J. Software Engineering & Applications, 2010, 3, 882-889
doi:10.4236/jsea.2010.39103 Published Online September 2010 (http://www.SciRP.org/journal/jsea)

Towards Lightweight Requirements Documentation

Zheying Zhang[1], Mike Arvela[2], Eleni Berki[1], Matias Muhonen[3], Jyrki Nummenmaa[1], Timo Poranen[1]

[1]Department of Computer Sciences, University of Tampere, Tampere, Finland; [2]Futurice GmbH, Taubenstraße, Berlin, Germany; [3]Nomovok Ltd., Keilasatama, Espoo, Finland
Email: {zheying.zhang, eleni.berki, jyrki.nummenmaa, timo.poranen}@cs.uta.fi, mike@arvela.net, matias.muhonen@nomovok.com

ABSTRACT

Most requirements management processes and associated tools are designed for document-driven software development and are unlikely to be adopted for the needs of an agile software development team. We discuss how and what can make the traditional requirements documentation a lightweight process, and suitable for user requirements elicitation and analysis. We propose a reference model for requirements analysis and documentation and suggest what kind of requirements management tools are needed to support an agile software process. The approach and the reference model are demonstrated in Vixtory, a tool for requirements lightweight documentation in agile web application development.

Keywords*: *Lightweight, Requirements Documentation, Requirements Management Tool, Agile Software Development

1. Introduction

Many agile methods emphasize working code [1,2] and working documentation (e.g. in the case of SCRUM), too. Agile methods concentrate on adding value to business and agile software development is an intensive communication process with users. The requirements analysis and documentation activities, however, are often carried out intuitively. Instead of a full requirements document, the most common form of user requirements includes a user story or a use case, which tells a story on how the user completes a meaningful task in interaction with the system to be built. Working on the increments based on user stories involves interaction with the end-user, where more information comes in the form of feedback from the user. This feedback contains changes, additions and refinements on requirements.

Well-defined requirements have traditionally been seen as a critical factor behind software project success [3-6]. Agile methods also emphasize the user requirements, but they are less documentation-centric. It is, however, important to consider how and to whom the requirements are presented and used. We want to develop a user-centric reference model to capture the requirements analysis and documentation environment; this can improve user participation and requirements representation, while supporting agile ways-of-working and values. The paper presents a lightweight requirements documentation environment by proceeding as follows. Initially we present the rationale of having a requirements documentation phase and comment on the three dimensions of requirements engineering (RE). The latter grounds the discussion of how and what might make the requirements documentation a lightweight process and, thus, more agile. Based on this discussion, we propose a reference model for requirements analysis and documentation, and further discuss what the requirements management tools should be like for agile projects. We finally propose Vixtory [7], a prototype tool for agile web application development, as an example to demonstrate the lightweight requirements documentation environment supported by our reference model.

2. The Three Dimensions of Requirements Engineering

Pohl [8] describes the RE activities and their goals along three orthogonal dimensions: specification, representation, and agreement. The framework assumes that there are three major facets of the RE process, namely documenting the requirements in a more complete manner, with more formality, and more consensus among stakeholders [8,9]. The *specification dimension* deals with the degree of requirements understanding at a given time [8]. Completeness and understandability of knowledge form the main concerns in this dimension. The *representation*

dimension copes with the degree of formality of knowledge expression [8]. In general, requirements can be documented in three types of representation: informal representation (e.g. natural language), semi-formal representation (e.g. ER diagram, state diagram, etc.), and formal representation (e.g. mathematics expression). From the informal presentation to the formal one, requirements documents are shifting from the user-oriented ones to the system-oriented ones. The *agreement dimension* deals with the degree of agreement reached on a specification. It focuses on a variety of views of the same system from heterogeneous groups of stakeholders, and emphasizes the common agreement.

