From Maintenance to Evolutionary Development of Web Applications: A Pragmatic Approach

Rudolf Ramler[1], Klaus Wolfmaier[1], and Edgar Weippl[2]

[1] Software Competence Center Hagenberg GmbH,
Hauptstrasse 99, A-4232 Hagenberg, Austria
{rudolf.ramler, klaus.wolfmaier}@scch.at
[2] University of Vienna, Liebiggasse 4/3-4, A-1010 Vienna, Austria
weippl@acm.org

Abstract. Development of Web applications is dynamic by its very nature. Web development processes have to facilitate a Web application's continual refinement and evolution based on feedback from end-users. Evolutionary development can easily be achieved by end-user involvement through seamless integration of feedback and issue reporting mechanisms into Web applications. This paper discusses the use of conventional methods and tools for maintenance and change management as an infrastructure for evolutionary development of Web applications. An example demonstrates the feasibility of the proposed approach. It describes our experience from integrating the open source issue tracking system Bugzilla into a Web application.

1 Introduction

Today's economic realities pressure organizations to continuously adapt to shifting environments. Accordingly, "systems should be under constant development, can never be fully specified and are subject to constant adjustment and adaptation" [1]. "Web developers have the capability to modify their systems for all users immediately, without being impeded by the manufacturing, distribution and sales channel delays inherent in shrink-warp software development." [2] Web applications are installed on a central server and modifications are instantly propagated to all users. As a consequence, Web applications can be developed in an evolutionary process. Once a feature is implemented, end-users can start using it and their feedback can be quickly incorporated into new releases. This iterative process of delivering the application in multiple small steps allows immediate response to rapidly changing business needs while continually maturing and adapting the application over time. Refinement and adaptation are in many cases the only viable option in view of the fact that business needs often change as development proceeds, making a straight path to an end product unrealistic [3].

The goal of this paper is to present an easy yet effective approach to seamless end-user involvement into the evolutionary development process of Web applications. By relying on existing maintenance infrastructure, the users' feedback can be routed directly to the developers with little additional overhead. According to this goal, the remainder of this paper is structured as follows. In section 2 we present requirements

N. Koch, P. Fraternali, and M. Wirsing (Eds.): ICWE 2004, LNCS 3140, pp. 287–299, 2004.

for a Web development process. Section 3 outlines our conventional maintenance approach and the associated infrastructure (methods and tools), while section 4 describes how we enhanced the maintenance infrastructure to support evolutionary Web development. Section 5 gives an overview of related approaches that deal with rapid changes in Web development, and section 6 concludes the paper.

2 Requirements for a Web Development Process

From our experience in developing Web applications as well as from literature research, we have derived a list of requirements for our Web development process. The most important requirements are to provide (1) end-user involvement, (2) prototyping, (3) change management, (4) immediate response, (5) risk minimization, (6) no administrative overhead, and (7) transparency and overall guidance.

End-user involvement. Among the top three reasons for challenged or failed projects is the lack of end-user involvement [4]. Knowing the end-users' requirements is essential for the development of successful Web applications. The customer, although defining the main goals for the development of a Web application, usually is not the actual end-user and, therefore, he or she is not able to define all the requirements important to end-users. Involving actual end-users is most effective when uncertainty about requirements is high [5], but difficult to achieve in Web projects. The main reasons are the potentially high number of end-users, their geographical distribution (possibly all over the world), and their anonymity due to the lack of direct interaction.

Prototyping. End-users should be capable to provide all necessary requirements. However, as Web applications and their related concepts and metaphors are still new to many users, they have difficulties to develop realistic expectations and to express their needs [6]. Prototyping is used to leverage the involvement of end-users in Web application development. End-users are potentially highly skilled evaluators of product functionality [7]. Thus, we find it viable to prepare a first, stable prototype and present it to the end-users as a basis for feedback. Thereby, the users can refer to a clearly defined part of the application, which helps to avoid a plethora of (unrealistic) ideas and wishes. With emphasis on evolutionary prototypes of reasonable usability and reliability, we strive to establish continuity in development and avoid confusion and frustration of end-users – the main pitfall of rapid application development in Web development according to [8].

Change management. In many cases, it is not possible to fully specify the requirements of a Web application at the beginning of the project, because they will either evolve over time or change during development. Typical reasons are frequent changes in the environment (e.g. new business partners and competitors), new user requirements (stemming from intermediate development results presented to the user), the availability of new technologies, methods, and tools, or the need for refactoring and error correction throughout development. Thus, the incorporation of effective mechanisms to manage the Web application's change and maintenance is one of the key steps of successful development [9]. A single channel for requests and issues has to be defined and a change control board to prioritize requests and issues has to be established to allow for flexible and prompt reaction.

