Designing and implementing a job scheduler using Scrum

Tomas Strandberg

November 30, 2007
Master's Thesis in Computing Science, 20 credits
Supervisor at CS-UmU: Jerry Ericsson
Examiner: Per Lindström

Umeå University
Department of Computing Science
SE-901 87 UMEÅ

SWEDEN

ABSTRACT

Originally, the job scheduling found in Netcompetence Training System has been developed by many different developers in as many different ways. Each of the old jobs was designed, implemented and installed in different ways making it difficult for other developers to know where to find already installed jobs. In order to improve the development process a study of agile development with an in-depth study of Scrum was made. Scrum was then used to develop a new job scheduler. The results of the study and utilization of Scrum have led to a new improved and easy to extend job scheduler.

CONTENT

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

BACKGROUND

Netcompetence AB is an Internet Service Vendor (ISV) who develops Learning Management System (LMS) software. The product, called Netcompetence Training System (NTS), is used for building, administrating and publishing e-courses, e-tests and surveys to geographically spread companies through the web.

During the development of the product there have been many different developers and managers involved with varying experience in management, design and development of software. The product has also gone through many different development platforms such as ASP, ASP.NET 1.0, 1.1, 2.0. This has resulted in a wide mix of design patterns for the different parts of the product.

The job scheduling function in the product is one of the areas where the different solutions make it hard to get a good view of installed and existing jobs. It is also hard to maintain the code that exists for each job because of the different solutions. Installation of existing jobs is also difficult to perform since every job has its own installation process.

AIM AND SCOPE

The goal for this master thesis project is to design and implement a new job scheduling solution for NTS using agile development. The solution will provide Netcompetence with a new job scheduling module that is easy to install, extend and maintain. The solution should be able to handle several NTS customer installations on one machine where every customer has different demands of when a specific job should be executed.

In order to keep the thesis project within a reasonable scope some restrictions had to be made. The decision was made not to make the windows-service multithreaded but the application has been prepared to support it in the future.

Since agile development is new to Netcompetence the development process has not been set in the organization, therefore the process has not been used to 100% in this project. The

project has acted as a test project for agile development using Test Driven Development (TDD) [Newkirk, Vorontsov].

METHODOLOGY

The work was divided into four parts. First a study of agile development and scrum was performed, in order to evaluate if this process is a better alternative than the current development process, which was close to non existing. Then an analysis and gathering of existing jobs in the product was made. Then a gathering of requirements were compiled from the old jobs. New requirements were then added so that the new job scheduling module would be more maintainable and extendable. Finally, the design and implementation of the new solution for the job scheduling was made.

After the first draft for the design was ready, the work continued in small iterations. Every iteration added new features to the product until all the functionality was fully designed and implemented.

The development environment and products that were used for this project are: Windows XP Pro/Windows Server 2003/Window Server 2000, SQL Server 2000/2005. Microsoft Visual Studio 2005, log4net, RadControls for ASP.NET.

Programming languages used during development: C#, MSSQL, JavaScript, XML/XSLT, HTML.

AGILE DEVELOPMENT

In the following chapter a summary of agile development will be given. A short comparison to plan-driven development will also be given.

MANIFESTO FOR AGILE SOFTWARE DEVELOPMENT

In February 2001 seventeen people, all involved in developing software through "light-weight methodologies" [AGILE], met to discuss new ways of creating software. The purpose was to get away from heavy processes that were slow to respond to change and heavily controlled by processes. They created the *Manifesto for Agile Software Development* [AGILE], which is widely regarded as the standard definition of agile development today.

Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

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

That is, while there is value in the items on
the right, we value the items on the left more.

Kent Beck	*James Grenning*	*Robert C. Martin*
Mike Beedle	*Jim Highsmith*	*Steve Mellor*
Arie van Bennekum	*Andrew Hunt*	*Ken Schwaber*
Alistair Cockburn	*Ron Jeffries*	*Jeff Sutherland*
Ward Cunningham	*Jon Kern*	*Dave Thomas*
Martin Fowler	*Brian Marick*	

FIGURE 1, MANIFESTO FOR AGILE DEVELOPMENT [AGILE]

In the next four paragraphs I will go through the "values" of the *Manifesto for Agile Development* in greater detail. The values of the manifesto have inspired a list of principles which describe

the difference between agile practices and heavyweight processes (Appendix D: Principles behind the Agile Manifesto for Software Development).

INDIVIDUALS AND INTERACTIONS OVER PROCESSES AND TOOLS

In the *Manifesto for Agile Software Development*, the first "value" emphasizes that individuals and interactions have more importance than processes and tools. This means that while processes and tools is an important part of the development of software, the individuals and the interaction between them in the team is more important.

It can be interpreted that without a strong team a good process will not save a project from failure. That is why the team has to be constructed from people who are strong team players, who are good at communicating and interacting with the other members of the team. This is more important than being a raw programming talent.

The right tools in a project can be very important to success, but tools can be overemphasized. If the tools are too complex and hard to use it is just as bad as having no tools at all. Therefore the team should be put together first and then the team should decide what environment it takes to complete the task.

WORKING SOFTWARE OVER COMPREHENSIVE DOCUMENTATION

Working software over comprehensive documentation is the second "value" in the *Manifesto for Agile Software Development*. Working software is the final goal of any software project and documentation of the final product is an important part of the product. But comprehensive documentation is hard to maintain and takes a long time to produce. If the documentation is not kept in sync with the development of the product it becomes a large source of misunderstandings and misdirection. Therefore the team members should decide what documentation is absolutely essential to deliver with the final product.

When a new team member joins the team it could be argued that a comprehensive documentation is the best way to get the new member going. But the easiest and most reliable way to transfer knowledge to the new team member is through team-members and through the code, since the code is the only unambiguous source of information in the project. In the end, the focus should lie in delivering working software over comprehensive documentation.

CUSTOMER COLLABORATION OVER CONTRACT NEGOTIATION

With the third "value" the *Manifesto for Agile Software Development* says that contract negotiation is important. But it cannot be decided what the cost, requirement and schedule will be in advance. The product is not a commodity to order and expect delivered in a number of months. A product is something that evolves over time through constant interaction with the customer. That is why customer collaboration is more important than a contract that specifies the cost, requirements and schedule. Through regular feedback and collaboration with the customer the chances of delivering a product that the customer wants will increase dramatically.

RESPONDING TO CHANGE OVER FOLLOWING A PLAN

The last "value" of the *Manifesto for Agile Software Development* states that responding to change is more important than following a plan. The requirements change all the time due to changes in business environment, or because the customer sees something he likes or dislikes in the evolving product. That is why it is hard to start the project by setting a plan that should be followed until the product is ready. There is no way to know all the changes and difficulties that the project will run in to, weeks, months or even years in advance. That is why it is better to have detailed plans for the next week, rough plans for the next couple of months and very crude plans beyond that. That way it is easier to respond to changing requirements and changing business environment fast.

AGILE DEVELOPMENT PROCESSES

Traditional software development methods are often built up around documentation of a "complete" set of requirements, followed by architectural and high-level design. Coding and testing often comes at the end of the process. To bridge the gap between requirements and coding specifications are added that describes the relationship between the physical-world and the system. This approach is often referred to as plan-driven development.

Agile development processes, also known as lightweight-processes, is different from plan-driven development in many ways. It is usually an empirical process which involves a primary outline with short-term goals. The goals are reached by developing software rapidly in short iterative and incremental time periods with no more than sufficient overhead.

There are many different agile development methods such as FDD (Feature Driven Development), DSDM (Dynamic Systems Development Method), MSF for agile (Microsoft

Solutions Framework) and Scrum. The choice was made to use Scrum at Netcompetence because it seemed to fit into the organization and development. In the next chapter Scrum will be discussed in greater detail.

SCRUM

"Scrum is an iterative, incremental process for developing any product or managing any work. It produces a potentially shippable set of functionality at the end of every iteration."[1]

Scrum is not an abbreviation. It comes from the term SCRUM in rugby [WIKI SCRUM].

Scrum is an empirical process for developing complex software products. It is based on the idea that everything is not predictable, especially when developing complex software. The rest of this chapter will explain how Scrum works, starting with an overview of the Scrum lifecycle.

SCRUM LIFECYCLE

In this paragraph a short overview will be given of the whole Scrum lifecycle. After that each item in Scrum will be described in more detail.

