

HOCHSCHULE MANNHEIM

Diplomarbeit

Thema: Agile Planning on Digital Tabletop Devices

Sebastian Weber

Matrikelnr.: 0530104



hochschule mannheim

Praktischer Teil angefertigt in der

e-Business Engineering Group

University of Calgary, Kanada



UNIVERSITY OF
CALGARY

Betreuer:

Prof. Dr. A. Schmücker-Schend, Hochschule Mannheim

Prof. Dr. F. Maurer, University of Calgary

Mannheim, den 21.02.2008

Statutory Declaration

Ich versichere, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit hat in dieser oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegen.

Sebastian Weber

Mannheim, den 21.02.2008

Abstract

Paper index-cards are a commonly used tool in the Agile Planning Game. One of their greatest advantages is the ease of their use. Structured information can be noted side by side with spontaneous ideas and detailed descriptions. The cards can be spread out on a conference table to serve as a focus for collaborative discussions and to share relevant information among stakeholders.

Prioritization and grouping of related stories is simply done by organizing the cards on the tabletop.

Existing tools that support Agile Planning mostly ignore the metaphor of card-based planning and force the users to adapt their behaviour to the application instead of integrating themselves into the existing workflow.

AgilePlanner for Digital Tabletops (APDT), the tool developed for this thesis, is designed to satisfy the requirements of card-based Agile Planning. It combines tabletop technology, handwriting-, gesture- and speech-recognition to provide agile practitioners with an environment that is very similar to their well-known practice while at the same time enhancing the process by offering increased flexibility in the ways that planning tasks can be accomplished.

German Abstract

Karteikarten sind ein im agilen Planungsprozess weit verbreitetes Werkzeug. Ihr großer Vorteil liegt in ihrer einfachen Nutzbarkeit. Strukturierte Informationen können Seite an Seite mit spontanen Notizen und detaillierten Erklärungen festgehalten werden. Auf einem Konferenztisch ausgelegt, können die Karten als Richtschnur für gemeinschaftliche Diskussionen dienen und dazu genutzt werden, allen Anwesenden relevante Informationen zugänglich zu machen.

Durch einfaches Anordnen der Karten auf dem Tisch können diese priorisiert und voneinander abhängige User-Stories gruppiert werden.

Die meisten verfügbaren Werkzeuge zur Unterstützung agilen Planens ignorieren die Karteikarten-Metapher. Somit ist der Benutzer gezwungen, sich an die Anwendung anzupassen, anstatt mit einer in den vorhandenen Arbeitsablauf integrierten Anwendung arbeiten zu können.

Das im Rahmen dieser Diplomarbeit entstandene Programm, AgilePlanner for Digital Tabletops (APDT), wurde speziell nach den Anforderungen kartenbasierten agilen Planens entwickelt. Es verbindet Tabletop-Technologie, Handschrift-, Gesten- und Spracherkennung, um agilen Teams eine Umgebung zu bieten, die ihre vertraute Arbeitsweise unterstützt und zugleich den vorhandenen Prozess durch erhöhte Flexibilität bei der Durchführung anstehender Aufgaben verbessert.

Acknowledgements

The path to this thesis was long and winding and I want to thank the people who helped me to finally get here:

My supervisors, Prof. Dr. Astrid Schmücker-Schend, for helping me with my application in Calgary and making me think ahead, to see where I'm going, and Prof. Dr. Frank Maurer, for giving me the chance to learn and work with a lot of fascinating technologies.

Robert Morgan, David Fox and Patrick Wilson, for a lot of morale support, for their help with everything from the language barrier to my work and for the countless hours of fun I had with them.

Johannes Fischer, for keeping me going in times of motivation lows and for a lot of fresh ideas that helped me view problems from another perspective.

My parents, who were always there when I needed them and who made a lot of things possible for me.

Dedication

Für meine Eltern, die mir durch alle Hochs und Tiefs geholfen haben.

Table of Contents

Statutory Declaration	I
Abstract	II
German Abstract	III
Acknowledgements	IV
Dedication	V
Table of Contents	VI
List of Figures	VIII
List of Tables	IX
List of Abbreviations	IX
1 Introduction	1
1.1 Agile Methods	1
1.2 Card-based Planning	1
1.3 Tabletop Interaction	2
1.4 Goal of Thesis Work	3
1.5 Structure of this Thesis	4
2 Related Work	5
2.1 Agile Development	5
2.2 Single Display Groupware	8
2.3 Tabletop Interaction	9
2.4 Existing Agile Planning Tools	10
2.4.1 Form-based Planning Systems	11
2.4.2 Combined Wiki- and Form-based Planning Systems	12
2.4.3 Board-based Planning Systems	12
2.4.4 Limitations of Existing Tools.....	13
2.4.5 Agile Planning Tool Requirements	14
3 Digital Tabletop Implementation Environment	16
3.1 Hardware	16
3.2 Software	17
3.2.1 Evaluation of Existing Frameworks	18
3.2.2 Development Environment.....	21
4 Agile Planner for Digital Tabletops (APDT)	23
4.1 Application Structure	23
4.1.1 Planning Artefacts.....	24
4.1.2 Visualization of Planning Data.....	25
4.1.3 Persistence and Communication.....	25

4.1.4 Application Core.....	26
4.1.5 Utilities	27
4.1.6 Unit Testing.....	28
4.1.7 Resources.....	28
4.2 User Interface Overview	28
4.2.1 Gestures and Handwriting Recognition	29
4.2.2 Speech Recognition	30
5 APDT in Detail	32
5.1 Realistic Interaction Behaviour	32
5.2 Multi-modal Input.....	38
5.2.1 Gesture Recognition	38
5.2.2 Handwriting Recognition	39
5.2.3 Speech Recognition.....	41
5.3 Planning Artefacts	42
5.3.1 Story cards.....	42
5.3.2 Iterations.....	45
5.3.3 Floating Toolbox	46
5.4 Command History.....	48
5.5 Persistence Layer	49
5.5.1 Synchronous Persistence	50
5.5.2 Asynchronous Persistence.....	50
5.5.2.1 Abstract Base Class	50
5.5.2.2 Web Service Based Persistence.....	50
5.5.2.3 Adaption from Synchronous to Asynchronous Persistence	53
5.5.2.4 Two-way Communication with DAP	54
6 Conclusion and Future Work	58
6.1 Problems	58
6.2 Thesis Contributions.....	59
6.3 Future Work	61
References.....	VIII

List of Figures

Figure 1: Rally planning interface; Source [34].....	11
Figure 2: MASE planning interface; Source [38].....	12
Figure 3: CardMeeting planning interface; Source [6].....	13
Figure 4: SMART Tech DViT; Source [46].....	16
Figure 5: SMART DViT table at the e-Business Engineering Group lab	17
Figure 6: Overview of Large Display Framework; Source [4].....	19
Figure 7: Package structure of APDT	23
Figure 8: UML diagram of AgilePlannerDT.Data	24
Figure 9: UML-diagram of AgilePlannerDT.Display	25
Figure 10: UML-diagram AgilePlannerDT.Persister.....	26
Figure 11: UML-diagram AgilePlannerDT.Application	27
Figure 12: UML-diagram AgilePlannerDT.Utils.....	27
Figure 13: Creation of a new Iteration using a gesture	29
Figure 14: Illustration of RNT motion; Source [19].....	33
Figure 15: Three required points determine the RNT algorithm: C - <u>C</u> enter of the object; O - the <u>O</u> riginal mouse position and T - the new <u>T</u> ouch position (or <u>T</u> arget); Source [19].....	35
Figure 16: The RNT algorithm uses a physically-based model to calculate the rotation at each frame. The object acts as if a force of friction is being applied opposite the direction of movement; Source [13].....	35
Figure 17: Handwriting recognition process.....	40
Figure 18: New story card	42
Figure 19: Story card lifecycle	43
Figure 20: An extract of available colours.....	43
Figure 21: Story card in handwriting mode	44
Figure 22: Story card with highlighted translate-only-area.....	45
Figure 23: Empty Iteration	46
Figure 24: Floating Toolbox.....	47
Figure 25: Join meeting dialog	48
Figure 26: Colours assigned to specific tasks.....	48
Figure 27: UML-diagram AgilePlannerDT.Application.Commands	49
Figure 28: UML-diagram of AsyncWebservicePersister.....	51
Figure 29: UML-diagram of AsyncWrapperPersister	53
Figure 30: Overview diagram XML serialization in .NET	55
Figure 31: Overview diagram XML-serialization with XStream	56
Figure 32: Overview diagram of communication between APDT and DAP	57

List of Tables

Table 1: Assessment for existing tool support for agile planning; Source [21]	15
Table 2: Tool comparison; Source [32].....	15
Table 3: Overview of voice commands and gestures in APDT; Source [11]	31
Table 4: Criteria assessment of APDT according to [21]	60
Table 5: Criteria assessment of APDT according to [38]	60

List of Abbreviations

APDT	Agile Planner for Digital Tabletops
API	Application Programming Interface
CI	Continuous Integration
CLR	Common Language Runtime
DAP	Distributed Agile Planner
DViT	Digital Vision Technology
EBE	e-Business Engineering Group, University of Calgary
GC	Garbage Collection
GEF	Graphical Editing Framework
GUI	Graphical User Interface
HCI	Human Computer Interaction
iLab	Interactions Lab
LDF	Large Display Framework
RNT	Rotate 'N Translate
SDG	Single Display Groupware
UI	User Interface
UofC	University of Calgary
VC++	Visual C++
VS	Microsoft Visual Studio
WPF	Windows Presentation Foundation

1 Introduction

1.1 Agile Methods

Over the past few years the public interest in agile methodologies has grown. Instead of setting up complicated processes that deliver products of often limited value to the customer, Agile Methods focus on involving the customer from the very beginning and handing over working versions of the product as soon as possible, to allow for instantaneous feedback.

Existing flaws can be detected and eliminated as soon as the customer notices them and the short development cycles allow for a greater flexibility than most other processes when it comes to reacting to changes in the customers' requirements.

As a guideline for agile practitioners and an introduction for starters, the Agile Alliance [1] defined a precedence hierarchy of principles that are regarded as the pillars of Agile Software Development:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

While the members of the Alliance do see value in the items on the right side, the ones on the left are valued higher and are seen as more beneficial to software development [2].

In agile teams, communication and direct interaction between people are used to compensate for a lack of formalized written documentation as requested by many process models and, ideally, on-site customers answering questions regarding development decisions help to keep the development efforts on track.

1.2 Card-based Planning

Nonetheless, there is a need for more long-term planning even for agile development teams. Following the paradigms of customer involvement and intense face-to-face communication, planning meetings in an agile environment bring both,

customers and developers, together to collect requirements for future development. Feature requests from customers, so called user stories, are written on index or story cards.

The developers then estimate the efforts required to accomplish the tasks of these user stories. In a third step, customers and developers negotiate a subset of these stories that will be worked on in the upcoming iteration, based on the estimates and the available effort for that iteration. The duration of an iteration varies between a few weeks and a few months, with a preference to the shorter time scale.

The result of such a planning meeting is not only the content of the compiled set of cards. Their layout on the table and the way they are grouped and ordered relative to each other often indicates relations between the tasks that are noted on the cards or the priority of one card over another. This implicit information is also of great importance and needs to be preserved.

After the meeting, the paper cards need to be transcribed to make them available to both, customers and developers, as an agreed on project plan. Transcribing cards is time-consuming and error-prone. Not only does it result in duplicate data that needs to be synchronized on updates but most developers also regard such tasks as very disturbing for their actual work [21]. While these problems are still somewhat solvable, it is even more demanding to save the information stored in the position and grouping of those cards. Taking pictures of the table is neither a very convenient nor an effective method to save all information necessary for a complete understanding of the outcome of the meeting. So what other ways are there, to accomplish this task?

1.3 Tabletop Interaction

Quite a few applications try to support the planning process, from the creation of information to its storage as a project plan, but available tools are mostly designed as desktop applications to run on personal computers. While common computers are sufficient for single users or very small groups, they are definitely not fit to serve as an environment for collaborative planning for larger teams for two reasons.

First, the display size is very limited thus making it difficult for all team members to recognize what is going on in the meeting.

Second, interaction with the application is usually controlled by a single person, who is in control of the mouse and the keyboard.

These limitations interfere with the natural workflow of agile planning where, ideally, every member of the team is involved in the process.

The development of a new type of hardware platform, designed specifically to support the collaborative work of multiple people, makes it possible to overcome these limitations.

In simple terms, Digital Tabletop Devices are large, horizontal displays with some kind of touch sensitive surface. While the actual technologies underlying this concept show a broad variety (from back projection displays to LCDs, from video analysis to touch sensitive foils etc.) the general idea is to provide a shared workspace that can be easily accessed by multiple users.

The use of this kind of hardware lifts the restrictions of single-user desktop applications but at the same time it creates new challenges.

Users can sit or stand at any position around the table, making directions like 'up' or 'down', which determine standard display orientations, irrelevant. Not only does this problem lead to the necessity of a convenient way to orientate and reposition content, to make it readable by every participant of a meeting, but it also creates the need for a new kind of user interface to control the application and its functionality since standard menus or toolbars are not useable in this kind of environment.

The size of the table and the resulting distance between team members are a second major issue. If content is to be edited (or viewed) by people at different ends of the table, some way of moving this content from one person to another without extra effort must be provided.

In addition, studies identified the inclusion of handwriting recognition as a key feature for this kind of planning tool as it provides a very natural way to create and share information [21].

1.4 Goal of Thesis Work

The goal of this thesis is not only to imitate the process of paper card based planning but to augment this powerful and easy to use metaphor by integrating the advantages that an electronic environment provides.

AgilePlanner for Digital Tabletops (APDT), the application developed for this thesis, is based on the work of Morgan [38] and Liu [21]. It is designed to support co-located collaborative Agile Planning on Digital Tabletop Devices. This thesis will also

give a brief outlook of how the application can be modified to allow for distributed planning.

1.5 Structure of this Thesis

The remainder of this work is presented as follows: The second chapter sets the work of this thesis in relation with existing work in this area. The third chapter summarizes the hard- and software that is used for the implementation and the upfront evaluation that was made.

Chapters 4 and 5 describe the architecture and design as well as the implementation in detail.

Finally chapter 6 concludes this thesis and gives an outlook on possible future work.

Source code, as well as names of interfaces and classes, is highlighted using the `Courier` font. Diagrams use UML for a better overview and are reduced to show the most important information.

2 Related Work

This thesis is based on findings from various research areas. The following chapter will give an overview of Agile Methods in general, the work that has been used as a foundation and of existing tools that support Agile Planning.

2.1 Agile Development

Agile Methods are a conglomerate of procedures and rules that try to overcome the limitations of 'traditional' or 'heavyweight' process models by being 'lightweight' and 'agile'. Agile Software Development is an iterative process that puts an emphasis on continuous customer involvement and the ability to react flexibly to changing requirements.

The Agile Alliance [1] defines the principles underlying the Agile Manifesto [3] as follows:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity - the art of maximizing the amount of work not done - is essential.

- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

Agile Methods prescribe a time-boxed, iterative process for the development. Being iterative means that the development lifecycle is subdivided into multiple iterations. System functionality is built incrementally until the system is completed as a whole. Iterative time-boxing describes the practice of setting a fixed deadline for an iteration and reducing the scope of this iteration if the deadline cannot be met [20].

Larman [20] describes twelve key motivations for the adoption of an iterative development process as follows:

- **Iterative development is lower risk; the waterfall is higher risk.** Results of the latest research show that an iterative lifecycle is associated with lower risk and better success, productivity, and defect rates. These results have led large and experienced software procurement organizations such as the USA Department of Defense (DoD) to promote the use of iterative development.
- **Early risk mitigation and discovery.** Risk-driven iterative development forces teams to tackle the hardest, riskiest problems first. Early development iterations exercise and reveal the true nature of the team and individual skills, the tools, and third-party software. The truth of the risks emerges: Perceived risks prove not to be, and unsuspected issues are forced into the open.
- **Accommodates and provokes early change; consistent with new product development.** Iterative development methods work with rather than fight against the high-change nature of software projects.
- **Manageable complexity.** The higher the complexity of a software project, the higher the failure rates are and the lower the productivity becomes. Iterative development decomposes complex projects or phases into small and bounded mini-projects of manageable complexity.
- **Confidence and satisfaction from early, repeated success.** Short iterations lead to a quick and repeating sense of completion, competency,

and closure. These psychological factors are important for individual satisfaction and building team confidence. They also build customer confidence in the team as they see early visible progress in the direction they care about.

