JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2003**

Vol. 2, No. 2, March-April 2003

Architectural Quality in Development Processes: A Case Study

Anna Grimán and Maria Pérez, Universidad Simón Bolívar. Caracas, Venezuela

Abstract

Software quality is expressed through various attributes, many of them are architectural. So, an architecture-focused development process, with an integrated self-evaluation, must be selected.

The purpose of this research is to present the incorporation of *Architectural Tradeoff Analysis Method* in the *Rational Unified Process* which emphasizes the definition of the software architecture through its 4+1 architectural views.

The improved RUP was applied to a case study: a Knowledge Management System (KMS), and the most important quality attributes for it were selected during the elaboration phase (Maintainability, Reliability and Efficiency), candidates architectures were proposed and the most suitable architecture was gotten.

1 INTRODUCTION

A stable architecture for guiding the system throughout its future lifetime is obtained iteratively and includes the identification of requirements, design, implementation and testing. Software architecture encompasses the different ways of presenting the system through components from different perspectives or points of view. However, software architecture is influenced not just by structure and behavior, but also by use, functionality, performance, flexibility, reuse, understandbility, restrictions, economic and technological commitments and aesthetics [Jacobson et al. 2000]. Many of these are intuitive; quality attributes that any architecture must have.

According to Jacobson [Jacobson et al. 2000] an architecture is needed in order to: *Understand the system* – all those involved in developing the system must understand its structure; *Organize the development* – by breaking the system down into subsystems and defining the interfaces and their relationships, thought can be given to the tasks to be developed in the next stages of development; *Foster reuse* – if components with specific functionality are specified, they can be used together in order to: *Make the system evolve* – changes in requirements can be implemented fairly effortlessly.

Bass et al. [Bass et al. 1998] believe software architecture is important because it facilitates communication among stakeholders and helps in decision-making on design issues by defining restrictions involving implementation, identifying *quality attributes*, handling changes and using transferable and reusable models.

This paper briefly describes the case study (DID-KMS PROJECT), the improved development process used in case construction, and then the architectural views and the scenarios as proposed Architecture Tradeoff Analysis Method (ATAM) activities. The following section describes the instantiation of the quality model for the DID-KMS PROJECT. Continuing with ATAM, the sensitivity Points and Tradeoff analyses are conducted, and lastly the conclusions and recommendations are presented.

2 DID-KMS PROJECT

KMS can be used to code, store and distribute a company's Knowledge Base. It can be used as a knowledge repository when knowledge needs to be coded. It supports the company's social capital by establishing structural links between people, regardless of the barriers imposed by time or geography, thereby improving the capacity to combine and exchange intellectual capital [McLure 1998].

According to O'Brien [O´Brien 1999], many organizations are developing KMS to manage Organizational Learning and business know-how. These systems help knowledge workers to create, organize and share important business information as and when they need it. Included are processes, procedures, patents, reference works, formulas, best practices, forecasts and arrangements. Internet websites and intranets, groupware, data mining, knowledge bases, discussion forum and videoconferences are just some of the key information technologies for gathering and distributing this knowledge.

Turban et al. [Turban et al. 2001] hold that the new challenge of Knowledge Management requires that organizations begin to acquire systems that support it. The activities that must mainly be supported by these systems are the following: a) *Knowledge identification*: Determination of knowledge that is critical in decision-making; b) *Discovery and analysis of knowledge*: use of search instruments, databases and data mining. Knowledge must be found, analyzed and put in the right context; c) *Setting up Organizational Knowledge Databases*: Organizational memory and best practices must be stored in an indexed, properly maintained Knowledge Base; d) *Use and distribution of knowledge:* definition of a suitable audience and placement of technologies to enable Knowledge to be delivered when it is needed.

It was within this context that the initiative to develop a KMS to support knowledge management related to the USB[1]'s research projects arose [Domínguez 2001].

The purpose of the KMS developed in this research is to encourage the professors at the la USB to manage their research projects through a Web interface, thereby fostering collaborative work and facilitating information sharing. The benefits of DID-KMS

[1] Universidad Simón Bolívar, Caracas – Venezuela.

PROJECTS will include the ability to capitalize on the knowledge generated by the research projects and keep it where it is accessible by everyone to so that information on specific areas can be sought quickly and easily. The objectives to be met are: to encourage people to apply for Research Project funding, as well as to foster the development of tools to facilitate the handling of these projects once they are approved. A further intention is to foster clarity and precision in the formulation of these projects in order to reduce initial rejections and be able to attain successful projects. All this must be done while supporting the processes that are characteristic of a KMS: capturing, generating, sharing and distributing knowledge.