3. The Meaning of 'Lightweight' Requirements Documentation

Requirements documentation provides a means of communicating between diverse stakeholders and achieving common agreement on the future software artifacts description [3]. Requirements elicitation is perceived as an essence for a software development project, but the requirements analysis, documentation, validation and maintenance are very tedious processes. Many researchers claim that most requirements specifications found in industry nowadays still include many poor-quality requirements [6,7], even though there are so many different techniques to ensure the requirements' quality. Poor requirements form an important reason that causes the failure of many IT projects [5,10].

An ideal requirements document shall be correct, complete, consistent, precise, testable, and traceable [4]. In practice, however, it is hard to address all requirements up front, and to maintain a correct and consistent document throughout the project in an ever-changing environment. In agile software projects, in particular, the traditional requirements process is replaced with iterations and increments, and the documentation is replaced with user stories, working software, and changing requests. We shall ensure that stakeholders express, understand, document, use, and maintain requirements in a correct and easy way. The requirements gathering and agreement process should shift from documentation to communication efforts. A more narrative and context-specific approach should be adapted to improve the requirements analysis process, and at the same time, keep it lightweight. The latter indicates the environment which deploys the available resources to effectively support the communication and the consequent analysis and documentation. The meaning of lightweight is next discussed along three important dimensions.

From the perspective of specification, requirements documentation is an ongoing process, and the details can be elaborated just in time. Requirements need not be fully specified up front, at a very early stage of the project, when many aspects are unknown and needs cannot yet be expressed, consistently and correctly, to say the least. We hereby agree with other researchers [3,12,13] supporting that requirements development is an ongoing process throughout a software development project. Meanwhile, there is no point in specifying highly detailed requirements before software or at least prototype development even starts. Software requirements can be elaborated at the right time when they are selected for implementation. This is natural as the application domains of the real world, to which the software targets, is subject to change. Furthermore, users change their understanding towards their requirements and needs as the development proceeds in new software releases that need feedback.

From the perspective of representation, prototypes or working software improve the requirements understandability by providing a context realism representation. To get some grip of the concept lightweight in the representation dimension, we make the following division of ways of working to document requirements. In the traditional waterfall model, the requirements have been documented before the actual software is being built. A typical way to express requirements is textual [7,14]. Different from the traditional waterfall model, prototyping [3,14] means building software to mimic the behavior and functionality of the target software to be built. The prototypes are used to validate requirements, but, they could also be seen as specification techniques, where requirements can be elicited upon and attached to what can be called prototyping software. Similarly, it is also possible to attach requirements to working software. This has become more of an option in incremental software development, where software is built on top of existing increments, as is typically done in agile software development. The working software can provide users a real and actual representation of the requirements. It can be regarded as a starting point to elicit and refine requirements rather than an end of a development cycle. Such a context enriched representation makes a smooth transformation from the high level requirements description to the detailed implementation, and enhances the clarity and understandability of requirements. At the same time, the requirements are not rigid with a specific form of representation, which forms a flexible and lightweight process to represent requirements.

From the perspective of agreement, timely feedback on small releases of working software supports the evolution from the individual views to a common agreement. Incremental development that utilizes prototypes or other prototype-like software artifacts, e.g. working software,

gives us a possibility to attach a part of the requirements to existing increments. This allows stakeholders not only to perceive the target applications, but also to present their individual views and wishes based on the existing version. With the frequent releases, stakeholders can review the working software, adjust their understanding of the target application, and provide timely feedback as the requirements for the follow-up iterations. It flexibly supports the evolution from the personal views to a common agreement on the target system, and avoids the aforementioned problems of prototyping methods: The software does not give a promise of a functionality which may be incorrect and also the design is "as-is".

4. A Reference Model for Requirements Documentation Environments

The most essential aspects in a requirements analysis and documentation environment can be captured in a reference model. The reference model has ER like notations. The rectangles represent entities which facilitate requirements documentation. They are integrated together through a number of traceability links, represented as arrows. As shown in **Figure 1**, the model consists of three basic entities: requirements, external documents, and software artifacts. These are the most essential artifacts found in every software project, and documented in the supporting tools. Entities are interlinked with each other through a number of traceability links. Each entity and links can be specialized and instantiated to create organization or project specific requirements documentation environments. Since lightweight documentation is our main concern and target in this study, the instances and attributes, which reflect the nature of agile software development approaches, are marked in bold. In the following section, we will discuss the reference model along different dimensions of lightweight requirements documentation.