Immediate response. Developing Web applications means "internet-speed" software development [10]. Immediate response to changing requirements, to user feedback, and to market shifts is required. Regarding the aforementioned change management, immediate response relies mainly on immediate reaction to feedback from the end-users to intermediate development results, such as prototypes. Thereby, immediate feedback to the user is necessary in order to inform the end-user about the consecutive steps and to show that his/her effort is taken seriously, even if not all suggestions can be incorporated right away. In addition, immediate response to market situations provides the ability to tackle the demanding time-to-market problem in Web development. By employing evolutionary prototyping [7] it is feasible to go on-line with a first working solution that serves as a starting point for further development efforts and to exploit the opportunity of being a first-mover.

Risk minimization. Web projects have to deal with a high level of uncertainty and a large number of risks [12]. Typical examples are unrealistic schedules, misunderstood or frequently changing requirements, underestimated complexity of non-functional requirements, and unstable technologies. An ideal Web development process explicitly addresses these risks. Therefore, a realistic estimation of the project situation is necessary. This is possible through frequent feedback from customers and end-users on working examples and the continuous adaptation of previous estimations.

No administrative overhead. Typical Web projects have to cope with harsh limitations of financial resources, and – in addition – with restrictive schedules, often less than one or two months [11]. Thus, it is important to keep the administrative overhead at a minimum. Lightweight or agile approaches have to be applied and the often immense burden of (inappropriate) regulation and tool usage has to be avoided in order to keep both process costs and reaction time low. Yet, lightweight tools can significantly increase effectiveness and efficiency with simple, problem-specific solutions.

Transparency and overall guidance. In evolutionary Web development establishing a clear vision and long-term goals for the project is essential to unite the project team's efforts and to assure that all the efforts are aimed in the right direction. The product management, comprising the main stakeholders (e.g., customer and project manager), is responsible to outline a roadmap for future work and to align the project goals with business goals. A shared vision is required. Misalignment with business goals is quoted as one of the ten deadly Internet and Intranet project risks [12]. The collected and reviewed feedback from end-users serves as a decent controlling instrument. For effective guidance, the whole process has to be transparent for the team members as well as for the customers.

Most of these requirements have been frequently cited throughout the Web engineering literature ([8], [13], [14], [15], [3]) as prerequisites for the development of Web applications. For us, the question is not whether or not to incorporate these requirements, but how we could do this without the burden of changing existing processes and established structures or applying new methods and new tools. Yet, a rigorous reduction of development cycles resulting in a blend of development and maintenance activities proved as an effective and pragmatic approach towards an evolutionary development process. Tool support helped to make this approach efficient and systematic.

3 (Conventional) Maintenance and Change Management

Maintenance is the modification of a software product after it has been placed into production. Commonly, four different types of maintenance are distinguished [8]: (1) *Corrective maintenance* to fix bugs and design deviations, (2) *preventive maintenance* to avoid future bugs or maintenance, (3) *adaptive maintenance* when a system's environment changes, e.g. when a new Web browser is released and the application has to be modified to work properly within that browser, and (4) *perfective maintenance* to introduce enhancements such as new functionality or to increase efficiency.

Keeping track of changes and their effects on other system components is not an easy task. The more complex the system is, the more components are affected by each change. For this reason, change management [3] – important during development – is critical during maintenance. For conventional software development, our approach is to establish a change management process lead by a change control board to oversee this process. Open source tools, such as CVS and Bugzilla, are employed to support control and collaboration among team members.

CVS (Concurrent Versions System) [16] is used for version control. This tool keeps track of the different versions of all the files that are part of a project. CVS version control significantly eases the collaboration between developers and the authoring of Web sites [17]. In addition, we rely on Bugzilla [18] to report and track issues (e.g. requirements, bugs, user requests, or remarks). Bugzilla is a popular open source issue tracking system that derived from the open source project Mozilla. It provides an extensive set of features useful for distributed development teams and end-user involvement, such as a Web-based user interface or an email notification mechanism. As the source code is freely available, Bugzilla can easily be adopted and extended to meet specific organizational needs. Nevertheless, the predefined workflow introduces a flexible yet clear and systematic way to deal with issues. The state diagram in Figure 1 illustrates the basic steps of the workflow from the perspective of an issue's lifecycle. Further details can be found in [18] or [19].

