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. Concept of Operations 3

2.3. Operating Environment 4

2.4. Intended Users & Intended Uses 4

2.4.1. Intended Users 4

2.4.2. Intended Uses 4

2.5. Assumptions & Limitations 4

2.5.1. Initial Assumptions 4

2.5.2. Initial Limitations 5

2.6. Market & Literature Survey 5

2.7. Expected End-Product & Other Deliverables 5

2.7.1. End-Product Deliverables 5

2.7.2. Administrative Deliverables 5

3. Proposed Approach 6

3.3. Technology Considerations 6

3.3.1. Script Storage 6

3.4. Technical Approach Considerations 6

3.4.1. Source Code Management 6

3.4.2. Coding Strategies 7

3.4.3. Development Process 8

3.5. Testing Requirements Considerations 8

3.5.1. Unit Testing 8

3.5.2. Integration Testing 8

3.5.3. System Testing 8

3.6. Security Considerations 9

3.6.1. Project Security 9

3.6.2. Product Security 9

3.7. Safety Considerations 9

3.8. Intellectual Property Considerations 9

3.9. Commercialization Considerations 9

3.10. Risks & Risk Management 9

3.10.1. Team Member Loss 9

3.10.2. Acquisition of Atlas APIs 9

3.10.3. Lack of Eclipse plugin Creation Experience 9

3.11. Project Milestones & Evaluation Criteria 10

4. Statement of Work 11

4.1. Task 1: Problem Definition 11

4.1.1. Subtask 1.1: Problem Definition Completion 11

4.1.2. Subtask 1.2: End-Users & End-Uses Identification 11

4.1.3. Subtask 1.3: Constraint Identification 11

4.2. Task 2: Technology Considerations and Selection 12

4.2.1. Subtask 2.1: Identification of Possible Technologies 12

4.2.2. Subtask 2.2: Identification of Selection Criteria 12

4.2.3. Subtask 2.3: Technology Research 12

4.2.4. Subtask 2.4: Technology Selection 13

4.3. Task 3: End-Product Design 13

4.3.1. Subtask 3.1: Identification of Design Requirements 13

4.3.2. Subtask 3.2: Design Process 13

4.3.3. Subtask 3.3: Documentation of Design 14

4.4. Task 4: End-Product Prototype Implementation 14

4.4.1. Subtask 4.1: Development Environment Setup 14

4.4.2. Subtask 4.2: Identification of Prototype Limitations and Substitutions 14

4.4.3. Subtask 4.3: Implementation of Prototyped End-Product 14

4.5. Task 5: End-Product Testing 15

4.5.1. Subtask 5.1: Test Planning 15

4.5.2. Subtask 5.2: Test Development 15

4.5.3. Subtask 5.3: Test Execution 15

4.5.4. Subtask 5.4: Test Evaluation 16

4.5.5. Subtask 5.5: Documentation of Testing 16

4.6. Task 6: End-Product Documentation 16

4.7. Task 7: End-Product Demonstration 16

4.7.1. Subtask 7.1: Demonstration Planning 16

4.7.2. Subtask 7.2: Faculty Advisor Demonstration 17

4.7.3. Subtask 7.3: Client Demonstration 17

4.7.4. Subtask 7.4: Industrial Review Board Demonstration 17

4.8. Task 8: Project Reporting 17

4.8.1. Subtask 8.1: Project Plan 17

4.8.2. Subtask 8.2: Project Poster 18

4.8.3. Subtask 8.3: End-Product Design Report 18

4.8.4. Subtask 8.4: Project Final Report 18

4.8.5. Subtask 8.5: Weekly Email Reporting 18

5. Estimated Resources and Schedules 19

5.1. Estimated Resources 19

5.1.1. Personnel Effort Requirements 19

5.1.2. Other Resource Requirements 19

5.1.3. Financial Requirements 19

5.2. Schedules 20

5.2.1. Project Schedule 20

5.2.2. Deliverable Schedule 20

6. Specific Requirements 21

6.1. Product Features 21

6.2. Functional Requirements 21

6.3. Constraints 21

6.3.1. Memory Constraints 21

6.3.2. Design Constraints 22

6.4. Software System Attributes 22

6.4.1. Reliability & Availability 22

6.4.2. Portability 22

6.4.3 Performance 22

6.5. Other Requirements 22

6.6. Interface Requirements 22

6.6.1. Software Interfaces 22

6.6.2. Hardware Interfaces 22

6.6.3. Communication Protocols and Interfaces 22

6.6.4. User Interface 23

6.6.5. System Input & Output 24

6.7. Operating Platform & Dependencies 26

6.8. Site Adaption Requirements 26

7. Design Goals 26

7.1. Correctness 26

7.2. Usability 26

7.3. Robustness 26

7.4. Efficiency 26

7.5. Maintainability 26

7.6. Extensibility 27

8. Decomposition Description 27

8.1. Module Decomposition 27

8.1.1. Atlas API Wrappers 28

8.1.2. JavaScript Parser 28

8.1.3. Output Results 28

8.1.4. Plugin 28

8.2. Concurrent Processes 29

8.2.1. Eclipse 29

8.2.2. Graphviz 29

8.3. States 29

9. Detailed Design 30

10. Test and Evaluation Plan 30

10.1. Testing Strategy 30

10.1.1. Testing Scripts 31

10.1.2. JUnit 31

10.2. Evaluation Criteria 31

10.2.1. Evaluating Testing Scripts 31

10.2.2. Evaluating JUnit 31

11. Modification & Maintenance Recommendations 32

11.1. Modifications 32

11.2. Maintenance 32

12. Conclusion 33

12.1. Project Team Information 33

12.1.1. Client 33

12.1.2. Advisor 33

12.1.3. Team Members 33

12.2. Lessons Learned 33

12.3. Closing Summary 34

13. Bibliography 34

13.1. Templates 34

13.2. Content 35

Apendix A – Communication Plan 36

Appendix B – Work Breakdown Structure 39

Appendix C – JavaDocs 40

Appendix C – Test Results 41

List of Figures

Figure 1: System Concept Sketch 3

Figure 2: Block Diagram 4

Figure 3: Project Schedule 20

Figure 4: System Use Case Diagram 21

Figure 5: User Interface Screenshot 23

Figure 6: JavaScript Query Example 24

Figure 7: System Output Example: Graph 25

Figure 8: System Output Example: XML Displayed in Microsoft Excel 25

Figure 9: Architecture Diagram 27

Figure 10: System State Diagram 29

Figure 11: Class Diagram 30

Figure 12: Work Breakdown Structure 38

List of Tables

Table 1: Revision History i

Table 2: Term, Acronym, and Abbreviation Definitions 1

Table 3 - Personnel Effort Requirements by Task 19

Table 4 - Non-personnel Resource Requirements & Costs 19

Table 5 - Labor Costs at $20 per Hour 19

Table 6: Deliverable Schedule 20

Table 7: Important Stakeholders 35

Table 8: Essential Stakeholder Approach 35

Table 9: Essential Stakeholders' Power 35

Table 10: Essential Stakeholder Support 36

Table 11: Stakeholder Communication Methods 37

# 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 |
| CprE 491/492 | Codes used by Iowa State University to represent the computer engineering Senior Design I and Senior Design II courses. |
| 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) |
| I/O | Input and output. |
| 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) |
| 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 all aspects of the Zeus project. It discusses the overall project description, the proposed approach, statement of work, resources and scheduling, software system requirements, software design goals, decomposition description, detailed design, testing plan and results, modification and maintenance recommendations, and team information.