4.1. Requirements

Requirements comprise the specifications of the services that the system should provide, the constraints on the system and the background information that is necessary to develop the system [16]. They are typically represented in a natural language, supplemented by diagrams such as the use case diagram, the data flow diagram, etc. Requirements are documented with attributes such as creation date, version number, author, description, priority level, status, etc.

Instead of limiting a requirements specification to a single and rigid representation, the informal representations of users' conception of their system such as user stories [1,2], use cases, and scenarios can be elaborated

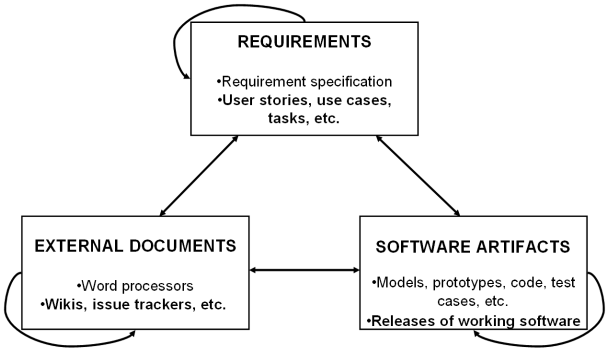

Figure 1. A reference model of a requirements analysis and documentation environment.

and attached to the working software at a right time, which can make the requirements documentation process intuitive and encourage customers' participation. User stories include a set of small sentences that express the user needs, in her/his own words [2]. The description tells a story on how the user completes a meaningful task in interaction with the system to be built. When a user story is selected for implementation, it can be further elaborated by the developers in their preferable forms.

Requirements can be documented in different levels of detail. The high abstraction level requirements, such as user stories, are documented by interacting with customers and by means of experimental use of prototypes or working software. They are customer's actual needs. When the user stories are selected for implementation, they are refined and adjusted into detailed tasks, and implemented and evaluated within the same iteration. Such divide-and-conquer tactics isolate the customers from complex technical implementation, while enable them to provide timely evaluation and feedback of accumulated implementations.

4.2. External Documents

External documents represent the documents which are not stored in any requirements management tools (RMTs). In traditional software development process, they typically describe and contain the requirements specified in general-purpose document tools, modeling tools, etc. These documents are structured ones and easy to create, but static. It is hard for different stakeholders to work in collaboration on the same document, and to document and exchange ideas in a lightweight process.

An agile principle is close collaboration between developers and customers. A lightweight documentation needs a platform that can support effective and efficient collaboration among an often large number of diverse needs and requirements stakeholders. The model we propose enhances collaboration by adding the instance of generic collaborative platforms, such as wikis [17,18]

and issue trackers. These can provide a flexible way of open review, elaboration, refutation, or refinement with a discussion thread. Further the discussion and communication comments give rise to the development of narrative descriptions of the features and requirements of the software under development. This can reduce the details needed in the requirements documentation, as the detailed contextual information can be linked to the discussion on the collaborative platform. The process can be adapted to support active stakeholders participation in requirements elicitation and documentation [18].

4.3. Software Artifacts

Software artifacts represent the final or interim byproducts during the development of software. Examples include specifications, prototypes, code, testing cases, as well as working software released at the end of each iterative process. They are connected with requirements through a variety of traceability links, which provide contextual information within the development team to support software development activities, such as change impact analysis, testing, maintenance, project tracking, etc.

