WEST VIRGINIA UNIVERSITY

COLLEGE OF ENGINEERING AND MINERAL RESOURCES

Lane Department of Computer Science & Electrical Engineering

DESIGN PROPOSAL

April 24, 2010

TEAM 6: Smart IDE

Ben Butler

Casey Corder

Zach Cowell

Arthur Otieno

FACULTY MENTOR/SPONSOR: Tim Menzies

INSTRUCTOR: Yenumula V Reddy

Table of Contents

1. Extended Problem Statement

1.1 Need

The history of programming and programming languages can span thousands of years. Since the development of the transistor in the early 50's, both the computers and the programming languages written to work with them have expanded many times over, often giving birth to new languages and getting rid of old outdated ones. The question is, are we able to keep up to pace as programmers. To do so, we need a programmer language.

Most programming languages are often inspired by other programming languages. A new language with a compiler or interpreter clearly fit the selection criteria into the long list of programming languages. New languageswith innovative features are listed if we can produce programs in this language. [1]

Consequently, as programmers we are again faced with another dilemma of choosing the best programming language from the long list of programming languages.

A gamble between runtime and amount of code might just sound right but as programmers we need a comprehensive environment where we can edit, compile, build and debug code. A version control system to manage changes made on code, a class browser and a hierarchy check would have a major impact on the field of programming. [2]

The Smart IDE typically presents a single program in which all development is done. This program typically provides many features for authoring, modifying, compiling, deploying and debugging software.

1.2 Objective

The team has laid out a course of objectives. The end objective is to have a fully functional integrated development environment. One objective is for the IDE to be able to read code. Next, the IDE should be able to write code. Writing includes the ability to edit; the use should be able to take a program and modify it, add to it, or remove from it. The system should identify syntactical errors in code, that is, it should know the rules of the language being used to write with. The system should support hypertext, being able to reference or use data outside of the system. The next objective is for the IDE to run the program. It should run short examples for the user to experiment with. The ability to store code is another goal. On compatible languages, the objective is to support at minimum Java, PHP, C, and Python.

1.3 Background

The early IDEs where done through a terminal. Since programming in early machines was done using flowcharts and entering programs with punch cards (or paper tape, etc) before submitting them to a compiler, the use of IDEs was impossible. Dartmouth BASIC was the first language to be created with an IDE (and was also the first to be designed for use while sitting in front of a console or terminal). Its IDE (part of the Dartmouth Time Sharing System) was command-based, and therefore did not look much like the menu-driven, graphical IDEs prevalent today [2]. However, code editing, compilation, debugging and execution in the early days weren't as consistent as the new age IDE.

The world's first integrated environment was developed by Softlab Munich in 1975.

Maestro I was installed by over 22,000 programmers worldwide. In 1995, Softbench was released but was immediately overshadowed by NetBeans. NetBeans started as a student project in the Czech Republic (originally called Xelfi), in 1996. The goal was to write a Delphi-like Java IDE in Java. Xelfi was the first Java IDE (Integrated Development Environment) written in Java, with its first pre-releases in 1997 [3].

Developing an IDE can be considered an open source project - which is a process. It takes time and determinations to find the right components to develop a good IDE that maximizes productivity therefore this remains an ongoing process as new ideas are generated and enthusiastic parties join in.

1.4 Stakeholder Goals

The following list is some key ideas that will be taken into account when developing the Smart IDE. The primary stakeholder for this project will be a programmer that would take advantage of the Smart IDE.

- The system must be able to be maintained such that defects can be corrected and features can be added/removed accordingly.
- During the development process, the Smart IDE will take advantage of existing technologies and features (e.g., Google code)
- The Smart IDE will help identify the similarities between programming languages rather than focuses on the differences.
- The design must be simplistic and efficient such that it does not consume excess resources while being accessed, yet shouldn't be simplistic such that quality and functionality is sacrificed.

