WORK TERM REPORT 2 COMP 3999A

FOR THE FALL 2013 PLACEMENT AT NAKINA SYSTEMS – NETWORK INTEGRITY

Ankush Varshneya (100853074)
AnkushVarshneya@cmail.carleton.ca
Manager: Greg Johnston (Development/Design)

Ankush Varshneya
415 Cresthaven Drive
Nepean ON,
K2G 6Z4

Monday, December 16, 2013

Dr. Irwin Reichstein
School Of Computer Science
Carleton University
Ottawa, Ontario
KIS 5B6

Work Term Report 2 COMP 3999A

Dear Sir,

This is my second work term report and was prepared in connection with course COMP 3999A for my Co-op placement at Nakina Systems – Network Integrity. I worked as a Developer in the design team of the development department under the supervision of my manager Greg Johnston and my mentor Benjamin Boudreau.

Nakina Systems provides a host of network integrity management solutions to the telecommunications industry worldwide. With the large variety of network platforms and devices available today, it can become difficult to communicate with a network consisting of various devices. Nakina Systems provides solutions to easily communicate with cross platform network elements, all from one secure and centralized group of products.

This report highlights some of the key concepts I learned during my placement as well as my contributions to Nakina Systems products. During my second term I mainly dealt with changing parts of the back-end by developing internal API's and the code they relied upon to fulfill consumer's requests for new features as well as to implement the product requirements I gathered during my first term; which were in the form of mockups and functional details documentation. To complement my learning I was given software defects to fix, which touched upon various sub products.

This report has been prepared and written by me and has not received any previous academic credit at this or any other institution. I would like to thank Greg Johnston (Software Design & Development manager) for helping in reviewing this report.

Sincerely,
Ankush Varshneya
Student number 100853074

ABSTRACT (EXECUTIVE SUMMARY)

I was hired by Nakina Systems as a Developer to help implement a new log management system for a future release of Nakina's platform software. This report begins with an introduction to Nakina Systems, and then goes on to document the key concepts I learned about in the development cycle at Nakina Systems.

During my first term I was assigned the responsibility of gathering requirements from the current users of the Log Manager application for what they would like to see in a new release of the Log Manager application. From the requirements and discussions, I developed conceptual User Interface designs (UI mockups) and design documents that meet the needs of clients and internal users of the Log Manager application.

Continuing on to my second term, I primarily dealt with implementing the requirements I gathered from a back-end prospective. The back-end work consisted of reorganizing the current structures of the logs into a more generalized structure. While working on the back-end, I also took on the responsibility of implementing some requested features that were closely related to the Log Manager application.

In addition to documenting and implementing features related to the Log Manager application, this document also includes details of some of the software defects I was assigned to fix, some of the minor features implemented, and the problems encountered while doing so.

This report also highlights my work experience, day to day activities, how I applied my academic knowledge in my position and further my knowledge by getting work experience, different problems faced at the work, and the type of approaches I used to tackle those problems.

Lastly this document elaborates on how I related my work experience to my academic studies, my accomplishments, and any abbreviations used in my work term report.

Acknowledgments

I would like to thank my mentor Benjamin Boudreau for helping me make the transition from a school environment to a work environment, for helping me understand the underlying technologies used by Nakina Systems Products and lastly for guiding me throughout the term.

I would also like to thank my manager Greg Johnston for helping me with developing my FS (Functional Specification) documentation and for helping me throughout the term with various tasks.

I would like to thank Benjamin Boudreau, Paul Han and Luke McDougall for helping me solve software and performance related defects.

Lastly I would like to thank Zoë Humphries and Trelaine Farrow for verifying some of my work and raising bugs accordingly.

Table of Contents

INTRODUCTION

Nakina Systems is a company based in Kanata. It is a leading manufacturer of telecom network management systems and provides network integrity management solutions to telecommunication industries worldwide. Nakina also provides products that enable service providers to more quickly and cost effectively deliver new services in a multi-vendor network infrastructure. Recently Nakina Systems won a prestigious COMET award. Nakina's customers include major telecom providers in the US, Canada, Europe and Africa.

