Zeus Final Project Report

*Query Scripting Tool*

SD-May1020

Kristina Gervais

Alex Kharbush

Cole Anagnost

Advisor

Dr. Suraj Kothari

4/28/2010



Table 1: Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Change** |
| 0.10 | 01/26/10 | KG | Initial Document – merged Project Plan & Design Document |
| 0.11 | 04/20/10 | KG | Updated sections 1 & 2 |
|  |  |  |  |
|  |  |  |  |

1. Introduction 1

1.1. Purpose 1

1.2. Scope 1

1.3. Acknowledgement 1

1.4. Term, Acronym, and Abbreviation Definitions 1

1.5. Executive Summary 2

2. Overall Description 3

2.1. Problem Statement 3

2.2. Operating Environment 4

2.3. Intended Users & Intended Uses 4

2.3.1. Intended Users 4

2.3.2. Intended Uses 4

2.4. Assumptions & Limitations 4

2.4.1. Initial Assumptions 4

2.4.2. Initial Limitations 4

2.5. Market & Literature Survey 5

2.6. Expected End-Product & Other Deliverables 5

2.6.1. End-Product Deliverables 5

2.6.2. Administrative Deliverables 5

3. Proposed Approach & Statement of Work 5

3.1. Proposed Approach 5

3.1.1. Functional Requirements 5

3.1.2. Constraints Considerations 5

3.1.3. Technology Considerations 6

3.1.4. Technical Approach Considerations 6

3.1.5. Testing Requirements Considerations 8

3.1.6. Security Considerations 8

3.1.7. Safety Considerations 8

3.1.8. Intellectual Property Considerations 9

3.1.9. Commercialization Considerations 9

3.1.10. Possible Risks & Risk Management 9

3.1.11. Project Proposed Milestones & Evaluation Criteria 9

3.2. Statement of Work 10

3.2.1. Task 1: Problem Definition 10

3.2.2. Task 2: Technology Considerations and Selection 11

3.2.3. Task 3: End-Product Design 12

3.2.4. Task 4: End-Product Prototype Implementation 13

3.2.5. Task 5: End-Product Testing 14

3.2.6. Task 6: End-Product Documentation 15

3.2.7. Task 7: End-Product Demonstration 15

3.2.8. Task 8: Project Reporting 16

4. Estimated Resources and Schedules 18

4.1. Estimated Resources 18

4.1.1. Personnel Effort Requirements 18

4.1.2. Other Resource Requirements 18

4.1.3. Financial Requirements 18

4.2. Schedules 19

4.2.1. Project Schedule 19

4.2.2. Deliverable Schedule 19

5. Overall Requirements Description 20

5.1. Product Perspective 20

5.1.1. Concept of Operations 20

5.1.2. User Interfaces 20

5.1.3. Hardware Interfaces 20

5.1.4. Software Interfaces 20

5.1.5. Communication Protocols and Interfaces 20

5.1.6. Memory Constraints 20

5.1.7. Site Adaption Requirements 20

5.2. Product Functions 21

5.3. User Characteristics &. Intended Uses 21

5.4. Constraints 21

5.5. Assumptions & Limitations 21

5.6. Dependencies 21

6. Specific Requirements 22

6.1. External Interface Requirement 22

6.1.1. User Interfaces 22

6.1.2. System Input & Output 23

6.1.3. Operating Platform 24

6.1.4. Software Interfaces 24

6.2. Features 24

6.3. Performance Requirements 25

6.4. Design Constraints 25

6.5. Software System Attributes 25

6.5.1. Reliability & Availability 25

6.5.2. Portability 25

6.6. Other Requirements 25

7. Design Goals 25

7.1. Correctness 25

7.2. Usability 25

7.3. Robustness 26

7.4. Efficiency 26

7.5. Maintainability 26

7.6. Extensibility 26

8. Decomposition Description 26

8.1. Module Decomposition 26

8.1.1. Control Module 27

8.1.2. Eclipse Plugin Module 27

8.1.3. Atlas Interface Module 27

8.1.4. Query Script Parser 28

8.2. Concurrent Processes & Threads 28

8.2.1. Eclipse 28

8.2.2. Graphviz 28

8.3. Data Decomposition 29

8.4. States 29

9. Dependency Description 29

9.1. Inter-module Dependencies 29

9.1.1. Atlas Interface Module 29

9.1.2. Control Module 30

9.1.3. Eclipse Plugin Module 30

9.1.4. Query Script Parser Module 30

9.2. Inter-process Dependencies 30

9.3. Data Dependencies 30

10. Interface Description 30

10.1. Module Interface 30

10.1.1. Atlas Interface Module 30

10.1.2. Control Module 30

10.1.3. Eclipse Plugin Module 31

10.1.4. Query Script Parser Module 31

11. Detailed Design 31

11.1. Class Descriptions 31

12. Test and Evaluation Plan 32

13. Modification & Maintenance Recommendations 32

14. Traceability 32

15. Conclusion 33

15.1. Project Team Information 33

15.1.1. Client 33

15.1.2. Advisor 33

15.1.3. Team Members 33

15.2. Closing Summary 34

16. Bibliography 34

16.1. Templates 34

16.2. Content 34

Apendix A – Communication Plan 35

Appendix B – Work Breakdown Structure 38

Appendix C – JavaDocs 39

List of Figures

Figure 1: System Concept Sketch

Figure 2: Project Schedule

Figure 3: Deliverable Schedule

Figure 4: Block Diagram

Figure 5: System Use Case Diagram

Figure 6: User Interface Screenshot

Figure 7: JavaScript Query Example

Figure 8: System Output Example

Figure 9: Architecture Diagram

Figure 10: System State Diagram

Figure 11: Class Diagram

Figure 12: Work Breakdown Structure

List of Tables

Table 1: Revision History

Table 2: Term, Acronym, and Abbreviation Definitions

Table 3: Personnel Effort Requirements by Task

Table 4: Non-personnel Resource Requirements

Table 5: Labor and Materials Costs

Table 6: Traceability Matrix

Table 7: Important Stakeholders

Table 8: Essential Stakeholder Approach

Table 9: Essential Stakeholders' Power

Table 10: Essential Stakeholder Support

Table 11: Stakeholder Communication Methods

# 1. Introduction

## 1.1. Purpose

The purpose of this document is to provide the reader with a detailed understanding of this project’s execution plan, requirements and specification, design, and testing and verification. The intended audience for this document is the student developers, the client the senior design instructor, the team faculty advisor, the industry grading committee, and anyone who may do follow up maintenance or enhancement on the Zeus application.

## 1.2. Scope

The scope of this document is limited to the Zeus project. It includes the project plan, project specifications, design considerations, and testing plan and results summary. This document describes the general project overview, approach and work breakdown, resources and scheduling, software specifications, architecture, detailed design, and testing plan and results for the Zeus application. Also included in the document is a communication plan which includes all project stakeholders.

## 1.3. Acknowledgement

The team wishes to express their gratitude to their advisor, Dr. Suraj Kothari, who has been helpful and offers assistance, support and guidance. In addition, the team would like to thank EnSoft representative, Jon Mathews, for his insight and help in designing the product.

## 1.4. Term, Acronym, and Abbreviation Definitions

Table 2: Term, Acronym, and Abbreviation Definitions

| **Term, Acronym, or Abbreviation** | **Definition** |
| --- | --- |
| API | Application programming interface: an interface that defines the ways an  application program can request services from a library and/or operating  system (http://en.wikipedia.org/wiki/Application\_programming\_interface) |
| Atlas | A software analysis tool developed by EnSoft to assist in the analysis of the internal structure of C source code |
| Eclipse | Eclipse is an open source community, whose projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle. (http://www.eclipse.org/org/) |
| ECprE Department | Iowa State University College of Engineering Electrical and Computer Engineering Department |
| EnSoft Corp | The client for the system as well as the company that produces Atlas |
| Graphviz | An open source visualization software from AT&T Research that has the capability to create directed and undirected graphs. |
| GUI | Graphical User Interface - type of user interface item that allows people to interact with programs in more ways than typing (http://en.wikipedia.org/wiki/Graphical\_user\_interface) |
| IDE | Integrated Development Environment - a software application that provides comprehensive facilities to computer programmers for software development normally consisting of: a source code editor, a compiler and/or an interpreter, build automation tools, and a debugger (http://en.wikipedia.org/wiki/Integrated\_development\_environment) |
| JAR | Java Archive – the aggregate of many files into one commonly used to distribute Java classes. |
| JavaDocs | API documentation formatted as a website generated from Java source code |
| JDBC | Java Database Connectivity – connectivity between the Java programming language and a wide range of databases (http://java.sun.com/javase/technologies/database/) |
| JRE | Java Runtime Environment - an implementation which includes JVM , Core libraries and other additional components needed to run applications and applets written in Java (http://www.java.com/en/download/faq/jre\_jdk.xml) |
| JUnit | An Eclipse test suite that allows for testing of java classes and methods. |
| JVM | Java Virtual Machine – the component of the JRE responsible for executing Java programs (http://www.java.com/en/download/faq/jre\_jdk.xml) |
| OSGi framework | A module system for Java that implements a complete and dynamic component model (does not exist in standalone Java/VM environments) allowing applications or components to be remotely installed, started, stopped, updated and uninstalled without requiring a reboot (http://en.wikipedia.org/wiki/OSGi) |
| plugin | A small unit of the Eclipse platform that can be developed separately (http://www.eclipsepluginsite.com/) |
| SQL | Structured Query Language – a database computer language designed for managing data in relational databases. (en.wikipedia.org/wiki/SQL) |
| SWT | Standard Widget Toolkit - an open source widget toolkit for Java designed to provide efficient, portable access to the user-interface facilities of the operating systems on which it is implemented (http://www.eclipse.org/swt/) |
| UI | User Interface – the means in which the user interacts with the computer system. |
| Zeus | An Eclipse plugin that supports composing Atlas queries, executing the composed queries, and storing the queries in a non-volatile form for future use |

## 1.5. Executive Summary

This document covers aspects of the Zeus software system requirement and design including the overall requirement description, specific requirements, design goals, decomposition description, dependency description, interface descriptions, detailed design, technical approach consideration and results, modification and maintenance recommendations, test and evaluation plan, and traceability. The overall requirement description discusses the product perspective, the product functions, constraints and dependencies. The specific requirements describe the external interfaces, especially the user interface, the features, and non-functional requirements such as performance requirements, design constraints, software system attributes, and other requirements.