As one of the agile principles is 'frequent releases', the development team can deliver working software to the customers for their experimental use and for feedback [1]. The experimental use process expands the contextual information within the development team to that between the customers and the developers. It facilitates customers to explore the real working product, and to provide individual feedback before the final delivery, which reduces the uncertainty between the development team and the customers. Being different from a prototype, which may represent some compromise from the final production design, the working software provides customers with an actual production design, and, thus, eliminates the risks of misunderstanding and misleading. The context specific information of working software is much more real than that of a prototype. Furthermore, compared with the throw-away prototype approach [3,14], the practice of short release of working software saves time and other resources in a software project. Therefore, an easy use of the traceability link among requirements and the working software is necessary to facilitate the communication of ideas and prompt feedback.

4.4. Traceability Links

Traceability links connect the instances of every component to provide contextual information on the target system [13]. They present the relationship of entities instantiated from the same element, such as the elaboration relationship between a high level user story and a list of low level tasks, the validation relationship between a test case and a segment of the software, or the hyperlinks available in wikis or any other collaborative platforms. Furthermore, the traceability links between different elements allow developers to trace code back to the conversation from which the artifact came, back to the user story, and finally to its initial requirements [19]. They also facilitate the customers to be involved in the development process by tracing between the user stories to the working software. Consequently, there are two categories of traceability links in the requirements documentation environment: links within tools and links between tools. It is undoubted that a set of tools are deployed within a project to support the development activities from different aspects. In general, each tool can provide some sort of traceability information within the use of tool, such as the aforementioned elaboration or validation links. Besides, it is necessary for an agile project to have an integrated environment by using hyperlinks to connect the requirements and other information scattered in different tools. Examples include the traceability link between the prototyping software and the RMTs, between the collaborative platform and the RMTs or a CASE tool. These hyperlinks ease the flexible documentation and use of requirements and the related information [20], which reflects the goal of the reference model.

Access to documented traceability provides different levels of context realism. It is indeed very valuable. However, the manual burden directly contradicts with the agile principles. Developers are often reluctant to participate in the effort of documenting traceability information [13,21,22]. On the basis of the existing tools, a solution to connect the scattered information manually or automatically into the iteration is very important. The next section elaborates further on this aspect.

5. Tools Supporting Lightweight Requirements Documentation

There is a number of tools supporting requirements analysis and documentation. A tool survey conducted by INCOSE [23] compares the features of over forty different RMTs from 2004 to 2009. These tools support the requirements documentation process and, clearly, influence the quality of the documentation. Before discussing the need for lightweight requirements documentation tools, we will have an overview of the features of tools that support the traditional requirements documentation phase.

5.1. Classification of Requirements Documentation and Management Tools

We attempt to broadly classify existing RM tools into four groups: general-purpose document tools, collabora-

tive tools, RMTs, and prototyping tools.

The *general-purpose document tools* mainly include office suites such as MS Office, Open Office, Lotus SmartSuite, etc. These are not too specialized, and many users are acquainted with since they are easy to adapt to the needs of different development environments. Surveys report that these general-purpose document tools, though not sophisticated, in practice are helpful with requirements documentation [5,6,24-26]. Though the non-specialized features can be considered as a great merit, it is difficult to support specific RE activities and ensure the quality of the derived documents. .

The *collaborative tools* offer a flexible platform that can involve a number of diverse users in common tasks to achieve their goals and for collaborative editing of contents. Wikipedia is an example for creating an encyclopedia openly and collaboratively by volunteers from all around. According to the level of collaboration, there are different categories of these tools ranging from a simple information sharing application (e.g. online chat, wikis, etc.) to sophisticated systems that facilitate group activities (e.g. knowledge management and project management systems). Instead of facilitating documentation, these tools provide a lightweight solution to creating, editing, sharing, and discussing information; the latter obviously improve the communication and collaboration for requirements analysis.