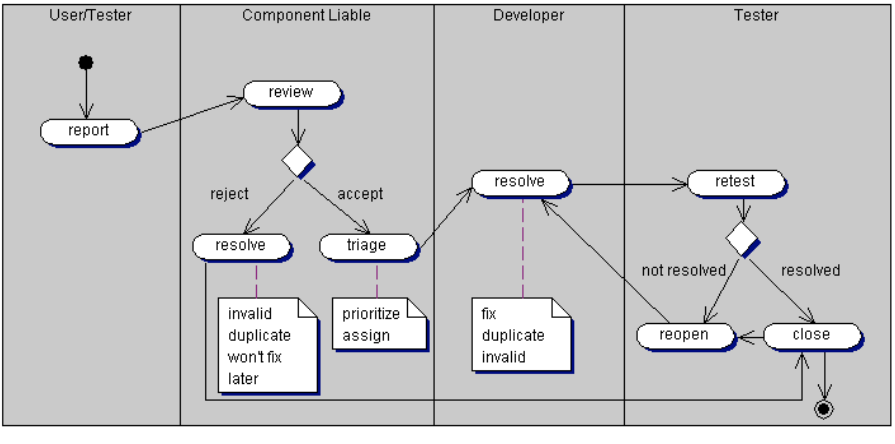

Fig. 1. Bugzilla issue management workflow

4 Transition to Evolutionary Development

David Lowe compares the development of Web applications with town planning and landscape gardening: "The evolution of Web applications is analogous to a garden changing as a natural part of its cycle of growth." From this, Lowe draws the following conclusions: "… Web development tends to differ greatly in that we are no longer aiming to develop a 'finished' product. Rather, we are aiming to create an organic entity that starts with an initial consistent structure, but continues to grow and evolve over time. This evolution is much finer-grained than the maintenance changes that occur with more traditional software products, and tends to be an integral part of the lifecycle of the product. Compare this to conventional software maintenance, which tends to be a coarse-grained response to errors in the products, changes in requirements, or a changing environment." [20]

In conformance to Lowe's suggestions, we reviewed and aligned our development process. Instead of aiming towards a final product from the very beginning, we started with laying out a general, flexible fundament for a more generic Web application by modularizing functionality and by relying on component-oriented development. For our first projects, we used an in-house Web application framework, similar to the architecture described in [21], since other available solutions were still in their infancies. The framework allowed us to develop modular function blocks and to specify their coupling, parameters, error conditions, and access rights through XML configuration files. This framework was later on superseded by the freely available Struts framework [22], part of the Apache project. On the basis of this initial structure we developed the functionality in small increments to enable evolutionary growth. These increments did not encompass fully featured implementations of functions, but rather initial, stable, and useable prototypes to demonstrate the conceptual and technical feasibility. Only the feedback from the end-users facilitated the function's continual refinement and maturation.

In the remainder of this section we describe how we implemented such a feedback mechanism by means of the issue tracking system Bugzilla for a Web application for internal use, the lessons we learned, and further implications. The experience and data presented in this section is a summary of the project's final report, the usage analysis of Bugzilla, and the notes from periodical project and process reviews conducted by quality management.

4.1 Enabling End-User Involvement

At a first glance, the approach to evolve an application based on user feedback does not seem much different to any conventional (iterative) software development process that includes a beta testing and stabilization phase. For us, this was an advantage as no major changes to the underlying processes and no new methods or tools were necessary. However, there exists an inherent difference regarding the nature of the feedback that is often overlooked, especially by development: A comparison of conventional and evolutionary development shows a significant shift in the importance of the different types of maintenance. While conventional maintenance approaches tend to emphasize corrective maintenance, evolutionary development is characterized by a focus on adaptive maintenance [23]. This is an important finding,

as it underlines that evolutionary development focuses on (new) requirements rather than on errors in existing functionality. Evolutionary development should therefore primarily address the problem that users are not able to fully specify their requirements at the beginning of the projects and, even if they do, requirements may still change over time as business procedures and technologies evolve. Furthermore, evolutionary development is often the preferable way to develop a common understanding about the goals of the project and the needs of the users [6]. "In other words, the specification of the system emerges from the design, rather than preceding and driving the design." [24]

Consequently, we used the maintenance infrastructure, namely the issue tracking system Bugzilla described in section 3, primarily to collect and maintain the continually evolving requirements of the application. Thereby, using the existing maintenance infrastructure for evolutionary development provided several benefits: First of all, a proven tool with a clear and systematic process was used. All the involved developers were familiar with the infrastructure. Thus, similar to the advantages initially mentioned, no radical changes of the underlying development process were necessary and overhead costs could be kept at a minimum. Second, as described below, the integration of the Bugzilla issue tracking system turned out as an effective vehicle to interact with end-users and to collect requirements. Third, it supported collaborative work throughout development as well as measuring and tracking the ongoing development effort.

