Measuring the functional size of web applications

Silvia Abrahão and Oscar Pastor

Department of Information Systems and Computation, Valencia University of Technology, Camino de Vera s/n, 46071 Valencia, Spain
Fax: 34-96-387-7359
E-mail: sabrahao@dsic.upv.es
E-mail: opastor@dsic.upv.es

Abstract: Today, more and more Web sites and Applications (WebApps) are becoming mission-critical systems. Measures of functional size are a prerequisite to successful quantitative management of software projects. However, the nature of the web has recently imposed new and challenging characteristics, which are not supported by the existing estimation metric and models. In order to avoid the Web crisis, urge the use of web engineering approaches for developing and estimating web projects in a systematic way. To achieve this goal, we present a novel size metric for accurately measuring the functional size of web applications early in the development life cycle. The main contribution of this work is the introduction of a measurement process that is embedded in the conceptual modelling phase of a model-driven approach for developing web applications. The results obtained from our experiments with this approach are outlined in this work.

Keywords: web engineering; conceptual modelling; object-oriented; web metrics; function points; functional size measurement.

Reference to this paper should be made as follows: Abrahão, S. and Pastor, O. (2003) 'Measuring the functional size of web applications', *International Journal of Web Engineering Technology*, Vol. 1, No. 1, pp. 5–16.

Biographical Notes: Silvia Abrahão is an Assistant Professor in the Department of computation and information systems at the Valencia University of Technology in Spain. She received a BS in computer science from FPTE in Brazil. She is currently a PhD candidate in computer science at the Valencia University of Technology. Her research interests include web engineering, object-oriented conceptual modelling, and software quality.

Oscar Pastor is currently the Head of the Department of computation and information systems, Valencia University of Technology (VUT) in Spain, and the leader of the OO-Method Research Group in the same department. He received his PhD degree from the VUT in 1992, after a research stay in HP Labs, Bristol, UK. He is Professor of Software Engineering at the VUT. His research activities have involved conceptual modelling, requirements engineering, information systems, web-oriented software technology and model-based code generation. He has received numerous research grants from public institutions and private industry and has devoted considerable effort to the issues of technology transfer from academia to industry.

1 Introduction

The continuous advances imposed by technological innovations have caused the web sites, which are initially designed to present information to evolve into complex web applications. This evolution has originated an increasing in the complexity of designing, developing, and maintaining systems of this kind. However, the absence of disciplined processes in this context [1] has increased the interest in the successful development and maintenance of web applications.

A great number of initiatives for the creation of web solutions to the classic design of applications have recently been introduced [2–4]. Nevertheless, the systematic use of these techniques for the specification and the design of web applications have not solved the problem of software production and the applications continue to be delivered without the required level of quality. Because of the increase in the size, the complexity, the quality needs and the quick time-to-market cycles for WebApps, several problems have frequently been reported [5] such as: exceeding budgets, the delivery systems which do not have the required functionality and quality, and the general lack of documentation for requirements, etc. It is widely recognized that an important component of process improvement is the ability to measure the process. Therefore, one of the main problems in web development is the lack of appropriate metrics for project estimation and quality assurance.

To avoid a web crisis [6] we need rigorous web engineering approaches that provide high-quality applications on time and within budget. A good web engineering approach should help developers to address the complexities of web applications, as well as minimize risks of development, deal with the possibility of change, and deliver applications quickly, while providing feedback for management as the project goes along.

However, as mentioned in [7–9] web application development has significant differences from traditional application development. In traditional software development, the management of software cost, development effort and project planning are key aspects. Functional size measurement has been proposed as an appropriate tool for these management requirements. A number of size metrics are identified in the literature; the most widely used are Lines of Code (LOC) and Function Point Analysis (FPA) [10].

Currently, there has been an economic change from the trends in software development.

The fact is that in web development, the most important business variable is time to market. The development life-cycle time is shorter than 6 months, within average life-cycle time of 3 months. Therefore, the current estimation models must be adapted to address the new estimation challenges in order to control projects which are involved in web application production.

We present OOmFP$_{Web}$ as a novel size metric for accurately measuring the functional size of web applications early in the development process. Measuring functional size helps us in the estimation and evaluation of the software process (e.g., controlling application quality, cost and schedules) [11]. The main contribution of this work is the introduction of a measurement process that is embedded in the conceptual modeling phase of a software production method for the Web called OOWS – an Object-Oriented Web Solutions modeling approach [12].

