Journal of Web Engineering, Vol. 3, No. 2 (2004) 077–094

e-PROTOTYPING: ITERATIVE ANALYSIS OF WEB USER REQUIREMENTS

WOLF-GIDEON BLEEK, MARTTI JEENICKE, RALF KLISCHEWSKI
Software Engineering Group, Department for Informatics, University of Hamburg
Vogt-Kölln-Str. 30, D-22527 Hamburg, Germany, +49 40 42883-2307
{*bleek, jeenicke, klischewski*}*@informatik.uni-hamburg.de*

Received August 2, 2004
Revised September 2, 2004

Projects developing Web applications face problems when it comes to identifying the Web users' requirements. There are a number of reasons for this. It is unclear how to gather initial requirements from potential users if there is no design artifact to communicate about. Developers have difficulty identifying the needs of the Web application users during the ongoing development process because of a lack of proper communication concepts. Development teams for Web-based systems include professionals from different disciplines with diverse cultures. Members of the development team often belong to many different organizations with varying stakes in the project.

This article presents a modified prototyping approach called *e-Prototyping*. This approach includes frequent releases of software versions (based on short development cycles) as well as integrated mechanisms for gathering feedback from users and other relevant actors via the live system. It underlines the need to offer various communication channels to the users and to systematically order the different streams of feedback to enable the developers to identify the user requirements. *e-Prototyping* encompasses the management of an agile software development process and the systematic evaluation of manifold feedback contributions.

Keywords: Web Engineering, prototyping, evolutionary software development, participation

Communicated by: M Gaedke & D Lowe

1 Introduction

To gather sound requirements for applications, developers must systematically integrate users in the development process. Such user-centered software development is a well-known approach in traditional Software Engineering, called participatory design [1], [2]. A problem arising in projects concerned with the development of Web applications (in short: Web Engineering projects) is that the users only learn how to formulate their requirements very late on in the process, i.e. after the design artifacts have been created. We agree with the findings of Lowe and Eklund, who describe the development process as a learning process among the stakeholders [3, sec. 2.2]. Giving users the opportunity to formulate their requirements enables them to understand and develop their own functionality needs [4].

Yet a well-founded understanding of the users' needs is a key factor in the successful development of Web-based systems [5]. This is especially true of voluntarily used systems, e.g. e-government systems like municipal Web sites, which are usually just additional channels alongside regular citizen services or community systems that depend on the willingness of people to engage in exchange with

others. This unique feature of Web information systems contrasts with traditional information systems where users normally have no choice whether to use the system or not (because their employer demands that they use a specific system at work). Thus the barriers for not using an application are much lower in Web systems that poorly address user needs. In addition, development teams for Web based systems include professionals from different disciplines [6]. Often, these persons belong not to one but to many different organizations (e.g. organizations specializing in software development, screen design or content delivery/production) and – depending on their background – they have a different stake in the system under development.

However, stakeholders and their requirements need to be carefully identified. Many Software Engineering approaches use prototypes as learning vehicles to establish communication and build a common understanding of an application domain [1]. Web Engineering is considered different from traditional Software Engineering [6, 7]. A number of questions arise, then, when trying to adapt a prototyping, participatory, or user-driven approach to the Web. These include how prototypes for Web users can be built, deployed and presented in order to support the learning process of all project stakeholders, how user feedback can be collected and distilled to identify user requirements and what will happen to prototyping approaches when they are shifted to the field of Web Engineering.

In this article, we show how the difficulties in gathering sound requirements for a Web application can be overcome by implementing a Web Engineering process that combines a prototyping approach with a well-established evolutionary and participatory Software Engineering development model. First, we look at traditional Software Engineering techniques that address the problem of identifying user requirements in the setting of closed software development projects. We discuss why current Web Engineering methods do not explicitly help to identify user requirements on the Web. Second, we introduce prototyping as an established tool and process for discussing the implementation of applications with the users. The findings presented here are based on experience gained in a number of Web Engineering projects we were involved in. Third, two of these case studies are used to substantiate our claims. The challenges and problems encountered in the project work are systematically addressed to draw on our own approach. Fourth, this is followed by the description of *e-Prototyping*. We then show how our approach helps overcome the problems faced in identifying the Web users' requirements.

2 State-of-the-Art Process Models for Software Engineering

Over the past decades, extensive work has been done on the development of process models for Software Engineering. There are a number of approaches that claim to provide the best support for developing many different kinds of software. Since our focus is on application-oriented software development, we look at existing methods specifically addressing this kind of development [8, 9] which we think can best be extended for Web Engineering. Evolutionary process models have proved to be suitable because they help take into account use context. We discuss the key features of one such method in Section 2.1. Agile process models, which have emerged in recent years, offer faster evaluation of development activities. These models partly match the evolutionary approaches we favor and help improve the feedback cycle. Section 2.2 briefly outlines the rationale behind these models. To summarize, in Section 2.3 we point to the drawbacks of existing Web Engineering approaches by referring to features that need to be supported in application oriented software development.

2.1 Application-Oriented Software Development

Many application-oriented software development methods can be classified as the so-called Scandinavian or participatory view of Software Engineering. One of these methods is the STEPS (Software Technology for Evolutionary Participative System development) approach [10], which supports the "cooperative development of software with users" (p. 49). The underlying rationale is that all participants gain a common understanding of the development process and its goals and are involved in decision-making processes throughout development. It thus emphasizes evolution as well as collaboration among all participants in the development process.

STEPS is a cyclic process model. The development of a system usually includes several iterations, which consist of the tasks *project establishment* (after the first iteration: *project revision establishment*), *production* resulting in a *system version*, and its *application*:

- Project (Revision) Establishment: This activity defines the problems addressed by the system and its functionality. It involves all project actors and is a participatory activity between users and developers.
- Production: The first activity in this task is system design, which leads to the *system specification*. Again, this is a participatory task involving users and developers. Following system specification, the next production step is divided into the users' task *embedment preparation* (i.e. installing the required system software, educating users) and a developers' activity: software *implementation* (building the software).
- System Version: This is the system resulting from the production task. After being deployed, it will be available for use.
- Application: This task can be divided into system *usage* by the users and system *maintenance*. Unlike the prevalent understanding of maintenance, it is limited to bug fixing. It does not include the development of new or the changing of existing functionality.