RMTs are dedicated to manage the large amount of data collected during the RE process and the volatility of the requirements [3]. There are many commercial RMTs such as DOORS, Requisite pro, CaliberRM, etc. Typically, these tools collect the requirements in a database and provide a range of facilities to access the information on the requirements. These facilities include requirements browsing, requirements converting, report generating, requirements tracing, change control, etc. The RMTs that support formal requirements representation can also facilitate requirements consistency checking and semantics verification [27]. Such tools aim at technical users, and provide a comprehensive environment to support the different dimensions of RE process. Empirical studies [25] support that RMTs provide better coverage of the RE process and the quality of requirements documentation. On the other hand, many surveys [5,6, 24-26] report that the mainstream practice relies on office and modeling tools rather than RMTs. Survey reports contradict on the industrial use and the rationale of RMTs.

The *prototyping tools* are specific tools, which rapidly represent, build, and realize important aspects of a software-based system. The prototype serves as an experimental system to demonstrate requirements and collect stakeholders feedback. Prototyping tools range from simple ones that develop a mock-up system to specialized ones that create interactive wireframes for websites and desktop software, and design user interfaces with high functionality. Examples include Axure RP, ProtoShare, etc., which generate web-based prototypes. Besides, some general-purpose CASE tools provide good support for prototyping for user interfaces and web design, such as the graphic design tools (e.g. Illustrator or Adobe Photoshop), the diagramming tools (e.g. Visio or SmartDraw), and the visual and text based HTML tools (e.g. FrontPage, Dreamweaver, etc.). Instead of specifying and managing requirements, the prototyping tools focus more on providing stakeholders with a real experimental system, which increases requirements understandability and avoids requirements creep and rework.

In addition to these four categories of tools for requirements documentation, there are also agile project management tools, such as Rally, Scrumworks Pro, which facilitate backlog (requirements) editing and report generating. All these tools provide support for requirements documentation in some aspects of the reference model depicted in **Figure 1**. In general, RMTs, as well as all other requirements documentation tools, provide support for requirements specification in different levels of formality. Besides, the collaborative tools provide more flexible support for the external documents, while the prototyping tools can provide links between the software artifacts and the requirements. Obviously, none of them can cover the components specified in the reference model of **Figure 1**.

The purpose of requirements documentation is communication among a number of stakeholders. The general-purpose document tools have widespread availability. They, however, lack adequate support for communication and collaboration in the RE process. The collaborative tools compensate for the deficiency of collaboration in the general-purpose documentation tools, but lack enough support in context enriched representation and just-on-time requirements documentation. The RMTs tools over-emphasize the specification and representation dimension of the RE process, i.e. the bureaucratic and rigid support for the RE process, but do not facilitate a close and smooth interaction between developers and customers [21,26]. The communication factor is lost. The prototyping tools, on the other hand, offer users the actual prototype for experimental use and feedback, but, most of them, lack necessary features that facilitate just-on-time specification. In a summary, **Figure 2** illustrates the tools previously discussed and their support of the goals set within the three dimensions of lightweight requirements documentation, as discussed in the beginning of this work.

Consequently, in order to better support requirements documentation, a tool should capture the three important dimensions of the RE process, as outlined in the context

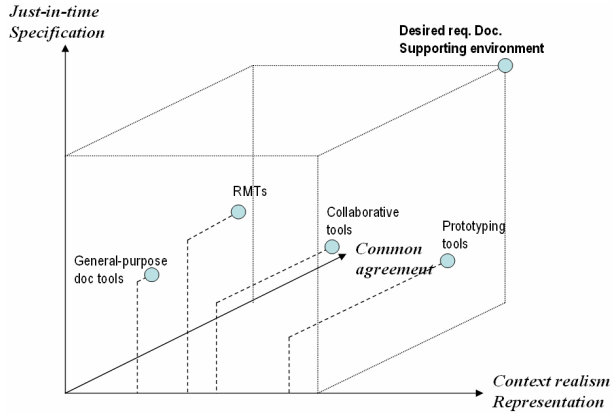

Figure 2. Tools within the three dimensions of lightweight requirements documentation.

of this paper. A single tool cannot provide all the desired features for a lightweight requirements documentation process. The latter should be facilitated by a set of simple, intuitive, and widespread availability tools [20,28], which could easily and flexibly be integrated into the development environment.

