

The Development Process

The Web technologies discussed in the previous chapter enable the development of Web applications ranging from small, ad hoc solutions to complex Web information systems. Before focusing on the actual development of such kinds of applications, i.e., the *products*, we would like to focus on the *process* that leads to the creation of a Web application. Understanding software development processes (in general, not only for Web applications), the main development activities that need to be performed, their interconnections, and their temporal order are of fundamental importance for the success of a software product. In this book we follow a top-down approach to the organization of the contents and discuss first the way (Web) application development is organized, so as to discuss then the single activities that we will identify in this chapter.

Note that if we shift our point of view from that of the developer or project manager to that of the software product, what was before a *software development process* can now be seen as *software life cycle*, that is, a model that describes the life of an application, from its inception to its dismissal. *Software development process* and *software life cycle* are thus synonyms used in the literature depending on which view one prefers to highlight. As this is a book about Web engineering, we will use more the term *software development process*, though *software life cycle* may be used as well.

In this chapter, we will first discuss software development and its processes that are generally executed for any software product, in order to introduce the reader to the basic concepts and activities. We will then describe a possible development process more specific to Web applications and discuss its differences with more traditional development processes. We will also introduce some examples of concrete Web development processes, in order to introduce the reader with the peculiarities of the Web. The rest of the book will then be structured according to a typical process model.

3.1 Decomposing the Software Development Process

In today's software industry it is hard to find products that are planned, implemented, and tested by a single developer, as the complexity of modern (Web) applications typically requires the involvement of several different experts who are able to address specific development requirements more precisely. Depending on the size of the application and the actors involved in the development process, building an application may be an intricate undertaking, exposed to a variety of risks that might compromise the success of the final application. In order to control the software development process, it is thus of fundamental importance to understand its constituent activities, its actors, and their interconnections.

3.1.1 Activities in Software Development

Software development is a creative process leading to an innovative software product or system. Usually, this process is not just one monolithic block of work that takes as input some ideas about the application to be developed and produces as output a perfectly fitting solution; the process can be decomposed into a set of basic activities with well-defined boundaries and meanings. Such activities aim at *understanding* the problem, *planning* a solution, *carrying out* the plan, *examining* the result for accuracy, and *resolving* possible errors or inaccuracies. Traditionally, the software development process is organized into the following basic activities:

- *Requirements engineering*: aims at understanding the problem.
- *Design*: aims at planning a solution to the problem.
- *Implementation*: translates the plan into running application code.
- *Testing and evaluation*: aims at identifying coding errors or inconsistencies between the collected requirements and their implementation.
- *Deployment*: brings the solution to the customers.
- *Maintenance*: aims at monitoring a running system and keeping it healthy and running.
- *Evolution*: aims at improving the developed solution over time, providing new input to the development process in the form of new requirements.

More precisely, *requirements engineering* aims at understanding a product's needed capabilities and attributes. The analysis concentrates on *functional requirements*, referring to the functions that the system must be able to support, as well as on *nonfunctional requirements*, referring mainly to the quality of the offered solution. This implies identifying the general idea behind the system, as well as the stakeholders that require the new solution, the motivations for the production of a new system and the final usage environment. The collected requirements are elaborated with the aim of producing some high-level models of the system that abstract from irrelevant details of the problem domain.

After a subset of the application's requirements has been understood, the *design* can follow. The design activity aims at specifying a solution, which must meet functional and efficiency requirements, as well as possible constraints derived from the target environment. Requirements previously collected are therefore refined, restricted, and enhanced to satisfy possible technological constraints.

There are different views characterizing software design. For example, Pressman [Pre05] describes software design as a system of activities for data/class design, component design, interface design, and architectural design. Considering different separate views helps us shape better the specific aspects of the system, such as structure, behavior, interoperability, data, and control flow. It also enforces *separation of concerns*, a basic software engineering principle stating that approaching a problem by separating the different involved concerns may help us cope with complexity and achieve some required engineering quality factors such as adaptability, maintainability, extendibility, and reusability.

During *implementation*, the different design views are transformed either manually or with the help of automatic generation tools into corresponding program code (structured into modules and/or files), database tables, and configuration files. Implementation may require the use of existing code libraries, a variety of different programming languages and communication protocols, and different hardware devices.

The *testing and evaluation* activity is typically conducted in parallel with the previous activities, because the correctness and reliability of intermediate results – not only of the final product – is of fundamental importance to guarantee the quality of an application. The most relevant quality concerns addressed by this activity are related to *functionality* (i.e., the correctness of the application behavior with respect to specified functional requirements), *performance* (i.e., the throughput and response times of the application in average and peak workload conditions), and *usability* (i.e., ease of use, communication effectiveness, and adherence to consolidated usage standards).

The *deployment* of a ready application delivers the developed application to its users. Depending on the nature of the application, this activity may imply the installation of the software on client PCs, the setup of central application and database servers, the configuration of communication middleware, and so on. Closely related with the deployment of a new software solution is the instruction and training of the future users of the application, especially in cases where the delivered solution represents a radical change rather than an incremental one.

Maintaining a deployed and running application means keeping the application in a healthy state, so as to guarantee high availability and to reduce failures. This may imply periodical checks of log files, bug reporting, and the cleaning up of temporary files, as well as the application of bug fixes or security patches, in order to keep the application always up to date.

Finally, *evolution* of an application aims at addressing new requirements that typically only emerge once the application has been used for a certain amount of time, and users start providing their feedback and comments. Evolving an existing application is more than bug or error fixing, and addressing the new requirements may require the whole development process to start anew in order to apply the required changes to the application. In addition – despite the rigorous application of software engineering techniques – oftentimes only after the deployment of the application does it become clear that certain requirements have not been met, and the application needs to be adjusted accordingly.