After the first iteration of the STEPS cycle, the actors must work on a task called *revision establishment*, in which the functionality of the next system version is defined. Like project establishment, it is again a participatory task. The process then continues with the next cycle. STEPS promotes the use of prototyping as a technique for building design artifacts that help in the discussion and learning process of both users and developers [4].

2.2 Agile Process Models

Agile software development methods are popular because of their quick results and short planning timeframe. Usually, three key features are associated with agile development methods: first, they promote short development cycles (max. 3 months); second, they reduce administrative activities to a minimum; third, they allow the change of the process during its application. The terms "adaptive software development" [11], "agile processes" and "agile software development" [12, 13] are used to discuss software development processes that incorporate these features.

The coining of the agile paradigm was motivated by constantly growing software processes that were producing vast quantities of documents during development. Changes in development produced an enormous number of documents, thus reducing development speed. Developers, users and contractors were frustrated and called for quicker results. In agile development, everything is geared to

ensuring a lightweight process and flexibility. The aim of such agile processes is to reduce the documentation produced during software development. Each document must be justified, and the creation and maintenance of by-products is minimized if the risk of doing so is limited.

The lightweight character of the processes underlines the fact that software development is the main focus, while process control and maintenance are additional tasks performed with as little effort as possible. Flexibility makes it possible to discuss and change each steering element of the process and to question its adequacy in the current situation. Moreover, the term "agile" underlines the flexibility to change the process at any time. Quick results foster discussion about the development outcome among all participants in the development process. They allow a realistic assessment of the development's progress as well as its direction.

These characteristics, especially the ability to quickly react to new requirements, which is needed in Web projects, would appear to justify the use of agile methods for Web Engineering (e.g. as proposed by [14]). However, there is currently no agile software development method explicitly tailored to participatory developing Web applications. We will therefore go on to show which features of agile processes make sense in the context of software development for Web applications and have been integrated into the *e-Prototyping* approach.

2.3 Process Models for Web Development

Many development projects have recognized that developing a Web application is quite different from developing a classic computer application. Some characteristics are obviously different, like the presentation of large amounts of content, the focus on the visual design of an application, and the numbers of standard technologies involved. However, these differences are easy to see. There are – depending on the type of project – more and other differences. Since the introduction of the Web, a number of development methods designed for the special needs and problems of Web-based systems have been developed. The field of Web Engineering has emerged.

According to Murugesan et al., Web Engineering is multidisciplinary and includes contributions from the areas of hypertext, information engineering, requirements engineering, system analysis and design, modeling and simulation, project management, testing, human-computer-interaction, multimedia, and Software Engineering [15]. This is reflected by the backgrounds of the different approaches to Web Engineering. Since we look at Web Engineering from an application-oriented Software Engineering perspective, we investigate existing methods that incorporate requirements engineering, software development and project management. Web Engineering methods that do not address these fields may well be suitable for some (parts) of the projects, but we are not concerned with them here. For instance, there are a significant number of methods that focus on hypertext structure for presenting non-linear information. Some examples are HDM or RMM. Other approaches, like WebML [16], focus on data-intensive applications. Since the WebML process model is based on a kind of waterfall process, it has to cope with the widely discussed disadvantages of such a process – especially with regard to the integration of users and the lack of support for evolution. It is therefore not discussed here.

Instead, we focus on application-oriented approaches associated with Web Engineering, e.g. the W Life Cycle model, OOHDM, WSDM and WebOPEN. We discuss below some of the characteristics of these approaches.

Sherrell and Chen discuss integrating prototyping with their W Life Cycle model [17]. They also seek to include users' contributions in the development process, stimulated by the use of a production

system. However, they assume a determinable and available user group, encountered only in certain types of Web projects such as intranet development within fairly small organizations. Such special settings and types of groups make the gathering of requirements much easier than with heterogeneous user groups that are distributed throughout the Web.

OOHDM [18] promotes a number of techniques for building Web applications. However, it does not describe in detail how the process of requirements gathering, and the use of prototypes for this task, is to be managed. OOHDM focuses mainly on user involvement in the design of an application's interface rather than its functionality (usability vs. functionality). Another limitation of this approach is that it assumes the use of an object-oriented programming or scripting language for the development of a Web-based System. Yet, a large number of well-known Web sites are based on commercial off-the-shelf (COTS) components embedded in a larger and complex IT infrastructure. Web information systems in particular make heavy use of content management systems, databases, application servers, and ad-servers. Examples are municipal Web sites, portals or community systems. The OOHDM approach does not specifically address the integration and characteristics of these typical components.

WSDM is a user-centered approach for the design of kiosk Web sites [19]. Unfortunately, like OOHDM, it is not user-driven or designed to let users participate in the design process (at least with regard to functionality), which hampers the learning process. In addition, it assumes that the user groups of Web sites can be easily classified and described. From our experience, documented in Section 4, this does not seem to be a realistic perception of all Web-based systems. Moreover, De Troyer and Leune fail to mention how iterations of the design process can be implemented. The WSDM approach can be classified as a big-bang approach.

WSDM and OOHDM focus on describing the structure and functionality of Web applications with sound formalisms. While this is an appropriate means for communication between developers, it cannot be extended to communication with the everyday Web users. In a way, it is like speaking different languages (the technical language of the developers vs. the colloquial language of the users). Thus, the two methods do not support communication between the different actors involved in the development process. However, they are an important prerequisite for the learning process and the gathering of user requirements.

In contrast to the methods mentioned so far, WebOPEN [20] is an adaptation of the widely accepted process model (OPEN), which is a full life cycle model for developing object-oriented and component-based software [21]. In the WebOPEN process, the system requirements are the result of a negotiation process. A role is played here by prototypes (in the form of white sites), which are used for analyzing and communicating about a Web site's architecture. Even though Lowe and Henderson-Sellers point out that the requirements of a Web project are extremely volatile, they fail to address this problem by extending the negotiation phase to cover the entire life span of the Web site. A problem here in terms of gathering Web users' requirements is the fact that WebOPEN focuses on usage-centered rather than user-centered or even participatory design.

In our view, what is missing are approaches to Web Engineering that focus on the participation of relevant project actors. This includes the question of identifying such actors as well as requirements gathering together with the identified actors.

