Story-Wall: Supporting Agile Software Development in a Distributed Context

Lorena Delgadillo[1], Olly Gotel[1] and David Leip[2]
[1] Department of Computer Science, Pace University, New York City, New York, USA
[2] ibm.com Chief Innovation Dude and Agile Methods Advocate, IBM Hawthorne, New York, USA

Abstract

The majority of commercial requirements management tools tend to be costly, document-driven and used by large organizations undertaking traditional forms of software development. While they are not immediately in the spirit of the agile philosophy, which advocates live dialogue over documentation and encourages small teams of developers to do the simplest thing possible to satisfy a requirement, there are some fundamental practices supported by these tools that play a role in more agile forms of software development. This paper examines the core requirements management needs that are common to software development of all flavors, at a high level, and describes a tool concept designed to bring lightweight requirements management to the agile (predominantly XP) context. This work is based on the experiences of a globally distributed team that uses agile development practices to develop and maintain ibm.com, and on their transition from manually handling paper-based story cards to the use of first generation story management tools.

1. Introduction

Requirements engineering is that part of systems engineering that is concerned with exploring and agreeing stakeholder needs [4]. Whether a software system is being built according to a traditional software development approach (e.g. waterfall-based and plan-driven or the Rational Unified Process) or following an agile software development approach (e.g. Crystal or eXtreme Programming), we claim that the fundamental requirements engineering activities are basically the same. Stakeholders need to be identified, candidate requirements need to be determined, analysis, discussion and negotiation has to take place to check, agree and prioritize the requirements, and some mechanism needs to be established to monitor progress, check that requirements have been satisfied and to deal with the inevitable requirements changes. The latter activities are the remit of requirements management. The major difference between development paradigms is the explicit emphasis and support given to each activity in requirements engineering and the importance that is placed on the documentation of such. Requirements management, however, is one term that does not appear so frequently in the agile software development literature.

This paper describes a research project undertaken to understand those features of requirements management that are inherent in most forms of software development process. It examines the support provided within current requirements management tools and prototypes a tool concept for lightweight requirements management, specifically targeted to agile (XP) teams. The initial concept is based on the experiences and needs of one team in particular within IBM, a team working as a globally distributed agile team, so is concerned with the distributed management of requirements. The concept has yet to be developed fully or validated more widely. The paper therefore reports on work in progress. The tool concept focuses on having a low start-up cost in terms of process and technology prerequisites, on preserving the benefits of manual and more social ways of working in an XP team, and on replicating important visual metaphors and physical activities that are prevalent in a co-located context. Feedback on the initial tool concept from practitioners and future work is discussed. This involves the obvious need to differentiate the concept from the many tools that are now emerging.

2. Research Method

A combination of approaches aided in the development of this work and in the prototyping of a concept for a lightweight requirements management tool to support distributed agile software development.

2.1 Literature

Literature on the topic of requirements engineering and requirements management, software development and agile methodologies, was studied and reviewed to gather knowledge about the different topics and issues addressed in this paper. This helped in understanding the essential requirements for requirements management across development paradigms.

2.2 Questionnaires

A one-page questionnaire was created in order to gather information from practitioners to corroborate the above findings and to gain more detail. The questionnaire was intended to investigate practitioner use of agile development practices within their work setting, as well as to gather details about any requirements-related tools used, to get an understanding of the state of the practice.

2.3 Observation and Participation

Observation of teams and participation in agile professional presentations aided in developing knowledge of agile methodologies and their most fundamental (although not explicitly articulated) requirements management needs. This involved discussions with practitioners within the New York Agile Project Leadership Network.

2.4 Tool Critiques and Use

Critiques were gathered from Pace University graduate students regarding different tools that they had been exposed to in their Systems Requirements Engineering class. These, plus personal exposure to over twenty various commercial and open source tools via demo versions, helped in comparing the features of current requirements management tools at a preliminary level (often referred to as story management tools in the agile community, where tentative requirements are labeled as stories). Other resources were also consulted (e.g. [5]).

2.5 Practitioner Feedback

Feedback on the early tool concept and prototype of such was obtained from practitioners using agile to support global working within *ibm.com*. The ideas were also discussed with other agile leaders at a meeting held in April 2007, including XP practitioners from within IBM and Google.