The design goals include the system correctness, usability, robustness, efficiency, maintainability and extensibility. The Decomposition Description explains the breakdown of the software system into components consisting of four modules, Atlas Interface, Control, Eclipse Plugin, and Query Script Parser, two processes, Eclipse IDE and Graphviz, multiple data structures and two states. These components have dependencies amongst themselves, which are explained in the Dependency Description. How these components work together is explained in the Interface Description, and the Detailed Description explains the Java classes that will make up the Zeus system. A brief overview is provided of the testing plan and modification recommendations. Lastly, this document includes a matrix to track that all requirements are paired with code and/or documentation as well as testing.

This document covers the plan for the Zeus software system project. A project overview is provided, followed by a proposed approach and statement of work.  The required resources and project schedule are then explained. A communication plan for the team is also enclosed in this document.  The project overview consists of the need for the project, the operating environment, the intended users and uses, the assumptions and limitations, and the deliverables.  The proposed approach discusses the functional requirements and constraints.  It also contains the technology, technical approach, testing requirement, security, safety, intellectual property, and commercialization considerations for the project.  Lastly this section addresses the project risks, milestones, and tracking procedures.  The statement of work is a breakdown of the project into eight tasks: problem definition, technology considerations, end-product design, end-product prototyping, end-product testing, end-product documentation, end-product demonstrations, and project reporting. The required resources are broken down into three categories: personnel effort requirements, other resource requirements, and financial requirements.  The schedule, or time requirements, is divided into a project schedule and a deliverable/milestone schedule. Lastly, the communication plan contains the means and the frequency each stakeholder is contacted for the project.

# 2. Overall Description

## 2.1. Problem Statement

Semantic analysis of large software systems is a daunting task without the use of tools. Developers and testers alike would benefit immensely from having a tool that could analyze the source code based on customizable scripts. Benefits include time, and ultimately financial savings as well as higher code understanding which leads to higher quality software. Additional cost savings would occur if those scripts could be stored for future use and/or transferred to other systems.

Atlas, current semantic analysis tool, is an Eclipse plugin that analyzes the internal structure of C source code. Atlas has a domain specific language for querying source code. It is not a general-purpose language, and therefore, lacks constructs such as loops and arrays. Atlas provides support for saving query results into internal memory for use in future queries within the same session; however, the tool does not support storing composed queries in a non-volatile state for use in later sessions. In addition, Atlas does not support the scripting of query function calls. Due to the limitations of Atlas, complex code requires manual analysis resulting in wasted time and a poorer understanding of the code.

The goal is to construct queries using a general-purpose language to enable more complexity in software analysis using Atlas. It is analogous to how JDBC handles the interaction between Java and SQL such that the user will be able to send Atlas query statements and process the results. The student team will build a second Eclipse plugin, called Zeus, which will support the scripting of Atlas queries. The Zeus application will also support storing queries in a non-volatile form to enable use during future sessions. To do this, Zeus will parse a query script input file, interpret the query script, and execute Atlas API calls. The users will also have the ability to load and edit their query scripts in the Eclipse IDE. In order to verify the accuracy and correctness of the application, scripts will be written to automate query compositions used in previous research done with Atlas. Figure 1 shows the concept sketch for the software system.

Figure 1 - System Concept Sketch

## 2.2. Operating Environment

Since this is strictly a software project, the product will not be subject to environmental concerns such as temperature, dust, etc., and the system design does not consider these factors.

## 2.3. Intended Users & Intended Uses

### 2.3.1. Intended Users

Undergraduates and graduate students with software engineering, computer engineering, and/or computer science experience will be the primary users of Zeus. Undergraduate students will be completing or have completed their third year of coursework, and therefore all users have an intermediate to advanced understanding of software architecture, design, and testing concepts. The users will primarily be using the tool for research purposes; however, users may potentially use it to assist with coursework.

### 2.3.2. Intended Uses

Zeus will not be used as a standalone application. It will be used as a supplement to EnSoft’s Atlas tool to analyze source code written in C of sizes larger than 5,000 lines. Atlas only supports the analysis of source code written in C, and small code segments do not benefit from Atlas’s analysis capability as much as larger code segments. The query editor capability will be used with the intent to create and edit scripts to perform Atlas functionalities.

## 2.4. Assumptions & Limitations

### 2.4.1. Initial Assumptions

The end-product will not be used outside of the United States.

The end-product will be a standalone application (does not support simultaneous users).

EnSoft will deliver Atlas APIs in November.

Atlas API calls will be possible.

### 2.4.2. Initial Limitations

Project shall be completed in May, 2010.

System shall be implemented as an Eclipse plugin (client requirement).

System shall interface with Atlas (client requirement).

System shall display script query results as an Atlas graph or artifact list (client requirement).

System shall have the ability to compose queries (client requirement).

The cost to produce the software system shall not exceed two-hundred and fifty dollars (budget constraint).

## 2.5. Market & Literature Survey

Due to the research nature of the project, no market or literature survey has been done.  The Atlas software analysis tool is still in its development stage and has not been released for public use.  The intention of this project is not for commercialization, but for advancement in software evolution and maintenance research.

## 2.6. Expected End-Product & Other Deliverables

Eight deliverables will be created for this project, one of which will be a software application. The rest of the deliverables will be documentation supporting the software project, development, testing, or the software itself. The deliverables can be split into end-product deliverables and administrative deliverables.

### 2.6.1. End-Product Deliverables

Zeus Eclipse plugin: the end-product of the project

User Manual: a printable guide for the user

### 2.6.2. Administrative Deliverables

Project Plan: a guide for managing the student project

Specifications & Design Document: a guide for managing the development of the project

Final Project Report: a guide that encompasses the entire project, especially the test plan.

Project Poster: an overview of the project used to describe the project to the public.

Website: a document providing the public with a description of the project.

Weekly Update Emails: project status reports.

# 3. Proposed Approach & Statement of Work

## 3.1. Proposed Approach

### 3.1.1. Functional Requirements

System shall execute script queries as Atlas queries.

System shall provide the user the ability to store query scripts.

System shall provide the user the ability to create query scripts.

System shall provide the user the ability to edit query scripts.

System shall compose queries (filtering).

System shall project queries

System shall display query script results as an Atlas artifact list.

System shall display query script results of Atlas artifacts in a table layout.

System shall display query script results as an Atlas graph.

### 3.1.2. Constraints Considerations

System will be written in Java.

System will function as an Eclipse plugin.

System must integrate with the Eclipse development platform.

System must integrate with the Atlas code analysis utility via use of Atlas APIs.

### 3.1.3. Technology Considerations

#### 3.1.3.1. Script Storage

The end-product must support storage to enable future script use. The methods of storage considered are a database or a file system.

Approach: Use a file system to store scripts.

Pros

Blah blah

Cons

Blah blah

Approach: Use a database system to store scripts.

Pros

Blah blah

Cons

Blah blah

Result: A file system will be used to store query scripts due to….

### 3.1.4. Technical Approach Considerations

#### 3.14.1. Source Code Management

Source code management increases the effectiveness of collaborative development. This technique allows the team to work together on the same code with minimal complications through the use of technology.

Approach: Use a source control program to check in and checkout code

Pros

Code is maintained in hierarchy structures

All programmers will have access to code.

Code submissions will be timed stamped.

The code can easily be timed stamped.

Cons

Need to setup a server for this to happen

Users not familiar with system

Approach: Email the code to each other and maintain code on each programmer’s computer

Pros

Simple to do

Quick to implement

Cons

Code structure is not saved

Cannot easily roll the code back

Code is in many places, could get complicated

Approach: Save the code to a Google doc

Pros

Code is one place

All have access to it

Roll back ability

Simple

Cons

Code structure is not saved

Result: A source control server will be user enabling the team to develop code from multiple locations.

#### 3.14.2. Coding Strategies

Multiple coding strategies can be employed when creating a collaborative project.

Approach: Break the system down into use cases and assign each use case to a programmer.

Pros

Makes the code manageable for one user to focus on specific sections

Works well with iterative approach

Splits the work so programmers can work in parallel

Cons

One programmer may not know how code works outside of their assigned cases

Still need some initial programming to be done to get the program at a baseline

Approach:All work together at the same time in a lab or through screen sharing.

Pros

All programmers know how all the code works

Many views on the same code allowing for best coding to happen

Cons

Very slow

All have to be together to work on code

Approach: Assign a lead programmer that does most of the coding and assign smaller, easier tasks to others.

Pros

One person knows how everything works

Some tasks are subdivided

Cons

Risk with only one lead programmer

Result: The system has been broken into different modules and different modules will be assigned to each programmer. This will allow parts of the project to be done in parallel and at multiple locations.

#### 3.14.3. Development Process

Blah blah blah

IterativeApproach: start with a limited version of code and build it up over time.

Pros

Simple

Shows progress

At the start of each iteration modules are working together

Cons

May get stuck on one iteration for a while

V FormApproach: design and implement the code in one iteration with reviews when needed

Pros

Allows for reviews

Design of code will be used

All have tasks to do until project is done

Cons

A large quantity of work

Will not get it working until everything is complete

Hybrid Approach: design the code and implement a small baseline to create a simple version from which to begin an iterative coding process.

Pros

Blah

Cons

Blah

Result: We will use the Hybrid approach of both the Iterative approach and V form. This will allow us to get a baseline up quickly and then break the rest of the project into smaller chucks so each programmer can work on their parts.

### 3.1.5. Testing Requirements Considerations

#### 3.1.5.1. Unit Testing

Unit testing will be performed throughout development.

#### 3.1.5.2. Integration Testing

Integration testing will be performed as separate components of the project are prototyped.

#### 3.1.5.3. System Testing

System testing will be performed after project components are integrated, in particular before a product demonstration.

### 3.1.6. Security Considerations

#### 3.1.6.1. Project Security

None

#### 3.1.6.2. Product Security

None

### 3.1.7. Safety Considerations

None

### 3.1.8. Intellectual Property Considerations

We as a team do not understand the intellectual property restrictions placed on the project by Iowa State University Electrical and Computer Engineering Department.

### 3.1.9. Commercialization Considerations

None. This system is designed with the intended use of academic research.

### 3.1.10. Possible Risks & Risk Management

#### 3.1.10.1. Team Member Loss

One of the major risks posed to any student project is the loss of a team member. This team member loss risk includes, but is not limited to, the following situations: member stops communicating with the rest of the team, member drops senior design class, member stops attending Iowa State University, etc. The impacts of the loss of a team member include additional work for the rest of the team, loss in knowledge of the project and end-product, loss of the end-product itself, etc. Methods to minimize the impacts of this risk include developing and executing an effective communication plan, keeping project and product documentation up to date, storing all project materials (documentation and source code) on a shared subversion control system (SVN), and participating honestly and openly during weekly updates (via e-mail or team meetings).

