CMAP
A Collaborative Multitouch Agile Planner
An Investigation into the use of multitouch enabled devices for collaborative Agile software design planning meetings

Stevenson Gossage and Robert Biddle

Carleton University, Shcool of Computer Science, Ottawa, Canada
`{ralf.gerstner,jutta.jaeger,lnbip}@springer.com`
`http://www.scs.carleton.ca/`

Summary. The main goals of this project were to explore the use of multitouch enabled surfaces and how this technology could be leveraged to create a new level of usability in terms of human computer support of group collaboration in a design process setting. Furthermore it was desirable to develop a fully functional system that could be used to help in the design and maintenance of any type of project. It was hoped that the process of developing such a system would deepen our understanding of the complexities involved in preserving and enhancing the human communication needs in a complex computer supported environment.*abstract* environment.

Key words: Agile, Planning, Design, Multitouch, Stories, Storycard, Cardwall, Digital table, tabletop

Contents

1.1 Introduction

1.1.1 Scope

This project focuses on two particular aspects of Agile software engineering, the user storycard and the cardwall. Both are physical artefacts used in traditional pen and paper type, Agile planning meetings. They support lightweight processes used for estimating and planning throughout the entire life cycle of a project. They have proven to be critical to the success of many projects, viz. The technique of expressing requirements as user stories is one of the most broadly applicable techniques introduced by the Agile processes [9]. The cardwall acts as an Information Radiator [7] that is on constant display, is intended to be used in a public place where information is gathered and where discussion is encouraged.

1.1.2 Motivation

Previous attempts to digitise this Agile software design process have fallen short, mainly due to the paradigm in which they were developed. These tools have been designed for the traditional single input, one device at a time, type computer and therefore have failed to capture the flexibility and simplicity afforded by pen and paper, in many respects losing the human factors that ensure that the process works e.g., "every attempt Ive seen to computerise stories has failed to provide a fraction of the value of having real cards on a real wall" [6]. For these reasons, lightweight tools for software design and development were chosen as the domain for this project. In particular, the project explores tools designed to leverage the advances in the latest multi-touch capable digital tables and walls. These advances are the key to developing truly innovative, intuitive user interfaces for the purpose of designing and maintaining Agile software projects.

Of particular interest is the exploration of these new technologies and how they can be harnessed by developers to create new tools for the design process. Tools that can realistically replace traditional paper based activities must adopt a value-added philosophy where the tool makes no attempt to replace the human users, rather, its aim should be to co-exist or facilitate the process, ensuring that human factors inherent in the activity are preserved. Of utmost importance in accomplishing this task is the true understanding of what makes these lightweight processes so successful. In fact the cognitive and social aspects of the process are fundamental to these systems and if the tools are not designed with this in mind they will never hope to completely replace their pen and paper counterparts. [14]

1.1.3 Method

Keeping this in mind, the goal of this project was to create a tool to support Agile software design methodologies. In particular we wanted to create a tool that could digitise the traditional use of the Agile storycard and cardwall using a multitouch enabled surface.

As touched upon previously, the exploration of the possibilities that the latest in multitouch enabled hardware devices afford was, at least as important as the final software package itself. Along with these new devices are a plethora of options available to develop multitouch applications. Everything from open source frameworks, Dot Net libraries and proprietary industry APIs; from which we choose to leverage the power of the open source community and the Python programming language where we found an excellent framework named PyMT, a framework designed for the rapid development of multitouch UI prototypes.

1.1.4 Research Question

Using this framework we wanted to see what kinds of new interfaces we could create that would enable the implementation of this software in a way that current users of these methodologies could transparently transition from their paper world to our digital counterpart. The idea was to create a digital storycard application for multitouch surfaces that would encourage a communicative Agile approach. The software should provide development teams with a useful collaborative tool for both co-located and geographically dispersed teams; ultimately facilitating a rapid and efficient design and development process.

1.2 Background

1.2.1 Agile Methodology

Today, many lightweight Agile processes have emerged, however, for the purpose of this paper we consider the two most common, Extreme Programming [5] and Scrum [1]. Both of these methodologies take an iterative approach where less focus is put on upfront analysis and more on quick consistent deployment of quality working software. One important difference between these two methodologies is the point at which the software is actually released to the customer and the distinction between XPs iterations and Scrums sprints. In Extreme Programming, working software is delivered to the customer at the end of every iteration, typically a one or two week cycle. In Scrum, the completion of each sprint signifies that there is working software that is ready to be released to the customer but does not have to be; a sprint typically lasts about one month. Keeping this in mind, the two terms are quite similar and this report will use the terms interchangeably. Figure 1.1 compares a two year project using XP, the traditional waterfall and Scrum methodologies. Notice the biggest difference is the number of deliverables. The storycard and cardwall are physical artefacts that help support lightweight Agile processes. The storycard (see Figure 1.2) and the cardwall (see Figure 1.3) are tools designed to help in the gathering of requirements, prioritization and scheduling of work as well a means to quickly amass critical information about the state of the project in a very public and transparent way. More importantly when used correctly they are the catalyst of communication between not only team members but also customers and other stakeholders. Initial relations between customer and developer are made when stories are first created, and continue as they are prioritized, scheduled and most importantly during active development. The wall on the other hand acts as an Information Radiator [7] and is invaluable in terms of the information that can be gathered at a glance. The wall quickly reveals whether or not a project is on schedule or falling behind. It conveys the progression of stories as they are physically moved to different locations. It can also be easily organized in a manner to show work completed, bugs, and work yet to be scheduled.

1.2.2 Traditional User Storycards

The storycard is a physical artefact usually similar in size to an index card, postcard or even a sticky-note. Typically, customers use storycards to record user stories. Stories use customer centric language to define features and requirements. The size of the storycard is deliberately small, preventing unnecessary verbose detail, The Card may be the most visible manifestation of a user story, but it is not the most important. [10]. Davies continues to explain that cards represent customer requirements rather than document them. This

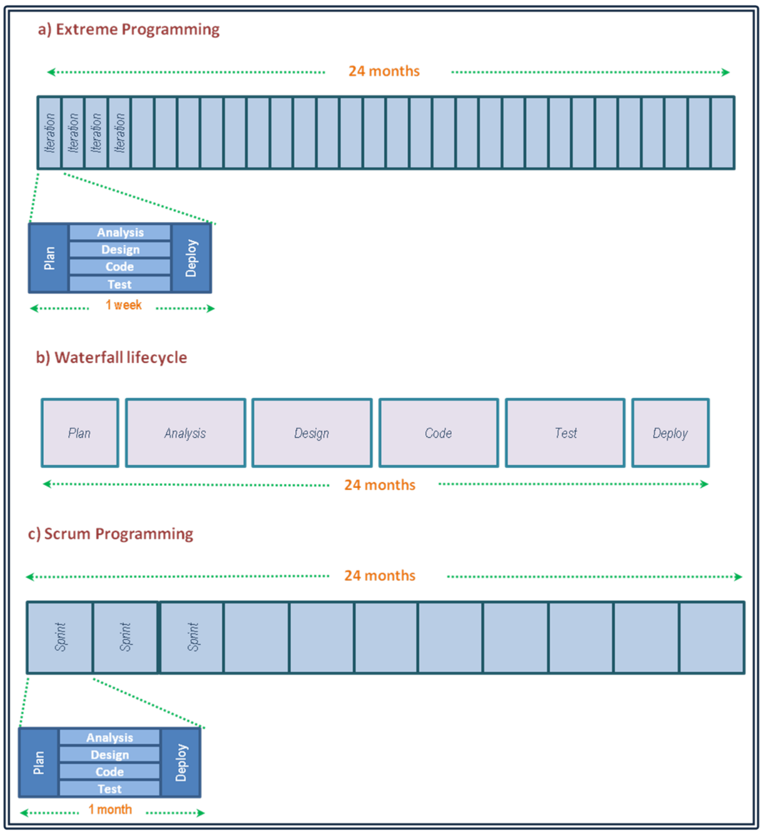


Fig. 1.1. Comparison of XP, Waterfall and Scrum

emphasises that the actual text on the card is simply a reminder or placeholder while the details are worked out in the Conversation and recorded in the Confirmation. [8] In the process of creating stories the following should be considered, Words, especially when written, are a very thin medium through which to express requirements for something as complex as software. With their ability to be misinterpreted we need to replace written words with frequent conversations between developers, customers, and users. User stories provide us with a way of having just enough written down that we don't forget and that we can estimate and plan while also encouraging this time of communication. [8] Jeffries describes the procedure as the combination of the three Cs, Card, Conversation and Confirmation [11]. The process starts with the assignment of a story to a developer; the physical possession of this card by a developer is a warrant that secures the conversation (and the confirmation process of the acceptance test) with a customer[14]. The subsequent communication between developer and customer explores the details of the