The DID-KMS PROJECT system enables stakeholders to capture, generate, share and distribute much of the knowledge handled at the USB, while enabling the university to capitalize on and store all this knowledge, giving it a competitive advantage over other organizations.

The development process undertaken for the construction of the system in which the architectural evaluation is incorporated, is described below. The RUP development process was used for this.

3 DEVELOPMENT PROCESS

The Rational Unified Process (RUP) [Kruchten 2000] was used to carry out the development process. RUP is a Software Engineering process, which provides a disciplined approach in order to assign tasks and responsibilities in a development organization, placing particular emphasis on architecture through its 4+1 views.

The main objective of RUP is to ensure high quality software production that recognizes the needs of the end user according to a schedule or foreseeable plan. Since RUP does not cover the architectural assessment, ATAM was included in the elaboration phase. The reason for evaluating the architecture is that this is the main determiner of quality attributes [Kazman et al. 2000b]. ATAM is a method for assessing software architecture considering multiple quality attributes [Kazman et al. 2000].

Fig. 1 illustrates each phase of this method where one can see that ATAM is a spiral model divided into four main phases, where each of them makes one or more contributions to the understanding of the system, reducing risks and modifying the design [Kazman et al. 2000].

Given below are the objectives attained at each stage without any level of detail except in the phases of interest to the research by reason of the architectural evaluation.

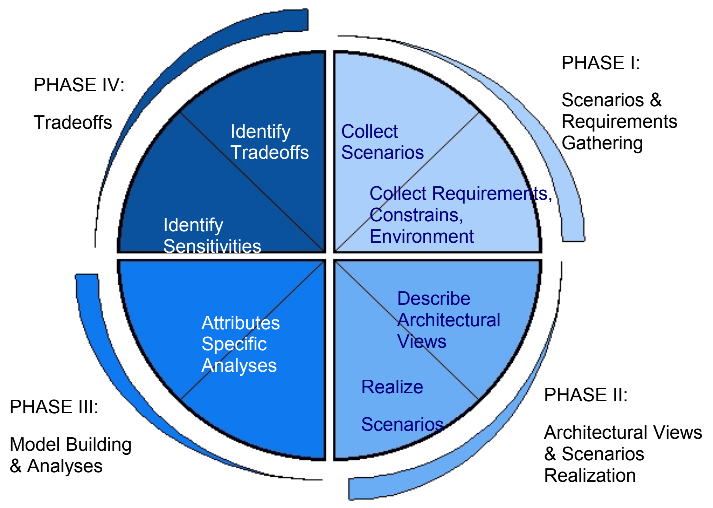

PHASE I:
Scenarios & Requirements Gathering
Collect Scenarios
Collect Requirements, Constrains, Environment
PHASE II:
Architectural Views & Scenarios Realization
Describe Architectural Views
Realize Scenarios
PHASE III:
Model Building & Analyses
Attributes Specific Analyses
PHASE IV:
Tradeoffs
Identify Tradeoffs
Identify Sensitivities

Fig. 1: The steps taken by the Architectural Software Tradeoff Method. ATAM

During the *Scenarios and Requirements Gathering* phase, two activities are included: *Collect scenarios* and *collect requirements, constrains and environment*. The objective of both activities is to elicit functional and non-functional requirements of the system vis-à-vis a diverse group of stakeholders. In order to do so, a brainstorming session was held before a representative group of stakeholders linked to the USB's system of research and development activities. As a result of this, fundamental use cases were identified.

Once these use cases were established, it was necessary to specify the architecture through its different views.

In the following section, the results of phase II of the ATAM are described.

4 ARCHITECTURAL VIEWS AND SCENARIOS REALIZATION

This phase also included two activities: Describe architectural views and realize scenarios. A view of the architecture is a simplified description of a system seen from a particular perspective or point of view, making available particular knowledge and omitting entities that are not relevant from its perspective [Bass et al. 1998].

Five views are normally specified in this activity but only the logic view will be described here since it accounts for the majority of the contribution for this research.

The Logic view mainly supports the functional requirements, in other words the services the system must provide to its end users [Kruchten 2000]. Hence this view includes: the Conceptual Model, the Class Diagram and the Entity-Relationship Diagram.

In order to be able to evaluate the architecture, two possible Class Diagrams (Candidate Architectures) have been proposed. In each diagram, some of the Gamma Patterns Design [Gamma et al. 1997] models were identified: Chain of Responsibility, Observer and Command. This will enable advantages and disadvantages to be obtained so they can later be compared with one another, the architecture best suited to the needs of the system can be proposed, and the one than meets the quality requirements desired can be chosen.