#### 3.1.10.2. Acquisition of Atlas APIs

Another major risk to this project is not obtaining the accurate and complete Atlas APIs from EnSoft Corp. by November, 2009. To minimize the risk status updates are provided to the EnSoft Corp. representative, who is also the faculty advisor. Also, status updates on the Atlas API delivery are requested from the company representative. Because the objective of this project is to create a query scripting tool to interface with Atlas, the failure to obtain the APIs would have a severe impact on the project’s outcome. In other words, the project will fail if EnSoft Corp. does not provide the Atlas APIs.

#### 3.1.10.3. Lack of Eclipse plugin Creation Experience

No member of the team has previous experience creating an Eclipse plugin. This poses as a risk to the project due to the requirement that the system serves as a query scripting interface for Atlas. This requirement limits the design to an Eclipse plugin since that Atlas itself is a plugin for Eclipse. The impact this risk may have on the project is missing deadlines. To minimize the effect, each team member will be researching how to create a plugin for Eclipse prior to the prototyping phase of the project, although only one team member will be assigned the implementation of the add-in portion of the system itself. Code reviews and debugging sessions will be much more timely and effective when each member of the team has an understanding of the design specifications at the beginning.

### 3.1.11. Project Proposed Milestones & Evaluation Criteria

Software Project Plan

*Entrance Criteria:* Group has been initialized and proper planning discussions have been completed.

*Exit Criteria:* The project plan is complete.

Software Design Document

*Entrance Criteria:* The project plan is complete and each team member has an understanding of the Eclipse plugin environment.

*Exit Criteria:* The design document is complete.

Setup SVN – GoogleCode

*Entrance Criteria:* Project group initialization

*Exit Criteria:* All student developers have access to the source repository and know how to use the functionality provided by the SVN.

Setup Development Environment - Eclipse

*Entrance Criteria:* Team knows software platform/environment requirements for Atlas.

*Exit Criteria:* Each team member has the development environment on his/her machine setup with an identical configuration that is compatible with the software requirements of Atlas.

Successful Execution of an Atlas Query via a Call to the Atlas API

*Entrance Criteria:* EnSoft has delivered the Atlas plugin and compatible Atlas APIs.

*Exit Criteria:* Execution of an Atlas API calls that returns expected output.

Working JavaScript Parser

*Entrance Criteria:* Team has an understanding of the functionality required by the JavaScript parser.

*Exit Criteria:* Successful call to a Java method from a JavaScript command.

Working Eclipse Plugin Prototype

*Entrance Criteria:* The project plan is complete and the development environment and SVN repository are set up on each team member’s machine.

*Exit Criteria:* The plugin prototype is able to read an input script, make the corresponding Atlas call(s) and save the result in the text file.

## 3.2. Statement of Work

### 3.2.1. Task 1: Problem Definition

*Objective*: To provide the team with a complete definition of the problem including elements such as anticipated end-product users and uses as well as the constraints that must be observed.

*Approach*: In order to define the problem in its entirety, the problem definition must be complete and the project constraints, end-product users and end-product uses must be identified.

*Results*: After completing the problem definition task, the team understands the overall project objective as well as the end-product users and uses. The team also knows the constraints which the project must be performed under. This information has been documented

#### 3.2.1.1. Subtask 1.1: Problem Definition Completion

*Objective:* To provide the team with the problem definition the software system being created is intended to solve.

*Approach*: The team discussed the client’s project proposal with the client to gain an understanding of the problem the client needs resolved. Open-ended and simple questions were asked in order to obtain a true and accurate understanding of the client’s problem. Open-ended questions are questions that cannot be answered simply with yes or no. Simple questions are questions such as “What happens next?”, and when asked repetitively can lead to an understanding of the big picture.

*Results*: The team understands and has documented the overall objective of the project and the problem at hand.

#### 3.2.1.2. Subtask 1.2: End-Users & End-Uses Identification

*Objective*: To identify the users of the end-product and how the end-product is intended to be used.

*Approach*: The team inquired about the nature of the client’s work in order to gain an understanding of the intended use of the end-product. They also asked the client about other end-product users to understand their backgrounds and capabilities. Again, open-ended and simple questions were asked in order to obtain a true and accurate understanding of the end-product users and uses.

*Results*: A descriptive list of both the end-product users and the intended uses of the end-product has been developed by the team.

#### 3.2.1.3. Subtask 1.3: Constraint Identification

*Objective*: To identify all project constraints that exists.

*Approach*: The team obtained a list of initial specifications from the course instructor and from the faculty advisor. These specifications include the CprE 491/492 course schedules and procedures that must be adhered. Discussions with the client have revealed additional constraints as well.

*Results*: The team understands and has documented the bounds – time, cost, environment, etc. – that the end-product must be developed within.

### 3.2.2. Task 2: Technology Considerations and Selection

*Objective*: To select the best technologies that could be used to create the solution to the client’s problem.

*Approach*: In order to select the appropriate technologies for the project, possible technologies needs to be identified and the selection criteria needs to be identified. Then the technologies must be researched prior to being selected. Technologies considered are for project development and as part of the make-up of the end-product.

*Results*: After completing the technology considerations and selection task, the team chose the technologies that will be used during the project and the technologies that are part of the end-product.

#### 3.2.2.1. Subtask 2.1: Identification of Possible Technologies

*Objective*: To identify applicable technologies that could be used to create the solution to the client’s problem.

*Approach*: After the team had an understanding of the problem, they discussed with the faculty advisor what technologies are applicable to the project within the constraints identified in Task 1.

*Results*: The team determined the technologies that can be controlled are the computer programming language, the data storage system, and the JavaScript parser used for the project.

#### 3.2.2.2. Subtask 2.2: Identification of Selection Criteria

*Objective*: To identify the properties needed by the identified applicable technologies in order to successfully create a solution to the client’s problem.

*Approach*: The team discussed the many properties of software projects. Properties were categorized as high-, medium-, and low-importance in accordance with the project. Based on this categorization, the selection criteria were determined.

*Results*: The team identified and documented the properties needed by the technologies to be used to create the end-product included falling within the project constraints as well as the accessibility, extensibility, intended use, and the stability of the technology.

#### 3.2.2.3. Subtask 2.3: Technology Research

*Objective*: To gain a better understanding of the technologies considered for use in the project development.

*Approach*: The team has assigned a week long research timeframe for each of the technologies. Each team member researched the technology using whatever means that member has deemed applicable. At the weekly team meetings, each member presented what he or she learned about the technology. This was followed by a group discussion about the pros and cons of the technology.

*Results*: The team knows enough about each of the identified technologies to be able to assess their properties based on the selection criteria.

#### 3.2.2.4. Subtask 2.4: Technology Selection

*Objective*: To determine which technologies will be utilized in the end-product.

*Approach*: During an in-person meeting, the team discussed the identified technologies to determine whether or not the technologies fit within the project constraints, if they are available for academic/development use, if they are compatible with the development and anticipated end-product environments, and if they are stable.

*Results*: The team selected and documented the technologies that will be utilized throughout the project and the technologies that will be part of the end-product.

### 3.2.3. Task 3: End-Product Design

*Objective*: To identify the end-product design requirements and to implement and document the system design.

*Approach*:  Before the design can begin, the team must obtain the requirements for the design. Then the team members will perform research on technology options with which they are not familiar. The team will meet weekly to discuss their findings and see how those results can be related to the design. Each weekly meeting will build upon the previous week’s results until the design is complete. During this process, the team will document its findings into one report.

*Results*: After having completed the end-product design task, the team has a list of functional and non-functional requirements for the project, a high- and low-level design of the end-product, and a single document explaining in detail the end-product design.

#### 3.2.3.1. Subtask 3.1: Identification of Design Requirements

*Objective*: To identify all of the functional requirements – what the system should do – and non-function requirements – how the system should behave.

*Approach*: The team discussed the problem statement and used their understanding of the client’s problem (see task 1) to establish both functional and non-functional end-product requirements. Requirements are represented in use cases, screenshots and program flow diagrams where applicable.

*Results*: A complete list of functional and non-functional system specifications, including use cases, screenshots and program flow is established.

#### 3.2.3.2. Subtask 3.2: Design Process

*Objective*: To design the architecture and determined the code design patterns of the end-product.

*Approach*: The team discussed the requirements established in the previous task in order to ensure that the system design fulfills all of the requirements. To establish a high-level design the team chose a system architecture that fulfills these requirements by discussing pros and cons of different architectural models. Low-level design was established by evaluating the many different code design patterns.

*Results*: A complete system design, high-level and low-level, has been established.

#### 3.2.3.3. Subtask 3.3: Documentation of Design

*Objective*: To create a thorough document describing the design of the end-product.

*Approach*: The team shall create a single document to explain the design chosen for the end-product. Each team member was responsible for completing different sections of the document (determined by the project manager). The team submitted the document to the faculty advisor for review prior to submitting it to the CprE 491 course instructor.

*Results*: A single document exists covering the elements of the end-product design.

### 3.2.4. Task 4: End-Product Prototype Implementation

*Objective*: To implement a prototype version based on the completed design.

*Approach*: Based on the research done during the design phase, the team implemented technologies by starting with the simplest cases. From these, the team determined prototyping limitations and iteratively constructed the end-product by adding features to successful prototypes.

*Results*: The team coded the end-product in an iterative manner using limited-scoped prototypes.

#### 3.2.4.1. Subtask 4.1: Development Environment Setup

*Objective*: To setup a consistent development environment on each team member’s computer.

*Approach*: Each team member downloaded the bundled version of Eclipse provided by the client. This bundle contains the Eclipse IDE for plugin development with the Atlas and subversion plugins installed. In addition, a GoogleCode SVN repository was setup.

*Results*: Each team member will has the bundled version of Eclipse installed on his or her machine.

#### 3.2.4.2. Subtask 4.2: Identification of Prototype Limitations and Substitutions

*Objective*: To identify the limitations of and substitutions needed for each end-product prototype.

*Approach*: The team determined what modules of the end-product require prototyping. The team also determined the module size (scope of prototype) and identified the limits of prototype based on the module purpose. Substitutions needed for the prototype were identified by analyzing the system design.

*Results*: The limitations of and substitutions needed for the system prototypes were identified and defined.

#### 3.2.4.3. Subtask 4.3: Implementation of Prototyped End-Product

*Objective*: To create the end-product by implementing and integrating prototypes.

*Approach*: Each of the team members was responsible for prototyping different modules of the system. The members integrated the prototypes together to form subsystems, followed by integration of the subsystems until the end-product exists. At each integration point, the team tested and demoed the prototype to the client/advisor.

*Results*: The proposed end-product has been coded according to the design to fulfill the project requirements.

### 3.2.5. Task 5: End-Product Testing

*Objective*: To thoroughly test the end-product to verify its accuracy and to ensure that it meets all the requirements of the project.

