In: Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC 2002), 26–29 August 2002, Oxford, England.

Developing Web-based Applications through e-Prototyping

Wolf-Gideon Bleek, Martti Jeenicke, Ralf Klischewski
Hamburg University, Department for Informatics, Software Engineering Group
Vogt-Koelln-Str. 30, 22527 Hamburg, Germany
Phone: +49 40 42883-2413, Fax: +49 40 42883-2303
{bleek, jeenicke, klischewski}@informatik.uni-hamburg.de

Keywords: Prototyping, Web-based Applications, Software Process Management, Evolutionary Development

Abstract

"Traditional" prototyping is based on assumptions that are not – or only partially – valid for the development of Web-based applications. In this paper, the "traditional" approaches of prototyping are recalled, and the new conditions, difficulties and challenges of involving and communicating with the relevant actors as a way of gathering sound requirements in the context of Web-projects are examined. The paper presents a modified prototyping approach called e-prototyping. It includes frequent releases of software versions (based on short development cycles) as well as gathering feedback from users and other relevant actors in "productive mode". The approach points at the need for offering various communication channels and systematically sorting out the stream of feedback to create sound requirements for evolutionary software process and release management.

1. Introduction

Prototyping has become a well established method within application-oriented software development – will this hold true also for the development of Web-based applications? In many projects prototyping is used within an iterative approach to meet the requirements of a specific application domain in which it is difficult to anticipate requirements. But "traditional" prototyping is based on assumptions that are not – or only partially – valid for the development of e-Business, e-Commerce, and e-Government applications (regarding e.g. timeframe, actors involved, communication, controllability and relevance of application). Is this the end of prototyping? Or will this method survive in the world of networked and distributed systems? And if so, what modifications would have to be made to the management of software development projects? Do we need 'e-prototyping'?

In this paper we recall the "traditional" approaches of prototyping (section 2) and examine the new conditions, difficulties and challenges of involving and communicating with the relevant actors as a way of gathering sound requirements (section 3). Based on this discussion, we outline an approach for e-prototyping (section 4): as there is "no lab on the Net" for prototyping, every release that is published runs as a productive system – project management should therefore strive for short development cycles with frequent releases of software versions including small steps of innovation, and collect feedback from users and other relevant actors in "productive mode" by offering various communication channels. Maintaining the channels needs special emphasis on systematically sorting out the stream of feedback to create sound requirements to be handled by the evolutionary software development and release management.

2. Prototyping in Application Oriented Software Engineering

Prototyping has become a well accepted technique in software engineering. Sommerville ([10], pp. 138-153) describes it as a means of requirements analysis and validation. Prototypes support the communication among developers and users, by enabling them to "experiment with requirements". Evolutionary prototyping is special form of prototyping characterized by an iterative process in which the requirements that are understood are implemented first. After they are implemented the developers move on to those requirements which are still unclear. Our argument will be based on this kind of prototyping (see also below), which is also well in line with the European/Scandinavian ideas of participatory design. Prototypes are an important type of artefact and a source of insight in a continuous learning process. 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 [4] takes a similar view on the process – they differ in the activities at the beginning and at the end: in contrast, Sommervilles' process structure starts with an explicit step in which the prototyping objectives are established. It also ends with an evaluation, while Floyd promotes a step in which a

decision on the further use of the prototype is made. In our view, the latter is very important for a prototyping approach to software development. We will therefore follow Floyd's view and describe her steps in greater detail:

- *Functional Selection:* part of the functional selection is a decision on which and/or how many functions the prototype should include.
- *Construction:* according to Floyd, this step comprises all efforts to make the prototype available. Since the prototype is a learning vehicle for all persons participating in the development process, only a few quality requirements of the future system will be considered important during these activities.
- *Evaluation:* the construction of a prototype only makes sense if it is going to be evaluated afterwards. This step is crucial for the prototyping process because it comprises the collection of information and experiences that are very valuable for the developed product itself. It should therefore be planned and performed thoroughly.
- *Further use:* depending on the experiences gained there are two possible kinds of further use of the prototype. One possibility is to use it solely as a learning vehicle. Many authors denote these prototypes as "throw away" prototypes. The second possibility is the use of the complete prototype or parts of it in the target system.

The resulting prototypes of this process support the learning among all project participants and helps to sort out misunderstandings at an early stage of the software development process. Over the years, experience has shown that prototyping leads to better quality, an overall reduction in misunderstandings, thus establishing its acceptance as part of the methodology of application oriented software development.

