Leveraging the Jazz Platform for Developing an Agile Planning Tool

Kai Nehring, Shelly Park, Frank Maurer
University of Calgary
Department of Computer Science
2500 University Dr. NW, Calgary, AB, Canada
(403) 220-3531

Nehring.kai@gmail.com, {parksh, maurer}@cpsc.ucalgary.ca

ABSTRACT

Tools often need to be integrated in and evaluated within a whole development process. Research ideas often impact only a small part of this process, but the impact of the idea can realistically only be determined when it is integrated properly in the whole process. This implies that if the new tool is not integrated with other tools, its usefulness for the practitioner is often limited. Researchers need to be creative, but also we need a framework that has enough penetration in the market that learning the technology will pay off in the long run for the students. In this paper, we describe our experience with integrating our Agile planning tool with Jazz platform. Our experience shows that we were able to save a lot of development time, but faced several obstacles as well.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments – Integrated environments, Interactive environments, Programmer workbench; D.2.9 [**Software Engineering]** Management – Programming teams, Software configuration management

General Terms

Management, Design, Human Factors

Keywords

Agile software engineering, collaborative programming, Jazz, Eclipse, IDE, integrated development environment

1. INTRODUCTION

Tools developed in research settings are designed to facilitate experiments, perform user studies, test the efficiency of new algorithms, enhance our understanding of the software development endeavor and help further the collaboration between researchers and industry practitioners. Therefore, even more than industry settings, rapid tool development is a necessity of research work in order to facilitate the research within the given amount of time and budget. However, tool development in a research setting faces a high developer turnover rate, high learning curve for new student developers and an unpredictable project outcome due to the experimental nature of research projects. One way to develop software tools quickly is to leverage the existing functionalities in other systems by integrating different software together rather than developing everything from scratch. However, most third-party software is difficult to extend because they are not designed to be integrated with other software. Trying to integrate between software that is not originally designed to be extendible is very difficult and the end-product may become very clunky and prone to failures due to the awkward interfacing. Even if the source code is available, extending and combining several different software products for the purpose of developing new software can be very difficult, because the existing code is too tightly coupled with the original software design rather than allowing for the flexibility of what the software may become.

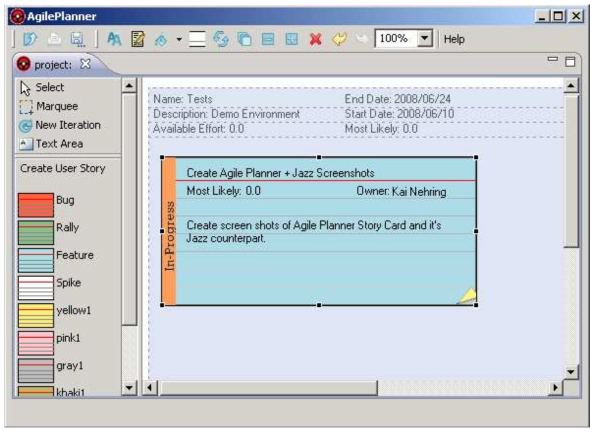
Recently, development platforms such as Eclipse [1] have been very successful, because of its extensibility and its plugin architecture. Jazz [2] extends Eclipse development environment by providing an additional set of collaboration functionalities for software developers. The benefit of building software using the Eclipse and Jazz platforms is that much of the reusable supporting functionalities are already available to the developers and the developers have access to a rich set of functionalities that are tested by many users already. The developers can focus more on developing new functionalities of the software rather than spending unnecessary time building code again that has little research value. Having these platforms can save time and developers are still able to produce very robust software products that can provide rich user experiences.