*Approach*: The team developed a series of test cases to verify that each of the requirements is met. The test design was influenced by the system design. Each test has a test case description (which will include a reference to the requirement being tested), and each time a test case was run, its status was updated in the test documentation.

*Results*: The team has thoroughly tested the end-product to verify its accuracy and to ensure that it meets all the requirements of the project by planning out the testing, developing applicable tests, executing and evaluating the tests, and documenting each of the testing phases.

#### 3.2.5.1. Subtask 5.1: Test Planning

*Objective*: To develop a test plan explaining how the project will be verified.

*Approach*: The team and the faculty advisor discussed the different types of test, testing methods and alternative verification methods. From this discussion, the team members determined the most appropriate combination of tests, testing methods and alternative methods to use in order to verify the project’s accuracy and that it fulfills the requirements. The criteria needed for a test to pass were also determined during this task.

*Results*: A detailed plan of how the project is to be verified exists.

#### 3.2.5.2. Subtask 5.2: Test Development

*Objective*: To develop a set of tests that verifies the accuracy of the end-product and ensures that the project requirements were fulfilled.

*Approach*: Each team member created tests for each of the modules he or she works on. These tests included unit, integration, and system tests, and they may or may not be automated. The set up and tear down for each test was determined during test development.

*Results*: Multiple test sets exist, and when executed they can verify the accuracy of the end-product and ensure that the project requirements were fulfilled.

#### 3.2.5.3. Subtask 5.3: Test Execution

*Objective*: To execute the tests developed to verify the system.

*Approach*: The team members review the test plan prior to executing the test developed in order to understand what should be being tested. After understanding the test plan, the team member executes the set of tests in accordance with the plan. After the test execution completes, the team member shall record the test results. If any tests fail, the tester shall inform the developer responsible for the code which caused the test to fail and record that a bug exists.

*Results*: All tests shall be executed at least once, and all their most recent execution results are PASS. If the test failed its most recent execution, an open bug has been documented.

#### 3.2.5.4. Subtask 5.4: Test Evaluation

*Objective*: To confirm the accuracy of the tests that are developed and the thoroughness of the test sets.

*Approach*: Test evaluation occurred throughout the testing phase. During team meetings, tests were discussed during their development to ensure that the tests developed actually test what they are intended to. This required the team to analyze the test code. After test execution, the tester presented the results to the team during a meeting or via email. The team then determined whether the testing was complete for that module or if additional testing should occur.

*Results*: The team has concluded that the system has been tested thoroughly allowing them to state the system is accurate and fulfills are requirements.

#### 3.2.5.5. Subtask 5.5: Documentation of Testing

*Objective*: To develop documentation containing the test plans and results for all testing of the system.

*Approach*: The team created documentation explaining the testing methods chosen to verify the end-product. The document explains the testing results. Each team member shall be responsible for completing different sections of the document (determined by the project manager). The team submitted the documentation to the faculty advisor/client for review prior to submitting it to the course instructor.

*Results*: Documentation exists covering every testing element for the end-product design.

### 3.2.6. Task 6: End-Product Documentation

*Objective*: To develop a user-manual for the end-product.

*Approach*: A team member analyzed other software user-manuals to determine the content of the document. Team members wrote sections of the document (determined by the project manager) and compiled the information into a single document.

*Results*: A user-manual for the end-product exists and is easy to navigate and understand.

### 3.2.7. Task 7: End-Product Demonstration

*Objective*: To demonstrate the capabilities of the end-product, the status of the project and answer all questions asked by the audience.

*Approach*: blah.

*Results:* The team has demonstrated the end-product to all necessary audiences – their faculty advisor, the client and the industry review board.

#### 3.2.7.1. Subtask 7.1: Demonstration Planning

*Objective*: To develop a plan of how demonstrations will be run, what each team member’s role is during demonstrations, and what evaluation criteria of exist.

*Approach*: The team discussed what is needed to demonstration the end-product effectively. The team determined this criteria and assigned roles to each team member for the various demonstrations (the same member will not be doing the same thing for each demonstration). Dates for demonstration pre-runs were established to practice as a group prior to the demonstrations.

*Results*: Each team member understands his or her role in the upcoming demonstration.

#### 3.2.7.2. Subtask 7.2: Faculty Advisor Demonstration

*Objective*: To demonstrate the current status of the end-product to the faculty advisor.

*Approach*: The team met with the faculty advisor on a regular basis to demonstrate the status of the end-product development. Each demonstration revealed the work that has been done since the previous demonstration showing the added complexity of the system. Each team member performed his or her assigned role during the demonstration (the same member did not be do the same thing for each demonstration). During and after the demonstration, the advisor was encouraged to ask questions. If the team could not answer the question during the demonstration, the answer was provided to the advisor during the next meeting time or via email.

*Results*: The faculty advisor understands what happened during the demonstrations, knows the status of the project, and has had all of his questions answered.

#### 3.2.7.3. Subtask 7.3: Client Demonstration

The client of the project is also the faculty advisor. See Subtask 7.2: Faculty Advisor Demonstration for a description.

#### 3.2.7.4. Subtask 7.4: Industrial Review Board Demonstration

*Objective*: To demonstrate the end-product to the industry review board.

*Approach*: The team met with the industry review board at the end of the spring semester to demonstrate the status of the end-product development. This demonstration revealed the work done for the project. Each team member performed his or her assigned role during the demonstration. During and after the demonstration, the team encouraged the board to ask questions and answered them to the best of their ability.

*Results*: The industry review board understands what the system was doing during the demonstration and has had all of his questions answered

### 3.2.8. Task 8: Project Reporting

*Objective*: To perform the required project reporting.

*Approach*: The team shall collaborate to create the five project reporting elements required by the course. The content of the reports shall fulfill all requirements provided by the course instructor.

*Results*: The team has developed the project plan, the end-product design, the project poster highlighting the project, and a final project report detailing all aspects of the project. These documents explain the project to those people not directly involved in the project. Lastly, a weekly email report has been sent to update the CprE 491/492 course instructors, faculty advisor/client, and the team members on the project status throughout the project’s lifecycle.

#### 3.2.8.1. Subtask 8.1: Project Plan Development

*Objective*: To develop the plan used to implement the project.

*Approach*: The team members used their knowledge of project management and software projects in general to develop a project plan. The team’s project manager was responsible for constructing the resource requirements and project scheduling. Other team members assisted in the document.

*Results*: A detailed plan of how the project will be designed, implemented, tested, and documented exists.

#### 3.2.8.2. Subtask 8.2: Project Poster Development

*Objective*: To design and construct a poster explaining the many aspects of the project.

*Approach*: The team’s project manager created a poster that highlights the project. The poster explains the problem statement and the many phases of the project. The poster also introduces the team members. The language used on the poster is concise, clear, and of a professional manner..

*Results*: A poster exists explaining the purpose of the project (problem statement), project design, project resources, project testing, and team accomplishments.

#### 3.2.8.3. Subtask 8.3: End-Product Design Report Development

See Subtask 3.3.

#### 3.2.8.4. Subtask 8.4: Project Final Report Development

*Objective*: To develop a report explaining the project as a whole that will be submitted to the course instructor and the industry review board.

*Approach*: The team created a single document to explain the project. Each team member was responsible for completing different sections of the document (determined by the project manager). The team determined the content of the document based on course requirements, discussing concepts applicable to the end-product development, and discussing team objectives, accomplishments, and struggles. The team submitted the document to the faculty advisor for review prior to submitting it to the course instructor.

*Results*: A single document exists covering every element of the project.

#### 3.2.8.5. Subtask 8.5: Weekly Email Reporting

*Objective*: To send an email to the team, the faculty advisor and the instructor stating the team’s current status.

*Approach*: One team member was responsible for sending the weekly update email each week. During the fall semester the communication liaison sent the weekly updates, and during the spring semester the project manager sent the weekly updates. Each team member was responsible for informing the report writer of the work that he or she did during the week. Current action items as well as new and relevant old business discussed in meetings were included in the emails. This email was sent prior to 9:00 AM every Monday during the fall semester and 5:00 PM every Wednesday during the spring semester.

*Results*: The team’s status was announced to team members, the faculty advisor, and the instructor each week.

# 4. Estimated Resources and Schedules

## 4.1. Estimated Resources

### 4.1.1. Personnel Effort Requirements

Table 3 shows the hourly work breakdown that is required from each team member. The differences in the hours for the various tasks are due to the role that each member plays in the team. Cole is the team webmaster, Kristina is the team project manager, and Alex is the team’s communication liaison.

Table 3 - Personnel Effort Requirements by Task

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Team Member** | **Task 1** | **Task 2** | **Task 3** | **Task 4** | **Task 5** | **Task 6** | **Task 7** | **Task 8** | **Totals** |
| Cole Anagnost | 4 | 10 | 30 | 90 | 80 | 5 | 8 | 45 | 272 |
| Kristina Gervais | 6 | 10 | 30 | 70 | 80 | 10 | 10 | 70 | 286 |
| Alex Kharbush | 6 | 10 | 40 | 90 | 80 | 5 | 8 | 50 | 289 |
| Totals | 16 | 30 | 100 | 250 | 240 | 20 | 26 | 165 | **847** |

### 4.1.2. Other Resource Requirements

Table 4 lists the additional physical resources needed to fulfill the project.

Table 4 - Non-personnel Resource Requirements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **Team Hours** | **Other Hours** | **Cost** | |
| Parts and materials: | | | | |
| a. Project Poster Including Printing | 12 | 0 | | $45.00 |
| b. Report Printing Materials | 102 | 0 | | $50.00 |

### 4.1.3. Financial Requirements

This project does not incur many financial requirements since it is a software system development project. The non-labor costs are limited to printing materials. All development software will be open source or provided by the client, and the team will use their personal computers to perform work. The labor costs in Table 5 are calculated based on the personnel effort requirements (see section 4.1.1.).