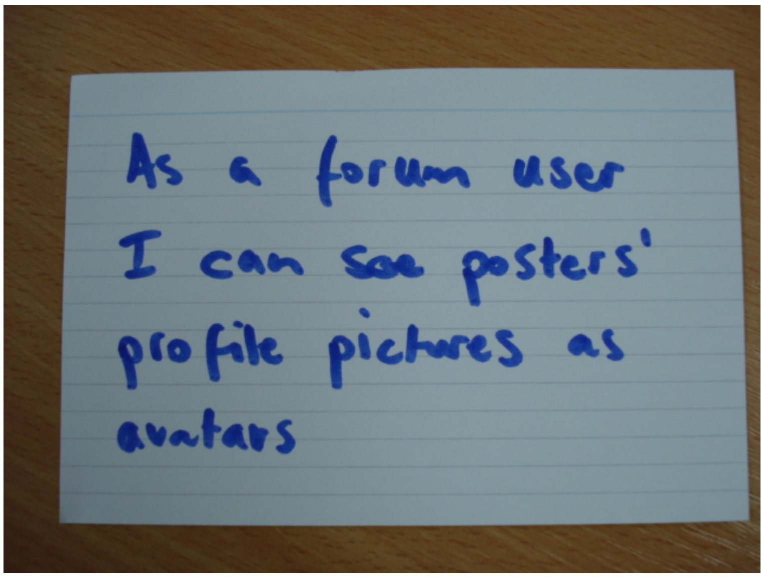

As a forum user
I can see posters'
profile pictures as
avatars

Fig. 1.2. Storycard thanks to www.flickr.com user JRoos

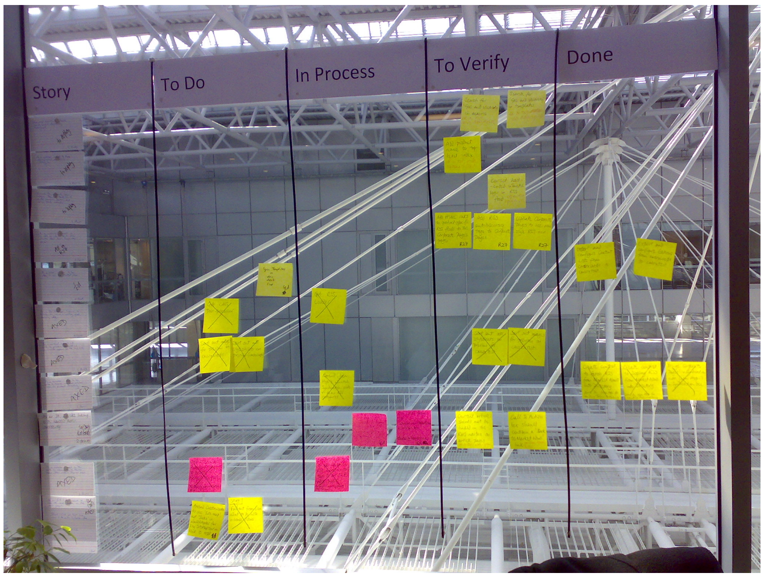


Fig. 1.3. Cardwall, thanks to flickr.com user owenblacker

story and the confirmation, should be mutually agreed upon, such that the storys completion is well understood.

1.2.3 Traditional use of the Agile Cardwall

The cardwall is typically used as a tool for the initial design and project management process and subsequently to re-organize, prioritize and manage new developments in an Agile project. The most common use tends to adhere to the following pattern; a project backlog (a collection of stories) is created which ultimately defines the scope of the project. Depending on the exact methodology used, initially as many stories as possible will be created in an attempt to get a complete view of the system and all of its requirements, or possibly only sufficient stories to get the project started. The physical representation of a backlog is a stack of cards on a table or a collection of sticky-notes on a wall. The stories in the project backlog are organized and prioritized into releases and sprints. A release is simply the term used to describe a deliverable working piece of code. It typically marks a milestone in the development of the product denoting that sufficient key features have been implemented and tested. Sprints are simply a cycle which is made up of a development phases and a retrospective phase. A sprint is a small unit of work typically lasting from one week to one month. The priorities assigned to the stories dictate which sprint they will be assigned to and thus when they will be implemented. At the end of each sprint there is a sprint retrospective which is where the progress is gauged, the process is analyzed and a general determination as to the overall health of the project is made. The sprint retrospective is the chance to learn from mistakes and correct them before they become a permanent part of the process. Figure 1.4 shows the basic workflow from a collection of stories in the backlog, to the selection of the most important stories for the sprint, to the actual development cycle or sprint and finally to the working piece of software which may or may not be delivered to the customer. This process is essentially repeated until the last sprint is done and the product backlog is empty. The cardwall will typically remain on display in a public area for the duration of the project so as to facilitate the rapid modification of stories, sprints and releases, due to changing requirements or priorities. Artefacts can simply be rearranged or completely removed, based on the new requirements. A natural question might be to wonder how this level of detail and complexity can be captured on a few cue cards pinned to a wall, and, maybe more importantly, how does this cardwall actually help software development teams to meet their goals and deadlines while producing quality code? Often, it is that human component that is the key to the success of the method. Any software trying to replace a physical task must consider this issue in depth and try and provide an environment, which supports and encourages those same interactions, which bring success to the physical task; Any Agile team looking to move towards digitising the teams support will

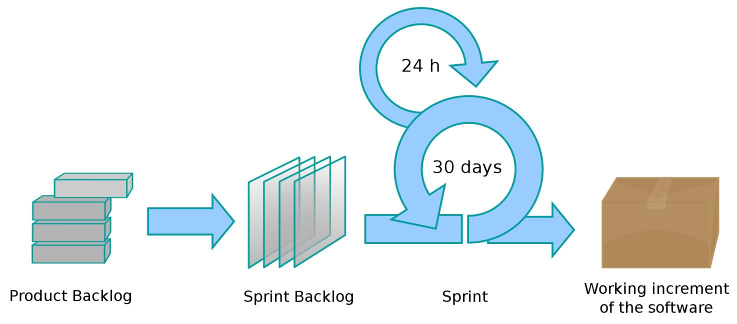


Fig. 1.4. Basic Agile Workflow, found @ `http://commons.wikimedia.org/wiki/File:Scrum_process.svg`

need to take account of the complex relationships that exist within this social system if they wish to retain key properties of successful teams. [14].

1.2.4 Related Work

There are a limited number of software design tools developed to harness the new possibilities in Human Computer Interaction afforded by the current technology in multitouch enabled devices such as digital tables and walls. The technology, although not new, is still in flux and continuously improving. Everyday more devices are being produced at a reasonable cost with support for two or more simultaneous touches; a critical feature for the development of truly collaborative tools. The best example of similar software is the Agile Planner for Digital Tabletop (APDT), [15]. APDT was designed based on a prototype by Weber [16] which was intended for co-located collaboration on a single touch surface. APDT chose to use this as a starting point but wanted to enhance it with support for multitouch, the ability to interface with other Agile planning tools and real world evaluation based on user studies; observing traditional Agile planning meetings as well as planning meetings conducted using DAP or Distributed AgilePlanner [12] As the name suggests DAP was designed to support distributed Agile teams in the planning and maintenance of an Agile project through the use of a digital whiteboard and storycards. Support for stories, their creation, deletion and manipulation as well as support for the digital wall where stories could be prioritized, assigned and scheduled. Finally, Dap also included visualizations using the whiteboard to quickly gather progress type information. Dap had been developed with a traditional single user interface paradigm (one keyboard, one mouse) such that users could collaborate from a distance but not so much in a co-located environment. APDT also studied and drew from the literature available on the use of multitouch tabletops in a collaborative but not necessarily Agile manner. APDT was developed as a multitouch enabled tool, specifically for two tables designed by Smart Technologies Ltd (Smart), using Smarts proprietary SDK. The first table used DViT (Digital Vision Touch) technology and had support for two concurrent touches. The second table used FTIR (Frustrated Total Internal Reflection) technology and had support for 40 concurrent touches. The two touch capabilities of the DViT table limited the users ability to work concurrently while the small form factor of the FTIR table meant that it was difficult to leverage its support for 40 simultaneous touches; since, only four adults could comfortably and concurrently interact with the table. APDT was a highly functional full featured tool, however, its dependence on the Smart SDK and therefore on Smarts hardware helped us make a key design decision; we wanted CMAP to be designed such that hardware and operating system independence was a goal, as well as support for multiple concurrent touches.

1.2.5 Theory

Why are physical storycards and walls so effective?

The following is a summary of some key points from the paper The role of physical artefacts in Agile software development: Two complementary perspectives [14] While a general template for stories usually exists such that key information is usually present like the As A¡Role¿, I want ¡Description¿, and So that ¡Benefit¿, the process is extremely flexible and we commonly find distinct notation across Agile teams. At the same time within any one team the notation and use of cards is strictly adhered to. Everything on the card has meaning which is not necessarily clear to an observer unfamiliar with the team specific notation. The location of, color, size of lettering and any annotations carry significant meaning and thus provide a high level of abstraction. A mature established team might have a well-defined notation while a new team might still be looking for what works best for them but the beauty of the physical cards is that they dont restrict a team to any one notation; thus teams are free to find their ideal solution.

The use of the wall is also an extremely flexible procedure but, has its generalities in that, teams use walls for the duration of a project and leave them on constant display somewhere they are easily seen, usually, in a common space where anyone walking by could get an idea of the progress of the team. The wall is generally regarded as an Information Radiator [7] and helps ensure the transparency of the project. Again, the wall is full of meaning not obvious to an observer who lacks familiarity with Agile methodologies or team specific notation. The key information that you should be able to gather almost instantaneously from any wall is the general progress of the project. The placement of stories signifies whether or not they are in active development, waiting to be started, finished etc. Again the wall is generally without structure when considering its use among distinct teams, but it is used in an extremely consistent manner within any one team.

Using cardwalls requires an active participation between customers and developers and was originally described as The Planning Game, [5] where the objective, basically would be, to prioritize the cards, sort them into releases and sprints, assign them to developers and have the developers accept or decline them until the cards were sufficiently sorted into at least the next sprint; while unassigned stories are left in the project backlog, to be dealt with at the next iteration or release planning meeting.

Maybe more importantly are the social interactions involved in the whole process; enabling teams to determine their best use of notation, annotation and layout. These interactions reveal the importance and meaning of the stories and thus drive their physical placement on wall, which, in turn drives the storys progress through the system. The physical size of the cards is also of utmost importance since it limits the information it can hold and therefore encourages a communicative dialog.

