A COMPONENT-BASED METHOD FOR DEVELOPING WEB APPLICATIONS

Jonas A. Montilva C. and Judith Barrios A.
University of Los Andes, Faculty of Engineering, School of System Engineering, Computer Science Department, Merida, Venezuela
Email: jonas@ing.ula.ve, ijudith@ing.ula.ve

Keywords: Component-Based Software Engineering, Web Engineering, Web Information Systems

Abstract: We describe, in this paper, a component-based software engineering method for helping development teams to plan, organize, control, and develop web applications. The method is described in terms of three methodological elements: a product model that captures the architectural characteristics of web applications, a team model that describes the different roles to be played by the members of a team during the development of web applications, and a process model that integrates the managerial and technical activities that are required to develop componentized web applications of high quality. The main features of the method are its component-based approach that helps reduce costs and development time; its ability to integrate managerial and development processes into a unique process model; and its emphasis on business modelling as a way of gaining a better understanding of the application domain objectives, functions and requirements.

1. INTRODUCTION

Developing web applications is a very complex process that demands highly skilled personnel. Its complexity is mainly due to the fact that it involves many different information technologies including WWW, human factors, information systems, business modelling, programming languages, distributed systems, and databases. Some typical examples of web applications are web information systems, e-commerce systems, e-business portals, e-government systems, and instructional web sites.

A set of modelling languages, methods, techniques, guidelines, and methodologies have emerged, during the last three years, to help development teams to produce better web applications. A language called WebML is devoted exclusively to the web development process (Ceri, et al, 2000). Some of the most well-documented methods have been proposed by Conallen (2000), Fraternali and Paolini (2000), Hadjerrouit (2001), Gaedke and Rehse (2000), Allen (2001), and Atzeni, et al (1997). The Conallen's method is an adaptation of the Rational Unified Process. It uses an object oriented approach to model web applications. Autoweb (Fraternali and Paolini, 2000) and ARANEUS (Atzeni et al, 1997) makes more emphasis on the hypermedia nature of web applications. The method proposed by Hadjerrouit (2001) stresses the use of software engineering process in the development of web application, but recognizes the relevance of user interfaces in this type of applications. Gaedke and Rehse (2000) and Allen (2001) emphasize the importance of reusability in the process of developing web applications. Their methods apply principles and process models borrowed from Component-Based Software Engineering.

We believe that a component-based approach to the development of web applications has several advantages over those approaches based on object-orientation, hypermedia, and human factors. Firstly, the reuse of components helps reduce the cost and time to deliver a solution, since the application is not designed and nor written from scratch. Secondly, the component-based approach promotes a better quality of the product due to the reuse of already proven components. Finally, better design of the web application architectures can be obtained by reusing architectural patterns that promote separation of concerns.

We propose, in this paper, a component-based method for developing web applications. The

Published in the Proceedings of the 5th International Conference on Enterprise Information Systems (ICEIS'2003), Angers, France, 23-26 April, 2003.

method uses a watch metaphor, described by Montilva, et al (2000), for integrating the management and development processes that a team requires to plan, organize, coordinate, control, and develop web applications using a component-based approach.

The design of our method, called the *Watch* method, was based on principles and concepts taken from Method Engineering (ME) and Software Engineering (SE). According to Brinkkemper (1996), a well-defined method should be described in terms of two elements: a model of the product to be developed and a process model that explains how to develop the product. We added a third element: a team model, which describes the roles that the members of the development team play during the development process.

Section 2 describes the *product model* which defines the generic characteristics of web applications. The *team model* is outlined in Section 3. The *process model* that describes the managerial and technical activities that are needed to develop web applications is explained in Section 4. Finally, the concluding remarks are given in Section 5.

2. THE PRODUCT MODEL

The product model of the Watch Method is described in terms of a domain architecture that is based on the architectural styles and patterns proposed by Conallen (2000) and Curphey (2002). The structural view of the model is shown in Figure 1. This view organises the components of a web application into three layers. Each layer includes one or more tiers, as described next.