The project overview consists of the need for the project, the operating environment, the intended users and uses, the assumptions and limitations, the market and literature survey, and the deliverables.  The proposed approach discusses the technology, technical approach, testing requirement, security, safety, intellectual property, and commercialization considerations for the project.  This section addresses the project risks and milestones.  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.

The specific requirements describe the product features, functional requirements, and non-functional requirements such as performance requirements, design constraints, software system attributes, and other requirements. Interface requirements and dependencies and dependencies are also discussed. 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, processes, and states. The detailed description explains the Java classes that will make up the Zeus system. The testing plan discusses the reasoning behind the testing that was performed in addition to test results. The modification and maintenance recommendations discuss where the system shortcomings exist. Lastly, this document includes information about the team.

# 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.

## 2.2. Concept of Operations

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 end-product, named Zeus, 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. 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

The solution will be an Eclipse plugin called Zeus. 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 2 shows, Zeus will interface with both Eclipse and Atlas. The user will interact with Zeus (create, load, and edit query scripts) 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 2 - Block Diagram



## 2.3. 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.

## 2.4. Intended Users & Intended Uses

### 2.4.1. 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.

### 2.4.2. 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.

## 2.5. Assumptions & Limitations

### 2.5.1. 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.

### 2.5.2. 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).

## 2.6. Market & Literature Survey

The purpose of the market and literature survey is to determine whether a tool we plan to develop exists, and if so, how well it works. After searching on the internet, we found a number of tools that had some similar attributes to what we were trying to accomplish. For example, query-based debugging tools do exist for OO languages, and they are designed to query the states and state history of objects. In addition, multiple semantic-based analysis tools are available for logic programming languages such as Prolog. There are also many static analysis tools designed for C/C++; however, they are based on syntactic analysis and model checking instead of semantic analysis. The closest thing we found was the Kaveri tool developed at Santos Laboratories, Kansas State University. It provides a scripting based query framework for Java applications accessible via the Eclipse framework and is powered by the underlying Indus toolkit. Although this is parallels what we are trying to do, the purpose of the underlying tool (not to mention the tool itself) differs from what the client needs.

## 2.7. 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.

### 2.7.1. End-Product Deliverables

Zeus Eclipse plugin: the end-product of the project

User Manual: a printable guide for the user

### 2.7.2. 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

## 3.3. Technology Considerations

### 3.3.1. 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

Portability

Standard file system with all OS – no need to install anything

Setup back up & restore features

Cons

Larger storage footprint

Slow lookup

Approach: Use a database system to store scripts.

Pros

Smaller size

File indexing

Setup back up & restore feature

Cons

Lacks portability

Need to install a database

Database system dependency

Result: A file system will be used to store query scripts due to portability and the fact that no other software dependencies are created. In addition the slower lookup and larger storage is not a concern of the client.

## 3.4. Technical Approach Considerations

### 3.4.1. 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.

### 3.4.2. 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.

### 3.4.3. Development Process

Multiple software development paradigms exist; however each has attributes that server different types of projects more effectively.

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

Shows progress

Allows for reviews

All team members have tasks to do until project is done

At the start of each iteration, modules are working together

Cons

A large quantity of work

May get stuck on one iteration for a while

Team members will have different amounts of work to do

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.

## 3.5. Testing Requirements Considerations

### 3.5.1. Unit Testing

Unit testing will be performed throughout development.

### 3.5.2. Integration Testing

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

### 3.5.3. System Testing

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

## 3.6. Security Considerations

### 3.6.1. Project Security

None

### 3.6.2. Product Security

None

## 3.7. Safety Considerations

None

## 3.8. 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.

## 3.9. Commercialization Considerations

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

## 3.10. Risks & Risk Management

### 3.10.1. Team Member Loss

One of the risks posed to any student project is the loss of a team member. However, due to the size of this team and the complexity of the project objective this is a major risk for this team. The 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).

### 3.10.2. 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 accurate and complete Atlas APIs If the APIs are delivered after November, 2009, the project scope may need to be reduced due to the time constraint placed on the project.

### 3.10.3. 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.

## 3.11. Project 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.

# 4. Statement of Work

## 4.1. 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

### 4.1.1. 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.

### 4.1.2. 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.

### 4.1.3. 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.

## 4.2. 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.

### 4.2.1. 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.

### 4.2.2. 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.

### 4.2.3. 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.

### 4.2.4. 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.

## 4.3. 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.

### 4.3.1. 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.

### 4.3.2. 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.

### 4.3.3. 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.

## 4.4. 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.

### 4.4.1. 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.

### 4.4.2. 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.

### 4.4.3. 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.

## 4.5. 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.

### 4.5.1. 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.

### 4.5.2. 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.

### 4.5.3. 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.

### 4.5.4. 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.

### 4.5.5. 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.

## 4.6. 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 and compiled the information into a single document.

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

## 4.7. 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*: The team discussed what is needed for each demonstration in order to show the end-product effectively. Dates for the different demonstration pre-runs for practice and demonstrations themselves were established. The team did demonstrations for their advisor, client, and the industry review board.

*Results:* The team has demonstrated the end-product to all necessary.

### 4.7.1. 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 demonstrate 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.

### 4.7.2. 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.

### 4.7.3. 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.

### 4.7.4. 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

## 4.8. 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.

### 4.8.1. Subtask 8.1: Project Plan

*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.

### 4.8.2. Subtask 8.2: Project Poster

*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.

### 4.8.3. Subtask 8.3: End-Product Design Report

See Subtask 3.3.

### 4.8.4. Subtask 8.4: Project Final Report

*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 all aspects of the project.

### 4.8.5. 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.

# 5. Estimated Resources and Schedules

## 5.1. Estimated Resources

### 5.1.1. Personnel Effort Requirements

Table 3 shows the hourly work breakdown for each team member. The differences in the hours for the various tasks are due to the role that each member plays in the team and the amount of effort each member contributed. 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 | 8 | 8 | 10 | 32 | 4 | 0.5 | 3 | 13 | 78.5 |
| Kristina Gervais | 12 | 10 | 30 | 35 | 18 | 7 | 5 | 60 | 177.0 |
| Alex Kharbush | 10 | 4 | 14 | 40 | 25 | 0.5 | 3 | 19 | 115.5 |
| Totals | 30 | 22 | 54 | 107 | 47 | 8 | 11 | 92 | **371** |

### 5.1.2. Other Resource Requirements

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

Table 4 - Non-personnel Resource Requirements & Costs

|  |  |
| --- | --- |
| **Item** | **Cost** |
| Printing Parts and Materials: | |
| a. Project Poster Including Printing | $45.00 |
| b. Report Printing Materials | $50.00 |
| Development Materials: | |
| a. Personal Computer *(each team member has own)* | $0.00 |
| b. Eclipse IDE *(free-ware)* | $0.00 |
| c. Atlas plugin *(provided by client)* | $0.00 |
| Overhead Materials: | |
| a. Meeting Rooms with projector and/or white board | $0.00 |

### 5.1.3. 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 overhead needed for the project will be covered by student tuition as the team met in campus facilities. The material costs are shown in Table 4, and the labor costs in Table 5 are calculated based on the personnel effort requirements (see section 4.1.1.).

