Array**. I am inclined to the latter because it is compiler-independent.

Figure 3. Conditional inclusion example: meta-text

```
<h2>Situação da Escola/Agrupamento em 30 Abril
2000</h2>

<if orgao:E=~|CEI|CPN|not|~>
Estava em regime de transição.
<p>
Órgão responsável pela gestão da
Escola/Agrupamento:
</if>

<if orgao:E=~|CEI|~>
Comissão Executiva Instaladora.
</if>

<if orgao:E=~|CPN|~>
Comissão provisória nomeada pela DRE.
</if>

<if orgao:E=~|not|~>
Órgão de gestão anterior à publicação do 115-A/98
.
</if>

<if orgao:E=~|CEI|CPN|not|~>
<p>
Fase do processo de instalação em que se
encontrava:
</if>

<if orgao:E=~|CEI|CPN|not|~ fase=~|notAC|~>
ainda não tinha Assembleia Constituinte.
</if>

<if orgao:E=~|CEI|CPN|not|~ fase=~|notRI|~>
já tinha Assembleia Constituinte, mas não
Regulamento Interno.
</if>

<if orgao:E=~|CEI|CPN|not|~ fase=~|notAE|~>
já tinha Regulamento Interno, mas não Assembleia
de Escola.
</if>

<if orgao:E=~|CEI|CPN|not|~ fase=~|AE|~>
já tinha Assembleia de Escola.
</if>

<if orgao:E=~|DEeleita|~>
Tinha Direcção Executiva eleita.
</if>

<if orgao:E=~|DEnomeada|~>
Tinha Direcção Executiva nomeada.
</if>

<p>
(Se estes dados estão incorrectos volte atrás e
corrija.)
```

Figure 4. Conditional inclusion example: after processing with `orgao:E=CEI` and `fase=notAC`

```
<h2>Situação da Escola/Agrupamento em 30 Abril 2000</h2>

Estava em regime de transição.

<p>Órgão responsável pela gestão da Escola/Agrupamento:Comissão Executiva Instaladora.</p>

<p>Fase do processo de instalação em que se encontrava:ainda não tinha Assembleia Constituinte.</p>

<p>(Se estes dados estão incorrectos volte atrás e corrija.)</p>
```

## 5. Evaluation and some remarks

The software metrics available for the example case are:

| | |
|---|---|
| Cost .....: | 0.5 |
| programmer/month | |
| Lines of code (Ada) | : 1.0 k lines |
| Meta-document size | : 1.5 k lines |

Note the cost. We are missing *precise* comparison data with other experiments, but our experience and intuition tells us that it is a very good number—given the degree of correctness attained in the final service; notably, no fatal defaults were found. I have worked also recently with a team developing a service similar to the example in intrinsic complexity but with much less form data, implemented with inter-calling PERL (www.perl.com/pub) scripts (essentially a *Great Ball of Mud*, vd. slashdot.org/articles/00/04/29/0926241.shtml)—it required much more work and delivered much less correctness. The service is still plagued with detected bugs that no one rectifies anymore.

Why not use PHP (www.php.net)? Our reasons include:

- our method offers more control over the design and processing of the meta-language
- PHP documentation is incomprehensible
- Why not use Mawl [2]?
- it is not extensible
- it is not maintained
- it seems to be very hard to achieve a working installation

I am particularly fond of the inevitable conclusion that Ada is a good choice for programming *in the small*. So, there is a real small software engineering after all, and it is not confined to the unadjusted *Personal Software Process* [4] we read about—but never practice.


### Acknowledgements

I wish to thank my research advisor at *CENTRIA*[5], Doctor Gabriel Pereira Lopes. His correct envisionment of research in informatics as a rich network of diversified competencies and interests has made possible the degree of reusability seen here, notably of the XML tools which were firstly developed for our research projects in information retrieval and natural language processing.[6] I am also indebted to Professor João Barroso of *CEESCOLA* for providing such an interesting case of Internet usage as the one described here. Thanks to my colleagues Pablo Otero and Alexandre Agustini, and to the *QUATIC'2001* reviewers, for their good comments. Thanks to my family, for letting our home be also a software house. And to Our Lord, for everything.


### References


1] The Web Usage Paradox [webpage] : Why Do People Use Something This Bad? / Jakob Nielsen. — Alertbox for August 9, 1998. — (http://www.useit.com/alertbox/980809.html)

2] Mawl : A Domain-Specific Language for Form-Based Services / David L. Atkins ; Thomas Ball ; Glenn Bruns ; Kenneth Cox. — pp. 334-346 — //In: IEEE Transactions on Software Engineering, vol. 25, no. 3, May/June 1999

3] Ad*lib : the software process and programming library [web site] / by Mário Amado Alves. — (http://lexis.di.fct.unl.pt/ADaLIB)

4] Results of applying the personal software process / P. Ferguson ; W. S. Humphrey ; S. Khajenoori ; S. Macke ; A. Matvya. — pp. 24-32 — //In: IEEE Computer, 30(5), 1997 — (description apud [5])

5] Software Engineering : An Engineering Approach / James F. Peters ; Witold Pedrycz. — John Wiley & Sons, Inc. : New York, 2000. — xviii, 702 p.


[5] Centre for Artificial Intelligence, Universidade Nova de Lisboa.


[6] Projects *Corpora de Português Medieval*, *PGR*, *IGM*, and, in great part, my post-graduation scholarship PRAXIS XXI/*BM/20800/99*, granted by the *Fundação para a Ciência e Tecnologia* of Portugal.