To provide a suitable method for the development of Web-based systems, we have chosen an application-oriented approach originating in the field of user-centered Software Engineering. We favor a cyclic process that allows feedback to be continuously incorporated into the development activities. This can be achieved using the STEPS process model (see below).

Prototyping is a way of establishing communication between developers and users. Combining the cyclic process with the characteristics of agile methods reduces the turnaround time for each cycle. Thus many development activities need to become shorter, making prototyping more important. As a result, prototyping structures all the other tasks in the development process. To use prototyping as a core technique, we need to clarify its characteristics. The next section gives two different perspectives on prototyping by comparing them on the process level as well as in terms of the elements involved.

3 Prototyping and Application-Oriented Software Engineering

Prototyping has become a well-accepted technique in Software Engineering [22, 23, 24, 25, 26, 9]. Sommerville [9, pp. 138-153] describes it as a means of requirements analysis and validation. Prototypes support communication between developers and users by enabling them to "experiment with requirements". He distinguishes two types of prototyping: evolutionary prototyping, which aims to have a complete system when all development effort has ended, and "throwaway" prototyping.

Evolutionary prototyping is an iterative process in which the requirements that are understood are implemented first. Once they have been implemented, the developers move on to those requirements that are still unclear. "The key to success in this approach is to use techniques which allow for rapid system iterations" (p. 142). Our argumentation is based on evolutionary prototyping (see below), which is also in line with European/Scandinavian ideas of participatory design. Prototypes are an important type of artifact and a source of insight in a continuous learning process. We therefore believe that "throwaway" prototypes are inappropriately named, even though the name may be apt in some cases, given how they are treated in many projects, e.g. after (some) requirements have been clarified in order to write the specification of the "real system". According to Sommerville, the process of prototyping in software development consists of four steps ("establish prototype objectives", "define prototype functionality", "develop prototype" and "evaluate prototype").

Floyd [26] takes a similar view of the process – her process model differs from Sommerville's in the activities at the beginning and the end. Sommerville's process structure starts with an explicit step in which the prototyping objectives are established. While both processes end with an evaluation, Floyd favors a step in which a decision on the further use of the prototype is made. In our view, the latter is important for a prototyping approach to software development that promotes learning across the heterogeneous group of participants. We therefore follow Floyd's view and describe the steps of her model in greater detail:

- Functional Selection: Part of functional selection is the decision as to which and/or how many functions the prototype should include. There are two common strategies: either the prototype includes a wide variety of functions, which are not implemented in detail (horizontal prototyping), or the prototype has only a few functions that are implemented in quite close detail (vertical prototyping). Moreover, prototypes can be classified according to their goals and target groups [27]. The different types include presentation, functional, and user interface prototypes as well as pilot systems. As in real-life projects, combinations of different types of prototypes can be observed in Web Engineering projects. Analysts have often found that the actors in a Software Engineering project use a certain form of prototyping without necessarily being aware of it or of the methodological issues involved [27].

- Construction: According to Floyd, this step comprises all efforts to make the prototype available. Since the prototype is a learning vehicle for all participants in the development process,

only a few quality requirements of the future system are considered important during these activities.

- Evaluation: The principal goal of the prototype construction should be subsequently evaluated. This step is crucial to the prototyping process because it covers the process of collecting information and experience that are valuable for improving the developed product. It should therefore be carefully planned and performed.

- Further use: Depending on the experience gained, there are generally two possible ways in which the prototype can be further used. One is to use it solely as a learning vehicle. Many authors call such prototypes as "throwaway" prototypes. The other is to use the complete prototype or parts of it in the target system.

Floyd distinguishes three types of prototyping by categorizing the goals the developers seek to achieve [26]. Exploratory prototyping is used if a problem is not yet clearly formulated. The requirements of management and users are explored by means of prototyping. This way, the developers learn more about the target domain and the tasks the users have to perform. Experimental prototyping is used to clarify the technical implementation of the requirements: the developers test their ideas to get an impression of the technical viability and the suitability of the application system. Evolutionary prototyping is powerful but farthest removed from the original meaning of prototyping. It is an incremental method for evolutionary software development in which the application is expanded in "small" steps (evolution). Developers who use this method can customize the application system quickly to adapt it to changing conditions in the application domain.

Irrespective of their original goals and target groups, the resulting prototypes have one aspect in common: they provide a valuable platform for discussing options, limitations and expectations as well as domain-specific and technical questions. When considering an existing artifact, many effects surface and become an explicit part of the discussion between people with different qualifications (developers and users). This supports the learning process among all project participants and helps to sort out misunderstandings at an early stage in the software development process. Over the years, experience has shown that prototyping leads to better quality, and an overall reduction in misunderstandings, thus establishing its acceptance as part of the methodology of application-oriented software development. The established integration of prototypes in software development and as part of Software Engineering practice is now being challenged by the conditions encountered by developers in Web projects.

We will go on to show how prototyping and the cyclic model can be married to a process called *e-Prototyping*. We have been developing this approach since 1999 to address new challenges and problems that we have encountered in a number of Web projects.

4 Problems of Iterative Web Development – An Empirical Account

The findings presented in this article are based on experience gained in a number of software development projects for Web applications in which we actively participated or for which we worked as consultants. We outline two of these case studies in the following sections: the development of the municipal Web site *hamburg.de*, and the development of a Web-based community system called *CommSy*. The challenges and problems encountered in these projects are systematically addressed in Section 4.3 to help develop our own approach.

Both cases are presented as qualitative case studies. They were conducted by members of each development team and are based on our own personal experiences as well as secondary sources. The data collected in both studies is documented in the form of personal diaries and archives of all communication media involved. In addition, we refer to a number of earlier studies that had already analyzed other aspects of the projects. The findings of these studies were critically evaluated in semi-structured interviews with selected participants. The software development process was actively reviewed during the development period of each project, and identified problems and obstacles were addressed by repeatedly tuning the process to the changing conditions.

4.1 The Municipal Web Site hamburg.de

In 1999, the city of Hamburg decided to operate its official Web site by creating a public-private partnership to foster innovation ([28], `www.hamburg.de`). The business model of the company set up to run the Web site involved hiring well-known firms to do the actual development work while concentrating on developing a vision for the future concepts, a business model, and taking care of administrative tasks. One obligation of the company in charge of the job was to implement freely available services for citizens, including free e-mail, Web mail, private homepages, etc.