Its product range includes NI-Guardian, a Centralized Identity Management and Single Sign On solution for networks that centralizes security administration, NI-Collector, an Inventory Discovery and Reconciliation solution, NI-Controller, a gold standard configuration audit solution. The above products work on NOS (Network Operating System) otherwise known as NI-Framework, which rely on Oracle Database and Fusion Middleware at the backend. In a high level detail the products are split into three tiers, Web tier, Business Logic tier and the Network Agent Tier. The Web Tier hosts the actual web applications and the front-end. The back-end and the middleware used are a part of the Business Logic tier. Lastly, The Network Agent Tier is what links the products with the devices otherwise known as network elements. This 'three dimensional' tier solution allows for scalability, availability, resiliency and reliability in all of the products. Using the three tiers allows a user to easily communicate between multiple Network Elements; this may be really hard to do manually as the Network Elements may have very different methods of communication.

Nakina Systems' research and development organization consists of three main departments which are Development, Verification, and Support. The development department consists of a design team, framework team, application team and a device adapter development team. The role of the support department is to provide support to Nakina Systems customers. The verification department consists of a quality assurance team of product testers that thoroughly test products before a major release, to ensure the product is defect free and up to the client's satisfaction. Any discrepancies gathered by verification or directly by support (via a customer) are reported to the development department through an internal system called JIRA, an electronic product tracker. A JIRA is not only limited to bugs but it may also contain request for new features, JIRA's are then usually assigned to a team in the development department based on the product it deals with. The job of a developer is to address JIRAs and fix defects or implement new features. After a JIRA has been addressed, it is then reassigned to a member of the verification department so that they can take any appropriate action such as further testing.

WORK EXPERIENCES

Each development team has weekly SCRUM meeting and is arranged by the team managers. This is where developers discuss their weekly progress in terms of what they are working on and if they need help from fellow developers. Additionally, new products and feature implementations are also discussed here. These meetings are a great opportunity to find out what's going around you and what your fellow colleagues are working on and it gives you a great understanding of the products and to follow up on any questions you may have. Occasionally some developers attend other teams' SCRUM meetings as well, as they may be working on a product that requires ongoing discussions with multiple teams; this was the case for me when I was working on the Log Manager related work tasks. Sometimes SCRUMs are also used to give business updates, such as contracts the company may have won or acquired.

Tools

Working on Nakina Systems products involves heavy use of JAVA programming based on JAVA J2EE architecture and OOP design principles. Its IDE consists of Eclipse and is integrated with Apache Ant to compile the code and Apache Subversion SVN/CVS to help with software version control.

Since Nakina Systems' product range consists of web applications, one would think that the application relied more on Java Script but this is not the case here. Java is used due to its object oriented design and use of coding. It is much easier to track bugs and implement new code in a strongly typed language such as Java.

From a front-end (UI) prospective web frameworks are used to compile the application from Java into Java Script. Traditionally Apache struts web framework is used in Nakina Systems application, but with the newer applications Nakina Systems has moved towards a Google Web Tool Kit (GWT/GXT) solution.

From a back-end prospective Java Persistence API (JPA) and the Hibernate framework (an object-relational mapping framework for Java) are used to query the database to retrieve and save information. Additionally, Oracle database is used for the back-end store, SQL Developer is used as a testing tool and Oracle Web Logic server is used to deploy the web applications/products used by Nakina Systems.

Application programming interfaces (API's) are used as a mediator to connect the back-end to the front-end. A user action from the front-end sends data through API's to the back-end from which the back-end can communicate that information to the database.

Nakina Systems uses an internal Wiki where developers and product testers post guides, code syntax, and other various snippets of useful proprietary information for some of the technologies used at Nakina Systems. The information provided by the Wiki helps newcomers to familiarize themselves with the large product range.

Objectives and Contributions

Log Manager

The main objective for me in this term was to improve the existing Log Manager. The main objective of working on Log Manager is to give an exposure to the full software delivery life cycle in four steps (that is requirements, design, verification, and delivery of the product).

Requirement Phase

In my previous work term I worked on the requirement phase in which I had to arrange meetings with the key users of the Log Manager application to gather what they would like to see in a future release of the Log Manager application. Following this, I generated various UI designs via mockups using Balsamiq Mockups software. These mockups were then shown and discussed with my manager, mentor, support users, and other selected people that used the application to allow for better requirement feedback. Ultimately, this lead to the writing of functional specification document (FS), which is a document that explains the proposed plans from a development prospective as well as to address new requested features.