This is a position paper describing our experience with extending AgilePlanner (based on the Eclipse platform) with collaborative project planning features in Jazz. Agile planning meetings bring developers and customers together to discuss the requirements for a software product and to estimate the tasks for the upcoming iteration. AgilePlanner supports distributed synchronous planning meetings by providing a shared workspace for team members at different locations. We describe how we integrated AgilePlanner with the Jazz platform to take advantage of its advanced progress tracking features. We discuss why the integration with Jazz was important in the research work. However, we also describe some of the difficulties we encountered while we were integrating Jazz with Agile planner and discuss why. Furthermore, we discuss the impact of development platforms, such as Jazz, particularly in research and teaching settings and describe some of the difficulties that we still need to overcome in terms of infrastructure integrations.

2. MOTIVATION

The AgilePlanner project started as a collaborative research project between a university software engineering lab and industry sponsors. The overall goal of the project was to understand the relationship between requirements gathering, task estimation, requirements negotiation, requirements representation and the role of different stakeholders in Agile planning, specifically to understand the role of planning tools. Agile planning is an important step in agile software engineering, because the meeting allows the customer and the developers to discuss the goal of the next iteration and negotiate the deliverables for the upcoming iteration [3]. AgilePlanner is a tool based on a story card metaphor where the functionalities desired by the customer are broken down into user stories and each user story is written on an index card. The index card or story card is used to estimate the resources required to complete the task and the story cards remain as an artifact for tracking the progress of the project.

AgilePlanner was built on top of the Eclipse framework using plugin architecture. This allowed the developers to leverage the features already existing in Eclipse and easily integrate other existing Eclipse plugins with limited effort. Using Java graphics libraries, the developers were able to use the index-card metaphors to simulate physical manipulation of index cards in the virtual world. This enhances the user interaction experience by tapping into the natural human cognition in manipulating physical objects [4]. The user experience is also augmented by allowing the users to easily duplicate digital story cards across many locations in order to carry out synchronous distributed agile planning sessions. The tool allows the users to stack up the digital story cards and flip the virtual story cards to associate acceptance tests with user stories [5]. The digital tabletop version [6] also allows the users to rotate the cards for multiple viewers around the digital tabletop and augment the card manipulation with a voice-recognition capability. AgilePlanner supports distributed planning by providing a shared distributed workspace and allowing team members to interact with each other in real time through telepointers. The tool allows researchers to evaluate the effectiveness of real-time distributed agile planning sessions. Figure 1 shows a screenshot of AgilePlanner.



Name: Tests
Description: Demo Environment
Available Effort: 0.0
End Date: 2008/06/24
Start Date: 2008/06/10
Most Likely: 0.0

Create Agile Planner + Jazz Screenshots
Most Likely: 0.0 Owner: Kai Nehring
Create screen shots of Agile Planner Story Card and it's Jazz counterpart.

Figure 1: Agile Planner displaying a user story

Tools often need to be integrated in and evaluated within a whole development process. Research ideas often impact only a small part of this process, but the impact of the idea can realistically only be determined when it is integrated properly in the whole process. This implies that if the new tool is not integrated with other tools, its usefulness for the practitioner is often limited. As a result, research ideas fail not because they are bad ideas, but because the research team does not have the development power to integrate their ideas into the whole software engineering and tool usage process. For example, being able to print information from the tool, version it and integrate it with existing tools are expected features in project planning tools, but these kind of features are not novel ideas that can be published. Unlike in industry settings where most of the developers' time is devoted to building robust software, the researchers are driven by the pursuit of attaining new scientific knowledge and do not necessarily have the development resources to build a new tool from scratch or have the time to build very large, robust, defect-free software. Therefore, platforms, such as Jazz, can reduce the development effort of these kind of pure development tasks, which are essential to the research project but hold little research value, in order to allow researchers to focus their work on innovation.