Table 5 - Labor Costs at $20 per Hour

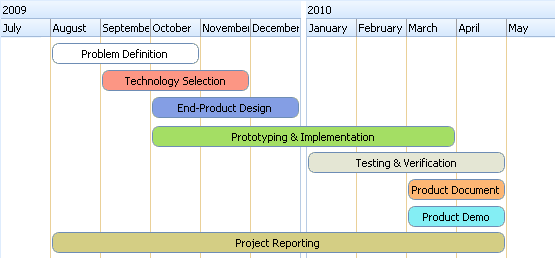
|  |  |  |
| --- | --- | --- |
| **Team Member** | **Total Hours** | **Labor Cost** |
| Cole Anagnost | 78.5 | $1,570 |
| Kristina Gervais | 177.0 | $3,540 |
| Alex Kharbush | 115.5 | $2,310 |
| **Total** | **371.0** | $7,420 |

## 5.2. Schedules

### 5.2.1. Project Schedule

The high-level project schedule is shown as a Gantt Chart in Figure 3. Each bar in the chart represents a separate task. The length and location of each bar indicates how much time is spent on each task and when each task shall be performed. Note that some of these tasks can be performed in parallel speeding up the project and helping to ensure that it will be completed by May 2010. The project manager plans what work is to be done each week based on this schedule, the milestone criteria discussed in section 3.11. and the deliverable schedule discussed in section 5.2.2.

Figure 3 - Project Schedule



### 5.2.2. Deliverable Schedule

Table 6 shows the deliverable schedule for the project. Two deliverables, the Project Plan and the Specifications & Design Report were completed during the fall semester. The remaining deliverables, the Project Poster, the Final Report, the Zeus Plugin, and the User Manual, were delivered at the end of the spring semester.

Table 6 –Deliverable Schedule

|  |  |
| --- | --- |
| **Deliverable** | **Due Date** |
| Project Plan | 10/12/09 |
| Design Report | 12/08/09 |
| Project Poster | 04/14/10 |
| Final Project Report | 04/28/10 |
| End-Product: Zeus Plugin | 04/28/10 |
| User Manual | 04/28/10 |

# 6. Specific Requirements

## 6.1. Product Features

The use case diagram in Figure 4 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.

Figure 4 - System Use Case Diagram



## 6.2. Functional Requirements

System shall execute script queries as Atlas queries.

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

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

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

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

System shall provide the user the ability to delete 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.

## 6.3. Constraints

### 6.3.1. Memory Constraints

None.

### 6.3.2. Design Constraints

The client has specified (directly or indirectly) the following list of constraints for the Zeus system.

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

System must integrate with the Eclipse development platform.

System shall function as an Eclipse plugin compliant with Eclipse Ganymede 3.4.2.

System input shall be written in JavaScript.

System will be written in Java.

## 6.4. Software System Attributes

### 6.4.1. Reliability & Availability

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

### 6.4.2. 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.

### 6.4.3 Performance

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.

## 6.5. Other Requirements

System shall include a user manual.

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

## 6.6. Interface Requirements

### 6.6.1. Software Interfaces

Zeus interfaces with the Atlas, Mozilla Rhino, and Eclipse software systems. Zeus will not provide any external software interfaces of its own. Zeus launches Atlas queries and obtains the results through Atlas API calls. The input query script is obtained from the Eclipse IDE and via calls to the Mozilla Rhino APIs determines the Atlas queries executed. The Eclipse IDE is responsible for launching Zeus via the configuration settings provided by the plugin.

### 6.6.2. Hardware Interfaces

The only hardware required for Zeus is a standard personal computer capable of running and operating system that has Java installed. This includes IO devices such as a keyboard, mouse, and monitor.

