A Feasible User Story Tool for Agile Software Development?

Michael J Rees
School of Information Technology, Bond University,
Qld 4229, Australia
mrees@bond.edu.au

Abstract

User stories form the heart of the Extreme Programming methodology planning game. In its turn, Extreme Programming is one of the supporting pillars of the wider Agile Software Development process. The user stories form a set of central work products that determine the software development processes. In the spirit of Extreme Programming the production of user stories is kept as simple as possible. Traditionally, user stories are hand written on index cards as they are easy to store, display, rearrange and distribute to the co-located development team.

However, virtually all other work products of an application development team are in electronic format, and Agile Software development is increasingly being adopted by teams working from remote locations. In these circumstances distributed development teams look to software solutions for creating and using user stories. This paper surveys some of the tools that are being used and examines their suitability for the task using a set of requirements.

As a result of this investigation the author has produced a prototype user story software tool for Agile Software Development called DotStories. The paper concludes by discussing the features of DotStories and its how closely it can approach the ideal user story software tool. Indeed, the proposition that software tools can ever improve on index cards is considered in the light of experience to date.

Keywords: User stories, software development tool, Agile Software Development, Extreme Programming

1. Introduction

Agile methodologies for software development [1] take a novel, lightweight approach to most aspects of designing and producing applications. This paper focuses on the requirements phase of software development which probably ranks as the crucial first step. Poor decisions in this early phase of activity are known to be exceedingly costly in the later stages of software production. It is not surprising that all tools for aiding the software production process place major emphasis on constructing detailed requirements as accurately as possible.

Knowing that success in constructing requirements has been difficult to achieve with conventional software construction methodologies, Agile development, and Extreme Programming [2] in particular, adopt a radically different approach. From the outset, it is assumed that full, detailed requirements for a software package can not be developed as these requirements will inevitably change over time for several reasons. These include changes to business requirements, technologies, user populations and the evolution of other software with which the new software must interact. Such changes appear to be fundamental to all significant software development.

Thus Agile software development takes the view that production teams should start with simple, knowable approximations to the final requirements, and then continue to increment the detail of these requirements throughout the life of the development. This incremental requirements refinement is thus intertwined with design, coding and testing at virtually all stages of production activity. In this way, the requirements work product is as accurate and useful as the final software itself. Indeed, the requirements work product, being oriented to tasks and features, can itself aid the production of user documentation, another essential work product.

This significantly different incremental, intertwined approach to requirements development is one of the flagship components of Extreme Programming. In this context the requirements are recorded as a set of *user stories* developed jointly by customer representatives and the development team. In keeping with the lightweight approach to software development, teams adopting Extreme Programming are advised to use traditional index cards to record these user stories. This has led to the synonymous term *story cards* also being used.

In this paper, the benefits and disadvantages of story cards are discussed in the context of Extreme Programming. Other electronic tools that are also used are examined. Out of this is developed a set of requirements for specific user story tools. The author has developed a web-based user story tool called DotStories based on these requirements. The development of this tool and its outcomes are presented, and compared with other software tool efforts. DotStories outputs user stories in a proposed standard format called User Story Markup Language, and the merits of this format are put forward. Finally, the question 'Can software tools improve on story

Proceedings of the Ninth Asia-Pacific Software Engineering Conference (APSEC'02)
0-7695-1850-8/02 $17.00 © 2002 IEEE

cards?' is discussed, particularly for remote teams, and some conclusions presented.

2. Extreme Programming

The Extreme Programming methodology is one of the main pillars of Agile Software Development. This philosophy and methodology was developed by the Agile Alliance, a group of like-minded individuals who favor lightweight development processes. They met in February 2001 to agree on the Agile Manifesto whose main tenets are:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan
-
- It is the last tenet that forms the focus of this paper.

In brief, Extreme Programming is designed for development teams of between 3 and 10 engineers. The team is enhanced with one of more customer representatives, and managers to provide architectural input and to act as coding mentors or *coaches*.

The team is co-located in one room or adjacent rooms. Programming is done to a coding standard, test-first in pairs aiming at development iterations of 2 to 3 weeks. After each iteration a working version of the software under development is available for user testing.

The programming process is directed by a series of user stories developed at a planning meeting held at the beginning of each iteration. This process is part of the *planning game* with a set of rules, players and moves. The meeting considers the pool of user stories and decides which are to be included in the next iteration. Daily stand-up meetings lasting of the order of 15 minutes review progress and change development priorities where necessary.