1.5 Marketing Requirements

1. The Smart IDE must be user friendly to a broad range of programmers.
2. The system must be stable.
3. The Smart IDE must have the following tool set:
    a. Code reader
    b. Code writing tools (e.g., edit, syntax, hypertext)
    c. Code running in a test driven development
    d. Code storage with version control (SVN, repository manipulation, etc.)
4. The Smart IDE must take advantage of many different programming languages rather than any one language.
5. The Smart IDE must be able to handle small programming projects as well as larger scale projects.
6. Concise documentation must be provided with the product for an easier transition for the consumer and to help provide as little as a learning curve as possible.

1.6 Objective Tree

The objectives can be categorized into a tree. The three is represented as a hierarchy, from left to right fist, bottom to top next, representing the team's priority towards the completion of such objectives. The leaves are subsets of the second tier, themselves being subsets of the root node.

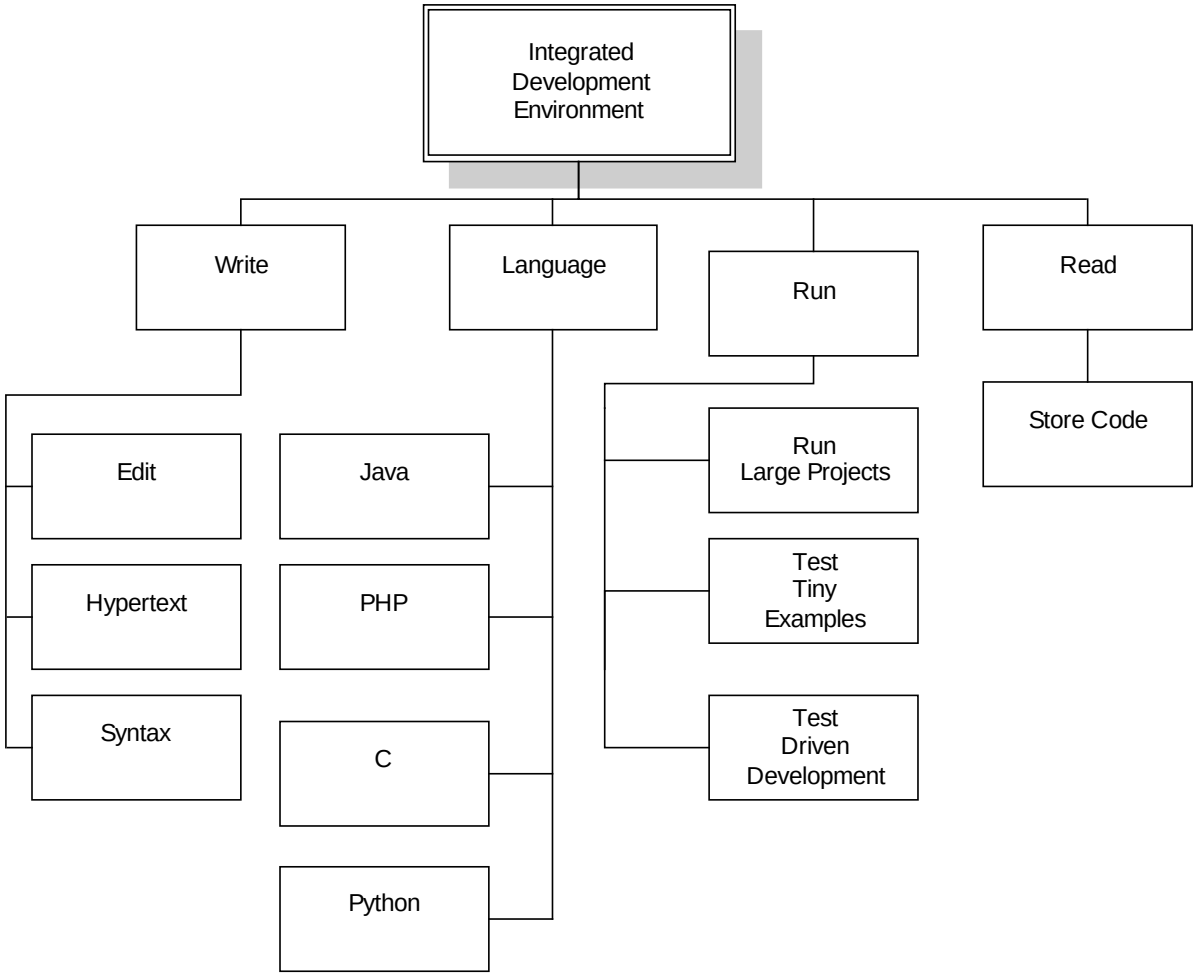


Figure 1: Smart IDE Objective Tree

1.7 Ranking of Needs

Our project is laced with many different needs. First and foremost, we need to be able to read and write our code. We will do this by saving code into a version control system to monitor changes made on the code. This is a basis for any programmer using an IDE. Secondly, the code written by the programmer needs to compiled, executed, and ran. Next, a debugging tool will be incorporated to help the programmer when compiling and looking through code. Lastly, we will try to then encompass as many different programming languages as we can, making the IDE as broad as possible.

2. Requirement Specification

2.1 Functional Requirements

This section of the document is a list of mandatory functions for the Smart IDE system.

2.1.1. CREATE NEW PROJECT

Will create a new code project. User should be able to specify what language(s) to be used for the project. Depending on the language specifications as laid out by the user, syntax highlighting and compile options will be available for the user.