In our approach, the functional size measurement of a web application is accomplished by mapping the Function Point Analysis (FPA) concepts proposed by the

IFPUG [13] to the OOWS conceptual model primitives. The main contribution of this work is that the measuring process is done on the conceptual schema: in consequence, the functional size of the web application is calculated in the problem space, improving the conventional approaches to the problem [8,14,15], that historically focus on the final software product.

2 Related work

Cleary has proposed the Web-Points metric [14] to estimate the size and effort provided by static web sites. The size of the web points is obtained taking into account the complexity of HTML pages of a web site. The complexity is classified into low, medium and high levels. The number of web-points for each page is obtained based on the size of the page in words, the number of existing links and the number of non-textual elements of the web page. This metric can be used with productivity data to determine the effort required for the development or improvement of a static web site.

The internet points [15] metric has been proposed as a solution to the problems found by the estimators in order to measure web applications in terms of function points. Seven functions are proposed which are based on the following: files, RDB tables, APIs, messages sent by the system, number of static HTML pages, number of dynamic HTML pages and the number of interactive pages. The interactive pages are those that collect data and include significant business logic and user interaction or error checking.

The GUI Metrics [15] was introduced in the 1990s. It is similar to the FPA and uses the graphical characteristics of the user interface to measure the project volume. This approach was designed specifically to consider the client side of the client–server development projects. GUI Metric measures the interaction with the environment and the users of the system. This metric is expressed in terms of dialogue boxes, menu options, tables, and reports.

Cowderoy [16] has proposed measures of size and complexity to improve quantitative methods and to control the costs of web projects. Following the GQM approach [17], this proposal identifies questions which are related to business objectives. The authors consider that the primary aspects that affect the web projects are: the identification of high-risk components, the costing of project changes and the identification of poor usability.

The measures proposed by Cowderoy not only take into account structural aspects of the application to be developed but also the possibility of changes that can be made in the establishment of the contracts. The measures of complexity and size are defined according to the type of resource used by the Web application (e.g., web-site structure, hypertext, database contents, images, audio, 3D objects, movies, etc).

Reifer [8] proposes the web objects size metric and an adaptation of the COCOMO II estimation model called *WebMo* for estimating the effort and duration of web applications. The Web Objects metric computes the size and take into account each one of the elements that make up a web application. These elements are the number of operands (the objects) and operators (actions that can done to the object). Examples of operands include: ActiveX components, commercial packages, 3D objects, text, video, clients and server tables, applets, images, etc. As in FPA, to make a repeatable prediction a set of counting rules are defined. A table of complexity measures with weights (low, medium and high) is also defined. The main advantages of this proposal are the use of a mathematical base (equations of Halstead) for prediction and the facility of extension through the addition of new operands and operators.

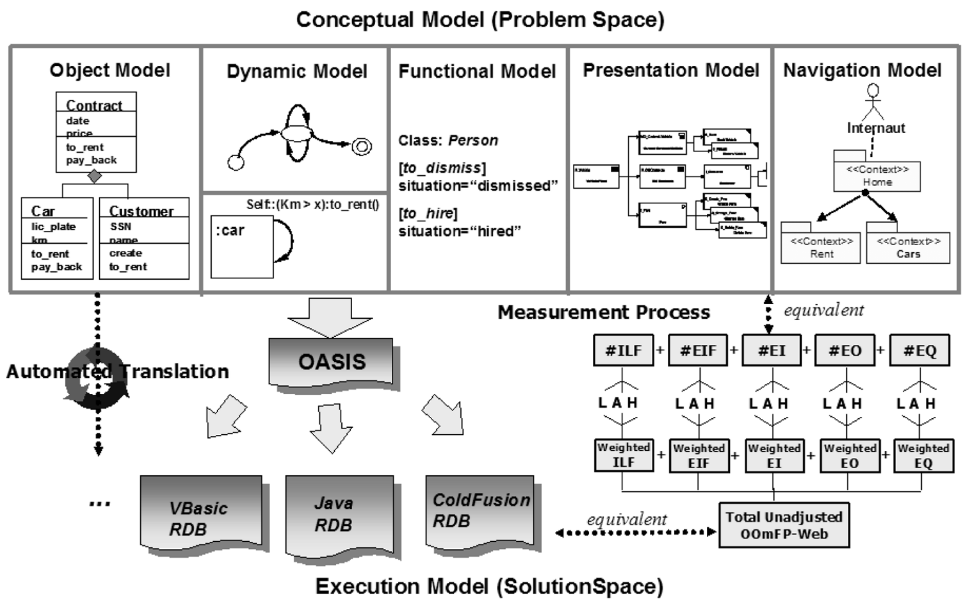
In their empirical study Mendes *et al.* [18] argue that complex web application structures can increase maintenance costs considerably. Even worse, for continually changing web applications (considering their evolving nature), the cost can be even higher. In order to diminish risks, the maintenance cost can be predicted through a functional size measurement.