It is important to realize that it is the customers that create the user stories. Each user story can be thought of as a small piece of functionality of the final software perceived from the end-user point of view. Once created by the customer, it is the developers that refine the user story by estimating the amount of development time needed to implement the functionality *in isolation* of all other stories. Ideally a user story should be capable of implementation in two to three days. If the estimate is longer, then the user story should be split into a series of smaller ones. The developers decide if splitting is necessary.

Once the collection of user stories is agreed, it is the customers again who decide on the priority sequence that dictates implementation order. As a guide, the customers decide on the business value of the stories and which are most valuable for the next release. After picking a subset of the stories, the implementation estimates are summed to indicate when that release can be expected. Thus the period for the next iteration is decided. Although the customer appears to be in the dominant decision-making role, recall that the all-important estimates are provided by the developers. The needs of both the business and the development team are taken into account.

3. User Stories

User stories are an excellent example of the lightweight nature of Extreme Programming. In his Extreme Programming bible [2] Beck suggests the ultimate low-tech tool, the index card, with one user story per card. The card contains the following recommended information:

- User story number or id
- User story title
- Person responsible for the story, typically a customer
- Date
- Estimate of implementation time, typically in days
- Risk level of the technology used
- User story description
- Other optional information includes:
  - Unit test description
  - User interface needs
- Other related story numbers

In practice, it appears that each development team evolves their own set of information fields that tend to be subsets of the ones listed above. Even in Beck [2] where an example of an actual story card is presented, the card format contains extra, more detailed, tabular information that is conspicuously blank. The lesson here is that extra complexity leads to non-use—the 'keep-it-simple' rule always comes into play.

This uncertainty surrounding exactly what information should appear on a story card and in what detail has often been mentioned by development teams hoping to start using story cards. Fortunately useful examples are now available from many sources, [3] [4] [5]

Story cards can be very effective when containing a very small amount of information. In Figure 1 is presented an example of a story card from a project with which the author is associated [6] . (Note that this is a difficult to read digital photo of the actual index card, see later.) There are about four developers involved, with pair programming recently introduced. Not shown is the reverse of the story card which is divided into Unit Test and User Interface sections, each consisting of a paragraph or two of text. In this particular case a suggested user interface layout of the user story functionality output is shown in tabular form on the front of the card.

Again from Figure 1 it will be noted that story card information fields like Person and Risk are not used. The user story numbering system employed a triple number representing 'month-day-sequence number'. Estimates were written initially as date targets. This was later changed to the more normal days-to-implement number.

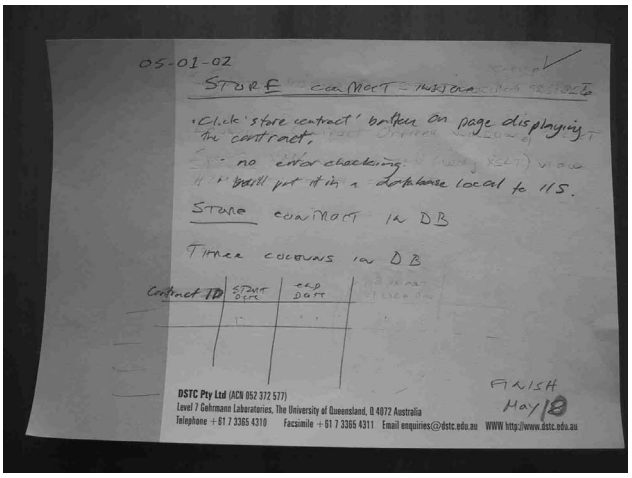

Figure 1. A Sample User Story Card.

4. User Story Cards

Although a seemingly primitive technology, index cards exhibit a number of outstanding benefits for development teams under constant time pressure. Probably the principal benefit is the ease with which a new story card can be prepared. Other benefits include ease of manual storage, manual grouping, laying out on a table surface for planning and pinning to a notice board for developing and stand-up meetings. Nevertheless, there are some down-sides to index cards though. Table 1 lists the benefits and disadvantages for comparison purposes.

Table 1. User Story Card Benefits and Disadvantages.

Advantages	*Disadvantages*
Create new story card in a short time by hand	No support for auto numbering
Easy to group/order in box or with rubber bands	No copy/paste
Small enough to layout all cards on a table	Deletion/amendment difficult or messy
Easy to display on a notice board in groups	
Only one copy prevents duplication	Single copy prevents sharing
Drawings easily incorporated	Possible to lose single copy

The lists in Table 1 were derived from the standard literature for Extreme Programming, and also an online survey designed by the author using the built-in mechanisms of Microsoft SharePoint Team Services [7] . This mechanism is a very straightforward way of eliciting information of all types from development team members.