2.1.2. SAVE PROJECT

Will save the project to a location of the user's choice. Saved projects can be opened at a later time.

2.1.3. OPEN PROJECT

Open an existing project within the Smart IDE.

2.1.4. COMPILE PROJECT

Depending on the specified language(s) by the user, the Smart IDE will compile the project and provide the user with output, if any. Both large scale and small scale projects will be supported for compilations.

2.1.5. VERSION CONTROL

Projects will have version control within the IDE itself rather than from a third-party application. This tool will allow the user to make changes to code and view past changes via a version tracker. If desired, code “rollbacks” can be made such that older revisions are used over newer ones.

2.1.6. BUG-TRACKING DOCUMENTATION

Will provide a list of known issues with the project. The bug-tracking module will have an identification number, a description of the issue, and whether or not the issue has been resolved.

2.1.7. CODE WRITER TOOLS

Writing code will be supported by the ability to mix “words and code” and storing it to a “.wak” file extension (words and kode). This will allow the programmer to mix up comments and code using a simple markup language to make easily, readable code that can be interpreted easily.

2.1.8. CODE BUILDING TOOLS

Will display file structures for .wak files and their dependencies. This will allow for automatic tool packaging of dependent files.

2.1.9. TEST DRIVEN DEVELOPMENT SUPPORT

Will allow the user to create tests for the project. These tests will evaluate specific aspects of the project and look for abnormal behavior. Tests will run to determine if the test passed or failed and output the results to a file.

2.2 Competitive Benchmarks

Many people learn a program using just a simple text editor, but eventually most end up using an Integrated development environment (IDE) for building applications. Any typical IDE has a set of tools that aids application development. The Smart IDE is based on a scripting language and on top of that, it has some features that makes it better than other typical IDEs.