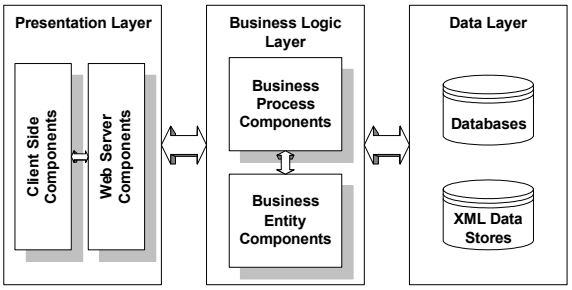

Figure 1. The product model of the method

The presentation layer is responsible for the interaction with the end users. It deals with the presentation of information to the users, the capture of inputs, and the management of the user's dialog. This layer is made of two tiers, one for each type of component: client-side and web server components. The client-side components are related with the HTML pages of the web application that are rendered by web browsers on client machines. They provide user interface functionality that is not achievable by HTML elements alone by using Java Applets, ActiveX controls, or Scriptlets. The web server components, on the other hand, are executed by web servers. They may be implemented as scripts or as compiled binary files.

The business logic layer consists of two kinds of components: business process components and business entity components. The business process components handle the services or transactions that are requested by users through the user interface. They determine the operations of the business entity components that must be invoked and the order in which they must be executed. The business entity components are persistent. They represent the business entity types of the application domain whose state must be stored by the application.

The data layer is responsible for managing the data stores where the states of the business entities are saved. Databases and XML data stores are the main architectural components of this layer.

Figure 2 illustrates the main architectural components of a web application and their relationship with the deployment infrastructure used to install and execute these components.

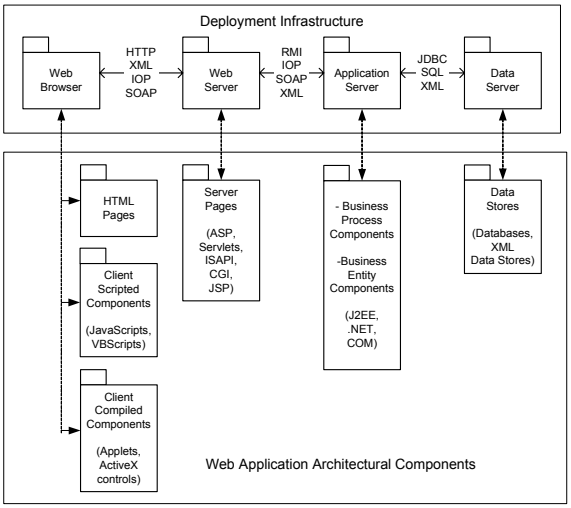

Figure 2. Main components of a web application

3. THE TEAM MODEL

An important aspect of any software development project is the organisation of the effort that will be required to develop the application. The definition of team roles are important because it helps the project

leader to select the right people and assign them based on the skills needed to perform the different types of task that a project requires. We will refer to the group of people that participates in the web application development process as the *development team*.

A development team can be organised in many different ways depending on the complexity of the web application to be developed. Allen and Frost (2001) provide a catalog of team roles organized into three groups: application development roles, component development roles, and support roles. Based on this catalog, we chose a set of team roles and adapted these roles to fit the development of componentised web applications. The resulting team configuration is made of the following roles and its associated responsibilities:

The **project manager/team leader** has associated the responsibility for planning, organizing, directing (leading), supervising, and controlling the application development process.

The **adviser user/ambassador user** brings to the project the knowledge about the business or application domain. This role must provide the business requirements and act as a bridge between the development team and the user community.

The **application developer/senior developer i**nterprets and models user requirements and uses technical skills to design and evaluate application architectures and their components.

The **web developer** is responsible for specifying, designing and developing web-based user interfaces. This role encompasses human factors skills (e.g., graphic design) and technical skills related to the web technology.

The **business component developer** models and interprets business requirements and uses technical skills to design, code, integrate, test, and deploy business components.