Whereas the use of index cards for user stories has achieved wide acceptance, Table 1 lists a number of deficiencies. The existence of a single copy appears to be the major problem, for example when a customer is asked to split a story into smaller stories they effectively 'check out' the story and prevent access by the developers. In the Elemental Project [6] developers usually resort to photocopying to alleviate the single copy problem, and this immediately introduces the duplicate update problem.

5. Distributed Development Teams

As soon as the development team or teams become physically remote, even at a single site, the single-copy index card problem requires a different solution. There simply must be more than one copy of each user story, or an electronic copy that can be shared.

It should be stated at this point that Cockburn [1] makes several valid points about this remote development situation which he refers to as *virtual teams*. He does distinguish between *multisite development* (larger team working in a few locations with customers and developers in all locations), *offshore development* where the team is literally split between remote sites usually in other countries, and *distributed development* where the team is distributed across many sites with a few people, maybe one person, at each site.

Cockburn points out the enormous communications barriers that exist with virtual teams, and the high cost per 'idea transfer'. Particularly with distributed development a great deal of effort must be expended in real-time communication such as phone and chat as opposed to emails.

As Cockburn does himself, it is very well worth mentioning here the apparently very successful distributed development undertaken by the Open-source community. Cockburn dismisses this effort in the Agile context by claiming that Open-source development is 'non-resource constrained'. While undoubtedly correct, this view masks some Open-source development software that can very well be employed in the Extreme Programming user story context.

As pointed out by Augustin et al [8] the Open-source community manage to recruit the right experts, ensure they work together from anywhere on the planet, communicate asynchronously with common web-based tools using a shared process encompassed by project and knowledge management skills. These authors all work at VA Software Corp, the company that supports the highly

successful Souceforge.net collaborative software development facility. The tools available at the sourceforge.net site are truly impressive. Significant tools include the full array of Internet communications mechanisms, shared source code control systems, version distribution filestore, knowledge management tracking, project web server, shell and compile farm, and database services.

Considering just the user story context addressed in this paper, one of the many Sourceforge tools, the issue tracker, encompasses several of the requirements of a user story tool. A section of the web page for issue tracking is shown in Figure 2. Here is an off-the-shelf tool that potentially be used to support user stories. Although lacking some of the required fields for user stories, these could be added via the 'Add A Comment' field in Figure 2.

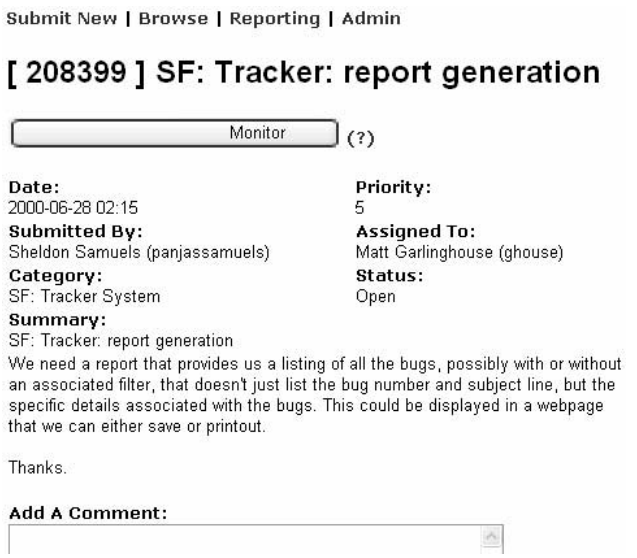

Figure 2. Sourceforge Issue Tracker.

Another simple approach to providing a read-only sharing of user story index cards is provided by a software tool primarily designed for recording the contents of regular whiteboards. The Whiteboard Photo application [9] expects a rectangular image in each digital photo taken from any angle. The application proceeds to identify the border and then skews the image to represent a perfect 'square-on' image suitably adjusted for contrast. The cleaned up image is shown in Figure 3. Although difficult to read at the size presented in this paper, the image is very clear at normal screen sizes and is about 7% of the size of the original image.

This same product, Whiteboard Photo, can also play a significant role in the Extreme Programming planning game where whiteboards are almost universally used for developing architectural designs that must be turned into user stories. In effect, the product turns any regular whiteboard into a full-color electronic whiteboard.