- **Early partial product.** Apart from increasing the client's confidence, early visible progress with an integrated and tested partial product provides new business opportunities such as earlier demos or a sooner shipment of the product with fewer features.
- **Relevant progress tracking; better predictability.** Following the waterfall model can give a false sense of progress during the early, easier phases, but with low reliability in predicting later phase schedules, which vary widely.
- **Higher quality; less defects.** Iterative development methods require testing early, often, and realistically, in all possible dimensions thus leaving fewer errors to be discovered late in the development.
- **Final product better matches true client desires.** Through early evaluation and feedback from clients or prospective users, the product is more likely to satisfy their requirements.
- **Early and regular process improvement.** A common practice of iterative development is a per-iteration assessment — for example, a fifteen minute discussion to discover concrete actions to take in the next iteration to address a problem or improve the living process. Thus, iterative development leads to a continuous process improvement over time.
- **Communication and engagement required.** Failure research reveals that lack of client or end-user engagement, as well as lack of coordination and collaboration between members or sub-teams are major factors in software project failure. Developing in iterations forces intense communication between development team members. Clients' engagement is increased by their presence and feedback during per-iteration demos and planning meetings.
- **IKIWISI required.** A well-known human-nature related problem in software specifications is IKIWISI ('I'll Know It When I See It'). The complexity, many

degrees of freedom in solutions, and intangibility of software seem to demand concrete and cyclic feedback from people evaluating prototypes or partially built systems to clarify and refine their vision.

2.2 Single Display Groupware

The term Single Display Groupware (SDG) describes a model for applications “... which enable co-present users to collaborate via a shared computer with a single shared display and simultaneous use of multiple input devices” [48]. Traditional groupware applications are run on individual workstations with dedicated display and input devices for each user. These workstations communicate with each other over some kind of network connection. SDG applications are different in that they provide each user with a private input channel (using a personal input device) but all users share the same display for output.

The research of Stewart et al. [48] identified several potential advantages of Single Display Groupware:

- **Enabling collaboration that was previously inhibited by social barriers.** The use of SDG can help to overcome peoples’ reluctance to invade their co-workers’ personal space, a social habit that prevents full involvement in collaborative endeavours.
- **Enabling types of interaction that require multiple users.**
- **Enriching existing collaboration at a computer.** SDG can make turn-taking interaction obsolete.
- **Reducing or eliminating conflict when multiple users attempt to interact with a single application.** By providing separate input channels for each user, potential conflicts regarding shared interaction channels are pushed one step further away.
- **Encouraging peer-learning and peer-teaching.**
- **Strengthening communication skills.** As no strong-willed user can monopolize input devices, thus hindering other users to interact with the application, conflicts may have to be resolved by increased communication between users.

These findings also include a set of potential disadvantages that might origin from the use of SDG applications:

- **New conflicts and frustration may arise between users when they attempt simultaneous incompatible actions.** The possibility to work in parallel might be disadvantageous if users have separate conflicting agendas.
- **SDG applications squeeze much functionality into a very limited screen space, which may result in reduced functionality.**
- **Due to increased processing requirement, SDG applications may be slower than single user and traditional groupware systems.**
- **SDG implementation is highly environment dependent which raises portability issues.**
- **Completing tasks may take longer time since there is no strong willed person who is in charge of system interaction.** In the absence of a person who takes the lead it might take longer to agree on certain actions.
- **Users may actually collaborate less.** With the possibility of working on their tasks in parallel, users might just complete their own tasks without ever communicating with each other.

Tang [50] identified several important aspects with implications to the design of Groupware applications. His observations conclude that collaborators use hand gestures to uniquely communicate significant information, the process of creating and using drawings conveys much information not contained in the resulting drawings, the drawing space is an important resource for the group in mediating their collaboration, there is a fluent mix of activity in the drawing space and that the spatial orientation among the collaborators and the drawing space has a role in structuring their activity [50].

2.3 Tabletop Interaction

An analysis of the state-of-the-art in digital tabletop systems research conducted by Scott et al. [44] identified a crucial requirement for co-located tabletop collaboration.

“Through years of experience collaborating around tables, people have developed skills for interacting with each other as well as with objects on a table. When integrating computer technology into a table, designers must support these skills.”

According to the findings of the analysis they suggest that technology which wants to support collaborative interaction effectively must follow several design guidelines.

- Natural interpersonal interaction.
- Transitions between activities.
- Transitions between personal and group work.
- Transitions between tabletop collaboration and external work.
- The use of physical objects.
- Accessing shared physical and digital objects.
- Flexible user arrangements.
- Simultaneous user interactions.

In the physical world, objects can be manipulated in various ways and people expect certain reactions as results of these manipulations. Several studies found that the separation of moving and turning of objects is perceived as counter-productive and unnatural [10] [16]. As a consequence, Kruger et al. [18] developed the Rotate 'N Translate algorithm (RNT) which enables the integration of translation and rotation into a single motion. In an empirical evaluation RNT performed considerably better than technologies that require separate actions to reorient items on a table [19].

2.4 Existing Agile Planning Tools

With the emergence of Agile Methods, the number of available tools tailored to support story-based planning is growing. While the focus of the majority of these tools is to present the accumulated planning data in a table-like fashion, there are only a few applications which make use of the well-known index-card metaphor. Liu [21] identified three categories of planning support systems which are described as follows.

2.4.1 Form-based Planning Systems

The bulk of tools that support Agile Planning belong to the category of form-based systems. Typical representatives of this category use web interfaces that provide forms to store a predefined set of information, e.g. effort estimates or priority rankings as shown in Figure 1.

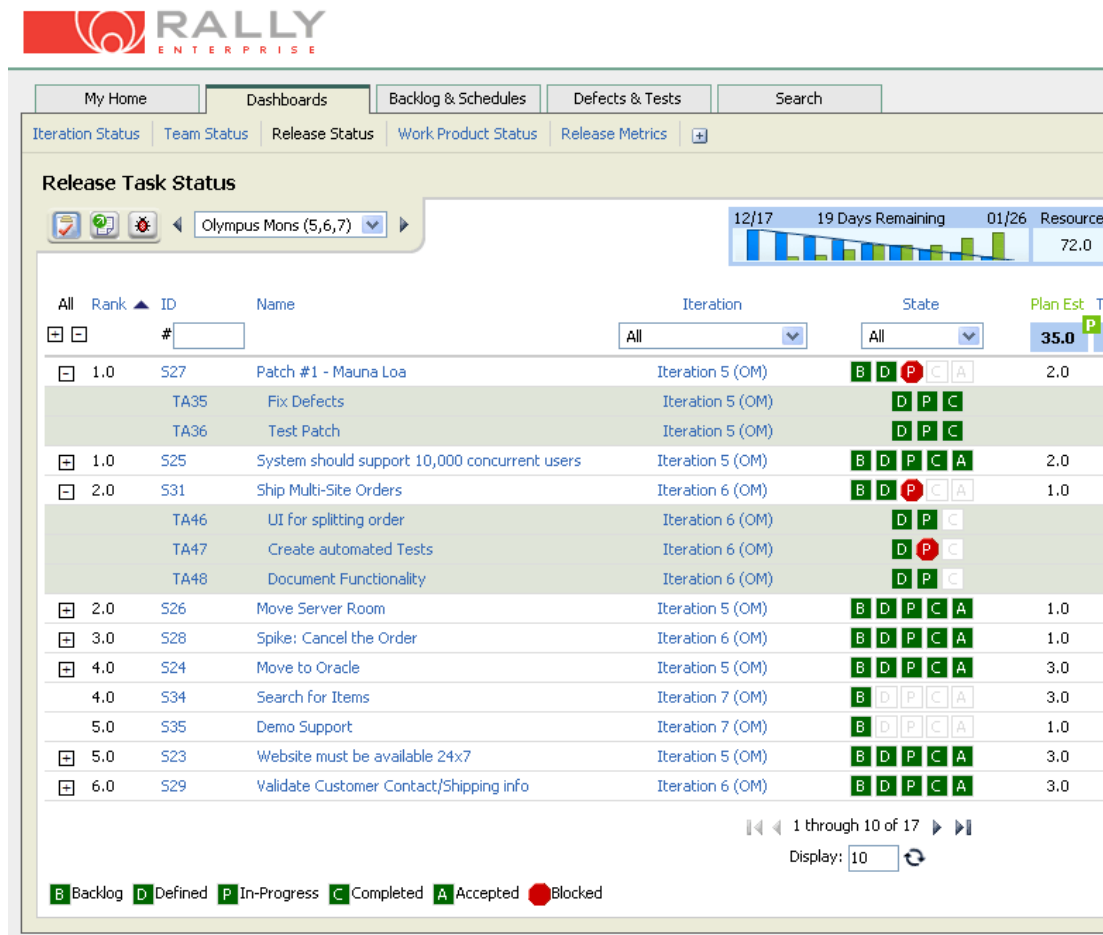


Figure 1: Rally planning interface; Source [34]

They provide basic functionality for creating and deleting as well as editing and prioritizing project planning artefacts and can derive supplementary information like total efforts for iterations or remaining work effort from existing data. In addition, they provide features for searching content and sorting of the presented information based on certain fields (e.g. priorities) [21].

Existing form-based tools comprise commercial products like Rally [43], VersionOne [54] and ScrumWorks [8] as well as open source products like XPlanner [59].

2.4.2 Combined Wiki- and Form-based Planning Systems

Tools like MASE [23] take the form-based approach one step further and combine it with another very popular way to share information between people. Users can attach Wiki-pages to stories that can be used to provide additional information related to a task.

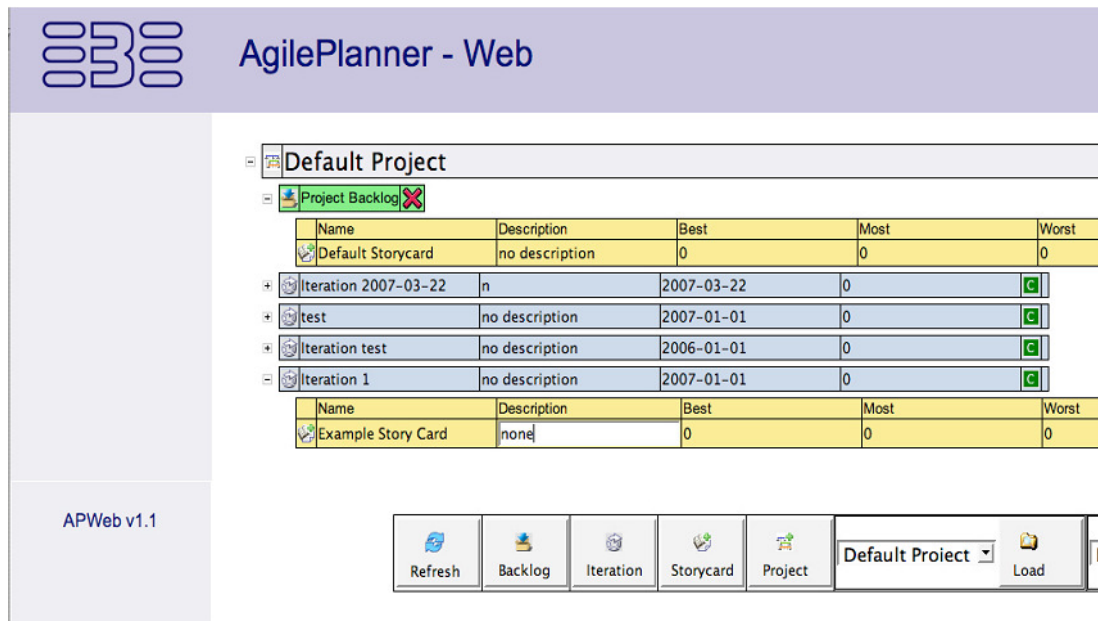


Figure 2: MASE planning interface; Source [38]

Projects like Wikipedia show how flexible wiki's are when it comes to present heterogeneous information. With the simple wiki-syntax even non-professional users can easily generate content.

2.4.3 Board-based Planning Systems

The third category, board-based planning systems, comprises tools like CardMeeting [6], AgilePlanner (AP) [21] and Distributed AgilePlanner (DAP) [38].

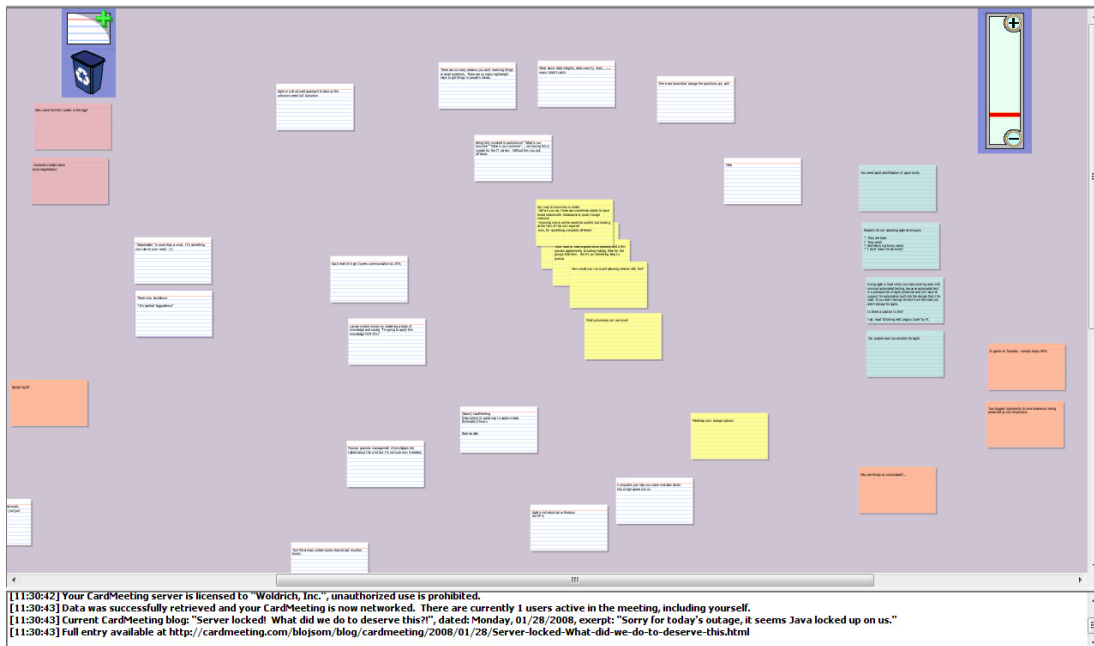


Figure 3: CardMeeting planning interface; Source [6]

A commonality of board-based systems is that they are all mimicking card-based planning to a certain extent. They provide ways to create, edit and delete index-cards and visually group those cards to indicate relationships.

2.4.4 Limitations of Existing Tools

In general, form-based systems are relatively good to obtain an overview of existing user-stories, but less suitable for collaborative planning work. They only provide limited ways for interaction and only capture a fixed set of information [21].

Hybrid Wiki- and Form-based tools have similar limitations as their pure form-based relatives, but are better suited for integrating information that does not fit into predefined categories. In addition, the Wiki-page attached to a user-story can be used to provide more detailed descriptions and explanations, links to external sources, diagrams, drawings or other helpful information. Still, as these applications are designed as regular desktop applications and rely on mice and keyboards for input, they are not apt to collaborative planning with multiple team members.

The category of board-based planning systems puts an emphasis on the index-card metaphor which is commonly used by Agile Teams. Aspects from two prominent

members of this category, namely AgilePlanner (AP) and Distributed AgilePlanner (DAP), are incorporated into this thesis.

While DAP is designed as a multi-client application, which focuses on the support of Agile Planning in a distributed setting and does not target Digital Tabletops in particular, AP was explicitly developed as a tool for Tabletop Devices.

Both tools have a Graphical User Interface (GUI) that displays planning information on index-cards. AP uses Storage Bins [45] to group cards that belong together, DAP has dedicated visual objects which represent Iterations and the project Backlog.

In AP, users can move story-cards unrestrictedly on the table surface but have to rely on supplemental hardware devices (Tablet PCs) to create or edit cards and make use of the handwriting features.

DAP in contrast is designed as a desktop application and does not provide integrated functionality to rotate stories but on the other hand allows direct creation and editing of the cards' content inside the application using mouse and keyboard.

Both applications do not make full use of available tabletop technology. AP forces meeting participants to use separate devices to create and edit user-stories, which interrupts the natural workflow of Agile Planning. In turns, DAP relies on standard Graphical Editing Framework (GEF) widgets, which cannot be resized properly to satisfy the requirements of a tabletop setting, nor can they be rotated. These significant usability deficiencies leave room for the development of a new tool, which incorporates the findings made by [21] and [38] and overcomes the identified weaknesses.

2.4.5 Agile Planning Tool Requirements

Both, Liu [21] and Morgan [38], identified a set of requirements for Agile Planning tools. While Liu focuses on co-located work, Morgan puts more emphasis on the distributed aspects of Agile Planning.