Table 5 - Labor and Materials Costs

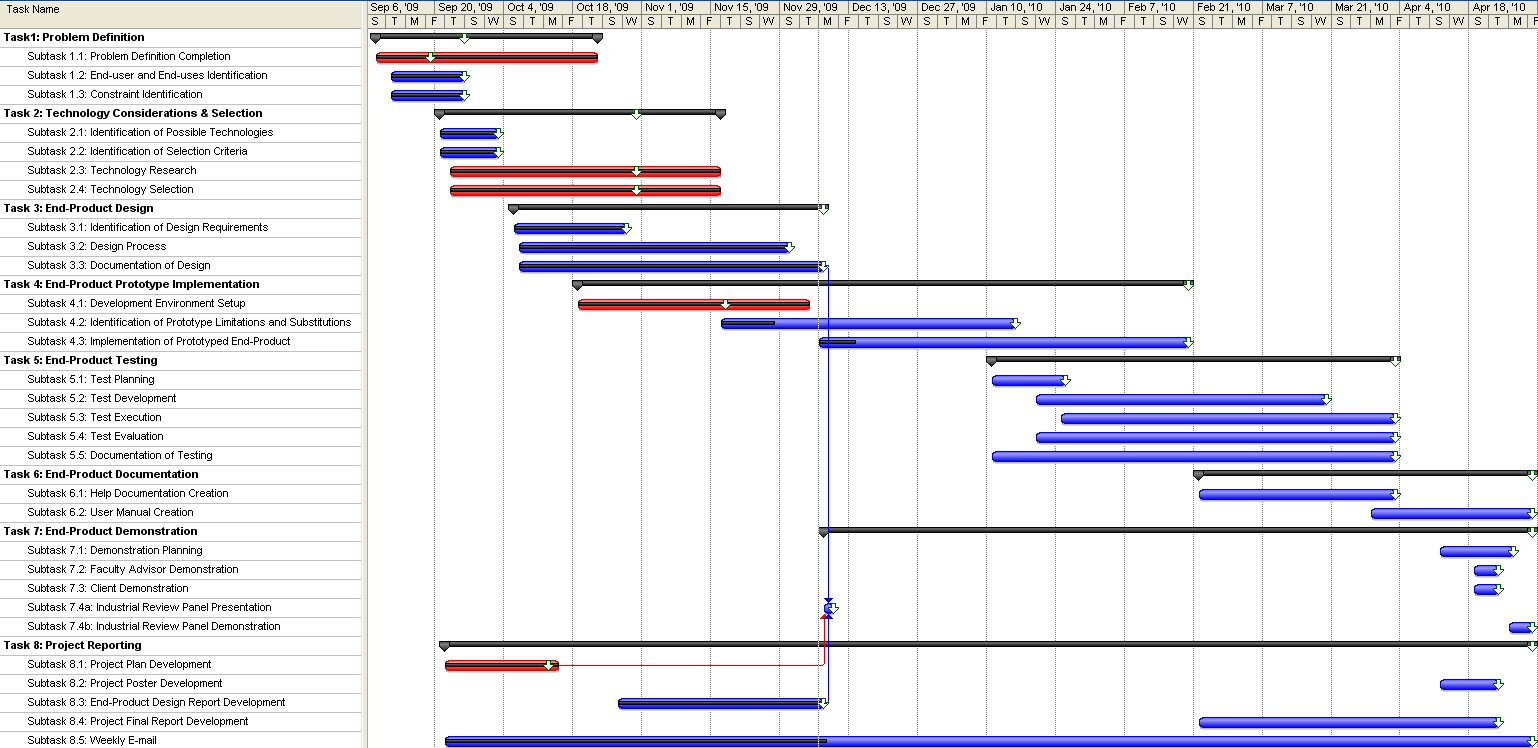
|  |  |  |
| --- | --- | --- |
| **Item** | **Without Labor** | **With Labor** |
| Printing Materials: | | |
| a. Poster Materials | $45.00-Provided by Department | $45.00-Provided by Department |
| b. Report Materials | $50.00-Included in Department Fee | $50.00-Included in Department Fee |
| Subtotal | $0.00 | $0.00 |
| Labor at $20 per hour: | | |
| a. Cole Anagnost |  | $5,240.00 |
| b. Kristina Gervais |  | $5,720.00 |
| c. Alex Kharbush |  | $5,800.00 |
| Subtotal |  | $16,760.00 |
| **Total** | $0.00 | $16,760.00 |

## 4.2. Schedules

### 4.2.1. Project Schedule

The detailed project schedule is shown as a Gantt Chart in Figure 2. Each bar in the chart represents a task or a subtask. The white down arrows on each task bar represent the task deadline. The black bar in the middle of each colored task bar indicates how much of the task has already been completed. Blue task bars are tasks that are on schedule, while red task bars represent tasks that are, or completed, behind schedule. Arrows that connect task bars together indicate that a dependency between the tasks exists.

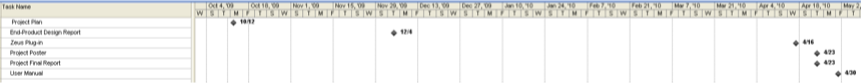
Figure 2 - Project Schedule



### 4.2.2. Deliverable Schedule

Figure 3 shows the deliverable schedule for the project. Two deliverables, the Project Plan and the Specifications & Design Report will be completed during the fall semester. The remaining deliverables, the Zeus Plugin, the Project Poster, the Final Report, and the User Manual, will be delivered at the end of the spring semester.

Figure 3 - Deliverable Schedule



# 5. Overall Requirements Description

## 5.1. Product Perspective

### 5.1.1. Concept of Operations

The solution will be an Eclipse plugin. The influencing factor on this design choice is that the solution must be a software application compatible with the Eclipse IDE and the Atlas software analysis tool. As the block diagram in Figure 4 shows, Zeus will interface with both Eclipse and Atlas. The user will interact with Zeus via Eclipse (due to Eclipse plugin properties). Zeus will interpret the scripts that the user inputs and send the appropriate query (including result display formatting) requests to Atlas. Zeus will not directly do any analysis of the C source code.

Figure 4 - Block Diagram



### 5.1.2. User Interfaces

The user interface must provide an editor for the query scripts. It also must provide a manner to display query results as well as a manner to open, close, save, and run a query script.

### 5.1.3. Hardware Interfaces

The only hardware required for Zeus is a standard personal computer capable of running Windows XP and Java. This includes IO devices such as a keyboard, mouse, and monitor.

### 5.1.4. Software Interfaces

Due to the nature of the client’s problem, the system will need to interface with the Eclipse IDE and Atlas. No other software interfaces are required.

### 5.1.5. Communication Protocols and Interfaces

None, the system will be a standalone application requiring no networking capabilities.

### 5.1.6. Memory Constraints

None.

### 5.1.7. Site Adaption Requirements

None, no site adaption requirement is necessary since this application is intended to display messages to the user only in English and is not intended for use outside of the United States.

## 5.2. Product Functions

The use case diagram in Figure 5 depicts the features available to the user. There are three actors on the Zeus system. The only primary actor is the system user. Two secondary actors exist; they are the Eclipse and Atlas systems. Three use cases exist and are described in detail below.

Figure 5 - System Use Case Diagram



## 5.3. User Characteristics &. Intended Uses

See section 2.3. in the Zeus Project Plan document.

## 5.4. Constraints

The client has specified the following list of constraints for the Zeus system.

System must integrate with the Atlas code analysis tool.

System must integrate with the Eclipse development platform.

System shall function as an Eclipse plugin.

System input shall be written in JavaScript.

## 5.5. Assumptions & Limitations

See section 2.4. in the Zeus Project Plan document.

## 5.6. Dependencies

Zeus is dependent on Atlas and Eclipse, and therefore is dependent on the JRE, Graphviz, and the SWT.

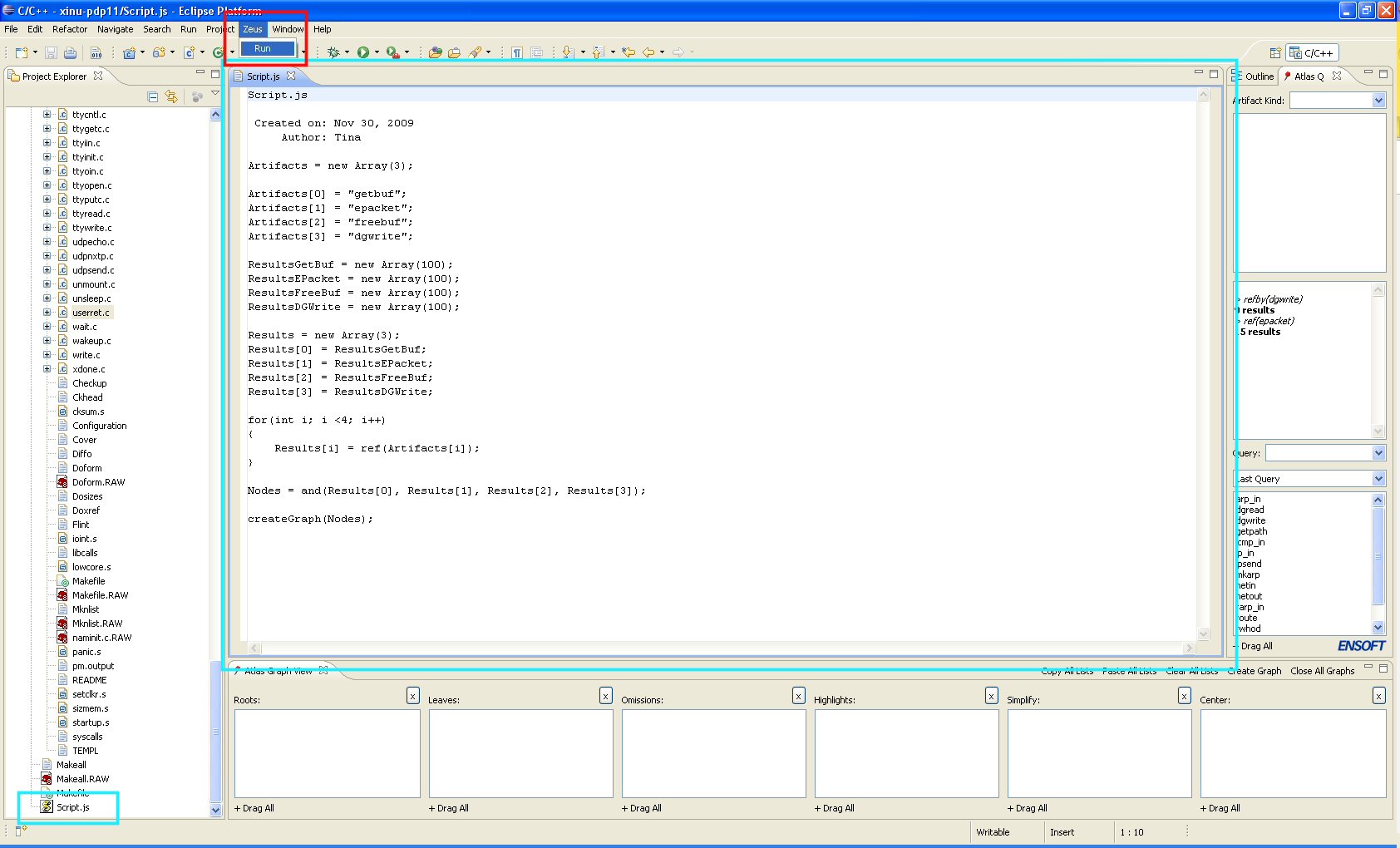
# 6. Specific Requirements

## 6.1. External Interface Requirement

### 6.1.1. User Interfaces

The user interface will be an extension of the Eclipse GUI. Many elements will be the same, such as a menu bar and simple buttons. The user will be able to enter queries into a text field and a simple way to launch them. The results of the queries will be shown in Atlas or in Zeus depending on the query. Figure 3 shows an example screen shot of a user interface that fulfills these requirements.