Fig. 2 shows the Candidate Architecture 1. Note that two patterns [Gamma et al. 1997] have been used in this architecture: Chain of Responsibility and Observer.

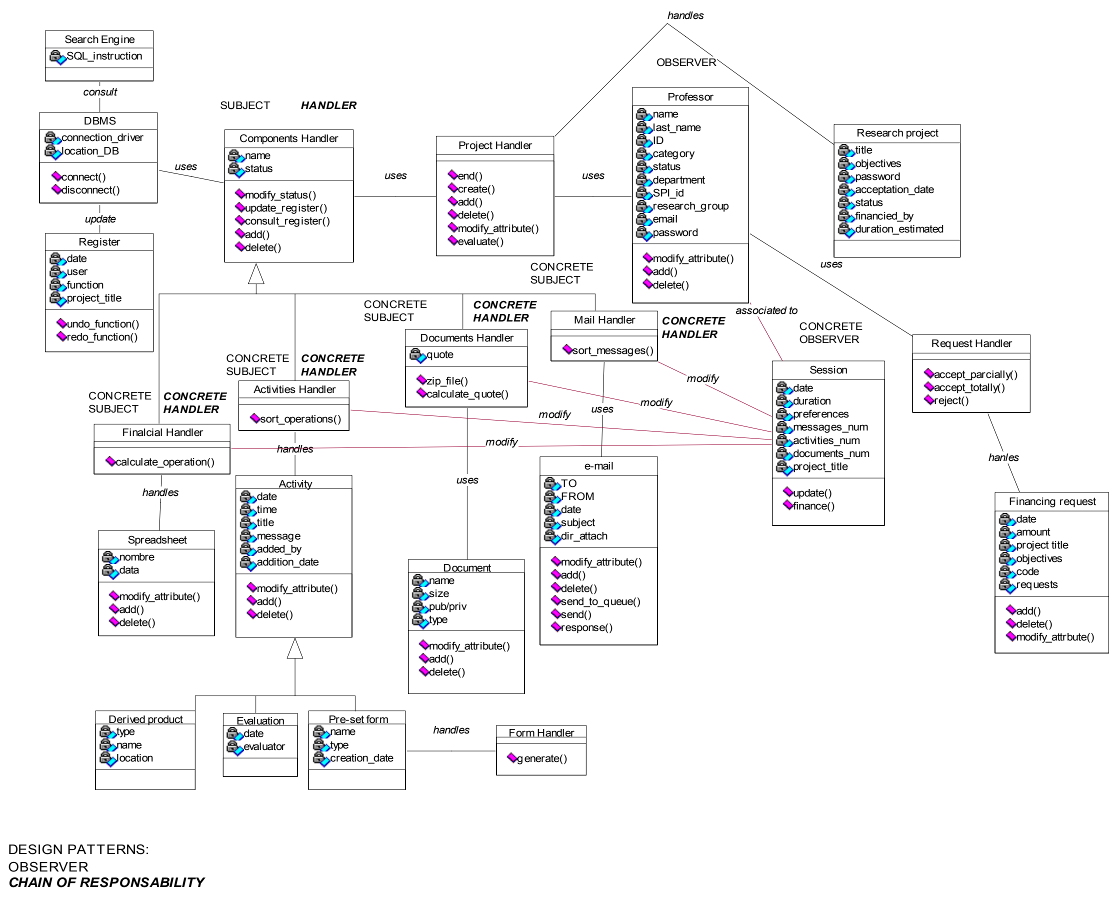


Fig. 2: Candidate Architecture 1

In the *Chain of Responsibility* Design Pattern, when a client issues a request, it propagates along the chain until it reaches a *ConcreteHandler* object, which takes responsibility for handling it. This leads to: reduced coupling, greater flexibility in the assignment of responsibilities and lack of guarantee in the receipt.

In the Candidate Architecture 1, the *Financial Handler, Activity Handler, Mail Handler* and *Document Handler* classes play a similar role to the *ConcreteHandler* object as these are levels in the *Components Handler* class that in turn would play the role of an *Handler* object.

The direct advantages of applying this pattern are: better distribution and control of requests and more scalability.

In the *Observer Design Pattern* all the observers are notified when a change occurs in the status of stored objects. This had the following consequences: it reduces the coupling between *Data* and *Observer Objects* and it supports broadcast communication and unexpected changes.