Design Phase

This term I started with the design phase, by implementing my changes from a back-end prospective. The Log Manager consists mainly of four types of logs, User Logs, System Logs, Security Logs, and NE Request Logs; all other logs are a subclass of one of these logs. As proposed in the FS, my first step was to change the structure of these logs to make them more generalized. My main approach was to look at the meaning of each field of the logs and try to come up with a generalized field, for example the 'User Action' field of User Logs and the 'Activity' field of Security Logs and NE Request Logs were generalized into an 'Activity' field which was shifted to an abstract base class. This type of generalization had to be applied to all the Search Parameter Objects, Business Objects (BO), Info Objects and Value Objects.

The BO's contain all the fields that are (or map to a field) in the database, along with the help of Java annotations and Java Docs. The persistence framework helps map the fields from the BO to the database as well as generate Transfer Objects (TO) and Search Criteria Descriptors (SCD). With the help of the Hibernate Framework these objects can be used to interact with a database. The Search Parameter Objects with the help of the SCD objects are used by Hibernate to query for data. The TO's are used to load the BO's with data or to save the data from a BO into the database.

Changing the BO's also meant changing the schema of the database to match the changes in the BO's. This meant writing an amender with SQL queries to change the schema. Unfortunately, updating millions of logs to follow a new schema is very slow. This led me to explore a bit about how databases work. SQL (Standard Query Language) is broken into two sub languages DML (Data Manipulation Language) and DDL (Data Definition Language), the latter being more efficient for the purpose of what I needed to do. So I created a new table with the data of the old table using a CREATE TABLE AS

SELECT query along with a DECODE function to move the data in a much faster way. To further increase the performance, I looked into using parallelism to split the operation into threads which showed obvious performance gains. Info objects are used within the back-end to transfer data between the frameworks.

The Value Objects and Search Parameters are used by the API layer of the back-end, so changing these meant that the API's needed to be changed to use the new generalized Log structure. Here I used polymorphism to replace 4 different API' calls into one which could just use the base class Value Objects and Search Parameters and then type cast to the proper derived class. Another benefit of polymorphism was that I could make the older API's calls use the new API call by type casting the objects and this allowed the API to remain backward compatible. This is crucial when you are working with such a large system as you want to minimize, if not completely eliminate, the side effects that may be caused by code change.

By generalizing the Log structure and using OOP principles, I was able to better reuse code and make the code as simple to understand as possible. The utility classes, APIs, and the Log Manager class's code were really simplified due to the fact that I could just send around an abstract Log Object with a lot of common fields. Rather than having large segments of code per log type I could have one set of code to deal with generic logs. Logs with specific fields only needed to have code written to work with the specific fields rather than every single field.

Aside from working with the Log Manager, I was also assigned to work on related feature requests such as support for CEF (Common Event Format) syslog. Syslog is a standard for transferring logs generated by a system to an external third party source. CEF is a standard set by HP ArcSight for formatting logs to include a vendor name, product name, version, severity and all other information as a string of extensions. Developing this feature taught me about what syslog is and how it can be incorporated into a system to allow logs to be transferred to a third party software. Implementing the configuration taught me about the transport protocols such as UDP and TCP and what their advantages and disadvantages are and how they are used and implemented by Java. The configurations for the syslog were parsed from an xml file which helped me to develop my understanding of how java can work with xml files to store configuration data rather than relying on a database to do it.

Verification Phase

As part of the verification phase I had to test all my work to detect any errors, since the API's are at the top layer in terms of the back-end these are what is needed to be tested. If they worked it means the rest of the underlying code worked as well. Before starting the changes I had observed the common behavior of the retrieving and saving of logs and had an idea how everything should work. After implementing all the changes and adding a new API I had to make sure it worked as it had previously. Using SAMMIE scenarios (Unit testing for API's), to check the validity of the API I introduced, as well as using the older APIs to test for backward compatibility. Writing the test cases challenged me to learn new concepts with which I was not previously familiar.

After testing my changes, it was time to release my changes and allow the members of the Verification department to further test my changes and raise bugs accordingly. This process allowed me to interact with the members of the Verification department and get to know how they contribute to Nakina; it also gave me an insight into how bugs are communicated and tracked in their internal JIRA system (see last paragraph of Introduction section above).

Delivery of Product Phase

Recently the new release (version 9) has shipped which contains the new features and changes I addressed.

Software Defect Fixing

In parallel to working on the Log Manager I was also assigned the task of fixing software defects. Throughout this process I really developed a greater understanding of how the products work. I had the opportunity to see the tools and technologies in action, as mentioned in the tools section. I also got an understanding of how the development department is organized and what developers deal with what part of the products as well as understanding how their bug tracking system works. It helped me develop my understanding of their software, the way it is written, and the guidelines followed in terms of the UI and the API development.