To establish a single channel for end-user feedback, we made our maintenance infrastructure an integral part of the Web application. Beside the solution presented here, we also experimented with various other approaches that are commonly suggested in the literature for end-user integration (see [23]). Useful to stimulate feedback and discussion are, for example, annotation mechanisms to Web pages or Wiki-style authoring [25]. However, these approaches focus mainly on content – the data, its structure and presentation. We required a more general feedback channel amenable to the different functional and non-functional aspects of Web applications (e.g., correctness, security, usability, or performance issues related to functionality, content, or infrastructure aspects) as described in [26]. Nevertheless, an easy to use and seamless integrated solution was required. When, in a first step, Bugzilla was integrated solely via a link, we experienced little acceptance of the feedback mechanism because of major usability deficiencies: The access to Bugzilla was rather uncomfortable due to switching to a (very) different application. The users had to logon again and were confronted with the (non-intuitive) user interface of Bugzilla, data had to be copied manually into Bugzilla's report form, and the users were burdened with the concepts of Bugzilla's issue tracking process.

To overcome these drawbacks, we integrated a simple issue report and enhancement request form into the Web application, which forwarded the data entered by the user to the issue tracking system Bugzilla. Even though the form resembled the structure of a conventional Bugzilla issue report, it was aligned with the style of the Web application (see Figure 2 for an example). A few additional field values, e.g. *enhancement request* for the field *issue type*, had to be added to support not only issue tracking but also requests for enhancements and new features. Hence, the users were relieved from directly accessing the Bugzilla interface and we could still utilize our preferable issue tracking system in the background. Furthermore, the form was made available within a single click from every page of the Web application. On accessing the form, the context of use was analyzed and the fields of the form were

automatically pre-filled with meaningful default values to minimize the amount of data the user had to enter. For example, the user's browser type was identified to easily reproduce issues related to the different behavior of the various Web browsers and the link referring to the page plus the name of the application module the user last accessed were automatically suggested.

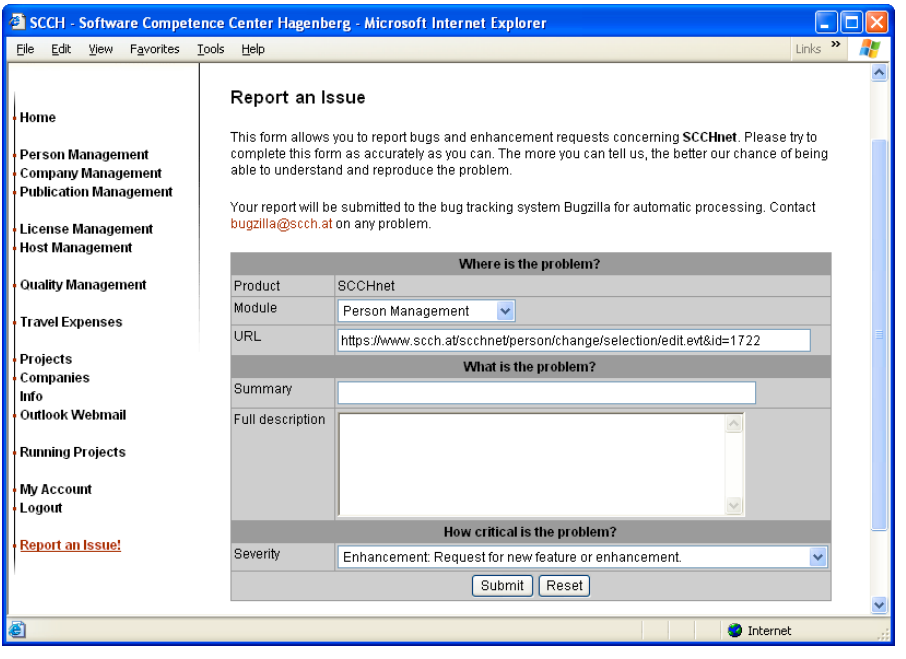

Fig. 2. Issue report and enhancement request form

To get a first version of the Bugzilla integration setup up and running, we relied on the Bugzilla email gateway. Technically spoken, the report form is submitted to a Java mailing servlet[1] part of our Web application. The servlet assembles an email containing the data from the report form structured according to the requirements of the Bugzilla email gateway and sends it to a dedicated email account on the Bugzilla server. There, the email is parsed, the report data is extracted and entered into the issue tracking system. Thus, we can easily access Bugzilla and initiate the issue tracking workflow. Bugzilla auto-assigns the report to the developer in charge for the affected module, tracks all changes, and sends email notifications on updates. Figure 3 illustrates the interaction between the user's Web browser, the Web application, and Bugzilla integrated via the email gateway.