Digital Collaborative Workspaces

An important paper describing system guidelines for co-located collaborative work on a tabletop display, finds that there are eight key elements that must be addressed via the physical hardware of the tabletop, via the software being used on the tabletop or by a combination of the two [13]. The following is a summary of those key requirements and a brief description of each.

Support interpersonal interaction The technology must support the mediation of the collaborative interaction and must not interfere with this interaction. Ideally it should be as natural to collaborate around a digital tabletop as it is to collaborate around a regular table.

Support for fluid Transitions between Activities Switching tasks during collaboration should be as seamless as possible. For example, if the activity needs to combine data entry and the ability to draw, then switching between these activities must be a natural process. This allows the focus to remain on communication. The use of multiple input mechanisms must be considered but, should be a feature that enhances the overall activity and not a hindrance for these transitions.

Support for Transitions between Personal and Group Work If the collaborative task involves a combination of both personal and group work, the system should try and capture this by providing a similar mechanism. The physical shape of the table might be a key factor for this point because the individuals must feel comfortable and their personal workspace must not feel cramped or invaded. One suggestion may be to support external devices or displays that could be used in conjunction with the shared workspace of the digital tabletop. This approach may however, interfere with the fluid collaboration Scott suggests that this area needs more study.

Support Transitions between Tabletop Collaboration and External Work It should be easy to integrate previous work performed on an external device such as a laptop. If the activity requires that the participants have personal work (created elsewhere) that should be shared and or discussed; that work in whatever form it takes should be accessible to be viewed or manipulated in the shared workspace.

Support the use of Physical Objects The system should support both the use of external objects related to the task as well as items that are non-work related. For example there should be no negative impact on the system if a cup were to be placed on the digital tabletop. Furthermore it would be desirable if the system could differentiate between work-related and non-work related objects such that, that same cup, being placed in a way that obscured critical information might cause the system to react by redrawing the obscured part of the screen in a slightly different position.

Provide Shared Access to Physical and Digital Objects While collaborating around a traditional table, pointing and other gestures are usually easily interpreted by the group. On a digital surface this may not always be the case. A digital tabletop may provide several representations of the shared object (maybe one for each person) using gestures to try and point something out in this scenario could lead to confusion as users are force to interpret how the gesture may apply to their copy of the object. If however only one representation of the object is present, those same gestures may be a contributing factor to the overall understanding and group collaboration. The digital representation of shared objects is a key to the successfulness of the collaborative session. The designers must consider the potential physical locations of the participants as well as the possibility of obstruction by other users, objects or even gestures. Obvious examples of physical objects are digital artefacts like an IPad. Another good example of physical artefacts might be the pieces or tokens of a board game implemented for the table top. The system could be designed such that it supported real game pieces to be shared and interacted with throughout the course of play.

Considerations for the Appropriate Arrangements of Users People are generally more comfortable when they feel like their personal space is not invaded or at least not for a prolonged period of time. The system must be designed to accommodate this and allow users to comfortably interact with each other. Also, the intended audience is important because the physical difference between adults and children suggest that more children may comfortably interact around a table since they are physically smaller, let alone the childs tendency to want to be in closer proximity to their neighbours then most adults [4]. The system should also not be affected by participants repositioning themselves around the table. Virtual objects displayed on the table should be rotatable such that from anywhere around the table a participant could manipulate the object so as to get a more understandable view.

Support Simultaneous User Actions This point is critical to any collaborative system and one area where true hardware and software advancements have developed since the publication of Scotts paper in 2003. Current tabletop implementations support both multiple input devices and concurrent user input.

1.3 Design

CMAP, A Collaborative Multitouch Agile Planner was primarily, a project intended to create software designed specifically to digitise the storycard and cardwall; thereby facilitating the planning and management of the software development process of Agile development teams. One of the driving forces behind the project was the exploration of multitouch in terms of the specific advantages it brings to help system developers create interfaces that enrich group collaboration for the purposes of designing software with an Agile approach. During the implementation and testing of the initial system it became clear that there were flaws that needed to be addressed. These issues gave birth to a new design and consequently a second version of CMAP. In this section a discussion of the general design goals relevant to both versions will be outlined, followed by the specific design of each version and the reasons for the change. Finally, an attempt will be made to outline specific design elements that targeted particular aspects of Scotts guidelines for group collaboration around digital tabletops.

1.3.1 Basic Goals

- Support for Agile design methodologies and processes. In particular, the use of Storycards and the cardwall were essential.
- Support multiple inputs.
- Support multiple surfaces. The system should be platform independent.
- Support concurrent multiple user interaction.
- Support the use of gestures.
- Support the orientation independent use of the system.
- Support for persistent data.
- Support for shared data.

To support these goals several design decisions were made including:

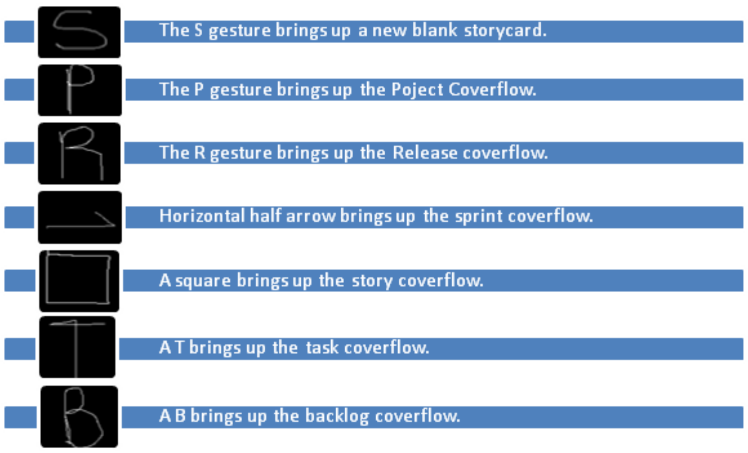
- The decision to use PyMT multitouch framework and therefore the Python programming language meant built in platform independence, multiple concurrent user interaction and support for multiple inputs. The PyMT framework allowed the use of gestures and rotatable widgets to deal with orientation issues. Furthermore, since both Python and PyMT are open source, we had the access and the freedom to change the code base if we found the need.

- The need to support persistent and shared data was achieved by using a combination of open source projects including Git, GitHub and Petaapans Google Application Server. This allowed for local storage of data that could be pushed to the GitHub server which would in turn request that the Google App Server notify its clients that new data was available to be pulled on GitHub. This process applied the publish subscribe design pattern.
- The Model View Controller pattern was used extensively as well as the Strategy pattern. These patterns allowed for view independence and for pluggable controllers which gave access to the models which were simple XML DOM objects. The use of the Document Object Model, XML and pythons introspection allowed for the real time addition of member variables to individual artefacts.(see Appendix C: Example XML Story Template)
- Through the use of command line options or a configuration file, the system was designed to work in many modes. This allows it to be used standalone in offline mode or individual components could be enabled to allow for full point to point data sharing or some subset. The key components that can be enabled or disabled are the use of the local repository, the GitHub server and the Google App Server. If all these features are enabled then data can be shared between remote copies of the application.(see Appendix B: Configuration file)

The use of gestures was an important feature and a unique gesture was provided as a shortcut to all of the most important activities allowed by the CMAP. The PyMT framework has built support for the use of standard gestures into many of their widgets, however, CMAP needed the ability to use new gestures no defined by PyMT. Figure 1.5, is a list of these custom gestures and a brief description.

Key decisions related to the support of Agile methodologies

- The wall traditionally gives structure to the stories and helps developers organise and manage their projects. To capture this functionality a hierarchical structure was created and the concept of type and a parent, child relationship was conceived for the artefacts. The idea behind this design decision was that artefacts could represent a project, a release, a sprint, a story or a task. Depending on its type, an artefact could have zero or more parents as well as zero or more children.
- The ability to group similar artefacts was considered important and the concept of a container was developed to be able to flip through different artefacts of the same type. This gives the system the ability to see for example all the releases in a project or all the stories in a sprint (see



The S gesture brings up a new blank storycard.

The P gesture brings up the Poject Coverflow.

The R gesture brings up the Release coverflow.

Horizontal half arrow brings up the sprint coverflow.

A square brings up the story coverflow.

A T brings up the task coverflow.

A B brings up the backlog coverflow.

Fig. 1.5. Custom Gestues

Figure 1.6, Figure 1.7 and Figure 1.8 for an illustration of how to flip through the contents of a coverflow widget).

- A global backlog was needed to contain unclassified artefacts that had no relation to a parent artefact. For example the creation of a story that was not part of any sprint, release or project would be placed in this backlog.