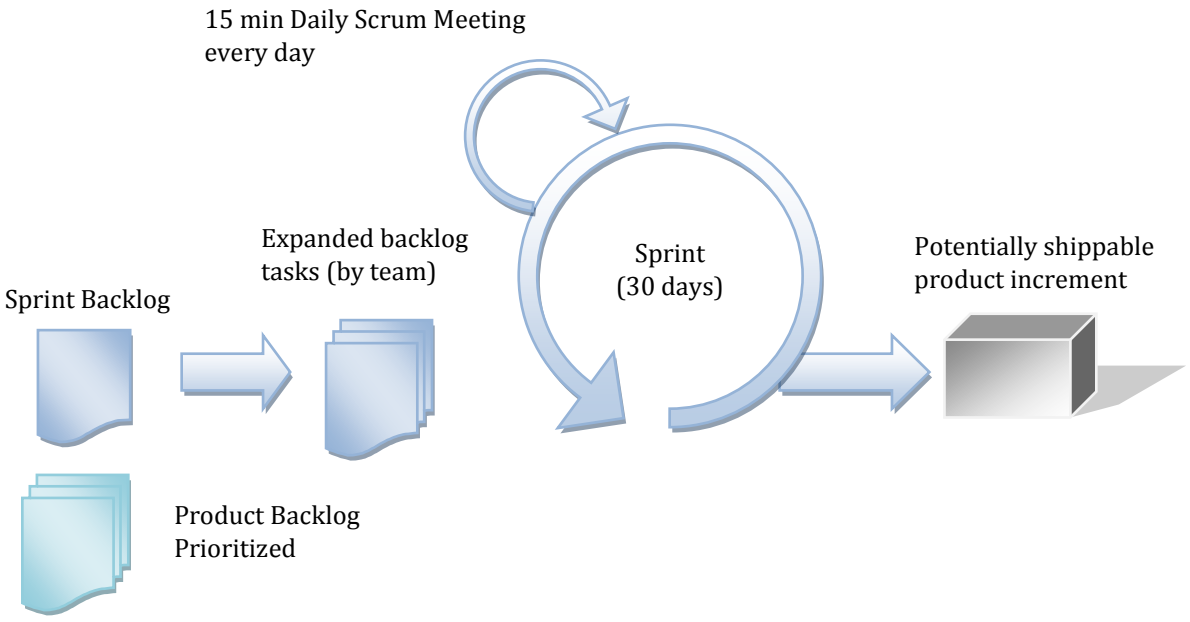

FIGURE 2, SCRUM LIFECYCLE.

[1] Taken from controlchaos.com

The Product Owner starts the Scrum lifecycle by collecting items for the Product Backlog. When there are enough items in the Product Backlog to fill a Sprint the Product Owner sits down and prioritize the items. Then the Product Owner, the Scrum Master and the Scrum Team sit down and create a Sprint Backlog. They fill the Sprint Backlog with items from the Product Backlog at the same time as they prioritize and estimate time for each item. When the Sprint Backlog is filled with the amount of work the Sprint Team thinks it can handle during the next Sprint, the Sprint starts. During the Sprint a Daily Scrum is held every day and the Burndown chart is updated every day. When the Sprint is done a Sprint Review meeting is held for the people that do not belong to the Sprint Team. In this meeting a demo of the latest Product Increment is shown. Then a Sprint Retrospective meeting is held. When this is done a new iteration starts with the creation of a new Sprint Backlog.

PRODUCT BACKLOG

Before the project can start, anyone who has any ideas about what features should be in the product sits down and starts a Product Backlog. In the backlog all requirements and features that anyone can think of are included. It can be anything from a bug-fix to new techniques or new features. The Product Backlog is a dynamic, prioritized queue which evolves over time. Whenever a customer, developer or manager comes up with a new business or technical feature it is added to the Product Backlog. The log is prioritized by the Product Owner and is used as part of the input to a Sprint Planning meeting. When a meeting is started, the items with the top priority are selected for implementation from the Product Backlog. To start the first Sprint the Product Backlog only has to include work for a 30 day Sprint (including weekends). The Product Backlog should be available to everyone so that it is easily accessible.

SPRINT PLANNING

In order to determine which items from the Product Backlog that should be implemented in the next Sprint a Sprint Planning meeting is held. The meeting actually consists of two meetings. In the first meeting the Scrum Team and Scrum Master meet with the users, managers and Product Owner to identify what items from the Product Backlog should be included in the next Sprint. After that meeting the Scrum Team and the Scrum Master meet to plan the Sprint.

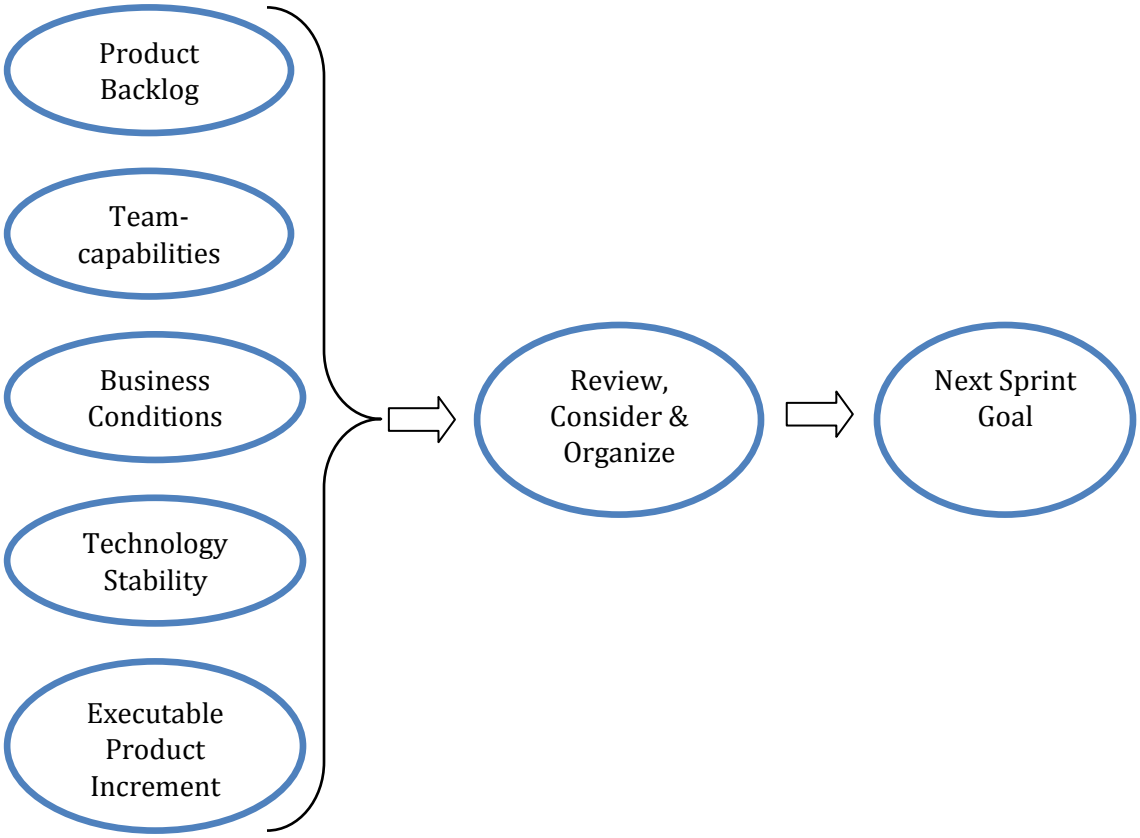

FIGURE 3, INPUT FOR NEW SPRINT.

The input to the first meeting is shown in Figure 3. The goal is to decide what items from the Product Backlog should be implemented, and to have a Sprint Goal produced. The Product Owner presents the top priority items from the Product Backlog. Combined with the team's capability and the output from the last Sprint the group tries to identify what items from the Product Backlog it believes can be developed during the next Sprint.

Some examples of top priority Product Backlog:

Create a viewer to keep track of the jobs.

Make the job handler multithreaded.

Add logging functionality to the jobs with the help of log4net.

Make the import of users more stable.

After selecting the Product Backlog the group formulates a Sprint Goal which could be:

Sprint Goal: Create a workflow for setting up, starting, executing and monitoring a job.

This vague Sprint Goal is used so that the Scrum Team has some room to interpret the goal in order to achieve it. For example, setting up could mean add some data in the database that triggers the job to start or it could mean, create an interface where the user adds information about the job and saves it to the database. The final implementation of the Sprint Goal is up to the Scrum Team. During the Sprint only the Scrum Team can decide how to meet the Sprint Goal. If the Product Increment does not satisfy management and the Product Owner at the Sprint Review, they can make decisions then about how to proceed.

In the second meeting, the Scrum Team creates a Sprint Backlog to meet the Sprint Goal. It contains a list of tasks that the Scrum Team needs to finish in order to meet the Sprint Goal. The tasks are more detailed than in the Product Backlog and are used to convert the Sprint Backlog into a working Product Increment. The tasks should take 4-16 work hours to complete. If the team finds a task that is estimated to take longer than sixteen hours the task should be split into smaller tasks. If a task is estimated to take less than four hours it is combined with another task so the total estimate lies within the interval. The combined tasks can be of any kind but it is preferred to combine two tasks that are similar.

The Sprint Backlog is updated after every Daily Scrum to reflect the changes and progress that has been made since the last Daily Scrum.

SPRINT

After the Sprint Planning meeting, the Scrum Teams starts the Sprint. In a period of 30 calendar days (one Sprint) the team has to meet the Sprint Goal set at the Sprint Planning meeting. To be able to meet the Sprint Goal, the Scrum Team has complete authority during the Sprint. No-one can tell the team what to work on next. No-one can add or remove items from the Sprint Backlog. The only one who can change anything is the Scrum Team. As long as they reach their Sprint Goal they can do anything that is needed to get there.

During the Sprint the team has two mandatory tasks besides reaching the Sprint Goal. The first is to attend the Daily Scrum meetings. Preferably in person, if that is not possible then over the phone. The second task is to update the Sprint Backlog after every Daily Scrum. By doing these tasks it is easy to get a good overview of the status of the Sprint at any given time. It is easy to see if the Sprint Goal will be met or if the team has taken on too much work.