At the recent Agile Australia road show [10] the distinguished speakers were asked of their experiences in the use of software tools for user stories in distributed teams. Several common tools were mentioned. They are listed in with their relative strengths and weaknesses.

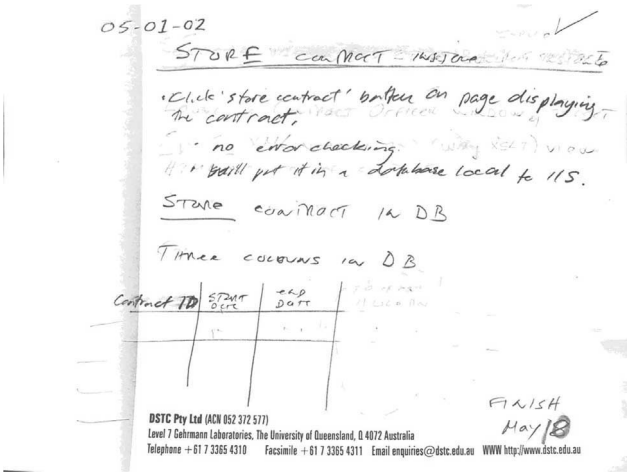

Figure 3. User Story Card Photo Processed by Whiteboard Photo.

There can be no doubt that all of the tools in Table 2 are easy to use and familiar to many users. (Wikis may not be familiar to all readers [11] .) When these tools are used in lieu of user story cards, they all suffer from gaps in their feature set. Nevertheless all have been used to support distributed user story development within Extreme Programming.

Table 2. Potential User Story Tools for Distributed Development.

Tool	*Benefits*	*Disadvantages*
Spreadsheet	Easy re-ordering & grouping; built-in searching	Tabular form constraining; difficult to control multiple update
Wiki	Multiple update fine; automatic linking useful	Layout too unconstrained; can't easily group stories
Database (Access)	Data record is good card metaphor; form entry very easy; multiple update no problem	Poor group visualization of all cards

6. Specific User Story Tool

Having considered potential electronic equivalents for story cards, the author determined to attempt the implementation of a specific user story tool eliminating as many of the defects in Table 2 as possible, while at the same time retaining the majority of benefits of index cards. An added incentive is the fact that the author's link with the Elemental project team described above is at a distance.

Of all the software tools discussed so far, the web-based issue tracker appears the most attractive with its universal access but control of editing options, provided, of course, specific user story fields are available. Most traditional web applications however are not able to support anything that simulates the movement of story cards across a flat table in a convenient manner.

As it happens, the author has been developing web applications for several years that allow the page content to be edited in various controlled ways [12] [13] . The DotNotelets tool for creating concept maps mentioned by the author [14] fits the requirements for a user story tool quite tightly. A further development of DotNotelets is the generic DotTegular tool for creating simple whole web sites. DotTegular is as yet unreported but its feature set fits even more closely with a user story card tool. Some details of the collection of Dot tools can be found at the On-The-Dot Software web site [15] .

Each *teglet* in DotTegular corresponds to a small rectangular area on a web page. The user can create, delete and edit the contents of each teglet using the powerful DHTML editor built in to the Internet Explorer browser. Actions such as splitting stories into an increased number of simpler ones become trivially easy operations. More importantly, users can drag teglets around the page in a similar manner to laying out cards on a table. This is probably the one feature that currently makes development teams retain the use of index cards.

While the contents of a teglet are free form, the use of templates allows the necessary fields expected in a user story to be incorporated easily. Again, very importantly, additional content can be added to suit the large variations in information recorded for each user story. This flexibility is often the key to the ready acceptance of a software tool, and overcomes the inflexible nature of database forms and spreadsheet cells, for example. In this regard, teglets can offer the content freedom of Wiki pages while allowing user story templates to suggest required content.

Traditional web page access mechanisms can be imposed so that only the appropriate members of a development team are allowed to edit the user story content. However, Extreme Programming encourages all team members to be designers and have input into user story production, the standard software tool would normally have no edit restrictions. Indeed, to foster trust between customers and developers, all should be free to change any aspect of a user story.

Many of the texts on Extreme Programming encourage the tearing up and rewriting of user story cards to ensure sensible changes of direction and refactoring. In light of the dearth of user story examples, it seems an unnecessary waste of effort to actually delete user stories that initially had some value. The tool about to be described in detail assigns a globally unique identifier to each user story teglet. By archiving all unwanted user stories, it is feasible to make these available in a global repository when one becomes available.