3. Requirements

Requirements engineering is the process through which requirements, both technical and non-technical, are developed and maintained by all of the stakeholders throughout the project life cycle. Requirements management provides the ability to see which requirements have been implemented and where, and to assess the impact of potential changes, as well as to help see those changes through. Many practitioners and experts have their own understanding of the various project and software development tasks that requirements management should help simplify and support [3]. We describe the primary roles and requirements to support them below.

3.1 Stakeholder Roles

Developers, requirement engineers, quality assurance (QA) testers and the business clients/customers of the project concerned are all stakeholders in the requirements management process. Each role has goals and tasks that need support.

Business Clients/Customers. The goal of the customers is to have a quality product that meets their requirements, without exceeding allocated time and cost. Customers need to be able to propose high-level needs and examine other suggested requirements, be able to discuss, prioritize and approve requirements, as well as make changes to requirements as they learn more about the problems and opportunities at hand. A requirements management tool must provide customers with the ability to capture their initial ideas about requirements, and organize and prioritize requirements to be developed according to their needs and deadlines. They should be able to check progress, make clarifications and request changes.

Developers. The role of the developers is to transform the requirements into a design or a prototype. Their main goal is then to develop a successful product that meets requirements and is fit for purpose, with a minimal acceptable amount of errors. By having requirements traced to design and code, developers can determine which requirements have been completed and which are yet to be developed. They need to know what requirements need to be worked on next, have details available to assist this work and the ability to track progress. Tracing to test cases for acceptance testing is also important for reaching sign-off with customers. Requirements management tools should provide a framework through which the developers can gain access to the information they need to plan and conduct their work.

Requirements Engineers. A requirements engineer focuses on writing representative requirements that can be checked and agreed by the technical team and the business customers, then developed. A requirements management tool should facilitate the way in which requirement engineers investigate and develop requirements, and handle changes. Support for writing better requirements is desirable. The role of a requirements engineer is rarely mentioned in the agile literature, these activities being the responsibility of the customer/developer team. It is possible that successful teams develop informal requirements processes that they implement implicitly through social protocols.

QA Testers. QA testers need to validate that processes and standards are being followed in the development of software and conduct proper testing of the requirements for a project. A requirements management tool should help them verify that requirements were implemented as specified and satisfy test cases. QA is a core concept in software development and one that also receives little explicit mention in the agile literature, again being the wider and implicit remit of all team members.

3.2 Essential Requirements

We suggest that a requirements management tool must have, at a minimum, the following features in order to assist with the primary stakeholder goals: Requirements Storage; Requirements Prioritization; Change Control; Requirements Progress; Requirements Traceability; and Better Requirements Assistance. These requirements for requirements management do not have to be implemented through a software product; they can be implemented through some form of manual documentation and human activity process, and they never have to be signaled out as requirements management activities as such. This is what appears to be the case on most XP projects we observed. However, with issues of project longevity, size and team distribution, as was found in the use of agile software development processes within *ibm.com*, some form of lightweight support is viewed as desirable.

Requirements Storage. A requirements management tool should be able to store and help organize requirements. Requirements can be stored in the simplest possible manner using a form, spreadsheet or, more usually, as a database schema. Requirements need to be stored in order for them to be rendered visible, shared, managed and tracked for any changes. The requirements management tool should be able to store requirements with as much or as little detail about them as necessary (i.e. metadata). Obviously, there are attributes about the requirements that can help with activities like change or scheduling (such as rationale and cost factors), but the collection of such information should not be enforced or given unconsidered default values that are meaningless. Progressive customization should be possible and never intrusive; a requirements management tool should be able to allow the user to change or add attributes as the project needs demand.

Requirements Prioritization. Not all requirements are considered equal. Trade-offs will always need to be negotiated as to what can and what should be done on a project. A requirements management tool must have support for requirements prioritization. Customers must be able to view and organize requirements according to their business value and needed completion dates. Developers need to be able to select requirements to work on so as to meet the customer demands in a timely fashion.