The **data component developer** models and interprets data requirements and uses technical skills to design, create, integrate, and evaluate databases and other types of data stores (e.g. XML data stores).

This configuration includes only core roles, i.e., roles that are more or less permanent throughout the project. Depending on the complexity and size of the project, more roles and responsibilities can be added or removed from this configuration.

4. THE PROCESS MODEL

The structure of our process model is based on the IEEE 1074 Standard for developing software life cycles processes (IEEE, 1995) and the UML Component process for developing component-based software (Chessman and Daniels, 2000). We chose and adapted those process groups of the IEEE 1074 standard and the UML Component process that are most appropriate to the development of small and medium size web applications. These processes were then reorganized into two groups: management and development processes, as shown in Table 1.

Table 1. The process structure of the method

Management Process Group	Development Process Group
Project Management Software Quality Management Software Configuration Mgmt. Verification and Validation V&V Risk Management Training	Business Modelling Requirements Definition Architectural Design Component Specification Component Provisioning Component Assembly Application Testing Application Delivery

The management process group encloses those activities that are concerned with the processes of managing the project, assuring the quality of the application, managing changes and risks, and training the team. The development process group, on the other hand, encompasses the technical activities that are needed to model the application domain, elicit and specify requirements, and design, code, test, and deliver the application. The development process group is adapted from the UML Components process.

We used a watch metaphor for designing the structure and dynamics of the *Watch* process model. Using this metaphor, the development of a web application can be seen as a watch or clock whose moving hands - the development phases - can be set and controlled by the project manager (see Figure 3).

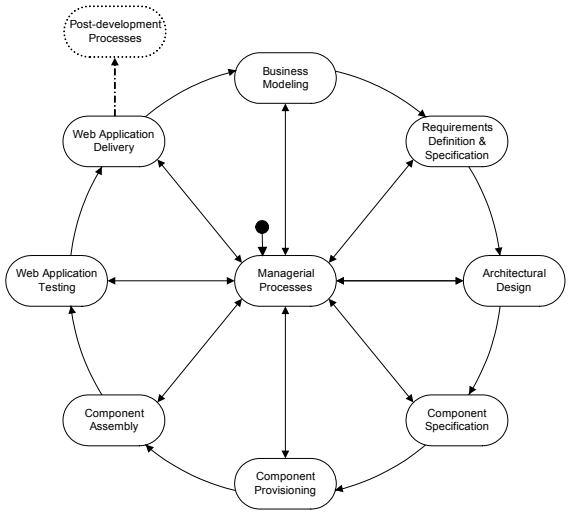

Figure 3. The Watch process model

The development of a web application starts at the center of the model, i.e., at the management process. Project initiation, project planning, and team organisation are the first project management activities to be performed. Business modelling is the first development phase to be executed after the project has been planned and organised. The other development phases are then performed in clockwise order under the control of the V&V processes, which determine, through a product review, if the progress of the project needs go back to a previous phase in order to correct failures, errors or deficiencies in the deliverables or products of the project.

For space reasons, we describe in the next section the phases of the development process only. The management process is outlined in Montilva, et al (2000).

4.1 The Business Modelling Phase

This phase identifies and describes in detail the business domain to be served by the web application. The purpose of this phase is to allow the development team to get a good understanding of the business domain before initiating the requirements definition phase. The deliverable of this phase is the *Business Model* which captures the objectives, business processes, business objects, actors, business rules, events, and job structure of the business domain.

This phase starts by defining the scope and objectives of the business domain. The business processes and actors are identified and modeled next. Business processes are executed by a set of actors (e.g. managers, employees or customers) with the support of the web application. Business objects and rules are modeled next. Business objects are the types of entities that are related to the domain (e.g. clients, accounts, resources and products) and whose state are accessed or modified by business processes. Business rules are precise statements that describe, constrain and control the structure, operations, and strategies of a business (ILOG, 2001). Events are then identified. An event triggers the execution of one or more business processes. They capture the dynamics of a business. Finally, the job structure of the business domain must be represented. A job structure describes the organisation of roles, responsibilities and authority lines of the business.