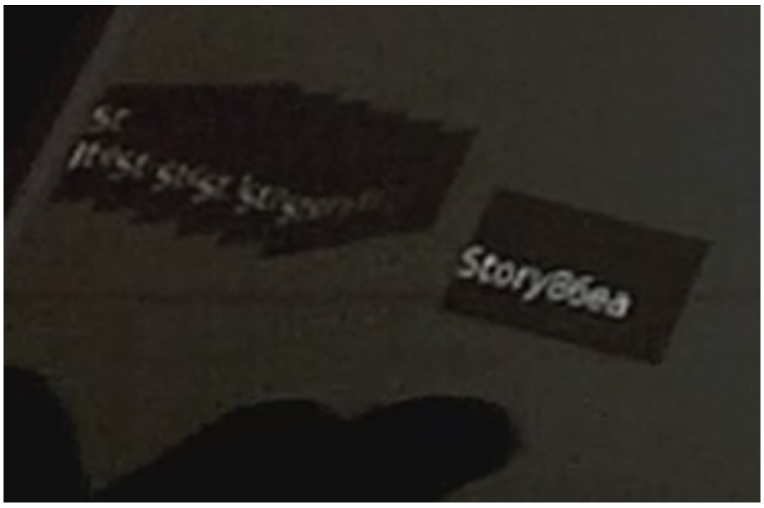


Fig. 1.6. Coverflow, all pages on left

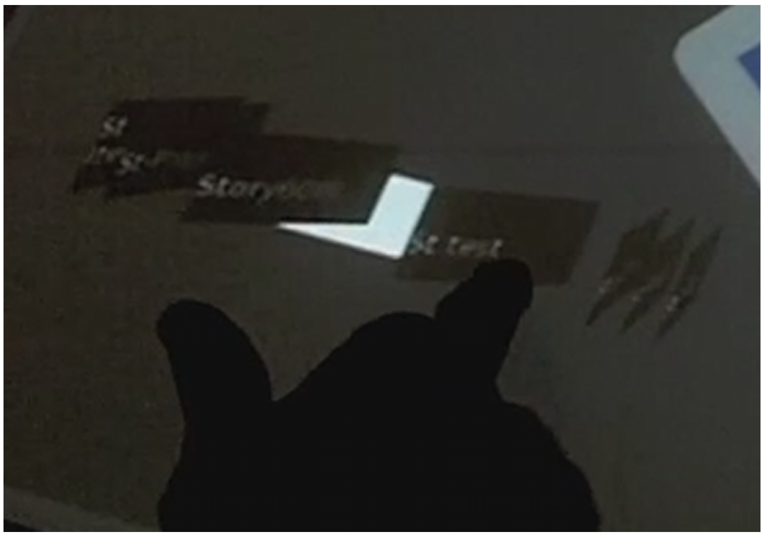


Fig. 1.7. Coverflow, flipping through the pages

1.3.2 Version 1 Design Strategy

This version took a structured approach to the Storycard and cardwall. This was an attempt to capture all the relevant data that seemed to be critical to the Agile planning process. XML was used to define the data of the Storycards and other artefacts and the view was a form based widget with labels and text fields. Users could use traditional data entry via real or virtual keyboards. In an attempt to limit the amount of information contained by an artefact, a default minimal view was created which only allowed the entry of a name for the artefact and a description. Figure 1.9 shows the minimal artefact view.

A second view was created for stories to allow the user to enter more information. This view was built so that the developer could, in conversation with the customer, elaborate on the information provided by the minimal view. Figure 1.10 shows this more detailed view of a Storycard.

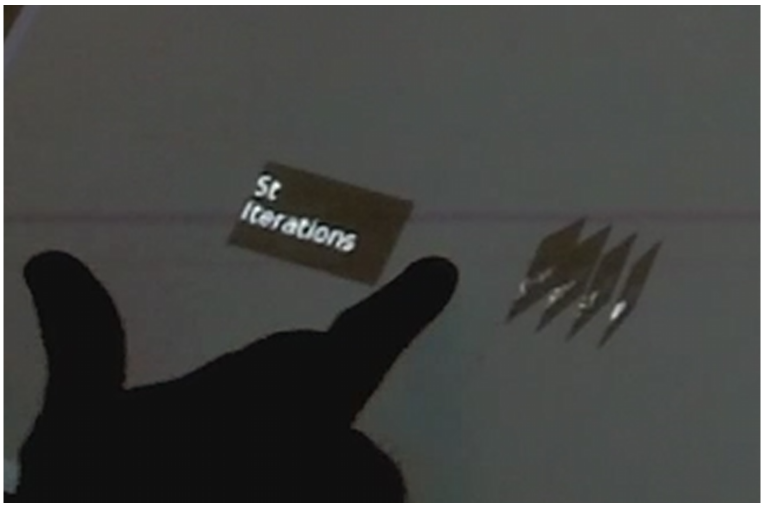


Fig. 1.8. Coverflow, all pages on left

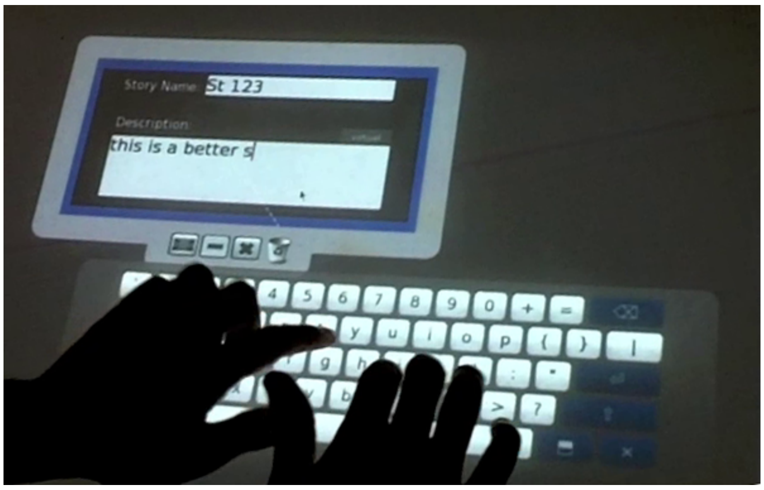
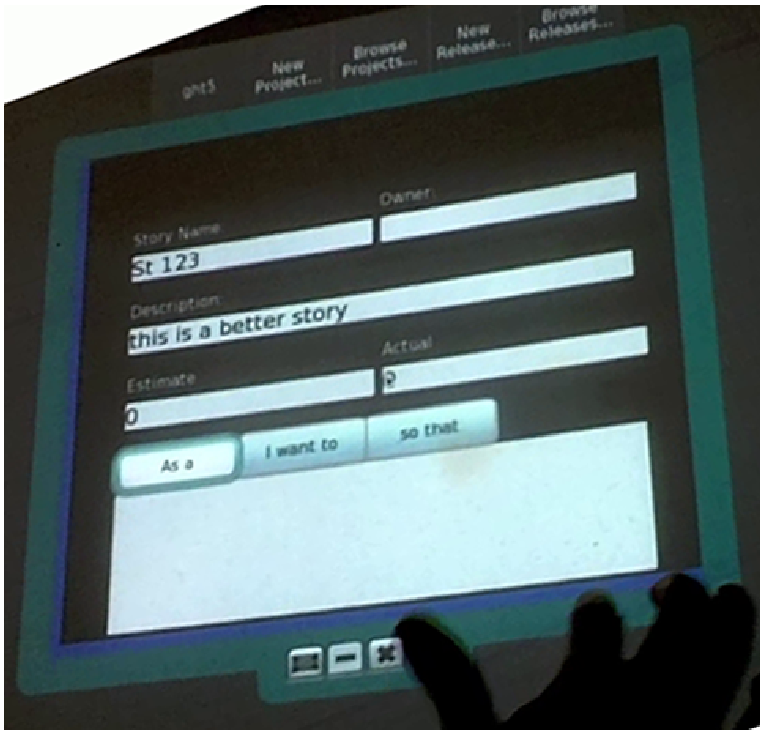


Fig. 1.9. Structured Minimal Storycard View

This structured approach seemed like a good way to capture the data and have it in a format that could easily be stored, shared and adapted to allow it to interface with other software tools like task management systems. This approach could give developers access to the stories directly from their IDE.



Owner:

Story Name
St 123

Description:
this is a better story

Estimate
0

Actual
2

Fig. 1.10. Structured Full Storycard View

1.3.3 Version 1 Walkthrough

To use this system the user is supposed to start by creating a project. Once created the next step is to create one or more releases, then one or more sprints, then one or more stories and finally if they choose to they could create one or more tasks. The only flexibility in this process is the ability to create stories at any point which are then placed into the backlog. Coverflow widgets were designed to hold artefacts of the same type. These containers are initially empty, except for the ability to create a new artefact of that type. So, to create a project the user could draw a P to bring up the project container. From the container clicking on the New Project button would bring up a new empty project artefact. This process makes the newly created project active such that a new release would automatically become a child of this project. This introduces the concept of the current artefact which is intended to control the hierarchical structure of the artefacts. For each artefact type, there could be a current artefact of that type. This allows the system to create the parent, child relationships of the artefacts automatically. For example once a project is created, it becomes active and will automatically become the parent of all newly created artefacts, if however, a second new project is created, it would then become the new current project and subsequent artefacts would become the children of this new project. To bring up the releases container one would use the capital R gesture. Again there is a button to create a new release. In a similar fashion the user can use the gestures outlined in Figure 1.5 to bring up the other containers, and create artefacts of those types. The capital S gesture was reserved as a shortcut to skip the story container and create a new story directly. As each artefact is created it becomes active and the creation of new artefacts either become siblings or children of the current artefact. However, if creating siblings these new artefacts then become the current artefact of this type. For example a User could go through the process up to creating a sprint, at which point the creation of many stories may be desirable, and as each one is created, it becomes current, allowing the optional creation of tasks for that story.

1.3.4 Version 2 Design Strategy

In the course of development the direction that was originally taken came into question for various reasons. Two important questions arouse as a result of attending the Surfnet Fall 2010 Workshop [3]. The first was a question of flexibility of user interaction with the system and the second was how might the rigidity of the design adversely affect or interfere with the human aspects of the process. In an attempt to address these concerns the original structured implementation was redesigned. The form based approach to the stories and other artefacts was replaced by a more flexible drawing type widget which was called a scribble widget. The intent was to allow users to enter anything they wanted on the cards, imposing no rules; much like the flexibility afforded them when using physical cards. The scribble widget allowed the user to scribble notes on the card anywhere as well as provided a mechanism to do text entry via virtual or real keyboards (see Figure 1.11 for the illustration). The text entered had the added flexibility of being able to be reposition anywhere on the card. There was also the ability to erase all or parts of the information including text. The XML representation of the data was only slightly modified to allow for these changes. Two new fields were added to accommodate the new information, including a list of points and text widgets, the colours, size and positions also needed to be stored. This allowed the data to be picked up by remote applications and the cards could be re-drawn as identical copies.