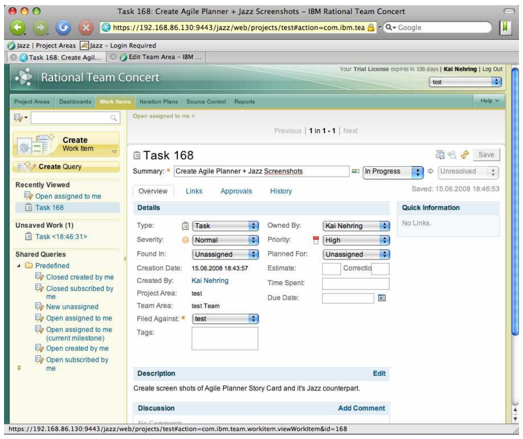
Similarly, AgilePlanner also faced problems of needing many features that hold little research value, but nevertheless required a lot of development work in order to evaluate the tool from the overall project planning process. Essentially, in order to evaluate a small part of the project planning process that was relevant for the research team, the research team was faced with a lot of development tasks. Additionally, the deliverables must be robust enough for user studies in order to obtain credible observational data, but research team had very little resources in terms of testing the software for defects.

The second issue in research settings is the high turnover rate of the developers. The developer turn-over rate is very high in academic settings because the students are only engaged in the project for the duration of their research. Because there is always a constant influx of students coming into the project to develop the software, it is very difficult to transfer the development knowledge to the new incoming students in a very short time frame. It is not uncommon for the new students to rewrite parts of the software again to their liking, which contributes very little in terms of new functionalities, because understanding the code design by the previous students are too difficult. Learning new technologies and frameworks takes time and applying those to a research project is a very difficult task. Therefore, building tools based on well-known technologies where students can easily apply transferable skills is very important in smooth hand-off between students and facilitating collaborative development between students.

AgilePanner lacks progress tracking and reporting capabilities. These are available in the Jazz platform and required by many industry teams. Thus, integrating AgilePlanner with Jazz allows to reuse existing functionality while making a research prototype much more applicable in industrial case studies. Working with Jazz would mean reducing the development time drastically for the student developers as we didn't have to write the functionalities that are already offered by Jazz. In the next section, we describe the implementation details of the integration between AgilePlanner and Jazz.

3. INTEGRATION

AgilePlanner supports synchronous distributed Agile planning, but it was missing project management functionality, task breakdown and project planning storage. We again started to look for third-party software that we could easily extend in order to add project planning and progress tracking aspect into the tool without huge redundant coding work that provides little research value. Integrating AgilePlanner with Jazz made a lot of sense because AgilePlanner and Jazz were both built on Eclipse framework and both were designed to facilitate our understanding of the collaboration in software engineering process. Figure 2 shows the user interface of Jazz planning tool.



Task 168

Summary: * Create Agile Planner + Jazz Screenshots In Progress Unresolved

Saved: 15.06.2008 18:46:53

Details

Type: Task
Severity: Normal
Found In: Unassigned
Creation Date: 15.06.2008 18:43:57
Created By: Kai Nehring
Project Area: test
Team Area: test Team
Filed Against: * test
Tags:

Owned By: Kai Nehring
Priority: High
Planned For: Unassigned
Estimate: Correctio
Time Spent:
Due Date:

Quick Information

No Links.

Description

Create screen shots of Agile Planner Story Card and it's Jazz counterpart.

Discussion

No Comments

https://192.168.86.130:9443/jazz/web/projects/test#action=com.ibm.team.workitem.viewWorkItem&id=168

Figure 2: Jazz project planning page

The core component of Jazz is a small kernel that offers basic functionality required for developing distributed collaborative software development tool, but much of the rich collaboration features are offered through built-in Jazz collaboration plugins module. Jazz offers work item management, project reporting abilities and team communication tools among many others. Jazz also offers a huge number of APIs that allows third-party software developers to extend Jazz.

However, Jazz doesn't offer synchronous distributed planning functionality like AgilePlanner. This meant that many of the artifacts produced or required in AgilePlanner had to rely on asynchronous communication interfaces in Jazz. In order to integrate the synchronous AgilePlanner communication layer into the asynchronous Jazz platform, we decided to build an adapter between Jazz and AgilePlanner that acts as a translator in between. Figure 3 shows the structure of the adapter layer. The Session Manager takes care of the communication between Jazz and AgilePlanner. The Component Manager hides the detailed workings of the adapter from both Jazz and AgilePlanner. The Error Facade catches Jazz specific exceptions and turns them into AgilePlanner specific errors so that the error messages from Jazz are appropriately handled by AgilePlanner.