To coordinate the development process, weekly meetings were held for all the firms involved in the development. These included representatives from the software development company, the screen designers, the manufacturer of the content-management system, the producer of the shopping-mall system, the database vendor, a hardware expert and consultants. Although the development process seemed well coordinated, crucial people were missing from the coordinative meetings and major design decisions were made outside the weekly meetings. Not all design decisions were, then, documented for those who attended the coordinative meetings, and most importantly, the development continued for more than a year without producing a single relevant piece of software. The development of the municipal Web site could therefore be characterized as a "big bang" approach, i.e. taking $1^1/_4$ years without leading to a single release.

The first development approach failed owing to a variety of problems (acquisition of technology, performance, interconnections and interoperability with other systems, misunderstandings, conflicts of interest). However, the company decided to continue developing the Web site using a new approach. Given a second chance, the project was eventually saved only by dramatic changes: short development cycles including a completely new development in a small team and a consistent functionality visible at a glance, making possible a short "time to release/time to market". The visible result was a first public release after approximately eight weeks. After that, minor releases were scheduled at frequent intervals. The planning of new releases was, however, driven completely by management decisions and was not communicated outside the project.

The lack of communication channels for the users and the customer of the system was a significant problem throughout the whole development of the municipal Web site. Representatives of the municipality were not regularly involved in the decision making process. This resulted in misunderstandings and then in failure to meet their requirements. Moreover, instead of users being invited actively to participate in the development, the only way for them to express their needs was through a Web-based guest book – which was not initially intended for this purpose.

To summarize, the development process of the *hamburg.de* project began with a long period devoted to realizing the vision of a complete system. To gather the requirements, weekly meetings were held. While many stakeholders were invited to these meetings, other important actors were not. The

system design was unsuccessful owing to the "big bang" approach, which produced an unmanageable number of nearly finished components and an unclear system structure. In the second phase of the project, small intervals were established to develop the system. After an initial interval, new releases were produced in short cycles. However, attempts to synchronize the development process by weekly meeting were abandoned, as was user involvement.

While the weekly meetings with all stakeholders proved to be a good communicational platform, important representatives were missing. Among those not attending the weekly meetings were representatives from the city's government departments, user representatives and other stakeholders from NGOs or non-profit organizations. The process failed to address the integration of stakeholders during the ongoing development. The underlying problem may have been the target-group specification. While all citizens of the municipality were members of the target group, this broad definition was of only limited help in answering questions about requirements.

An overall problem of this development setting was the relatively large number of organizations involved in the development process. Nearly every product was accompanied by a representative, development was delegated to yet another firm, and the project itself employed a number of consultants. All those involved acted in not only the interest of the project, but also tried to pursue their own interests.

In the second phase of development, the advantages gained by the short development intervals were nullified by the fact that hardly any relevant stakeholders were integrated into the process. The only communication channels available once the system was online were an electronic Web-based guest book and a call center. The development team managed a list of necessary error corrections and requests from contractors, which they received by e-mail. Despite these measures there was no systematic gathering of use scenarios or feedback.

In both phases, the initial requirements were hard to specify. Developers drew on their own experience and observations. Some requirements were easy to meet and proved useful; other requirements were unrealistic and hindered the project's progress.

4.2 The Community System CommSy

CommSy stands for community system and is a Web-based system designed to support communication and coordination in working groups ([29], `www.commsy.org`). *CommSy* has been developed and tested in various educational settings in the Department for Informatics, University of Hamburg, since May 1999. The development of the system was initiated by a working group of research assistants and student volunteers. A subset of this group developed the initial version of the software by repeatedly formulating requirements, implementing them and reflecting on the handling of the implementation. After a few weeks, when the systems's stability, usability and functional coverage had reached a certain level, the researchers decided to use the system in their teaching. In various teaching projects, students volunteered to further develop the system to meet emerging needs. A year later, a large group of people were involved in developing and deploying the system.

Users and developers quickly learned to use the system as a medium for providing feedback during the development process. This was necessary because direct communication among the developers and users was impossible, given the many different areas in which the system was deployed. The development team installed electronic discussion forums, which were used extensively for technical and professional feedback (some users also expressed the desire for workshops). Motivated and active users organized themselves and exerted influence on the development process. Problems were

encountered with the evaluation of the different feedback channels. Users either posted feedback in one of the discussion forums or sent it directly to the developers, which made it difficult to keep track of users' wishes and ideas (different e-mail addresses and up to ten forums).

After the community system had been deployed several times in large and small projects, the software was further developed to make it available on a national student portal. This was done because demand for the system was growing, while at the same time the volunteers were finding it increasingly difficult to handle user support and communication. The developers' hope was that the national portal's feedback channels, which were already established, could be used to direct user communication. However, the users did not stick to the channels provided and the portal provider failed to adopt a systematic approach. This resulted in even greater confusion among users and frustration on the part of the developers.

One difficulty was the unidirectional communication of feedback: volunteers' feedback went unanswered, so they had no idea whether their remarks about the system had been taken note of and had led to changes or not. Furthermore, it was not known when and in which future version any changes implemented by the developers would be perceptible to users. These open questions were systematically addressed in the next development phase of the system.

The next step in the development of the community system was a research project at the local university department, which allowed systematic development of the software, the provision of services and the establishment of an organizational framework. The provision of a complete range of services in an integrated project helped alleviate the main problems. However, the prospect of having more manpower available for development resulted in longer development intervals, which in turn led to frustration on the part of users, who were waiting for improvements.

A recurrent problem in the development process is communication with a system's users. Each project that uses the system creates a new context that must communicate a consistent picture of the system. Moreover, different contexts demand that different – and to some extend contradictory – requirements be met. It is therefore not easy to gather requirements systematically and weigh them according to the needs of the target group. With changing contexts, another challenge is integrating new relevant actors from different organizations and handling their different interests.