Liu [21] divides his 17 criteria into functional and interaction-related requirements as shown in [Table 1]. The 16 requirements identified by Morgan [38] cover Agile Planning and Groupware Tool Recommendations [Table 2].

Criteria	Criteria Sub Categories	Existent Agile Planning Support						
		Physical Planning Media	XP Planner	VersionOne	RallySoft	Wiki	MASE	GlueWiki
Agile planning objects creation and editing	Functional Requirements	√	√	√	√	√	√	√
Visual characteristics for different types of story		√	√/X	√/X	√/X	√/X	√/X	X
Agile planning metrics management		X	√	√	√	√	√	√
Planning for multiple iterations		√	√	√	√	√	√	X
Systematic organization of planning objects		X	√	√	√	√	√	X
Fluid transition between plan changes and the consequent results		√	√/X	√/X	√/X	√/X	√/X	√/X
Concept of team and identity authentication		√	√	√	√	√	√	√
Real-time exposure of the plan via the Internet		X	X	X	X	X	X	√
Simultaneous planning information organization	Interaction-related Requirements	√	X	X	X	X	X	X
Simultaneous planning information editing		√	X	X	X	X	X	X
Story editing with handwriting inputs		√	X	X	X	X	X	X
Handwriting recognition capability		X	X	X	X	X	X	X
Fluid transition between individual and collaborative work in agile planning		√	X	X	X	X	X	X
Fluid transition between agile tabletop collaboration and external work		√	X	X	X	X	X	X
Flexible user arrangement		√	X	X	X	X	X	X
Shared access to digital objects in agile planning		√	X	X	X	X	X	X
Use of physical planning objects		√	X	X	X	X	X	X

Table 1: Assessment for existing tool support for agile planning; Source [21]

Criteria	DAP	CardMeeting	AgilePlanner	MASE	Rally	VersionOne	Nplanner	Scrum works	GlueWiki	DotStories	XPWiki	Moomba	Tukan
Creating/Editing/Deleting Artifacts	F	F	F	F	F	F	F	F	F	F	F	F	F
Planning Metrics Management	F	N	F	F	F	F	F	N	N	F	P	P	
Planning Multiple Iterations	F	P	F	F	F	F	F	P	P	F	F	N	
Moving Stories between Iterations	F	P	F	F	F	F	F	P	P	F	F	N	
team & Identity Authentication	N	F	F	F	F	F	F	F	F	F	F	?	?
Real time exposure via Internet	F	F	F	F	F	F	F	F	F	F	F	F	F
Visual Characteristics for Story Cards	N	F	F	F	F	F	N	N	F	N	N	N	N
Integration with the development Env.	F	N	N	N	N	N	N	N	N	P	N	N	
Fluid Transition between personal/ private space	P	P	F	P	P	P	P	P	P	P	P	P	P
Telepointers	F	N	N	N	N	N	N	N	N	N	N	N	N
Flexible Information Shairing (WYSIWIS)	P	P	P	N	N	N	N	N	?	N	F	F	
Change Notification	P	P	N	P	P	?	P	N	?	?	F	F	
Radars for Awareness	N	N	N	N	N	N	N	N	N	N	N	N	N
Flexible Participant Management	F	F	F	F	F	F	F	F	F	F	F	F	F
Support for Subgroups	P	P	P	N	N	N	N	P	P	?	?	?	
Simultaneous interaction	F	F	P	N	N	N	N	N	?	?	?	P	

?=Not enough data available

Table 2: Tool comparison; Source [32]

The second part of Morgan’s requirements, namely the Groupware Tool Recommendations, is outside the scope of this thesis. As this work focuses on co-located collaborative planning, the criteria for distributed Groupware are not taken into account.

3 Digital Tabletop Implementation Environment

3.1 Hardware

The e-Business Engineering Group possesses a digital table designed by SMART Technologies [47]. Eight 19" flat-screen displays are embedded into a table surface with a size of 7' x 4' (210cm x 120cm). The visible display area measures approximately 5' x 3' (160cm x 65cm). These displays are surrounded by a 1' (30cm) wide border that can be used to put notes, handheld devices, mouse and keyboard or other auxiliary devices on it.

Each of the displays has a standard resolution of 1280x1024 pixels which sums up to an overall resolution of approximately 10 megapixels. Commonly used back-projection systems have a considerably lower display resolution, mostly in the order of one or two displays. This advantage of the LCD technology allows for a much higher number of objects to be shown on the screen in a better quality.

Cameras in the corners of the display area (SMART Tech DVIT) are used for image processing to determine the co-ordinates of a touch. Compared to the input resolution of standard devices like mice or the even more accurate stylus devices of Tablet PCs the DVIT's is much more coarse-grained. The closer to one of the cameras or the border of the table a touch occurs, the worse the accuracy of the recognition gets. This sometimes results in problems with the handwriting and gesture recognition functionality which requires a high number of input co-ordinates to work properly.

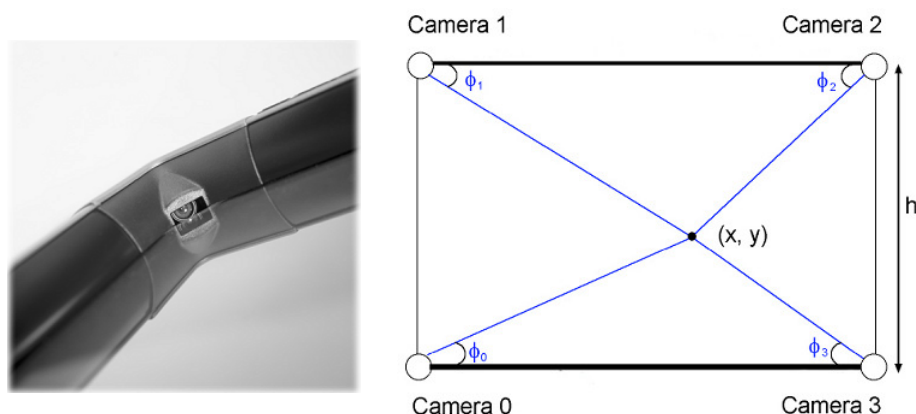


Figure 4: SMART Tech DVIT; Source [46]

The system is powered by a standard desktop PC running Microsoft Windows XP Professional. The PC is equipped with two Matrox QID Pro graphics cards [22] that can drive up to four displays each. These cards are designed for business applications that do not require first-grade hardware acceleration like it is provided by current high-end gaming cards.



Figure 5: SMART DVIT table at the e-Business Engineering Group lab

3.2 Software

Most of the requirements for the software part of APDT are a direct result of the problems identified when working with tabletop devices. Whatever technology is to be used it must:

- Allow for easy and smooth reorientation of content
- Allow for ‘handing over’ of any artefact on the table in order to bridge the distance between collaborators
- Provide handwriting recognition capabilities

Another requirement originates from the hardware platform. The total screen resolution of the digital table is approximately 10 megapixels. This means an enormous amount of graphics data that needs to be processed while the application must still be responsive to the users’ input.

Another non-functional requirement is related to the programming language used for the implementation. The time-frame for the development is very tight and thus only

high-level languages that automatically take care of low-level issues like memory management are taken into consideration. This basically narrows down the list of available languages to either Java or one of Microsoft's .NET languages (C#, Visual C++ .NET or Visual Basic).

The handwriting recognition requirement indicates the use of one of the .NET languages as there is no comparable counterpart available for Java at the moment. In addition, an approach to create rotatable widgets using Java Looking Glass [14] proved to be fairly complicated as it involves low-level hardware and operating system programming to achieve reasonable performance.

3.2.1 Evaluation of Existing Frameworks

The University of Calgary's Interactions Lab (iLab) [15] currently develops a graphics framework that specifically targets large horizontal displays. The current implementation of the Large Display Framework (LDF) supports widgets that can be moved and rotated unrestrictedly using OpenGL. The architecture of the framework also allows for implementations using other technologies (e.g. DirectX).

The framework is implemented in unmanaged (or native)¹ C++ which is not compatible with Microsoft's .NET enhanced version of C++. Thus, some kind of wrapper needs to be implemented to make its functionality available to an application written for the .NET Framework.

A simplified overview of the LDF is shown in Figure 6

¹ The terms unmanaged C++ and native C++ are used interchangeably and in the context of this thesis they both refer to the version of Microsoft's Visual C++ which is not interpreted by the .NET Common Language Runtime (CLR).

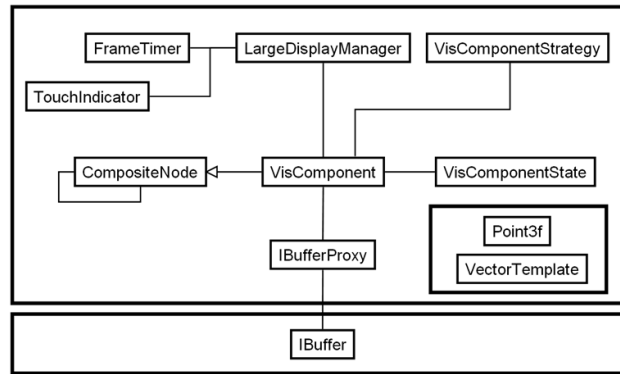


Figure 6: Overview of Large Display Framework; Source [4]

The three parts of the framework that are most important for the development of an application are the manager that takes care of the creation and destruction as well as the drawing of all displayable objects, the components that represent these objects and the strategies that define how a component looks like and behaves. A set of strategies is already provided by the framework. This includes RNT (unrestricted rotation and movement), Tossing (which makes the objects slide across the table) and strategies to resize and pick objects.

In its current state of development LDF does not support standard text widgets like text boxes, which allow for input, or labels, which display fixed text. This major drawback cannot be compensated easily. Although it is possible to display text on the components based on Glut's [12] text rendering it is only a very weak replacement for real widgets which take care of layout and formatting of text automatically.

A massive overhead arises from the necessity to write additional strategies for any extra functionality not contained in the framework in unmanaged C++ as well as writing a managed wrapper to make it accessible for the .NET part of the application.

The wrapping itself is quite troublesome. The automated Garbage Collection (GC) of the .NET Common Language Runtime (CLR) causes a lot of problems when it is necessary to access managed objects from unmanaged C++ code. The Garbage Collection moves objects in the memory and updates the (managed) object references (called handles to distinguish them from unmanaged pointers) automatically. The well-known pointers in C++ in contrast are 'static' which means

that they will still point to the same memory address even though the GC moved the object that was originally referenced, thus causing security exceptions at runtime.

```
IntPtr namePtr =  
System::Runtime::InteropServices::Marshal::StringToHGlobalAnsi(name);  
  
pin_ptr<char> namePinPtr =  
reinterpret_cast<char*>(static_cast<void*>(namePtr));
```

This code-sample is used to convert a simple C# `string` object to a C++ `char*` pointer. The second command orders the CLR to create a 'pinned pointer' which means that the GC will not relocate the managed object in the memory. To find and eliminate the cause of runtime-exceptions with no meaningful error message, with constructs like the above, is extremely time-consuming and slows down the development process considerably.

Yet another problem is the portability of the application to different machines. It is rather hard to identify and include all necessary libraries and make them available at the proper locations whereas the .NET environment is quite simple and convenient to setup.

Microsoft's version 3.0 of the .NET Framework fulfils the above requirements. It comes with a new graphics framework called Windows Presentation Foundation (WPF), which provides an easy-to-use mechanism for the rotation of all available UI widgets apart from application windows.

With the Tablet PC technology, there is a convenient way available to include gesture and handwriting recognition support in an application based on the .NET Framework.

During a short evaluation, WPF proved to be high-performance even on a screen as large as the one in the table. The performance of LDF drops dramatically when the application window is enlarged to a point where both graphics cards have to be used to display content. This is due to the fact that cross-graphics-card data exchange is not optimized for these cards. The effects of this lack of optimization are negligibly small for WPF.

3.2.2 Development Environment

Considering the problems with LDF, the decision to use WPF and completely rely on C# for the implementation is a reasonable decision for this project. Unfortunately, version 3.0 of the .NET Framework is not very well supported by Microsoft's Visual Studio 2005 (VS2005). The extensions which retrofit this support still contain well-known bugs but with hindsight to the upcoming release of VS2008 their development is already discontinued. For better development performance it is recommended to use VS2008 Beta 2 ('Orcas'). Although this beta-version is still far away from a stable working state, it provides the crucial functionality missing in the preceding version.

The final development environment consists of:

- Microsoft Windows XP Professional (Service Pack 2)
- Microsoft Visual Studio 2008 (Beta 2)
- Microsoft .NET Framework 3.5 (Beta 2)
- Windows XP Tablet PC Edition Development Kit 1.7
- Windows XP Tablet PC Edition 2005 Recognizer Pack
- Ink Analysis and Input Supplement for the Windows XP Tablet PC Edition Development Kit 1.7
- Speech SDK 5.1

For the use with Windows Vista² the Vista SDK [30] contains all components to support gesture-, handwriting- and speech-recognition thus the supplemental components are obsolete.

² The Tablet PC functionality is currently only available for 32-Bit applications. Software compiled for 64-Bit processors will not be able to use handwriting- and gesture-recognition

To document the code, a tool called GhostDoc [55] is used which generates the outline of comments that can be compiled into API description pages similar to those known from JavaDoc.

Unit tests are written using MbUnit [24] along with the TestDriven.NET plug-in for Visual Studio [52].

4 Agile Planner for Digital Tabletops (APDT)

This chapter will present an overview of Agile Planner for Digital Tabletops (APDT), its architecture and design as well as its User Interface (UI).

4.1 Application Structure

For a better organization and maintainability the application source-code is divided into several packages (or namespaces in the .NET terminology). One each for data objects and the according interface definitions, one for display related classes, one for persistence and communication, one for the application core, unit tests, resources like images and the compulsive utilities that fit nowhere else.

The application is based on the Microsoft .NET Framework which provides the means for graphical output (WPF) as well as voice (Speech) and gesture input (Ink).

Figure 7 shows an overview of the package structure of APDT.

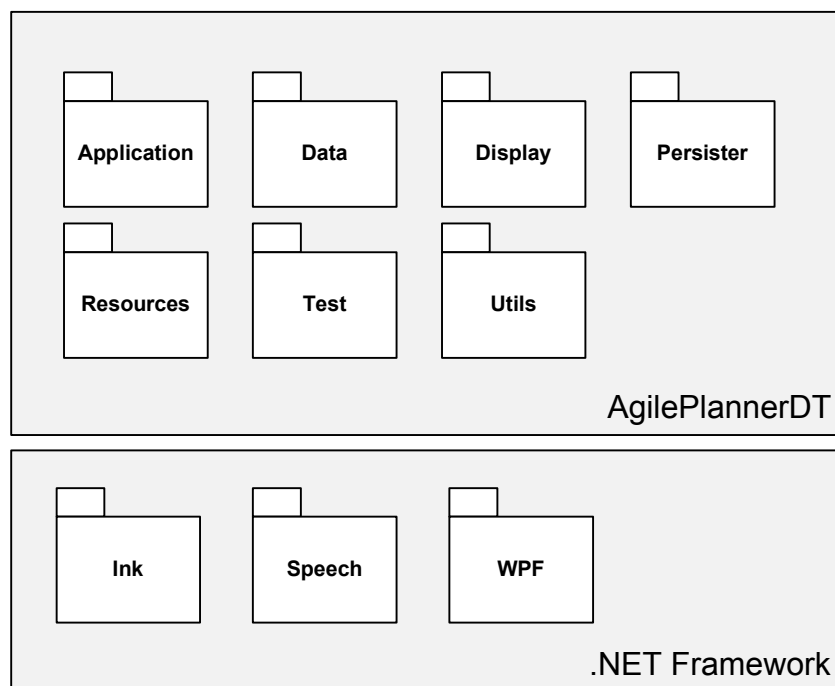


Figure 7: Package structure of APDT

4.1.1 Planning Artefacts

The `Data` package contains interfaces and implementations of all planning data related objects. The contents of this package encapsulate all functionality that is needed for interacting with planning data like changing estimates of a user story, setting deadlines for iterations and the like.