7. DotStories

The software tool offered by the author as a feasible replacement for user story index cards is named DotStories. Being based on DotTegular, the DotStories tool offers any number of web 'sites' with the intention that a web site corresponds to a single software development project. Each web site is a collection of user story groups or web pages. Finally a web page contains any number of user stories. Such a three-level organization is adequate for fairly large projects of several hundreds of user stories. It is possible to have multiple installations of DotStories on a single web server so that a fourth layer can be added if needed to take the number of manageable user stories to about the ten thousand.

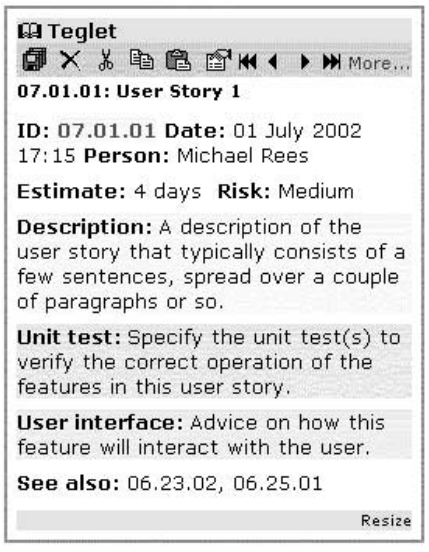

Figure 4. A User Story Teglet.

A suggested layout for a *user story teglet* is shown in Figure 4, and as can be seen there are several distinct areas:

- Title bar that indicates the type of teglet and allows the teglet position to be dragged around the page for grouping user stories.
- A toolbar that gives access to editing tools in a minimal and extended format.
- The main editing area that provides rich editing DHTML facilities that represent a significant subset of those found in web page publishing packages.
- The Resize bar to allow the dimensions of the teglet to be resized.

All the edited features in Figure 4 can be achieved with the editing tools in the extended editing toolbar. The main page (user story teglets collection) manager is shown in Figure 5.

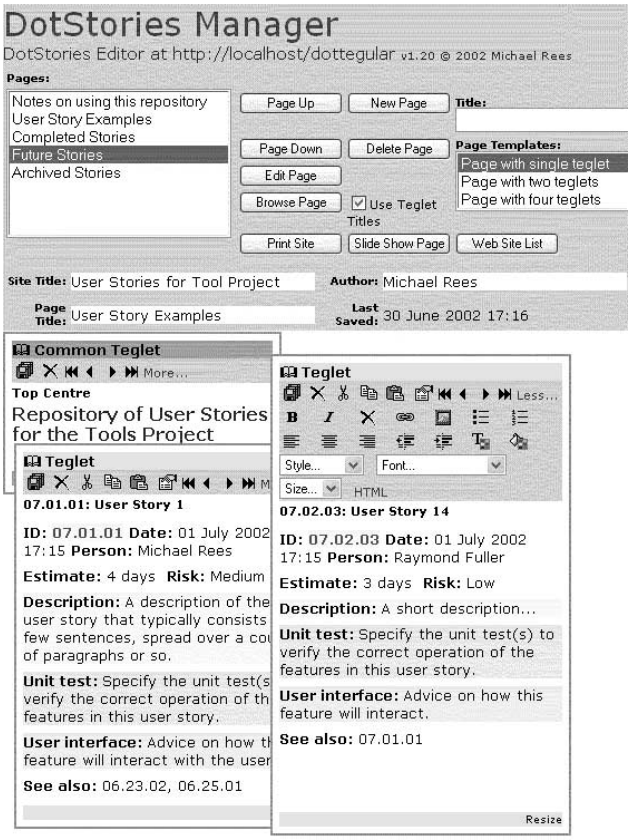

Figure 5. Managing User Story Teglets on a Page.

Bearing in mind that teglets can be resized to smaller areas it is feasible to fit up to about 50 on single page that represents a major grouping. The second teglet towards the right of Figure 5 shows the extended editing tools which can be hidden by the user when not in use. Editing tools are available for:

- Hyperlink and image insertion (absolute URLs only are supported currently)
- Bulleted and numbered lists to any level of nesting
- Text alignment and undent/indent
- Text and background colors
- HTML styles, font faces and font sizes

Not easy to see in Figure 5 is the ability to bring up a text editor so that a user experienced in HTML can edit at that basic level. This allows the user to insert any valid HTML such as tables, form elements, frames and the like. Of course, such detailed editing is not recommended for the general user of this tool.