3.1.2 Actors in Software Development

As already hinted at in the introductory paragraph of this section, usually the above activities are not performed by one and the same person. Instead, the software engineering experience has led to the definition of a set of professional profiles, each of which dedicated to specific problems or activities in the software development process:

- During requirements engineering, the *application analyst* collects the motivations that trigger the development of the application and turns them into a specification of the application requirements. In doing so, he interprets the long-term strategic business goals and constraints and transforms them into short-term, concrete, application requirements.
- In application design, the *data architect* focuses on those application requirements that deal with content and domain data. He produces a conceptual data model that organizes the data into a structure and a representation that can be accessed and used by the application.
- The *application architect* focuses on those application requirements that deal with the functions and services that are to be delivered. He develops a conceptual solution of the application logic (expressed by means of models, figures, or specification languages) that builds on top of the data model.
- Based on the specifications produced, the *programmer* or *developer* implements the solutions sketched by the data and application architects and tests and evaluates the implemented solutions. In most cases, the programmer also manages the deployment of the application.
- The application *administrator* is then the main actor in the deployment and evolution activities, being in charge of maintaining the application, providing for periodical backups, managing the community of users, and collecting feedback from the users.

Of course, the overall development process also involves the actual *users* of the application, especially in the evaluation of the usability of the application and its evolution over time. But users themselves are not actively involved in the production of the software artifact, the reason we do not list them as main actors in the development process.

3.2 Structuring the Software Development Process

The decomposition of the software development process into its basic activities and the identification of its main actors is a first step toward the successful management of the development process. A successful management, however, also demands some additional knowledge, i.e., the *order* of the activities and possible *transition criteria* [Boe88]. It is the structuring of the software development process into well-formalized *process models*, starting from the previously identified activities, which enables the easy definition of a suitable order and of intermediate results and milestones [GJM02].

3.2.1 The Waterfall Model

One of the first explicit formalizations of a development process is the so-called *Waterfall model*. The Waterfall model suggests a sequential organization of the development activities. Only completing one activity allows starting its successor activity. The completion of an activity is typically associated with the delivery of a product, e.g., documentation or program code; therefore, the Waterfall model is oftentimes regarded as a *document-driven* process model [Boe88].

The Waterfall model was probably the first popular process model, and it is still widely adopted in many development situations today. Its main shortcoming is its inflexibility in adapting already completed activities in response to changing requirements or knowledge emerging in later stages of the development process. Also, bad design decisions that are taken early in the process are propagated unchanged to subsequent activities, i.e., in a strict Waterfall model it is difficult to undertake retroactive actions to fix errors made in already completed activities.

A variance of the Waterfall model, which has been introduced to address this shortcoming, is the Waterfall model *with feedback*. It keeps the sequential order of activities, but also allows backward communication (e.g., from the implementation activity to the design activity) in order to accommodate changes that impact previous activities.

3.2.2 The Spiral Model

As time passed, it became increasingly more evident that the simple sequential order of the Waterfall model does not suffice to describe the real situation of many large software projects. Indeed, in most cases several of the constituent activities of the process model may need to be repeated two or more times, which is in clear contrast with the sequence imposed by the Waterfall model.

As an answer to the growing practice to iterate several times over the same activities, in 1988 Boehm [Boe88] proposed the so-called *Spiral model*, an incremental development process model that pays special attention to *risk management*. The Spiral model is graphically shown in Figure 3.1. The model

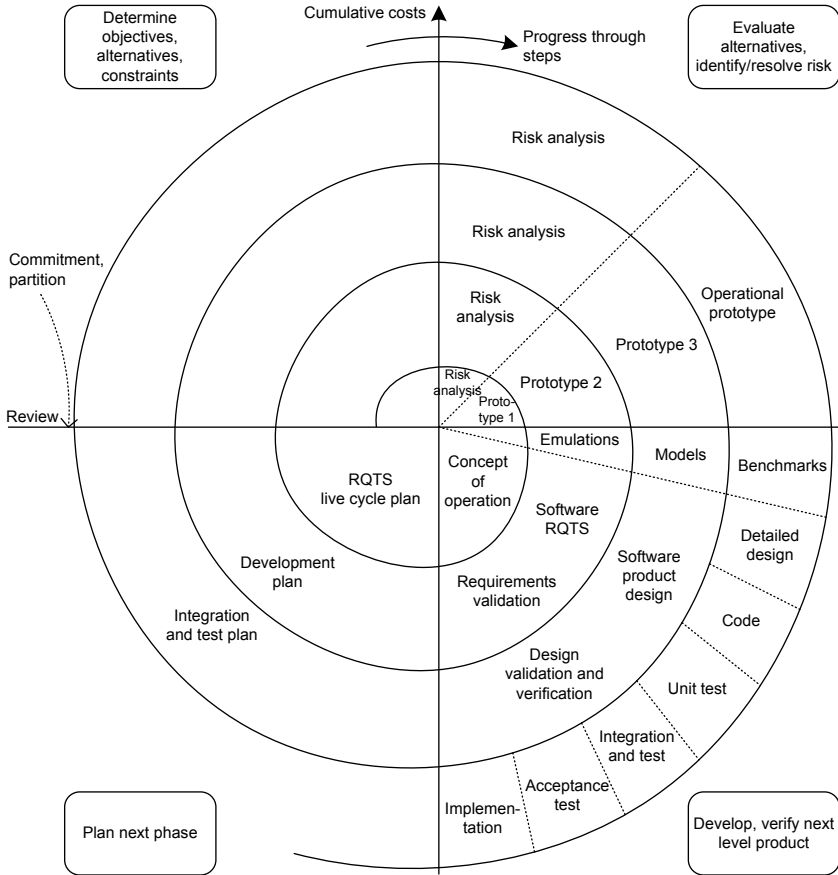


Fig. 3.1. The spiral model according to Boehm [Boe88]

explicitly suggests developing a software project in an incremental and iterative fashion by means of four convolutions, each one aimed at solving a specific development subproblem. Each convolution results in a prototype documenting the achievements of the respective convolution, accompanied by a risk analysis. The risk analysis considers various alternatives for achieving the project objectives, highlighting possible risks and their relevance, and suggesting solutions for preventing or eliminating such risks. The model is based on the idea that the incremental development of different versions of prototype applications implicitly reduces risk.