This kind of software engineering practice is now challenged by the conditions developers find in Web projects.

3. Requirements Gathering for Developing Web-based Applications

Prototyping is a way of involving other actors than software developers in the process in order to enhance the product development. In this section, we point at the new conditions, difficulties, and challenges of involving and communicating with the relevant actors as a way of finding out about requirements within a development process for Web-based applications. The use of the term Web-based application is inconsistent within the literature. For this paper we define a Web-based application as an information system targeted at a large and vast user group. The system is running on a server presenting its user interface over the Internet on a Web-browser. Before discussing the new challenges and problems we evaluate some of our experiences from a Web development project we have been involved in:

Example: In the development of a municipal information system [2] the underlying process model was a "Big-Bang" approach, i.e. around 1¼ years of development without a single release. The approach failed before results could be presented to the public because of a variety of problems (acquirement of technology, performance, interconnections and interoperability with other systems, misunderstandings, conflicts of interest). Offered a second chance, only dramatic changes saved the project: short development cycles including a complete new development in a small team, a consistent functionality visible at a glance, enabling a short "time to release/time to market". But there was still no systematic collection and evaluation of communication (e.g. complaints via email or call center) with the users. Meanwhile, the planning of new releases is completely driven by management decisions and not communicated outside the project. The lack of communication channels for the users (the only way for them to express their thoughts is a guest book) and an underestimated role of the users in the development process has repeatedly led to disappointment in the case of municipal information systems.

Observed Problems in Developing Web-based Applications

Our evaluation of the project example showed that the same questions arose repeatedly and pointed at substantial problems such as:

- How can the (initial) requirements for network-based Web applications be defined?
- How can the requirements be gathered systematically, if the target (Web-) users of the system are unknown and hardly describable in their characteristics?
- Which actors should participate in the process and how?
- What are the consequences for requirements gathering and the development process of the continuous expansion of the technical system that supports the applications?

We claim that these problems arise in many other Web projects as well. However, we here focus on public corporate Web sites, which includes Web portal construction [9]. These systems offer a pool of topically related services. Examples of such systems from the field of e-Government are, among others, city, county or country Web sites offering a variety of services and information

provided by the site's owner or partner organizations. We will not discuss intranets and extranets in which developers can relate to a well-defined group of users and other actors (for intranet development see e.g., [3] and [8]). Most of our findings will also apply to Electronic commerce systems (comprising classical shop and auction systems), although we will not focus on specifics of workflow and transaction management.

There already exist approaches to cope with some of the problems identified; for instance Sherrell and Chen [9] discuss integrating prototyping in connection with their W Life Cycle model. They likewise strive for including users' contributions into the development process, stimulated by the usage of a productive system. However, Sherrell and Chen assume a determinable and available user group, at best only available in some types of Web projects such as intranet development within rather small organizations. In the following we try to analyse difficulties of the user-developer-relation within Web projects in more detail, relating them to the typical problems of software development and contesting some of the basic assumptions about "traditional" prototyping.

Requirements in software projects mostly relate to milestones structuring the overall software process. Within this kind of development process, prototypes are built in order to gain new insights and support decision-making if applicable, embedded in the iterations of requirement analysis, prototyping, realization, releasing the product and revising. E-projects do not enjoy this kind of freedom:

- Any software released to use on the Web is without protection: publicly accessible Web prototypes are always exposed to public criticism – no more "playing around" with a development system.
- Each feedback round with users needs time to prepare, present, communicate, and evaluate. But strong market pressure and high expectations usually do not allow Web projects to wait 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 developer to define work packages for shorter time periods.
- What e-applications have in common is that they are "early adopters" [8] in their domain, i.e. they offer a new service on the Internet. Development has to keep in mind that the application is expected to feature high quality and innovation.

"Traditional" prototyping contributes to software requirement definition through cooperation between developers and (future) users, in most cases working for an organization which orders/owns the future product. There are usually different perspectives and interests (e.g. management vs. user), but the relevant actors are identifiable. In Web projects, which reach beyond organizational borders, requirements gathering for network applications is framed quite differently:

- initial requirements are defined by the providers' view for a potential application (the wishes and demands of the current user group will 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 an (un-) limited number of partitions;
- the relevant actors cannot be represented within a simple actor model (e.g. including developers, users, and management) – actors contributing to the system development take on new roles such as, e.g. "technology champion" [3], (sub-)service provider, and others;
- actors with different perspectives and interests are usually not part of the same organization which precludes direct and personal discussions (e.g. users or decision makers cannot easily be invited or are not available for single or group interviews) or even simple user observation.