### 6.6.3. Communication Protocols and Interfaces

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

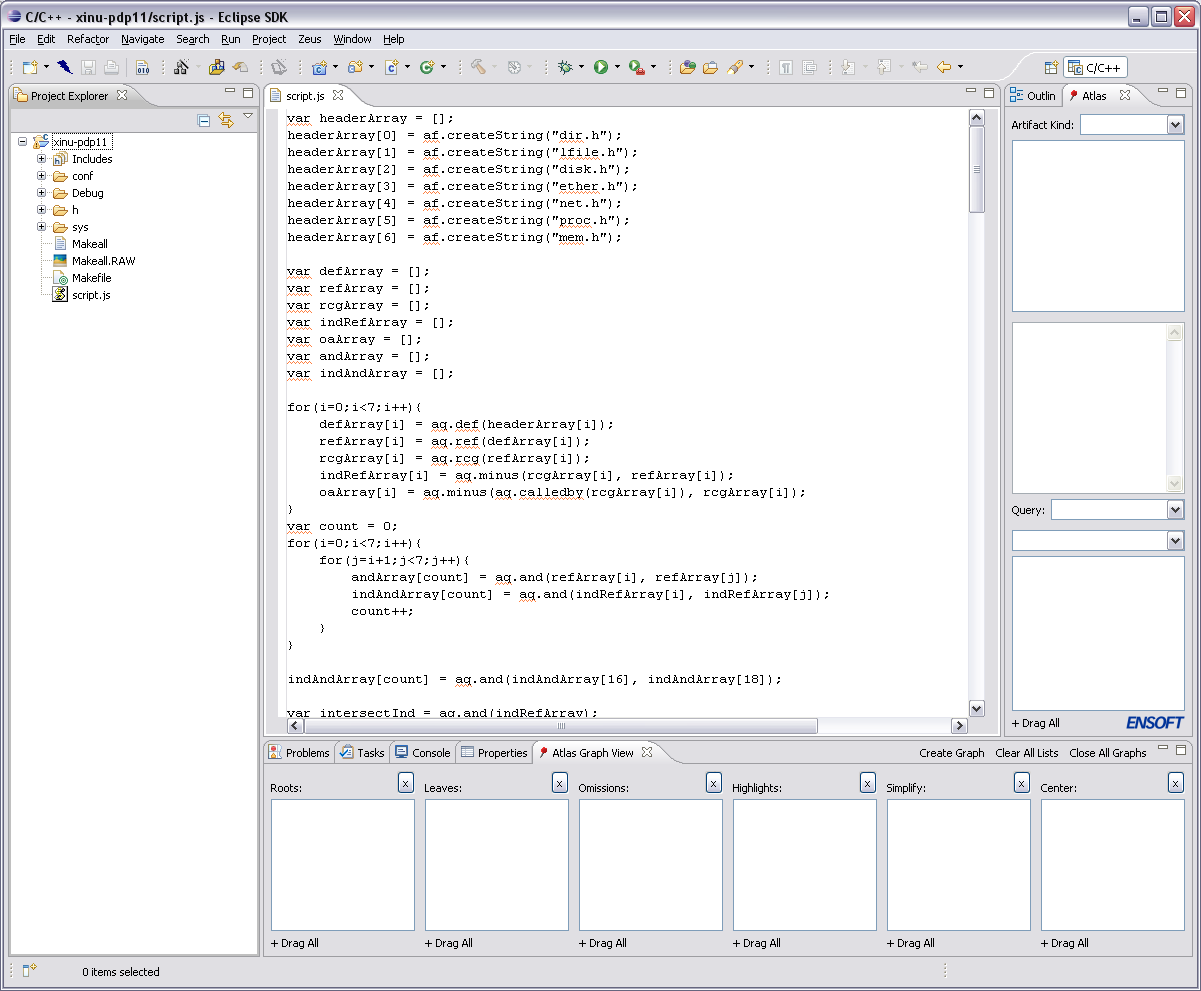
### 6.6.4. User Interface

The user interface requirements are as follows.

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

Due to these requirements and the plugin nature of the program, 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 save, edit, load, and delete functionalities are all provided by Eclipse IDE and require no additional work for the Zeus appliction. The user will be able to enter queries into the editor pane and can execute them by clicking the Zeus run button. The results of the queries will be displayed in Eclipse or written to a file. The output form depends on the commands in the script. Figure 5 is a screenshot of the applications user interface.

Figure 5 - User Interface Screenshot



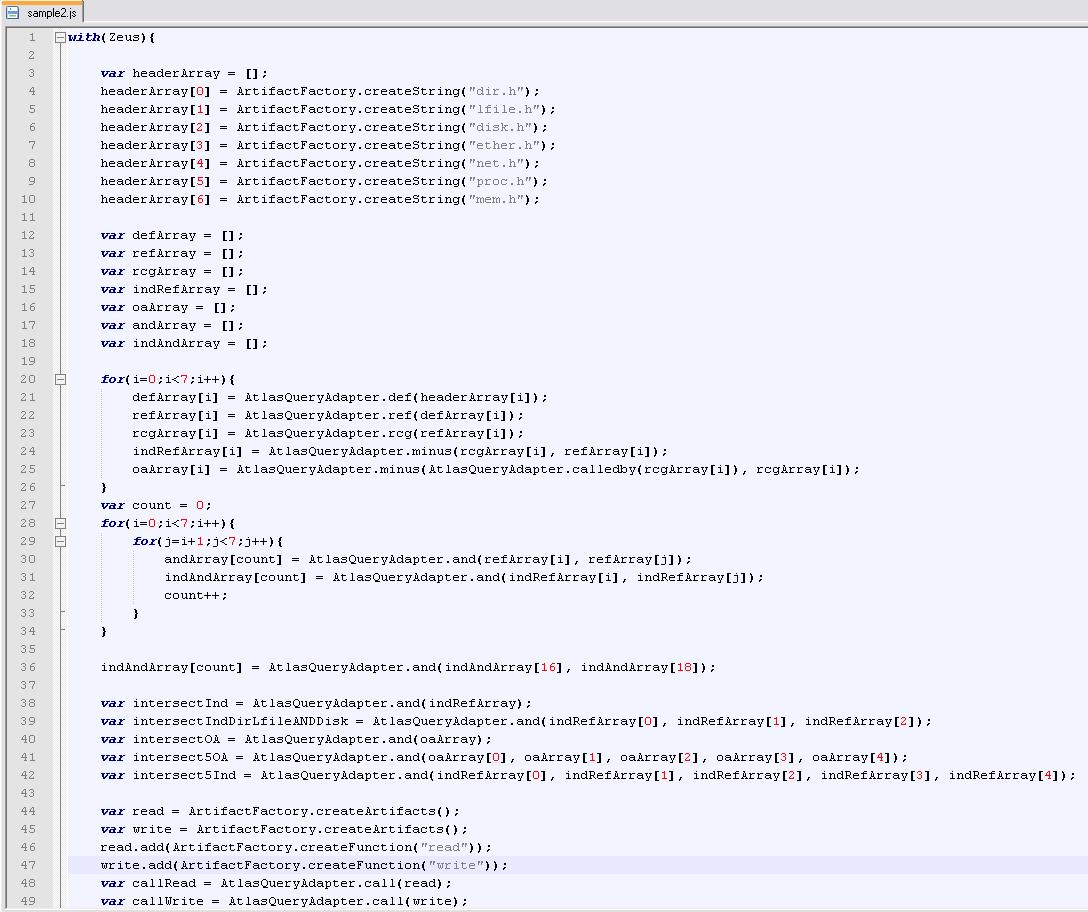
The red box highlights the button that launches the execution of the query script. The large blue box in the middle highlights the text editor supplied with Eclipse that will be utilized as the script editor. The green box to the left shows where the script file is shown in the Eclipse Project Explorer.

### 6.6.5. System Input & Output

#### 6.6.5.1. 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 6 shows a sample of what the JavaScript should look like.

Figure 6 - JavaScript Query Example



#### 6.6.5.2. System Output

All graphical system output will also be display via the Eclipse IDE. Displaying the results will be handled by a call to Atlas. The other output method that is available is writing to a file. Two file formats are supported: XML and text. The input script will determine which results are given to the user and how those results are formatted. Figure 7 shows the results displayed as a call graph, and Figure 8 shows the XML output as it is displayed by Microsoft Excel. For more details on output layouts see the User Manual.