The Spiral model may also be interpreted as a *metamodel* that is able to accommodate different development models, adding risk as new dimension to the management problem [GJM02].

3.2.3 The Unified Model

Over time, the incremental or iterative practice of the Spiral model has inspired several other process models. One prominent example is the *Unified Software Development Process* (Unified process [JBR99]) and its adaptation to the development of Web applications [Con99] and Catalysis [DW98].

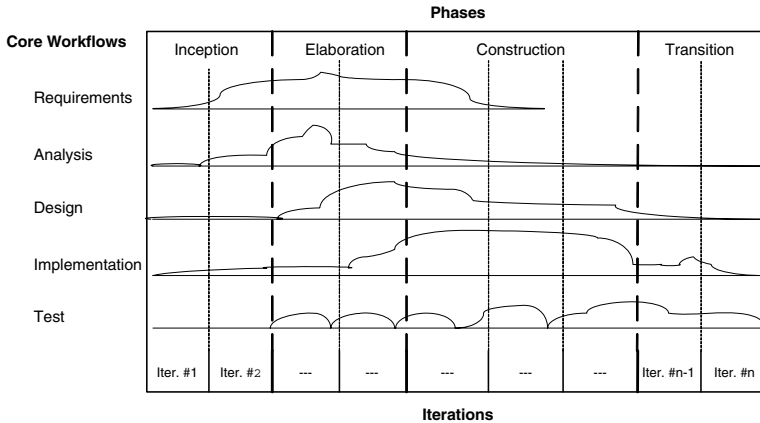


Fig. 3.2. Phases, workflows, and iterations in the Unified Process Model [JBR99]

According to the Unified process [JBR99], a software product is built along several cycles; Figure 3.2 shows a high-level view of the process. Each of the cycles ends with a product release and is executed in four separate phases:

- *Inception*: in this phase, the general idea of the system along with a tentative architecture and a set of critical use cases is developed.
- *Elaboration*: in this phase, several architectural views of requirements and design models are created and the most critical use cases are realized. At the end of the phase the project manager should be able to justify the resources to be allocated for the software project and to claim that the project is feasible under the identified risks and the granted budget and human resources.
- *Construction*: in this phase, the complete product is developed, and all the requested use cases are realized. Minor changes to the architecture are allowed if developers uncover better solutions. At the end of the phase, a product is transferable to the users.
- *Transition*: in this phase, a small group of experienced users tests the product and suggests improvements and discovers defects and shortcomings. The phase also involves personnel training and support. Finally, the product is exposed to the full user community.

Each phase is further divided into several iterations. The phases are executed according to known workflows: requirements, analysis, design, implementation, and test. Each workflow produces several artifacts. The adopted analysis and modeling techniques are those suggested by the Unified Modeling Language (UML) [Gro00].

3.2.4 Other Models

Recent practices influenced by agile development approaches, like extreme programming [Bec00], do not prescribe which activities should be executed and in which order. Organizations and particular projects may adopt just some of the above-mentioned processes, activities, phases, or workflows according to the needs of the projects. However, in order to be able to learn from projects, the adopted processes should be well defined. Well-defined, controlled, measured, and repeatable processes help organizations to continuously evaluate and improve software project management.

CMM and SPICE are reference models for organizing processes. CMM defines five levels of organizations with respect to the maturity of their software processes: *initial*, *repeatable*, *defined*, *managed*, *optimized*. The levels are characterized by the way in which software processes are defined, measured, controlled, and improved based on feedback. They are also characterized by the software development processes that are considered and standardized in the organization. The processes then are based on the activities described above. In addition, some other product management activities, like planning, risk identification, configuration management, contract management, project tracking, and process management (including peer reviews, training, quality management, process change management, and defect prevention), may also be involved. For further details on these, the reader is referred to books such as [Hum89, IPW⁺95, FC99] and reports such as [PCCW93, EG96].

3.3 Web-Specific Software Development Processes

Web applications are a special instance of generic software applications, and, hence, Web engineering can be seen as special instance of software engineering. Developing applications for the Web implies adhering to a few well-defined rules or conventions, which provide for a stable, robust, and scalable development and execution framework. Taking into account such Web-specific peculiarities allows a better tailoring of development process models. In the following, we introduce a characteristic process model, the so-called *online evolution model*, which stems from our experience in the development of Web applications and from the simple observation of the life cycle of modern Web applications that are available on the Web.

3.3.1 The Online Evolution Model

Figure 3.3 graphically shows the structure of the online evolution model. The model consists of five main activities, i.e., *requirements analysis*, *design*, *implementation*, *testing and evaluation*, and *maintenance and evolution*, and of seven transitions among the activities. The coarse activities in the online evolution model very much resemble the traditional activities in the software development process. A main difference is the interpretation of the *deployment* as transition, and not as a first-class activity. In the domain of the Web, deploying an application to its users is indeed not a big deal, as the centralized architecture typical of Web applications, the absence of independent client-side application code, and the browser as execution environment greatly facilitate and speed up the deployment activity.

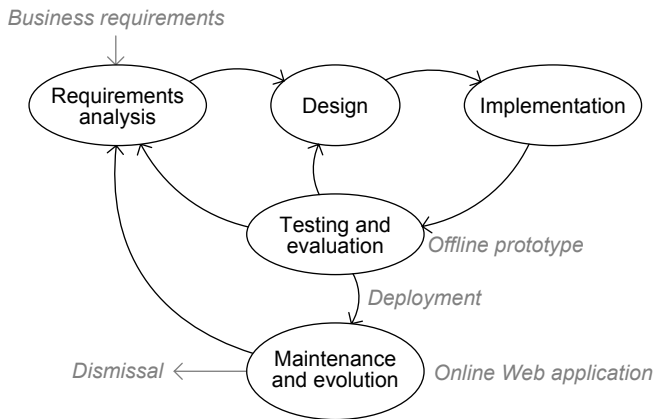


Fig. 3.3. The online evolution model for modern Web applications.