1.3.5 Version 2 Walkthrough

The procedure for using the system in version 2 is the same as in version1. Containers and gestures are still the key elements for accessing and creating

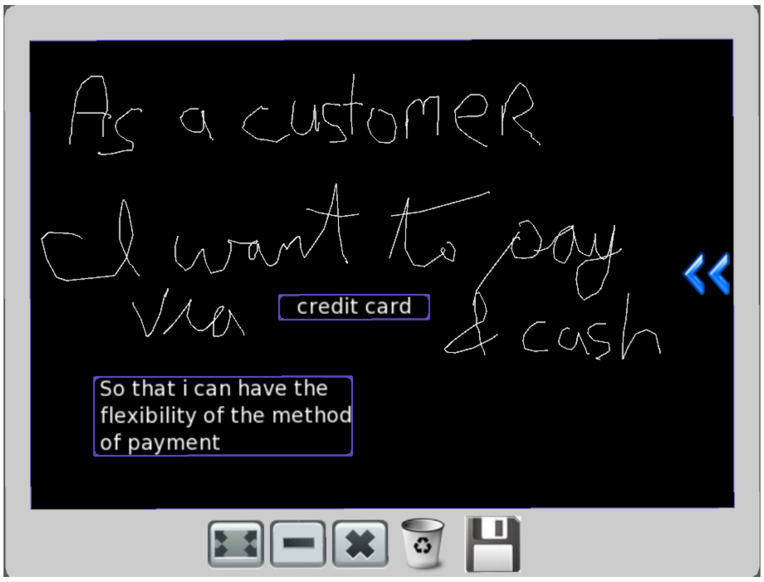


Fig. 1.11. Scribble Storycard

new and existing artefacts. Besides the new scribble widget the only other change was the ability to create tasks at any point such that tasks and stories could now be created and stored in the backlog. The biggest change was obviously the scribble widget which gave the user much more flexibility in the way they could interact with the artefacts.

1.3.6 Design Shortcomings

The biggest problem with the design of both versions is the rigid imposition of a hierarchical relationship of the artefacts. The system imposes a top down approach to the creation of these artefacts. In other words, a Project is first created, then a release, then a sprint etc. This is very different from the way it would be done on a real wall; where stories would become part of a sprint or release by placing them in the appropriate location on the wall. See Future Work for more detail on this point.

1.3.7 How does CMAP comply with Scotts guidelines for group collaboration around digital tabletops

Support Interpersonal

Interactions Collaboration is a communicative process driven by human interactions and this is supported by CMAP in that the software makes no deliberate attempts to interfere with this driving force. The success of this claim would have to be verified by conducting user studies which should reveal if this is truly the case. In future implementations an attempt to capture this interaction will be made by integrating the capability to record the planning session either as a whole or any individual part of the planning meeting including voice recording controls which can be enabled or disabled on key artefacts like individual storycards as well as in the shared workspace (the cardwall).

Support for fluid Transitions between Activities

The original CMAP design had no handwriting capabilities and therefore had no real transitions between different types of user input; although it may have been fluid it was too rigid for a truly collaborative application. The revised version allows the user to seamlessly transition between data entry via keyboard or handwriting. CMAP has built-in support for multiple user input devices via the PyMT framework. This allows the application to support multiple users interacting with the system concurrently. For example multiple users could all be creating and working on different stories at the same time. Multiple users could technically also edit a story at the same time but, just as with the paper storycard, the size of these widgets may limit the usefulness of this process. The PyMT framework also has built-in support for other input devices such as stylus or even the Wii-remote.

Support for Transitions between Personal and Group Work

Transitions between personal and group work will be supported in future versions of this application. In particular the future work already being considered and planned for to support these types of transitions are:

- Create a personal space work widget which will be a modified coverflow. The traditional coverflow widget is static and displays its child artefacts as pages in a book which you can flip through page by page. The modified coverflow widget will allow the user to transform the widget (scale, rotate and move). Each user will have their own coverflow to use as a personal work area in which they can create and manage their artefacts in their own personal space. The widget must be capable of enabling or disabling a shared view of all or part of their widget. By default the widget will be private with the user being able to share any part as necessary. The widget will support the seamless transition from the current artefact to any adjacent artefact plus the ability to quickly jump to an artefact that is physically separated by many artefacts. The primary method of sharing artefacts contained in a users personal coverflow will be to drag the artefact to the shared workspace which essentially creates a copy of the artefact in the public domain where it can be manipulated by all users. The coverflow will have a smaller scaled view to support easier interaction and placement

of the contained artefacts into the shared workspace as well as the ability to maximize any one artefact in the users personal workspace.
- The primary goal of this shared workspace will be to simulate a story wall where stories are placed and sorted into releases, sprints, etc. Most importantly the shared workspace is intended to enhance the collaborative experience and facilitate discussion which is a critical component, necessary to ensure the success of any Agile based design meeting.

Support Transitions between Tabletop Collaboration and External Work

Future versions of this software will support a variety of external work inputs and outputs:

- Allow handheld devices to interact and exchange data with the application (from cell phones to laptops).
- Allow the application to seamlessly allow data import and export to and from external applications. This should include other Agile development tools, bug trackers, schedulers and task management software. Direct integration into the developers IDEs, starting with Eclipse, using Mylyn as a task management system is an objective.
- The largest benefit of this software may in fact lie in its ability to interoperate between devices and an ideal setup might in fact be a digital table top for group collaboration combined with a wall mounted surface to truly give the look and feel of the traditional paper based Agile design process.

Support the use of Physical Objects

Currently the PyMT Framework supports the use of stylus pens, Wii-remotes and as stated in the previous point future versions will support the use of other digital artefacts. Non-work related physical objects are to some degree supported by the hardware in that many tabletops for example provide a border where items like drinks or notepads can be placed without interfering with the application.

Provide Shared Access to Physical and Digital Objects

Most of the widgets in the application are rotatable which gives users the opportunity to use them in any appropriate orientation. When artefacts are in the shared workspace gestures like pointing should be easily understood and thus should contribute to the collaborative process. Careful consideration and further study may reveal to what extend the application needs to support other types of physical objects.

Considerations for the Appropriate Arrangements of Users

The physical size of the table and for future versions, the wall should dictate the number of users that could comfortably interact with the system. The future ability to use handheld devices should allow for more simultaneous user interactions and therefore allow for more participants especially when interacting around the wall. The only parts of the program that that cannot be rotated are the menus. Using these menus may be sensitive to the orientation of the users. These menus are however, optional since gestures are available to replace the menu items functionality and the gestures have no orientation. The rest of the widget as stated previously can be rotated to suit the needs of the users. For future versions where the personal workspace is implemented there will need to be careful consideration as to how the system reacts when people physically change their positions around the workspace. One simple solution might be similar to what might happen in a real meeting. For example, in a traditional pen and paper style meeting the task at hand requires that you change your position for whatever reason and you forget to bring all your personal stuff it may be reasonable to simply ask someone to pass your things to you; since all the artefacts are draggable this could easily be accomplished by the CMAP.

Support Simultaneous User Actions

The support for simultaneous user actions is limited only by the hardware since the PyMT framework gives the application support for any number simultaneous user actions. This support is of utmost importance to the success of any software designed to support co-located group collaboration.

1.3.8 Distributed Agile Teams

Although for the most part distributed Agile team support is out of scope for the project, it is however supported in a limited fashion. This support includes the ability to work using the application in different locations and when data is saved it will be available to all applications that have registers with the Google app server. Registration with the Google App server is dependent on this feature being enabled via the configuration file or the corresponding command line argument. While not real time collaboration is does gives distributed team the ability to share their artefacts.

1.4 Implementation

This seqction provides a brief description of the major elements implemented in CMAP. It is a companion to the sequence diagrams that describe how CMAP works.

- Storyboot: The entry point for the application, it manages orderly start-up and shutdown.
- AgiConfig: Creates the configuration from a set of configuration files and command line arguments and makes this information available to other components.
- Console Logger: Provides configurable logging services to the application
- Async: Provides non-blocking access to the Collaboration Server and the Git Manager.
- PyMT: The multitouch framework provides windowing services, manages I/O with multitouch devices, maps them into a uniform interface, and integrates keyboards (physical and virtual) and mice onto the standard interface. StoryApp: The main application controller. It manages the main PyMT window and serves as a container for all application containers such as coverflows, menus, lists etc. MTWindow: The main PyMT window. All drawing occurs here or a child of this widget.
- MTCoverflow: The primary container, used to hold application artefacts in CMAP.
- ArtifactController: The base class for all CMAP artefacts. Provides the controller element of the Model/View/Controller (MVC) design pattern. Functions as the public interface to artefact data and provides the control logic for all displayable artefacts.
- BaseModel: The base class for the model element of the MVC design pattern. Encapsulates the data for an artefact and interfaces with Document Object Model.
- GitManager: Interface to the GIT repository, it abstracts away all the complexities of using Git and allows Git to be accessed in an non-blocking manner.
- ServerManager: Provides an HTTP server that listens for notification messages about file changes from GitHub via the Collaboration Server hosted on Google.
- ToCollaboration: Provides subscribe and unsubscribe services to the Collaboration Server.
- Collaboration Server: Implemented as a Google App Engine application hosted on Google. Provides publish/subscribe and store and forward services to CMAP. Currently allows CMAP users in a team to receive file change notification messages from GitHub in real time or at subscribe time.