Eriksson and Penker (2000) describe an extension to UML that can be used to build the structure of the business. The dynamics aspects of the business can be modeled using Petri Nets extensions for workflows, such as the Proclets notation described by Van Der Aalst, et al (2000).

4.2 The Requirement Definition & Specification Phase

The purpose of this phase is to discover, define and specify the functional and non-functional requirements to be satisfied by the web application. The main deliverables of this phase are the Requirements Definition Document (RDD) and the Requirements Specification Document (RSD). The RDD documents the functional and non-functional requirements as seen by users. It is written in plain English with the help of the users. The adviser and ambassador users plays a major role in the elaboration of this document, which is written for the users applying the terminology or "business language" of the application domain. The RSD expresses the user's requirements in a formal or technical way that can be understood by the developers without any ambiguity. It describes the user's requirements using a formal, semiformal, or graphical notation or language, such as the Unified Modelling Language (Booch, et al, 1999).

The structure and behavior of the web application are specified using three models: functional, structural and dynamics. The *functional model* of the application can be built using the UML use case notation, which helps describe the functionality to be provided by the application. The *structural model* may be built using UML class diagrams. This model captures the classes of business entities to be managed by the system, their relationships, their attributes, and theirs operations. The *dynamics model* represents the behavior of the application. UML sequence diagrams can be applied here to describe how each function specified by the functional model must be performed by the application. Workflow techniques can also be very helpful in documenting the application dynamics. Petri Net based techniques such as PROCLETS (Van der Aalst, et al, 2000) and WF nets, UML swimlanes and activity diagrams (Booch, et al., 1999), can be applied to describe the interaction between the business processes and the application.

4.3 The Application Design Phase

This phase translates the requirements specification into a solution, i.e., a design specification of the web

application. The deliverable of this phase is the Application Design Document (ADD) which describes and specifies the web application architecture, the user interface and the database(s) to be used by the application.

Following the structural view of the product model, we divided this phase into the following three steps:

a) **User Interface Design.** The purpose of this step is to define the structural, aesthetics and visual characteristics of the web application user's interface. The web developers design first the user interface structure based on the use cases described by the functional model. The aesthetics and visual properties of the interface are then decided with the collaboration of the adviser users. A prototype of the user interface is usually made, in this step, to visualize these properties and allow the users to validate and incorporate new requirements. Web design principles, such as those described by Lynch and S. Horton (1999), can be used to ensure the usability requirements imposed to the application.

b) **Architectural Design.** In this step, the development team designs the web application architecture. This architecture is made of a set of software components, connectors, and constraints that are normally organized into layers or tiers. The product model, shown in Figures 1 and 2, provides an architectural pattern that can be refined, extended or elaborated using the requirements, in order to define the application architecture. Web application architecture patterns, such as those described by Conallen (2000), can also be used to design the application architecture. Once the global view of the architecture is defined using the patterns, the components that conform the three architectural layers (presentation, business logic, and data layers) must be identified and initially specified. Each of these layers is made of a set of components. Each component is, in turn, composed of a collection of interrelated classes and one or more interfaces that define the services offered by the component.

c) **Database Design.** The data component developer must define, in this step, the type, structure and content of the data store(s) to be used by the application. The conceptual design of the data components (databases or XML data stores) must be produced in this step. The structural model contained in the RSD document provides the information needed to design in detail the components of the data layer.

4.4 The Components Specification Phase

The purpose of this phase is to specify, in more detail, the components that comprise the application architecture and the interactions between these components. The component model and the deployment framework to be used for implementing the components must also be defined in this phase. A component model specifies the standards and conventions used to implement the components. CORBA, J2EE and .NET provide their own component models that describe how the components interact in a distribute environment. A deployment framework provides a standard compose-time and run-time environment for deploying the components and executing the application (Bachman, et al., 2000).