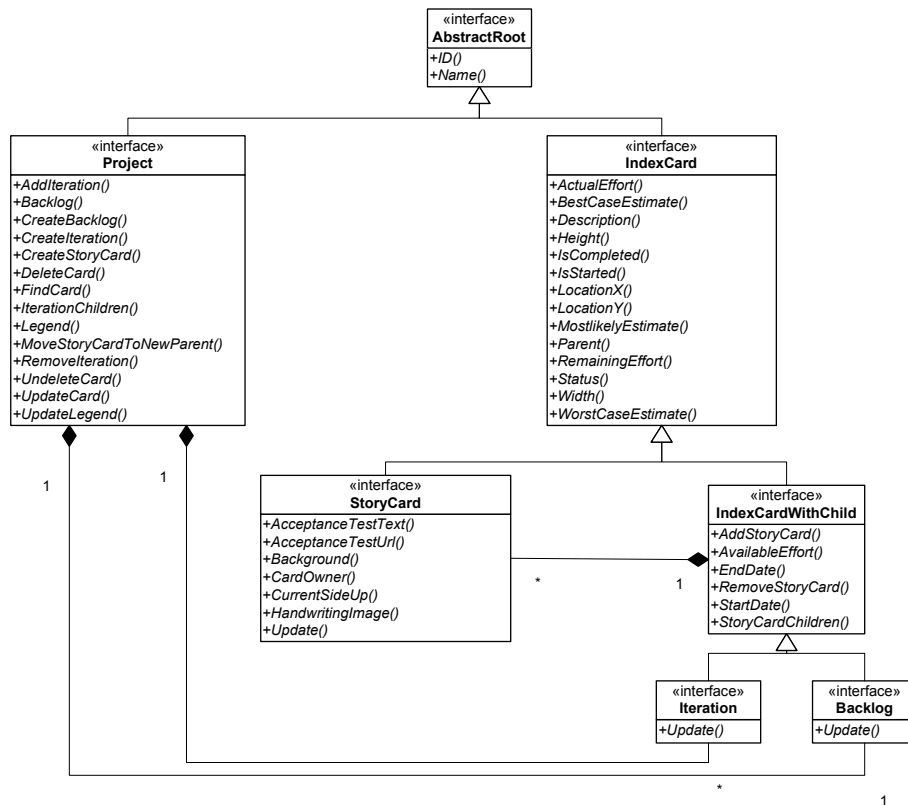


Figure 8: UML diagram of AgilePlannerDT.Data

The base of the data inheritance hierarchy is called `AbstractRoot`. It defines functionality that is necessary to explicitly identify planning artefacts.

The top-level container for planning data is the `Project`. It consists of a `Backlog` which contains all user stories that are of potential interest for the future but did not make it into the plan of an iteration, and of an arbitrary number of `Iterations`. Iterations are used to group the user stories, which are planned to be worked on in a specified timeframe. Although the number of iterations is not limited, there is a standard of having between one to three iterations (last, current and next).

Older iterations may be saved to be able to keep track-records of the project but

often they are simply discarded after the work on them is finished.

`StoryCards` represent user stories and contain various information that is relevant for their tasks e.g. effort estimates, the name of the responsible developer, a detailed description etc.

4.1.2 Visualization of Planning Data

The `DisplayObject` interface defines basic behaviour for all objects that may appear on the screen starting with geometric properties like center coordinates or size, to functionality related to the movement of objects on the display. `IterationDisplayObject` and `StoryCardDisplayObject` further refine this definition by adding functionality that is specific for the management of iterations and story cards like adding or removing cards to and from iterations or resizing cards for handwriting.

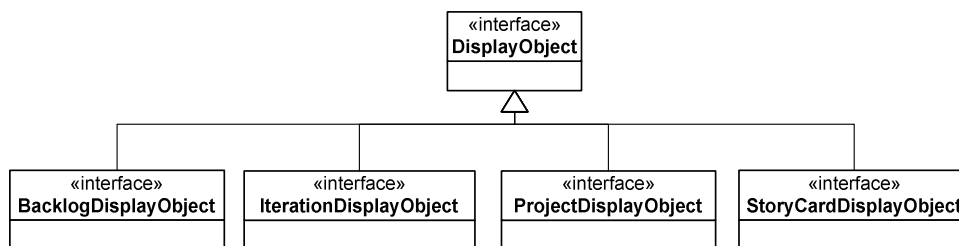


Figure 9: UML-diagram of AgilePlannerDT.Display

4.1.3 Persistence and Communication

The `Persister` interfaces are used to hide low-level operations related to persistence and communication. They offer a well-defined interface that abstracts from the underlying implementation.

The application only deals with implementations of the asynchronous interface and must itself implement the `PlannerDataChangeListener` interface to provide a call-back-channel to be notified of changes to data objects.

The synchronous interface defines a very similar set of methods. Experience from the DAP project shows that this interface should also be provided for convenience.

A set of application specific exceptions allows for proper handling of undesired application states and events.

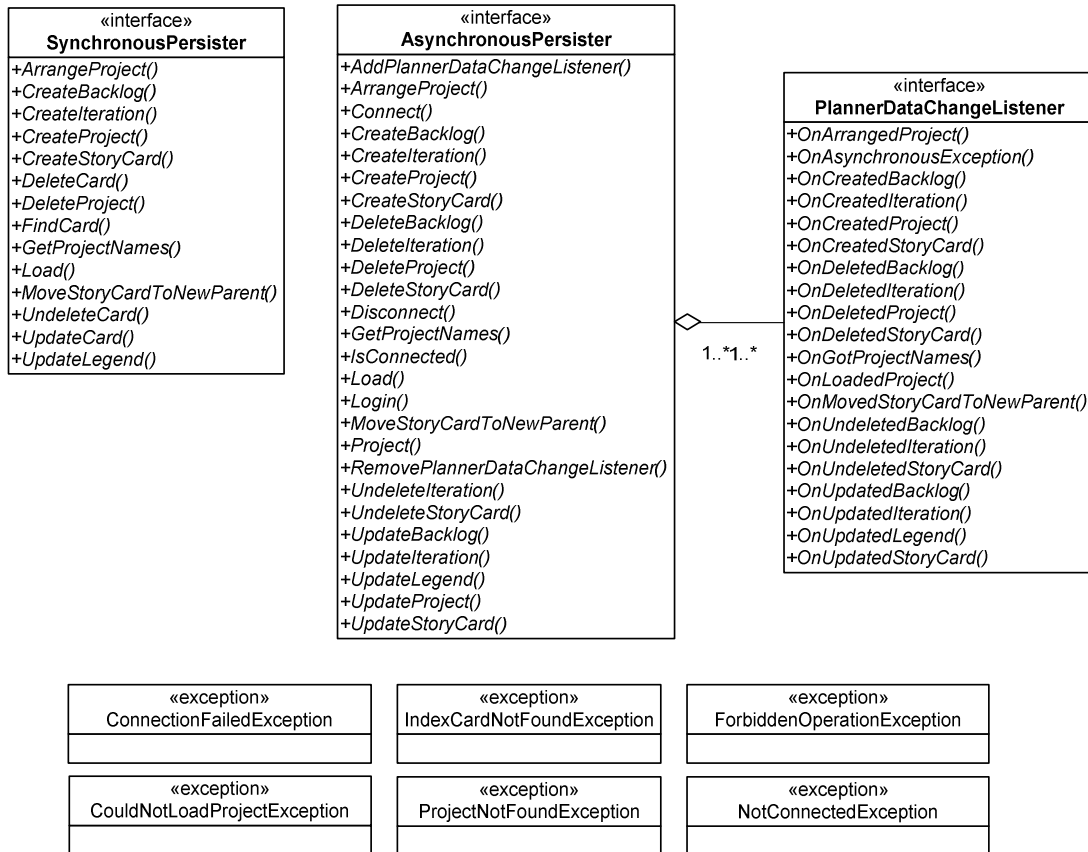


Figure 10: UML-diagram AgilePlannerDT.Persister

4.1.4 Application Core

The application core combines all auxiliary components and serves as the hub that manages input and output.

The `Commands` are used to allow for navigation through the history of interactions with the application. They provide the means of a back/forward functionality known from web browsers or desktop applications.

The `ApplicationModules` are used to outsource certain functionality to allow for a better overview and maintainability of the application. Currently implemented modules cover gestures, speech, card originated event handling and persistence. Using the predefined interfaces, new modules can be easily added if necessary.

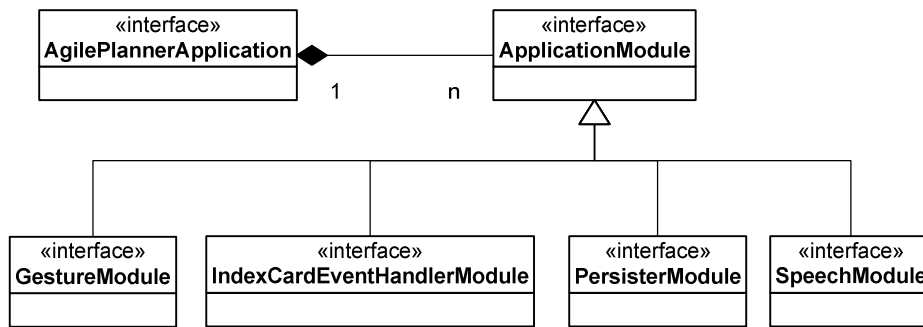


Figure 11: UML-diagram AgilePlannerDT.Application

The `GestureModule` is responsible for the definition of a set of gestures that the application can recognize as well as the handling of recognized gestures.

The `IndexCardEventHandlerModule` deals with all events that are related to `Iterations` or `StoryCards`. This comprises `DropEvents`, which signalize the end of a card motion, value changes of card properties (e.g. estimates, size etc.) or specific input events that need to be handled separately from standard inputs.

The `PersisterModule` is the link between the application and the `AsynchronousPersister` and handles all incoming and outgoing calls through the persistence API.

The `SpeechModule` encapsulates functionality related to speech recognition and corresponding actions that are triggered on recognized commands.

Finally, the `AgilePlannerApplication` interface defines a set of commands that is used by the modules and the commands to interact with the application.

4.1.5 Utilities

The `Utils` package contains custom converters for application specific data types and a class that bundles helper functionality, mainly for geometric operations.

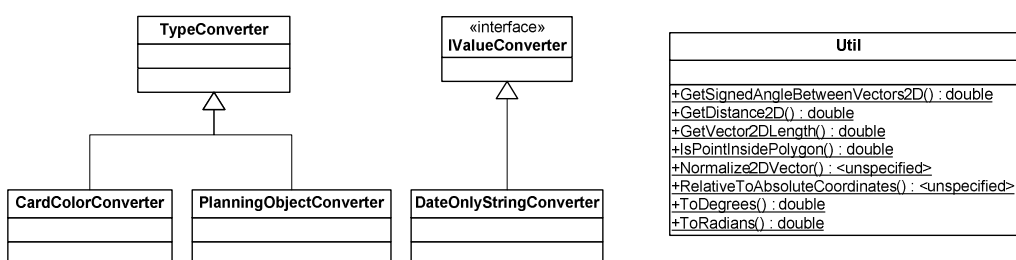


Figure 12: UML-diagram AgilePlannerDT.Utils

The converters follow the structures defined by the .NET Framework and can be integrated into the WPF infrastructure.

4.1.6 Unit Testing

The `Test` package contains the unit tests for the backend part of APDT, namely the data objects and the `Persister`. They are used for the automated continuous integration builds of the application which are triggered at certain times by the EBE build server.

4.1.7 Resources

This package contains resource files like pictures to be used in the application.

4.2 User Interface Overview

When most people talk about a User Interface (UI) they automatically think about menus, toolbars and buttons as they are the most common representatives of their kind for desktop applications.

But a UI in the broader sense refers to much more than just that. To interact with an application, one must do so through its UI. One example is mouse gestures that have become increasingly popular for applications like internet browsers [9] [41]. Another means to communicate with the application are Alouds or Voice Commands that trigger predefined actions in an application or an operating system such as Windows Vista [35].

Tabletop environments render standard graphical user interfaces (GUI) almost useless. People sit or stand at any position around the table, they can be too far away to reach a specific menu item or they have to read a button's text upside down.

One way to overcome this issue is to use personal toolboxes for every user, which allow similar manipulation (i.e. unrestricted rotation and movement on the table) as it is required for any content.

A major drawback of this kind of user interface is that all of these toolboxes require extra space on the display and therefore they may hide content. In addition these toolboxes interrupt the flow of card-based planning. Nobody would click on a button to create a paper index-card nor would anyone select an option in a menu to remove the card if it becomes obsolete.

4.2.1 Gestures and Handwriting Recognition

As mentioned above, gestures have become increasingly popular for controlling applications. Microsoft introduced this command mechanism with the Windows XP Tablet PC Edition and it is now an integral part of the .NET Framework 3.0.

The technology underlying the gesture recognition is called Ink and is fully integrated with the design of graphical interfaces.

Ink enhanced input areas are easy to create and provide a rather large set of functionality. The framework comes with a built-in set of gestures it can recognize. Also, there are tools which offer the functionality to create custom gesture recognizers [49].



Figure 13: Creation of a new Iteration using a gesture

The orientation of these gestures can be used to determine the user's position relative to the table so that, in example, the creation of a card with a chevron gesture [26] displays the new card 'the right way up' and the user can instantly fill the card with content instead of having to turn it towards him to make it readable.

Table 3 shows an overview of the gestures that are currently supported.

In addition to the gesture support, Ink input areas can be used for handwriting recognition. As mentioned earlier, this functionality is considered a crucial feature for tabletop planning tools.

A problem with handwriting input is that the DVIT is not able to distinguish between the touch of a fingertip and a touch with the wrist as it might occur when writing with a pen. A solution to this problem is to enlarge the index cards when the user wants to write on them, and shrink them to their normal size when the writing is finished. That creates more of a feeling of writing on a whiteboard than on a paper index-card. But although this kind of handwriting input is not exactly the same as its

archetype, users perceive this method to be quite convenient especially in comparison with the need for additional hardware devices like Pocket PCs.

A benefit that comes with this solution is that the recognition process is explicitly triggered after the editing is finished and there is no need to estimate the end of the user's work with the card.

4.2.2 Speech Recognition

The concept of voice commands has been around for some time now but is not used much in mainstream applications. The recognition of everyday speech still is a field of intense research. Recent frameworks like Dragon Naturally Speaking [39] or Microsoft Speech [28] support the dictation of texts with rather good results but still need fine-tuning and training on a per-user basis [42].

The Microsoft Speech technology is available for the .NET Framework and offers the integration of basic speech recognition features in stand-alone applications as well as server-based recognition for more complex areas.

This thesis focuses on the implementation of easy-to-distinguish voice commands as an alternative to the mouse gestures rather than the full support for text dictation. The user should be able to invoke all main actions either using the mouse or by vocalizing the commands assigned to those actions.

The accuracy of the recognition is surprisingly high. Even without previous training the application is able to recognize commands by a randomly selected group of members of the EBE group. Most of whom are non-native speakers with varyingly strong accents.

While gestures and handwriting require physical interaction with the table surface voice commands do not. Thus the distance to the table is not an issue for this type of user input.

By default, all new items are created in the center of the display as there is no way to identify the speaker's position relative to the table. Currently, there are no voice commands to turn or move items in order to bring them into a position where the speaker can read or edit them manually. These commands may be added in a later release.

Table 3 shows a list of the voice commands that are currently supported.




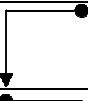

Application Command	Action	Voice Command	Gesture
Create card	Creates a new user story at the location of the gesture or in the center of the screen if voice command	Create story card	
Create iteration	Creates a new iteration container and collects all existing user stories at the creation position	Create iteration	
Delete card/iteration	Deletes the artefact that is scratched out or the one currently selected if voice command. Deletes all user stories in an iteration.	Delete card	
Select Card	Displays the IDs of all artefacts on the table and allows to specifically select one of them.	Select [ID]	--
Next / Previous Card	Navigates between cards on the table.	Next Card / Previous Card	--
Undo	Reverts the last executed command.	Undo	
Redo	The last reverted command is executed again.	Redo	

Table 3: Overview of voice commands and gestures in APDT; Source [11]

5 APDT in Detail

5.1 Realistic Interaction Behaviour

When people write on paper index-cards, they follow a metaphor that has been used for a very long time. They have certain expectations about how the writing should feel and what behaviour the cards should show when being subjected to different actions. A card, that is given enough momentum, should slide across the surface. It should be possible to rotate a card to achieve a better readability and so on.

These criteria need to be met by an application whose goal is to contribute to an existing process and not to alter it by forcing the users to change their behaviour.

A lot of research has been conducted in the area of Human Computer Interaction (HCI) to find out how it is possible to make interaction with non-physical objects feel 'real'. A finding of special interest for this thesis is the Rotate 'N Translate (RNT) algorithm by Kruger et al. [19]. This algorithm allows for integrated moving and rotating of objects on a screen with just one touch-point, e.g. a fingertip.

Objects can be repositioned in one fluid motion and there is no need for a second action to give them the right orientation. The algorithm is designed to be platform independent and therefore takes into account that the restriction of single-touch-point-input applies to the largest part of all existing computer environments.

Experience from work with APDT shows that people tend to 'grab' objects with their hands rather than touching them with only one fingertip when they want to rotate them. Nonetheless, users adapt very quickly to the single-touch restriction that the system enforces.