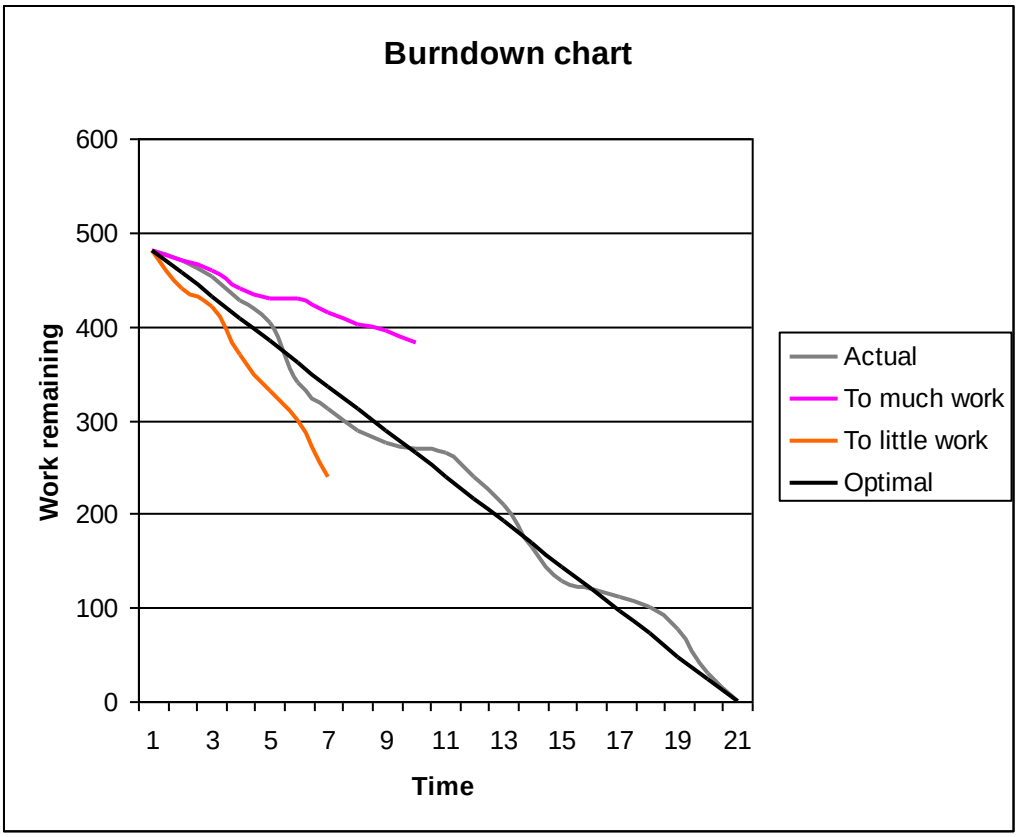

FIGURE 4, BURNDOWN CHART SAMPLE.

After every Daily Scrum the Scrum Master updates the burn-down chart (Figure 4). If the chart is going in the direction of the pink line then the team has taken on too much work in this Sprint and has to return some tasks from the Sprint Backlog to the Product Backlog. If it follows the red line then the team has taken on too little work and has to add some new tasks to the Sprint Backlog from the Product Backlog.

A Sprint is not often terminated but it can happen. When the team feels it does not get the support it needs to complete a Sprint, or if someone changes the scope of the Sprint to much. The team can ask for the termination of the Sprint. This is a very important part of Scrum. If the team has the authority to terminate the Sprint there will be less chance of someone trying to change the scope of the Sprint once it has started. Management can also cancel a Sprint if for example the Sprint Goal becomes obsolete.

ROLES

According to Scrum practices, there are three different roles, the Product Owner, the Scrum Master and the Scrum Team. The Product Owner is responsible for the product; the Scrum Master is the manager for the Scrum Team which consists of the developers, testers, technical writers etc.

PRODUCT OWNER

The Product Owner is one person, not a committee, who is responsible for managing and controlling the Product Backlog. The Product Owner maintains the Product Backlog and ensures it is visible to everyone in the organization. This ensures that only one set of requirements drives the development since all prioritizing will go through the Product Owner and there will only be one set of priorities. The Product Owner can be influenced by management, committees, customers and sales people etc, but the Product Owner is the only person who prioritizes the Product Backlog. The Product Owner is the filter between everyone who wants features and priorities added or changed in the product and the Scrum Team. Since the Product Owner is the only person who prioritizes the work, the Scrum Team can focus more on the current tasks and the interruptions will be reduced.

Another important task that the Product Owner has is estimating the time it takes to develop a feature in the Product Backlog. The estimates are made with the help from people who know the product. This can be developers, technical writers, testers or any other person who knows the product and the technology used. The estimate includes all the time it takes to develop a feature. The estimates have to include architecture, design, development and testing. The Product Owner is also officially responsible for the project.

SCRUM MASTER

The Scrum Master is the person who manages the Scrum process in the organization. The Scrum Master is responsible for enacting and enforcing the Scrum values, rules and practices. When the project starts, the Scrum Master and the customer and management finds and assigns a Product Owner to the project. Then management together with the Scrum Master work to form Scrum Teams. The Product Owner, Scrum Master and the new Scrum Teams then work together to create a Product Backlog for the next Sprint. The Sprint is initiated and planned by Scrum Master and the Scrum Teams. When the Sprint starts the Scrum Master conducts Daily Scrums. During the Daily Scrum the Scrum Master is responsible for removing impediments and making decisions based on the information at hand. This is the way the Scrum Master helps the Scrum Teams work at the highest possible level. By removing impediments, either personally or by causing them to be removed, the

Scrum Master makes sure the Scrum Team can work without interruption. By taking decisions on the information at hand, the Scrum Master ensures that the work continues. If the decision is the wrong one it can always be changed later.

SCRUM TEAM

A Scrum Team should contain 5-8 people because that is the size of the team when the team dynamics will work best. If there are fewer than five people in the Scrum Team it limits the productivity gains, since the amount of interaction that can occur is limited. If more than eight people are in the Scrum Team, the Daily Scrum may become too difficult for the Scrum Master to handle. In that case the team should be split into several teams instead. The different teams should then select items from the backlog and make a commitment for a Sprint. The items selected from the backlog should minimize the interaction needed between the Scrum teams and maximize the cohesion of work within each team.

A Scrum Team is self-organizing in the way that all members on the team contribute to the outcome of the Sprint; they organize themselves to meet the Sprint goal they have committed to. Each member in the team applies his or her skills to every problem the team faces which improves quality and raises productivity. There are no roles in a Scrum Team, instead the Scrum Team have to be cross functional since all the items in the backlog have to be architected, designed, implemented, tested and documented during one Sprint.

The Scrum Team has the authority to do whatever is needed to meet the goal they have committed to for the Sprint. It must however conform to and use any existing standards, technology and architecture, in order to ensure that the Sprint delivers a Product Increment that fits into the organization and that it can be understood by other people.

CHICKENS AND PIGS

A chicken and a pig are together when the chicken says, "Let's start a restaurant!"

The pig thinks it over and says, "What would we call this restaurant?" The chicken says, "Ham n' Eggs!" The pig says "No, thanks. I'd be committed, but you'd only be involved!" [Schwaber, Beedle]

The team members are the pigs in the joke, and all the other people are chickens. The members are called pigs because they commit to a goal and do the work required to meet that goal. The chickens can attend Daily Scrum meetings but they cannot talk or interfere in any way. They should stand in the periphery of the room, since they only attend the meetings as guests.

DAILY SCRUM

The Daily Scrum meeting is a fifteen minute meeting where the members of the Scrum Team deliver the status of their work since the last Daily Scrum. The goal of the meeting is to enhance communication in the team, eliminate other meetings and to identify and remove any obstacles that can delay work for the team. This is accomplished by setting up a Scrum Room where the meeting is held every day at the same time. The Scrum Master is responsible for conducting the meeting and for keeping the time. The Scrum Master asks each member on the team three questions:

What have you done since last Scrum?

What will you do between now and the next Scrum?

What got in your way of doing work?

The purpose of the questions is to resolve any issues the developer might have and to inform the rest of the team about the progress. Examples of issues can be, server is down, asked to attend meeting with management or do not know how to proceed. If any obstacles are identified it is the Scrum Masters top priority to solve them. This can be done by asking other members on the team for help or by getting help from outside the group when needed. The important thing is that the obstacles and issues should not be solved during the Daily Scrum. This meeting is only there to bring up any issues and to inform everyone about the progress. If another team member thinks he or she can help the person with a problem another follow-up meeting is scheduled.

The Daily Scrum meeting is a chance for everyone both in the project and those not in the project to get a good feel for how things are going. People who are not in the Scrum Team are allowed to attend the Daily Scrum if they wish, but they cannot talk or interact in any way. They are present as guests and are only there to get information about how things are progressing.

SPRINT REVIEW

After each Sprint a Sprint Review meeting is conducted. This is done to let management, Product Owner, and anyone else interested to see what the Product Increment contains. This is often done by a demo of the latest product increment. The Sprint Review meeting also prepares people for the next Sprint Planning meeting by letting them know what was

developed and what was not developed. It also shows the strengths and weaknesses in the product increment that can influence the next Sprint Planning meeting.