Figure 7 - System Output Example: Graph

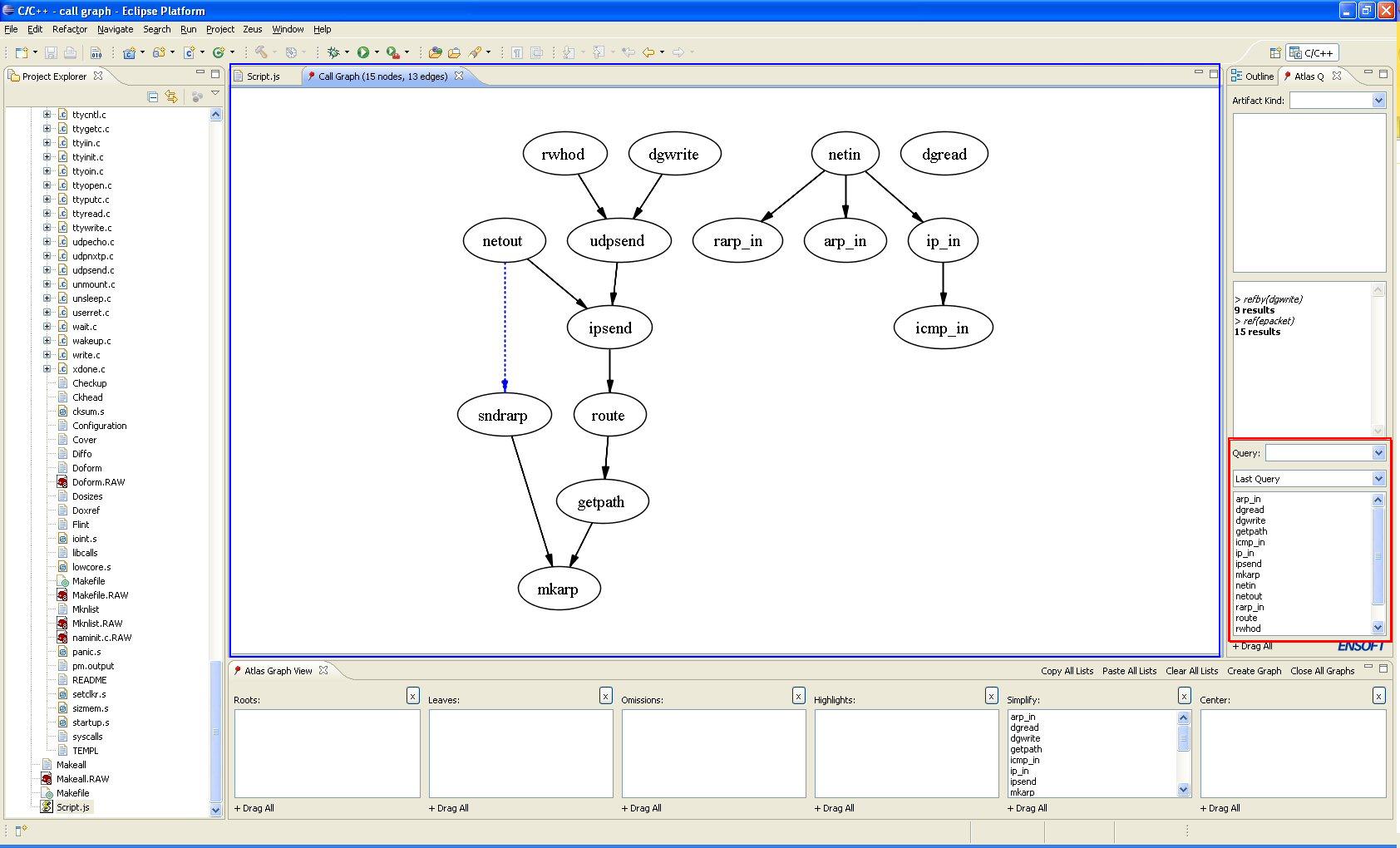
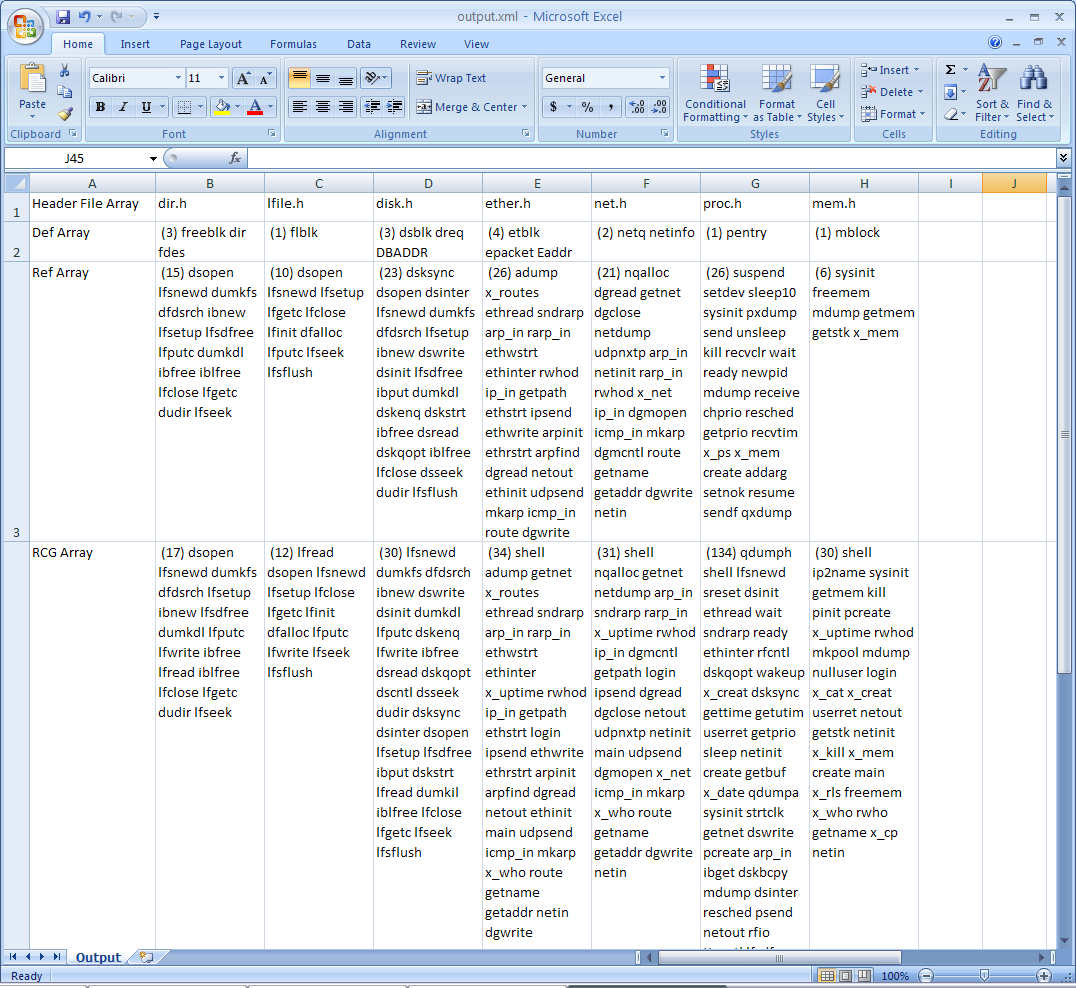


Figure 8 - System Output Example: XML Displayed in Microsoft Excel



## 6.7. Operating Platform & Dependencies

The software-operating environment will be the Eclipse IDE since it is 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. However, Zeus is dependent on Atlas (and inherently Graphviz). Depending on the Atlas version, it may or may not be portable to all operating systems.

## 6.8. 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.

# 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

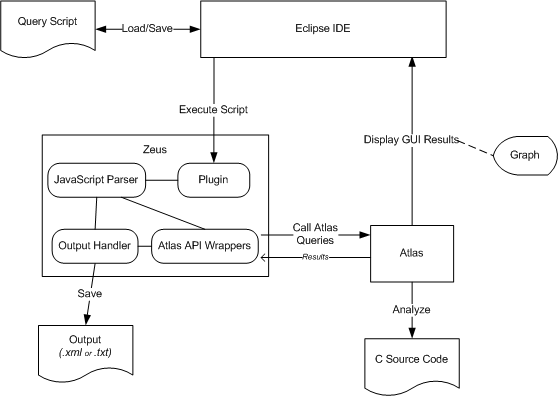
The system shall be designed to allow for the extension of different types of the same base category. This may occur by means of abstract classes and extension of functionality through creating additional class methods or an abstract class and several derived classes.