As for the transitions, the model proposes an explicit connection from the maintenance and evolution activity to the requirements analysis activity. It is this transition that characterizes the model: Connecting the maintenance and evolution activity to the requirements analysis activity closes a second cycle in the model that involves the requirements analysis activity; we call this the *evolution cycle*. The first cycle is the one that spans the design, implementation, and testing and evaluation activities; we call this the *build and test cycle*. The two cycles correspond to two phases that are peculiar to modern Web applications: offline development and online evolution. Indeed, as highlighted in Figure 3.3, the build and test cycle refers to the incremental development of the application that will go online, while the evolution cycle refers to the incremental evolution which the application undergoes over time once it is online.

In general, the two cycles are characterized by different cycling times: the former faster, the latter slower. The two iterative and incremental cycles

appear particularly appropriate in the context of the Web, where applications must be deployed quickly (in “Internet time”), and requirements are likely to change during the development phase. As a matter of fact, increasingly the common practice in Web development is to substitute documentation artifacts (as heavily adopted in the Waterfall model) with real application prototypes, and to involve end users as early as possible for testing and evaluation. Also, while in traditional software engineering an application is released only once all the requirements have been met, for Web applications it is more and more common practice (and desirable) to publish online applications, even though not all the requirements have been met yet. Early user feedback is becoming more important, and evolution is finally being seen as a real opportunity for improvement and less as an annoying adaptation of a functioning application. The evolution cycle is thus increasingly gaining importance.

As an example of this trend, consider the Google search engine’s Web site. Although users might not always be conscious of simple changes, Google is continuously evolving the features and interface of the search engine. According to Adam Bosworth’s (Vice President at Google) keynote speech at the 2007 International Conference on Web Engineering, Google is constantly adding new features to its Web applications, measuring whether the expected improvements or user behaviors can be accommodated, and consolidating features that prove their viability. Think, for instance, of the Web site thumbnails accompanying the single search results that were added some time ago to enrich the user’s browsing experience, but then dropped because of little value to users who were not yet familiar with those sites. A successful evolution, instead, was the introduction of the suggestion to switch to Google Maps for user inquiries that contain location data.

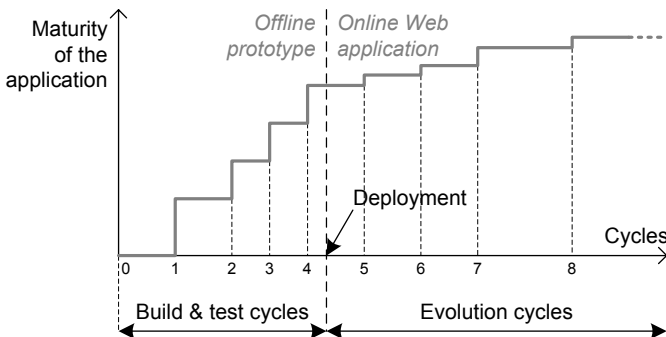


Fig. 3.4. The maturity of the Web application increases as a result of the combination of incremental development and continuous evolution

The effect of the two development cycles in the online evolution model on the maturity of the actual Web application under development is schematically represented in Figure 3.4. The incremental releases of the offline prototype in

the build and test cycle occur rapidly, and the maturity of the application increases in big steps. After the deployment of the application, the incremental upgrades of the online application occur less frequently (evolution cycles), and they add less value to the application. However, they to add value, continuously, and are an integral part of the application's life cycle.

The online evolution model discussed here is not intended to prescribe any rigid development process for Web applications. Rather, it describes the product life cycle of modern Web applications, as can be determined by observing the online dynamics of such kinds of applications. As such, the online evolution model may also be the result of the application of, for example, the Unified process to Web applications. In that case, however, each instance of the evolution cycle would result in a new instantiation of the development process.

It is worth noting that taking into account the peculiarities of the Web also allows us to further refine the core development activities of Web applications, adding a second layer of detail to the development process model. Web applications share some architectural, technological, and usage characteristics that allow us to further separate the previously discussed development activities into smaller concerns. For instance, the design activity of Web applications can typically be separated into *data design*, *navigation design*, and *presentation design*.

At this stage, we will not deepen the structuring of the above-described Web development activities into their subactivities. The following chapters, however, will discuss the single activities of the development process by providing some more insight into Web-specific peculiarities.

3.3.2 Web-Specific Actors

Independently of the previous analysis of the main development activities, we can say that Web application development involves the actors already discussed in Section 3.1.2, with two additional roles that are integer to the Web: the *graphic designer* and the *webmaster*.

Graphic designers are in charge of presentation design. The graphical appearance of Web applications is very important for both usability considerations and the attractiveness of the application. Graphic designers conceive the graphical appearance of the application, structure contents and images into layouts, and select suitable style properties (e.g., fonts, colors, and the size of images) based on the nonfunctional requirements dealing with the customer's graphical corporate identity and with acknowledged communication standards. The strong separation of concerns applied to the design activity demands only little, if any, programming skills from graphic designers,¹ which fully enables them to work with the software and design tools they are used

¹ We will come back to this consideration in Section 7.1 when discussing some presentation implementation issues.

to and to integrate their sketches with little effort. Especially, the use of so-called *mock-ups* (graphical interface prototypes that do not yet support any real application feature) is very popular for discussing appearance characteristics with the customer and to get early user feedback.

Webmasters are in charge of the maintenance and partly also the evolution of a Web application. Typically, each Web application that is online offers somewhere (e.g., in the contacts page or in the footer of the pages) the possibility to contact a person (the webmaster) to communicate, for example, broken links or other problems with the application. The role of the webmaster is common practice today, and it is new in the software development scenario, as there is no such role in traditional software development processes.

Considering that a large part of the applications developed for the Web can be categorized as content management systems or data-intensive Web applications, we can distinguish two more actors, not directly involved in the development of the application itself but rather focusing on the contents of the application: the *content author* and the *content manager*. The content author creates new content (e.g., news articles, documentations, photos, blog entries, etc.) to be added to and published by the application. The content manager is responsible for content aggregation, content evaluation, quality assurance, and the final publishing.