During the Sprint Review the Product Owner checks off the items on the Product Backlog against the items that have been implemented in this Sprint. The Sprint goal and the actual result are compared and any differences are discussed.

SPRINT RETROSPECTIVE

After the Sprint Review the Scrum Master, Product Owner and the Scrum Team sits down and discusses the last Sprint. The goal of this meeting is to analyze and learn from the last Sprint. Every member gets to say what he/she thought was good in this Sprint and what could be improved in the next Sprint. This meeting is important to make the team evolve and improve while the project is running.

TECHNOLOGIES

In the following chapter, a very short presentation of the techniques used when developing the job scheduler is described.

MARKUP LANGUAGES

A markup language combines text and extra information about the text. The extra information is expressed using markup. The markup is intermingled with the primary text and holds information about the text's structure and/or presentation.

Example: <bold>Title </bold>

HTML (HYPERTEXT MARKUP LANGUAGE)

HTML is the major markup language used to create web pages. It is used to describe the structure of text-based information in a document using tags. The tags can consist of paragraphs, lists, headings and so on. To supplement the HTML document interactive forms, images and other objects such as Flash can be added. It can also be made dynamic, called Dynamic HTML (DHTML), by adding embedded script that can be used to manipulate the behavior of HTML processors such as browsers.

XML (EXTENSIBLE MARKUP LANGUAGE)

XML is a general-purpose markup language that is used primarily for sharing data between different applications on different platforms. One of the advantages with XML is that it is designed to be relatively human readable.

Application languages such as Extensible HTML (XHTML), Really Simple Syndication (RSS 2.0) and thousands of others are implemented in XML by adding semantic constraints. The constraints tell the application which tags are allowed, and what data types are allowed for the information contained in the tags.

JAVASCRIPT

JavaScript is a scripting language that can be used on web pages to manipulate content and respond to events such as a button click. It is a dynamic language that supports prototype based object construction which makes it possible for it to work both as a procedural and object oriented language. Objects are created by attaching properties and methods to empty objects in runtime. The new object can then be used as a prototype to create similar objects.

AJAX (ASYNCHRONOUS JAVASCRIPT AND XML)

The purpose of Ajax is to make web pages feel more responsive. This is accomplished by exchanging small amounts of data with the server instead of reloading the entire page when the user requests a change. The exchange with the server is made through the XmlHTTPObject which makes an asynchronous request to the server. The response is then added to the page through manipulation of the DOM (Document Object Model) with JavaScript.

.NET FRAMEWORK

The .NET Framework is a development and execution environment from Microsoft. The framework gives the programmer the capability to develop different kinds of applications in any of the different .NET languages. The applications that can be developed are console applications, Web Forms, Windows Forms, XML Web services and Windows services.

The following chapter will briefly summarize the two main parts of the .NET framework, the .NET framework class library (FCL) and the common language runtime (CLR). I will also give a short description of some other technologies that were used from the .NET Framework to create the job scheduler.

.NET FRAMEWORK CLASS LIBRARY

The .NET FCL is an object-oriented, language independent library that provides the following functionality to the programmer:

- Representation of base data types and exceptions.
- Encapsulation of data structures.
- I/O.
- Provides information about loaded types.
- Invokes .NET Framework security checks.
- Data access.
- Rich client-side Graphical User Interface (GUI) called Windows Forms, and server-controlled, client-side GUI called Web Forms.

COMMON LANGUAGE RUNTIME

The Common Language Runtime (CLR) is a language-neutral managed execution environment that contains several components that provide the application with a run-time environment and run-time services. The components that constitute the common language runtime are:

- Class loader.
- IL to native code compiler.
- Code manager.
- Garbage collector.
- Security engine.
- Type checker.
- Thread support.
- Exception manager.
- Debug engine.

- COM marshaler.
- Base class library support.

ASP.NET

Active Server Pages for .NET (ASP.NET) is a technology that is used when developing dynamic web-pages on the .NET platform. It essentially contains two parts, the client-side code and the server-side code, called code-behind. The client-side code contains markup and script code while the server side contains any .NET Framework compatible language code. The server side is developed in an object-oriented, event-driven paradigm not in a sequential paradigm, like old ASP.

WINDOWS FORMS

Windows forms classes in the .NET Framework are used to develop GUI applications for Windows. The classes contain buttons, textboxes, menus, toolbars and other screen elements needed to develop a desktop application.

XML WEB SERVICES

A XML Web Service implements program logic and exposes functionality to remote applications. The functionality is accessed using standards such as XML, HTTP and SOAP. XML Web Services can be used to integrate applications that are written in different programming languages and run on different platforms.

WINDOWS SERVICE

A Windows service is a program that runs as a background process. The program can be managed from the Service Control Manager. It does not have a user interface which makes it ideal for tasks that do not require any user input. Since a Windows service can be run under a specific account it makes it possible for the service to run even though no user is logged on

to the computer. A Windows service can be run under the logged in user account or you can set it up to run under a specific user account.

CASCADING STYLE SHEETS

Cascading Style Sheets (CSS) is used to describe how markup language (HTML, XML etc) pages are presented on the screen and in print. Cascading Style Sheets add fonts, colors and layout to the information in the document.

RESULTS AND IMPLEMENTATION

In this chapter the result and implementation will be discussed.

BACKGROUND

The problems with the old job scheduler were:

- It was not working if no-one was logged-in on to the server where the batch jobs where installed.
- All customer instances hade to execute the same job at the same time.
- If the server was down one day the job scheduler could not execute jobs that were configured to run on that day.
- All available jobs where set up as their own batch job. This made it hard to keep track of which jobs were installed and at what time they would run since there were multiple starting points for the installed jobs.

To resolve these shortcomings the decision was made that a new job scheduler would be developed.

A job of gathering a list of old problems and adding new features was started. The result was a list of requirements that would be used as a starting point for the development of the new job scheduler.

REQUIREMENTS

After the gathering of requirements for the new job scheduler was accomplished, the following Product Backlog was produced (not prioritized).

- One installation of the job scheduler should be able to handle multiple customer installations on the same server.
- It has to be easy to add new types of jobs to the scheduler without the need for a recompile and without any downtime.
- It should be possible for customers to run different versions of the product and the scheduler should still work for all instances.
- It should be easy for administrators to add/remove and update jobs in the scheduler from NTS.
- Logging and tracing capability with the ability to track in real-time or log to disk or any other permanent storage.
- Ability to run a job for an earlier date than today. Gives the ability to run jobs for days when the scheduler/computer is down.
- All the old jobs have to work with the new scheduler.
- A customer should be able to choose when a specific job is scheduled.
- Easy to find and maintain the code for the jobs.
- The solution should run on the existing platform.

TABLE 1, JOB SCHEDULER PRODUCT BACKLOG

After the requirements were defined, a first draft of the systems architecture was produced. In the next section a deeper explanation will be made about each part. But first a short overview will be given.

OVERVIEW

When the first product backlog was defined, five major parts for the job scheduling where identified.

- *Viewer* – for monitoring the progress.
- *Job scheduler* – scheduler that starts jobs on the right time.
- *Jobs* – the jobs that can be executed from the scheduler. Sample of an existing job is the daily import of users.

- *Web interface* – for monitoring, adding and editing jobs to the scheduler for a specific customer.
- *Web services* – exposes the customer instances to the job scheduler.