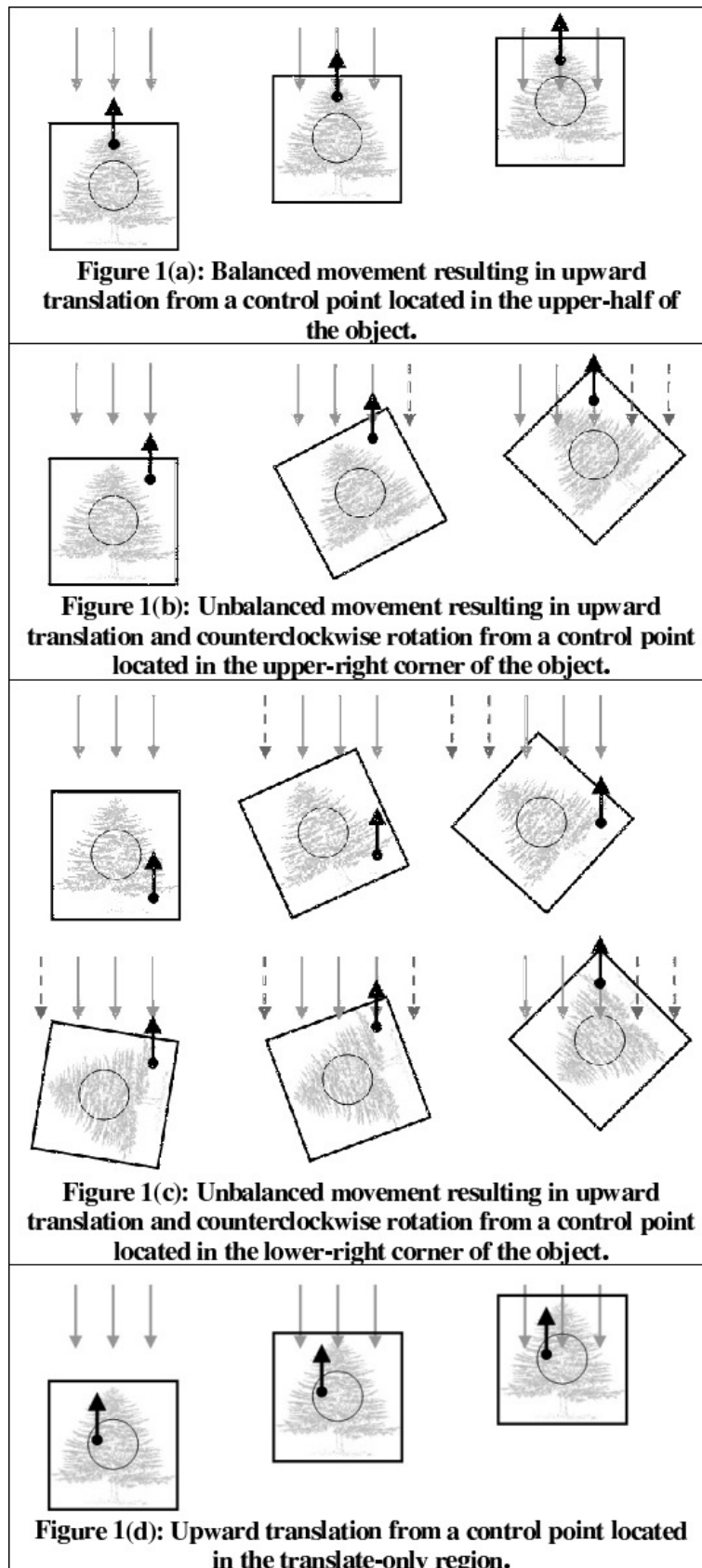


Figure 14: Illustration of RNT motion; Source [19]

The pseudo-code-sample describes Kruger's RNT algorithm [18] in more detail.

```
/* This function is called each time the mouse is pressed to determine
   if the object has been clicked on, and, if so, the appropriate
   interaction mode (i.e. translate-only vs. simultaneous translation and
   rotation) is set. */

Function MousePress

    If(mousePress is within object) Then
        If(mousePress is within translate-only region) Then
            Mode = Translate
        Else
            Mode = Translate & Rotate
        End If
    End If

    OldMousePosition = CurrentMousePosition

End MousePress

/* This function is called each time the mouse is moved, calculating the
   required translation vector and rotation angle (if necessary) to
   re-position the object. */

Function MouseMove

    If(Mode = Translate) Then
        Translate between OldMousePosition and CurrentMousePosition
    Else If (Mode = Translate & Rotate) Then
        CO = Vector from object center to OldMousePosition
        OT = Vector from OldMousePosition to CurrentMousePosition
        CT = Vector from object center to CurrentMousePosition

        alpha = ArcTan (CT.Y / CT.X)
        beta = ArcTan (CO.Y / CO.X)
        theta = beta - alpha

        Rotate object about OldMousePosition by theta
        Translate object by OT
    End If

    OldMousePosition = CurrentMousePosition

End MouseMove
```

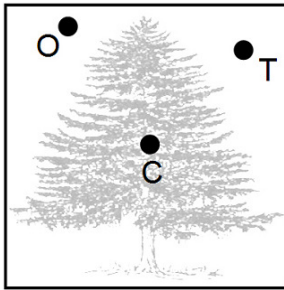


Figure 15: Three required points determine the RNT algorithm: C - Center of the object; O - the Original mouse position and T - the new Touch position (or Target); Source [19]

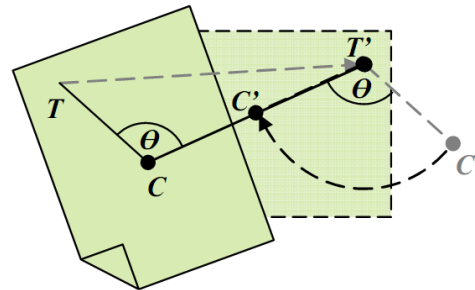
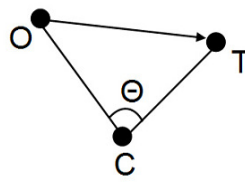


Figure 16: The RNT algorithm uses a physically-based model to calculate the rotation at each frame. The object acts as if a force of friction is being applied opposite the direction of movement; Source [13]

The actual graphical rotation and movement of a displayable object is very smoothly integrated into WPF. Geometric transformations are available to move (`TranslateTransform`), rotate (`RotateTransform`), scale (`ScaleTransform`) and skew (`SkewTransform`) objects in 2D-space.

All standard UI-elements possess two designated properties that are used to register these transformations. The `RenderTransform` property applies the transformations without affecting other UI-elements, i.e. enlarging an object by scaling it does not influence the layout of the surrounding widgets, whereas `LayoutTransform` can be used, if an influence on other elements is the desired behaviour.

If multiple transformations are to be applied to the same object, a special container for these transformations, called `TransformGroup`, must be used.

With these transformations in place, it is quite simple to reposition an object by just manipulating the appropriate transformations. All transformations can be applied relative to a specific location in the co-ordinate system, e.g. objects would usually be rotated around their center. If applied to the main-window with an origin in the center of the WPF co-ordinate system, a `ScaleTransform` can be used to scale down the whole application in order to make it fit on smaller screens. This approach is used to demo APDT on a Tablet PC but not integrated as a configurable feature.

With this implementation users can place objects at any position within their accessible radius but what can be done about handing objects over to people that are not close enough? In the physical world, paper index-cards are simply thrown or

tossed across the surface of the table to come around this obstacle. In order to stick as close as possible to that metaphor, a mechanism which makes it possible to toss a card is necessary to make the users feel comfortable in the electronic environment.

WPF does not permit low-level influence on the rendering process, like for example the Qt Toolkit [53] does in C++. Instead of redrawing objects frame-by-frame, WPF provides a more convenient mechanism, called `Animations`, which automates this process to a large extent.

`Animations` are available for a wide range of data types of the .NET Framework and cover an equally wide field of application scenarios [25]. A simple example for such an animation is the `DoubleAnimation` which generates double precision floating-point values in a definable range. One scenario where such a `DoubleAnimation` can be used is a fading effect for objects where the opacity value of the widget is animated from 0 (invisible) to 1 (solid). The parameters of the animation can be largely influenced for example by setting the step size of the generated values or the duration of the animation.

Animations are designed to work on a special type of object property, called `DependencyProperty`. As virtually all UI-element properties in WPF, e.g. an object's opacity, width, height etc., are implemented as `DependencyProperties`, this limitation is only of importance for custom object properties a developer might want to animate.

It is important to know that these properties are not really altered by the animation. They keep their original value but the currently effective value, which means the one that is used as a basis for the rendering, is changed. These effective values can be made permanent at the end of an animation or they can be reset to the original value. Properties, which were formerly animated, are 'locked' as long as the animation is in effect. If such a property is changed elsewhere in the application, this change will have no effect until the animation is either stopped or the property is explicitly removed from the target list of the animation.

If several animations are to be applied to an object (or a set of objects) at the same time, they can be combined to a `Storyboard` which controls all animations at once instead of having to deal with them one-by-one.

In the physical world, an object continues to move, if its original momentum is high enough. If a paper index-card is pushed fast enough, it will slide across the table until the friction of the surface slows it down enough to stop it.

A virtual object on the screen can only be moved by the touch of a user, either with a mouse pointer or a fingertip. Thus, it is sufficient to monitor the `MouseMoveEvents` that are fired after the initial contact with a tossable object is established. Each of these objects contains a ring buffer for the co-ordinates of the last few `MouseEvents`. When the object is released, i.e. the fingertip is removed or the mouse button is released, the distance between the first and the last of the co-ordinates in this buffer is calculated. If the distance is big enough which means the movement of the object was fast enough the tossing function is triggered.

The RNT algorithm is also integrated into this tossing function to calculate the intermediate steps of the animation. Thus the resulting movement can also rotate the object if it is given enough spin.

```
/* This function is called each time an RNT-enabled object is moved fast
   enough across the surface, to calculate the intermediate steps of the
   object's movement to simulate realistic behaviour. */

Function RNTTossing

    While (Tossing conditions apply)
        Simulate friction to slow down movement
        Calculate next step of object's movement using RNT
        Assure that object remains within window bounds
    End While

    Calculate duration of movement using rule of thumb approximation
    Animate object

End RNTTossing
```

Realistic behaviour implies that a sliding card can be stopped when a user puts its finger on the card. In order to achieve this, the animation must be stopped. As mentioned before, the animation does not really alter the values of animated properties but only manipulates their effective value. The intended behaviour of stopped animations is either to reset the effective value or to make the final state permanent. Both would make the object jump on an untimely interruption, either forward or backward, which is definitely not desirable. However, there is a way to

access intermediate values of an animation and change the real property value accordingly before the animation is terminated.

As shown in this chapter, transformations and animations provide a powerful set of functionality that, in combination with algorithms like RNT, create very realistic behaviour of virtual objects.

5.2 Multi-modal Input

New technologies offer new possibilities but they also pose new challenges. The digital table technology allows for the integration of computers in formerly 'analogous' environments. But in order to make a beneficial contribution to the process of Agile Planning, new ways to interact with the machine must be investigated.

Mouse and keyboard, two input devices which are used since the invention of the very first graphical interfaces, are not very well suited for environments where multiple users want to be able to interact with an application. Nor does their usage feel 'natural', in the sense that nobody would use these devices for a regular planning meeting. Instead, people would grab a pile of empty index-cards and a pen and then bring the team together at a table or in front of a whiteboard.

This kind of team interaction has been around for a long time and people are used to following this procedure. So, the question is: How can an application support this process in a non-intrusive way in order to preserve its familiarity to the user while making valuable additions to it and therefore encourage the migration to an electronic environment?

5.2.1 Gesture Recognition

The use of gestures to control an application has been made popular especially by web browsers like Opera [41] and Mozilla Firefox [9]. Lately, the Apple iPhone [5] set of a hype about gesture-controlled electronic devices.

These gestures offer a time- and effort-saving way to accomplish routine tasks which would otherwise require a complicated or interrupting sequence of actions. An example for such an action is the 'Back' function of a browser. If the user wants to go back to the last visited web site, he has to move the mouse pointer to the menu bar and then back into the content area of the browser window to continue his work. In Firefox, a gesture that triggers the same function is a simple flick to the left with

the right mouse button pressed. This gesture is easy to learn and offers a more convenient way of traversing through the history of visited web sites than using the button-navigation. And by allowing the cursor to stay inside of the content window, it reduces the interruption of the workflow to a minimum, thus saving time.

Handwriting and gesture recognition are the core of Microsoft's Tablet PC platform and are fully integrated into the Windows XP Tablet PC Edition and Windows Vista operating systems. In order to make them available on a regular Windows XP machine, a set of supplemental software is necessary [29] [31] [27].

Microsoft calls the technology underlying this input method Ink and the base UI-element for stylus-input in .NET 3.0 is the `InkCanvas`. This element provides means to recognize a certain predefined set of gestures [26] and to display the strokes drawn by a stylus (or a mouse simulating a stylus). With the `GestureRecognizer` and the `InkPresenter`, separate classes for the integrated functionality of the `InkCanvas` are available which can be attached to any container widget.

5.2.2 Handwriting Recognition

After the integration of the gesture recognition into APDT, doing the same for handwriting is the next logical step. The technology of the handwriting input follows the same basic principles as the one for gesture input. The problems with this next step are therefore not really caused by the handwriting recognition itself but by limitations of the hardware used for the implementation.

The technology, which SMART uses to identify the location of a touch on the table surface, is based on cameras and video analysis. The accuracy of this method is subject to several factors: The closer to one of the cameras or to the border of the table a touch occurs, the worse the precision of the recognition becomes. Dust on the tabletop also seems to have a negative impact, although this one can be avoided by cleaning cameras and surface regularly.

But even then, the touch-point of a solid object like a pen or marker, seems to 'jump' on the surface as the DVIT reports different co-ordinates for the, in fact, immobile tip of the object. This effect is even more problematic when a fingertip is used for writing, not to speak of a user placing its wrist on the surface to write in a natural pen-and-paper style. In addition, the DVIT registers objects in close proximity to the

surface as a contact. Although SMART considers this to be a feature, the effect is quite disturbing for the interaction with the application.

To solve these issues, a compromise has to be found between imitating the real-world process and the limitations of the virtual environment. The current solution is a dedicated 'handwriting mode', in which a card is enlarged to several times its original size. In this size, writing on the card is very similar to writing on a whiteboard with either a marker or a fingertip. Chapter 5.3.1 explains the handwriting-mode in more detail.

It is also possible to erase formerly written text from the `InkCanvas`, so the first step for the recognition is to check if any strokes have been drawn at all. If that is the case, the data of these strokes is converted into an appropriate data structure, which is stored along with the rest of the planning data of the respective user-story. The actual recognition is done by an `InkAnalyzer`, which analyses the set of collected strokes and returns a string containing the recognized text if the process was successful.

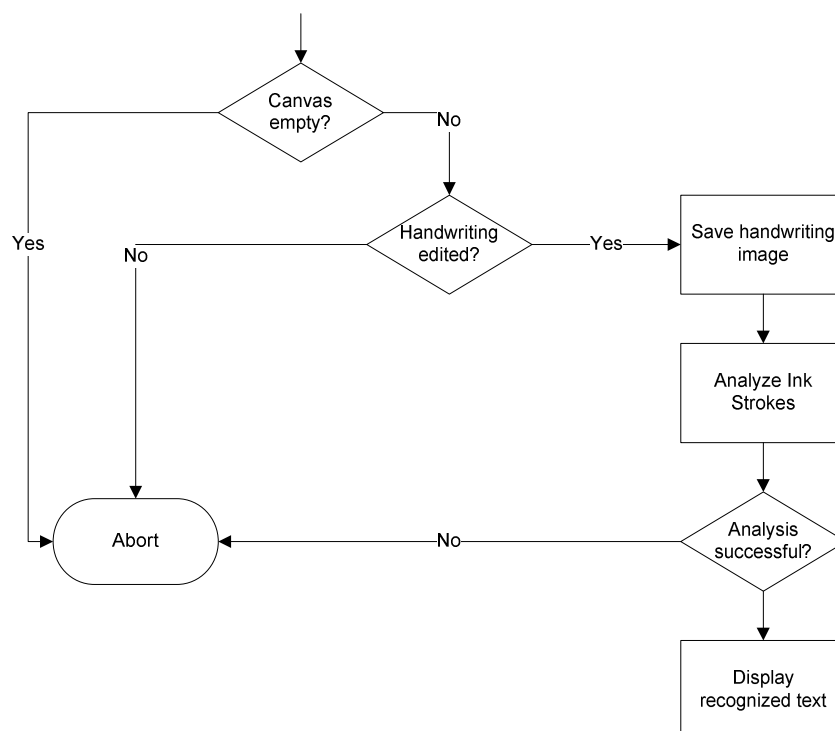


Figure 17: Handwriting recognition process

Currently, the recognized input is displayed as computer text along with the original handwriting, as shown in Figure 21.