3.4 Examples of Web-Specific Development Processes

The previous discussion introduced the reader to the online evolution model. In this section, we describe a few Web-specific application development models that can to some extent be accommodated by the online evolution model. The models refer to three well-known conceptual Web application development methods, i.e., WebML [CFB⁺02], WSDM [TL98], and OOHDM [SR98]. They will be explained in more detail in Chapter 5 when discussing the design phase in the development process.

3.4.1 The WebML Model

The Web Modeling Language (WebML) [CFB⁺02] is a visual language and development method for specifying the content structure of a Web application and the organization and presentation of contents in the form of hypertexts. The WebML method was proposed in 2000 [CFB00a] and then refined until it ensured a complete coverage of the development process [BCC⁺03], thanks to the availability of generative techniques supporting the automatic production of the application code.

The main contribution of WebML is the proposal of a mix of concepts, notations, and techniques for the construction of data-intensive Web applications, which can be used by Web development teams to support all the

activities of the application life cycle, from analysis to deployment and evolution. The proposed mix blends traditional ingredients well known to developers, such as Use Case specification with UML and conceptual data design with the Entity-Relationship model, with new concepts and methods for the design of hypertexts, which are central to Web development. Therefore, the value of the proposed approach is not in the individual ingredients, but in the definition of a systematic framework in which the activities of Web application development can be organized according to the fundamental principles of software engineering, and all tasks, including the more Web-centric ones, find adequate support in appropriate concepts, notations, and techniques.

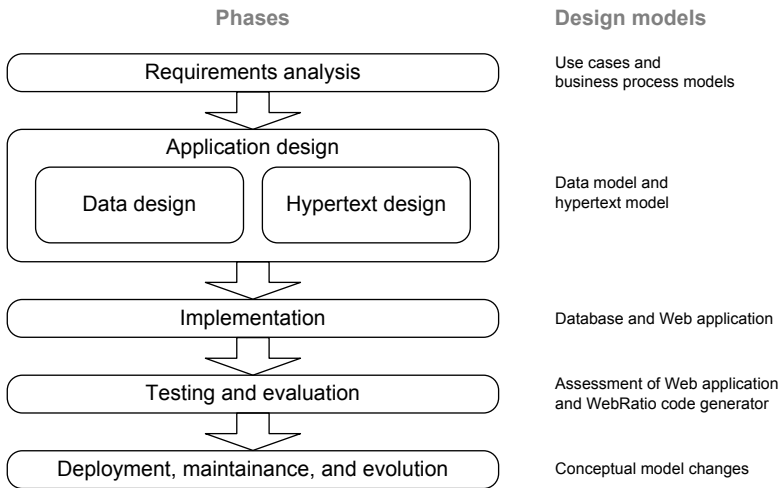


Fig. 3.5. Phases in the WebML development process [CFB⁺02]

Figure 3.5 shows the WebML approach to the development of Web applications. Inspired by Boehm’s Spiral model, and in line with modern methods for Web and software application development [Con99, JBR99], the WebML process is applied in an iterative and incremental manner, in which the various phases are repeated and refined until results meet the application requirements. The product life cycle therefore undergoes several iterations, each one producing a prototype or a partial version of the application. At each iteration, the current version of the application is tested and evaluated, and then extended or modified to cope with the already collected requirements, as well as with newly emerged requirements.

Out of the entire process illustrated in Figure 3.5, the “upper” phases of analysis and design are those most influenced by the adoption of a conceptual model. The WebML method therefore focuses on them. However, as shown in the rest of this section, the adoption of a model also benefits the other phases.

Requirements analysis

In WebML, the requirements analysis phase aims at producing the following results:

- The identification of the *groups of users* addressed by the application. Each group represents users having the same profile, or performing the same activities with the same access rights over the same information classes.
- The specification of *functional requirements* that address the functions to be provided to users. For each group of users, the relevant activities to be performed are identified and specified; each activity is a cohesive set of elementary tasks.
- The identification of *core information objects*, i.e., the main information assets to be accessed and/or manipulated by users.
- The decomposition of the Web application into *site views*, i.e., different hypertexts designed to meet a well-defined set of functional and user requirements. Each user group will be provided with at least one site view supporting the functions identified for the group.

The WebML method does not prescribe any specific format for requirements specification. However, table formats are suggested for capturing the informal requirements (such as the group description table or the site views description table). UML use case diagrams and activity diagrams can be also used as standard representations of usage scenarios.

Application design

Application design is achieved by means of WebML-based conceptual schemas, which express the organization of the application domain and navigation components at a high level of abstraction, independently of implementation details. According to Figure 3.5, application design involves two activities:

- *Data Design*: corresponds to organizing core information objects identified during requirements analysis into a comprehensive and coherent data schema.
- *Hypertext Design*: produces site view schemas on top of the data schema previously defined. The distinguishing feature of the WebML approach is the emphasis on conceptual modeling for hypertext specification.

The models provided by the WebML language for data and hypertext design will be better described in Chapter 5.

Implementation

The WebRatio CASE tool [CFB⁺02, Web07b] largely assists designers in the implementation of the database and of the Web application. First of all, it offers a visual environment for drawing the data and hypertext conceptual

schemas. Such visual specifications are then stored as XML documents, and these are the inputs for the WebML code generator, which supports data and hypertext implementation. In Section 7.2.2 of this book we will come back to the WebRatio CASE tool and provide a brief discussion of its features.

Testing and evaluation

The WebML model-driven approach benefits the systematic testing of applications, thanks to the availability of the conceptual model and the model transformation approach to code generation [BFTM05]. With respect to the traditional testing of applications, the focus shifts from verifying individual Web applications to assessing the *correctness of the code generator*. The intuition is that if one could ensure that the code generator produces a correct implementation for all legal and meaningful conceptual schemas (i.e., combinations of modeling constructs), then testing Web applications would reduce to the more treatable problem of validating the conceptual schema.