5.2. Vixtory–A Target Application Based Requirements Documentation Tool

As shown in **Figure 2**, prototyping tools are the ones most close to the desired lightweight requirements documentation. The target is, thus, to improve the specification dimension of such tools and provide users with an actual experimental use of the target application. Motivated from these needs and the tools features discussion, we developed a requirements management tool for agile web application development, namely Vixtory [7]. It provides a lightweight and less burdensome documentation approach by annotating requirements directly to the target application. The stakeholders are allowed to participate in the development process and review the target application even during development.

Vixtory [7] was implemented with Groovy and the Grails framework [29] using Asynchronous Javascript and XML (AJAX) to store requirements in a relational database. Vixtory models requirements in an intuitive way: the requirements are part of the application being developed. There is no need to maintain a separate requirements document. The stakeholders can add a new version of the web application being developed to Vixtory's project database. Each version is identified by an URL address. Stakeholders can freely navigate in the Vixtory web application with a standard web browser.

As can be seen from **Figure 3**, the web page under development is on the right side of the screenshot, and the requirements pane showing a list of the requirements

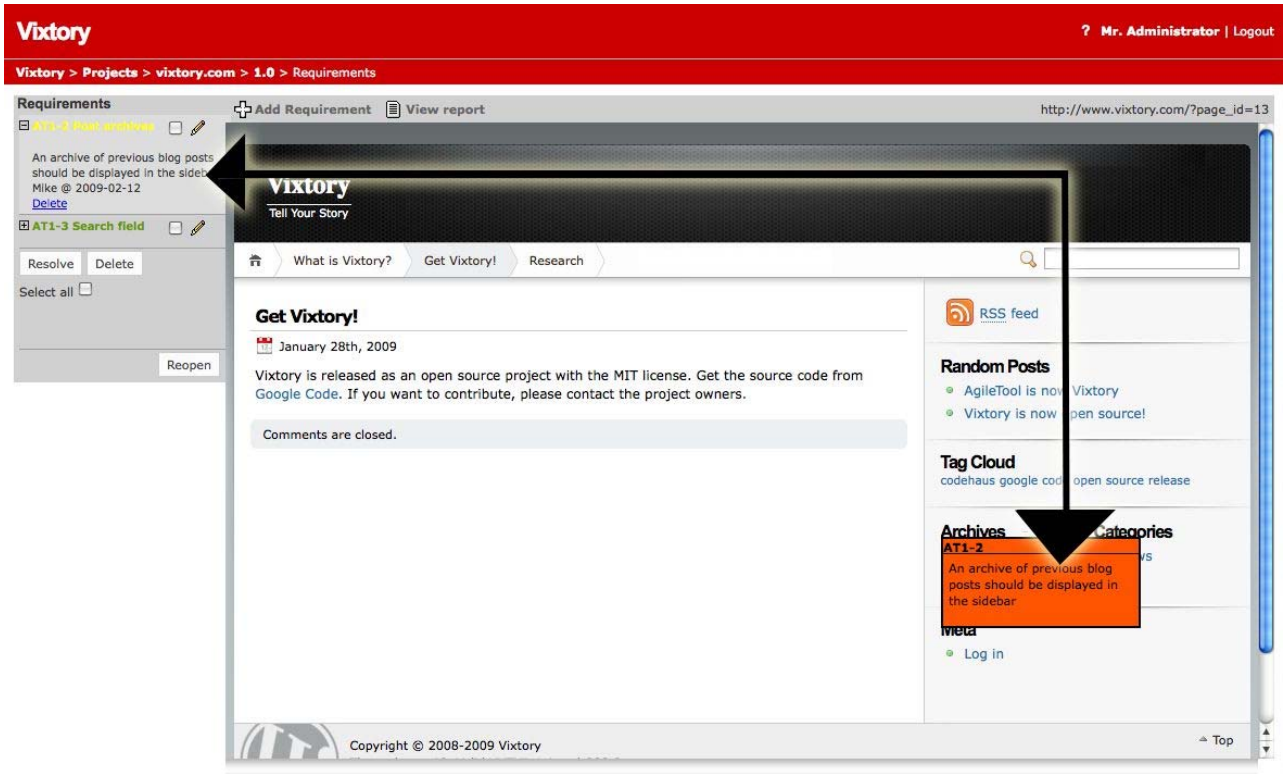

Figure 3. The layout of Vixtory.