This explanation on web metrics highlights certain deficiencies: the lack of desirable measurement properties, the majority is centred in static Web sites without taking into account behaviour and navigational properties presenting in a complex WebApp. Also, they have a high-level dependence on implementation technology. To overcome these deficiencies we present next the $\text{OOmFP}_{\text{Web}}$ approach that constitutes the contribution of this paper.

3 The $\text{OOmFP}_{\text{Web}}$ proposal

Because the web applications are built using a variety of web objects (html or xml files, scripts, multimedia files, etc.) no agreement on how to measure the size has yet been reached within the estimation community. In order to face this problem, we have developed a novel metric that measures the functional size of the web application independently of the technology used for the implementation. It is an extension of a measure for object-oriented systems from conceptual models called OO-method Function Point (OOmFP) [19].

Figure 1 A model-driven approach for developing web applications with an underlying measurement process.



In the context of the OO-method project [20], efforts have been oriented towards the development of a new model to enrich OO-method with the required expressiveness to specify navigation features. This model provides navigational support for object-oriented web-solution modelling (OOWS) [12]. In the $\text{OOmFP}_{\text{Web}}$ proposal, the functional size measurement of a web application is accomplished by mapping Function Point concepts defined by the IFPUG [13] to the OOWS conceptual model primitives. As showed in Figure 1, this mapping is conducted in four steps:

- the conceptual models are analysed to determine the application boundary and the measuring scope
- the conceptual models are analysed to identify the data and transactional candidate function types
- the complexity of each identified function is determined. The logical functions are mapped for low, average and high complexity levels in accordance with IFPUG tables
- the complexity levels are mapped into values. The total value results in unadjusted OOmFP for the web ($\text{OOmFP}_{\text{Web}}$)

Currently, we are using $\text{OOmFP}_{\text{Web}}$ as an unadjusted measure value. As demonstrated in several empirical studies [21,22], the factors proposed for system adjustment do not improve the size model or amount of effort, suggesting that a simplified sizing metric may be appropriate. Now, we introduce the main steps of the $\text{OOmFP}_{\text{Web}}$ proposal.

3.1 Applying conceptual models to web application design

When facing the conceptual modelling of a given web application, we have to determine the components of the object society without being worried about any implementation considerations. Moreover, it is important to separate the information design from the behavioural aspects of the application and from the navigational and interface concerns. Clear separation of concerns is widely regarded as a key factor for obtaining design quality, as well as ease of maintenance.

According to the OOWS approach (highlighted in Figure 1), we can distinguish two components: the conceptual model (problem space) and the execution model (solution space). At the problem space level, a precise system definition is obtained through the specification of conceptual models that capture the relevant information of a web application. The following conceptual models are specified: the object model for structural properties, the dynamic and functional models for behavioural and functional aspects, the presentation model for user interaction characteristics, and finally, the navigation model for navigation properties. They describe the object society from five complementary points of view within a well-defined OO framework.

- *Object model* is a graphic model where system classes are declared, including their attributes and methods. Aggregation and inheritance hierarchies are also graphically depicted representing class relationships. Agent relationships are introduced to specify who can activate each class method. Additionally, legacy views are defined as a filter placed on a class by a pre-existing software system.