WebML development also fosters innovative techniques for quality evaluation. The research in this area has led to a framework for the model-driven and automatic evaluation of Web application quality [FLMM04, LMM04, MLME04]. The framework supports the *static* (i.e., compile-time) analysis of conceptual schemas, and the *dynamic* (i.e., runtime) collection of Web usage data to be automatically analyzed and compared with the navigation dictated by the conceptual schema. The static analysis is based on the discovery in the conceptual schema of design patterns, and on their automatic evaluation against quality attributes encoded as rules. Conversely, usage analysis consists of the automatic examination and mining of enriched Web logs, called conceptual logs [FMM03], which correlate common HTTP logs with additional data about i) the units and link paths accessed by the users, and ii) the database objects published within the viewed pages.

Maintenance and evolution

In the WebML model-driven process maintenance and evolution also benefit from the existence of a conceptual model of the application. Requests for changes can in fact be turned into changes at the conceptual level, either to the data model or to the hypertext model. Then, changes at the conceptual level are propagated to the implementation. This approach smoothly incorporates change management into the mainstream production life cycle and greatly reduces the risk of breaking the software engineering process due to the application of changes solely at the implementation level.

3.4.2 WSDM

The Web Site Design Method² was initiated by De Troyer and Leune [TL98] in 1998, and therefore was one of the first Web design methods.

² Later re-baptized as Web Semantics Design Methods.

Although WSDM was originally aimed at creating kiosk Web sites, it steadily evolved to a complete (semantic) Web design method supporting both functionality and a wide range of additional design concerns (localization, accessibility, semantic annotations, adaptivity, etc.). Some of these issues will be discussed in more detail later in this book.³

WSDM is a multi-phase Web design method, where each phase focuses on one particular design concern. It possesses the following characteristic features:

- *Methodology*: more than other methods, WSDM is a methodology. In addition to offering explicitly defined design primitives and models to describe a Web application at different levels of abstraction, WSDM also offers the designer aid on how to obtain the instantiations of these different models in order to obtain a well-structured, consistent, and usable Web application. WSDM thus offers the designer guidelines and techniques, thereby providing an explicit and systematic way to define Web applications.
- *Audience-driven*: consistent with knowledge established from user interface design and usability research, WSDM recognizes the importance of the users and thus takes as an explicit starting point an analysis of the different kinds of users (called audiences) and their (different) requirements and characteristics. This analysis will subsequently steer the impending design. Such an approach, where the users are taken as a starting point for the further design, is called an *audience-driven* approach.
- *Semantic Web technology*: with the rise of the Semantic Web, WSDM has been transformed to take advantage of Semantic Web technology. This was done in two ways. First of all, the Semantic Web Ontology Language OWL was used internally to define the different WSDM design models and for describing the information and functionality present in the Web application. Secondly, WSDM was also extended to generate semantic annotations alongside the (traditional) Web application, thereby effectively enabling the Semantic Web.

Figure 3.6 shows the different WSDM phases, along with the design models each gives rise to. As already explained, all these models are expressed in the Semantic Web Ontology Language OWL. Together they form the WSDM Ontology.

The next paragraphs explain in more detail each of WSDM's design phases:

Mission statement

The specification of the mission statement is the first phase of the WSDM design process. The intention is to clearly set the boundaries for the design by identifying the purpose of the Web site, the topics, and the target users. The mission statement is used during the design to ensure all required information

³ Interested readers can read all about WSDM on <http://wsm.vub.ac.be/>.

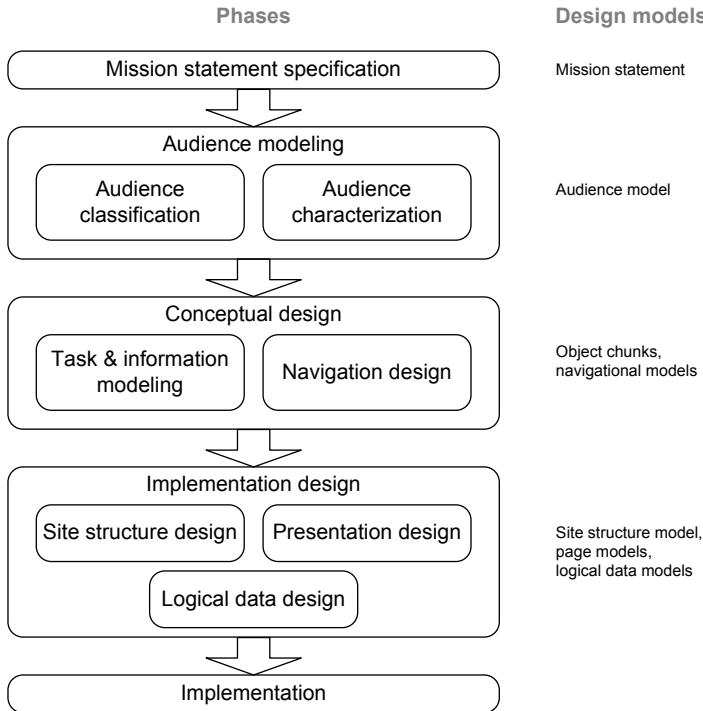


Fig. 3.6. Overview of the WSDM design method

and functionality are present, and all targeted users are supported. After the design process it is used to verify whether the formulated goals set for the Web application have been fulfilled. The mission statement is formulated in natural language.

Audience modeling

During the audience modeling phase, WSDM takes into account the fact that different visitors may have different needs and goals, and thus require particular support more specifically tailored to their needs. During the *audience classification* subphase, the targeted visitors, who were informally identified in the mission statement, are refined and classified into audience classes. An audience class is a group of visitors that has the same information and functional requirements. Any audience class that has (the same or) some additional requirements compared to another audience class is called an audience subclass. This partial order relationship gives rise to a hierarchical structure, called the audience class hierarchy. The “visitor” audience class is always the top of this hierarchy. It represents the requirements that all visitors have in common. During the *audience characterization* subphase, for each audience class, the characteristics, navigation, and usability requirements for their members are

also formulated. These will be taken into account in the subsequent design phases.