# 8. Decomposition Description

## 8.1. Module Decomposition

The architecture diagram in Figure 9 shows the division of Zeus into four conceptual subsystems: Atlas API Wrappers, JavaScript Parser, Output Handler, and Plugin. By designing the system in a modular manner (i.e. by grouping related tasks together), coupling is reduced. This improves the quality of system implementation and maintenance.

Figure 9 - Architecture Diagram



### 8.1.1. Atlas API Wrappers

The Atlas API wrapper component is responsible for all interaction with Atlas regarding queries and artifact creation. Therefore, it handles almost all calls to the Atlas APIs. The only exception to this is the creation of graphs.

Components:

*AtlasQueryAdapter Class:* wrapper for the Atlas query functions

*ArtifactFactory Class:* wrapper for the Atlas object creation functions

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

Services:Launches Atlas queries and creates artifact objects to pass as input to the queries

### 8.1.2. JavaScript Parser

The script parser component 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.

*ZeusRunButtonAction Class:* button icon is clicked.

*RunHandler Class:* launches script parsing/execution when the run menu option is selected.

**Services:** Parses JavaScript input and executes corresponding Java methods

### 8.1.3. Output Results

The output results component is responsible for provided all system output. It performs the Atlas query calls to generate a call graph in addition to (over)writing the output.txt and output.xml files.

Components:

*OutputResults Class:* provides three manners in which to output the results: graph, text file, XML file

**Services:**  Displays the specified Atlas artifacts as a call graph or writes them as a list to a text file or as a hierarchy that can be displayed as a table to an XML file

### 8.1.4. Plugin

The Eclipse plugin component 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.

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

*ZeusMenu Class:* contributes Zeus menu components to the Eclipse GUI.

*OpenHandler Class:* provides file browser window to the user so they can select a file to open and immediately run.

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

## 8.2. Concurrent Processes

### 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. States

The state diagram in Figure 10 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.

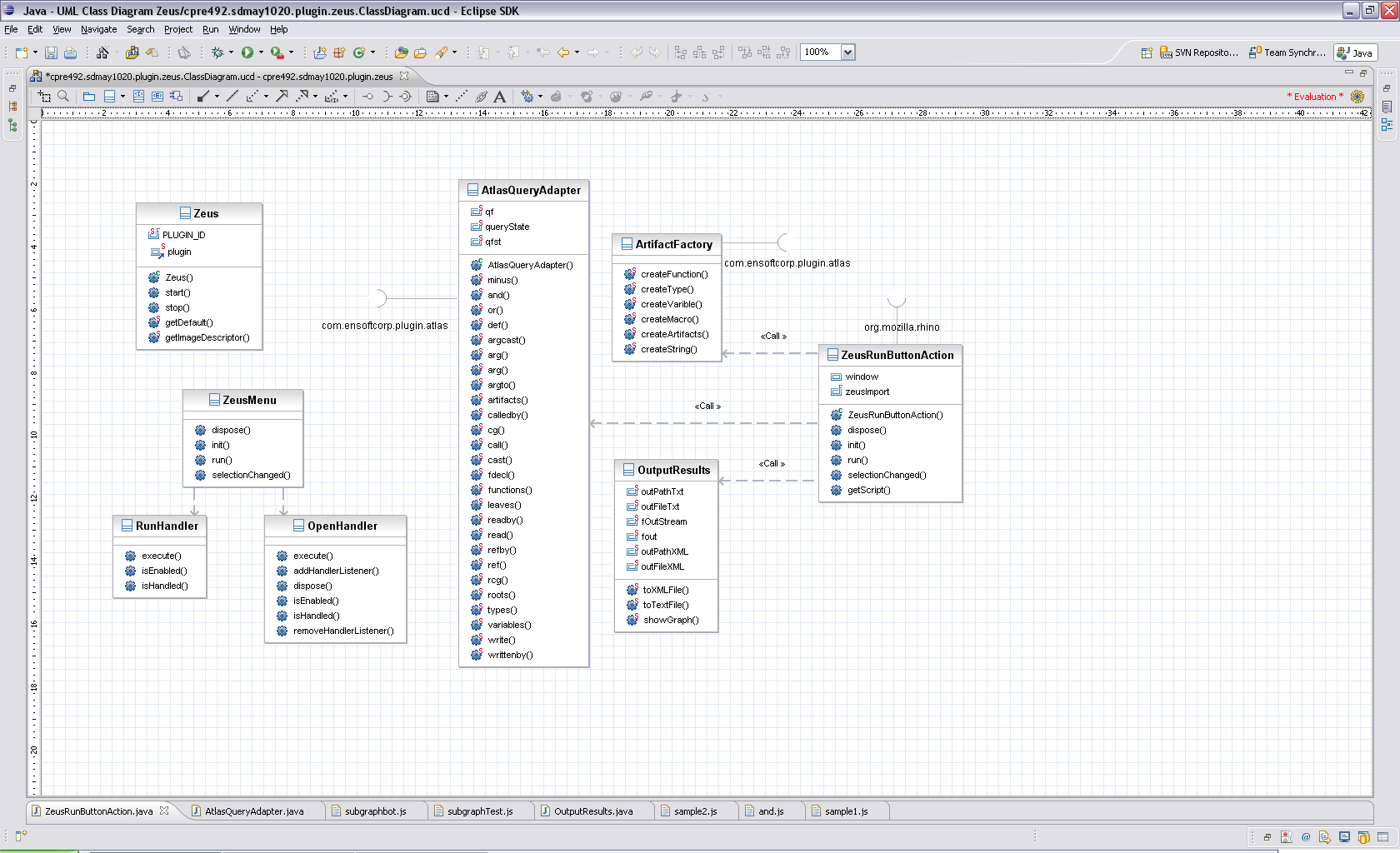
Figure 10 - System State Diagram



# 9. Detailed Design

Figure 11 depicts the relationships among the Java classes in the system. The Zeus class is the main class for the plugin. The ZeusMenu class handles the addition of a menu to the Eclipse GUI. The ZeusMenu class instantiates RunHandler or OpenHandler to handle the event that occurs when the user selects Run or Open in the menu, respectively. The handler classes provide the framework to open and launch the execution of a script (although this is not supported at this time). The ZeusRunButtonAction is responsible for execute the script in the current editor pane. This execution launches the Rhino JavaScript parser, which parses the input script. Based on the script, the parser calls the appropriate methods in AtlasQueryAdapter, ArtifactFactory, and OutputResults classes. To obtain more in-depth information on the class descriptions, see the JavaDocs in Appendix C.

Figure 11 - Class Diagram



# 10. Test and Evaluation Plan

## 10.1. Testing Strategy

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 also plans to 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 could be used and must be documented.