Figure 1.12 is shows the relationships between the StoryApp controller and the local GIT repository, the GitHub server and the Google App Server. Beside the artefacts previously discussed in the design section the next most important widget used for interaction with the system are the containers that hold like artefacts. For this purpose a coverflow was chosen because of its ability to flip through its pages, where the pages of our coverflow widgets are buttons representing existing artefacts and a default button to create a

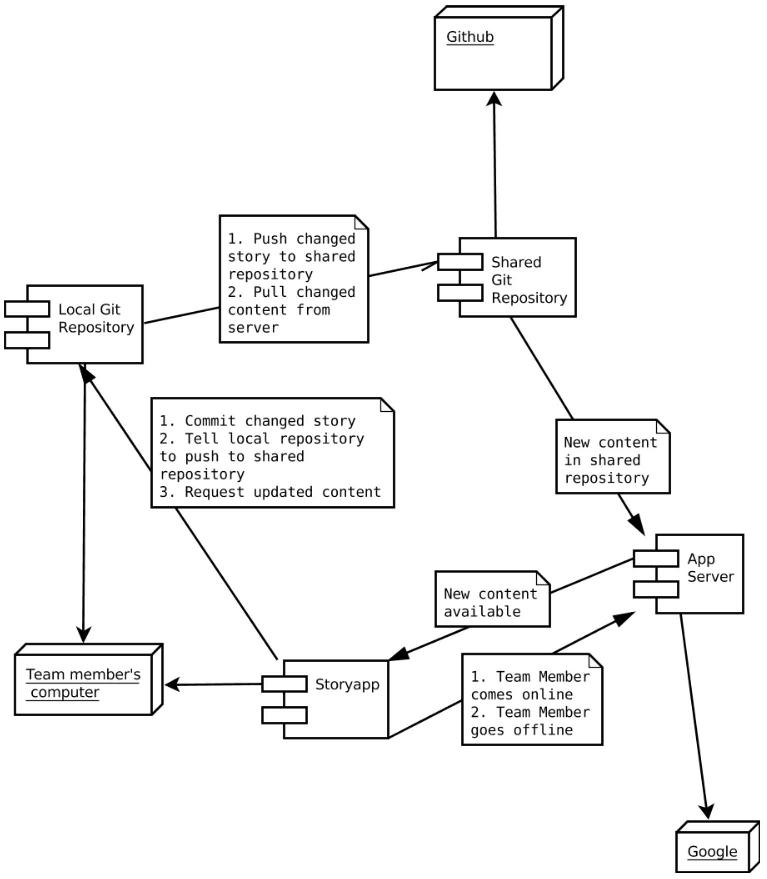

Github
1. Push changed story to shared repository
2. Pull changed content from server
Shared Git Repository
Local Git Repository
1. Commit changed story
2. Tell local repository to push to shared repository
3. Request updated content
New content in shared repository
App Server
New content available
Team member's computer
Storyapp
1. Team Member comes online
2. Team Member goes offline
Google

Fig. 1.12. Relationships between Git, GitHub and Google App Server

new artefact. The coverflow widget was customized so that the users could drag them around as well as do the standard coverflow stuff like flip through the pages and interact with the current page by selecting it. Figure 1.13 outlines the flow of populating a coverflow with artefacts. Figure 1.14 shows the sequence of subscribing and unsubscribing with the Google App Server as well the notifications sent from the GitHub server to the Google App Server and the subsequent messages sent to each subscriber. The final loop in this sequence diagram outlines what the StoryApp controller would do to update the artefacts. Unfortunately due to time constraints this was not implemented but there is a skeleton function in the StoryApp controller ready to do what the diagram depicts.

Figure 1.15 shows the data flow of a save operation including alternate paths it might take depending on the availability of the services like the local

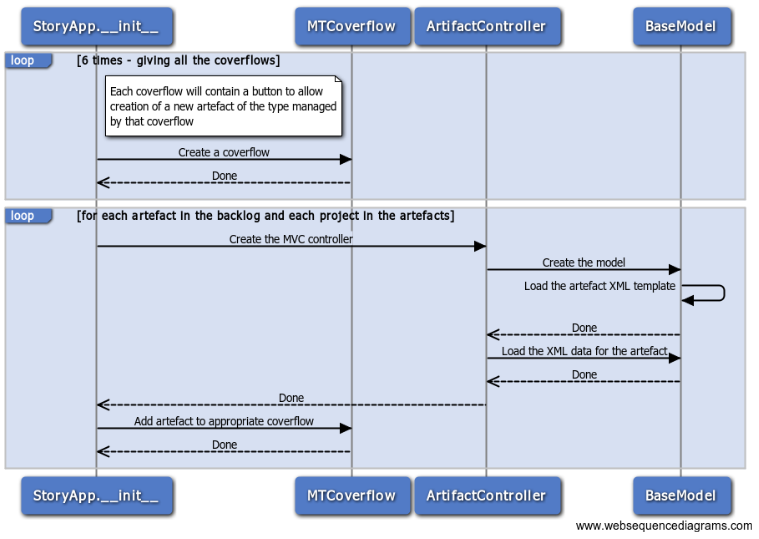

StoryApp.__init__
MTCoverflow
ArtifactController
BaseModel
loop
[6 times - giving all the coverflows]
Each coverflow will contain a button to allow creation of a new artefact of the type managed by that coverflow
Create a coverflow
Done
loop
[for each artefact in the backlog and each project in the artefacts]
Create the MVC controller
Create the model
Load the artefact XML template
Done
Load the XML data for the artefact
Done
Done
Add artefact to appropriate coverflow
Done
StoryApp.__init__
MTCoverflow
ArtifactController
BaseModel
www.websequencediagrams.com

Fig. 1.13. Container Creation

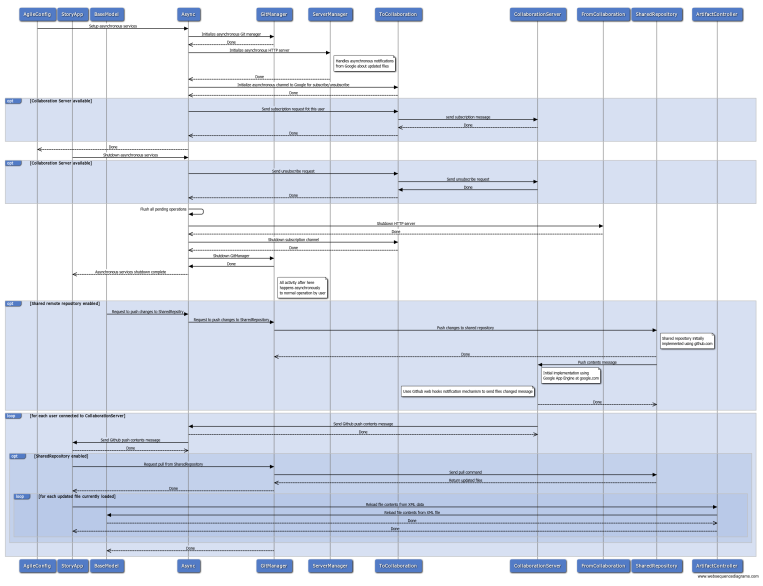

AgileConfig
StoryApp
BaseModel
Async
GitManager
ServerManager
ToCollaboration
CollaborationServer
FromCollaboration
SharedRepository
ArtifactController
Setup asynchronous services
Initialize asynchronous Git manager
Done
Initialize asynchronous HTTP server
Handles asynchronous notifications from Google about updated files
Done
Initialize asynchronous channel to Google for subscribe/unsubscribe
Done
opt
[Collaboration Server available]
Send subscription request fot this user
send subscription message
Done
Done
Done
Shutdown asynchronous services
opt
[Collaboration Server available]
Send unsubscribe request
Send unsubscribe request
Done
Done
Flush all pending operations
Shutdown HTTP server
Done
Shutdown subscription channel
Done
Shutdown GitManager
Done
Asynchronous services shutdown complete
All activity after here happens asynchronously to normal operation by user
opt
[Shared remote repository enabled]
Request to push changes to SharedRepsitry
Request to push changes to SharedRepository
Push changes to shared repository
Shared repository initially implemented using github.com
Done
Push contents message
Initial implementation using Google App Engine at google.com
Uses Github web hooks notification mechanism to send files changed message
Done
loop
[for each user connected to CollaborationServer]
Send Github push contents message
Done
Send Github push contents message
Done
opt
[SharedRepository enabled]
Request pull from SharedRepository
Send pull command
Return updated files
Done
loop
[for each updated file currently loaded]
Reload file contents from XML data
Reload file contents from XML file
Done
Done
Done
AgileConfig
StoryApp
BaseModel
Async
GitManager
ServerManager
ToCollaboration
CollaborationServer
FromCollaboration
SharedRepository
ArtifactController
www.websequencediagrams.com

Fig. 1.14. Collaboration Diagram, please see full sized diagram in the Diagrams and Pictures folder of the submission

repository. It also highlights the fact the save will automatically traverse the chain of children and parents to update all artefact affected by the change.