[1] The Java source code of this servlet can be obtained from the authors.

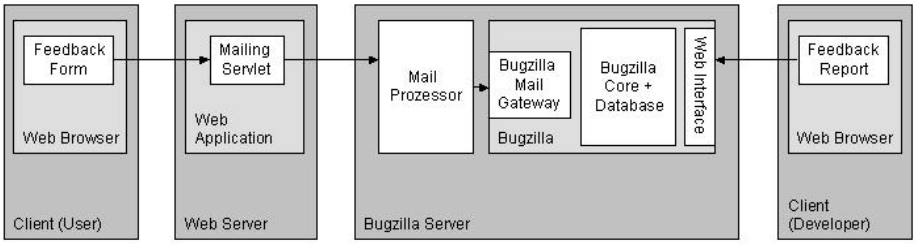

Fig. 3. Bugzilla Integration in Web Applications

4.2 Experience and Lessons Learned

The experience and lessons learned we present here are derived from integrating Bugzilla as part of an internal Web-based application. The main results were largely equivalent to the experience we made in consecutive projects. The target audience of the application was a group of experienced users from within our organization as well as partner companies at distributed locations.

The possibility to submit comments was well accepted by end-users. Even though no particular measures were taken to encourage user feedback, we received more than 100 reports within one month from a group of about 50 potential users. Many of these reports would otherwise have been submitted via email or through informal personal communication, e.g. by phone or in hallway conversations. Or, more likely, they would have simply been omitted. The development team greatly benefited from the issue tracking capabilities of Bugzilla that supported a quick triage as well as the systematic management of the submitted reports.

The total number of reports was distributed as follows: 17% critical errors, 41% errors of normal severity, and 42% improvement suggestions and enhancement requests. Critical and normal errors together made 58% of all collected reports, despite the fact that we estimated a much higher potential for improvement comments and enhancement requests than for error reports. The reason seems to be that the motivation to react and comment on problems increases with the perceived severity of the problem. From personal feedback we know that, in some cases, users even did not report obvious errors when they found easy workarounds. Thus, incentives may be necessary to motivate feedback and, furthermore, the costs for providing feedback must be kept at a minimum. The ease of use of the feedback mechanism is of uttermost importance. A short and concise form and a careful pre-selection of default values are required as lengthy forms to fill in deter many users.

However, including user session data in reports may possibly conflict with privacy, a major concern of many users. Therefore, we used only a few pre-filled entries and we did not log any usage data related to the reported issues. Rather, the user was asked to give a description of the issue and the necessary steps to reproduce it. The optional statement of an email address allowed us to contact the user in case of further questions. Besides, we were able to offer the user email notification on certain events, e.g., when the reported issue had been accepted or resolved. So we involved the end-users into the development process instead of frustrating them with the feeling as if reporting "into a black hole".

Two important lessons we learned were, first, that we could not expect users to positively confirm correctly working solutions. To some extent, the absence of comments may only indicate a correct working solution. Second, since we provide the feedback form for existing functionality only, users did not come up with new features or any "great new ideas". Hence, evolutionary Web development still requires a great deal of planning ahead and the proactive development of new functionality, e.g. motivated by competitive analysis [2]. In addition, careful analysis of the end-users' reports and a "creative mindset" are necessary to stimulate new ideas within the development team. A shared vision serving as overall guidance for the project team helped to direct these ideas into the right direction.

Furthermore, reports have to be brought in relation to overall usage statistics to evaluate the relative significance of the feedback. Other sources that provide an insight into the attitude and behavior of users should be included in the data collection to support the reasoning about improvements and to strengthen the conclusions drawn. Therefore, we are currently extending our feedback mechanism to allow real-time analysis of application logs as described in the following section.

4.3 Towards a Bugzilla Web Service

The approach described in section 4.1 allows a quick and easy integration of Bugzilla into any Web application. However, the Bugzilla email gateway is an extension to the Bugzilla project with some limitations regarding functionality and flexibility. It lacks full access to all of the features of Bugzilla. Currently, only a one-way access to the issue tracking system is possible – issues can only be submitted from the Web application to Bugzilla. Full access should permit creating new issues, adding comments, changing issue related metadata (e.g. priority or severity), querying for existing issues, or voting for issues (so the frequency of issues can be documented and the number of duplicate reports can be reduced).