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

- Contractors, at least in principle, expect a return on their investment. But Web projects are often not accountable in terms of rationalization effect, the result may be 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. strong complainers who will criticize errors or missing functions on a regular basis, volunteers trying to play an active role in the further development of the Web application by spending a lot of time on evaluation and making constructive suggestions for improvement.
- The developers' interests significantly differ from other perspectives. Their activity is steered by keeping the software error-free, making the latest back end technology run, and implementing state-of-the-art features. Their perspective is limited to one of a "power-user". On the other hand, project restrictions apply owing to technical conditions and limits set by other actors. The developer's job is to integrate a system in a given environment coping with existing or predefined technology and to make it run reliably.

New challenges to bring the relevant perspectives into the development process within a Web project include the acknowledgement of new perspectives, but also of the new channels and media used to give these perspectives a voice. We now have a situation in which the users themselves foster the communication within the medium (the Internet), e.g. volunteers moderate forums and organize user groups (see next section how developers can make use of these communication channels).

Web projects are facing new operating conditions which become visible step by step as the application is already in productive mode with reactions from "real" users. Thus requirement gathering in Web projects cannot make use of "traditional" prototyping as many assumptions no longer hold true. However, there are many relevant actors, and their perspectives should be included to help save expenses and foster innovation. We will now discuss how a new way of e-prototyping could support an evolutionary approach based on short development cycles and organize and maintain user feedback through active communication channels supplying the development with the valuable information needed.

4. How to do e-Prototyping

In this section we describe our approach to e-prototyping. Firstly, we argue that the current trend in software engineering towards shorter development cycles leads to an intertwining of prototyping and release management. Secondly, we describe the steps of our e-prototyping approach in relation to "traditional" prototyping activities. Thirdly, we show how obstacles in the user-developer relation can be overcome by fostering communication and integrating this into the development process.

Shortening development life cycles seems to be an issue in various fields of software engineering. One example of a new method that propagates shorter development cycles is Extreme Programming [1]. Beck calls for shorter cycles on all levels of software engineering in order to increase the quality of a software product. The system should grow constantly through continuous integration and frequent releases. Frequent releasing is also very common in projects of the open source community [6]. Communication in the form of feedback of the results is very important in this kind of development project (e.g. Mozilla project). This poses new management tasks to the project. Crucial management challenges have been identified in a study commissioned by the German software industry ([11], p. 10). In short, evolutionary approaches and making use of user feedback seem to become state of the art, only that, especially in Web projects, these development approaches have to be intertwined with the 'productive mode' of any software developed. We see e-prototyping supporting an evolutionary approach for 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.

Steps in e-prototyping

Within evolutionary and participatory software development, cyclic approaches were suggested as early as in the 1980s, putting emphasis on the communication between developers and users. E.g. the STEPS model [5] proposes cycles consisting of (1) revision establishment, (2) production, (3) releasing a system version and (4) application of the version. Based on this kind of approach, we propose prototyping to realize an evolutionary software development process within Web projects. Framed by the four steps of evolutionary prototyping – functional selection, construction, evaluation, and decision on further use (see section 2) – we outline how to do e-prototyping (see figure 1):