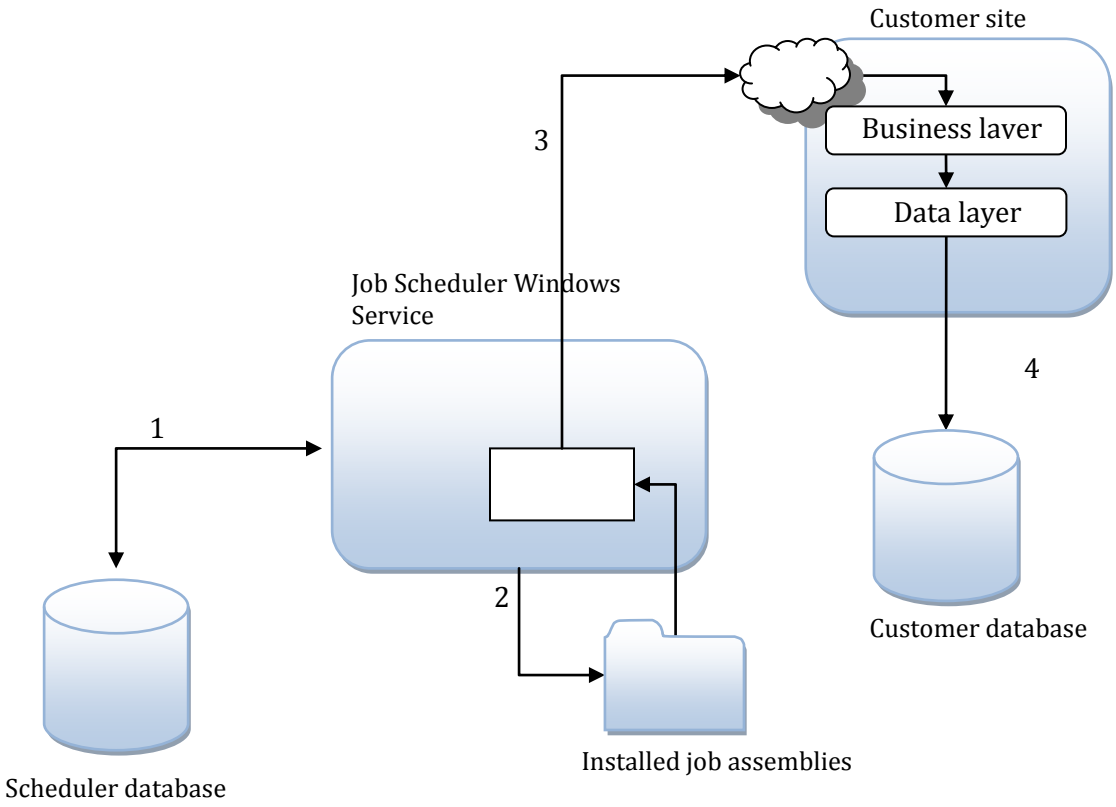

FIGURE 5, JOB HANDLING OVERVIEW

In Figure 5 an overview of how the system works is shown. The process starts with the job scheduler asking the scheduler database for any planned jobs (1). The job scheduler gets the next job to process and loads the appropriate job assembly, based on the data from the database, from the installed job folder (2) and calls the Run method in the loaded assembly. The job assembly the makes a web-service call (3) to the customer site that added the current job schedule to the job scheduler database. The customer site performs the necessary work according to the web-service that has been called. When the job has finished executing everything is reported back and the job scheduler database is updated with the correct result.

JOB SCHEDULER

The job scheduler was developed as a Windows Service. One of the advantages with this is that it can be run even though no-one is logged-in on the server. Another advantage is that it can be run under a specific account that has limited privileges on the server. The service starts when Windows starts and runs as a background process. This solves the problem with multiple starting points for the jobs since everything have to go through the new job scheduler.

The scheduler is in charge of scheduling jobs for all customer installations on the server. All the scheduler knows is what database to look in to make the decision about when the next job should be started. If no job is currently in the queue, the job scheduler sleeps for 30 minutes, until it is time to execute the next job or until someone tells it to start.

The scheduler can be forced to reload the job scheduler queue from any NTS customer instance. When this happen the job scheduler queue is flushed and re-read. This allows a customer to run a one-time job whenever they like. If no other job is currently running the new job will execute at once. If a job is running, then the new job is queued like any other job. Since the jobs in the job scheduler are synchronously executed in this version, it is very important that the scheduler can be activated when it is in sleep mode.

When the job scheduler starts a job, it extracts the parameters necessary to execute the job from the database. Among those parameters, there are two very important ones, the Assembly Name and the Class Name. The job scheduler dynamically loads the assembly and creates an instance of the class name in the loaded assembly. It then calls the `Run(XmlDocument parameters)` method on the newly created object. The Run method is defined in the *IScheduledJob* interface (see Appendix B: *IScheduledJob* interface) which all job-classes that can be run by the job scheduler have to implement.

In the Run method the job prepares the data that will be sent to the customer site where the actual work will be done. When the job has prepared the data it looks for the URL to the web-service to call inside the parameters xml. It then calls the web-service with the prepared data. When the web-service returns the job scheduler saves the result in the database and updates the job history table.

DATABASE DESIGN

To keep track of all the customer installations and all the customers' different job schedules a SQL-server database was developed. The database tracks and keeps information about when a customer wants to run their next job. It also stores history about the execution results. The database consists of five tables which are described below. The table name is shown in bold text and the column names are in italic.

- **Instance** - Site information about where the jobs can find the web services needed to execute the job on the right customer instance.
    - *ID* – Primary, Unique identity column (value generated by SQL-Server).
    - *URL* – Base URL to the site where the web services are installed.
    - *InstanceName* - Name on the customer site.

- **Job -** Information about installed jobs that can be run. If a new job assembly is installed, the information about it should be entered into this table. When it is, all customers that have their application on the same server can see the new job in their site at once.
    - *ID* – Primary, Unique identity column (value generated by SQL-Server).
    - *AssemblyName* - Name of the assembly that the executing code exists in. The assemblies should be installed in the job scheduler service folder.
    - *ClassName* - Fully qualified name of the class where the *SchedulerWKT* interface is implemented.
    - *Name* – Name for the job.
    - *Description* **–** Description for the job.

- **JobInstance –** Stores information about jobs for a specific customer site
    - *ID* – Primary, Unique identity column (value generated by SQL-Server).
    - *InstanceID* – Id of the instance that the job belongs to.
    - *Parameters* – The parameters for the job to execute. Generated by the web-control for the job in NTS. See for explanation.
    - *JobID* – Id of the job that the instance belongs to.
    - *Active* – Flag for active/inactive mode for the current job instance.
    - *Description* – Description for the job instance.

- **JobInterval** - Contains information for the execution interval for a specific job instance.
    - *ID* – Primary, Unique identity column (value generated by SQL-Server).
    - *JobInstanceID* – Instance that the current information is valid for.

- *IntervalType* – If this is a recurring job the interval type is set here.
  - 0 = Once.
  - 1 = Daily.
  - 2 = Weekly.
  - 3 = Monthly.
- *NextProcessTime* – Next time to start the current job instance.
- *RetryCount* – Max number of retries for the current interval.
- *RetryInterval* – Minutes between retries.
- *Active* – Flag for active/inactive mode for the job interval.

- **JobHistory** - Keeps track of the history of executed job intervals.
  - *ID* – Primary, Unique identity column (value generated by SQL-Server).
  - *Status* – Status for the job instance.
    - 0 = Not started.
    - 1 = Running.
    - 2 = Failed.
    - 3 = Success.
  - *JobID* – ID for the job.
  - *InstanceID* – ID for the customer instance.
  - *JobIntervalID* – ID for the interval.
  - *Parameters* – Parameters used when starting the job. These parameters are used if the customer for some reason wants to restart this specific execution.
  - *ExecutionTime* – When the job should have been started.
  - *FinishTime* – When the execution was finished.
  - *ProcessTime* – When the execution was started.
  - *Details* – Stores details about this execution.

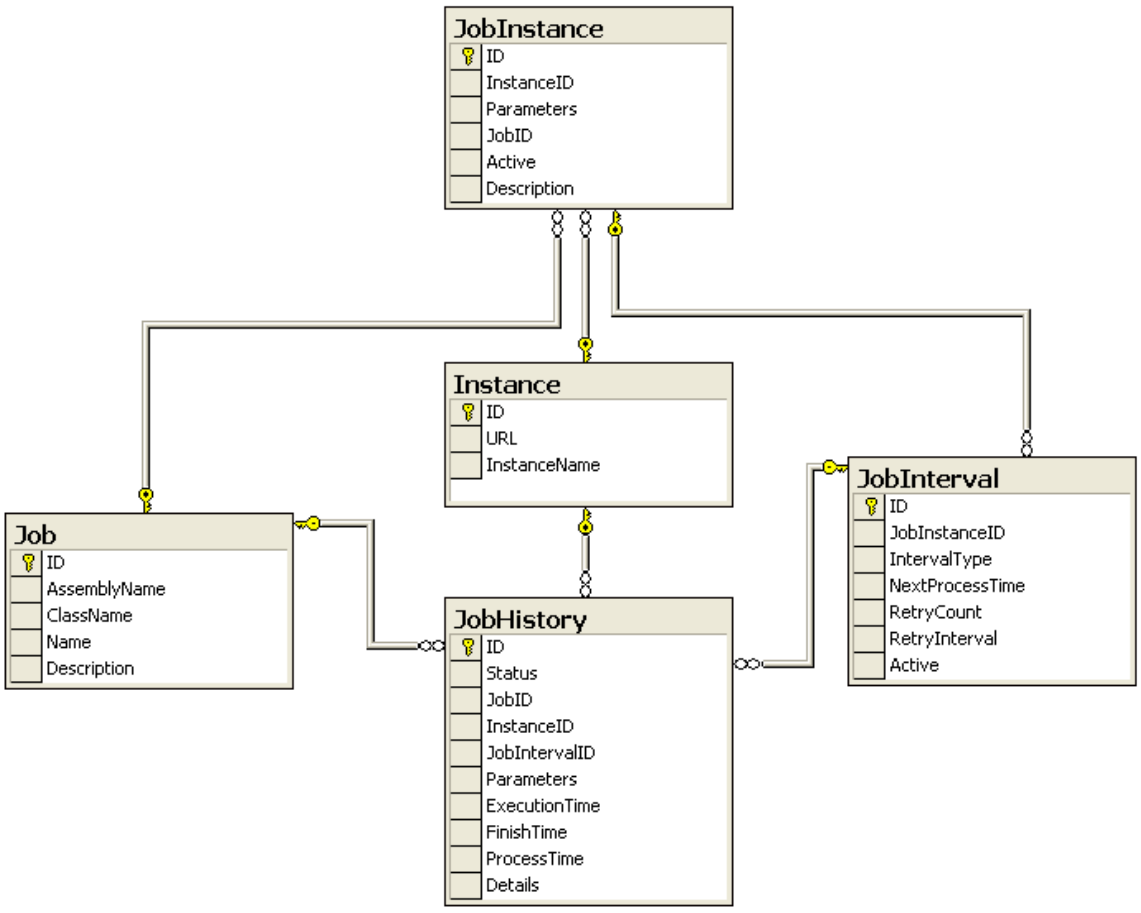

FIGURE 6, JOB SCHEDULER DATABASE, DATABASE DIAGRAM OF RELATIONS.