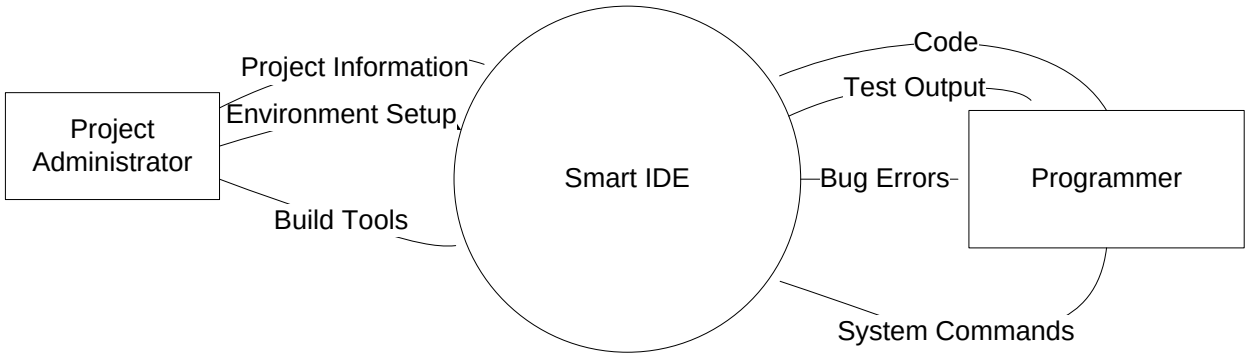
	Smart IDE	Eclipse	Netbeans	Sun Java Creator IDE
Write and edit source code	✓	✓	✓	✓
See errors as you type	✓	✓	✓	✓
See highlighted code syntax	✓	✓	✓	✓
Automate repetitive tasks	✓		✓	
Compile code	✓	✓	✓	
Browse class structures	✓	✓	✓	✓
Documentation	✓			✓
Templates for quick creation	✓			✓
Code-completion	✓	✓	✓	✓
Automated classes, methods, and properties creation	✓	✓	✓	✓
Version control	✓			
Integration with web application	✓			
Macros and abbreviations	✓	✓	✓	✓
Refactor code	✓	✓		✓

3. System Design

3.1 Scope

The KNIT system is the part of the program in which the user or administrator interacts with. KNIT will all the user to create their own language bindings to compile and language they want and even create their own applications. These simple roles will be elaborated on throughout this section of the document based on the outline given in the overall design document.

3.2 Context Diagram



3.3 User Definitions

All users are created/given permission to by the Administrator. This will give all users below the Administrative level access to the website in which they will be able to form development teams and create new language bindings and applications.

3.3.1 Administrator

The Administrator runs the system by creating and changing permissions to files, changing code inside the system to fix bugs, and developing the website to explain how to use the system. The available functionality is depicted in the next section, Interface Definitions, under the Administrative User Diagram.

3.3.2 Head Developer

The head of a development team will have different permissions than that of the development team members. The Head Developer will be allowed to see what everyone is working on and to make any changes necessary. In short, he simply manages the team that has been assembled. The available functionality is depicted in the next section, Interface Definitions, under the Head Developer User Diagram.