Figure 6 - User Interface Screenshot



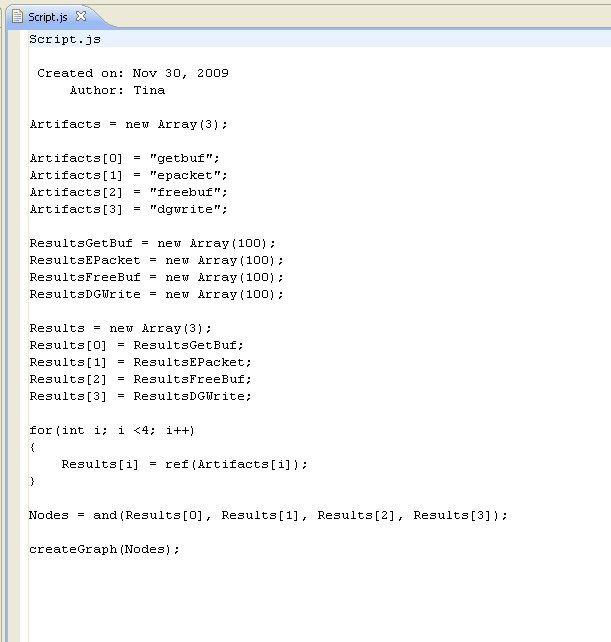
The red box highlights the menu that enables the user to launch the query script. The large blue box in the middle of Figure 3 shows the text editor supplied with Eclipse that will be utilized as the script editor. The blue box in the bottom, left corner of the image shows where the script file is show in the Eclipse Project Explorer. The Atlas panels are shown on the right side and along the bottom of the image. No changes will be made to these panels. Results will be shown in the Atlas panel on the right or in the location of the text editor depending on if a list of Atlas artifacts or a graph is the result format, respectively.

### 6.1.2. System Input & Output

#### 6.1.2.1. System Input

All input to the system will be done via the Eclipse IDE. The user will select the query script to execute and select the run option for Zeus. Zeus will then obtain the JavaScript from the selected query file and execute the query. Figure 4 shows a sample of what the JavaScript should look like.

Figure 7 - JavaScript Query Example



#### 6.1.2.1. System Output

All system output will also be display via the Eclipse IDE. The displaying of the results will be handled by a call to Atlas. Figure 5 shows the results displayed as both a graph (surrounded by the blue box) and an artifact list (surrounded by the red box).

Figure 8 - System Output Example

### Output.jpg

### 6.1.3. Operating Platform

The software-operating environment will be the Eclipse IDE as the program will be a plugin for Eclipse. Eclipse is portable to any operating system compatible with Java since it runs on the JVM; therefore, Zeus will operate on any system the supports the JVM.

### 6.1.4. Software Interfaces

Zeus shall interface with both the Atlas and Eclipse software systems. Zeus shall launch Atlas queries and obtain the results through API calls. The input query script that shall be obtained from the Eclipse IDE shall determine the Atlas queries executed. Zeus shall also be launched from the Eclipse IDE.

## 6.2. Features

System shall interpret JavaScript queries and execute corresponding Atlas queries.

System shall provide user the ability to create JavaScript queries.

System shall provide user the ability to edit JavaScript queries.

System shall provide user the ability to delete JavaScript queries.

System shall provide user the ability to store JavaScript queries.

System shall provide user the ability to load JavaScript queries.

System shall display script query results as an Atlas artifact list.

System shall display script query results as an Atlas graph.

## 6.3. Performance Requirements

None, the client has not specified any performance requirements. Also, the dependency of the system on Eclipse, Atlas and the size of the input makes it difficult to test the overall performance Zeus.

## 6.4. Design Constraints

Based on the constraints the client placed on the product, Zeus must be an Eclipse plugin compliant with Eclipse Ganymede 3.4.2.

## 6.5. Software System Attributes

### 6.5.1. Reliability & Availability

None, the Zeus software system will be used for research purposes and is a standalone application that is only run periodically.

### 6.5.2. Portability

System shall not be OS specific (although its dependencies may be).

System shall fail gracefully with an appropriate error message and will not close the program. System shall not be subject to stack overflow creating system failure.

System shall not crash due to invalid input.

System shall not cause memory leaks.

## 6.6. Other Requirements

System shall include a user manual.

System shall display text in the user interface only in English.

# 7. Design Goals

Several requirements influence the system design. The order of following list is priority of importance.

Correctness

Usability

Robustness

Efficiency

Maintainability

Extensibility

## 7.1. Correctness

Zeus shall satisfy both the functional and non-functional requirements listed in this document. Verification for what the system does and the actions it takes to do it will be done through testing, static code analysis, and code inspections.

## 7.2. Usability

Zeus shall be easy to use by providing an intuitive GUI, standard layout, and adequate documentation. This will be verified through user questionnaires/interviews.

## 7.3. Robustness

The system shall be tolerant of misuse (faulty data, bad use, or bad environment) without catastrophic failure. This will be achieved through data abstraction and encapsulation, using simple interfaces, shielding from data corruption, initializing variables, qualifying all inputs, formal parameters to a method, invariants and post-conditions.

## 7.4. Efficiency

The system shall make intelligent use of the processor and memory to insure the performance requirement metrics are met. This will be achieved by writing well-designed algorithms and using appropriate data structures (e.g., the use of an ArrayList versus HashTable).

## 7.5. Maintainability

The system shall be designed to reuse object code, source code, assemblies of related classes (software frameworks) and patterns of design. It will be modular by using classes and interfaces, which are independent and as general or specific as necessary. Classes and methods shall be described with good naming conventions and documentation. Naming conventions would include using camel case and avoiding shorthand whenever possible (e.g. username instead of uname). Minimize dependency between classes and maximally abstract and general or precisely matched to real objects and their function. In addition, methods shall not be coupled closely to classes. In order to provide adequate and accurate understanding of methods, documentation shall explain the algorithm and specify preconditions, post-conditions and invariants. Documentation includes JavaDocs in addition to this document.

## 7.6. Extensibility

The system shall be designed to allow for the extension of different types of the same base category. This may occur by means of abstract classes and extension of functionality through creating additional class methods or an abstract class and several derived classes.

# 8. Decomposition Description

## 8.1. Module Decomposition

The architecture diagram in Figure 6 shows the division of Zeus into four subsystems: Atlas interface module, control module, Eclipse plugin module, and script parser module. Design the system in a modular manner by grouping related tasks together reduces coupling and thus improving system implementation and maintenance.

Figure 9 - Architecture Diagram



### 8.1.1. Control Module

The control module contains the overall logic for the system. The control is responsible obtaining requests from Eclipse and launching the execution of the query scripts.

Components:

RunHandler: launches the query parser upon user request event.

**Services:** launches the execution of query scripts

### 8.1.2. Eclipse Plugin Module

The Eclipse plugin module provides the user interface elements for Zeus enabling the user to interact with the application. It is also responsible for obtaining the input query script for execution as well as the startup and shutdown actions required by Zeus. (Note: plugins are not automatically launched by Eclipse; instead they are loaded and started only when demanded or required.)

Components:

Manifest File: defines the runtime information of the plugin

plugin.xml: defines extension information of the plugin. Zeus will not expose and APIs.

ZeusPlugin Class: the main plugin class that contains the activator method called when the plugin starts.

ZeusView Class: contributes Zeus components to the Eclipse GUI.

**Services:** process Eclipse menu inputs, adds features to Eclipse GUI

### 8.1.3. Atlas Interface Module

The Atlas Interface Module is responsible for all interaction with Atlas, and therefore will handle all calls to the Atlas APIs.

Components:

AtlasAdapter Class: wrapper for the Atlas APIs

Atlas API JARs: the extension to the Atlas plugin which enables external calls to the system

**Services:** Makes Atlas API calls

### 8.1.4. Query Script Parser

The query script parser module is responsible for parsing the JavaScript input queries. Using the *LiveConnect* feature provided by Mozilla Rhino, the JavaScript will interact with our Java objects. Essentially, this enables the JavaScript to call Java methods.

Components:

Mozilla Rhino JAR files: enables the JavaScript input to call Java methods provided within Zeus. The primary package with the JAR used is the javascript package.

Artifact Abstract Class: handles all data related to the Atlas Artifact type. This is implemented as an abstract class to enable any changes in functionality to be inherited by all subclasses.

Function Class: handles all data related to the Atlas Function artifact type and extends the Artifact class.

Type Class: handles all data related to the Atlas Type artifact type and extends the Artifact class.

Variable Class: handles all data related to the Atlas Variable artifact type and extends the Artifact class.

**Services:** Parses JavaScript input

## 8.2. Concurrent Processes & Threads

### 8.2.1. Eclipse

Eclipse is the main process of the application. Eclipse plugins are loaded and run from within the Eclipse process.

Creation: Application start

Termination: Application close

Threads: Eclipse creates multiple threads to handle various tasks performed by the system. Similar to other plugins, Zeus will be executed within the main thread. Zeus will not create additional threads.

Main Thread: Runs the UI event loop and dispatches events

Worker / Daemon Threads: Created by Eclipse or plugins for background processing

Operations: Responds to user events (mouse clicks, keystrokes, etc.) and performs background processing provided the user with a source code editor, compiler, debugger, and software analysis.

### 8.2.2. Graphviz

Graphviz is the software application used by Atlas to create the graph outputs. See Atlas documentation for more details.

## 8.3. Data Decomposition

To support the composition and projection capabilities of Atlas queries, collections of Artifact objects will be used to store the results of Atlas query calls. This prevents using Atlas variables explicitly with Zeus. This ultimately simplifies the data management because Atlas variables must be passed as strings and also store large amounts of system state data.

## 8.4. States

The state diagram in Figure 7 shows an abstract description of the behavior of the Zeus system. Once the plugin is loaded and started within Eclipse, Zeus will be idling until the user selects the run query script feature from the Eclipse GUI. This event will cause the system to begin interpreting the query script which includes parsing the script, executing the Atlas query calls, and displaying the results. Upon completion of the script, the system will return to the idle state until the user the user selects the run query script feature again.

Figure 10 - System State Diagram



# 9. Dependency Description

Zeus is dependent on Atlas, and therefore is dependent on the JRE, Graphviz, and the SWT.