The top sections of Figure 5 show the simple management tools that allow new pages to be created, existing page deleted, and the page order to be defined.

The smallest teglet shown with the dark title area is an example of a common teglet. This type of teglet is used to incorporate common content in the top and bottom left, center and right of each user story teglet collection when viewed at the group level. Common teglets are used for title, date and logo images, for example.

Right clicking the mouse on a page gives access to important features such as:

- New teglet creation
- Copy and paste of whole pages (teglet collections)
- Saving the teglets
- Automatically rearranging the teglets on a page

Further buttons in the manager section allow the pages to be displayed in a number of different ways. Space does not allow screen dumps here, but textual descriptions are given:

Browse mode: the teglet contents are displayed on a single web page in an order specified by the user. A simple top-to-bottom and left-to-right ordering is implemented. Thus if the top of two teglets coincide, the one furthest to the left appears first. This appears to be an intuitive order, and users react to it well. Links to all other pages appear at the left, so the user can easily and quickly browse to any user story collection.

Slide Show mode: every teglet, ie user story, becomes a separate slide that occupies the whole web page. A special style sheet automatically expands the font size to make the text appear more like a slide. This mode is intended for use with a data projector. A special navigation bar appears at the bottom of the page to allow navigation to the next/previous user story and a selection list to show any story directly. The cursor keys also affect the navigation in this view.

The slide show mode can also be useful during the daily stand-up meetings where a projector can be used in place of the static notice board.

Printable mode: all teglets are concatenated in this mode so that all stories in all groups appear on a single web page suitable for printing. Obviously this can be used as a documentation aid, and to provide snapshots of the state of the project at any time.

The default page in each DotStories web site generates a listing of all the possible modes for each user story project collection. It is a simple matter to click on the required display format for the user story collection required. Once in a project collection of user stories, the user may follow the links at the left of the browsed page to move to project groupings of user stories. Thus locating individual user stories involves at most three clicks and maybe a partial page scroll. A simple search mechanism allows a keyword search of all user stories.

With this range of features all the basic requirements of a user story tool are fulfilled. A page group of teglets is shown in a small scale in Figure 6. This is a partial page dump of a larger number of user story teglets to give a better impression of how the DotStories tool will appear in practice.

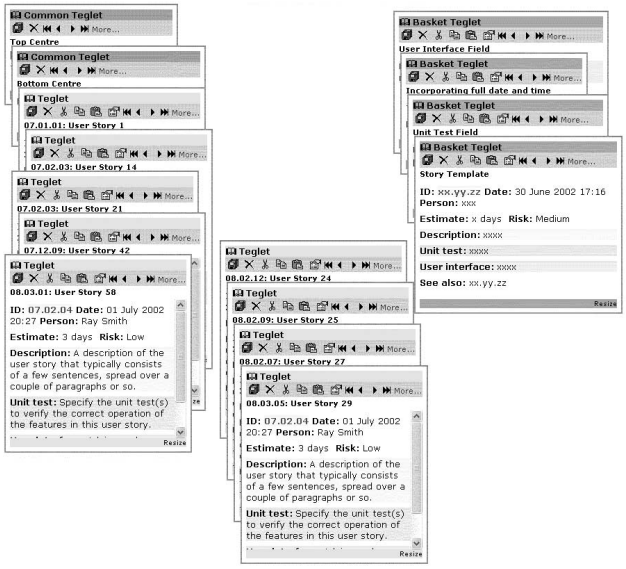

Figure 6. A User Story Teglet Group with Basket Templates.

Figure 6 shows examples of the third type of teglet, the scratchpad or *basket teglet*. The user has the option to make the basket teglets visible or not. Basket teglets are common across all web sites, ie across all project collections of user story teglets. It is intended for basket teglets to act as templates; so for example, the basket teglet fully exposed on the right of Figure 6 is a full user story teglet. The user simply duplicates this basket teglet using the editing tool provided in the toolbar to create a new regular user story teglet in the current collection. The user story teglet can then be edited for the new user story details.

Two subgroups of user story teglets can be seen in Figure 6 on the left and center. By simply dragging the title bar of any teglet, the user can switch the user story between subgroups or create new subgroups.

DotStories stores each project user story collection in a separate XML file. The content of each whole user story teglet appears as HTML contained in a CDATA section within the XML elements. This straightforward format allows the XML files to be exported and imported into other DotStories installations with ease. However, this is a simple proprietary format and does not allow user story exchange with other developer teams not using the DotStories tool. A more portable form of XML representation of the user story collections is needed.