An important problem regarding the Ink technology is that it currently works only with 32-bit applications. In applications compiled for 64-bit processors, the use of any of these components will result in an exception with a misleading error message, which points to a missing library instead of the actual compatibility issues.

5.2.3 Speech Recognition

People communicate with each other using speech and body-language. One approach to make the interaction with a computer feel more natural is to make the machine understand human language.

Several commercial tools are available [39] [28] which try to fulfil the promise of machines understanding humans. As Microsoft Speech is integrated with the .NET Framework, it is obviously a good candidate for a proof-of-concept utilization as an additional input channel.

Speech recognition is available for Windows XP and Vista but only Vista supports the very latest version of the Microsoft Speech API (SAPI). Not only does this prevent the use of some interesting features related to the recognition of natural speech, but it also limits the number of languages which can be recognized.

```
public SpeechModuleImpl(AgilePlannerApplication app, Dispatcher dispatcher)
{
    ...

    sharedRecognizer = new SpeechRecognizer();
    sharedRecognizer.Enabled = true;
    sharedRecognizer.UnloadAllGrammars();

    GrammarBuilder gb = new GrammarBuilder(new Choices(new string[]
        {"create iteration", ... , "delete card"}));

    //on WinXP the speech recognition only works with US english
    gb.Culture = CultureInfo.CreateSpecificCulture("en-US");

    Grammar commandsGrammar = new Grammar(gb);
    commandsGrammar.SpeechRecognized += new
        EventHandler<SpeechRecognizedEventArgs>(speechRecognized);

    sharedRecognizer.LoadGrammar(commandsGrammar);
}
```

The code sample illustrates the creation of a simple speech recognition function for a predefined set of commands. In order to listen for commands in every day

language, e.g. a team member stating that 'we need a new story card describing task xyz', a more sophisticated grammar is needed. This grammar is specified using XML-files of a standardized format [57] and can be used to handle less formalized vocal input within a limited topic (e.g. ticket reservations).

Another feature of the Speech API is the possibility to dictate text. While the command recognition is rather robust to the different dialects or accents of non-native-speakers, a short evaluation of the text dictation module, which is shipped with Windows Vista showed that this is not the case for this module.

5.3 Planning Artefacts

5.3.1 Story cards

APDT supports two basic visual types of planning artefacts: Story cards and Iterations. Story cards are explicitly designed to look like paper index cards to keep the metaphor as plain and simple as possible and minimize the time users need to learn how to interact with the application. These cards have predefined fields for various information like the name of the task, estimates of how long it will take to finish work on this user story, the name of the developer who is responsible for this story and a field for a detailed description of the user story.

Figure 18 shows a newly created story-card with the default color and content.

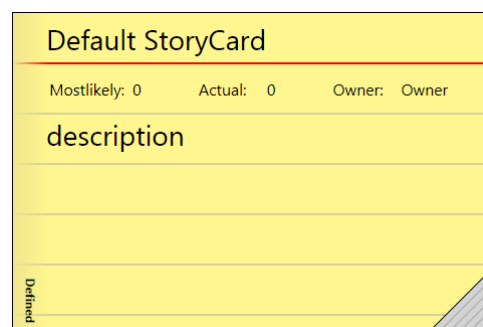


Figure 18: New story card

The bottom right corner of the card can be used to resize the card. A standard 3:2 aspect ratio is maintained during the resizing process to keep up the visual similarity between paper cards and their electronic counterparts.

The left side of a card is used to display its current status. The predefined set of status consists of 'Defined', 'ToDo', 'In Progress', 'Completed' and 'Accepted'. A

newly created card is automatically assigned the 'Defined' status. When the card is part of the current iteration it moves from 'Defined' to 'ToDo'. As soon as a developer starts to work on the user-story the status will change to 'In Progress' and later to 'Completed' when the work has finished. 'Accepted' is the final stage in the lifecycle of a card which is assigned when the customer declares the result as satisfying.

In addition to the text description the left border of the card is coloured differently as a visual indicator for each status as shown in Figure 19.

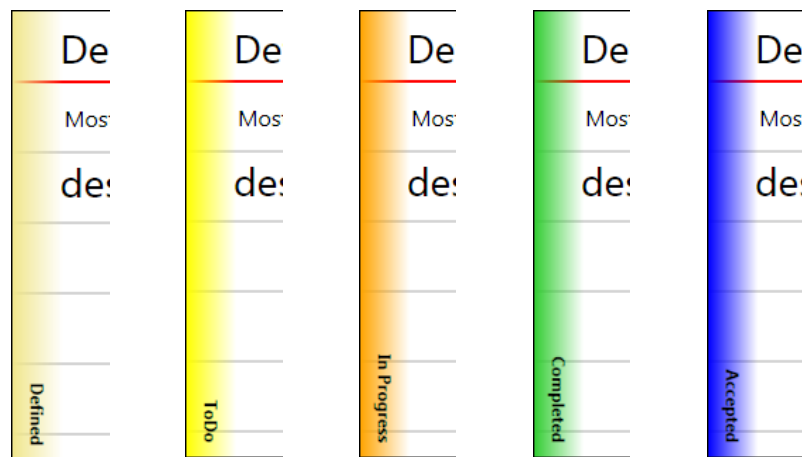


Figure 19: Story card lifecycle

The background color of the cards can be changed in order to assign different colors to different types of tasks. E.g. the color red might be used to indicate a high-priority task or a bug fix.

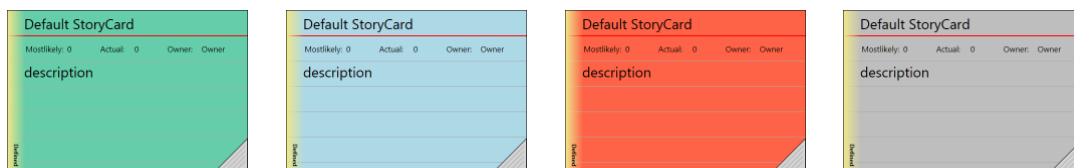


Figure 20: An extract of available colours

A double-click on the card or a tap in the upper right corner sets the card to handwriting-mode. In that mode, the card is enlarged to make writing easier and the description part of the card becomes sensitive for handwriting. A red pin in the upper right corner is used as a visual indicator that the card's position is fixed so that the card cannot be moved until it is switched back to normal mode. Moving the card is prohibited as there is no programmatic way to distinguish between a touch or click

which is meant to write on the card and one that shall move it.

The blue button with either a pen or a cross symbol, that is only displayed in handwriting mode, can be used to switch between writing and erasing text.

A second tap in the corner or a double-click sets the card back to normal mode and the recognition process is started. At the moment, the recognized handwriting is displayed as computer text in the description part of the card. For the future, a multi-side card is planned that acts like the front side/backside of a paper card but is not limited to just two sides. This would provide space for additional content like pictures or web pages to be displayed without taking over too much space on the card as well as it would divide between handwritten and typed text.

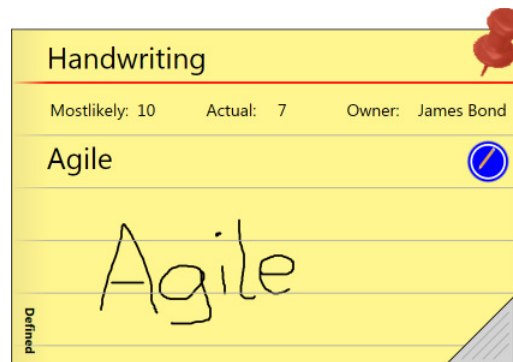


Figure 21: Story card in handwriting mode

In order to fulfill the requirement of unrestrictedly orientable content, the user can move and turn the card with one finger (or using the mouse) and the card will follow the movement of that finger in a very realistic way. This is achieved by using an adjusted version of the RNT algorithm developed by Kruger et al. [19].

Following Kruger's findings, an area in the center of the card, called translate-only region, is reserved for movements which shall not rotate the card. A semi-transparent grey circle appears when a user touches the card which highlights this area.

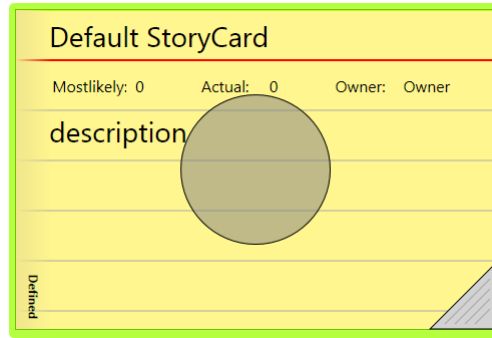


Figure 22: Story card with highlighted translate-only-area

To make interaction with the table feel even more realistic and to provide a possibility to bridge the distance between far ends of the table cards can be tossed across the surface by simply moving them fast enough. They slow down with growing distance as would their paper counterparts. The tossing algorithm also integrates RNT so that the cards turn realistically when given enough spin.

5.3.2 Iterations

Iterations are containers that serve as a means of visually grouping the stories that belong to one development cycle. Iterations display basic information of their respective cycle such as start and end date.

In addition they allow for keeping track of efforts involved in working on their user stories. The top left corner value is called Available Effort and refers to the total available work hours of an iteration (e.g. 3 developers working 40 work hours per week for the 2 weeks duration of an iteration = 240 hours of Available Effort). The top right corner is used to display the Remaining Effort which is calculated as the Available Effort minus the sum of the estimated efforts of all story cards in that iteration.

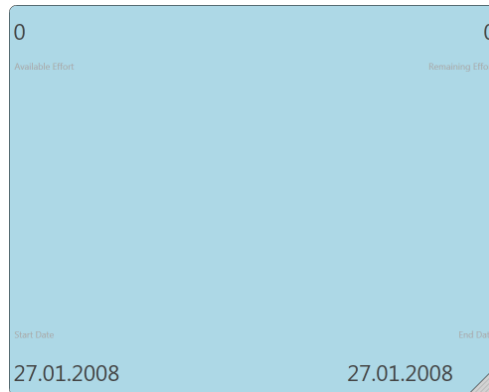


Figure 23: Empty Iteration

In contrast to story cards, the main purpose of iterations is not to present content in the most convenient way to the user but to group the information on the story cards in an easy-to-understand way. Thus, iterations can be moved and resized (using the same resizer-handle as the story cards) but they cannot be rotated or tossed.

In order to be able to contain multiple stories iterations take up a considerable space on the screen because they are, by their nature, much larger than story cards. Rotation seems to be counterproductive when it comes to fulfilling the purpose of grouping. The tossing feature is simply not necessary due to the size of the iteration container, which can usually be reached conveniently from any position around the table.

To provide easy handling of the grouped cards, moving an iteration will automatically move all of its story cards.

5.3.3 Floating Toolbox

The Floating Toolbox is designed as a substitute for the standard menu bar known from desktop applications. It is used for core control of the application such as selecting a project to load or changing the color of a card.



Figure 24: Floating Toolbox

Like story cards, the toolbox can be rotated unrestrictedly as well as it can be tossed across the surface. The translate-only region in its center allows for the same non-rotating movements as the one on the story cards does.

When the toolbox is not used it becomes transparent with the intention of removing it from the focus of attention so that users can concentrate on the planning part of the application instead.

The 'Join meeting' button of the toolbox opens a dialog where the user can add the contact information of the server to be connected to. Choosing 'localhost' switches to a local file-based persistence.

After a successful connection the dialog presents a list of available projects from which the user can pick one. If no project exists, a default project is created and added to the list. If the user wants to create a new project with a specific name he just has to enter the name in the box that is used to display the available projects' names and press the 'Join meeting' button of the dialog. The project will be automatically created and loaded.

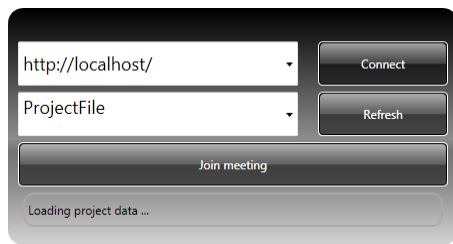


Figure 25: Join meeting dialog

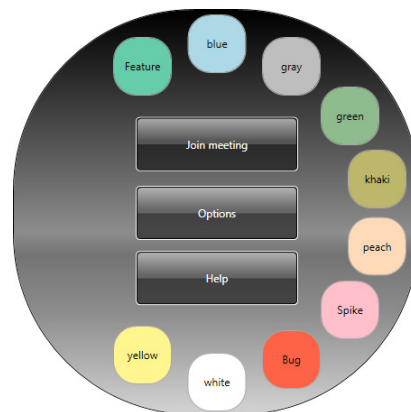


Figure 26: Colours assigned to specific tasks

Pressing the ‘Help’ button opens a dialog with an overview of available gestures and voice commands. The ‘Options’ button is functionless at the moment.

The toolbox is designed to look somewhat similar to a painter’s palette. Clicking one of the coloured circular areas, which are aligned around the buttons in the center, changes the colour of the currently selected story card which is highlighted by a neon-green border around the card. Changing the colour of an iteration is not possible.

Labels on the coloured areas can be used to indicate a relation between a colour and a specific type of task (e.g. ‘Bugfix’ can be used as a label for the colour red). As of now, the labels cannot be edited from within APDT but the projects can be loaded in DAP which already includes the functionality to change the inscription of the labels.

5.4 Command History

Being able to reverse the last couple of actions is a comfort feature that has become so common in desktop applications that users only notice it if it is not available.

Although this undo/redo functionality was not part of the original design the demos and evaluations conducted at the EBE group showed that it is of utmost importance to have such a feature at hand.

An accident in one of these planning meetings, where an iteration was unintentionally deleted along with a whole lot of user stories, made it clear that at least the deletion feature needs some kind of safety precaution to prohibit such accidents in the future.

Confirmation dialogs, as known from desktop applications, have a very negative influence on the workflow. The dialog boxes are popping up at random places on the screen and a connection between the action and the message is not immediately recognizable. Due to the sheer size of the screen, these messages often go unnoticed until the user recognizes that the application is stuck and waiting for a confirmation.

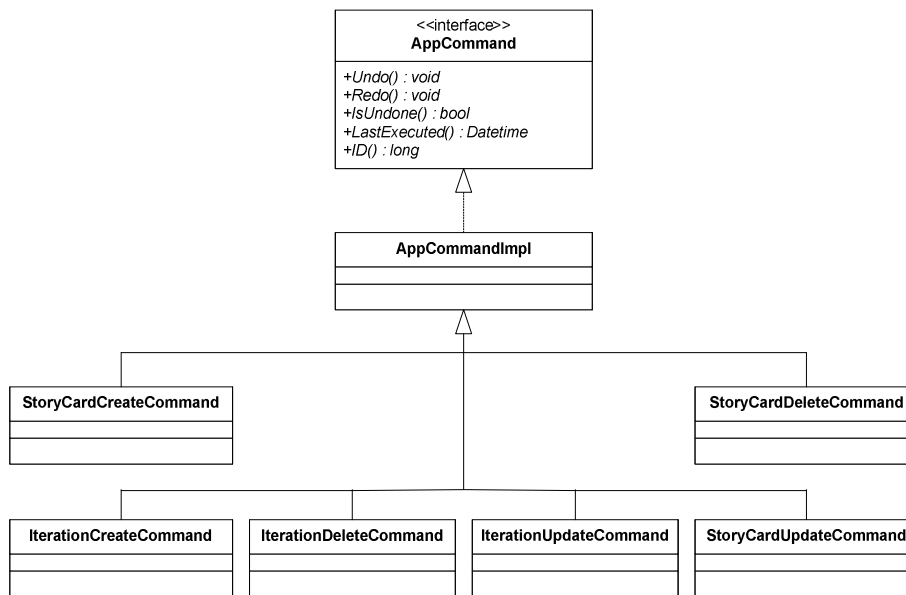


Figure 27: UML-diagram AgilePlannerDT.Application.Commands

Due to time constraints this feature is only partially implemented and only works 100% for reversing the deletion of iterations and the creation and deletion of story cards.

5.5 Persistence Layer

As described in chapter 4.1.3 the `Persister` interfaces add an additional layer of abstraction that hides the underlying implementation of the communication and persistence mechanisms. There are several implementations of the asynchronous interface that can be used interchangeably without any effects on the application. One implementation of the synchronous interface is used for local file-based persistence.

5.5.1 Synchronous Persistence

In the current design, the application relies on the availability of a web service which is supposed to store the planning data. However, it is important to break this dependency down to provide the service of this application at all times. The implementation of the `SynchronousPersister` interface in combination with the `AsyncWrapperPersister` is intentionally written to find a way around this problem by storing data in a local XML-file. As most features of APDT require the presence of a `Persister` implementation it also provides a fast way to test changes to the UI without the need of a working network connection.