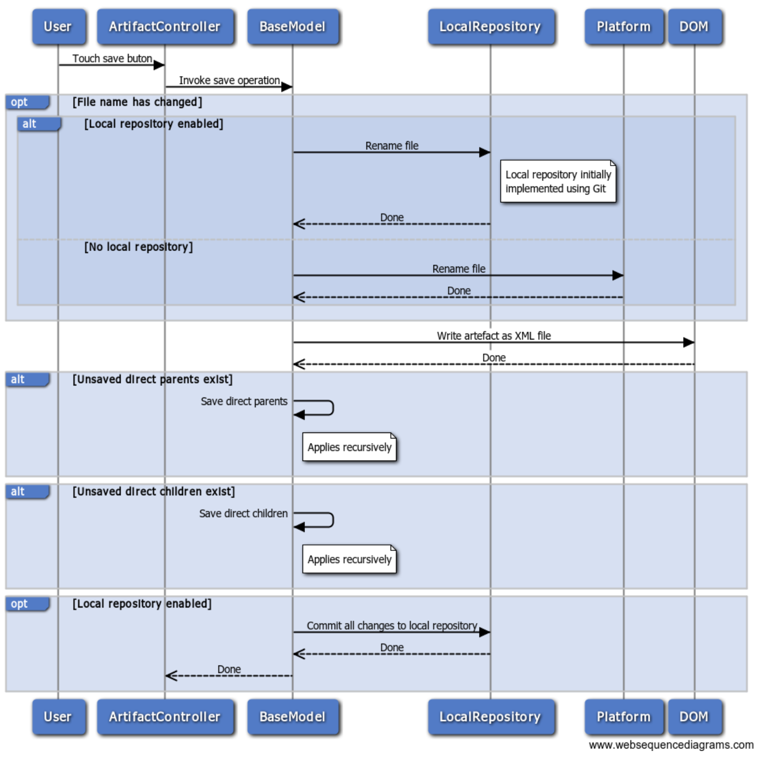

User
ArtifactController
BaseModel
LocalRepository
Platform
DOM
Touch save buton
Invoke save operation
opt [File name has changed]
alt [Local repository enabled]
Rename file
Local repository initially implemented using Git
Done
[No local repository]
Rename file
Done
Write artefact as XML file
Done
alt [Unsaved direct parents exist]
Save direct parents
Applies recursively
alt [Unsaved direct children exist]
Save direct children
Applies recursively
opt [Local repository enabled]
Commit all changes to local repository
Done
Done
www.websequencediagrams.com

Fig. 1.15. Saving Artefacts

Figure 16 shows the overall start-up/shutdown procedure and how it ties into the PyMT event loop.

1.4.1 The PyMT Multitouch Framework

We used PyMT framework because PyMT supports many input devices as well concurrent use of these devices. It has a rich library of widgets and support for gestures. Since it is an open source framework, any extensions or modifications needed could be easily applied. It clearly provided many highly useful features but, most importantly, because it is platform and operating system independent, it gave us the ability to run the app on a variety of hardware available to us in the lab. It needed to work on both our 23 and 24 inch Dell and Acer two touch monitors. This support was crucial so we could easily test as we coded. We also needed it run on out Smart Technologies Ltd.

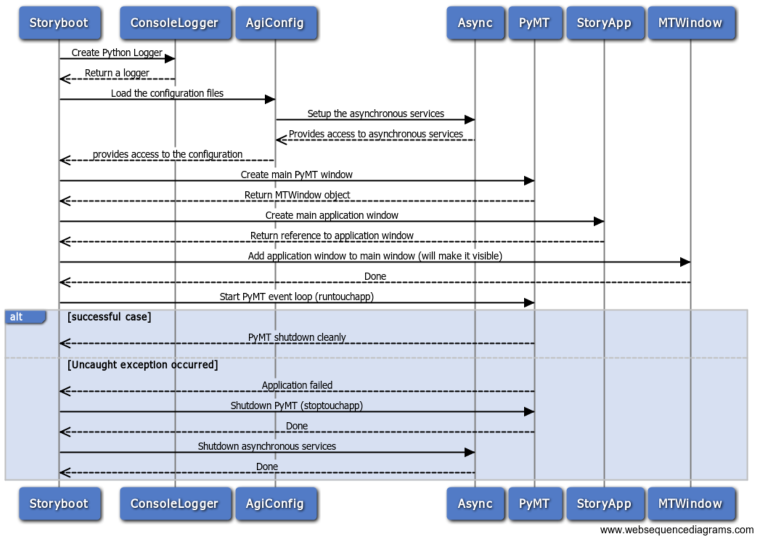

Storyboot
ConsoleLogger
AgiConfig
Async
PyMT
StoryApp
MTWindow
Create Python Logger
Return a logger
Load the configuration files
Setup the asynchronous services
Provides access to asynchronous services
provides access to the configuration
Create main PyMT window
Return MTWindow object
Create main application window
Return reference to application window
Add application window to main window (will make it visible)
Done
Start PyMT event loop (runtouchapp)
alt
[successful case]
PyMT shutdown cleanly
[Uncaught exception occurred]
Application failed
Shutdown PyMT (stoptouchapp)
Done
Shutdown asynchronous services
Done
www.websequencediagrams.com

Fig. 1.16. Start-up and Shutdown

FTIR table and on our custom built DI (Diffused Illumination [2]) table. The PyMT multitouch framework is still in beta but is well on its way to becoming an extremely useful tool for developing GUI interfaces for multitouch enabled hardware. It is in continuous active development and lead by a strong team leader and he has what appears to be a truly motivated team judging by the amount of activity in the projects repository. They have a good support system and are quick to answer questions and provide feedback. They are also quite willing to accept fixes and often will apply a patch to their master repository within a few hours of having request to pull. There are, however, challenges when using any beta product. This combined with my lack of experience developing user interfaces and GUI applications meant that there was far too much time debugging code thinking that it was a lack of understanding when in reality it was a bug in the framework. A developer more experienced in these areas would have been able to spot the difference. That being said it does have a rich library of useful widgets and is particularly good at providing the developer the means to truly showcase the multitouch capabilities of the system. There are probably two main weaknesses in this framework. The first is its support for text editing. While it does support text editing using real or virtual keyboards, it does not give you the functionality we all take for granted when dealing with text. This is actually an area that they are currently working on improving and have just recently, in the 0.52 development branch added the capability to select text, copy and paste it. This functionality is however not ready for production and does work as you would expect. The second problem and this admittedly may stem from my lack of experience in this type of programing but it seems that their layout and layout management is quite fragile and hard to use. This could be supported by the fact that they complete re-wrote their base layout class for the official release of 0.51. Overall the benefits of PyMT truly outweigh the disadvantages and there are absolutely no regrets as to the choice made to use this framework.

1.4.2 Status

CMAP accomplished many goals but still fell short in some areas for various reasons to be discussed later in this section. What was accomplished was the ability to use and collaborate around a multitouch enabled surface in an intuitive flexible manner.

Key Features Fully implemented in CMAP

- Full support for multiple concurrent inputs, limited only by the hardware. By using the PyMT framework we were able to provide this without any extra code.
- Support for multiple operating systems. Since the system is implemented in Python using the PyMT framework and both Python and PyMT are platform independent, this gives us the flexibility to run on many hardware devices. One example is that we were able to use CMAP on a tabletop designed by Smart Technologies without using the SMART SDK, instead we installed an open source package CCV (Community Core Vision) which was able to interface with sensor camera and output the touch event as defined by the TUIO protocol (Tangible User Interface Objects), which is supported by PyMT.
- Ability to create and manage many aspects of a project in an Agile manner. The system supports the hierarchical creation of different artefacts. Users can define projects and for each project they can create releases, for each release they can create sprints, for each sprint they can create stories and finally for each story they can create tasks.
- Flexibility is provided by the fact that users can choose to only deal with stories and are not forced to create any higher level artefacts like projects, releases and sprints. This mimics the real use of storycards at the start of a new project. Also the new interface gives the user complete flexibility in terms of how and where they enter data as well as the format, i.e. supports handwriting, printing or keyboard text entry.
- Ability to use gestures to create and interact with all artefacts. Although there are menus that give access to these same features, the menus are orientation sensitive while the gestures have no orientation. Also all the artefacts can be rotated to suit the needs of the users. The use of gestures is important in a system trying to eliminate orientation issues.

- Shared data is supported by the ability to commit to a local repository which in turn triggers a notification in the form of a pull request to other remote applications.
- Fully configurable to allow for on or offline line use or any combination of features. Full point to point collaboration commits changes to the local Git repository which pushes to the GitHub server which triggers a notification to be sent to the Google App Server to notify any other applications registered with the Google App Server that they should pull from the GitHub server.

Key Features Partially Implemented

- One of the most important features of the combined use of the traditional Storycard and cardwall is the use and organization of the stories on the cardwall. The system, as implemented, does allows users the flexibility to place the stories anywhere on the wall; however it lacks any ability to infer information from the placement of these stories. On a real world cardwall, the placement of cards and their relationship to other cards has meaning. The software needs the ability to infer the hierarchical structure of the artefacts simply based on their position and proximity to other artefacts. For this to work the system would have to be able to learn the way individual teams work so as to get these relationships correct.
- Real-time collaboration is partially implemented. When artefacts are saved to the local repository and the registration with the Google App Server has been enabled via the configuration file or command line option; the remote application(s) receive the request to pull. On successful completion of the pull there is a skeleton method that needs to be implemented that receives a list of files that need to be updated or added. Each file in the list is actually an XML representation of an artefact.
- In favour of performance the system currently on pushes to the GitHub server on exit. This means that concurrent remote system must restart in order to get updates from other systems. As a prototype this may be acceptable however, as a production system this mechanism would need to be revised. One possible piece of this solution would be to replace the use of GitHub and Google App servers with servers hosted in house. This would avoid excess network traffic on systems that were not designed for constant real-time updates.