- *Dynamic model* specifies valid object lives and interobjectual interaction. To describe valid object lives, we describe one *State Transition Diagram* (STD) for each class. To deal with object interaction, we use an *Object Interaction Diagram* (OID) for the entire System.
- *Functional model* captures the semantics associated to the changes of state of the objects, which are motivated by the method occurrences. This model specifies the effect of an event on its relevant attributes through an interactive dialogue. The value of every attribute is modified depending on the action activated, the event arguments involved, and the current object state.
- *Presentation model* captures the semantics of the user interaction using the following conceptual interface patterns: introduction, selection (defined or population), dependency, action selection and information presentation. The three first patterns are related to the data input in the system by the end-user, whereas the other ones are related to the manipulation of the environment and to the way the information is presented. Due to current space limitations, detailed information about these patterns can be found in [23].
- *Navigation model* is a navigational view over the object model. The navigational semantics of a web application is captured based on the point of view of each agent (audience), which is identified in the object model. This model is essentially composed of a *navigational map* that represents the global view of the application for an audience. It is represented by a directed graph where nodes are *navigational contexts* and arcs are *navigational links*. A navigational context represents the perspective a user has on a subset of the object model, while a navigational link allows for the navigation from one context to another.

A navigational context is composed by a set of classes stereotyped with the reserved word «view», connected by navigational relationships. These classes are called *navigational classes* and they represent views over specific classes in the object model. These views make a restriction to a given subset of attributes and services of the involved classes. For each navigational context there is a main class that is called *manager class* from where navigation starts. The others classes are called *complementary classes* and they contribute to giving additional information to instances of the manager class. These classes are connected using navigational relationships, providing related information between them. There exist two types of navigational relationships: context relationship and contextual dependency relationship.

A *context relationship* is a navigational relationship that also defines a directed navigation from this navigational context towards the target one, where the target class of the context relationship acts as the manager class. Four kinds of attributes can be specified to this type of relationship: (a) context attributes, which specifies the target navigational context of a navigational link, (b) link attributes, which specifies which attribute of the target class is involved in the connection, (c) role attributes, which indicates the referenced relationship-role when two classes have more than one relationship, and (d) filter attributes, which introduces selection criteria on the population of the target class, based on the value of the involved attribute. In this way, the search for specific objects is made easier.

In a contextual dependency relationship, the navigational semantic towards the target class is not defined: this kind of relationship is used to provide the required additional information in the current context without denoting any further navigation. Thus, it is not necessary to specify any kind of related information.

Finally, the following information can be specified during the design phase: population filters and order. A population filter establishes a selection of objects over the manager class. It is represented by a logical expression evaluated on the state of these classes. The *order* primitive indicates a traversal order of the context elements. Detailed information about the primitives of the navigation model can be found in [12].

From these analysis models, a corresponding formal and OO *OASIS* specification (the OO-method design tool) can be obtained in an automated way. This is done through an automatic translation process. The resultant *OASIS* specification acts as a complete system repository, where all the relevant properties of the component classes are included.

Once we have an appropriate system description, a well-defined execution model (in the *Solution Space* level) will fix the characteristics of the final software product, in terms of user interface, access control, method activation, etc. According to the execution model a prototype that is functionally equivalent to the specification is built in an automated way.

3.2 Determining the application boundary and the measuring scope

The *application boundary* indicates the border between the project being measured, and the external applications or user domain. It defines what is external and what is internal to the application according to the user's point of view. The *measuring scope* limits which functionality will be measured in a particular measurement (a subset). A web application can be understood as a group of logically related functions that fulfill specific business requirements. Its scopes varies widely: from small web services to large enterprise applications which are distributed across the internet.

We determine the application boundary from the user view. In OOWS approach, we build a navigational map for each audience. A navigational map represents the global view of the application for a specific audience. The proposed mapping rule is to accept each agent as a user of the WebApp. A measuring scope can include a complete WebApp (taking into account all navigational maps and agents), a navigational map for a specific agent (to specify what is perceived by an audience of the WebApp.), or a navigational context. The first one is applied in a new development whereas the latter in the restructuring phase of a WebApp (e.g. adaptive or corrective maintenance).

3.3 Determining data and transactional function types

As in FPA, we take into consideration data (OOmFP_D) and transactional (OOmFP_T) functions. The meaning and concept of each function type is analogous to FPA. Thus, given a conceptual model produced at the OOWS conceptual modelling step, OOmFP_{Web} is calculated as follows:

$$\text{OOmFP}_{\text{Web}} = \text{OOmFP}_{\text{D}} + \text{OOmFP}_{\text{T}} \tag{1}$$

In adapting function points to OOmFP_{Web}, we need to map function point concepts to OO concepts. In the OO paradigm, an object is a collection of data (attributes and properties) and functional logic (methods). The data defines the state of the object and the functional logic defines the behaviour of the object. In accordance with FPA concepts, data functions represent the functionality provided by users to satisfy their internal and external data needs (ILFs and EIFs). The data is a logical group, which is maintained or modified through transactional functions (EIs, EOs and EQs). Based on these considerations, we have identified the following functions:

- *Internal Logical File* (ILF): Select every *class* of an object model as an ILF. In function point terms, the parts of an object correspond to the logical structure of the file concept. Optional and mandatory subgroups of files are called Record Element Types (RETs). An object that is aggregated into (part of) another object constitutes a subgroup.
- *External Interface File* (EIF): Select every *legacy view* of an object model as an EIF. A legacy view is defined as a filter placed on a class by a pre-existing software system. In general, the base class from which this legacy view is derived is outside the problem being sized.
- *External Input* (EI): Select each *method* of a class as a candidate for an EI. A method is a unit of functional logic contained within an object. The primary intent of an EI is to maintain one or more ILFs and/or to alter the behaviour of the application.
- *External Inquiry* (EQ): Select each *population pattern* of a presentation model and each *navigational context* of a navigational model as a candidate for an EQ. A population pattern is used for visualizing the object society of a class. This pattern models unit of interaction focused on showing sets of instances of a class (an object collection). A navigational context also can be defined as a user interaction unit. It is composed by a set of navigational classes connected by relationships. These classes are views over existing classes in the object model (ILFs).

3.4 Sizing data and transactional function types

As in FPA, the key to developing repeatable predictor counts is a well-defined set of counting rules. Thus, a functional complexity is assigned to each class and legacy view according to the number of Data Elements Types (DETs) and the number of record element types (RETs). The formula for sizing data functions is the following:

$$\mathrm{OOmFP_D} = \sum_{i=1}^{n} \mathrm{OOmFP}_{\mathrm{ILF}_i}\left(\mathrm{DETs_{class}}, \mathrm{RETs_{class}}\right) + \sum_{j=1}^{m} \mathrm{OOmFP}_{\mathrm{EIF}_j}\left(\mathrm{DETs_{lView}}, \mathrm{RETs_{lView}}\right) \qquad (2)$$

In traditional function point, one RET is associated to each ILF or EIF because it represents a ‘user recognizable group of logically related data’. A DET represents a ‘unique user recognizable, non-recursive field of the ILF or EIF’. Thus, we have proposed the following counting rules for DET and RET of a class:

- DET: for each simple attribute (such as integer and strings) of a class, for each attribute in the Identification Function (IF) of an association relationship, and for each attribute in the IF of the superclasses it descends from.

- RET: for each class of the object model, and for each multivalued association relationship.

In the same way, the proposed counting rules for DET and RET identification of a legacy view are:

- DET: for each simple attribute of a legacy view, and for each attribute in the IF of a univalued association with a class.
- RET: for each legacy view of the object model, and for each multivalued association relationship with a class.

Then, a functional complexity is assigned for each transactional function (EIs and EQs) based on the number of data element types (DETs) and the number of File Types Referenced (FTRs). The formula for sizing transactional functions is the following:

$$\mathrm{OOmFP_T} = \sum_{i=1}^{n} \mathrm{OOmFP}_{\mathrm{EI}_i}\left(\mathrm{DETs_{method}}, \mathrm{FTRs_{method}}\right)$$

$$+ \sum_{j=1}^{m} \mathrm{OOmFP}_{\mathrm{EQ}_j}\left(\mathrm{DETs_{pPattern}}, \mathrm{FTRs_{pPattern}}\right) \qquad (3a)$$

$$+ \sum_{x=1}^{w}\sum_{y=1}^{z} \mathrm{OOmFP}_{\mathrm{EQ}_{xy}}\left(\mathrm{DETs_{nContext}}, \mathrm{FTRs_{nContext}}\right) \qquad (3b)$$

A transactional function represents a user's interaction with the software system. In function point terms, a FTR is a file type (ILF or EIF) referenced by a transaction. In $\mathrm{OOmFP_{Web}}$, each method of a class is considered an external input (EI). Nevertheless, a method is counted once (in the class which they are declared), even it is a shared event or inherited by several subclasses. Shared events affect the state of one or more objects. They are specified in all the classes that share them, but are sized only once. We have proposed the following counting rules for DET and FTR identification of a method:

- DET: for each simple argument of a method, and for each attribute in the IF of an object-valued argument (is a complex argument like an object).
- FTR: for the class in which is declared, for each object-valued argument of the method. We have also proposed a set of optional counting rules that take into account the dynamics for a method specified in the dynamic and functional models. More information about these rules can be obtained in [19].