During the work on a full two-way communication with the DAP clients, the changes including XML-serialization of the planning objects proved to be of an immense value. This serialization is also used in the prototype implementation of the `AsyncSocketPersister` to exchange text-messages in an XML format that both Java and C# clients can understand.

Chapter 5.5.2.4 describes the XML-serialization in detail.

5.5.2 Asynchronous Persistence

5.5.2.1 Abstract Base Class

The `AbstractAsyncPersister` serves as an abstract base class for all implementations of the asynchronous interface. It provides basic services like the registration and deregistration of delegates for event notification and the declaration of the events that might occur as a reaction to calls to the persistence API.

5.5.2.2 Web Service Based Persistence

The `AsyncWebServicePersister` is the first implementation of the `AsynchronousPersister` interface which was built for APDT. It uses the DAP backend for storing and retrieving planning data.

To connect the C# application to the Java server, a JAX-WS web service is implemented that provides an additional communication channel to the persistence backend. The service publishes all functionality that is described in the asynchronous persistence interface as well as some methods which are tailored for the use of supplemental input devices like Tablet or Pocket PCs.

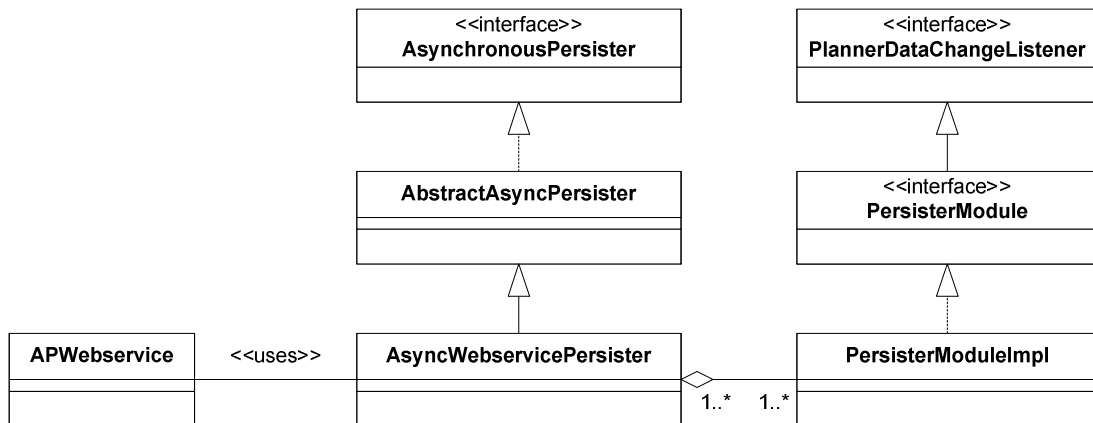


Figure 28: UML-diagram of AsyncWebservicePersister

The web service combines programs written in two different programming languages (Java and C#) using two different IDEs (Visual Studio and Eclipse). This combination is rather difficult to handle during debugging of the application. As Visual Studio is not capable to hook up to a Java Virtual Machine (JVM) for debugging and in turn Eclipse cannot be used to debug C# code both environments have to be used simultaneously to identify the root of occurring problems.

The Java part of the web service uses a minimalist web server that is shipped with the Java Standard Edition Development Kit 1.6 (JDK 1.6). The deployment of the service is very simple compared to a full-fledged J2EE Application server.

```

Endpoint endpoint =
Endpoint.publish("http://localhost:1234/service", new APWebservice(host, port));
  
```

This code sample is all it takes to publish a service that is available at the URL <http://localhost:1234/service/APWebservice>. In order to make a web service publicly available on the Internet either the server's fully qualified name or its IP address must be used. A service published at the 'localhost' address will only be reachable from the same machine. The web service class itself is a 'Plain old Java object' (POJO) with a set of annotations to enable the special web service features.

```
@WebService
@SOAPBinding(style = SOAPBinding.Style.RPC)
public class APWebservice
{
    @WebMethod
    @WebResult
    public WsProject load(@WebParam(name = "projectName")
        String projectName)
    {...}
}
```

The `@WebService` annotation indicates that this class should be deployed as a web service. `@SOAPBinding` determines which method should be used to communicate with the web service. Methods annotated with `@WebMethod` are published when the service is deployed; `@WebResult` indicates that this method has a return value. `@WebParam` can be used to further refine the information available about method parameters.

While these features are very well documented and examples can be found on the web, documentation on exception handling for web services is very poor. Intense research was necessary to figure out how the Java methods can throw exceptions as a reaction on certain undesired events e.g. the deletion of a non-existing story card.

In order to write an exception that is throwable by a web service method the exception class needs to be annotated as shown in the following listing.

```
@WebFault(name="WsIndexCardNotFoundException",
    faultBean="persister.webservice.WsIndexCardNotFoundExpectionBean")
public class WsIndexCardNotFoundExpection extends Exception {
    public WsIndexCardNotFoundExpection()
    {...}
}
```

The `@WebFault` annotation is used to define the name of the exception, while the 'faultBean'-parameter identifies the bean class (using its fully qualified name) which is necessary for the web service to use the exception. If this parameter is not specified or the bean is not present, the deployment of the web service will fail with an error message although compilation passes without errors.

Another requirement is the explicit implementation of a parameterless constructor.

The implicit default constructor is not sufficient at this point.

The formerly mentioned bean class is very simple in its structure, yet necessary.

The generation of the web service client is conveniently integrated into Visual Studio. It provides the stubs necessary to make calls to the web service along with a complete set of methods and events to provide asynchronous communication. As APDT heavily relies on fluent user interaction, asynchronous calls to the service which do not block the application are an explicit requirement. For each method published by the web service, Visual Studio generates two methods for asynchronous calls that only differ by an additional parameter. This parameter, called 'userState', can be used to identify responses which arrive out-of-order. If the application makes another call to the service before the response of a prior request arrived the use of the stateless method raises a runtime exception.

5.5.2.3 Adaption from Synchronous to Asynchronous Persistence

The `AsyncWrapperPersister` adds a thin wrapper layer around any implementation of the `SynchronousPersister` interface and makes its functionality available to the application which does only deal with asynchronous persistence calls itself.

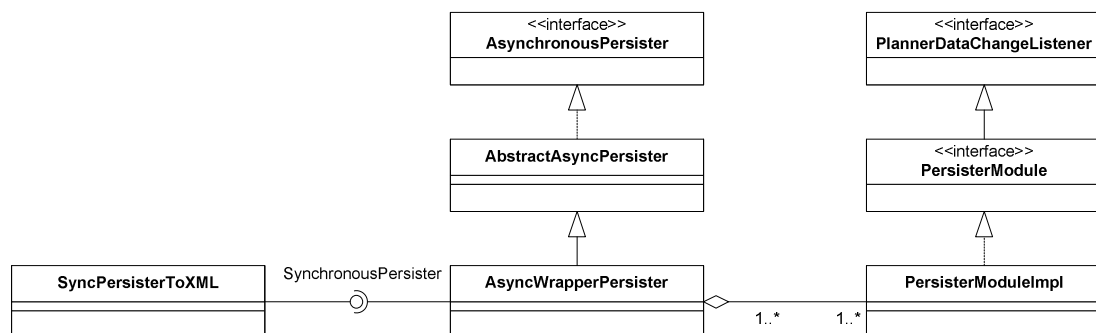


Figure 29: UML-diagram of AsyncWrapperPersister

This component offers a convenient method to re-use any `SynchronousPersister` without the need to rewrite the same functionality for asynchronous calls.

5.5.2.4 Two-way Communication with DAP

As the scope of this work does not include distributed aspects of Agile Planning, yet the topic seems too important not to address it at all, a prototypical implementation of a two-way communication between APDT and DAP is presented here. This prototype allows the exchange of text-messages in an XML-format over socket connections that both APDT and DAP can interpret and use as a way of communicating with each other. It should be considered as a technology outlook which already incorporates much of the functionality that is necessary to couple APDT and DAP in a heterogeneous distributed environment.

While APDT uses the XML-serialization functionality that is built into the .NET Framework, the Java side uses the XStream library [7] to achieve the same goal.

.NET's default serialization mechanism is very straightforward to use and only requires a minimum effort by the developer. The drawback of this mechanism is that it does not work properly when development follows the best-practice of defining interfaces and code against those interfaces instead of the actual implementation classes.

Together with the need to match the XML-format of DAP, it is easier to provide a means to convert objects to XML manually. The `IXmlSerializable` interface defines a set of methods that the .NET Framework uses for this conversion. The interfaces that are defined for the objects of interest extend the `IXmlSerializable` interface and the implementation classes have to take care of the proper serialization.

```
public XmlSchema GetSchema();

public void ReadXml(XmlReader reader);

public void WriteXml(XmlWriter writer);
```

With the data objects implementing the interface shown in the previous listing, it is not difficult to read data from a file or to store the information of these objects inside a file.

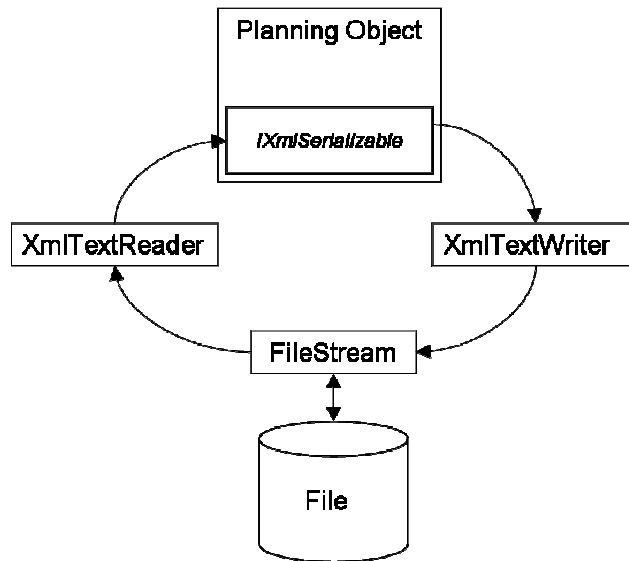


Figure 30: Overview diagram XML serialization in .NET

XStream takes a slightly different approach to achieve the same goal. One way to enable the serialization of objects is to annotate the objects while the other is to provide custom converters for them.

Serialization of certain objects which need to be transferred between APDT and DAP is hard to realize using annotations due to their complex data structure. To avoid this problem, converters for all objects of interest are provided.

The `Converter` interface defines a set of methods similar to those of .NET.

```

public void marshal(Object value, HierarchicalStreamWriter writer,
    MarshallingContext context);

public Object unmarshal(HierarchicalStreamReader reader,
    UnmarshallingContext context);

public boolean canConvert(Class clazz);
  
```

As with .NET, the serialization itself is quite convenient. The converter needs to be registered in order to make its functionality available. After that, the serialization process is completely transparent for the developer.

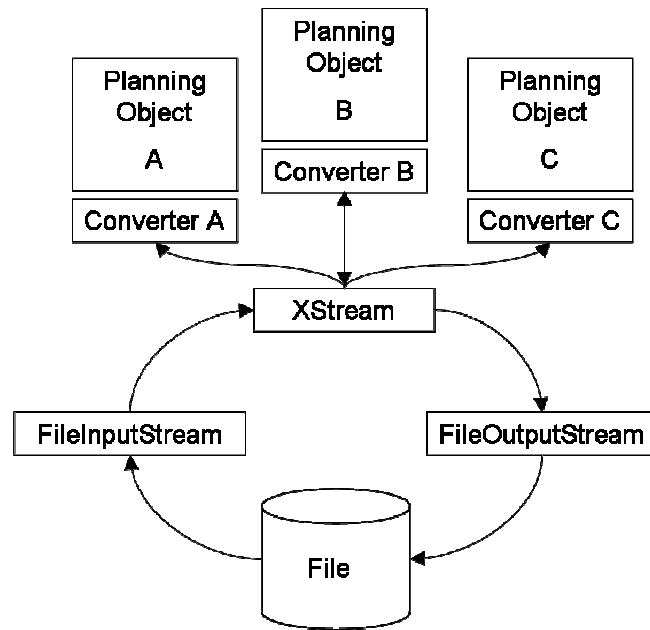


Figure 31: Overview diagram XML-serialization with XStream

As of now, messages on both ends can be converted into an identical format, which makes communication between clients written in different programming languages possible. In addition, these conversions can be used to store the application data in format-compatible XML-files in both languages. Currently, DAP uses a DOM-based implementation to store its data into XML-files. This will be replaced with a solution based on this prototype in the near future to make the maintenance of those worker classes easier.

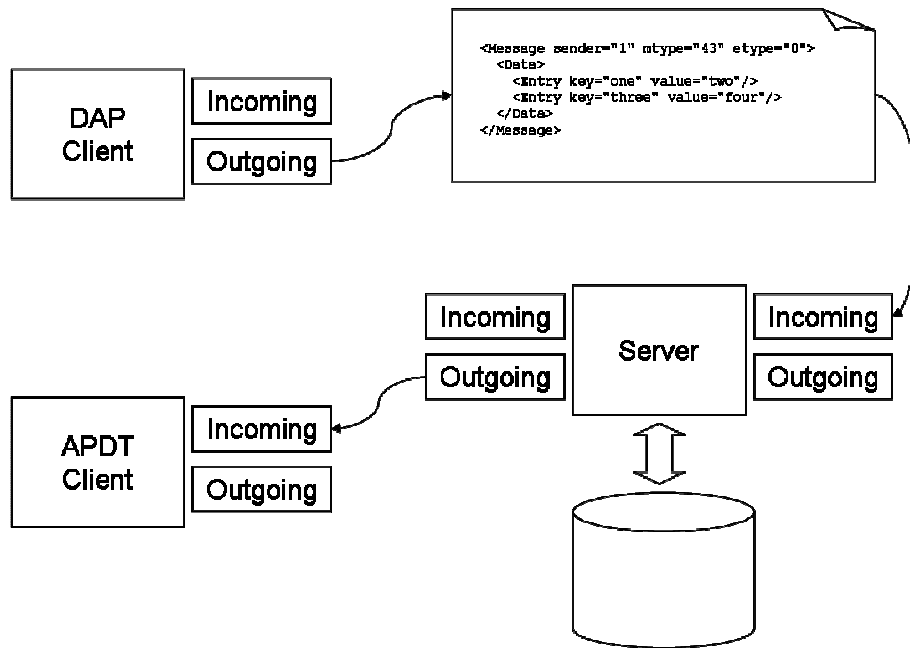


Figure 32: Overview diagram of communication between APDT and DAP

The communication infrastructure uses a standard client-server-architecture. The server is implemented in Java, clients can basically be written in any language that supports the transmission of UTF-encoded text via sockets.

Clients and server are highly decoupled in order to allow for non-blocking message-transmission. DAP and APDT clients have separate connection threads for incoming and outgoing messages. The server has one pair of threads for each client which handles the communication with that client. Dead connections are automatically recognized and removed.

The messages can be either UI related, such as updates for the telepointers that DAP offers as a remote-awareness feature, or data related, e.g. content updates for story cards.

Data updates are transparently persisted by the server. The current implementation stores the data in an XML-file with the same format that APDT and DAP clients use. UI updates are transient and will be lost if client or server go offline.

6 Conclusion and Future Work

6.1 Problems

The goal of this thesis was to bring an existing analogous process to a digital environment. On one hand, it was a requirement that the tool should not force the users to change their behaviour while on the other hand it was supposed to improve the original process and contribute new features which might help teams in conducting Agile Planning Meetings.

The first problem that needed to be solved was the choice of a programming language and proper graphics framework for the implementation. The framework had to provide the following functions:

- Unrestrictedly reorient content
- Hand over any artefact on the table in order to bridge the distance between collaborators
- Integrate handwriting recognition

While it had to be high-performance enough to allow for fluent user interaction on a screen with a resolution that is several times higher than that of a standard computer display.

It would be beneficial if the development environment programming language offers additional features like garbage collection to save time in the development process.

The iLab's Large Display Framework looks very promising at first glance but several weaknesses make it unsuitable for this thesis work.

First, there is no easy and convenient way to access the framework from .NET. The implementation of a wrapper layer is troublesome and error-prone. Second, the need to implement additional functionality in native C++ largely increases the complexity of the development tasks because most of them would require using two different programming languages with very different syntax, rules and limitations. Third, the absence of built-in text widgets like text boxes is a huge counter-argument for the framework's integration as the basis for an application which heavily relies on displaying text to share information. Last, the framework's performance does not

seem to scale well enough for the use with a tabletop display with such a high resolution.

The evaluation of the LDF was quite time consuming and the implementation of a somewhat working prototype cost several weeks with no real payback in the end.

The Windows Presentation Foundation met the above criteria much better and even the unfamiliarity with the new technologies did not have a too negative effect on the speed of the development. Although the use of C# limits the application to the Windows platform, this trade-off was willingly accepted for the additional comfort of the .NET Framework.