Towards the last two months of this work term I was assigned performance related defects. These defects involved working more with the back-end with a little bit of front-end UI work. I learned a lot about the frameworks that were used and how they worked as described in the tools section above. While working with the back-end, I acquired a great understanding of the different layers of the back-end as described in the Log Manager Task Section above. Some of these bugs required analyzing the flow of data within different tiers of the system. For example a cloning operation was making a back-end call to retrieve the original data to the front-end, and then the front-end was cloning the data and sending it to the back-end to be saved not to mention that the front-end then contacted the back-end to reload the newly cloned data. Although this was an acceptable solution, the frequent communication between the front-end and back-end made the solution difficult to scale to large amounts of data. Rewriting the API to do the cloning in the back-end by sending the targets name and then return the cloned results allowed for a scalable solution.

Fixing the defects involved a large variety of code written by many writers. Looking at such a variety of code helped me explore a variety of coding styles and practices and in turn influenced my own style by a great degree. Sometimes certain code segments lead me to interact with other developers, which allowed me to explore how other developers contribute to Nakina Systems products. Working with the back-end helped me gain API development skills and learn about new technologies as well as improve my existing skills on other technologies.

Challenges and Solutions

During my defect fixing phase I encountered defects that were interrelated to many products, so it was very confusing as to where to start from. My mentor provided me with some quick walkthrough tutorials of their code and appropriate articles to read on their internal Wiki, which complemented his tutorials. After reading the Wiki articles I had a better understanding of where I needed to start looking (code wise) to fix the bug. Some of the tips and techniques suggested by my mentor helped me greatly increase my productivity and knowledge.

While debugging I would always try to follow up with my mentor and explain to him the logic behind, how I was approaching fixing the bugs. Discussing the bugs helped me get feedback, so I could stay on the right track. As part of my debugging strategy I used truth tables and logical statements to help better discuss my findings with my mentor as well as to help pin point bugs. I was able to make use of the GWT debugger and the debugger provided in eclipse to better find bugs. Using the SQL developer, SAMMIE tests (Unit testing for API's), and JAMon (Java Application Monitor) I was better able to pin point time related defects.

Solving performance related defects was very difficult at times as there were so many ways in which performance can be compromised, it required a great deal of analyzing. Luckily I was able to get a lot of suggestions from my team members which helped guide me in the right direction.

Developing and changing the API's and working with the back-end were very involved processes as there are so many layers to go through. I was able to use UML diagrams to help me learn about the system and know what kind of changes needed to be made to implement my changes. I was able to efficiently analyze the code using shortcuts in eclipse, which helped me to get a better grip of what needed to be done and how.

Reflection

Reflecting back on the term, it was a fantastic learning experience for me. I learned about a formal software cycle that is present at Nakina Systems. Attending weekly SCRUM meetings helped me with getting the bigger picture of how organizations such as Nakina Systems work at large.

I learned a great deal about the Java Platform and, how the Enterprise Edition architecture work and how it is structured as well as how everything is layered from the database all the way up to the API's. Compiling the project is done through the terminal which helped me develop my Linux bash skills. The projects themselves are hosted on Apache Subversion (SVN) which is a version control mechanism used to maintain different products and their various releases conveniently organized by 'branches'.

Working with the back-end and parts of the front-end has taught me about the frameworks involved and how they work together across the three tiered environment.
I learned how the Java Persistence API (JPA) and the Hibernate framework are used in an application to easily query the database. The TO's, SCD and Search Parameter objects are used to allow saving and retrieval of information to and from BO Objects. I learned about tools such as SQL developer which can be used to interact with the database outside of the actual application. Simplifying the log structure helped me learn the importance of OOP concepts like inheritance and how it can really simplify the general flow of code and the application. Furthermore, working with the Log Manager and software defects helped me to understand how the applications are structured; it taught me a lot about API development and the side effects of changing something like the log structure.

Aside from developing my Java skills I have also been able to further my knowledge about databases and how they work. I have learned how SQL is split into DDL and DML, how to distinguish between the two and when to use one over the other to improve the database transaction. Additionally, I have learned about how databases such as Oracle can make use of parallelism to split large operations into threads, hence improving the performance greatly when it comes to changing the database through a SQL amender. Moreover, implementing syslog support for the logs helped me understand the importance of syslog and how the Linux operating system uses syslog to generate important information about the system. I learned about syslog daemon and how syslog can be generated from a java framework. I also learned About CEF formatted logs. The heavy use of the terminal while compiling, deploying products and viewing the logs has helped me to improve my Linux bash skills.