4.3 Observed Problems

In their survey, Lowe and Eklund state that requirements "emerge in commercial specifications after design artifacts have appeared" [3, p. 10]. According to their findings, "commercial practice can be described as a form of design-driven requirements elicitation" [3, p. 12]. Our experience was similar in the two outlined projects. Our evaluation of the project examples showed that the same questions arose repeatedly and pointed to substantial problems, e.g.

- How can the (initial) requirements for Web-based applications be defined?
- How can the requirements be gathered systematically, if the target system (Web) users are unknown and their characteristics are difficult to describe?
- What techniques are useful to promote communication between developers and users in order to gain a common understanding of the users' requirements?
- Which actors should participate in the process and how?

We believe that these problems arise in many other Web projects as well. But here our focus is public corporate Web sites, which includes Web portal construction [17, pp. 5]. We do not discuss intranets and extranets, in which developers can relate to a well-defined group of users and other actors. Most of our findings also apply to electronic commerce systems (including classical shop and auction systems), though we do not focus on the specifics of workflow and transaction management.

Requirements in traditional software projects mostly relate to milestones that structure the overall software process [9, p. 51-57]. In this kind of development process, prototypes are built in order to gain new insights and support decision-making where applicable. They are embedded in the iterations of requirement analysis, prototyping, implementation, product release and revision [22]. Web Engineering projects do not always enjoy this kind of freedom:

- Any software released for use on the Web is unprotected: publicly accessible Web prototypes are always exposed to public criticism – no more "playing around" with a development system.
- Each feedback round with users requires time for preparation, presentation, communication and evaluation. But strong market pressure and high expectations do not normally leave Web projects the time for this.
- Web users expect new versions regularly, especially when waiting for requested functionality.
- This leads to considerably shorter development cycles, and consequently to pressure on the developers to define work packages for shorter time periods.
- What Web-based applications have in common is that they are "early adopters" [30] in their domain, i.e. they offer a new service on the Internet. Developers must bear in mind that the application is expected to be innovative and of high quality.

"Traditional" prototyping contributes to software requirements definition through cooperation between developers and (future) users, most of whom work for the organization that has ordered or owns the future product. There are usually different perspectives and interests involved (e.g. management vs. user), but the relevant actors are identifiable [31]. In Web projects, which go beyond organizational borders, requirements gathering takes a quite different form:

- Initial requirements are defined by the providers' view of a potential application (the wishes and demands of the current user group become evident only through the first running version of the system).
- The user group is not well defined – unlike in companies, where users can be characterized by their jobs or functions, Web users may be structured in a large number of groups (at least for the types of Web-based systems considered in this article) or there may be no identifiable structure at all because of the users' heterogeneity.
- The relevant actors cannot be represented in terms of a simple actor model (including developers, users and management) – actors contributing to the system development take on new roles such as, "technology champion" [32], (sub)service provider, etc.
- Actors with different perspectives and interests do not normally belong to the same organization, which precludes direct and personal discussions (users or decision makers cannot easily be

invited to or are not available for individual or group interviews) or even simple user observation (owing to the physical distance).

With more groups of actors involved, recognizing and acknowledging the different perspectives becomes a crucial task for requirements gathering within projects. Of course, we can also identify well-known roles, such as contractor (e.g. the funder of the project), user, developer and customer, but there are significant shifts of interest:

- Contractors, at least in principle, expect a return on their investment, but Web projects are often not accountable in terms of rationalization. Their implementation is more likely to result in an image improvement, an increase in market potential or an expansion of the service portfolio, and in many cases project investments are "strategic".
- With a large and ill-defined group of users, Web projects need to acknowledge a multitude of different user perspectives. Special roles can/should be identified, e.g. major complainers who regularly criticize errors or missing functions, volunteers trying to play an active role in the further development of the Web application by spending a lot of time on its evaluation and making helpful suggestions for improvements that can be directly implemented.
- The developers' interests differ significantly from other perspectives. Their activity is directed by the need to keep the software error-free, to make the latest back-end technology work and to implement state-of-the-art features. Their perspective is limited to that of the "power user". On the other hand, there are project restrictions due to technical conditions and the limits set by other actors. The developers' job is to integrate a system into a given environment with an existing or predefined technology and make it run reliably.

Integrating the relevant perspectives into the development process in a Web project includes new challenges. These comprise acknowledging not only new perspectives but also new channels and media for voicing these perspectives. Web projects provide a situation in which the users themselves foster communication within the medium (the Internet), e.g. volunteers moderate forums and organize user groups.

Web projects face new operating conditions, which become visible step by step because the application is already in a productive mode with reactions from "real" users. Thus requirements gathering in Web projects cannot make use of "traditional" prototyping because many of its assumptions no longer hold. However, there are many relevant actors, and their perspectives should be included to help save expense and foster innovation.

In the next section, we discuss how a new *e-Prototyping* approach could support the gathering of Web user requirements by adopting an evolutionary approach based on short development cycles. We show how to organize and maintain user feedback through active communication channels that support the development team with the valuable information needed.

5 Towards a Development Process Model: How To Do *e-Prototyping*

In this section, we describe our approach to *e-Prototyping*. First, we argue that the current trend in Software Engineering toward shorter development cycles lead to a dovetailing of prototyping with release management. Second, we explain the steps of our *e-Prototyping* approach in relation to "traditional" prototyping activities. Third, we show how obstacles in user-developer relations can be overcome by fostering communication and integrating it into the development process.

Shortening development cycles seems to be an issue in various fields of Software Engineering. An example of a new method that propagates shorter development cycles is Extreme Programming [33]. Beck calls for shorter cycles at all levels of Software Engineering in order to increase software product quality. The system should grow constantly through continuous integration and frequent releases. Communication in the form of results feedback is important in this kind of development project. This poses new challenges for project management. In short, evolutionary approaches and the integration of user feedback seem to be becoming state-of-the-art, except that – especially in Web projects – these development approaches have to be dovetailed with the 'productive mode' of any software developed. We see *e-Prototyping* supporting an evolutionary approach in Web projects based on short development and release cycles with each of the releases being treated as an e-prototype for the next development effort.