After the initial evaluation, the actual implementation work had to cope with an ever recurring problem which originates from the hardware-platform of the SMART table. The imprecision of the DVIT input device has a tremendous impact on the design of the graphical interface. In order to accomplish a convenient interaction with the planning artefacts, the touch sensitive areas have to be much larger than originally planned.

The fact that the DVIT also recognizes objects in close proximity to the screen and not only actual touch contacts with the surface, sometimes results in undesired behaviour when moving objects.

In addition, the graphics cards which drive the table are not designed for high-performance hardware-acceleration. Instead, Matrox puts more emphasis on the above-average number of supported displays per card. Thus, a lag in the visualization becomes noticeable when the number of displayed objects exceeds a certain threshold.

6.2 Thesis Contributions

The majority of the requirements identified by [21] are either fully or partially satisfied by APDT. The only two criteria not addressed at all are related to team authentication (which was not considered crucial for a research environment) and the use of physical planning objects (which would contradict the purpose of making such items obsolete by using APDT). Simultaneous interaction in the planning process is limited by the hardware platform that is used for the implementation.

Criteria	Criteria Sub Categories	APDT
Agile planning objects creation and editing	Functional Requirements	F
Visual characteristics for different types of stories		F
Agile planning metrics management		F
Planning for multiple iterations		F
Systematic organization of planning objects		F
Fluid transition between plan changes and consequent results		F
Concept of team and identity authentication		N
Real-time exposure of the plan via the Internet		F
Simultaneous planning information organization	Interaction-related Requirements	P
Simultaneous planning information editing		P
Story editing with handwriting		F
Handwriting recognition capability		F
Fluid transition individual and collaborative work in agile planning		P
Fluid transition between agile tabletop collaboration and external work		F
Flexible user arrangement		F
Shared objects to digital objects in agile planning		F
Use of physical planning objects		N

Table 4: Criteria assessment of APDT according to [21]

F Fully satisfied **P** Partially satisfied **N** Not supported

Of the set of requirements that are listed in [38] only those related to Agile Planning are part of the topic of this thesis. The Groupware related criteria which apply to distributed settings are not relevant in this context.

Criteria	Criteria Sub Categories	APDT
Creating/editing/deleting artefacts	Agile Planning Related	F
Planning metrics managements		F
Planning for multiple iterations		F
Moving stories between iterations		F
Team and identity authentication		N
Real-time exposure via Internet		F
Visual characteristics for story cards		F
Integration with the development environment		N

Table 5: Criteria assessment of APDT according to [38]

F Fully satisfied **P** Partially satisfied **N** Not supported

Again, authentication is not considered a crucial feature in the target environment and as APDT is designed as a stand-alone application for Tabletop Devices, the integration into any kind of IDE is not part of the thesis' topic.

Although no formal evaluation of APDT's usefulness and usability was made, informal evaluations conducted at the e-Business Engineering Group indicate that the utilization of APDT offers a perceived benefit to the Agile Planning process. Users experienced the UI as intuitive and easy to understand. The realistic behaviour of the electronic planning artefacts mimics that of their paper counterparts with great success and offers a natural way to interact with the application.

The various input methods allow the users to find the most effective or convenient way to accomplish a task. The planning data is presented in a way that users are familiar with and can be stored and shared in electronic form, without the need of transcribing it. The automated persistence of planning data and its easy distribution are some of the features that only a computer environment can provide.

Although several features of APDT only have demo or prototypical characteristics, even those proves of concept seem to be beneficial to the process of Agile Planning. During the planning meetings conducted at the EBE as well as in several demos given for sponsors and colleagues, the intuitive use of the UI became apparent. People got used to the gestures remarkably fast by just watching them once or twice even without any formal instruction.

Orientating and tossing cards on the table surface seems to be so close to the natural way of handling paper index-cards that the users 'just do so' without really giving the process a second thought.

Another feature of great interest is the voice-command control of APDT. In its current state of implementation it has to be considered as a technology outlook with a high potential to be an important alternative to interact with Tabletop Devices.

6.3 Future Work

A definite shortcoming of APDT is the lack of multi-touch support. In every planning meeting or demo, users tried to interact simultaneously with the application either by accident or on purpose. That was the case even when they were warned that concurrent touch events are not supported and might cause unexpected behaviour. In addition, the support of multiple touch points would enable the use of multi-touch

gestures like the resizing gesture that Apple implemented for its iPhones [5].

Although the SMART drivers provide some functions to identify two different touch points, the support is very limited and not well integrated with C# (or any other programming language) and the extra effort did not seem to pay off for the loss of flexibility which would result from tying the application so close to SMART's infrastructure.

With the presentation of Surface Computing to a public audience [32], Microsoft introduced a hardware and software platform which was designed with hindsight to that issue. According to its specifications [56], the Surface supports up to 52 simultaneous touch points which should be even enough for larger teams.

It seems reasonable to presume that other vendors will adapt the framework or at least provide compatible interfaces for use with their hardware as soon as Microsoft finally releases the Surface platform.

In contrast to the tactile interaction with the table, the speech recognition was not evaluated due to time constraints and problems with the setup of multiple input devices, namely headsets or microphones, on the development machines. With the availability of wireless headsets, well known as accessories for cell-phones, there should be no need to use wired headsets which also tend to block out the voices of other team members and thus interfere with the communication which is crucial to conduct a planning meeting.

In the future, the use of voice commands can be extended to support dictation of content without the need to interact physically with the table. The Microsoft Speech framework includes such text dictation functionality which could not be evaluated for this thesis.

With the use of more complex grammars [33] instead of a set of immutable commands it should be possible to integrate 'natural speech' to trigger specific actions.

As the research of Morgan [38] pointed out, distributed planning meetings have become increasingly popular, even in agile environments that highly rely on face-to-face communication and human-to-human interaction. This tendency is likely to increase in the future as more and more companies outsource their development efforts.

Digital Tabletop Devices are still rare and expensive. Thus, the chance of a development team having more than one table at its disposal is rather low. A way to allow for distributed planning meetings would be to fully integrate APDT with the Java based DAP. Although both applications use a compatible XML format to store their data and the web service implementation of the Persister interface in APDT already uses the DAP backend, there is no fully integrated two-way communication between those different clients yet. A working prototype implementation that exchanges XML-messages over socket connections exists but it still needs to be integrated with the DAP backend.

Issues of this topic which need to be addressed are mostly related to the different coordinate systems of GEF and WPF. A proper translation needs to be implemented, in order to make planning artefacts appear at the same positions on both types of clients (table and desktop computer). The size of the table display makes it difficult to keep an overview of the whole project on the DAP clients. One solution to this problem may be some kind of 'radar' or 'overview map' that shows all cards in a very scaled-down size in the sidebar.

DAP offers several remote-awareness features that proved to be beneficial to users and should be considered for a distributed setting. These features include telepointers which display the position of the mouse cursors of remote clients in the workspace and live-updates that instantaneously show text updates to story cards and iterations. Extending these live-updates to handwriting input might also be a topic for future work.

Many of the ideas and approaches presented in this thesis were not formally evaluated. This evaluation is important to find out what users really miss and what features are crucial for the application in order to contribute to the productivity of agile teams.

References

- [1] Agile Alliance; Homepage (Online <http://www.agilealliance.com>); Last visited February 15th 2008
- [2] Agile Alliance; Manifesto for Agile Software Development (Online <http://www.agilemanifesto.org>); Last visited February 4th 2008
- [3] Agile Alliance; Principles behind the Agile Manifesto (Online <http://www.agilemanifesto.org/principles.html>); Last visited February 12th 2008
- [4] Anastácio, Fabrício; UofC Large Display Framework, Presentation given at the UofC Interactions Lab; 2007
- [5] Apple; iPhone (Online <http://www.apple.com/iphone>); Last visited February 3rd 2008
- [6] CardMeeting; CardMeeting (Online <http://www.cardmeeting.com>); Last visited February 11th 2008
- [7] Codehaus; XStream (Online <http://xstream.codehaus.org>); Last visited January 31st 2008
- [8] Danube; ScrumWorks (Online <http://www.danube.com/scrumworks/basic>); Last visited February 11th 2008
- [9] Firefox; Mouse Gestures Add-on (Online <https://addons.mozilla.org/de/firefox/addon/39>); Last visited February 3rd 2008
- [10] Fox, Armando et al.; Integrating Information Appliances into an Interactive Workspace, Computer Graphics and Applications, IEEE, pp. 54-65; 2000
- [11] Ghanam, Yaser et al.; APDT: An Agile Planning Tool for Digital Tabletops; 2008
- [12] GLUT OpenGL Utility Toolkit; Homepage (Online http://www.opengl.org/resources/libraries/glut/glut_downloads.php); Last visited January 30th 2008
- [13] Hancock, Mark et al.; Rotation and Translation Mechanisms for Tabletop Interaction; Proceedings of IEEE International Workshop on Horizontal

- Interactive Human-Computer Systems (TableTop2006), IEEE Computer Society, pp. 79-86; 2006
- [14] Hömig, Sascha; Diplomarbeit: Feasibility Study and Prototypical Implementation of a Window Framework for Digital Tables; Department of Computer Science, Hochschule Mannheim; 2007
- [15] Interactions Lab; Homepage (Online <http://grouplab.cpsc.ucalgary.ca>); Last visited 30th January 2008
- [16] Jacob, Robert J.K.; Integrality and Separability of Input Devices, ACM Transactions on Computer-Human Interaction (TOCHI), ACM Press, pp.3-26; 1994
- [17] Kolenda, Henning; M.Sc. Thesis: Analysis, Design and Implementation of a Framework for Digital Desk Applications, Department of Computer Science, Hochschule Mannheim; 2007
- [18] Kruger, Russel; M.Sc. Thesis: Fluid Orientation on Tabletop Displays, Department of Computer Science, University of Calgary; 2004
- [19] Kruger, Russell et al.; Fluid Orientation on a Tabletop Display: Integrating Rotation and Translation; 2005
- [20] Larman, Craig; Agile and Iterative Development: A manager's Guide; Addison-Wesley; 2004
- [21] Liu, Lawrence; M.Sc. Thesis: An Environment for Collaborative Agile Planning, Department of Computer Science, University of Calgary; 2005
- [22] Matrox; Matrox QID Pro graphics cards (Online <http://www.matrox.com/graphics/en/pid/products/qid/qidpro.php>); Last visited January 30th 2008
- [23] Maurer, Frank; Supporting Distributed Extreme Programming; Proceedings of Agile Universe / XP Universe 2002, Springer; 2002
- [24] MbUnit; Generative Unit Test Framework for the .NET Framework (Online <http://www.mbunit.com>); Last visited January 30th 2008

- [25] Microsoft; Animation Overview (Online <http://msdn2.microsoft.com/en-us/library/ms752312.aspx>); Last visited February 1st 2008
- [26] Microsoft; ApplicationGesture Enumeration (Online <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tpcsdk10/lonestar/microsoft.ink/Enumerations/applicationgesture/applicationgesture.asp>); Last visited February 8th 2008
- [27] Microsoft; Ink Analysis and Input Supplement for the Windows XP Tablet PC Edition Development Kit 1.7 (Online <http://www.microsoft.com/downloads/details.aspx?FamilyID=884012e7-dd97-4dbc-bf79-68ce8ee2b590&DisplayLang=en>); Last visited February 8th 2008
- [28] Microsoft; Microsoft Speech (Online <http://www.microsoft.com/speech/speech2007/default.msp>); Last visited February 2nd 2008
- [29] Microsoft; Microsoft Window XP Tablet PC Edition Development Kit 1.7 (Online <http://www.microsoft.com/downloads/details.aspx?FamilyID=69640b5c-0ee9-421e-8d5c-d40debee36c2&DisplayLang=en>); Last visited February 8th 2008
- [30] Microsoft; Microsoft Windows Software Development Kit for Windows Vista and .NET Framework 3.0 Runtime Components (Online <http://www.microsoft.com/downloads/details.aspx?familyid=C2B1E300-F358-4523-B479-F53D234CDCCF&displaylang=en>); Last visited January 30th 2008
- [31] Microsoft; Microsoft Windows XP Tablet PC Edition 2005 Recognizer Pack (Online <http://www.microsoft.com/downloads/details.aspx?FamilyID=080184dd-5e92-4464-b907-10762e9f918b&DisplayLang=en>); Last visited February 8th 2008
- [32] Microsoft; Surface (Online <http://www.microsoft.com/surface>); Last visited January 31st 2008
- [33] Microsoft; Using Grammar Elements to Recognize Speech (Online <http://msdn2.microsoft.com/en-us/library/ms873284.aspx>); Last visited February 4th 2008

- [34] Microsoft; Visual Studio 2005 extensions for .NET Framework3.0 (WCT & WPF), November 2006 CTP (Online <http://www.microsoft.com/downloads/details.aspx?FamilyId=F54F5537-CC86-4BF5-AE44-F5A1E805680D&displaylang=en>); Last visited January 30th 2008
- [35] Microsoft; Windows Speech Recognition (Online <http://www.microsoft.com/enable/products/windowsvista/speech.aspx>); Last visited January 31st 2008
- [36] Morgan, Robert et al.; MasePlanner: A Card-Based Distributed Planning Tool for Agile Teams; 2006
- [37] Morgan, Robert et al.; Using Horizontal Displays for Distributed and Co-located Agile Planning; 2007
- [38] Morgan, Robert; M.Sc. Thesis: Distributed AgilePlanner: A Card Based Planning Environment for Agile Teams, Department of Computer Science, University of Calgary; 2007
- [39] Nuance; Dragon Naturally Speaking (Online <http://www.nuance.com/naturallyspeaking>); Last visited February 2nd 2008
- [40] OpenGL; Homepage (Online <http://www.opengl.org>); Last visited February 2nd 2008
- [41] Opera; Homepage (Online <http://www.opera.com/products/desktop>); Last visited February 3rd 2008
- [42] Park, Shelly; M.Sc. Thesis: A Daily Scrum Meeting Summarizer for Agile Development Teams, Department of Computer Science, University of Calgary; 2007
- [43] Rally Software; Rally's Agile Lifecycle Management Solutions (Online <http://www.rallydev.com/products.jsp>); Last visited February 11th 2008
- [44] Scott, Stacey D. et al.; Design Guidelines for Co-located, Collaborative Work on a Tabletop Display, Proceedings of European Conference of Computer-supported Cooperative Work, Kluwer Academic Publisher, pp. 159-178; 2003

- [45] Scott, Stacey et al.; Storage Bins: Mobile Storage for Collaborative Tabletop Displays; Computer Graphics and Applications, IEEE, pp. 58-65; 2005
- [46] SMART Tech; Digital Vision Touch Technology White Paper (Online http://smarttech.com/dvit/DViT_white_paper.pdf); Last visited January 30th 2008
- [47] SMART Tech; Homepage (Online <http://smarttech.com/>); Last visited January 30th 2008
- [48] Stewart, Jason et al.; Single Display Groupware: A Model for Co-present Collaboration; Proceedings of ACM Conference on Human Factors in Computing Systems, ACM Press, pp. 283-293; 1999
- [49] Swigart, Scott; Simple Gesture Recognizer – SIGER (Online <http://msdn2.microsoft.com/en-us/library/aa480673.aspx>); Last visited February 4th 2008
- [50] Tang, John C.; Findings From Observational Studies of Collaborative Work; International Journal of Man-Machine Studies 34(2); 1991
- [51] Test-driven Development; Wikipedia (Online http://en.wikipedia.org/w/index.php?title=Test-driven_development&oldid=187777541); Last visited January 31st 2008
- [52] TestDriven.NET; Homepage (Online <http://www.testdriven.net/>); Last visited January 30th 2008
- [53] Trolltech; Qt: Cross Platform Rich Client Development Framework (Online <http://trolltech.com/products/qt>); Last visited February 2nd 2008
- [54] VersionOne; Agile Project Management Tools (Online <http://www.versionone.com/products.asp>); Last visited February 11th 2008
- [55] Weigelt, Roland; GhostDoc (Online <http://roland-weigelt.de/ghostdoc>); Last visited January 30th 2008
- [56] Wikipedia; Microsoft Surface (Online http://en.wikipedia.org/w/index.php?title=Microsoft_Surface&oldid=182173015); Last visited January 31st 2008

- [57] Wikipedia; Speech Recognition Grammar Specification (Online http://en.wikipedia.org/w/index.php?title=Speech_Recognition_Grammar_Specification&oldid=183206220); Last visited February 10th 2008
- [58] Wikipedia; XAML (Online <http://de.wikipedia.org/w/index.php?title=XAML&oldid=41251720>); Last visited January 30th 2008
- [59] XPlanner; XPlanner Overview (Online <http://www.xplanner.org/index.html>); Last visited February 11th 2008