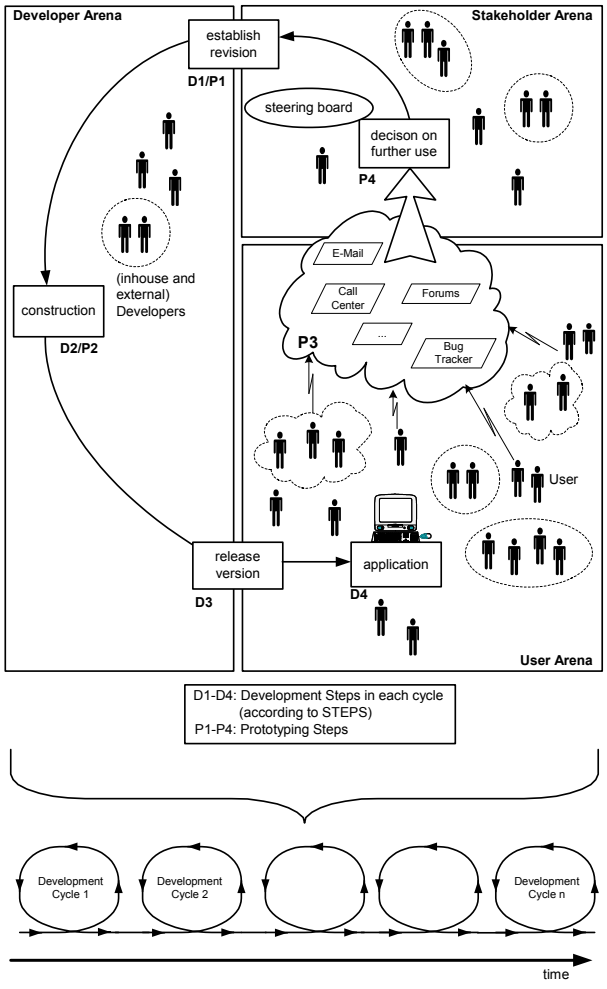

Figure1 : The e-prototyping process as part of a cyclic development process

1. In order to perform a **functional selection,** requirements need to be gathered. However, in the area of Web applications and their e-Prototypes, "traditional" approaches fall short as, e.g., the user group cannot be determined beforehand (or at least only very vaguely) so that a systematic approach is practically impossible (see section 3). The initial requirements therefore have to be anticipated at the beginning of the project by the stakeholders (members of the development teams, the (Web) provider organization, and of business partners), the so-called "steering board" (see figure 1). To start with, the goal in mind is "to go public" fast to reduce the "time to market", to face a public discussion with the users of the new system version in a short time and to integrate users into the development process as soon as possible. The plan for the first usable version should cover only essential functions that can easily be handled by the developing team regarding the knowledge barriers mentioned above (technology-related, project-related, application-related). Therefore an appropriate functional selection is the basis for a cyclic development that helps to cope with the barriers. The experiences gained during each cycle help the developing team to master further steps in the development. The functional selection in the next cycles is then based on decisions from the steering board evaluating user feedback (see below, steps 3 and 4).
2. In each cycle, **construction** focuses on technical and functional requirements selected. After construction the software will be released, i.e. made accessible for use through the Internet. It will then be treated as a productive system by the people who use it, although it is regarded as a prototype from the development perspective and used as "a learning vehicle". In contrast to "traditional" prototypes, it is being used in real life conditions, and is not labelled as a prototype. E-Prototypes, i.e. releases, must therefore meet higher standards on the quality of the software than "traditional" prototypes, which puts an additional emphasis on the quality of the technical design. Construction must aim at a working system as a precondition obtaining and evaluating user feedback.
3. The **evaluation** heavily relies on communicational means (see next subsection) established in parallel to the use of each e-Prototype/release. Feedback concerning the current software version may consist of
   - error reports collected from users and system administrators,
   - usability problems excerpted from discussions, and
   - additional (and new) requirements of the users (technology pull).

   Error reports, usability problems, and additional requirements are collected through diverse communication channels and published. Calls for new 'strategic' applications from stakeholders to gain a competitive advantage (technology push) are collected and discussed in the development team and a decision making board (see figure 1).
4. **Decisions on the further use** of the software version result from taking into account the evaluation. In the application domain one has to deal with users, providers and other stakeholders. Prototyping should serve as a way of integrating their view into the development process. In the end, decisions on the further use are taken from the management perspective (steering board) and form the basis of the next cycle's functional selection.

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 give input for the next cycle starting with functional and technical selection prior to the construction of the next version. The requirements for the follow-up version (based on necessary corrections and selected innovative changes) should be limited in such a way that the construction and release of the next version (e-Prototype, release) will not take longer than three months.

Communication and Management

As **communication** between users and developers is essential for driving the prototyping process, we need communicational means to help establish some interaction with the (mostly) "unknown" Web users. The following channels have proved to be particularly useful: email sent to an address reserved for that purpose (e.g. feedback@web-organization.com), a call-centre where users' problems and suggestions can be recorded, and a Web site containing error report forms, and electronic discussion forums. As far as possible, contributions and calls have to be answered if necessary.