The job scheduler gets the information from the SQL-server through a component layer developed using ADO.NET. The solution is a 3-tier solution where the windows service represents the interface layer.

WEB SERVICE INTERFACE

Each customer instance exposes a set of web-service methods that are used by the jobs when executing. The web-service methods are called from the loaded job assembly in the job scheduler. The URL to the web-service, that the current job should call, is embedded in the parameters XML that is passed to the job. The job extracts the base URL and adds the

path to the web-service that the job wants to call. It then calls the method with the parameters that the method demands.

In the old solution the job database had to store each customer database connection string. The business layer also had to pass the connection string to the database layer. This is not a good solution since it connects the business layer to tightly with the database layer. With the new solution it is possible for the jobs to execute on the correct customer instance without knowing anything but the base URL for the web-services for the customer. This solves the problem with the connection string since it can be stored only in the web.config for the customer.

WEB USER INTERFACE

Each customer installation has a set of web pages where the customer can add/edit and delete jobs in the scheduler. The customer can also use the web-pages to check progress on a running job and to check history for old executions.

ACTIVE JOBS

The active jobs page (Figure 7) shows a list of all active jobs for the customer. From this page it is possible to create a one-time user-import job using the “Import users…” button. This button is important when setting up a new customer instance since it is a quick way to get a large amount of users into the system.

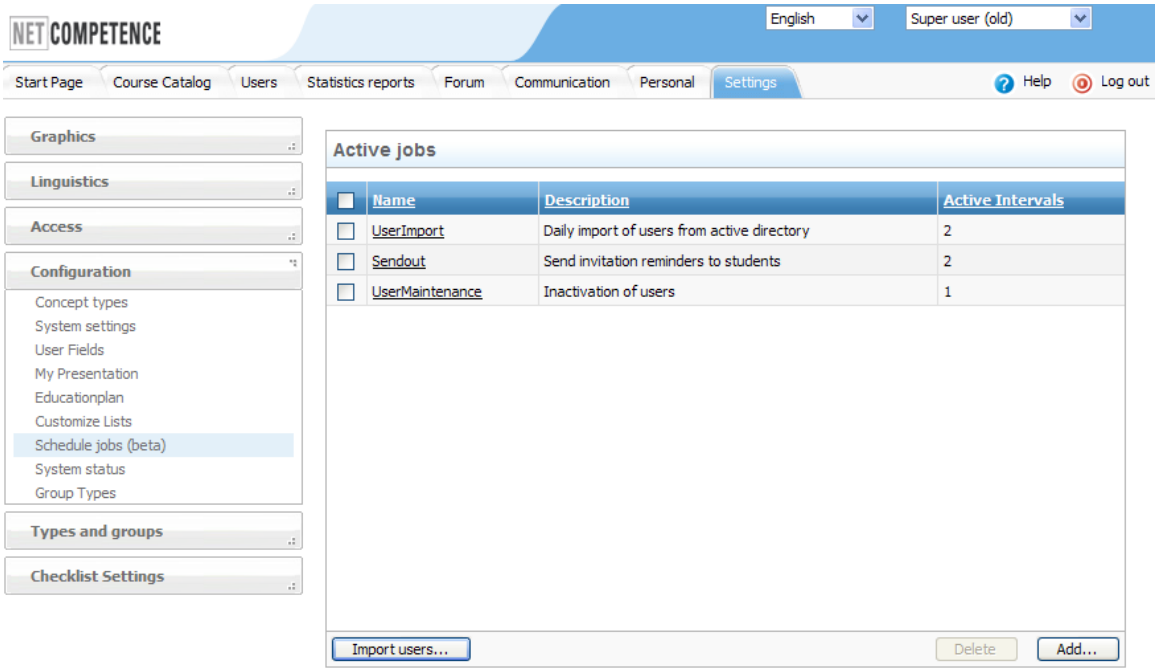

FIGURE 7, ACTIVE JOBS OVERVIEW.

JOB EDITING

The job editing consists of a number of pages where the customer can modify the settings for a specific job. Here the customer can change the execution intervals for a job and look at the execution history for a specific job interval.

JOB OPTIONS

On the job options page (Figure 8) the customer sets up a specific job. This page generates the parameters XML necessary for the job to execute. All jobs have the first three options, Job Type, Active and Description, in common but the parameter part is different for each job. This part is presented as a dynamically loaded control based on the selection in the job type drop down control. The parameters control automatically generates the inner-element part of the parameters XML when the save button is pressed.

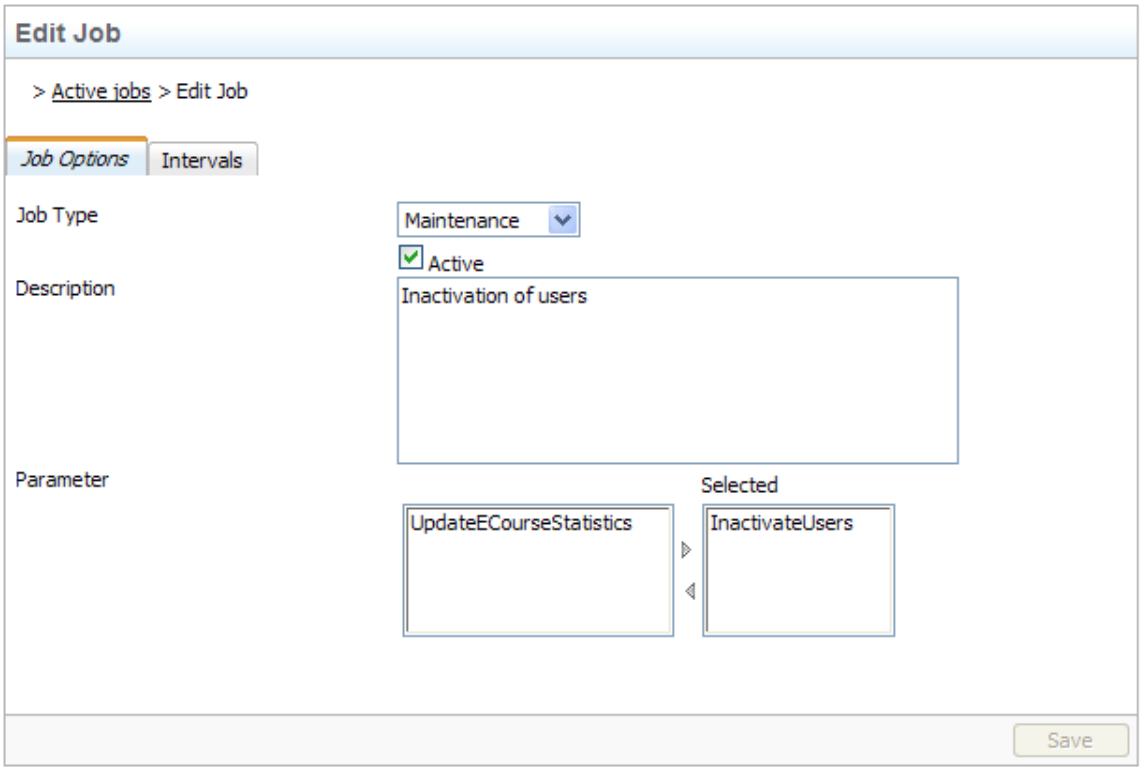

FIGURE 8, JOB OPTIONS PAGE.

JOB INTERVAL

Job interval is the interval on which each job executes. Each job can have one or more job intervals. On the overview page (Figure 9) the customer can check the results for the last time the intervals for the selected job were executed. If the customer wants he/she can choose to look at the history for each interval.