Conceptual design

The conceptual design phase is split into two subphases: *task and information modeling* and *navigation design*.

During task and information modeling, the designer models the tasks that need to be performed by the different audience classes, along with the required content and functionality. For each requirement that was formulated during audience modeling, a task model is defined. Each task model consists of a decomposition of the task needed to fulfill the particular requirement into elementary tasks, along with the temporal relations between them (e.g., sequential, order-independent). To perform this task analysis, WSDM uses a slightly modified version of Concurrent Task Trees (CTTs) ([Pat00] for CTTs, [TC03] for WSDM modifications to CTTs). Subsequently, for each elementary task a so-called object chunk is created, which describes exactly what information and/or functionality is required to perform this elementary task. WSDM uses OWL (see e.g. [CPT06]) to formally describe these object chunks.

During navigation design, the (conceptual) navigation structure is modeled in an implementation-independent way. It indicates the general organization structure of the Web application, i.e., how the different visitors will be able to navigate through the site. In WSDM, the basic navigation structure is based on the audience class hierarchy: for each audience class, a navigation track is constructed. Such a navigation track can be considered as a sub-site, containing all and only information and functionality needed for this audience class. The internal navigation structure within a navigation track is based on the (temporal relations in the) task models. The navigation model consists of three basic modeling elements: conceptual navigation nodes indicating units of navigation, links between these navigation nodes, and the object chunks which are connected to the navigation nodes.

Implementation design

During the implementation design, the conceptual models are complemented with all necessary details to prepare for the actual implementation, which can be generated automatically. In the first subphase, the site structure design, the conceptual navigation structure is mapped onto actual pages. Several site structures are possible, depending on device, context, and platform (e.g., different screen sizes may give rise to different site structures). During the presentation design, the general look and feel for the Web site is defined. For the different kinds of pages (e.g., home page, leaf page) templates may be defined that will serve as a base when designing the actual pages. During page design, the designer decides for each actual page the concrete interface elements (e.g., a dropdown list or radio buttons to represent a single-choice list), how to position these elements and the information and functionality described in

the object chunks, and the general look and feel of the page. This results in so-called page models. In the case of a data-intensive Web site, a database or CMS (Content Management System) can be used to store the data. In this case, the actual data source, and a mapping between the conceptual data model (i.e., the object chunks) and the data source, are specified during the *(logical) data design* subphase.

Implementation

Given the relevant (instantiated) design models (i.e., the object chunks, navigation model, site structure model, page models), and, in the case of a data-intensive Web site, the data source and the (logical) data design, the actual Web site can be generated automatically. Literature describes two prototype implementation performing this code generation process: an XSLT-based transformation pipeline [PCY⁺05] and a Java-based servlet [Cas05]. For more information on implementation of Web applications in the context of Web site design methods, see Section 7.2.2.

3.4.3 The OOHDM Model

Object-Oriented Hypermedia Design Method (OOHDM) [SR98] is one of the first methods adopted for Web application development projects. It has its roots in the hypermedia domain and focuses on helping the development of applications that involve hypertext/hypermedia paradigm features explore distributed heterogeneous information. The OOHDM method features object-oriented abstractions for analysis and design of information-intensive Web applications. Besides the modeling abstractions, and similarly to WSDM and WebML, it also provides a methodology which guides a developer through different activities in the Web application development. The main features of OOHDM are [SR98]:

- *Navigation views*: OOHDM adopts a notion of navigation views for specifying how information objects should be grouped when explored by a user in a navigation session.
- *Navigation contexts*: OOHDM proposes navigation contexts as grouping abstractions to organize the navigation space.
- *Separation of concerns*: OOHDM features separation of concerns. The domain conceptual issues are separated from navigation issues and both of them are separated from presentation issues. Query language is used to connect models from different viewpoints.

Figure 3.7 depicts the OOHDM phases together with the design models that result from them. Here we briefly describe these phases. We will concentrate on the details of some of the models and modeling techniques provided by OOHDM later, in Chapter 5.

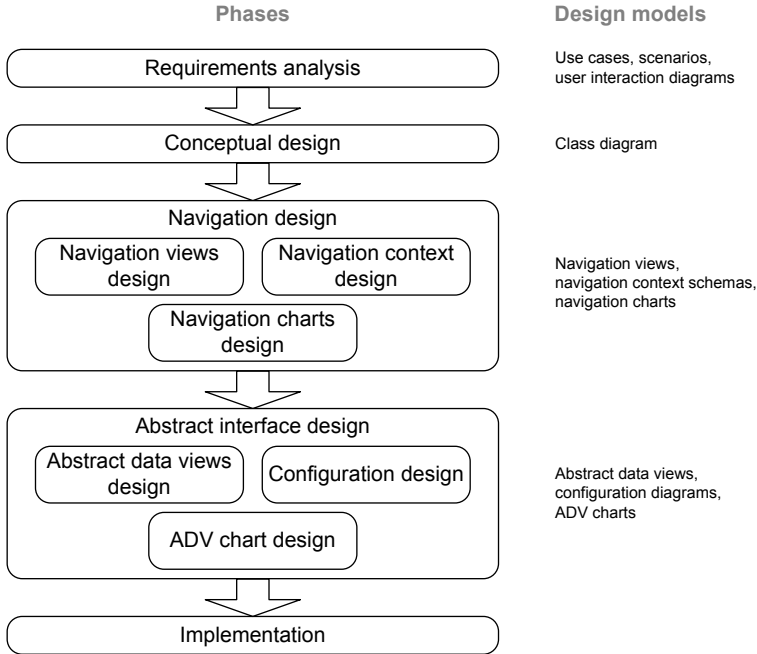


Fig. 3.7. Overview of the OOHD design method

Requirements analysis

The primary goal of this phase in OOHDM is to capture and understand functional and nonfunctional requirements of the Web application. The requirements analysis, sometimes also called requirements capture, is use case driven. This means that the functional requirements are elicited with a help of use cases, actors, and stakeholders of a Web application. The use cases are further refined to scenarios which reflect use tasks. OOHDM features so called user interaction diagrams [VSdS00], which capture how a user should interact with the application when fulfilling certain use cases.