Change Control. Customers and developers do not always agree on requirements and, with many changes in the requirements, up to date requirements are not always what are stored and managed by these tools. At the end of the day, the potential of having a certain portion of the project not being traced back to accurate requirements poses a potential risk to both the cost and quality of the product, since it may mean that the wrong requirements are satisfied, a situation that can bring about project delays in rectifying. Change control is thus an essential part of requirements management. Every requirement can change and it can change more than once. Changes in requirements should be tracked in a requirements management tool in order to know the cause for the change and the impact on quality, cost, schedule, etc. The more detail maintained about requirements and the changes, the better it may be to understand impact and to roll-back any problematic changes. However, the level at which changes are tracked needs to be totally customizable, since atomic granularity may be over-excessive for some project needs. Again, enforcing too much detail can prevent any relevant information from being maintained at all. There should also be a way for teams to decide whether they want consequential changes to requirements to be authorized and agreed, and by whom, rather than enforced policies.

Requirements Progress. Requirements are subject to changes and their differing priorities mean they are subject to different development schedules. Keeping track of the status of a requirement is vital to the requirements management process. Requirements can be preliminary, to be negotiated and discussed, or can be in a development stage, tested, implemented, approved, etc. Awareness of where a requirement is in relation to the project as a whole and with respect to other requirements enables better lifecycle monitoring.

Requirements Traceability. A requirements management tool must be able to link requirements to design and code and back to requirements. This is referred as forward-backward post-requirements traceability [3]. Further, vertical traceability is the linkage between a requirement, its design and its code, so shows its development path. Horizontal traceability is the linkage between versions of the same artifact, so shows the progression of ideas about a requirement. Traceability between requirements and the source of requirements is known as pre-requirements traceability. Requirements traceability is an enabler for managing change and monitoring progress.

Better Requirements Assistance. How does the user know whether they are collecting and managing 'good' requirements, or whether they are managing nonsense? Most requirements management tools do not help the user to formulate requirements. Rather, they leave a free form text area where the user can input whatever information they please. Overly bureaucratic requirements management processes and tools can actually sometimes end up managing out of date garbage as a consequence! Tools always rely on the people to do a good job, and some tools can make this easier to encourage and realize than others. A requirements management tool should help in this area.

3.3 Good Enough Requirements Management

A good enough requirements management tool will have all the essential requirements. It should be able to store requirements in a simple form of the team's choosing, prioritize requirements to business value and development tasks, keep track of any changes at whatever level of granularity is considered desirable, provide awareness of the progress of requirements, and trace links between and amongst requirements and other artifacts to enable some of the above tasks. Traditional requirements management tools strongly support these features, but generally require that rigorous processes are followed. This makes them somewhat unappealing for lighter and more agile development processes. Rarely does any tool focus on helping users to write better requirements though.

4. Agile Requirements Management

In contrast to traditional requirements management approaches, agile software development practices do not focus on detailed requirements documentation. In fact, XP [1] uses physical paper index cards, known as 'story cards', to record requirements. Story cards are used as a way to prompt discussion about requirements between the developers and the customers, so can be viewed as very loose requirements. Communication between the customer and development team is one of the core tenets of agile and most approaches do not explicitly cite the need for a requirements engineering or QA role on the team, their tasks being assumed by the others. Stories are usually placed on a physical whiteboard or pinned to a wall in the room of the developers. Such a wall often demarcates different stages in which a story can be placed, such as 'to do', 'in progress' or 'completed', as well as others, the exact terms being defined by the team.

Throughout the software development life cycle, stories are created, prioritized (for business value), sized (for estimating time to build) and placed in iterations for development. Iterations are short cycles when features of the software are built incrementally, tested and released. Customers choose stories they want developed during an iteration, based on priority and sizing, as well as decide upon those stories that either require changes or are no longer needed.

During the creation of stories, the developer and the customer undergo various forms of communication in order to better understand the customer need, which more than likely is not recorded or stored. This poses a potential problem when developers try to remember what the story was about and when the person they communicated with is not immediately available or no longer present. Agile methodologies don't hold requirements traceability as essential to the development and change management process as do traditional methodologies. This is somewhat due to the perception that traceability is costly, manually intensive and heavyweight. It is also possibly because some agile practices may be considered requirements management 'in disguise' (e.g. a shared codebase with ticket tracking and the emphasis on testing procedures and tools). However, agile projects can leverage some of the simpler practices, especially since traceability is an enabler of the other activities, and some form of pre-requirements traceability may be essential since the understanding of stories is all wrapped up in the discussion. The issue of writing better requirements becomes somewhat mute in this context.