5.1 Steps in e-Prototyping

In evolutionary and participatory software development, cyclic approaches were suggested as early as the 1980s, with emphasis being placed on the communication between developers and users. The STEPS model proposes cycles consisting of (1) revision establishment, (2) production, (3) system version release and (4) version application [10]. Founded on this kind of approach, we propose using prototyping to realize an evolutionary software development process in Web projects. Based on the four steps of evolutionary prototyping – functional selection, construction, evaluation and decisions on further use (see Section 3) – we show how to do *e-Prototyping* (see Figure 1):

1. The functional selection is based on requirements gathering. However, in the area of Web applications, "traditional" approaches are inadequate because, for example, the user group cannot be (clearly) determined beforehand, making a systematic approach practically impossible. Thus the initial requirements must be anticipated by the stakeholders (members of the development teams, the (Web) provider organization and business partners), the so-called "steering board" (see Figure 1, box "Stakeholder Arena"). In order to reduce time to market, to engage in a public discussion with the users of the new system version and to integrate users into the development process as early as possible, the plan for the first usable version should cover only essential functions that can be easily handled by the development team.

2. In each cycle, construction focuses on the technical and functional requirements selected. After construction, the software is released and treated as a productive system by the users, although it is regarded as a prototype from the development perspective and used as a "learning vehicle". Unlike "traditional" prototypes, it is used in real-life situations and is not labeled a prototype. Thus e-prototypes must meet higher software quality standards, which places additional emphasis on the quality of their technical design.

3. The evaluation relies heavily on communication channels established parallel to the use of each e-prototype/release. Feedback on the current software version may consist of error reports (from users and system administrators), usability problems and additional user requirements. The feedback is collected through communication channels. Calls by stakeholders for new 'strategic' applications to gain a competitive advantage are collected and discussed by the steering board (see Figure 1).

4. Decisions on the further use of the software version are taken based on the evaluation. In the application domain, the interests of users, providers and other stakeholders have to be addressed.

Prototyping should serve as a means to integrate their views into the development process. Ultimately, decisions on the software's further use are made from the management perspective (steering board) and constitute the basis of the next cycle's functional selection.

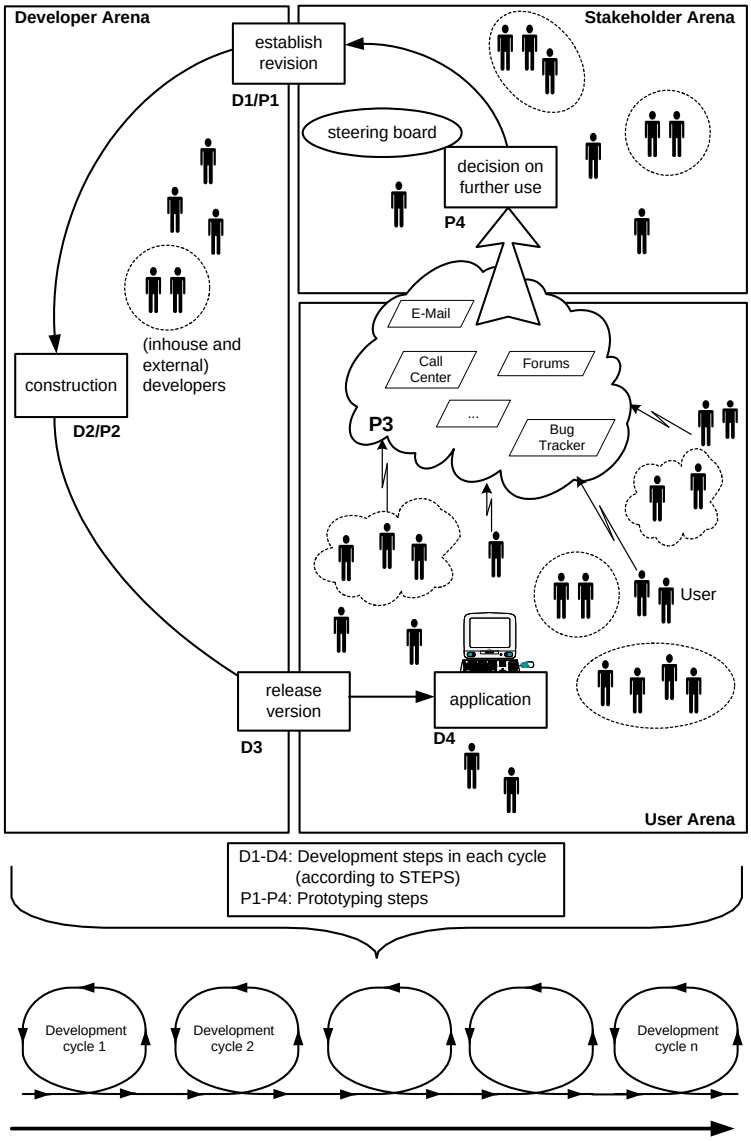

Fig. 1. The *e-Prototyping* process as part of a cyclic development process

These four steps can be regarded as one cycle in an evolutionary software development process steered by the prototyping approach. The decisions taken after evaluation provide input for the next cycle, starting with functional and technical selections prior to construction of the next version. The requirements for the follow-up version (based on necessary corrections and selected innovative changes)

should be limited so that the construction and release of the next version (e-prototype, release) takes no longer than three months.

5.2 Communication and Management

As communication between users and developers is essential to drive the prototyping process, we need communication channels to help establish a degree of interaction with the (mostly) "unknown" Web users. The following channels have proved particularly useful: e-mail sent to an address reserved for that purpose (e.g. feedback@web-organization.com), a call center where users' problems and suggestions can be recorded, a Web site containing error report forms, and electronic discussion forums. As far as possible, all contributions and calls should be answered to make users feel that they are being taken seriously. For example, if a user reports a bug, the user's input should be answered with a "thank you" mail. As soon as the bug is fixed, the reporting user should be notified.

In the Internet community, users often voluntarily take an active role in a project without deriving any direct benefits from it (cf. newsnet forums), e.g. because they are interested in a particular piece of software. For successful interaction between developers and users, it is important that these users feel they are being taken seriously and that the software provided is 'reliable' (which means, among other things, an implicit guarantee that help is available to the voluntary users in the case of a software error that causes serious damage on the user side).