In the Candidate Architecture 1 the data that will be subject to frequent changes are in the classes of the four main components, which is why the *Financial Handler, Activity Handler, Mail Handler* and *Document Handler* classes play a similar role to a *ConcreteSubject* object, whereas the *Component Handler* will play the role of a *Subject* object. Moreover, notification of updates makes sense if several professors are working on the same project. This can be known thanks to the sessions. Therefore, the *Session* class would play a similar role to a *ConcreteObserver* object, whereas the *Professor* class would play the role of an *Observer* object.

The direct advantage of applying this pattern is: it facilitates group work since updating the data handled by the different components could be notified to all the "registered observers", in this case it would be professors working on the same project.

In Fig. 3, the Architecture Candidate 2 is presented. Note that the design patterns identified in this Architecture are *Chain of Responsibility* and *Command*. For the purpose of this paper, the *Chain of Responsibility* pattern is not described again.

In the *Command Design Pattern* [Gamma et al. 1997], when a client issues a request it propagates along the chain until it reaches a *ConcreteCommand* object that takes responsibility for handling it. This leads to the following consequences: reduced coupling, easy addition of new commands, support for recording changes, support for transactions and support for the *Undo* operation.

In the Candidate Architecture 2, requests would reach each handler that uses some of the functions defined in the *Functions Library*, but before executing them the *Record of Modifications* which in turn checks the feasibility of the function and invokes its execution, is updated. For all these reasons, the *Format Handler*, the *Component Handler*, the *Project Handler* and the *Request Handler* play a similar role to a *Receiver* object, while the *Function Library* class would play the role of a *ConcreteCommand* object and the *Record of Modifications,* would play in turn the role of *Invoker*.

The direct advantages of applying this pattern are: it permits better distribution and control of functions, enables audits to be carried out, increases the possibility of recovery and facilitates scalability considerably.

After this analysis, continuing with the ATAM method, in the next section the Utility Tree of quality for the DID-KMS PROJECT is built .