## 9.1. Inter-module Dependencies

The Zeus program will have four inter-modules. The Control module will depend on both the Eclipse and the Query Script Parser modules. The Atlas Interface module will only interact with the control module and hence will be dependent on the control module. The Eclipse Plugin module will only work with the control module and hence will be dependent on the control module. The Query

Script Parser module will be dependent on the Atlas Interface Module and the Control Module as it is in charge of parse the Atlas queries in the input script and is launched by the Control.

### 9.1.1. Atlas Interface Module

The Atlas Interface Module is dependent on the Query Script Parser since it provides the data input to the Atlas Interface module. It is also dependent on the Atlas APIs that it calls.

### 9.1.2. Control Module

Eclipse Module - The control module will reside within Eclipse and will depend on everything that Eclipse does.

Atlas Module - The control module launch the Query Script parser.

### 9.1.3. Eclipse Plugin Module

An Eclipse plugin is built on top of existing Eclipse plugins and is therefore dependent on those existing plugins in order to compile. This section declares all those dependencies.

JRE - The Java Runtime Environment is required as Eclipse is executed within a JVM.

Graphviz - The visualization application that handles the graphing of nodes for Atlas.

SWT - The Standard Widget ToolKit will help the Eclipse Module create the GUIs minus the graph section.

### 9.1.4. Query Script Parser Module

The parser module depends on the Mozilla Rhino APIs to provide the JavaScript parsing capabilities. It depends on the return values from the Atlas Interface Module. In addition the input for the Query Script Parser is passed from the Control Module when the parser is launched.

## 9.2. Inter-process Dependencies

No inter-process dependencies exist for Zeus; however, Atlas has process dependences. See the Atlas User Manual for more details.

## 9.3. Data Dependencies

Atlas query input parameters and results will be temporarily stored as a collection of Artifact (or the appropriate subclass) objects to support the composition of queries. This data does not need to be persistent and will not be kept after the script execution has completed.

# 10. Interface Description

## 10.1. Module Interface

### 10.1.1. Atlas Interface Module

The interface for this module consists of the AtlasQueryAdapter methods, which wrap the individual Atlas API calls.

### 10.1.2. Control Module

The control module is the layer of abstraction between the Eclipse plugin module and the Query Script Parser and is responsible for passing the input script in the correct format to the parser.

### 10.1.3. Eclipse Plugin Module

This module provides the user interface components (menu items, toolbar button, text editor) for the application.

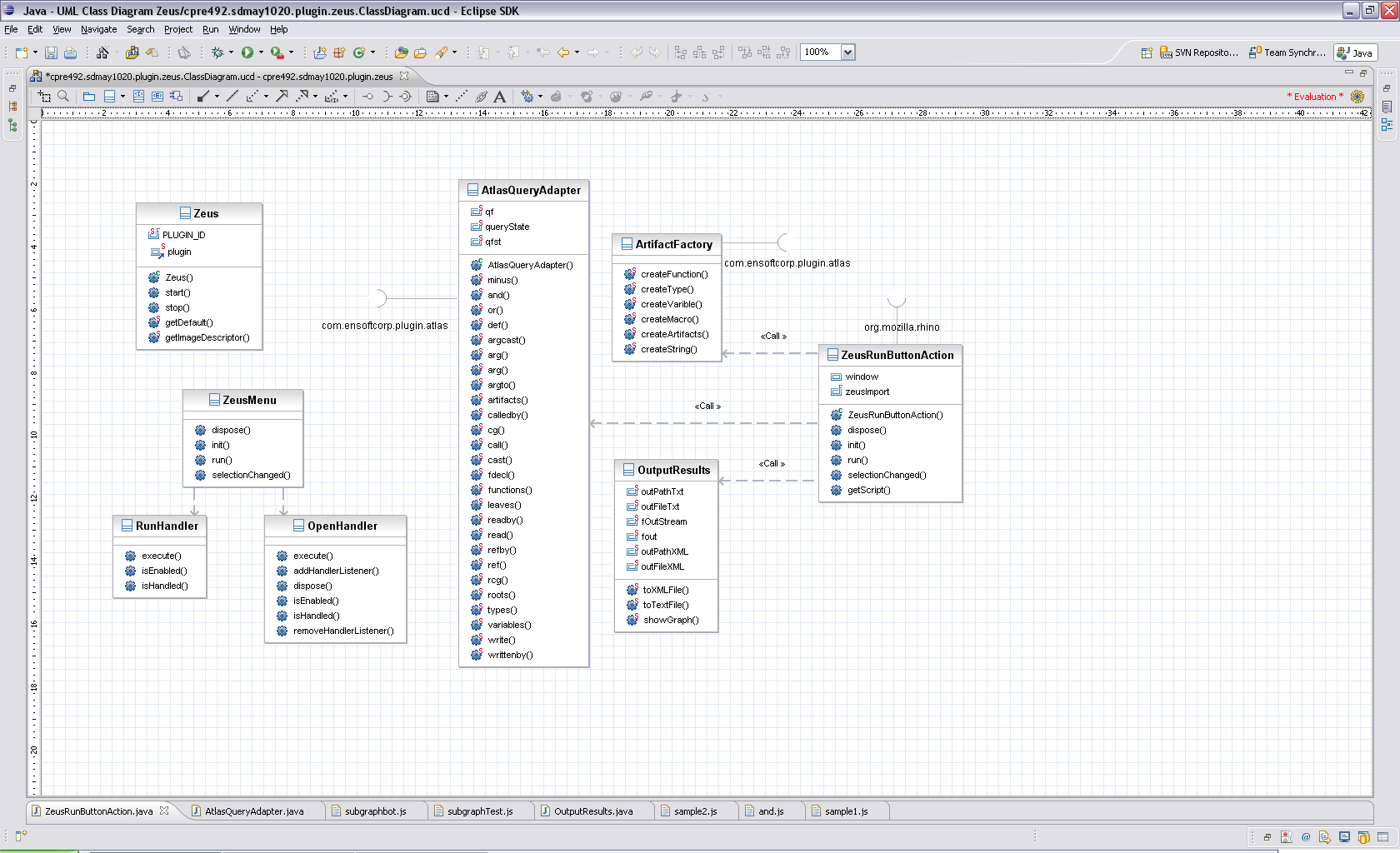
### 10.1.4. Query Script Parser Module

This module takes a query script as input.

# 11. Detailed Design

Figure 8 depicts the relationships among the Java classes in the system. The ZeusPlugin class is the main class for the plugin and contains one ZeusView object, which handles the changes to the Eclipse GUI. The ZeusView class calls the RunHandler to handle the event that occurs when the user selects the option to run the query script. RunHandler will then launch the RhinoParser, which will handle the parsing of the input JavaScript query and call the appropriate method in AtlasQueryAdapter. The results from the AtlasQueryAdapter method calls are stored as a collection of Atlas artifacts.

Figure 11 - Class Diagram



## 11.1. Class Descriptions

To obtain more in-depth information on the class descriptions, see the JavaDocs in Appendix C.

# 12. Test and Evaluation Plan

The client has previously created Atlas query sequences for other research and educational applications. These Atlas query sequences will be written as scripts, which will then be used for system test cases. We will use the results of queries that have been already created by the client for testing as oracles in the testing process.

The team will conduct the tests in JUnit, where we can test small portions of the code and the code as a whole. Testing will be done to optimize program correctness. For operational tests like saving and loading queries, ad hoc methods will be used and documented.

# 13. Modification & Maintenance Recommendations

None at this time.

# 14. Traceability

The traceability matrix shown in Table 3 shows the relationships among requirements or stakeholder rationales and the design artifacts, code and test cases.

Table 6 - Traceability Matrix

| **No.** | **Use Case/Non-functional Description** | **Subsystem/Module/Classes that Handle it** | **Test Cases that Handle It** |
| --- | --- | --- | --- |
| 1 | System shall interpret JavaScript queries and execute corresponding Atlas queries. | Atlas Interface Module, Control Module, Query Script Parser Module |  |
| 2 | System shall provide user the ability to edit JavaScript queries. | Eclipse Module |  |
| 3 | System shall provide user the ability to delete JavaScript queries. | Eclipse Module |  |
| 4 | System shall provide user the ability to store JavaScript queries. | Eclipse Module |  |
| 5 | System shall provide user the ability to load JavaScript queries. | Eclipse Module |  |
| 6 | System shall display script query results as an Atlas artifact list. | Atlas Interface Module, Control Module, Eclipse Module |  |
| 7 | System shall display script query results as an Atlas graph. | Atlas Interface Module, Control Module, Eclipse Module |  |
| 8 | Zeus shall be compliant with Eclipse Ganymede 3.4.2. | Control Module |  |
| 9 | System shall not be OS specific (although its dependencies may be). | Eclipse Module |  |
| 10 | System shall fail gracefully with an appropriate error message and will not close the program. System shall not be subject to stack overflow creating system failure. | Control Module, Eclipse Module |  |
| 11 | System shall not crash due to invalid input. | Control Module, Query Script Parser Module |  |
| 12 | System shall not cause memory leaks. | Atlas Interface Module, Control Module, Control Module ,Query Script Parser Module |  |
| 13 | System shall include a user manual. |  |  |
| 14 | System shall display text in the user interface only in English. | Atlas Interface Module |  |
| 15 | System shall include a help document regarding its functionality. |  |  |

# 15. Conclusion

## 15.1. Project Team Information

### 15.1.1. Client

EnSoft Corp.

Dr. Suraj Kothari

3132 Coover Hall

Ames, IA 50011-3060

W: 515-294-7212

F: 515-294-8432

kothari@iastate.edu

### 15.1.2. Advisor

Dr. Suraj Kothari

3132 Coover Hall

Ames, IA 50011-3060

W: 515-294-7212

F: 515-294-8432

kothari@iastate.edu

### 15.1.3. Team Members

Cole Anagnost

Computer Engineering

212 South Hyland Avenue #6

Ames, IA 50011-0001

C: 515-554-1457

cole.anagnost@gmail.com

sd-may1020@iastate.edu

Kristina Gervais

Software Engineering

502 Hayward Avenue

Ames, IA 50014-7346

C: 507-828-3591

tina.gervais@gmail.com

sd-may1020@iastate.edu

Alex Kharbush

Software Engineering

2627 Kent Avenue

Ames, IA 50010

C: 319-471-5311

alexf4@gmail.com

sd-may1020@iastate.edu

## 15.2. Closing Summary

The overall project plan for Zeus explains how the student team will accomplish the goal of creating an Eclipse plugin that executes scripted Atlas queries. The project has been broken into eight tasks: problem definition, technology considerations, end-product design, end-product prototyping, end-product testing, end-product documentation, end-product demonstrations, and project reporting. Some of these tasks can be performed in parallel speeding up the project and ensuring that it will be completed by May 2010. Also, the software nature of the project suggests that little to no financial requirements are needed since labor is not considered.