### 10.1.1. Testing Scripts

Due to the design, testing was predominately focused on system and integration testing. This is because writing unit tests for Eclipse plugins can be challenging due to the manner that Eclipse handles JUnit tests. Specifically, to test the features of the plugin a new version of Eclipse is launched with the plugin enabled, the test methods are called, and Eclipse is shut down. Due to some multi-threading occurring during the Eclipse load, testing any methods not directly invoke by Eclipse was not possible because they are typically executed prior to Atlas being fully loaded. Therefore, to test the invocation of Atlas calls the following test scripts were written.

Atlas API Wrapper Test Scripts

* Each call one API query function
* Results compared to same query made manually via Atlas

Graph Test Scripts

* Only create a graph based on given input
* Visually compared to the graph made in Atlas with the same input

Research Based Test Scripts

* Most complicated test scripts
* Use loops to call same query multiple times
* Make many different API calls
* Compose and project query results within the script
* Manually compared results of the scripts to previous research results

### 10.1.2. JUnit

Due to the limitations of JUnit testing of Zeus discussed in section 10.1.1., a limited number of JUnit tests will be created. JUnit tests are automated tests that provide unit testing of Java methods. The main components being verified via JUnit will be the invocation of Mozilla Rhino, interaction with Eclipse, and writing to output files. The JUnit tests shall be stored in a separate location from the source code so that they are not compiled with the deliverable.

## 10.2. Evaluation Criteria

To see specifics of the evaluation criteria see the Appendix D: Test Results.

### 10.2.1. Evaluating Testing Scripts

Results from semantic analysis research done by the client manually via Atlas were the oracles for the system testing. This research was provided to the team by the client, and the manual query sequences performed were gathered from this previously documented research The Zeus system test results were compared by hand results to the results of the manual queries to determine correctness. Our interest is not whether the research results are correct, but rather that Zeus generates the same results as Atlas does manually.

### 10.2.2. Evaluating JUnit

Unlike the system testing, the oracle for each JUnit test must be determined by the programmer. The JUnit framework will perform the expected to actual comparison automatically saving a lot of time during the testing project. In addition the team will review the tests and test results to verify that the JUnit tests thoroughly test the plugin features.

# 11. Modification & Maintenance Recommendations

## 11.1. Modifications

We have determined four areas where the Zeus application may be extended: auto-saving the Atlas graphs, adding support for future Atlas queries, exporting the results into different formats, adding a right-click option to run JavaScript files.

The ability to auto save graphs would help out in cases where many graphs are to be created via JavaScript. This feature could be implemented as an Atlas API call if the support exists or by obtaining and the graph information from Eclipse and saving it to a file.

The ability to specify where the user wants to save the output files and what they should be named is a feature that would be extremely useful. Currently the user needs to manually move and/or rename output files if he does not want to be overwritten. This could be done by adding an additional method to specify this information in the OutputResults class.

It may be the case that a user will want to export their data to a format to which Zeus does not currently output. Extra functions may be added to Zeus to allow users to export to different output formats. These should be added to the OutputResults class.

Many programmers have become accustomed to certain features in Eclipse. One of these features is the ability to right-click on a program and run it. Currently it is not that easy to execute a script with Zeus. Having the ability to right-click on a JavaScript file to run it would be an excellent addition to the Zeus program.

## 11.2. Maintenance

We have determined two maintenance concerns for the Zeus application: adding/removing support for future Atlas queries and additional system testing.

As Atlas is being actively updated, more queries will be created and/or deleted. Thus addition API calls will be made available (e.g. MACRO search support) or some will be removed. Adding/removing support for these future queries is crucial to extending the life of Zeus. This could be done by writing a new wrapper, recompiling the project, and redistributing the JAR files. Another way would be to provide a mechanism allowing end-users to add Atlas API calls to the existing Zeus wrapper classes without having to recompile the project.

The JUnit testing that was planned was never completed, and thus still should be done. Methods/components that should be tested are those that interact directly with Eclipse (e.g. OpenHandler) and those that write to the output files (e.g. OutputResults).

# 12. Conclusion

## 12.1. Project Team Information

### 12.1.1. Client

EnSoft Corp.

Dr. Suraj Kothari

3132 Coover Hall

Ames, IA 50011-3060

W: 515-294-7212

F: 515-294-8432

kothari@iastate.edu

### 12.1.2. Advisor

Dr. Suraj Kothari

3132 Coover Hall

Ames, IA 50011-3060

W: 515-294-7212

F: 515-294-8432

kothari@iastate.edu

### 12.1.3. 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

## 12.2. Lessons Learned

Configuration

Why it was a problem: Our largest problem was configuring our development environment. Atlas has many system requirements. It was very difficult to set up our versions of eclipse with all the necessary system requirements.

How we over came it: Eventually we got the client to package up all the required subsystems and correct configurations where we could just unpack a zip file filled with our development environment and easily program from there on.

Team Dynamics

Why it was a problem: Our team was very small and we had to rely on each other to get our work done. This became a problem when a team member got busy with other responsibilities the whole group would suffer.

How we over came it: As a group we had to keep moving forward.

Changing requirements

Why it was a problem: As Atlas is still being developed our group ran into a couple of changing requirements. We had to change some of our code to accommodate new feature request.

How we over came it: Our team was very agile and we were able to accommodate new requirements.

Market and Literature survey

Why it was a problem: We were required to do a Market and Literature Survey. This was a problem because we were never taught how to do they survey.

How we over came it: Kristina really did some independent research and taught herself how to write the Market and Literature Survey. She then took upon herself to write it for the group.

Interactions with developers:

Why it was a problem: We had a couple of problems with our communication chains. Sometimes messages were not passed along to the whole group. We found it difficult to have meetings with our client because there was confusion about what the client was trying to communicate.

How we over came it: We eventually worked out communications problems by adapting our questions to be more specific to the audience we asked them to.

## 12.3. Closing Summary

The overall project plan for Zeus explains how the student team accomplished their goal of creating an Eclipse plugin that executes scripted Atlas queries. The project was 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. This information as well as milestones and deliverables was used to create a schedule that ensured that the project was 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 a functioning 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 concepts, the Atlas API wrappers, the JavaScript parser, the Eclipse plugin, and results output mechanism. System and integration testing was performed and a list of maintenance and modification recommendations has been generated.