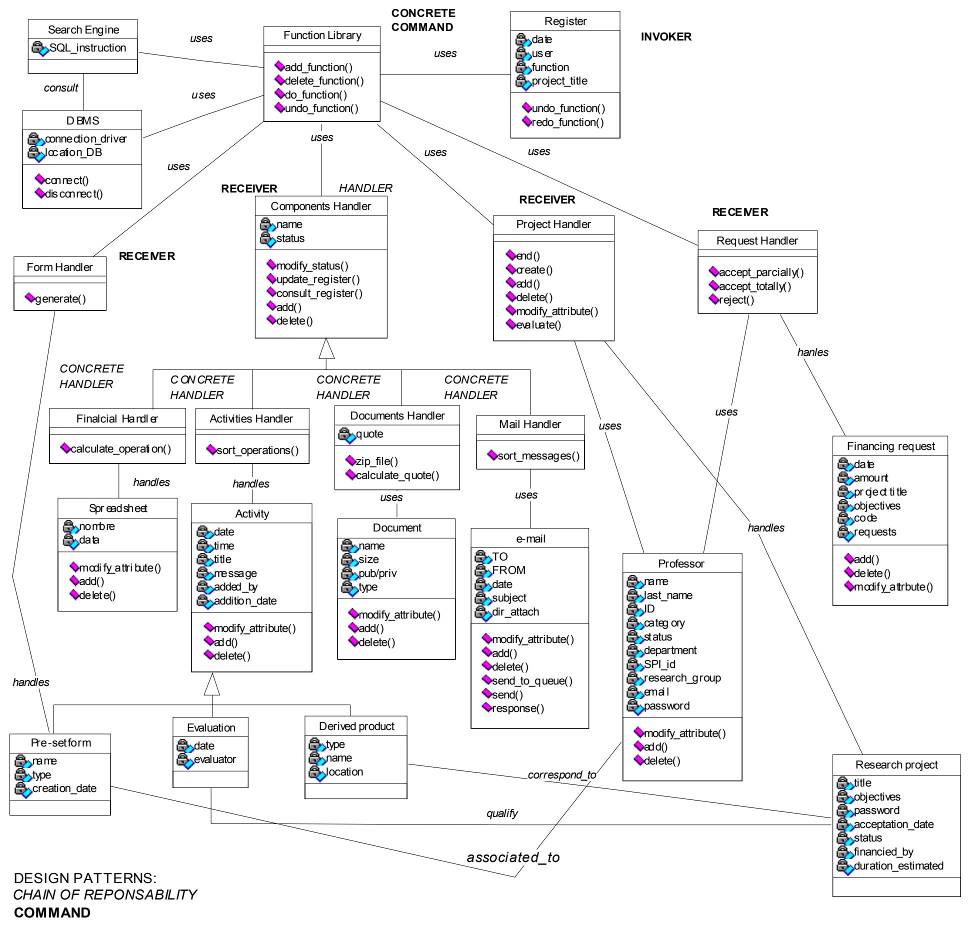


Fig. 3: Candidate Architecture 2

5 QUALITY MODEL BUILDING AND ANALYSES

At this stage a Utility Tree, where the most important quality characteristics and attributes for the DID-KMS PROJECT are identified, must be built. Since ATAM does not give any specific way of doing this, the Product Quality Model proposed by Ortega [Ortega et al. 2000] will be used as the basis. This is a **software product quality** model but the idea at this stage is to identify **quality** characteristics and attributes **for the architecture** of KMS. Therefore, architecture had to be taken as the product to be evaluated, and certain changes made to the model, specifically at the level of metrics. Fig. 4 shows the quality model proposed by Ortega [Ortega et al. 2000], which integrates the ideas of several quality models and presents them in a coherently related manner.

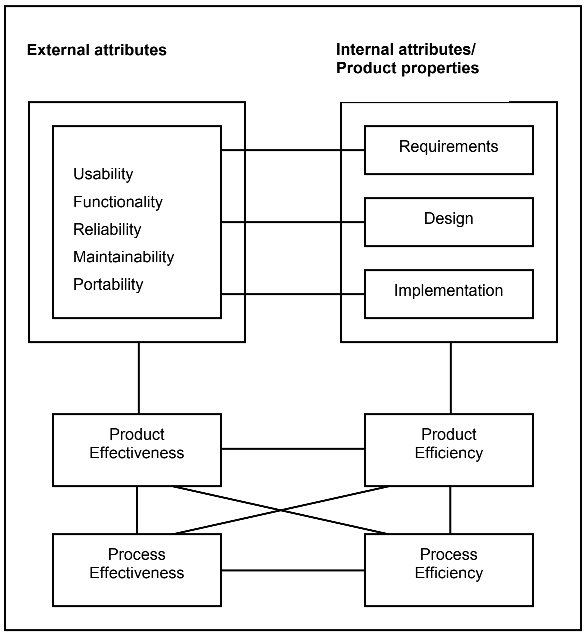


Fig. 4: Software Quality Model

Taking the Ortega et al. [Ortega et al. 2000] model as the basis, the quality characteristics and attributes expected for the DID-KMS PROJECT were identified.

Selection of architectural attributes for a KMS

Bass et al. [Bass et al. 1998] describe *Functionality as* "a system's capacity to do the work for which it was designed or proposed." The authors also affirm that *Functionality* is orthogonal to the structure and is therefore not an architectural characteristic by nature; i.e. any possible number of structures can be conceived in order to implement any functionality. The system could even exist as a monolithic component without any internal structure [Bass et al. 1998]. Therefore, *Functionality does not depend on software architecture, so* ***it must not be taken as one of its quality characteristics.***

Although focused on systems in general, Jacobson et al. [Jacobson et al. 2000] state the following (applied perfectly to a KMS): "If we can be sure about anything, it is that any sizeable system will evolve. It will even evolve if it is still under development." This is possible thanks to the architecture, which is why ***maintainability has to be considered one of the architectural quality characteristics to be taken into account.***

As far as system *Reliability* is concerned [Bass et al. 1998], it is defined as the system's ability to remain operational over time and, like Ortega [Ortega et al. 2000], they point out that *Reliability* is related to fault tolerance and the time it takes to recover, both aspects being attainable through the architecture. Thus for KMS's domain, ***Reliability must also be considered a quality characteristic or attribute to be propitiated by the architecture.***

Now the following doubt arises: what is the reason for the faults from which the system has to recover, putting Reliability at stake? There can be many kinds of fault (physical damage of a piece of hardware, electrical current failure, etc.), software failures (unexpected cases, invalid operations, etc.) or faults caused by external effects (request overload, unauthorized access, etc.). Although *Security*, according to Ortega [Ortega et al. 2000], is part of the *Functionality* and, as shown above, the latter is not architectural, Bass et al. [Bass et al. 1998] consider that *Security* is, as they say that prevention, detection and response to such effects involve architectural strategies that may require the existence of special components to solve it [Bass et al. 1998]. A KMS in particular will not be taken as just another characteristic, but as a sub-characteristic of *Reliability.*