The main deliverable of this phase is the Component Specification Document (CSD). A component specification is made for each component that comprises the presentation and business logic layers of the application architecture. The data layer is, commonly, comprised by a collection of tables or relations that comply with the Relational Model. Its data components do not require to be specified as programmed components, because of the nature of its constituents. Instead, the database conceptual design produced in the previous phase must be transformed into an implementation design and a physical design based on the DBMS used at the data layer. The relationships or mappings between the business entity components in the business logic layer and the data components in the data layer must also be specified in this step.

Contracts are useful mechanisms for specifying the presentation and business logic components. Two types of contracts can be used to specify these components: usage contract and realization contract (Cheesman and Daniels, 2001). A usage contract specifies the interface of a component. An interface is a set of operations. Each operation defines a service (functionality or responsibility) that the component instances will perform when the clients request this service. The usage contract includes: (1) the list of operations that the interface provides, (2) a specification for each operation of the interface that describes the input and output parameters, and the constraints that apply to the operation, (3) a specification of the set of classes that implements the interface, and (4) the pre- and post conditions that specify the effect of each operation. A

realization contract, on the other hand, is concerned with the implementation of a component specification. A realization contract must be consistent with the component model. The component model and the deployment infrastructure must, therefore, be defined before specifying the realization contracts. A realization contract describes the interactions between the implementation of the component specified by the contract and the other components. The mapping of the components and the deployment infrastructure must also be specified by the contract.

4.5 The Component Implementation Phase

The purpose of this phase is to implement the design specifications produced in the Application Design and the Components Specification phases. The main deliverable is a set of presentation, business logic and data components that are ready to be assembled.

This phase is divided into four steps:

a) **User Interface Refinement.** The user interface prototype, produced in the Application Design phase, is refined in this step to accommodate new aesthetics and visual requirements. This prototype is an HTML user interface framework that is ready to be assembled with the presentation components.

b) **Component Provisioning.** The purpose of this step is to acquire or develop the presentation and business logic components that were specified in the Component Specification Phase. Reusability is a normal practice to be applied in this step. The first activity to be performed in this step is, therefore, the search for components that can be reused from previous projects, component repositories, or acquired from third parties. Based on the results of this searching step, the development team must accomplish next some of the following activities:

- **Acquire.** Some components can be purchased from third parties or specialized software companies.
- **Subscribe.** The services to be offered by some components can be externally supplied on a subscription basis as web services or megaservices.
- **Wrap.** The functionality required from some components can be provided by legacy applications that must be wrapped to provide the required services.
- **Adapt.** Components that have been used in previous projects or that have been found in internal component repositories can be adapted to comply with component specifications and implementation restrictions, such as the component model and the infrastructure selected for deploying the components.
- **Develop.** Those components that can not be reused must be developed in-house from scratch based on the corresponding component specifications.

c) **Component testing.** The software components that were developed, adapted or wrapped are tested individually using the testing techniques and procedures that are specified in the V&V plan.

d) **Data components implementation.** The databases or XML data stores that were specified in the previous phases are implemented, in this step, using the selected DBMSs.

4.6 The Component Assembly Phase

The components that were acquired, adapted or produced in the Component Implementation phase must now be assembled to produce an integrated application. The implementation view of the application architecture is refined and used in this phase to guide the assembly process.

The user interface prototype is extended with the presentation components which, in turn, are linked to the business logic components using the appropriated middleware of the deployment infrastructure (e.g., RMI, IOP, SOAP, and XML). The business entity components must also be linked to the data components using middleware facilities such as XML, JDBC, and ODBC.

The assembly of components must be tested using top-down, bottom-up or combined integration testing techniques, as indicated by the V&V plan.

4.7 The Web Application Testing Phase

The functional, performance, and acceptation tests are prepared and conducted, in this phase, based on the testing plan provided by the management processes. The main outcome of this phase is a tested web application that is ready to be installed.

4.8 The Web Application Delivery Phase

In this phase, the web application is installed and tested in its operational environment. The users and operators are trained based on the training plan produced by the management processes. Once the application is fully operational, it can be formally delivered to the client.