Working at Nakina Systems helped me understand that web applications don't have to be restricted to just languages such as JavaScript or PHP. Languages such as Java can be used with the help of frameworks to create equally powerful and robust web applications as their counterpart to other weakly typed languages. Adding and fixing code in Java is much easier and more organized as opposed to a language like JavaScript.

Fixing software defects helped me understand the importance of defect/project tracking software such as JIRA and how organizations such as Nakina Systems use it as a medium to communicate between their various teams to better provide solutions to their clients. I also learned how version control software, such as Apache's Subversion, is used to maintain different versions of software. I also learned key version control principles such as branching software to fit needs of different clients, merging and propagating changes among branches.

Relation to Academic Studies and Career Development

This placement helps me enhance the learning I received at university. It helped me see some of the software engineering concepts such as design patterns in action in real life programs. It helped me develop my skills in Java, SQL, Linux and so much more.

Working with so many types of objects and their hierarchies needed a great deal of organization to understand and communicate to my mentor and other members of my team. For this, I found myself using tools that I learned about in Software Engineering courses. Using UML diagrams I was able to organize and communicate my thoughts, it made understanding the inner workings of such a large system quite easier than it would have been otherwise.

Furthermore, while solving software defects I found myself using techniques acquired from discrete math and logic courses that I would not have expected to use for those applications. For example, I was using truth tables and logical statements to help pin point bugs that would have otherwise taken much longer to solve. Hence I was able to communicate my bug findings to my mentor much more clearly with the use of logical statements and truth tables than I would have been able to do otherwise. To improve performance I was using my experience of analyzing the performance complexity based on what I learned from data structure courses. It allowed me to connect contents learned from non-programming courses into my coding skills and helped develop my analytical, debugging and coding skills.

Debugging helped me work on the pre-existing material, which is really hard to come by in a course offered at university. Since the pre-existing code has been written by many developers, going through it helps one to really understand the new styles and gives you a variety of coding practices and it helps you build your own style in the process. Solving defects really helps in learning trouble shooting tips, which are easier to learn in a work scenario. In contrast the courses I have been through in university required you to write your own code with the exception of 1 or 2 courses, where there is very little code written by the professor or teaching assistant, which doesn’t offer as large a variety.

While doing group work for courses, the work ends up being divided unequally at times. For example in a group, one member may have expertise in documenting mockups and UML diagrams, hence they may end up doing a good chunk of that work while another member focuses on another aspect. Here at Nakina, I was able to get a good experience with each of the aspects. When I first began going in to the coding side of my term it was really overwhelming at times dealing with such a large system. At first I did not feel confident making changes, let alone submitting them. I would usually review them with my mentor before submitting. With the help of my mentor and his advice to trust myself, I was able to increase my confidence and hence be more independent to the point where I feel more comfortable working with multiple aspects of the product from a back-end prospective. This experience has given me the confidence to deal with more of the facets involved in a development cycle. I feel more comfortable with the documentation side such as Mockups and UML diagrams whereas in past courses those tasks were usually done more by my peers than me.

In conclusion, I was able to make use of some of the concepts and techniques I acquired from my studies to be able to better adapt myself to the work environment. After gaining the confident I am now able to better trust myself even when I work on different areas of the product. Working at Nakina Systems has given me a great experience as well as a new appreciation for this type of career.

ABBREVIATIONS

NOS – Network Operating System
UI – User Interface
NI – Network Integrity
OOP – Object-oriented programming
IDE – Integrated Development Environment
GWT – Google Web Tool Kit
GXT – Ext GWT
FS – Functional Specification Document
NE – Network Element
BO – Business Object also referred to as an MO
TO – Transfer object
SCD – Search Criteria Descriptor
SQL – Structured Query Language
DDL – Data Definition Language
DML – DATA Manipulation Language
API – Application Programming Interface
UML – Unified Modeling Language
CEF – Common Event Format
UDP – User Datagram Protocol
TCP – Transmission Control Protocol
XML –Extensible Markup Language
JAR – Java ARchive
EAR – Enterprise ARchive
WAR – Web application ARchive
PHP – Hypertext Preprocessor
SVN – Apache Subversion-
CVS – Concurrent Versions System
JAMon – Java Application Monitor