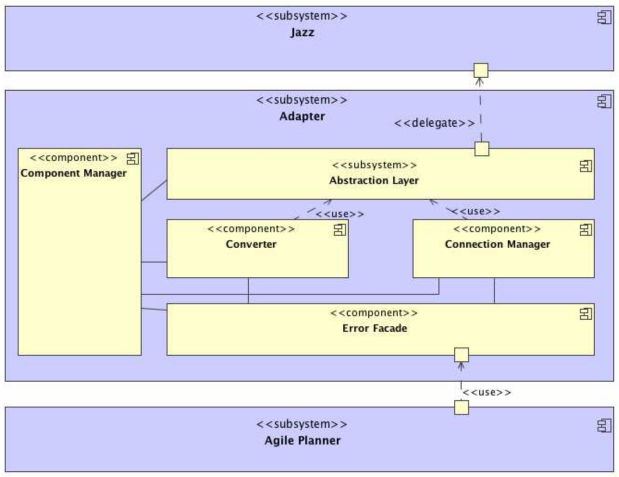


Figure 3: Adapter controls the data synchronization between AgilePlanner and Jazz

We found that using the Jazz API is quite complex. The Jazz API offers the not only the ability to extend Jazz, but also to develop components that are capable of replacing parts of Jazz completely. If developers are not careful, the flexibility of the Jazz interface can also lead to waste of huge amount of time trying to understand Jazz code. Instead, we decided to create an abstraction layer (AL) using Business Delegate design pattern [7]. Using design patterns allows other developers to quickly understand how the abstraction layer is designed and to be able extend the abstraction layer later if necessary. The abstraction layer can be developed and maintained independently as long the developers understand the design of the abstraction layer. Additionally, any sudden behavior changes in subsequent releases of Jazz will be caught by the abstraction layer, making the maintenance and the integration easier by decoupling AgilePlanner from Jazz. The AL is packaged into a single jar file that not only includes AL code, but also the necessary Jazz libraries. This design simplifies the use of Adapter, shown in Figure 3, because any projects that use the Adapter can simply import a single jar file.

4. DISCUSSION

The benefit of using Eclipse and Jazz platform is the easy integration of new functionality. Even if the original platform developers did not imagine how Eclipse and Jazz could be extended by other developers, the amount of APIs that are available through these platforms, in addition to many other plugins, allow the researchers to be as creative as they can without spending huge development hours. The tool developed on Eclipse platform can also run as Eclipse plugin or as an independently software, so the end-users are not restricted to Eclipse environment in order to run the application separately. In addition, as long as the developers know how Eclipse plugins work, integrating other plugins developed on top of Eclipse to their new tool is easy, because they all share the common Eclipse plugin infrastructure underneath. For example, integrating the project planning and team collaboration functionality from Jazz into AgilePlanner only took several months (we are still integrating more features now). It shows how quickly researchers can develop a new functionality when they leverage platforms such as Jazz and Eclipse.

However, based on our experience, there are still several drawbacks. The first problem is dealing with the rate at which Jazz platform was changing. The changes made in Jazz for the new version was not obvious to the developers, hence sometimes it leads to many hours trying to debug the application without really understanding why. Our experience with integrating infrastructures show that developing an additional layer between two different software products was extremely helpful in order to safeguard against code changes in either of the two software products. By creating the adapter layer, we were able to narrow down the integration problem but it still involved many hours of navigating through Jazz APIs that were unfamiliar to the developers. The knowledge transfer between the Jazz developers and the researchers were very difficult because there were really no easy way to ask the right questions.