In Figure 4, we outline an integration concept based on a SOAP (Simple Object Access Protocol) interface [27] to Bugzilla. Thus, the SOAP interface provides access to all of the Bugzilla functionality via a Web Service and enables communication in both ways – from the Web application to the Bugzilla issue tracking system (e.g. to submit an issue) and from Bugzilla to the Web application (e.g. to return the status information of an issue). The example of an automated issue reporting system, part of a Web application, illustrates the advantages of such an interface to Bugzilla. We are currently using the Log4J framework [28] for application logging. So, a log recorder/analyzer can be used to observe and analyze the activity and state of the Web application and react to certain events, e.g. application errors, in real-time by submitting appropriate reports directly into the issue tracking system without further human interaction.

We are currently evaluating the Jagzilla Web Service API, part of the Jagzilla System [29] as a way to realize an integration of Bugzilla into Web Applications like depicted in Figure 4. Jagzilla provides a simple re-implementation of the core functionality of Bugzilla while relying on the original database of the Bugzilla issue tracking system. By the date of writing, the Jagzilla Web Service API is still in early alpha status.

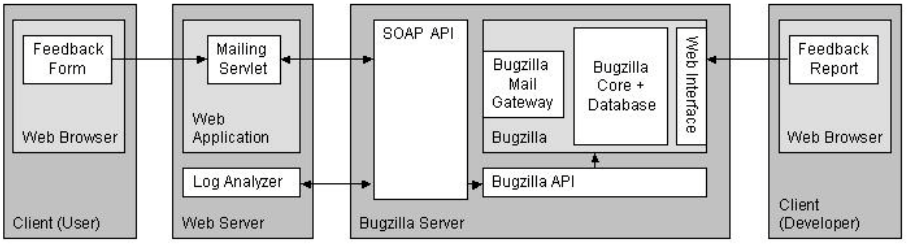

Fig. 4. Bugzilla integration based on Web Services

5 Related Work

Being in a constant state of flux, the adaptation and continuous evolvement of Web applications as an answer to rapidly changing requirements has long been a topic in Web engineering. Various approaches to deal with this problem have therefore been proposed. In this section, we give an overview of related work, which we consider as a prerequisite, a logical next step, or a useful basis in combination with the approach that has been described in this paper.

Modeling and design. Web applications must be built with a mindset towards frequent changes throughout the lifecycle. Development has to establish a basis for applications that can be modified, fixed, or maintained at little cost of time or money. Common approaches are modular architectures, e.g. based on re-useable components, Web application frameworks, or code generation from domain models. Various design methodologies support these strategies. An overview of methods and appropriate tools specific for Web applications can be found in [30], [20], [23], or [31]. From the maintenance perspective, re-engineering and reverse engineering are of particular interest. Methods and tools based on modeling and design strategies have been developed that support maintenance and evolutionary development. (Prominent examples are STRUDEL [32], ReWeb [33], WARE [34], or Rigi [35].) Evolutionary prototyping-based development, furthermore, has a rich tradition in User-Centered [36] and Participatory Design [37]. Participatory Design explicitly considers social, ethical, as well as political viewpoints in addition to technical issues and takes a "democratic" approach to system design by actively involving users. From our point of view, modeling and design techniques are a prerequisite for effective maintenance and, thus, complementary to our evolutionary development approach.

Adaptive and customizable Web applications. With the transition to dynamic Web sites and Web-based applications, where Web pages can be generated on the fly, the idea of self-adaptation in response to external changes emerged. Several approaches have been published that demonstrate this idea. For an overview of this topic please refer to [23], [38], [39], or [40]. In contrast, our approach requires a human (e.g. a programmer or product manager) to look at all submitted feedback reports, to validate them, and to initiate proper measures. The necessary changes – the adaptation and the extension of the Web application – are done by development, not by the system itself. However, an integration based on Web Services, as described

above, may provide enough flexibility to serve as a first step towards adaptive applications.