is on the left. The stakeholders can browse the web application in question freely page-in-page and identify the requirements for individual views of the web application. The identified requirements are attached with the requirements annotation tool to the corresponding view of the web application. An annotation in Vixtory is a prioritized requirement containing a textual description, listed on the requirements pane.

The annotated requirement is visually linked to an element on the web application. Any elements from links to complicated forms can be annotated with a requirement. The annotations provide a clear traceability link between requirements and the implementation without adding a burden of a specification document, which also facilitates the communication and collaboration between stakeholders. Vixtory provides developers and on-site customers with a straightforward view of the web application being developed, which forms the actual target application context. It also supports and manages change by allowing effortless updating and replacing of requirements.

Vixtory was created with user experience and ease of use as top priorities [30]. Given that Vixtory is in its first commercial release iteration, much work still remains to be done. The requirements specification and representation will further be improved in order to provide end-users with more flexibility in the documentation process. The hypertext links, for instance, between Vixtory and the existing project management tools or collaborative platforms are missing and this is something that will further be considered. How easily Vixtory can be integrated and used with other development platforms and organizational cultures are open questions, worth considering for our ongoing research.

6. Conclusions

We discussed the need for lightweight requirements documentation and presented a reference model for addressing this need. We provided an existing RM facilitated tools taxonomy and drew conclusions on how these tools support requirements analysis and documentation in agile software development. Upon the comparison and contrast of these tools, we identified further needs for requirements documentation that have not been adequately addressed. Therefore, we proposed the adoption of the Vixtory tool and illustrated how it can be used to flexibly document requirements for agile development.

As Vixtory is a prototype tool, we do not yet have enough feedback from the Vixtory tool production use. The feedback upon the initial experimental use of Vixtory has been positive. The project managers, in particular, like the tool. An obvious reason is that the tool makes end user participation easier and it offers less vague and ambiguous requirements due to the actual target system context. In the future, we need to empirically evaluate the acceptability of the tool, asking more stakeholders on their experiences. We currently expect to gain experience from industrial and student software projects. We are particularly interested in the users' communities feedback for improvement.

REFERENCES

[1] K. Beck, "Extreme Programming Explained: Embrace Change," Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[2] A. Cockburn, "Agile Software Development: The Cooperative Game," 2nd edition, Addison-Wesley Professional, 2002.

[3] G. Kotonya and I. Somerville, "Requirements Engineering: Processes and Techniques," John Wiley & Sons, Chichester, 1998.

[4] IEEE Recommended practice for software requirements specification. IEEE Standard 830-1998, 1998.

[5] H. F. Hofmann and F. Lehner, "Requirements Engineering as a Success Factor in Software Projects," *IEEE Software*, Vol.18, No.4, 2001, pp. 58-66.

[6] U. Nikula, J. Sajaniemi and H. Kälviäinen, "A State-of-the-practice Survey on Requirements Engineering in Small- and Medium-Sized Enterprises," Research Report, Telecom Business Research Center, Lappeentanta, 2000.

[7] Vixtory, "Tell your story". http://www.vixtory.com/

[8] K. Pohl. "The Three Dimensions of Requirements Engineering: A Framework and its Applications," *Information Systems*, Vol. 19, No. 3, 1994, pp.243-258.

[9] C. Rolland and N. Prakash, "From Conceptual Modelling to Requirements Engineering," *Annals of Software Engineering*, Vol. 10, No. 1-4, 2000, pp.151-176.