Key Features Not Implemented

- Probably the most important feature not implemented is the flexibility to assign type to any artefact at any time. This is normally done by organising stories on the wall such that their placement is what signifies their relationship to other artefacts. It is extremely important to add this functionality to the system. Among other techniques dragging one artefact and

release it on another should create this relationship as well as sorting the artefacts on the wall (this would require that the system understands team specific layouts)
- Real-time editing of artefacts should be supported so that changes to artefacts are seen in real-time on remote systems. This includes the implementation of the hook in the StoryApp controller to update the artefacts and add new ones.
- Ability to identify which artefacts as current for each type.

1.4.3 Discussion

Lessons Learned

This project became an excellent lesson in the dangers of using too many new technologies all at once. The following were all new at the start of the project.

- The Python programming language
- The PyMT multitouch framework
- The principles of Agile methodologies
- GUI interface design and lack of GUI programming experience

In the end, the novelties associated with multitouch concepts became only a small portion of the new material that had to be digested to bring the project to a successful conclusion. The following discussion reflects on the impact of each technology on the success of the project. Because the use of the Python programming language was completely new it was a challenge to write good, clear reusable object oriented code. In particular, many practices traditionally considered important in the object oriented paradigm, like programming to interfaces and the use of composition over inheritance, were not as clear when using Python, instead of a more familiar language like Java. To start with Python intentionally has no direct support for what might be considered an interface; instead similar functionality is obtained through the use of abstract classes from which you inherit. Being force to inheritance in this way cause some confusion about when composition could be used over inheritance. But the dynamic nature of the language proved to be extremely useful as it allowed us to construct our artefacts from XML files including adding new attributes and this capability will be key to allowing the customization of the artefacts by the end users. The on-going learning process of the Agile methodology was in many respects an advantage for the development of a system that was trying to digitise and capture this behaviour. This meant that the upfront planning by the very nature of the knowledge base being drawn from was quite brief and only enough to get a general idea of the scope of the project. As more was understood about the process these revelations found themselves being applied throughout the course of development in subsequent iterations as well as during refactoring. Problems aside the lessons learned and the accumulated knowledge gained through the course of this project were invaluable and overall it was an incredible experience.

New Features For Future Versions

Many enhancements could be made to the system such as the following list of features that will be added to CMAP in subsequent versions

- The system must have an out of the box default behaviour that is intuitive and useful but it also needs to provide teams with the flexibility of adapting it to fit their particular process. Any method particular to a team should be configurable and supported by CMAP. This may include the teams use of color, symbols and notation and possibly other(s). All the things that they do that may convey additional meaning to users familiar with their system should be easily integrated into CMAP. We need the ability for the user to define all of this notation such that for example a scribble or text of a certain color might signify to the system that what is being written is a benefit or a As a, Want to or So that type of field. Likewise a certain symbol placed on the card could signify that this artefact is in progress, assigned etc. This flexibility is important but at the same time it should be up to the user to decide if they want to use this capability and not force it on the users. That being said, we must remember that the more the system understands the meaning of the content the more useful the digitization of the system becomes, especially in terms of overhead and duplication of work.
- The storycards themselves should have a way to be flipped such that the back of the card is usable and maybe a way to simultaneously view both back and front of any one artefact.
- Default templates to allow users to base their artefacts on a predefined structure should be supported. These templates should support customization similar to what was already discussed to support changes in the way team wants to use the artefacts. Different templates based on difference in common uses of the cards from different Agile methodologies would be a good start.
- If the users are using a template and or have defined any notation, then they should be able to switch between a structured and non-structured view of the cards (scribble vs. form type views).
- Each card needs the ability to record audio to try and capture the social interaction and intent of each story.
- Similarly, the whiteboard or wall should have its own flexible recording capabilities to capture the interaction during the sorting, prioritization and scheduling of the stories.
- Ideally a collaborative type group area like a digital table, combined with an information radiator like a digital wallboard as well as the ability to toss stories onto the wall or table from other portable devices or remote computers.
- The recognition of handwriting is critical in order to interface and exchange data with other tools.

- Size matters; the cards must not be re-sizable other then maybe the ability to see both sides simultaneously as previously mentioned. The physical size ensures a true communicative process and dialog between developers and customers (although we still want to be able rotate and obviously move them).

1.5 Conclusion

1.5.1 Innovations and Contributions

There are a few aspects of this project that could be considered as contributions to computer science. Most noticeably is a proof of concept that large multitouch surfaces can be leveraged to support dynamic, team collaboration in collocated and distributed environments. Multitouch enabled hardware and software provides a new interface that is very well suited for group collaboration and gives developers of collaborative design software the tools needed to address many of points outline in Scotts paper on the guidelines for group collaboration around a digital tabletop. CMAP provides an open source platform independent framework that can be used to develop and test further concepts in co-located and distributed collaboration. These concepts can be tested on any number of multitouch enabled hardware devices. This flexibility is important for future study aimed at establishing guidelines for the best combination of size, resolution, number of concurrent touch inputs and shape of future multitouch devices for the express purpose of group collaboration in a design setting. The novel use of the distributed code repository may also be considered important; it demonstrated a clean reusable solution to the difficult problem of sharing versioned data and the possibility to remotely collaborate using the shared distributed repository. A number of important lessons were learned during the development and use of the two versions of CMAP. The most important is to consider the relations between the physical actions being performed and the cognitive processes that facilitate the activity. A user that picks up a storycard from a backlog pile and places it on a pile for a particular sprint is hopefully thinking "I am scheduling this story into that sprint" rather than "this is how I move this card to that pile". It is important that the physical gestures not interfere with the flow of thought. Far too often in computer based systems this is not the case and the necessity to perform several computer interactions to execute a seemingly simple operation interrupts the train of thought. A rigorous re-examination of how we interact physically with the computer is needed to eliminate distracting computer operations.

1.5.2 Future Work

CMAP can be used as a starting point for future study. Of particular interest would be to conduct user studies to observe its usefulness in a real world project development process in both collocated and distributed teams. Further development of the application could then be explored, implementing the results of those findings combined with the continued development of the system with the addition of the features outlined in the Implementation section. Further exploration of improvements needed for the real-time collaboration in distributed teams is critical for the system to be successfully integrated as a widespread tool in industry. Furthermore, there needs to be more exploration with respect to the potential use of 3D graphics in conjunction with multi-touch. PyMT is fully OpenGL compatible and will permit the exploration of rich alternatives to existing means of interaction.

1.5.3 Closing Remarks

As software engineers and developers we are constantly presented with the opportunity of creating software intended to replace activities traditionally performed using pen and paper. Although our ability to design and implement these systems is improving, we still tend to fall short more often than not. Research has shown that the versatility of paper contributes to its persistent use in many work environments, even along-side computers meant to replace it. [13] Key factors that contribute to this failure have been outlined in this paper. Most important is the flexibility of the system, and the human factors that make the activity a success. The digitization of tasks must consider the communication, mediation and human interactions that are inherent in the tasks, and must be designed in a way to encourage or enhance these interactions (or at the very least, not interfere with them).

References

1. *Agile Software Development with Scrum (Series in Agile Software Development).* Prentice Hall, 2001.
2. Diffused illumination, from the nui group. `http://wiki.nuigroup.com/Diffused_Illumination_(DI)`, December 2010.
3. Surfnet, fall 2010 workshop. `http://www.nsercsurfnet.ca/pmwiki.php?n=SurfNet.Workshop`, September 2010.
4. J Aiello. *Human Spatial Behavior. In D. S. (eds.)*, volume Handbook of Environmental Psychology, vol. 1, pages 389–505. John Wiley Sons, Toronto, ON, 1987.
5. K. Beck. *Extreme Programming Explained.* Addison-Wesley Professional, 1999.
6. K. Beck and C. Andres. *Extreme Programming Explained: Embrace Change, Second Edition.* Addison-Wesley Professional, 2004.
7. A. Cockburn. *Agile Software Development: The Cooperative Game, Second Edition.* Addison-Wesley Professional, 2006.
8. M. Cohn. *User Stories Applied: For Agile Software Development.* Addison-Wesley Professional, 2004.

9. M. Cohn. User stories for agile requirements. `http://www.mountaingoatsoftware.com/presentations/130-user-stories-for-agile-requirements`, December 2010. Mountain Goat Software.
10. R. Davies. The power of stories. In *The Power of Stories*. XP 2001, Sardinia, Springer Verlag, 2001.
11. R. Jeffries. Essential xp: Card, conversation, and confirmation. *XP Magazine*, August 2001.
12. R. Morgan. Distributed agileplanner: A card-based planning environment for agile teams. Master's thesis, University of Calgary, Department of Computer Science, 2008.
13. Proceedings of ECSCW'03, European Conference Computer-Supported Cooperative Work. *System Guidelines for Co-located,Collaborative Work on a Tabletop Display*, Helsinki, Finland, September 2003.
14. H. Sharp, H. Robinson, and M. Petre. The role of physical artefacts in agile software development. *The Interdisciplinary Journal of Human-Computer Interaction*, 21, 2009. issues 1-2.
15. X. Wang. Using digital tabletops to support agile project planning. a. Master's thesis, University of Calgary, Department of Computer Science, Calgary, Alberta, Canada, September 2009.
16. Ghanam Y. Wang X. Maurer F. Weber, S. Apdt: An agile planning tool for digital tabeltops. In *APDT: An Agile Planning Tool for Digital Tabeltops*. Proceedings 9th International Conference on Agile Processes and eXtreme Programming in Software Engineering (XP2008), 2008.