The end of the Delivery Phase signals the beginning of the post-development processes, which take control of the initiation of the web application, its operation, maintenance, and retirement.

6 CONCLUSIONS

We have introduced in this paper a component-based method for developing web applications. The distinguished features of the method are the following: (1) its ability to integrate managerial and technical activities into a process model; (2) its emphasis on reusability of components that contributes to reduce development costs and time; (3) its ability to iterate between development phases which is essential to incorporate changes and new requirements; (4) the emphasis on validation and verification that contributes to increase the quality of the deliverables of the project, and (5) the addressing of nontechnical issues such as a business modelling that helps developers to understand much better the system's overall functions, its objectives, business processes, and requirements.

REFERENCES

Allen, P., 2001. *Realizing e-Business with Components*, Addison-Wesley, London.

Allen, P. and Frost, S., 2001. Planning Team Roles for CBD. In Heineman, G.T and Councill, W.T (Eds.), *Component-Based Software Engineering*. Addison-Wesley, Boston, 113—129.

Atzeni, P., Mecca, G. and Merialdo, P., 1997. *Design and Maintenance of Data-Intensive Web Sites* - Technical Report n. 25-1997 - Dipartimento di Informatica e Automazione, Universita' di Roma Tre, June., http://www.dia.uniroma3.it/Araneus/articles.html

Ceri, S., Fraternali, P. and Bongio, A., 2000. Web Modeling Language (WebML): a modeling language for designing Web sites. *WWW9 / Computer Networks 33*(1-6): 137--157.

Bachman, F. et al, 2000. Volume II: Technical Concepts of Component-Based Software Engineering. Technical Report, *Software Engineering Institute*, CMU/SEI-2000-TR-008, May.

Booch, G., Rumbaugh, J. and Jacobson, I., 1999. *The Unified Modeling Language User Guide*, Addison-Wesley.

Brinkkemper, S., 1996. Method engineering: Engineering of information systems development methods and tools. *Information and Software Technology*, **38**, 275--280.

Cheesman, J. and Daniels, D., 2001. *UML Components, A Simple Process for Specifying Component-Based Software*. Addison-Wesley, Boston.

Conallen, J., 2000. *Building Web Applications with UML*. Addison-Wesley, Reading, Massachusetts.

Curphey, M. et al, 2002. A Guide to Building Secure Web Applications and Web Services. *The Open Web Application Security Project (OWASP).* http://www.owasp.org

Fraternali, P. and Paolini, P., 2000. Model-Driven Development of Web Applications: The Autoweb System. *ACM Trans. on Information Systems*, 28 (4), 323-382.

Gaedke, M. and Rehse, J., 2000. Supporting Compositional Reuse in Component-Based Web Engineering, *Proc. Of the ACM SAC'2000 Conference*, Como, Italy, 927—933.

Hadjerrouit, S., 2001. Web-based Application Development: A Software Engineering Approach, *ACM SIGCSE Bulletin*, 33 (2), 31—34.

Eriksson, H. E. and Penker, M., 2000. *Business Modeling with UML: Business Patterns at Work*, John Wiley & Sons, New York.

IEEE Std. 1074, 1995. *IEEE Standard for Developing Software Life Cycle Processes*, IEEE Computer Society, New York.

ILOG., 2001. *Business Rules: Powering Business and e-Business*. White Paper. May. www.ilog.com

Lynch, P. J. and Horton, S., 1999. *Web Style Guide: Basic Design Principles for Creating Web Sites*, New Haven: Yale University Press.

Montilva, J., Hazam, K. and Gharawi, M., 2000. The Watch Model for Developing Business Software in Small and Midsize Organizations. *Proc. of the IV World Multiconference on Systemics, Cybernetics and Informatics - SCI'2000.* Orlando, Florida, Julio.

Van der Aalst, W.M.P, Barthelmess, P., Ellis, C.A., and Wainer, J., 2000. PROCLETS: A Framework for Lightweight Interacting Workflow Processes. *Int. J. of Cooperative Information Systems*, 1—40.