The second issue was the learning curve of understanding the code structure of Jazz in order to call the correct methods. Not all levels of the Jazz classes were necessarily needed by AgilePlanner and often too many layers of classes only added to the confusion as to which APIs needed to be used. Much of the coding done in the abstraction layer is to hide the unnecessary complexities in Jazz for AgilePlanner. To take an example, the following is the code required to create a working copy of the team project area.

```
(ITeamAreaWorkingCoopy)((IProcessITemService
)connection.getRepositoryConnection().getCli
entLibrary(IProcessItemService.class)).getWo
rkingCopyManager().createPrivateWorkingCopy(
teamArea)
```

The major drawback of using Eclipse and Jazz platform from an integration point of view is the number of hours required to understand the APIs. Our experience shows that working with existing platforms such as Jazz reduced the development time drastically, but it highlighted the issue of how difficult it really is to understand someone else's code in a short period of time. The problem with infrastructure integration is a catch-22 situation between the API flexibility requirements and the complexity that comes with it. We need flexibility in the API design so that it can accommodate all kinds of creative research projects. We also need a well-tested infrastructure that we can readily use. However, flexible APIs give a lot of power to the developers and the developers might get overwhelmed with too many possible choices.

In theory, designing software for extensibility is good and desired, not only for the original developers who may add more functionality to the software in the future but also the third-party users who would like to customize the software with their own extensions. If the developers do not carefully plan out the architecture ahead of time, the software code can become less readable, difficult to understand and riddled with too many APIs and interface layers. This problem is increased by a lack of good documentation or example code that uses the API.

Even with the difficulties we encountered during the integration, we still feel that using Eclipse and Jazz platform was the right choice for the development requirements for our research. Despite the difficulties in understanding the Jazz APIs, it saved considerable amount of time. As a research community, what we really need is a set of development environments and infrastructure that everyone can commonly use and that is easy to learn. More user involvement from the research community in Eclipse development means more APIs would be available and more platforms that we can take leverage. If other researchers can also provide their tools as Eclipse plugins, we have a better chance of integrating each other's work and have more time for research and less time coding. Much the same way our original decision to use Eclipse and Jazz was influenced by the number of users supporting the community and the easy accessibility to these environments, more user involvement will encourage development of Eclipse plugins that we can readily take advantage of.

5. CONCLUSION

Researchers need to be creative, but also we need a framework that has enough penetration in the market that learning the technology will pay off in the long run for the students. The more we gain re-usable software products that can be easily integrated with others, such as Eclipse plugins, the more time we will have for research. This is not an issue specific to research settings. There really is no boundary between research and industry settings for our necessity to develop better software faster by reusing existing components. This is not just a goal for the researchers but for industry practitioners as well. As with all software, working with a new code base becomes easier with more exposure. Additionally, the incentives to learn and use these platforms increase if both researchers and industry practitioners use them. It is important to try to build the bridge between the industry and research settings, because researchers can definitely take advantage of the good, robust code out there.

6. REFERENCES

[1] Eclipse, http://www.eclipse.org/

[2] IBM Rational Jazz
http://www-306.ibm.com/software/rational/jazz/

[3] Schuh, P. 2005. Integrating Agile Development in the Real World, Charles River Media

[4] Morgan, R. 2008, Distributed Agile Planner: A Card-Based Planning Environment for Agile Teams, MSc thesis, University of Calgary

[5] Park, S., Maurer, F. 2008. The Requirements Abstraction in User Stories and Executable Acceptance Tests, Research-in-progress track, Agile Conference 2008, Toronto, Canada

[6] Wang,X., Ghanam, Y., Maurer, F. 2008, From Desktop to Tabletop: Migrating the User Interface of Agile Planner, Engineering Interactive Systems 2008, Proc. from 2nd Conference on Human Centered Software Engineering (EIS. HCSE 2008), Italy

[7] Business Delegate Pattern, http://java.sun.com/blueprints/corej2eepatterns/Patterns/BusinessDelegate.html