Agile development. Agile development methods have successfully found their way into Web development processes [41, 42] and, conversely, agile processes have been developed for Web engineering [43]. There are a number of rapid development software methodologies all referred to as "agile development" [44]. Many of these methods explicitly address the requirements of an evolutionary Web development process as stated in Section 2. They offer, for example, rapid prototyping, comprehensive end-user involvement, support for frequent changes (e.g. by unit testing), and they emphasize low administrative overhead. As we do not prescribe any development process in our approach, the workflow and tools proposed in this paper may be combined with most of the agile development methods. Some processes, e.g. Feature-driven Development [45], even suggest similar ideas to cooperate with stakeholders or to organize work.

6 Conclusions and Future Work

Within this paper we first presented a requirement's perspective for an ideal Web development process. In order to address and fulfill the presented requirements, we then elaborated on how we successfully implemented a process for evolutionary development based on conventional maintenance concepts. Our pragmatic approach has the advantage that established development processes can be retained, no radical changes are necessary, and existing tools can be utilized. Furthermore, while maintenance such as adding new features or correcting errors may be initiated by the development team, in practice maintenance is often triggered by feedback, both direct and indirect, from end-users. The seamless integration of the issue tracking system Bugzilla into the Web application permits easy and flexible involvement of end-users and encourages the feedback necessary to adapt and grow Web applications. Thereby, the workflow implied by Bugzilla supports an ordered and systematic development process (as demanded in [9]), regardless of which particular process model is actually used. From our point of view, agile development processes conveniently harmonize with Web development and prevent ad-hoc approaches and undisciplined hacking (see also [46]).

In a next step, we will consider to integrate the interface to the Bugzilla issue tracking system into an existing Web application framework (e.g. Struts). This would considerably ease the automated collection of end-user feedback and, thus, further support the development of evolving Web applications as described in this paper.

Acknowledgements. This work has accrued in the framework of the K-plus Competence Center Program, which is funded by the Austrian Government, the Province of Upper Austria and the Chamber of Commerce of Upper Austria.

References

1. Truex D.P., Baskerville R., Klein H.: Growing Systems in Emergent Organizations. Communications of the ACM, vol. 42, no. 8, August 1999, pp. 117-123
2. Norton K.S.: Applying Cross-Functional Evolutionary Methodologies to Web Development. pp. 48-57, in: [13] pp. 48-57
3. Pressman R.S.: Software Engineering: A Practitioner's Approach. 5.Ed., McGraw-Hill, 2001
4. Standish Group International: Extreme Chaos. Update to the CHAOS Report, Standish Group International, Inc., 2001
5. Emam K.L., Quintin S., Madhavji N.Z.: User Participation in the Requirements Engineering Process: An Empirical Study. Requirements Engineering, 1996 (1), pp. 4-26
6. Lowe D.: Web System Requirements: An Overview. Requirements Engineering Journal, 8 (2), 2003, pp. 102-113
7. Pomberger G., Blaschek G.: Object Orientation and Prototyping in Software Engineering. Prentice-Hall, 1996
8. Powell T.A.: Web Site Engineering: Beyond Web Page Design. Prentice-Hall, 1998
9. Ginige A., Murugesan S.: The Essence of Web Engineering: Managing the Diversity and Complexity of Web Application Development. IEEE Multimedia, vol. 8, no. 2, April-June 2001, pp. 22-25
10. Baskerville R., Levine L., Pries-Heje J., Slaughter S.: Is Internet-Speed Software Development Different?. IEEE Software, vol. 20, no. 6, Nov.-Dec. 2003, pp. 70-77
11. Pressman, R.S.: What a Tangled Web We Weave. IEEE Software, Vol. 17, No.1, Jan-Feb 2000, pp. 18-21
12. Reifer D.J.: Ten Deadly Risks in Internet and Intranet Software Development. IEEE Software, vol. 19, no. 2, March-April 2002, pp. 12-14
13. Murugesan S., Deshpande Y. (Eds.): Web Engineering – Managing Diversity and Complexity of Web Application Development. LNCS 2016, Springer, 2001
14. Kappel G., Pröll B., Reich S., Retschitzegger W. (Eds.): Web Engineering – Systematische Entwicklung von Web-Anwendungen. d-punkt Verlag, 2003
15. Lowe D., Hall W.: Hypermedia & the Web. An Engineering Approach. Wiley, 1999
16. Concurrent Versions System. Web (http://www.cvshome.org)
17. Dreilinger S.: CVS Version Control for Web Site Projects. Technical report, 1999 (http://www.durak.org/cvswebsites/howto-cvs-websites.pdf)
18. Bugzilla Bug Tracking System. Web (http://www.bugzilla.org)
19. Allen M.: Bug Tracking Basics – A beginners guide to reporting and tracking defects. STQE Magazine, vol. 4, iss. 3, May-June 2002, pp. 20-24
20. Lowe D.: A Framework for Defining Acceptance Criteria for Web Development Projects. pp. 279-294, In: [13], pp. 279-294
21. alphaWorks: ServletManager. IBM, 2002 (http://alphaworks.ibm.com/tech/servletmanager)
22. The Apache Struts Web Application Framework. Web (http://jakarta.apache.org/struts)
23. Scharl A.: Evolutionary Web Development. Springer, 2000
24. Lowe D., Eklund J.: Client Needs and the Design Process in Web Projects. Proc. of the 11th Int. World Wide Web Conference, Hawaii, 2002
25. Leuf B., Cunningham W.: The Wiki Way – Quick Collaboration in the Web. Addison-Wesley, 2001
26. Ramler R., Weippl E., Winterer M., Schwinger W., Altmann J.: A Quality-Driven Approach to Web-Testing. Proc. of the 2nd Int. Conf. on Web Engineering, Santa Fe, Argentina, Sept. 2002
27. Gudgin M., Hadley M., Mendelsohn N., Moreau J.J., Nielsen H.F.: SOAP Version 1.2 Part 1: Messaging Framework. W3C Rec., June 2003 (http://www.w3.org/TR/SOAP)
28. Logging Services - Log4J. Web (http://logging.apache.org/log4j)