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

The **management of** development processes that use e-prototyping must strive for short releases, communication, and innovation. Updates of a running e-application should be made at short intervals (a few weeks, 3 months at maximum). Bug fixes (patches) are required more often because they keep the above-mentioned feedback channel

clear of bug reports. Thus leaving room for the essentials of the feedback channel. Only a bug-free system enables communication about functionality and usability. The more "buggy" a system is, the more of the communication is about errors or the existence of bugs. Market's pressure is another factor, that contributes to very short development intervals and frequent releases of innovative system versions.

The process described is more risky and much less controllable as it is in "traditional" software development. For example, a successful application attracts more users, which leads to a greater load on the system and in turn provokes problems and erroneous behaviour. As a consequence, a redesign of the system's architecture might become inevitable. Thus the emphasis of development activities can shift from a solely functional oriented approach to a structural redesign in order to meet demands of scalability and a high load service. Additional security needs on the part of the user can lead to safety features within the system initially not foreseen and planned.

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

5. Conclusion

Software development in Web projects faces difficulties in gathering sound requirements from the application domain, because, among other things, Web users are scattered and out of reach for prototyping in a laboratory setting. Analysing the new conditions, difficulties and challenges of requirement gathering in Web projects, we have outlined an approach for e-prototyping by integrating cycles of evolutionary development and prototyping approaches. The development of Web-based applications can make use of e-prototyping in three ways:

- systematically collecting, streaming and sorting out feedback to create sound requirements any time a new release is envisioned within the frame of evolutionary software development and management of short-period releases
- integrating e-prototypes as 'productive systems' to manage the development of Web applications in short development cycles with frequent releases of software versions including rather small steps of innovation and less project risks
- progressing in small steps with continuous evaluation, revealing possible knowledge gaps, and promoting knowledge sharing between all the actors involved.

Of course, elements of e-prototyping are already part of many Web projects. However, having drawn on a concept, which has proved to be successful in application oriented software development, we conclude that e-prototyping is a useful way of enriching the methodology for developing Web-based applications. From the academic viewpoint, e-prototyping may provide a new focus within "Web Engineering" [7] or simply for advances in software engineering. In any case, a systematic approach to prototyping may support application development throughout the World Wide Web.

6. References

[1] Beck, K., "Extreme programming explained: embrace change", Addison-Wesley, Reading, Mass, 2000.

[2] Bleek, W.-G., "Situations in Life to Support the Use and Modelling of Municipal Information Systems", Remenyi D. and Bannister, F., Proceedings of the European Conference on Electronic Government, Trinity College Dublin, Ireland, 2001, pp. 49-60.

[3] Damsgaard, J. and Scheepers, R., "A Stage Model of Intranet Technology Implementation and Management", in: Proceedings of the 7th European Conference on Information Systems, 1999, pp. 100–116.

[4] Floyd, C., "A Systematic Look at Prototyping", in: R. Budde, et al. (eds), "Approaches to Prototyping", Springer, Berlin, 1984, pp. 1-18.

[5] Floyd, C., F.-M. Reisin, G. Schmidt, "STEPS to Software Development with Users", in: Ghezzi, C., J.A. McDermid (eds.), ESEC '89, pp. 48-64, LNCS 387, Springer, Berlin-Heidelberg, 1989.

[6] Jørgensen, N., „Putting it all in the trunk: incremental software development in the FreeBSD open source project", Informations Systems Journal, 11(4), 2001.

[7] Murugesan, S. and Deshpande, Y. (eds.), "Web engineering: managing diversity and complexity of web application development", Springer, Berlin, 2001

[8] Scheepers, R., "Key Role Players in the Initiation and Implementation of Intranet Technology", in: New Information Technologies in Organizational Processes – Field Studies and Theoretical Reflections on the Future of Work, Proceedings of IFIP WG 8.2, Chapman and Hall, August 1999, pp. 175–195.

[9] Sherrell, L. B. and L.-D. Chen, "The W Life Cycle Model and Associated Methodology for Corporate Web Site Development", Communications of the Association for Information Systems, (5), Article 7, April 2001.

[10] Sommerville, I., "Software Engineering", 5th edition, Addison-Wesley, Harlow, UK, 1996.

[11] Stahl, P., et al. "Analyse und Evaluation der Softwareentwicklung in Deutschland", GfK Marktforschungs GmbH. http://www.dlr.de/IT/IV (last visited 11/27/01)