As regards *Efficiency*, Bass et al. [Bass et al. 1998] relate it to the time required to respond to a particular stimulus (event) or the number of events processed in the same time interval. These authors also say *Efficiency* can be measured on the basis of the amount of information and communication between system components, which clearly is an architectural characteristic, since components can be implemented within the server layer that handles users' requests efficiently. Generally, because they capture, distribute, share and generate Knowledge, KMS have this type of information flow, therefore handling must take place as efficiently as possible. So, ***efficiency must also be taken into account when it comes to guaranteeing product quality through the architecture as far as a KMS's domain is concerned.***

Thus the most important architectural quality characteristics to be taken into account when developing the KMS are: **Maintainability, Reliability and Efficiency.**

It is necessary to show the attributes and their respective metrics resulting from, as a result of selecting the most important characteristics of the Quality Model proposed by Ortega et al.[Ortega et al. 2000] for the KMS. Table 1 only shows the attributes associated with the quality characteristics identified above, that could be fostered through the software architecture, as well as the metrics to be applied to evaluate the corresponding attributes. *Note that the metrics included here in those proposed by Ortega have been marked with (***), they are extremely important, not just for the development of a KMS, but for any system in general.*

MAINTAINABILITY	RELIABILITY	EFFICIENCY
✓ **Capacity for Analysis** -Easy to diagnose Record of diagnoses ✓ **Capacity for Change** -Parameterization Record of changes -Functional independence of the modules -Incorporation of new requirements -Monitoring of client's needs	✓ **Maturity** -Average time between failures - Which failures were solved? -Percentage of test cases from point of view of operation by the user -How many test cases pass the product successfully? -Operational risks	✓ **Behavior over time** -Response time - Satisfaction with the number of tasks completed in a specified time. ✓ **Use of resources** -Use of the CPU. -Use of Memory -Satisfaction with the processing speed considering other

-Improvements due to technological changes -Traceability between requirements, design and implementation. -*** Access and capacity to update programmer's manual ✓ **Stability** -Number of variables modified compared to all the variables in a module -Existence of collateral effects when making changes -Number of global variables compared with the modules that use them -Existence of an impact matrix. ✓ **Testability:** -Check points. -Detection of impact. ✓ **Coupling** -Level of coupling ✓ **Cohesion** -Level of cohesion. ✓ **Control system structure.** -Number of modules. -Average size of the modules. -Maximum depth of module nesting. -Average depth. ✓ **Descriptive** -Level of description -Specified. -Documented -Self-descriptive -*** Existence of the programmer's manual ✓ **Parameterized** -Use of parameters. -Unnecessary parameters -Poor parameter passing	-Adjustment of the operating environment according to the requirements -Definition of mechanism to ensure that software capacities meet the client's needs -Volatility of product requirements. -Level of satisfaction with audits to the software -Strategy to integrate additional elements -Acceptance criterion -Integration check -Test record -Maintenance strategy - Update of specifications, documents and strategies - Component tests - Update of system in the user's environment - Control in modifications to minimize upsets for clients. - Development techniques used. ✓ **Fault tolerance** -Canceling of incorrect operation. ✓ **Recovery** -Restart capacity -Reinitialization speed -Existence of processes that reduce software product hanging time. - Availability of the software product. - *** Data integrity checking ✓ **Security** -Control of access -Auditing capacity	products competing for this resource -Satisfaction with the memory considering other products that compete for this resource

Table 1: Attributes selected from Ortega's model

In the next section the Sensitivity Points and Tradeoff are identified.

6 TRADEOFFS

This phase covers two activities: *Identify Sensitivities and Identify Tradeoffs*. In the first of them, the sensitivity of the attributes analyzed was discovered; which means that if one of them changes, so does the architecture. The models must then vary in order to reflect those changes and assess the results. Any value that significantly affects the architecture must be considered a Sensitivity Point. In the second activity, the Sensitivity Points identified were examined in a little more detail. If a sensitivity point affects another attribute positively, but in turn has a negative influence on a different attribute, then there is a Tradeoffs.

Control System Structure

The *Control System Structure* (See Table 1) within the *Maintainability* characteristic, affects the architecture since it is related to the *Number of Modules* in the system. If the number of modules increases or diminishes, changes must be made to the models to reflect the transformation. Depending on the use frequency, this addition may affect the *Use of the CPU and of the Memory,* and even reduce *Response Time,* not to mention the increase in both *Coupling* and *Cohesion*.