3.3.3 Team Members

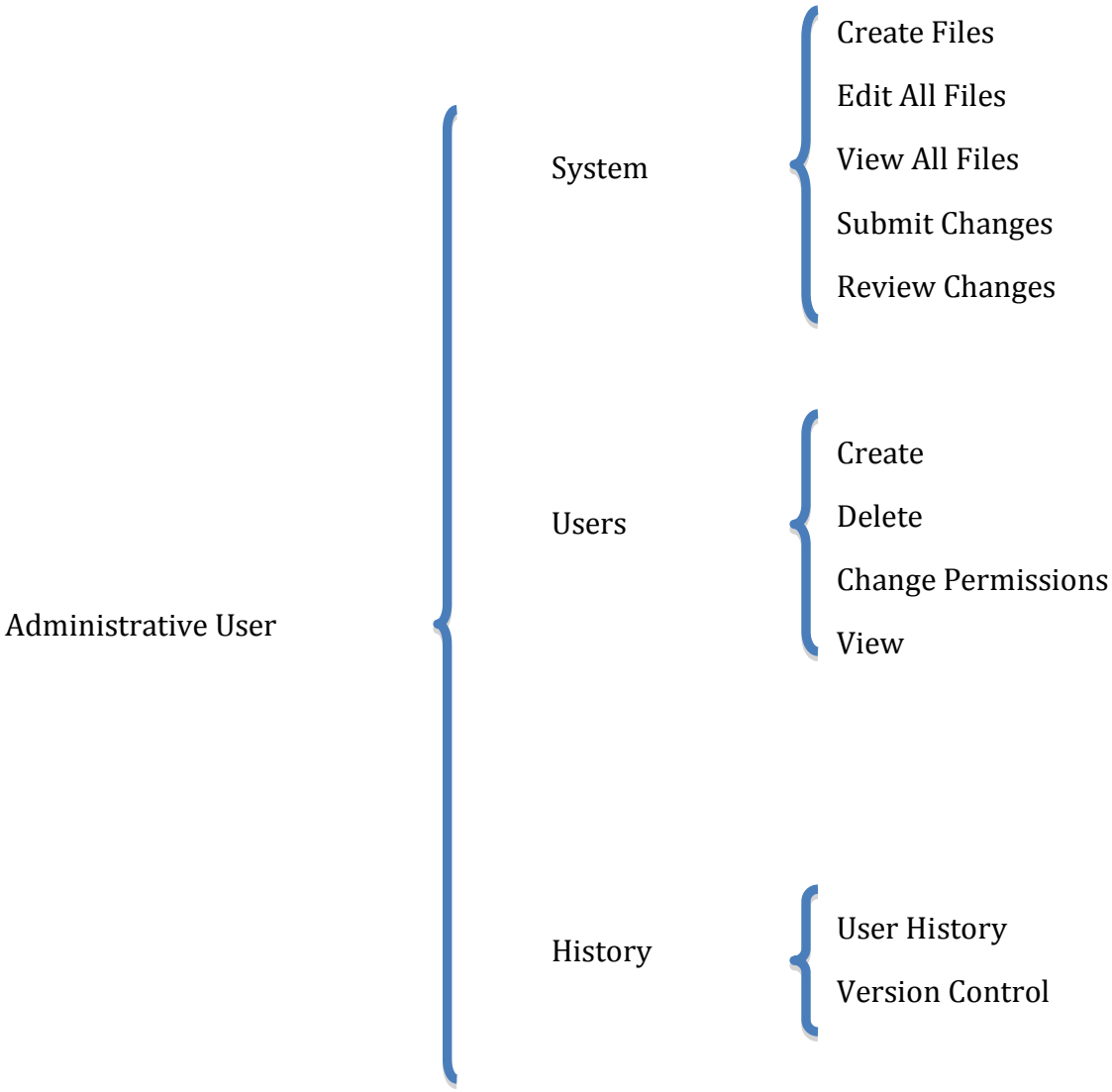
The development team members work on the bulk of the project. Not being able to mess with the rest of the system but simply what they are working on. The available functionality is depicted in the next section, Interface Definitions, under the Team Members User Diagram.

3.3.4 Lone User