The management of development processes that use *e-Prototyping* must strive for short release cycles, communication and innovation. Updates of a running Web-based application should be made at short intervals (a few weeks, 3 months at the most) and communicated explicitly. Bug fixes (patches) are required more frequently because they keep the above mentioned feedback channel free from bug reports, thus leaving room for essential communication. Only a bug-free system allows communication on functionality and usability. The "buggier" a system is, the more communication there is on errors or the existence of bugs. Market pressure is another factor contributing to short development intervals and frequent releases of innovative system versions.

The process described is riskier and much less controllable than in "traditional" software development. For example, a successful application attracts more users, which puts a greater load on the system, and this in turn leads to problems and erroneous behavior. Consequently, redesign of the system's architecture may become inevitable. Thus the focus of development activities can shift from a purely function-oriented approach to structural redesign in order to meet demands for scalability and a high-load service. Additional security needs on the users' side can lead to the implementation of initially unforeseen or unplanned safety features within the system.

To manage this process, all feedback collected from the different channels must be associated with a particular version and evaluated by a steering board. Its members decide what to put on the development agenda. This provides the basis for the next release, addressing bugs that should be removed immediately, and for feature enhancements. Those reporting a bug should be directly notified about improvements. It should also be made clear at what point the improvements will be integrated into the live system. In order to avoid duplicating reports, information about known problems should be made available to other users (cf. Mozilla and Bugzilla).

5.3 Discussion: Identifying Web User Requirements

Identifying Web user requirements is not an easy task because basic prerequisites have to be met and only regular users may consider giving feedback at all. First, the users must be willing to communicate with the developers. This may be difficult because it involves a fair amount of work and is time-

consuming. In addition, there may be a loss of privacy if the developers ask too many questions, e.g. about the users' hardware and software settings or the context in which they normally use the software. At the other end of the communication channels, the developers, too, must be willing to communicate with the users. It is always easier to simply implement a given list of features than to discuss functionality with the users. This is especially true if the number of users participating in the process is large. In this case, work with the users may be excessive compared to the time needed to build the system. We can overcome the latter problem by using the *e-Prototyping* approach with centralized collection and management of user feedback. This also enables developers to collect valuable information about different kinds of user groups, which helps in gradually defining the group of users as well as helping the developers to get a feeling for the needs of most users.

e-Prototyping helps to establish communication by promoting quick responses to user inquiries or notifications when a system change affecting them is made (e.g. the fixing of a bug or the implementation of a new feature suggested by a user). This way, the users feel they are being taken seriously and are motivated to continue communicating with the developers. Feedback ensured through bug tracking is a valuable incentive for both sides.

The short release cycles also help to identify user requirements more easily by avoiding too many bugs in one version of the software. This reduces white noise in the communication channels because less bug reports are necessary. Also, if bugs do not disturb the users, they can spend more time thinking about their needs in terms of functionality. At the other end of the communication channels, less bugs mean that the developers have more time to consider feature related user comments supporting the key question of how to identify requirements.

Identifying user requirements is a shared learning process in which the users acquire skills in understanding the possibilities a Web system may offer. In addition, users practice and learn to formulate their requirements. This process is supported by *e-Prototyping*, which promotes the development of an initial system with limited functionality that is extended with each system iteration. This approach gives users more time to gain a shared understanding.

We should be aware of the fact that identifying user requirements has its advantages and disadvantages for the developers. The disadvantage is that using the described process entails a loss of control. On the other hand, identifying user requirements and allowing users to participate are prerequisites for a better and thus more successful Web system in terms of addressing real user needs.

The *e-Prototyping* approach can be viewed as a framework for software development processes. It goes beyond most software development processes in that it also covers a process to integrate stakeholders in decision-making as well as in managing communication between developers, stakeholders and users. It adopts some aspects of Extreme Programming, e.g. the short cycles, but does not require all characteristics of XP to be met. However, it might be a good idea to consider XP for the software development part.

In this respect, the *e-Prototyping* approach can be seen as the next generation of the STEPS framework for software development processes. *e-Prototyping* adapts to the new conditions that have arisen from Internet-based software development.

e-Prototyping guarantees improved software quality through a transparent and short-cycled process. A lesson learned from Software Engineering is that quality cannot be achieved by products but only by processes. Since quality is always defined by the relevant users and contractors, *e-Prototyping* allows them not only to voice their requirements but also to define the group of relevant actors on a meta level.

6 Conclusion

In this paper, we have introduced the *e-Prototyping* approach to address problems encountered when developing application-oriented software for the Web. *e-Prototyping* is a way of letting users participate in the development process on a dependable basis. It was developed in several iterations on the basis of our project experience. The lessons learned so far indicate that it offers a way of building and deploying prototypes on the Web. It thus supports a learning process and presents an artifact to communicate about. It shows ways in which user feedback can be collected and distilled – thus helping to identify Web users' requirements. Finally, it is a way of marrying prototyping to software development or Web Engineering methods. It can be used both for programming or scripting language-oriented Web projects and for adapting commercial off-the-shelf products (i.e. content management systems, application servers, etc.). This makes *e-Prototyping* flexible enough for use in a variety of project settings and enables it to be used in combination with other Web Engineering methods.