29. Jagzilla Home Page. Web (http://jagzilla.sourceforge.net)
30. Christodoulou S.P., Styliaras G.D., Papatheodrou T. S.: Evaluation of Hypermedia Application Development and Management Systems. Proc. of the 9th Conf. on Hypertext and Hypermedia, Pittsburgh, US, 1998
31. Schwinger W., Koch N.: Modellierung von Web-Anwendungen. In. [14], pp. 49-75
32. Fernandez M., Florescu D., Kang J., Levy A., Suciu D.: STRUDEL: A Web-site Management System. Proc. of the Int. Conf. on Management of Data, Tucson, US, May 1997
33. Ricca F., Tonella P.: Understanding and Restructuring Web Sites with ReWeb. IEEE MultiMedia, vol. 8, no. 2, April-June 2001, pp. 40-51
34. Di Lucca G.A., Fasolino A.R., Pace F., Tramontana P., De Carlini U.: WARE: A Tool for the Reverse Engineering of Web Applications. Proc. of the 6th Eur. Conf. on Software Maintenance and Reengineering, Budapest, Hungary, March 2002
35. Martin J., Martin L.: Web Site Maintenance With Software-Engineering Tools. Proc. of the 3rd Int. Workshop on Web Site Evolution, Florence, Italy, Nov. 2001
36. Vredenburg K., Isensee S., Righi C.: User-Centered Design: An Integrated Approach. Prentice Hall, 2001
37. Muller M.J., Kuhn S.: Participatory design. Communications of the ACM, vol. 36, no. 4, June 1993, pp. 24-28
38. Perkowitz M., Etzioni O.: Adaptive Web sites. Communications of the ACM, vol. 43, no. 8, August 2000, pp. 152-158
39. Brusilovsky P., Maybury M.T.: From Adaptive Hypermedia to the Adaptive Web. Communications of the ACM, special section: The Adaptive Web. vol. 45, no. 5, May 2002
40. Patel N.V. (ed.): Adaptive Evolutionary Information Systems. Idea Group Publishing, 2002
41. Engels G., Lohmann M., Wagner A.: Entwicklungsprozess von Web-Anwendungen. In [14], pp. 239-263
42. Hansen, Steve: Web Information Systems:- The Changing Landscape of Management Models and Web Applications. Workshop on Web Engineering, Proc. of the 14th Int. Conference on Software Engineering and Knowledge Engineering, Ischia, Italy, July 2002
43. McDonald A., Welland R.: Agile Web Engineering (AWE) Process, Department of Computing Science Technical Report TR-2001-98, University of Glasgow, Scotland, December 2001 (http://www.dcs.gla.ac.uk/~andrew/TR-2001-98.pdf)
44. Abrahamsson P., Warsta J., Siponen M.T., Ronkainen J.: New Directions on Agile Methods: A Comparative Analysis. Proc. of the 25th Int. Conf. on Software Engineering, Portland, US, May, 2003
45. Coad P., Lefebvre E., De Luca J.: Java Modeling Color with UML: Enterprise Components and Process. Prentice Hall, 1999
46. Boehm B.: Get Ready for Agile Methods, With Care. IEEE Computer, vol. 35, no. 1, January 2002, pp. 64-69