8. Markup Language for User Stories

To aid in the global exchange of user stories, a purer form of XML is generated by DotStories. This format is tentatively called Markup Language for User Stories (MLUS). (USML would be a better acronym but is already used for UDDI Search Markup Language!)

In DotStories, a command from the context menu can be used to generate MLUS for all user stories in a project collection. MLUS still employs CDATA sections for the user story description, unit test and user interface requirements, as there can in general be no agreed structure to these descriptions. A fragment of MLUS is shown in the text box below.

Each user story has the ID assigned within the project, but DotTegular and DotStories always generate a unique global ID (GUID) for every teglet internally. This GUID is added as an attribute to the XML element for each user story. As will be the intention for DotTegular which aims to foster the reuse of information teglets, the GUID can be used to lodge every user story as a unique entry in a future global repository of user stories. Reuse, or at least sharing of experience, then becomes possible amongst Agile development technology proponents. Also, not the least useful manner of exploiting the GUID is the creation of an archive of all old, discarded user stories for each project.

9. User Story Tool Outcomes

At the time of writing DotStories is very close to a beta release with all the features described in previous sections. Online help information and the development of a useful collection of user story templates and basket teglets remain to be written. Developing a series of table fragments included in basket teglets would allow a greater range of non-textual content to be inserted and edited within user stories, hopefully approaching the capabilities of the tabular diagrams that handwritten index card user stories can provide.

Some preliminary tests of DotStories for user stories for its own development have been used. As usual with such early tests, a number of improvements to the intricacies of the user interface have come about. Interlock checking, that has already been incorporated into earlier

tools like DotNotelets and DotEdit [12] still must be added. Such checks prevent two users making conflicting changes to the same user stories.

```
<StoryCollection title="User Stories for Tool Project"
   lastsaved="200207011715" author="Michael Rees">
  <StoryGroup title="User Story Examples">
    <Story title="07.01.01: User Story 1" id="07.01.01"
        guid="{0CD89310-498C-4F3C-AB89-74C3E099C403}">
      <Estimate>4 days</Estimate>
      <Person>Michael Rees</Person>
      <Risk>Medium</Risk>
      <Description>
        <![CDATA[<P><STRONG>Description: </STRONG>A
description of the user story that typically consists of a few
sentences, spread over a couple of paragraphs or so.</P>]]>
      </Description>
      <UnitTest>
        <![CDATA[<P><STRONG>Unit test: </STRONG>Specify
the unit test(s) to verify the correct operation of the features in this
user story.</P>]]>
      </UnitTest>
      <UserInterface>
        <![CDATA[<P><STRONG>User interface:
</STRONG>Advice on how this feature will interact with the
user.</P>]]>
      </UserInterface>
      <SeeAlso>06.23.02, 06.25.01</SeeAlso>
    </Story>
    <Story title="07.02.03: User Story 14" id="07.02.03"
        guid="{31AEB20F-B952-4C9F-AB30-B7DF268A0ED9}">
    ...
    </Story>
  </StoryGroup>
</StoryCollection>
```

DotStories is mainly implemented as a web site containing a large body of JScript functions embedded in a series of .html pages with some .asp pages to manage the XML files on the server. Thus Internet Explorer 5.5 or later is needed as the thin client software, particularly for the DHTML editing component it contains. While perhaps 75% of general computer users would have no difficulty with this, a much lower percentage applies to software development teams. However, since the display modes outlined in Section 7 are read-only, it is a small matter to convert the JScript to JavaScript that is compatible with other browsers. This will still allow the whole team to share the user stories in reading and browsing mode.

An influential paper by Elssamadisy & Schalliol [16] describes the use of Extreme Programming for a considerably larger project. Amongst other observations, they reported their first 'bad smell' at the user story creation level as many iterations pass on the large scale software. Their solution is to alter the granularity of user stories as later releases are reached. This means that the information fields must change in each user story as time passes. DotStories is ideally suited to this as new basket templates can be evolved in time with the rethinking of the user story granularity without breaking the tool.

It is quite possible to print out user stories from DotStories in a form suitable for pinning to a notice board. Thus the standard daily stand-up meeting can still take place next to the notice board. This seems to defeat some of the benefits that DotStories brings. It is suggested that a better method is to use the notice board as a screen for a data projector, and use the various DotStories display modes, particularly slide show mode, to allow development team members to bring up the particular user story with ease as required in the reporting of progress. Such a projector is also of great use during the planning meetings so that all present can see the shuffling of the user stories around the web page to correspond to the next and future iterations.