In this previous work, we also have proposed counting rules for counting the DETs and FTRs for each pattern of a presentation model. In fact, the functions described at the moment provide the functional size of an OO system. We consider navigation as a critical feature in the conceptual modelling of WebApps.

Then, in order to measure the functional size of a web application we have incorporated the counting rules related with the navigational model (see the part (b) of the formula (3)), where, x corresponds to each navigational context of a navigational map, and y corresponds to each navigational map of a navigational model. The proposed counting rules for DET and FTR of a navigational context are:

- DET: for each attribute of all navigational classes that appears in the context; for each attribute of a context relationship; for each context relationship or contextual dependency relationship that has the source the class that is being sized; for each method that can be performed; and finally, for the traversal ordering of the elements of the context.
- FTR: for each navigational class that appears in the context; for each navigational class referenced in the definition of a filter attribute (in a context relationship); for each class referenced in the population filter formula.

Concluding, the $\text{OOmFP}_{\text{Web}}$ are arrived at by considering four basic functions (ILFs, EIFs, EIs, and EQs), each at one of three levels: low, average or high. We may express this as follows:

$$\text{OOmFP}_{\text{Web}} = \sum_{i=1}^{4}\sum_{j=1}^{3} w_{ij}c_{ij} \tag{4}$$

where c_{ij} is the count for component i at level j (e.g., inputs at low complexity) and w_{ij} is the fixed weight assigned by IFPUG tables [13]. Using the counting rules described above, we are able to predict the functional size of a WebApp. One advantage of our approach is that the measurement process takes into account all the characteristics (static and dynamic) of complex web applications captured by the different primitives of the conceptual models.

4 Conclusions and future work

We have introduced a measure for estimating the functional size of web applications from a model-based software production environment. We have presented the counting rules that support this measuring process. The contribution of our work is to provide a measuring process that can be done on the conceptual schema: in consequence, the size of the WebApp is calculated in the problem space, improving the conventional approaches, that historically focus on the final software product.

Despite their use by researchers and their growing acceptance in practice, the traditional FPA measures are not without drawbacks [21,22,24]. However, the use of $\text{OOmFP}_{\text{Web}}$ in a model-based code generation environment can solve some limitations of the traditional measuring process, such as:

- Changing requirements. As reported by Kemerer [24], if we compare a function-point count generated from an initial specification with the count obtained from the resulting system, we can find an increase of 400–2000%.
- Difficulty to automation. The original approach does not easily automate data collection.

In our approach, a specification described in the problem space has a concrete correspondence in the final software product. Thus, a size measurement made in the conceptual level is equivalent to a measurement made in the final web application. This characteristic corresponds to a technological support that bridges the gap between high-level design and low-level implementation. This is possible due to the model-based code generation features of the OOWS approach, which generates a final web application from an object-oriented conceptual schema in a semi-automated way.

Currently, we are applying this measure to several real-world applications. The experience gained has allowed us to put this measure into practice and to refine the proposed counting rules. A prototype that automates the counting process has been implemented and applied in an industrial environment [19]. Moreover, we are using this metric to predict the size of changes of evolving WebApps in the context of a methodology for quality evaluation called WebFP_QEM [25]. It can be a useful approach for assessing the functional size of the functions that are added or deleted after a restructuring of a WebApp. Consequently, the maintenance cost of the web application can be estimated.

There are many issues we need to address in the future: to design experiments in order to empirically validate the presented metric taking into account the observations pointed out in [21,22]; to calibrate the complexity tables proposed by IFPUG; and to define a cost estimation model [26] that integrates the $\text{OOmFP}_{\text{Web}}$ metric in a disciplined and repeatable manner.

References

1 Ginige, A. and Murugesan, S. (2001) 'Web engineering: an introduction', *IEEE Multimedia*, Vol. 8, No. 1, pp. 14–18.

2 Ceri, S., Fraternali, P. and Bongio, A. (2000) 'Web modeling language (WebML): a modeling language for designing Web sites', *Proc. 9th International WWW Conference*, Amsterdam, Holland, pp. 137–157.

3 De Troyer, O. and Leune, C. (1997) 'WSDM: A user-centered design method for web sites', *Proc. of the 7th International World Wide Web Conference*, Brisbane, Australia.