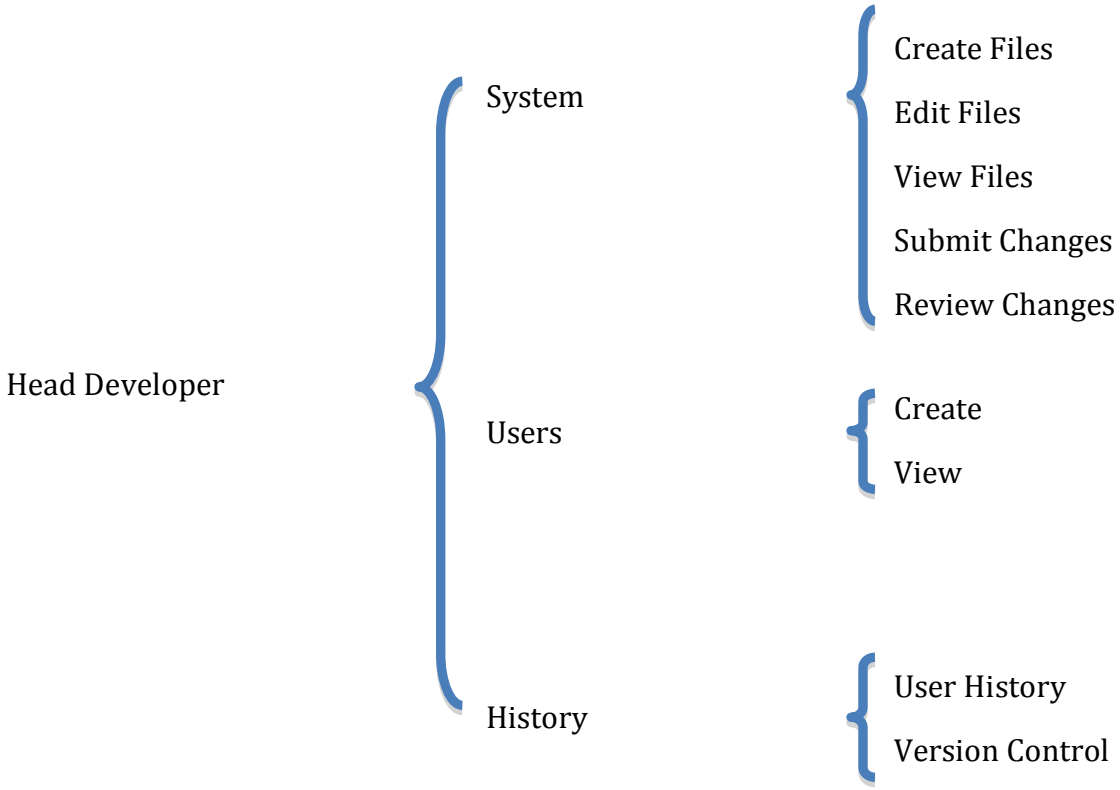
Simple users of the system are those who don't have a development team but plan to still develope on the system. Normally newer users will be lone users and will have most of the same permissions as a Head Developer would have. The available functionality is depicted in the next section, Interface Definitions, under the Lone User Diagram.