The majority of XP teams appear not to use a tool to manage their story cards. This poses a cost and flexibility advantage for small co-located teams working on short projects, but also a potential problem since story cards can be lost or misplaced, which can affect the project at hand. Managing stories becomes more problematic when dealing with a globally distributed project, where the different team members are placed in various locations around the globe (as per the *ibm.com* team), because development teams will not be able to view the physical wall of stories. Ensuring teams are working to the same story becomes more difficult with project longevity because each team might have different versions of the physical story card. Unobtrusive shared awareness of the project status, or the 'big picture', is also visually lost.

5. Requirements Management Tools

There is a diversity of requirements management tools currently in use by many companies. A high-level study was conducted on a sample of approximately twenty requirements management tools on the market, including those used for traditional and agile approaches, the latter often being referred to as story management tools. These included Analyst Pro, Cradle, DOORS, ExtremePlanner, In-Step, Leap SE, OptimalTrace, Project Cards, Rally, RequisitePro, XPlanner, Project Cards, Rally, RaQuest, RMTrak, Target Process, TopTeam Analyst, Trac and TRUEreq, along with a few open source Eclipse plug-ins. Each tool was compared to the essential requirements defined in Section 3.2. Since this was a cursory examination to gain an initial awareness of the state-of-the-art, highlights are summarized below.

In general terms, requirement management tools can be classified as either lightweight or heavyweight in their support. Heavyweight tools tend to place demands on the procedural configuration of the system, such as the enforcement of prescribed processes, the necessary sequencing of activities or the required data to be collected for traceability to work. They tend to be centered on document production activities and multiple components often need to be installed in order for the requirements management tool to function. Together, this can present a high start-up cost and barrier to use for many agile teams. Some requirements management tools are dedicated only to requirements management activities, while others can be used to manage requirements throughout the project life cycle by integrating with other development tools, again often necessitating more complexity in the technical configuration, which is not always suitable for non-technical customers. There are some emerging lightweight tools that are generally less expensive or open source, and easier to install and use from uptake. The intention of this work was to focus on the start up cost, the ability to ramp up process support as and when needed, and on sharing the stories across locations in a visual and usable way, affording usability benefits over existing tools.

Each tool surveyed had a means of storing requirements. Some tools use servers in order to store the requirements and other data. Some tools are standalone and can be installed on a shared drive for multiple users to use. The interesting point about many of the agile tools is that the notion of the story card for writing requirements, while often replicated, is frequently allowed to grow to exceed the amount of text that can be written on a paper-based index card, which causes problems for any physical display.

Not all the requirements management tools allow for the ready prioritization of requirements, but there are other tools that help users prioritize requirements by assigning levels and numbers. However, none of the tools appeared to support the physical manner by which customers have been seen to order stories in practice or to support distributed prioritization.

Most tools on the market have some form of change control. This can be as simple as generating a change report that details the changes at a summary level or as complex as having notifications emailed to managers or ensuring change requests are prior approved by managers. Not all tools give the team the ability to customize the software development process they can use to undertake their project though and so the steps they need to undertake to handle change do not always fit lightweight distributed working practices well.

Most of the tools surveyed showed the progress of requirements. Some showed which phase the requirement was in, such as analysis or development; others show if a requirement is overdue or completed, proposed or approved, etc. Shared visual awareness of this status across locations was not a focus though.

Since not all tools support the full development life cycle, most only provide traceability between requirements and not onwards to design and code. Tools supporting traditional development, such as OptimalTrace by Compuware, Doors by Telelogic and RequisitePro by IBM, can develop requirements through the development life cycle. Tools purporting to support agile do not address the traceability issue per se, so do not have a trace matrix or report.

Current tools can improve requirements management practice only if they can first help users create better requirements and support the creative and exploratory interchange process that surrounds their development. This interactive dialogue is the cornerstone of many agile approaches. Unfortunately, not much of the requirements creation and development problem has been addressed by many of the tools. In the study, only one tool appeared to help users to write 'good' requirements. Although the tool concerned helps the user to compose requirements according to templates, it does not have any other features or distributed authoring capability, so is limited in scope once a requirement is first crafted.

While vendors have created various tools as a solution to the requirements management problems sometimes experienced in agile software development projects, the study found that those tools, in contrast to agile practices, tended to be either complex to set up, procedurally restrictive, adorned with unnecessary functionality or had poor customer usability. These factors somewhat defeat the idea of going agile and of involving customers (who may be at a remote location and less technically savvy than the developers).