In summary, a change in the Number of Modules affects the following metrics: *Record of Changes*, *Use of Parameters*, *Documented*, *Level of Cohesion*, *Use of CPU*, *Use of Memory*, *Response Time, Level of Coupling*.

Therefore the metric *Number of Modules* can be considered a Sensitivity Point for the *Control System Structure* attribute.

Capacity for Change

The metric *Incorporation of Requirements* is even more subjective and depends on the extent of the change in question. However, the *Capacity for Change* attribute within the *Maintainability* characteristic affects the architecture since it is related to the *Incorporation of Requirements.* The *Incorporation of Requirements* may not involve the creation of a new module or component, but it does involve the creation of new relationships and can affect *Efficiency.*

The *Incorporation of Requirements* affects the following metrics: *Data Integrity Check*, *Documented Amount* of variables modified compared with the Total number of variables in a module.

Hence the *Incorporation of Requirements* metric may be considered a Sensitivity point for the capacity for change attribute.

Recovery

Given the use frequency and order of magnitude of the *Recovery* attribute, the architecture is affected because it relates to the *Data Integrity Check.* Therefore, the *Data Integrity Check* metric can be taken as a sensitivity point for the *Recovery* attribute.

In short, these were the sensitivity points identified: *Number of Modules* (Maintainability), *Inclusion of Requirements* (Maintainability) and *Data Integrity Check* (Reliability)

Once the Sensitivity Points were identified, the second activity in this fourth phase of the ATAM was carried out: *Identify Tradeoffs*. This activity involved analyzing the model built in Phase III: Model Building & Analyses. Table 2 summarizes the analysis for the two candidate architectures.

SENSITIVITY POINTS	FOR CANDIDATE ARCHITECTURE 1	FOR CANDIDATE ARCHITECTURE 2
Number Of Modules	Since each module or handler added could play the role of *Concrete Subject* within the *Observer* pattern, a link must be added to the respective class *Observer* which, in this case, is the *Session* class, which means there is one more variable for the server for each professor connected, which increases *the Use of the CPU* and of the *Server Memory*, and hence reduces the *Response Time*, jeopardizing the *Use of Resources* and the *Stability*. At the design level, this implication cannot be measured, but once the system is implemented it can be done in execution time. Furthermore, it is mandatory for the *Change Record* and the *Programmer's Manual* to be updated.	The addition of new modules or components will only involve, as well as the necessary updating of the *Record of Changes* and the *Programmer's Manual*, the addition of the new functions in the *Functions Library,* which is not particularly complicated, fostering on the other hand the *Parameterized* nature of the *Maintainability*.
Inclusion of Requirements	Likewise if the change requires a variable in the class *Session* it jeopardizes the *Use of Resources* and the *Stability*. In the worst-case scenario, the inclusion of the new requirement might involve the creation of one or more modules, with the same consequences as in the previous case.	As explained above, the addition of new handlers involves not only the addition of their functions in the *Functions Library*.
Data Integrity Check	*Recovery* would not be affected, since the *Session* variable are eliminated when the user quits the page or after more than 20 minutes and the statuses are stored. Even these may be erroneous, but the information necessary for its recover is kept in the record of modifications.	On the other hand, *Recovery* is fostered since the *Record of Modifications,* prior to invoking a call to any object from the *Functions Library*, can check its feasibility and actually execute it, still keeping the old values, facilitating the *Data Integrity Check*. The information stored in the *Record of Modifications* is far more complete and reliable.

Table 2: Behavior of the two candidate architectures.

Based on the results of applying the ATAM method, the Candidate Architecture 1 cannot be considered any more advantageous than the Candidate Architecture 2. Therefore the latter is deemed to be the most appropriate. It was also noted that *Efficiency* lies in Tradeoffs with *Maintainability*, and that greater *Reliability* can reduce it considerably. So, with the right equipment and sufficient memory, these inconveniences can be secondary.

Having completed all the steps required by the ATAM method for Evaluating the Architecture, the level of detail was found to be sufficient to begin developing the system with a good degree of certainty as regards the ideal structure of the architecture and the quality characteristics and attributes expected.

7 CONCLUSIONS AND RECOMMENDATIONS

The development process proposed by RUP was used and the Software Quality Model proposed by Ortega [Ortega et al. 2000] was instantiated as part of this process, in particular in the specification of the architecture, Thus, the architectural quality characteristics that must be taken into account for the development of a KMS are: Maintainability, Reliability and Efficiency. Additionally, some of the metrics proposed by Ortega [Ortega et al. 2000] were redefined, and three more added, thereby adding to the research area covered by this paper.