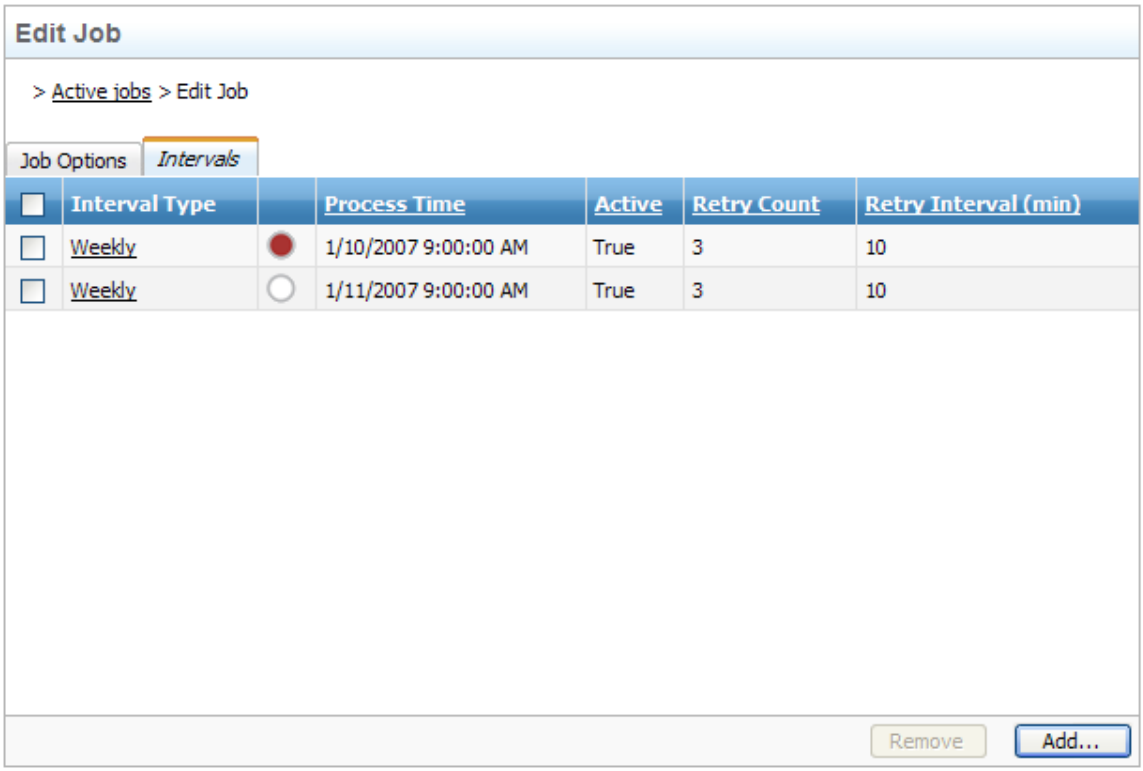

FIGURE 9, JOB INTERVAL OVERVIEW.

JOB HISTORY

The job history overview (Figure 10) shows all executions for a specific job interval. If the customer clicks the circle at the start of the line he/she will see the details for the specific (Figure 11) execution.

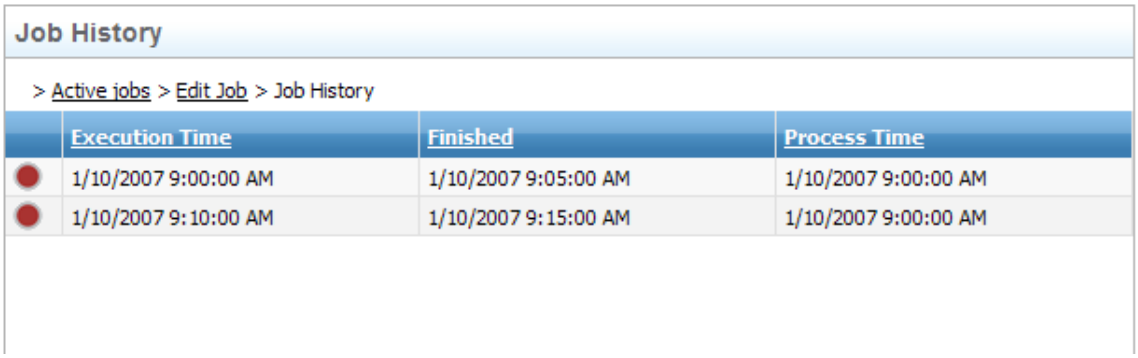

FIGURE 10, JOB HISTORY OVERVIEW

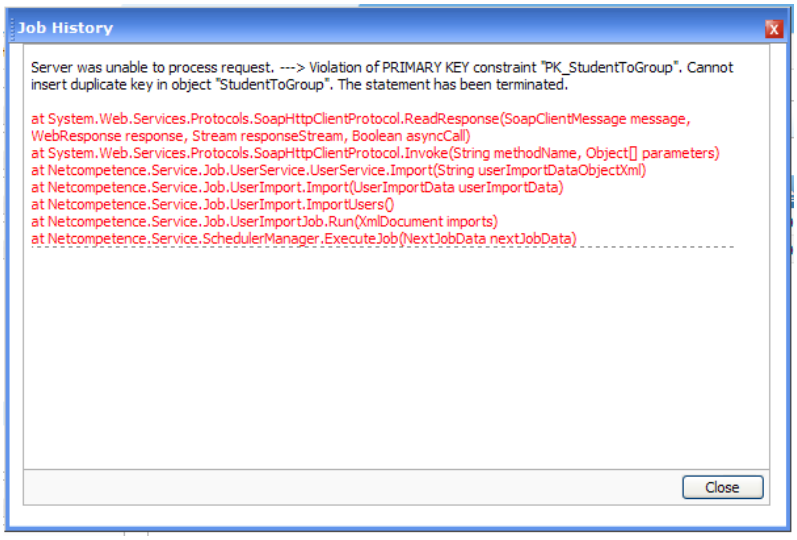

FIGURE 11, JOB HISTORY DETAILS

VIEWER

Since the job scheduler runs as a windows service it is hard to debug and get feedback from it. The viewer was developed to be able to monitor and debug the behavior of the job scheduler. It is developed as a simple win-form application that monitors the job scheduler and shows all the logging that takes place inside the job scheduler. The viewer can also be used to make the job scheduler get the next job to execute. The viewer also logs all the information from the job scheduler to a file. This is useful when trying to debug a customer installation since the job scheduler only logs to a remoting interface.

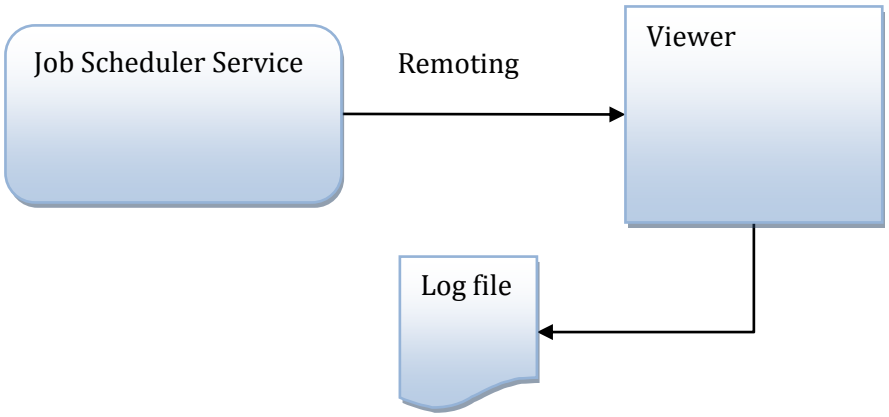

FIGURE 12, JOB SCHEDULER AND VIEWER.

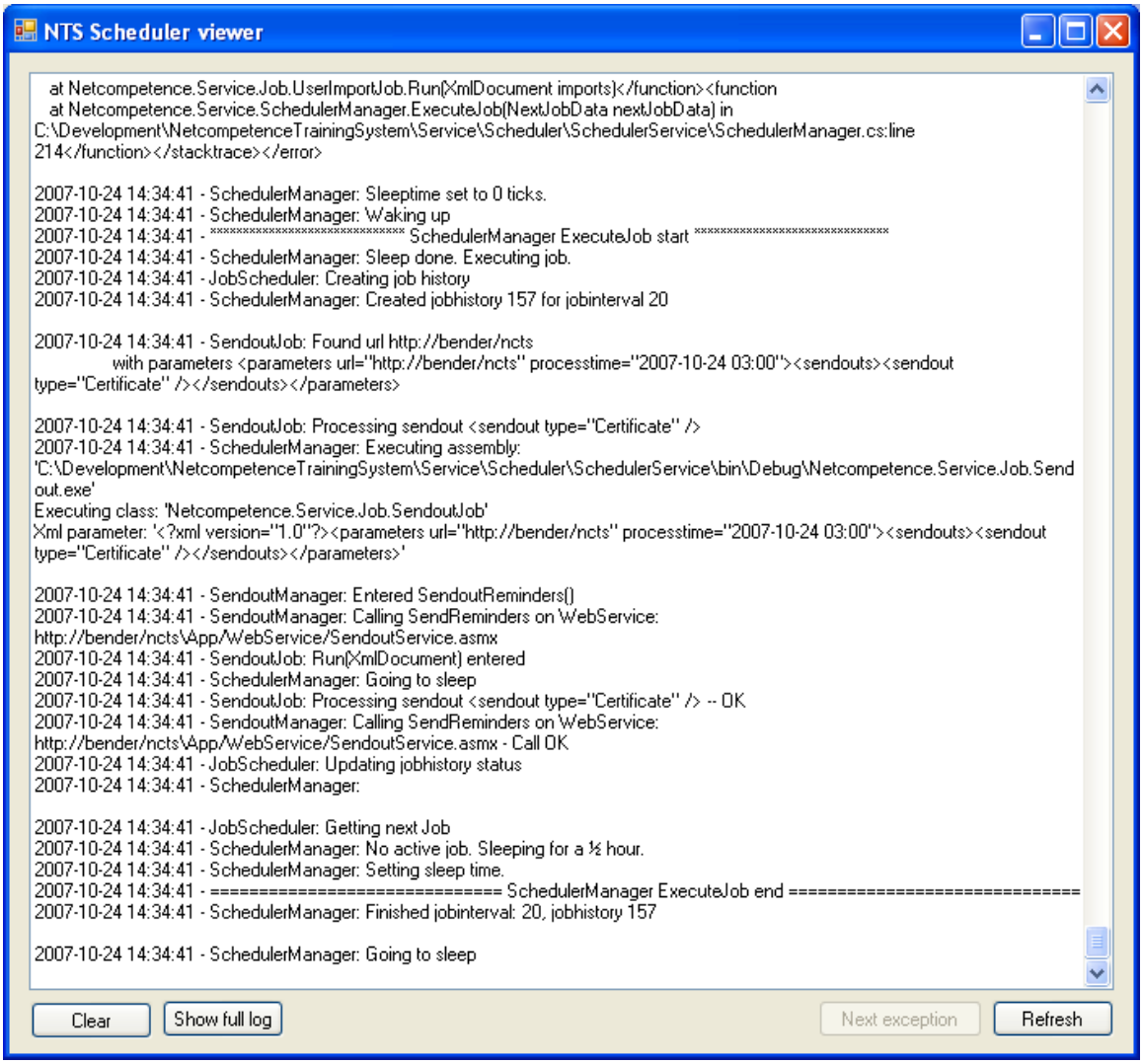

FIGURE 13, SCHEDULER VIEWER UI.