Many of the current agile 'requirements management' tools overload the tool with features that are not needed very often and, in turn, create a more complex tool than may be necessary to support what is meant to be an agile and lightweight process. This presents a barrier to use and enforces the perception of requirements management as being heavyweight. Hence a need for an open source and lightweight requirements management tool for agile software development. Extending this support to a distributed context is essential for global teams like *ibm.com*.

The *ibm.com* global XP team uses a tool called Extreme Planner. Although a lightweight tool, it does not have any requirements traceability or overall story visibility, presenting shared awareness problems for the globally distributed team. An XP development group at Google uses a recent and custom made tool to suit their needs for agile story management, but claim they are yet to find the ideal tool (personal communication with the first author).

6. Story-Wall Concept

In order to address the need for lightweight requirements management for agile software development, we have been exploring a concept for a new tool. This tool concept addresses the essential requirements mentioned in Section 3.2, seeks to have a low start-up cost, focuses on accessibility to the customer as well as to the technical parties, attempts to maintain important physical and visual metaphors from co-located practice, and promotes a solution to managing requirements in globally distributed projects.

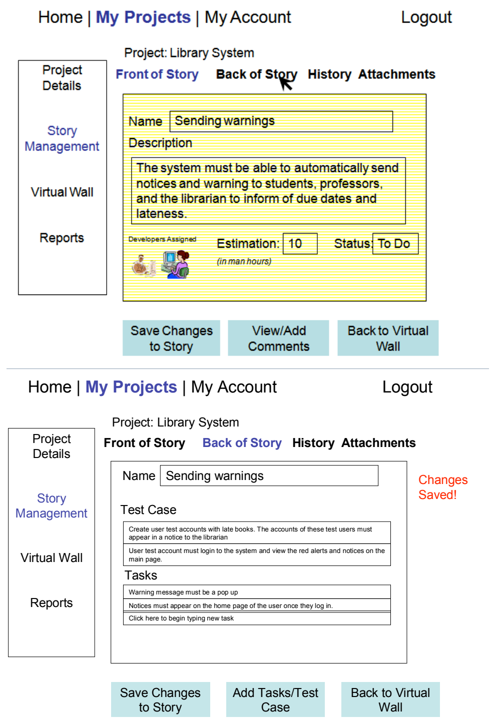
6.1 Users

Story-Wall provides the ability for developers and customers to all contribute to and view the stories for a project without having to install dedicated tools; it is wiki-based. There is support for a member of the agile team to set timelines and time boxes for the project, and to allocate developers to the project and stories. According to roles, stakeholders can create, elaborate, size, prioritize and allocate stories to iterations. Story-Wall is about raising awareness of the stories in a project and their developmental progress. Story-Wall does not promote the role of a requirements engineer or QA tester per se, though supports the essential activities such roles would cover (which are generally undertaken by the customer and developer combination, and often quite tacitly, in an XP project).

6.2 Features

Story-Wall focuses on providing a high level view of the project's stories and on managing the development and implementation details about them. Its main features naturally tie back to the essential requirements identified earlier and include: Story Card Simulation; Prioritization and Sizing; Iteration Selection; Virtual Wall; History of Discussion and Changes; and Lightweight Requirements Traceability.

Story Card Simulation. In physical reality, story cards are limited in size according to the dimensional constraints of the index cards used. In addition, there is both a front and a back side to the card that tends to be used in different ways by different agile teams. Within the IBM team targeted by this work, the details of the story are written on the front of the card, as per Figure 1. The back of the story card is often used to show the different tasks the story involves and/or the test cases for the story. This same concept was adapted for Story-Wall. When a user selects a story card, they can view the front and back of the story card, and contribute to either as needed. Information about the story is maintained with the story in this way.



Project: Library System

Name: Sending warnings

Description: The system must be able to automatically send notices and warning to students, professors, and the librarian to inform of due dates and lateness.

Estimation: 10 (in man hours)

Status: To Do

Project: Library System

Name: Sending warnings

Test Case

Create user test accounts with late books. The accounts of these test users must appear in a notice to the librarian

User test account must login to the system and view the red alerts and notices on the main page.

Tasks

Warning message must be a pop up

Notices must appear on the home page of the user once they log in.