4 Schwabe, D. and Rossi, G. (1995) 'The object-oriented hypermedia design model', *Communications of the ACM*, Vol. 38, No. 8, pp. 45–46.

5 Cutter Consortium (2000) 'Poor project management – problem of e-projects', October, *http://www.cutter.com/consortium/press/001019.html*.

6 Zelnick, N. (1998) 'Nifty technology and nonconformance: the web in crisis', *Computer*, October, pp. 115, 116 and 119.

7 Offutt, J. (2002) 'Quality attributes of web software applications', *IEEE Software*, Vol. 19, No. 2, pp. 25–32.

8 Reifer, D. (2000) 'Web development: estimating quick-to-market software', *IEEE Software*, Vol. 17, No. 6, pp. 57–64.

9 Symons, C. (2000) 'E-commerce changes the economics of software production', *Newsletter of Software Measurement Services*, No. 6, Spring.

10 Albrecht, A.J. (1979) 'Measuring application development productivity', *IBM Application Development Symposium*, pp. 83–92.

11 Low, G.C. and Jeffery, D.R. (1990) 'Function points in the estimation and evaluation of the software process', *IEEE Transaction on Software Engineering*, Vol. 16, No. 1, pp. 64–71.

12 Pastor, O., Abrahão, S.M. and Fons, J.J. (2001) 'Object-oriented approach to automate web applications development', *Proc. 2nd Int. Conference on Electronic Commerce and Web Technologies* (*EC-Web*'01), Springer Verlag, Germany, pp. 16–28.

13 IFPUG (1999) *Function Point Counting Practices Manual, Release 4.1*, International Function Points Users Group (IFPUG), Mequon, Wisconsin, USA.

14 Cleary, D. (2000) 'Web-based development and functional size measurement', *Proc. IFPUG Annual Conference*, San Diego, USA.

15 Cost Xpert Group, Inc., (2002) 'Estimating Internet development', *http://www.costxpert.com/Reviews_Articles/SoftDev/*.

16 Cowderoy, A. (2000) 'Measures of size and complexity for web-site content', *Proc. of European Software Control and Metrics* (*ESCOM*'2000), Munich, Germany.

17 Basili, V.R. and Rombach, H.D. (1988) 'The TAME project: towards improvement-oriented software environments', *IEEE Transactions on Software Engineering*, Vol. 14, No. 6, pp. 758–773.

18 Mendes, E., Mosley, N. and Counsell, S. (2001) 'Web metrics – estimating design and authoring effort', *IEEE Multimedia*, Vol. 8, No. 1, pp. 50–57.

19 Pastor, O., Abrahão, S.M. *et al*. (2001) 'A FPA-like measure for object-oriented systems from conceptual models', *Proc. 11th International Workshop on Software Measurement* (*IWSM*'01), Montréal, Canada, Shaker Verlag, pp. 51–69.

20 Pastor, O., Gómez, J., Insfrán, E. and Pelechano, V. (2001) 'The OO-method approach for information systems modelling: from object-oriented conceptual modeling to automated programming', *Information Systems*, Vol. 26, No. 7, pp. 507–534.

21 Abran, A. and Robillard, P.N. (1996) 'Function points analysis: an empirical study of its measurement processes', *IEEE Transactions on Software Engineering*, Vol. 22, No. 12, pp. 895–910.

22 Jeffery, R. and Stathis, J. (1996) 'Function point sizing: structure, validity, and applicability', *Empirical Software Engineering*, Vol. 1, No. 1, pp. 11–29.

23 Molina, P., Melia, S. and Pastor, O. (2002) 'Just-UI: a user interface specification model', *Proc. of 4th International Conference on Computer-Aided Design of User Interfaces*, France.

24 Kemerer, C.F. (1993) 'Reliability of function point measurement: a field experiment', *Communications of the ACM*, Vol. 36, No. 2, pp. 85–97.

25 Abrahão, S., Olsina, L. and Pastor, O. (2002) 'A methodology for evaluating quality and functional size of operative WebApps', *Proc. of* 2*nd International Workshop on Web Oriented Software Technology*, ECOOP Workshops, Malaga, Spain, pp. 1–20.

26 Kitchenham, B. (1993) 'Using function points for software cost estimation – some empirical results', *Proc. of the 10th Annual Conference of Software Metrics and Quality Assurance in Industry*.