# 13. Bibliography

## 13.1. 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

## 13.2. 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) |

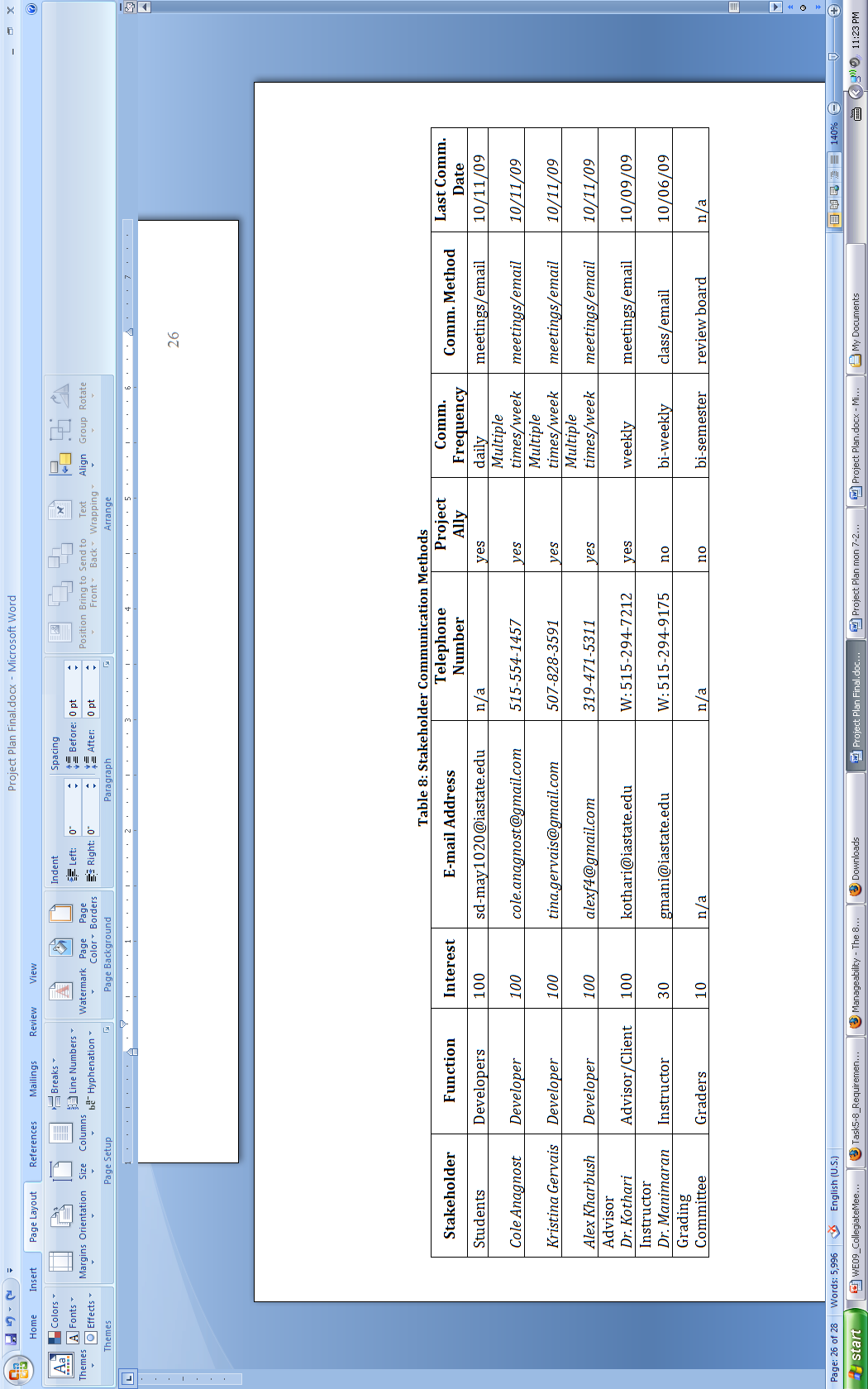


Table 11 – Stakeholder Communication Methods

# Appendix B – Work Breakdown Structure

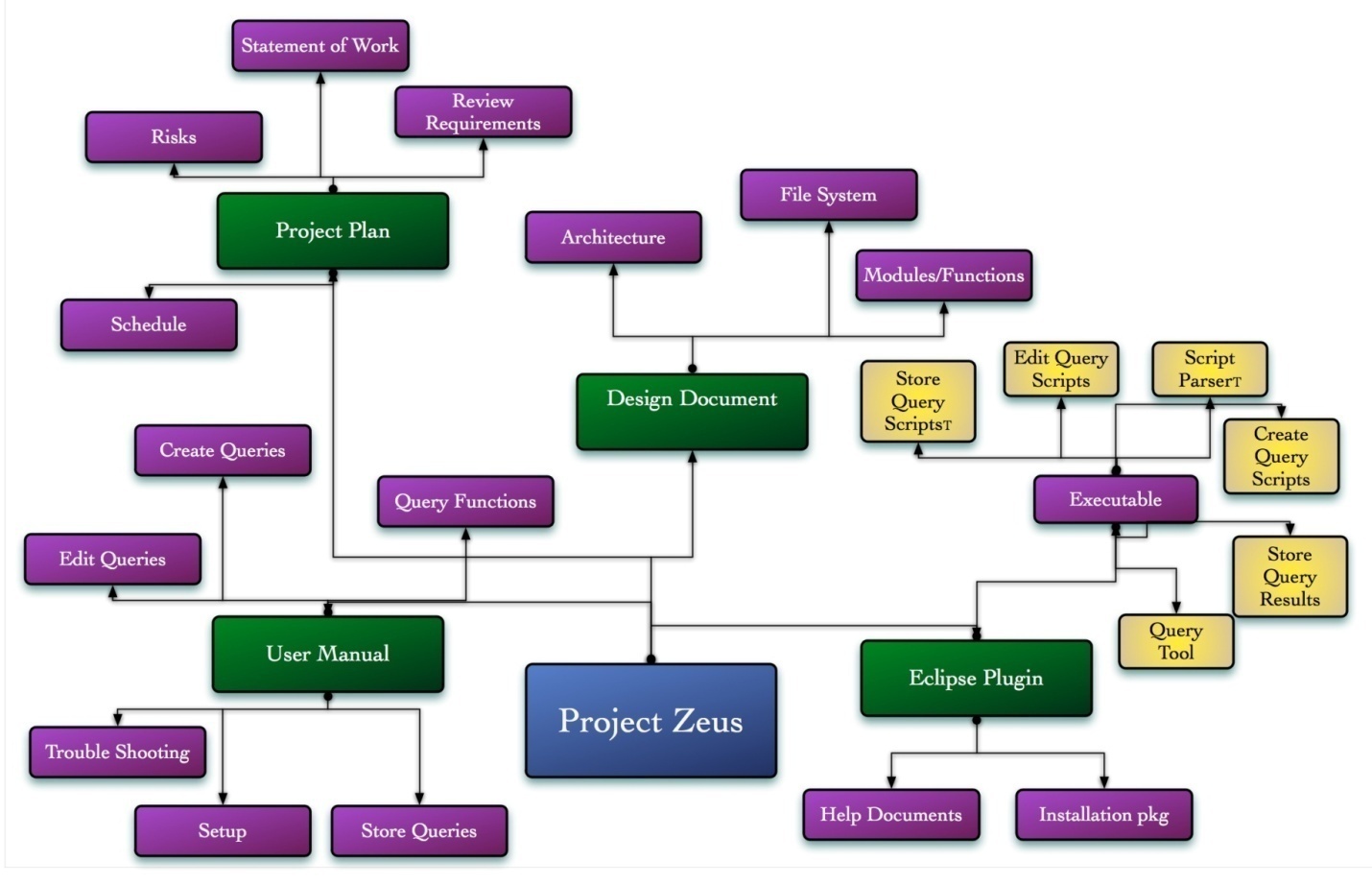


Figure 12 – Work Breakdown Structure

# Appendix C – JavaDocs

# Appendix C – Test Results