The importance of evaluating software architecture, to enable certain advantages or disadvantages that might not be visible in the design, was highlighted. In order to evaluate the architecture, the ATAM method included in the development process was used. This implied identifying and analyzing several design patterns.

It is recommended that the improved development process be applied with the architectural evaluation in other domains in order to refine it.

8 ACKNOWLEDGEMENT

The authors wish to thank Kenyer Domínguez and Yemala Castillo for their help in preparing this paper. This research has been co-financed by the following projects: FONACIT S1-2000000437 and S1-2001000794, and USB-DI-CAI-011-01.

REFERENCES

[Bass98] L. Bass, P. Clements, and R. Kazman: *Software Architecture in Practice*. Addison Wesley. pp. 9, 17, 23, 28-36, 75-81, 1998.

[Gamma97] R Gamma, R. Helm, R. Johnson, J. Vlissides : *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison Wesley Professional Computing Series. pp. 2, 3, 8, 9, 223-242, 293-304, 1997.

[Jacob00] I. Jacobson, G. Booch, J. Rumbaugh: *El Proceso Unificado de Desarrollo de Software*. Addison Wesley. pp. 58, 64-65, 232, 306, 307, 345-365, 2000.

[Kazma00] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, J. Carriere: *The Architecture Tradeoff Analysis Method. Software Engineering Institute*. Carnegie Mellon University. Pittsburgh, USA. http://www.sei.cmu.edu/ata/ata_method.html

[Kruch00] Phillipe Kruchten: *The Rational Unified Process*. Addison Wesley Longman, Inc. pp. 17-18, 23, 67-73, 83-85, 122-123, 1999.

[McLure98] Molly McLure: "A Framework for Successful Knowledge Management Implementation". *Proceedings of AIS*, 1998.

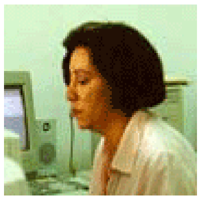
[O´Brien99] James O´Brien: *Introduction to Information System*. 8th. Ed. Irwin Book Team, 1999.

[Ortega00] M. Ortega, M. Pérez and T. Rojas: "A Model for Software Product Quality with a Systemic Focus". July 2000. Presented in *The 4th World Multiconference on Systemics, Cybernetics and Informatics SCI 2000 and The 6th International Conference on Information Systems, Analysis and Synthesis ISAS 2000*. Orlando, Florida, USA.

[Kazma00] R. Kazman, M. Klein, and P. Clements: *ATAM: Method for Architecture Evaluation*. http://www.sei.cmu.edu /pub/documents/00.reports/pdf/00tr004.pdf , August 2000

[Turba01] Rainer Turban and Potter: *Introduction to Information Technology*. John Willey & sons inc. N.Y. pp. 338-340.

[Domín01] Kenyer Domínguez: *Sistema de Gestión del Conocimiento para Proyectos de Investigación*. Trabajo Especial de Grado. Universidad Simón Bolívar, 2001



María Pérez is member of the Association of Information Systems. She is titular professor at Simón Bolívar University where she heads a research group on Information Systems (LISI). She got a Ph.D. Computer Science degree in 1999. Some of her publications are: ISACC'95 (Mexico), AIS'96 (USA), AIS'97 (USA), AIS'98 (USA), AMCIS '99 (USA), CLEI '99 (Uruguay); JOOP 12(6); AMCIS '00 (USA); Journal Information & Software Technology, 2000; JOOP, 2000; SCI '00 (USA); Revista de la Facultad de Ingeniería de la UCV 15(2); AMCIS '01 (USA), JIISIC '01 (Argentina), Revista colombiana de computación 2(2), ISAS'02 (USA), AMCIS'02 (USA), JIISIC'02 (Brasil), SCCC'02 (Chile), SAIS'03 (USA). She can be reached at movalles@usb.ve

Anna Grimán is professor at Simón Bolívar University where she is part of LISI research team. She got a MSc. Systems Engineering degree in 2000. Some of her publications are: SCI '00 (USA), AMCIS '01 (USA), JIISIC '01 (Argentina), CLEI'01 (Venezuela), AMCIS '02 (USA), Information Systems Management 19(2); Revista de la Facultad de Ingeniería de la UCV, Vol. 16; CLEI Electronic Journal, Vol. 15., Software Quality Profesional 4(4), JIISIC'02 (Brasil), AMCIS'02 (USA), SCCC'02 (Chile), SAIS'03 (USA), Information System Management 20(1). She can be reached at agriman@usb.ve.