Click here to begin typing new task

Figure 1: Story card simulation (front and back).

We also retained the concept of having a fixed physical size for the stories. Many agile tools, while preserving the concept of an index card, actually allow the text within it to expand via a scrollable text field. We suggest that this loses some of the power of the fixed size paper card metaphor. A small writable area forces the story to remain tentative and to act as a prompt or reminder for discussion about requirements as opposed to conveying the impression of an exhaustive, finalized and stand-alone requirement.

We purposely kept the meta-data associated with a story to a minimum. In Figure 1 it includes the most critical data for the agile team of the study – priority, sizing, allocation and status, but this can be left blank or customized. The cards can be used as teams require.

Prioritizing and Sizing Stories. A direct manipulation drag and drop interface was created to enable customers to sort their story cards into priority lists according to perceived business value or importance. The idea of adding numbers to cards to depict priority, or levels such as 'high', 'medium' or 'low', did not always reflect what appears to be done with physical cards in practice. We decided to provide a means for customers to physically order their cards by direct manipulation into piles or lists. To maintain this priority information gained through physical ordering, the cards are shaded, the darker the yellow (the default card color), the higher the business value. Visual cues on relative priority can help requirements not look like a monolithic un-prioritized list. To promote distributed prioritization we have been investigating how to do this via Delphi techniques.

Sizing is the anticipated development effort to build a story (also referred to as the estimate). Writing the sizing information on an index card was something we noted development teams to do, but sometimes it was remarked as easier to estimate tasks relative to others (i.e. it is easier to say that one story will take approximately twice as long as another). We therefore explored approaches to carry out sizing by directly stretching the story card so that the area the card is sized to represent the numerical estimate, as an alternative to writing a number directly. The team can physically manipulate cards until a consensus is reached and a visual indication of effort is immediately available to assist this distributed consensus building.

This priority and sizing information is visually represented in the virtual story wall that manages the story cards within Story-Wall. This displays the big picture context for the agile project.

Iteration Selection. Typically, the customer uses the priority and sizing information to select those stories to be developed per iteration. The constraint is the velocity or total amount of effort that the developers have available to spend developing stories during the time-boxed iteration. Stories are selected for the iteration in a further direct manipulation drag and drop manner, to fill a container proportionally sized to reflect the available velocity. The idea being that it is easier to visually see whether the iteration is filled with small effort tasks implementing features of low value to the business, or high value stories requiring large amounts of effort. This is intended to assist the customer in undertaking this process and the development teams in organizing their subsequent work, perhaps focusing on those stories per iteration with most business value and possible ease.

Virtual Wall. Both the customers and developers of XP projects rely on story cards being placed on a wall or a whiteboard to allow them to see what needs to be or what has been done on a project, and to easily move a story between stages of completion. It provides for on-going situational awareness of a project. Obviously, cards can fall off walls, get mislaid, and the ability to share and collaboratively manipulate cards on a physical wall does not translate across distances. Further, while it is ideal to have a co-located customer participating in the project, experience within this one team at IBM illustrates that often some form of tele-presence is the reality. A story management system is currently being used to write and allocate stories to developers and iterations by this team, but an issue was found that this visual story-wall metaphor was lost. We therefore explored the concept of a virtual story-wall as the cornerstone of distributed agile support. This concept was adapted in the Story-Wall prototype and different ways to replicate this wall explored. An early and somewhat crude illustration is given in Figure 2.

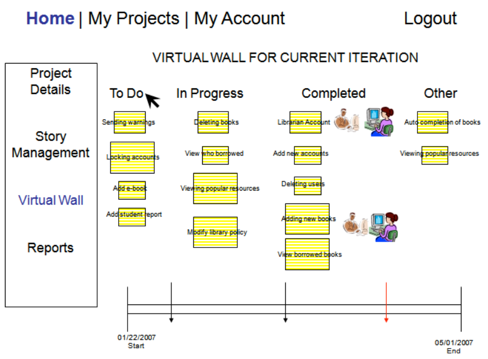


Figure 2: A virtual story-wall.

Upon opening a project, the current status of the virtual wall is displayed, the iteration being centralized in the context of the end-to-end project timeline. Users are able to scroll back to different iterations and view the virtual wall at past moments in time. Users can view the details of a story card by just selecting it. By hovering over the story, the developers assigned to it can be viewed. Using drag and drop functionality, users can drag story cards from stage to stage until completion and customer acceptance. The names for stages are customizable. The shared virtual story-wall allows for users of the tool to share up to date story card information and see on-going progress.