1. Douglas Schuler and Aki Namioka, editors. *Participatory design: principles and practices*. Lawrence Erlbaum Ltd., Hillsdale, NJ, 1993.
2. Christiane Floyd, Wolf-Michael Mehl, Fanny-Michaela Reisin, Gerhard Schmidt, and Gregor Wolf. Out of Scandinavia: Alternative Approaches to Software Design and System Development. *Human-Computer Interaction*, 4(4):253–350, 1989.
3. David B. Lowe and John Eklund. Client Needs and the Design Process in Web Projects. In *Proceedings of the Eleventh International World Wide Web Conference*, Honolulu, Hawaii, 2002.
4. Christiane Floyd and Michaela Fanny Reisin. STEPS Projekthandbuch. Technical report, Technische Universität Berlin, 1986.
5. Athula Ginige and San Murugesan. Guest Editors' Introduction: The Essence of Web Engineering-Managing the Diversity and Complexity of Web Application Development. *IEEE Multimedia*, 8(2), 2001.
6. Athula Ginige and San Murugesan. Web Engineering: An Introduction. *IEEE Multimedia*, 8(1), 2001.
7. Sabine Madsen. Web Development – Making Sense of Opinions, Positions and Perspectives in the Literature. In Sanna Laukkanen and Sami Sarpola, editors, *Electronic Proceedings of the 26th Information Systems Research SEminar in Scandinavia*, volume 26, pp. 1–14, Haikko, Finland, August 9-12 2003. Information Systems Research in Scandinavia (IRIS).
8. Christiane Floyd and Heinz Züllighoven. Softwaretechnik. In Peter Rechenberg and Gustav Pomberger, editors, *Informatik-Handbuch*, pp. 763–790. Hanser, München, 2nd edition, 1998.
9. Ian Sommerville. *Software Engineering*. International Computer Sciences Series. Addison-Wesley, Harlow, UK; Reading, Mass., 5th edition, 1996.
10. Christiane Floyd, Fanny-Michaela Reisin, and Gerhard Schmidt. STEPS to Software Development with Users. In Carlo Ghezzi and John A. McDermid, editors, *ESEC89*, number 387 in Lecture Notes in Computer Science, pp. 48–64. Springer-Verlag, Berlin, 1989.
11. James A. Highsmith III. *Adaptive Software Development – A Collaborative Approach to Managing Complex Systems*. Dorset House Publishing, New York, 1999.
12. Alistair Cockburn. *Agile Software Development*. The Agile Software Development Series. Pearson Education, Inc, Boston, 2001.
13. Jim Highsmith. *Agile Software Development Ecosystems*. The Agile Software Development Series. Addison-Wesley, Boston, 2002.
14. Doug Wallace, Isobel Raggett, and Joel Aufgang. *Extreme Programming for Web Projects*. The XP Series. Addison-Wesley, Pearson Education, Boston, 2003.
15. San Murugesan, Yogesh Deshpande, Steve Hansen, and Athula Ginige. Web Egnineering: A New Discipline for Development of Web-Based Systems. In San Murugesan and Yogesh Deshpande, editors, *Web Engineering 2000*, number 2016 in Lecture Notes in Computer Science, pp. 3–13. Springer, 2000.
16. Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, and Maristella Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann, Amsterdam, 2003.

17. Linda B. Sherrell and Lei da Chen. The W Life Cycle Model and Associated Methodology for Corporate Web Site Development. *Communications of the Association for Information Systems*, 5(7), April 2001.
18. Daniel Schwabe, Luiselena Emeraldo, Gustavo Rossi, and Fernando D. Lyardet. Engineering Web Aplications for Reuse. *IEEE Multimedia*, 8(2), 2001.
19. O.M.F. De Troyer and C.J. Leune. WSDM: A User-Centered Design Method for Web Sites. In *Proceedings of the seventh WWW conference*, 1998.
20. Brendan Haire, Brian Henderson-Sellers, and David Lowe. Supporting Web Development in the OPEN Process: Additional Tasks. In *Proceedings of COMPSAC'2001: International Computer Software and Applications Conference*, Chicago, Illinois, USA, October 8-12 2001.
21. Brian Henderson-Sellers. OPEN: The first full lifecycle, third generation OO method. In S. Zamir, editor, *Handbook of Object Technology*. CRC Press, Boca Raton, Florida, 1999.
22. Reinhard Budde, Karin Kuhlenkamp, Lars Mathiassen, and Heinz Züllighoven, editors. *Approaches to Prototyping*. Springer Verlag, Berlin, Heidelberg, 1984.
23. Reinhard Budde, Karl-Heinz Kautz, Karin Kuhlenkamp, and Heinz Züllighoven. *Prototyping*. Springer Verlag, Berlin, Heidelberg, New York, Tokio, 1992.
24. Horst Lichter, Matthias Schneider-Hufschmidt, and H. Züllighoven. *Software Project Management - Readings and Cases*, chapter Prototyping in Industrial Software Projects - Bridging the Gap Between Theory and Practice, pp. 306–317. Irwin, Chicago, Boston, 1997.
25. Gustav Pomberger and Gnther Blaschek. *Object-Orientation and Prototyping in Software Engineering*. The Object-Oriented Series. Prentice Hall, Hempstead, 1996.
26. Christiane Floyd. A Systematic Look at Prototyping. In Reinhard Budde, Karin Kuhlenkamp, Lars Mathiassen, and Heinz Züllighoven, editors, *Approaches to Prototyping*, pp. 1–18. Springer, Berlin, Heidelberg, New York, Tokio, 1984.
27. Antoinette Kieback, Horst Lichter, Matthias Schneider-Hufschmidt, and Heinz Züllighoven. Prototypen in industriellen Software-Projekten – Erfahrungen und Analysen. *Informatik-Spektrum*, 15(2), 1992.
28. Wolf-Gideon Bleek. *Situations in Life* to Support the Use and Modeling of Municipal Information Systems. In Dan Remenyi and Frank Bannister, editors, *Proceedings of the European Conference on e-Government*, pp. 49–60, Trinity College Dublin, September 2001. MCIL.
29. Bernd Pape, Wolf-Gideon Bleek, Iver Jackewitz, and Michael Janneck. Software Requirements for Project-Based Learning – CommSy as an Exemplary Approach. In *Proceedings of the 35th Hawaii International Conference on System Sciences*, Big Island, Hawaii, Januar 2002. IEEE Computer Society.
30. Rens Scheepers. Key Role Players in the Initiation and Implementation of Intranet Technology. In Ojelanki K. Ngwenyama, Lucas Introna, Michael Myers, and Jan DeGross, editors, *New Information Technologies in Organizational Processes*, pp. 175–195. Kluwer Academic Publisher, August 1999. Proceedings of IFIP WG 8.2.
31. Wolf-Gideon Bleek. *Software-Infrastruktur – von analytischer Perspektive zu konstruktiver Orientierung*. Hamburg University Press, Hamburg, Germany, 2004.
32. Jan Damsgaard and Rens Scheepers. A Stage Model of Intranet Technology Implementation and Management. In Jan Pries-Heje, Claudio Ciborra, Karlheinz Kautz, J. Valor, E. Christiaanse, D. Avison, and C. Heje, editors, *Proceedings of the 7th European Conference on Information Systems*, pp. 100–116, Copenhagen, Denmark: Copenhagen Business School, June 23-25 1999.
33. Kent Beck. *Extreme programming explained: embrace change*. Addison-Wesley, Reading, Mass., 2000.