CONCLUSION

The new job scheduler has solved a number of issues that existed in the previous version. This has been accomplished by developing a central job scheduler that starts all job executions for all customers on one system. The job scheduler has also been designed to support installation of new jobs in one central place instead of separate installation paths for each job.

In my opinion Scrum improved the development process a lot compared to previous methods used at Netcompetence. The product backlog and the sprint backlog was a great help when deciding in what order items should be researched and implemented. They also helped in structuring the work and setting up a short term plan. I think Scrum will be a very good method for Netcompetence to use when going forward with their development of the product, since it will give them more structure and a better overview of where the product is heading. It will also provide a better way to see what will be developed and when it is finished. This will lead to better communication with the customers regarding bug and feature-requests that they have asked for, since it will be easier to tell each customer what release their specific request will be in.

FUTURE WORK

There were some items in the product backlog that was not implemented as part of this thesis. These features will be added to the job scheduler solution in the future. The items are listed below.

- Validation of the parameters XML with XSD.
- Multithreaded job scheduler when needed.
- Implementation of more jobs such as
    - User-import from XML-files.
    - User-import from excel-files.
- More advanced viewer with sorting and grouping on each execution.

ACKNOWLEDGEMENTS

- Netcompetence AB, for letting me complete my work.
- Joaquim Weibull, for his guidance and suggestions.
- Jerry Ericsson for guidance during my thesis work.
- Per Fahlberg for proofreading and comments.
- Janne Westberg for proofreading and comments.
- Linn Strandberg for all her love and support.
- Izabella Strandberg for not bothering me too much during my writing period.

REFERENCES

Books

[K. Burton] Kevin Burton, 2002,.*.NET Common Language Runtime Unleashed*, Sams Publishing

[J. Mayo] Joseph Mayo, 2002, *C# Unleashed*, Sams Publishing

[Newkirk, Vorontsov] James W Newkirk and Alexei A Vorontsov, 2004, *Test-Driven Development in Microsoft .NET*.

[Schwaber, Beedle] Ken Schwaber and Mike Beedle, 2002, *Agile Software Development with Scrum*, Prentice Hall

[Schwaber] Ken Schwaber, 2004, *Agile Project Management with Scrum*, Microsoft Press

[Martin, Martin] Robert C Martin and Micah Martin, 2007, *Agile Principles, Patterns, and Practices in C#*, Prentice Hall

[Boehm, Turner] Barry Boehm and Richard Turner, 2004, *Balancing Agility and Discipline*, Addison Wesley

[MS 1] Microsoft, 2003, *Developing XML Web services and server components with Microsoft Visual Basic .NET and Visual C# .NET*, Microsoft Press

Internet

[W3C CSS] Bert Bos

http://www.w3.org/Style/CSS/

2007-04-30

[MOZ JS] http://developer.mozilla.org/en/docs/About_JavaScript
2007-05-10

[MSDN FCL] http://msdn2.microsoft.com/en-us/library/hfa3fa08.aspx
2007-05-20

[MSDN REMOTING] http://msdn.microsoft.com/library/default.asp?url=/msdnmag/issues/02/10/netremoting/toc.asp
2007-05-22

[AGILE] http://www.agilemanifesto.org/

2007-07-29

[SCRUM] http://en.wikipedia.org/wiki/Scrum_(rugby)

2007-07-29

[ASSD] http://www-users.cs.york.ac.uk/~paige/Writing/xp2004.pdf

2007-10-13

APPENDIX A: ABREVIATIONS

AppDomain	Application domain.
ASP	Active server pages.
ASP.NET	Active Server Pages for .NET
CLR	Common Language Runtime
CSV	Comma separated values
DOM	Document Object Model
FCL	.NET Framework class library
GUI	Graphical User Interface
I/O	Input / Output
log4net	Open source framework for logging to multiple destinations
SOAP	Simple Object Access Protocol
XML	Extensible markup language
XSD	XML Schema Definition
XSLT	Extensible style sheet language transformation
web.config	A configurations file for ASP.NET web applications

APPENDIX B: *ISCHEDULEDJOB* INTERFACE

The IScheduledJob is the interface that all assemblies who wants to be called by the job scheduler have to implement.

```
public interface IScheduledJob
{
        string Run(XmlDocument parameters);
}
```

APPENDIX C: PARAMETER XML SAMPLE AND EXPLANATION

When a user adds a scheduled job from the web UI, a parameter xml is generated. The xml is used by the job to know where the web-services are installed. It is also used to figure out which job to execute. The common xml for all jobs are:

```
<?xml version="1.0" encoding="utf-8" ?>
<parameters url="url-value" processtime="time-value" />
```

The url-value tells the job which root-url to use when a job calls a web-service. The process time is the time when the job should be executed for.

Automatic send out job:

```
<?xml version="1.0" encoding="utf-8" ?>
<parameters url="##url" processtime="##processtime">
	<sendouts>
		<sendout type="##type" />
	</sendouts>
</parameters>
```

`##type` is the type of automatic send out job to execute. Possible values are ***Reminder*** or ***Certificate***.

Sample:

```
<?xml version="1.0" encoding="utf-8" ?>
<parameters url="http://localhost/ncts" processtime="2007-01-01 12:00:00">
	<sendouts>
		<sendout type="Reminder" />
		<sendout type="Certificate" />
	</sendouts>
</parameters>
```

Maintenance job:

```
<?xml version="1.0" encoding="utf-8" ?>
<parameters url="##url" processtime="##processtime">
	<maintenancejobs>
		<maintenance type="##type" />
	</maintenancejobs>
</parameters>
```

`##type` is the type of maintenance job to execute. Possible values are ***UpdateECourseStatistics*** or ***InactivateUsers***.

Sample:

```
<?xml version="1.0" encoding="utf-8" ?>
<parameters url="http://localhost/ncts" processtime="2007-01-01 12:00:00">
        <maintenancejobs>
                <maintenance type="InactivateUsers " />
        </maintenancejobs>
</parameters>
```

User import job:

```
<?xml version="1.0" encoding="utf-8" ?>
<parameters url="##url" processtime="##processtime">
  <imports>
    <import>
      <settings header="##header" />
      <file filetype="##filetype" separator="##separatorchar" path="##filepath"/>
      <fields>
        <field id="##filedID" index="##index"/>
        <field id="##filedID" levels="##levels" index="##levelindex "/>
      </fields>
    </import>
  </imports>
</parameters>
```

Importerar användare från fil. Filen kommer från en extern källa t.ex kundens AD.

`##header` indicates if the first row in the file is a header row. (true/false)

`##filetype` type of file to import. Only supports csv-files in this version. (csv)

`##separatorchar` separator character between the columns in the file. Possible values are ”tab” or a single character.

`##filepath`, path to the file that will be imported.

`##filedID`, StudentData property namn.

`##index`, position of the column in the file. (0-based).

`##levels`, number of columns of this type. Used when importing groups.

`##levelindex`, first column index for this column type group. (0-based).

Sample:

```
<?xml version="1.0" encoding="utf-8" ?>
<parameters url=" http://localhost/ncts" processtime="2007-10-12 12:00:00" >
  <imports>
    <import>
      <settings header="true" />
      <file filetype="csv" separator="tab"          path="C:\Temp\Import.txt"/>
      <fields>
        <field id="FirstName" index="0"/>
        <field id="LastName" index="1"/>
        <field id="LoginName" index="2"/>
        <field id="Student" index="3"/>
        <field id="Administrator" index="4"/>
        <field id="Domain" levels="5" index="5"/>
```

```
        <field id="Role" levels="5" index="10"/>
        <field id="CompanyName" index="15"/>
      </fields>
    </import>
  </imports>
</parameters>
```

APPENDIX D: PRINCIPLES BEHIND THE AGILE MANIFESTO FOR SOFTWARE DEVELOPMENT

The *Manifesto for Agile Software Development* has inspired a list of principles which describe the difference between agile practices and heavyweight processes. The principles are listed in this appendix.

"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 behavior accordingly."