[10] The Standish Group, "Chaos Chronicles Version 3.0.", 2003. http://www.standishgroup.com/chaos/

[11] Agile manifesto, 2001. http://agilemanifesto.org/

[12] E. Berki, "Formal Metamodelling and Agile Method Engineering in MetaCASE and CAME Tool Environments," *The* 1st *South-East European Workshop on Formal Methods*, South-Eastern European Research Center (SEERC): Thessaloniki, 2004, pp. 170-188.

[13] B. Ramesh and M. Jarke, "Toward Reference Models for Requirements Traceability," *IEEE Transactions on Software Engineering*, Vol. 27, No.1, 2001, pp.58-93.

[14] C. Ghezzi, M. Jazayeri and D. Mandrioli, "Foundamentals of Software Engineering," 2nd Edition, Prentice Hall, 2003.

[15] E. Georgiadou, K. Siakas and E. Berki, "Agile Methodologies and Software Process Improvement," *Proceedings of the Virtual Multi Conference on Computer Science and Information Systems*, Virtual Online Conference, 2005,

pp. 412-417.

[16] P. Zave, "Classification of Research Efforts in Requirements Engineering," *ACM Computing Surveys*, Vol. 29, No. 4, 1997, pp. 315-321.

[17] P. Louridas, "Using Wikis in Software Development," *IEEE Software*, Vol. 23, No. 2, 2006, pp. 88-91.

[18] B. Decker, E. Ras, J. Rech, P. Jaubert and M. Rieth, "Wiki-Based Stakeholder Participation in Requirements Engineering," *IEEE Software*, Vol. 24, No. 2, 2007, pp. 28-35.

[19] C. Lee and L. Guadagno, "FLUID: Echo–Agile Requirements Authoring and Traceability," *Proceedings of the 2003 Midwest Software Engineering Conference*, Chicago, June 2003, pp. 50-61.

[20] B. Boehm, "Requirements that Handle IKIWISI, COTS, and Rapid Change," *Computer*, Vol. 33, No. 7, 2000, pp. 99-102.

[21] Z. Zhang and J. Kaipala, "A Conceptual Framework for Component Context Specification and Representation in a MetaCASE Environment," *Software Quality Journal*, Vol. 17, No. 2, 2009, pp.151-175.

[22] O. Gotel and A. Finkelstein, "An Analysis of the Requirements Traceability Problem," *Proceedings of the* 1*st International Conference on Requirements Engineering* (*ICRE '94*), Colorado, 18-22 April 1994, pp. 94-101.

[23] INCOSE requirements Management Tool Survey. http://www.paper-review.com/tools/rms/read.php

[24] A. Forward and T. C. Lethbridge, "The Relevance of Software Documentation, Tools and Technologies: A Survey," *Proceedings of the* 2002 *ACM symposium on Document engineering*, McLean, Virginia, USA, ACM Press, pp. 26-33.

[25] A. Persson and J. Stirna, "Advanced Information Systems Engineering," 16*th International Conference*, *CAiSE*, Riga, Latvia, June 7-11, 2004, Proceedings.

[26] A. Manninen and E. Berki, "An Evaluation Framework for the Utilisation of Requirements Management Tools-Maximising the Quality of Organisational Communication and Collaboration," *Proceedings of BCS Software Quality Management* 2004 *Conference*, British Computer Society: Swindon, 2004, pp. 139-160.

[27] C. Heitmeyer, J. Kirby and B. Labaw, "Tools for Formal Specification, Verification, and Validation of Requirements," *Proceedings* of *the* 12th *Annual Conference* (*COMPASS*'97).

[28] B. Kernighan, "Sometimes the Old Ways are Best," *IEEE Software*, Vol. 25, No. 6, 2008, pp. 18-19.

[29] G2One Inc., Grails Web Application Framework, http://grails.org/

[30] M. Arvela, M. Muhonen, M. Piipari, T. Poranen and Z. Zhang, "Agile Tool-Managing Requirements in Agile WWW Projects," *Proceedings of BIR* 2008, Gdansk, 2008, pp. 211-215.