This discussion of user story tools is not complete without a mention of the Storm user story tool under development as an Open-source project at Sourceforge.net [17] . Storm is a web-based user story tool along the lines of DotStories, but has a much wider range of capabilities such as:

- Every user has an account to allow access control to user stories within and between accounts, and to provide presence information
- Release management with user stories sorted by risk, completeness, value or by date of change
- Remote files can be uploaded an linked to each story
- User stories can be linked to user stories from other webs like Bugzilla or ViewCVS
- User story version control

These features are very impressive but, inevitably, increase the complexity of the user interface substantially. This drives the users away from the simple, lightweight nature of Extreme Programming. It may be unfortunate that the screen dumps provided on the Storm web site bring to mind the dreaded Microsoft Project application, the bane of all practitioners of Extreme Programming. Nevertheless, Storm will offer a significant range of user story support features. Release 1.0 appeared in March 2002.

10. Conclusion

The use of index cards for user stories in many Extreme Programming software development projects has been widespread. So far, only when the project involves virtual teams with geographically remote team members has it been necessary to look for software tools as

alternatives. Several popular tools like spreadsheets, databases and Wikis have been used in a successful fashion, but all lack some of the powerful features for ordering and layout that index cards possess.

This paper has described the layout requirements and discussed their implementation in the DotStories user story tool. By using the concept of teglets that can be dragged into position on a web page, DotStories is able to simulate the iteration planning process, and thus improves upon the popular tools just described. More sophisticated layout aids and intelligent clustering algorithms remain to be added to DotStories, and the outcomes of these designs will be reported in future papers.

The experience with DotStories and the coming, extensive Storm user story tool indicate that it is feasible to contemplate improving upon user story index cards in Extreme Programming. The DotStories web application already approaches the ideal user story tool.

11. References

[1] Cockburn, A., *Agile Software Development*, Addison-Wesley, 2002, ISBN: 0-201-69969-9.

[2] Beck, K., Extreme Programming Explained, Addison-Wesley, 2000, ISBN: 0-201-61641-6.

[3] User Story Examples, available at http://c2.com/cgi/wiki?UserStoryExamples.

[4] Beck, K. & Fowler, M., Planning Extreme Programming, Addison-Wesley, 2001, ISBN: 0-201-71091-9.

[5] Jeffries, R., Anderson, A. & Hendrickson, C., Extreme Programming Installed, Addison-Wesley, 2001, ISBN: 0-201-70842-6.

[6] Milosevic, Z., Project Leader, Elemental Project, DSTC Pty Ltd, CRC for Enterprise and Distributed Systems Technology, http://www.dstc.edu.au/Research/elemental-ov.html.

[7] Microsoft SharePoint Team Services, http://www.microsoft.com/frontpage/sharepoint/.

[8] Augustin, L., Bressler, D. & Smith, G., 'Accelerating Software Development through Collaboration', Proceedings of ICSE'02, May 19-25, Orlando, Florida, pp 559-563.

[9] Whiteboard Photo, http://www.websterboards.com/products/wbp.html.

[10] Agile Australia 2002, Speakers and Presentations, http://www.synop.com/topic/index.phtml/id/140.

[11] Cunningham, W., WikiWikiWeb, http://c2.com/cgi/wiki?WikiWikiWeb.

[12] Rees, M. J., "Implementing Responsive Lightweight In-page Editing", Proceedings of AusWeb 2000, Cairns, June, 2000, pp 134-142.

[13] Rees, M. J., "Implementing Shared Document Preparation with Lightweight Editing", Proceedings Fifth Australasian Document Computing Symposium (ADCS), December 2000, Twin Waters Resort, Sunshine Coast, pp 25-33.

[14] Rees, M. J., 'Evolving The Browser Towards A Standard User Interface Architecture', User Interfaces 2002, Third Australasian User Interface Conference, Australian Computer Science Communications, Vol. 24, No.4, pp 1-8.

[15] Rees, M. J., On-The-Dot Software, http://comet.it.bond.edu.au/dot/.

[16] Elssamadisy, A. & Schalliol, G., 'Recognizing and Responding to "Bad Smells" in Extreme Programming', Proceedings of ICSE'02, May 19-25, Orlando, Florida, pp 617-622.

[17] Storm User Story Tool, http://xpstorm.sourceforge.net/.