Conceptual design

OOHDM conceptual design is concerned with the design of information structures for representing the content provided in Web application. Well-known object-oriented principles are applied during this phase. The result is a class diagram extended with special constructs to attribute multiple values and perspectives. This feature is especially important for multi-modal Web applications and Web applications with semi structured content. The classes with relationships can be grouped into subsystems. Conceptual design is separated from other activities and deals only with application domain classes without a connection to any further application solution for viewing and organizing the content.

Navigation design

OOHDM navigation design is concerned with navigation structures supporting a user exploring information provided in a Web application. Cognitive issues are taken into account to reduce the information overload and to support the user in getting oriented in the information hyperspace. The navigation design produces views of the information structures, which can be different for different audiences. This is reflected in navigation context schemas where common views are grouped under one context. Navigation design may be also extended with behavioral specifications through navigation charts specifying some reactive behavior. Navigation models are connected to conceptual models. They use underlying concepts from the conceptual models to derive right perspective on information structure either by restricting it, projecting it, or transforming and combining it with other concepts. Different navigation views and schemas can be built for different purposes or users from a single conceptual model.

Abstract interface design

Abstract interface design follows object-oriented design principles and focuses on perceivable objects defining how navigation views should be displayed and augmented with further interaction elements, such as buttons and links. Abstract data view charts may be used to specify the behavior of presentation objects. The abstract data views follow the same principle as the navigation views. They can be seen as façade abstractions, representing different appearances of the navigation nodes to different users in different contexts. They feature also a behavioral and interactive aspect and are therefore very suitable for describing also modern interactive Web applications.

Implementation

OOHDM does not use any specific implementation framework. It is up to the development team to decide how to transform the results of the aforementioned phases into implementation. The development team makes a decision on architecture such as client-server, database management system to store information structures and data, application and web server to compute navigation and presentation views and handle user interaction events. Refer to [JSR02] for details on mapping to an architecture based on J2EE and the Model-View-Controller model.

3.5 Summary

Summing up the lessons learned in this chapter and the considerations that led to the definition of the online evolution model, we can say that Web-specific development processes in general distinguish themselves from traditional software development processes because of the following general characteristics:

- *Continuous* and *fast* development and release times are paramount.
- Web development processes are less documentation-based and, rather, put high emphasis on *prototypes* (prototypes are much more expressive than technical documents, especially to unskilled customers).
- High *user involvement* and early feedback is desirable.
- A new actor enters the development process: the *graphic designer*.

If we look at the activities in the development process, we can also identify the following activity-specific characteristics:

- The *requirements analysis*, *design*, and *implementation* activities can be further detailed into typical Web-specific subactivities. This allows for the conception of specific processes, instruments, models, and tools, assisting developers and lowering risks.
- The *implementation* activity is highly standards-based. This contributes to the fast adaptation of developers to new projects, to higher interoperability of the conceived solutions, and to elevated robustness.
- The *deployment* of Web applications is typically fast. There is no need for client-side code that requires manual installation procedures, and, hence, there is no need for complicated installation and deployment processes. Installation and deployment come almost for free, and consistency among clients is implicitly guaranteed.
- The continuous (online) *evolution* of Web applications is an integral part of the development process. The development cycle continues even after the deployment of the application. This may be indispensable if we want to keep the attractiveness of the Web application high and enlarge the application's audience.

The organization of the following chapters is based on the online evolution model depicted in Figure 3.3.

3.6 Further Readings

Web engineering processes have been described from several points of view in a number of publications. The development of hypermedia-oriented Web application was discussed in [NN95, NN99]. They concentrate mostly on the design process, which is characterized as a motion in a four-dimensional space of guidelines for *hypermedia application*, *hypermedia design and development*, *hypermedia system*, and *human factors*. The readings are recommended as a general introduction to Web process guidelines frameworks. Similarly, [DB02b] concentrates on the design organization framework as a five-dimensional space of hypermedia application, notation, development process, aspect, and degree of formality. The design process is then considered as a chain of instances over the dimensions.

Another Web application development model, inspired by Software Engineering Institute's Capability Maturity Model (CMM) framework [PCCW93] and the SPICE architecture of process assessment [EG96], the IMPACT-A method [LBW99], is based on a three-dimensional space formed by following dimensions: *process entities*, *hypermedia entities*, and *time*. The process dimension is characterized by entities such as **Resource**, **Activity**, and **Artifact**. The **Resource** can be a **tool**, a **skill**, or a **person**. Hypermedia high-level entities are **structure**, **navigation**, **behavior**, and **interaction**. The method contains two high-level phases: preparatory phase and execution phase. The preparatory phase is concerned with choosing a model, the quality attributes for assessment, the measuring methods for those attributes. The execution phase is carried out for each development project. The model serves as a guide for developers to understand development. Attributes assist developers identify particular aspects that are important for assessment. Tasks are guides for attribute assessment.

The Hypertext Design Model HDM [GP93], W2000 [BGP01], the Relationship Management Methodology (RMM) [ISB95], UML-based Web Engineering (UWE) [HK00], and Scenario-Based Object-Oriented Hypermedia Design Methodology (SOHDM) [LLY99] are examples of other development methods with slightly different views on the organization of activities and processes.

Proceedings from the World Wide Web conference and the Web engineering conference, collections such as [MD01, KPRR03], and books such as [DLWZ03, PJC98, Pre05, CFB⁺02, Con00] are sources for further valuable insights into Web application development processes.



<http://www.springer.com/978-3-540-92200-1>

Engineering Web Applications

Casteleyn, S.; Daniel, F.; Dolog, P.; Matera, M.

2009, XIII, 349 p. 109 illus., Hardcover

ISBN: 978-3-540-92200-1