History of Discussion and Changes. Yet to be completed stories are always changing and these changes are sometimes never tracked. The Story-Wall tool concept keeps track of the changes made to a story by adding a simple history to the story. A user can choose to view the history of the story and see who created the story through to the latest change contributor. Annotations and comment facilities are provided. Font types differentiate contributors, as per a physically annotated card, which can be a quick visual way to see who has worked on what.

Lightweight Requirements Traceability. Requirements traceability, as mentioned previously, is often side-lined or even ignored in an agile context (within explanations of agile approaches more so than in practical reality), as perceived as burdensome and even redundant. However, an increasing number of development situations and contexts are actually seeing the need for some form of explicitly planned traceability to support project longevity, scale and distribution. Lightweight requirements traceability can be achieved via the story -wall concept as a by-product of everyday use. Additionally, story details can be traced back to the discussions between clients and the development team by facilities integrated to the wiki implementation to record a meeting or phone conversation (exploiting a number of new web 2.0 technologies), or by saving a chat record and uploading it to the tool. The uploaded file is then accessible for users to either listen or read once they view the story details. Tracking multimedia requirements information of this nature to support requirements understanding is a current research topic within many organizations [2].

7. Ongoing and Future Work

Currently, the tool concept is in early prototype phase. The platform on which this tool is being developed is Wiki-based, like many of the emerging agile tools, since they allow for an easy and intuitive way for members in teams to communicate and share up to date information. All the individual ideas have been prototyped with the target technologies and are in the process of being developed and refined more fully. The tool concept has been validated in principle with a small number of targeted practitioners, but this requires more extensive validation with additional practitioners to help refine the concept further. An important element of this evaluation will be determining how well such a tool can actually fit in with existing team practices and commonly used tools when undertaking distributed agile development, especially without disrupting implicit but mature social protocols.

The traceability features are presently very minimal and support for writing better stories is not yet provided. Ongoing work is examining how to maintain traceability between connected stories, in addition to the story derivation and discussion process, and integration with subsequent development artifacts.

8. Conclusions

This paper reports on work in progress on the development of a lightweight requirements management tool for distributed agile software development. A concept is being developed and prototyped to explore how to bring some of the fundamental benefits of traditional requirements management and traceability, painlessly, to the needs of an agile team. The concept is influenced by the needs of one agile team working within IBM using XP.

Requirements management tools should be able to help handle changing requirements and facilitate their eventual implementation and approval. In order for requirements management tools to work efficiently they must be able to store requirements, prioritize requirements, track changes to requirements, track the progress of requirements, provide requirements traceability and offer some support for ensuring that the requirements they manage are of quality.

Current requirements management tools predominantly support traditional development processes and practices and are known to be heavyweight; they are hence not the obvious first choice for the growing community of agile software developers. They tend to force processes and procedures that can be viewed as overly burdensome and contrary to the agile philosophy of 'doing the simplest thing possible' at all stages. Current agile story/requirements management offerings do not have all of the most fundamental of requirements for a requirements management tool, and many organizations are now looking to fill this gap, particularly those than operate with distributed teams around the globe. This work is a first step to exploring some potential concepts that may be applicable to such future lightweight requirements management tools.

9. References

[1] Beck, K. *Extreme Programming Explained, Embrace Change.* Addison-Wesley. 2001.

[2] Gall, M., Bruegge, B. and Berenbach, B. Towards a Framework for Real Time Requirements Elicitation, *Proc. 1st Intl. Workshop on Multimedia Requirements Engineering*, Minneapolis, MN, September 2006.

[3] Gotel, O.C.Z. and Finkelstein, A.C.W. An Analysis of the Requirements Traceability Problem. *Proc. 1st IEEE Intl. Conf. on Requirements Engineering*, IEEE Computer Society Press, Colorado Springs, CO (April 1994), 94-101.

[4] Hull, M. E. C. Jackson K., and Dick J.J., *Requirement Engineering*. Springer-Verlag, London, UK, 2002.

[5] Ludwig Consulting Services, LLC, *http://www.jiludwig.com. July 2006.*