The Zeus software system is an Eclipse plugin which parses query scripts, calls the corresponding Atlas queries and requests that Atlas display the results within the Eclipse IDE. The system design fulfills the functional and non-functional requirements determined from the problem statement, client requests, and course constraints. The user input is a query file written in JavaScript that is loaded into the system via the Eclipse IDE. The system has been designed with correctness, usability, robustness, efficiency, maintainability, and extensibility in mind. The system is broken into four modules, the Atlas Interface, Control, Eclipse plugin, and Query Script Parser. Each of these modules has a different role in the project, interface with each other and are dependent on each other. The requirements and design will be tracked using a traceability matrix.

# 16. Bibliography

## 16.1. Templates

IEEE SRS - **Software Requirements Specification** IEEE 830

IEEE SDD - Software Design Description IEEE 1016

End-Product Design Report Requirements – CprE 491 WebCT

Project Plan Requirements – CprE 491 WebCT

## 16.2. Content

www.eclipsepluginsite.com

# Apendix A – Communication Plan

Table 7 - Important Stakeholders

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Stakeholder** | **Resources (A)** | **Guidance (B)** | **Network (C)** | **Dependence Upon Stakeholder (A+B+C)** | **Their Power (D)** | **Your Power (E)** | **Total Power of Stakeholder (D-E)** |
| Students | 10 | 10 | 10 | 30 | 100 | 100 | 0 |
| Adviser/Client | 10 | 10 | 9 | 29 | 100 | 60 | 40 |
| Instructor | 8 | 2 | 4 | 14 | 80 | 60 | 20 |
| Grading Committee | 0 | 4 | 1 | 5 | 80 | 100 | -20 |

Table 8 - Essential Stakeholder Approach

|  |  |
| --- | --- |
| **Accommodate** | **Work With**  Students(30, 0)  Adviser/Client (29, 40) |
| **Work Around**  Grading Committee(5, -20) | **Ignore**  Instructor(14, 20) |

Table 9 - Essential Stakeholders' Power

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Stakeholder** | **Interest (A)** | **Agreement (B)** | **Support (C)** | **Agreement (A+B+C)** | **Integrity (D)** | **Trust (E)** | **Influence (F)** | **Relationship Quality (D+E+F)** |
| Students | 10 | 10 | 10 | 30 | 100 | 100 | 100 | 300 |
| Adviser/Client | 10 | 10 | 10 | 30 | 100 | 100 | 100 | 300 |
| Instructor | 3 | 7 | 4 | 14 | 100 | 100 | 0 | 200 |
| Grading Committee | 1 | 3 | 2 | 6 | 100 | 90 | 0 | 190 |

Table 10: Essential Stakeholder Support

|  |  |
| --- | --- |
| **Yes Men** | **Allies**  Students(30, 300)  Adviser/Client (30, 300) |
| **Adversaries** | **Challengers**  Instructor(14, 200)  Grading Committee(6, 190) |

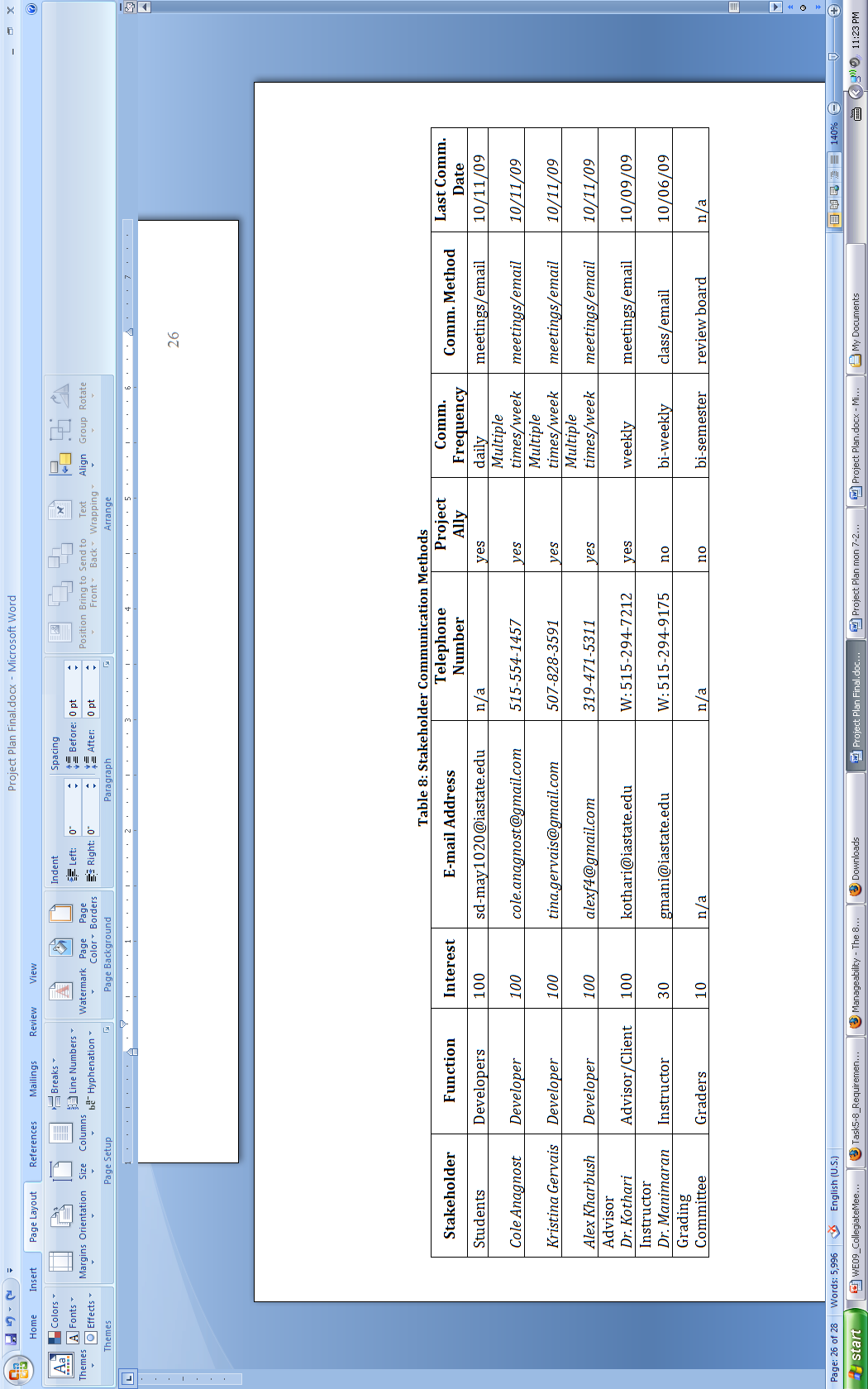


Table 11 – Stakeholder Communication Methods

# Appendix B – Work Breakdown Structure

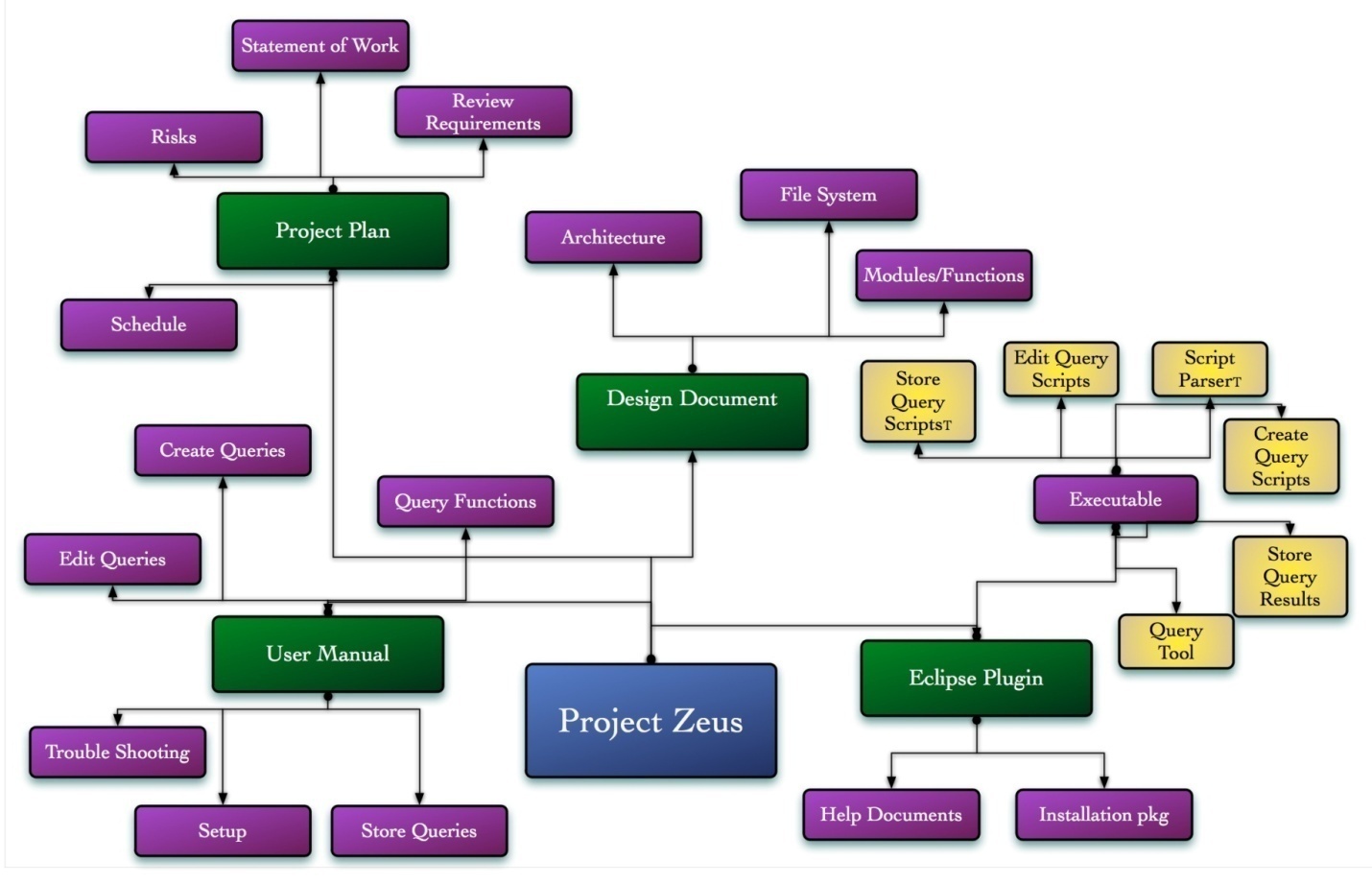


Figure 12 – Work Breakdown Structure

# Appendix C – JavaDocs