3.4 Interface Descriptions

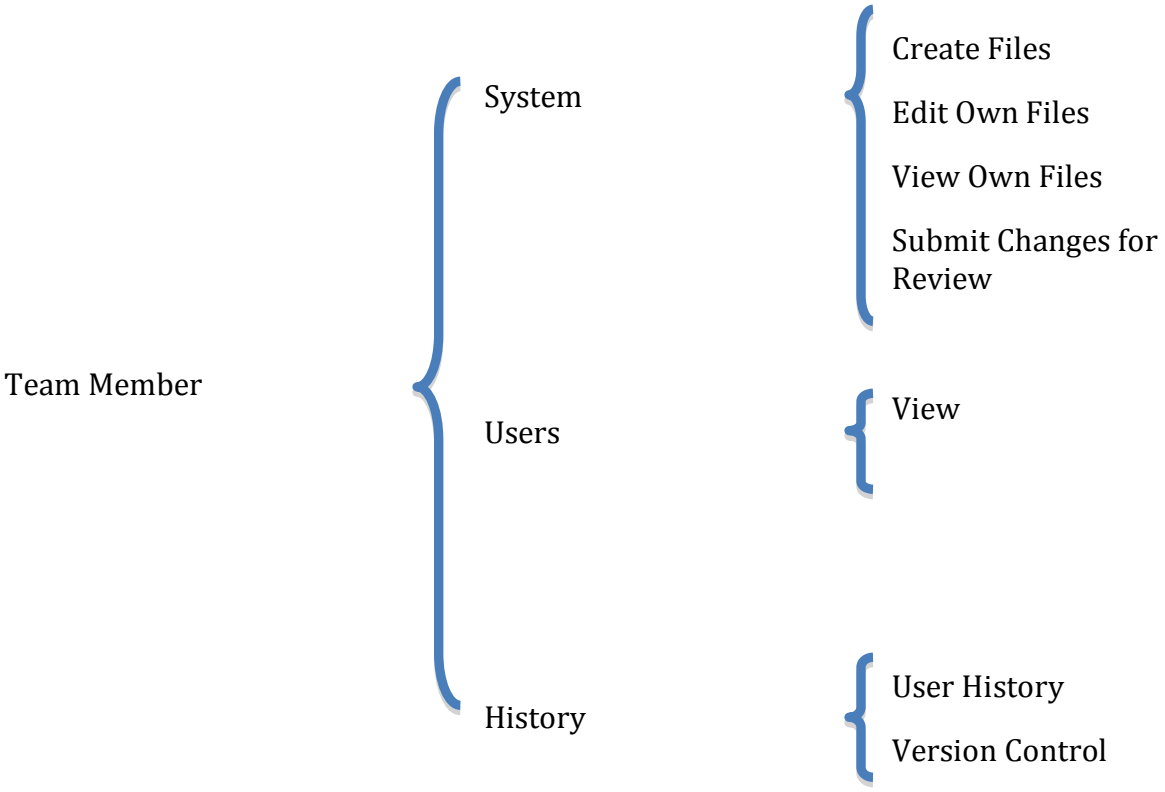
3.4.1 Administrator User Interface



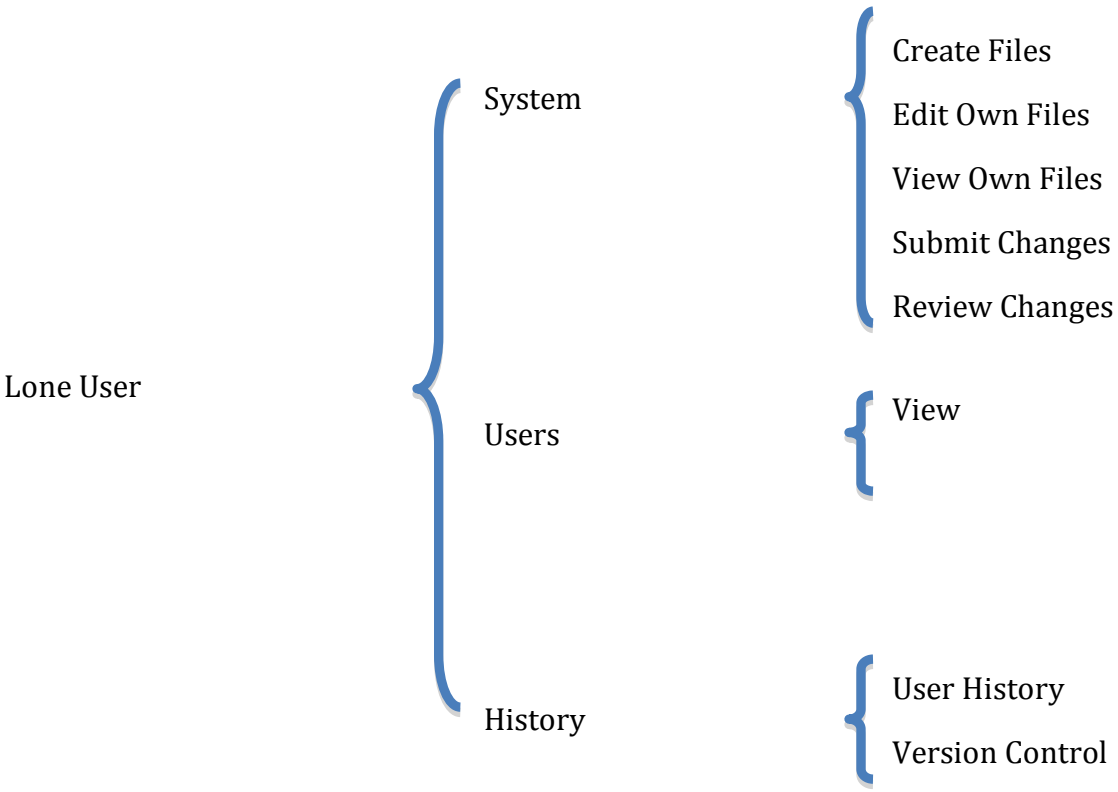
3.4.2 Head Developer User Interface



3.4.3 Team Member User Interface



3.4.4 Lone User Interface



3.5. System Functions

Name of function:	Create New Project
Description:	Creates a new source file for writing code.
Inputs:	Source File Name
Source of input:	User
Outputs:	A empty document, display asking the user if he/she wants to save
Destination of output:	Memory
Processing:	Take the Source File Name, validate name, create an empty file with the said name
Requirements:	Names of other existing files
Pre-condition:	A valid name.
Post-condition:	A new file created
Side Effects:	None
Responses to Abnormal Behavior:	Display: “Save current project?” Display: “Error”
Stability:	Stable
Necessity:	Required

Name of function:	Save Project
Description:	Saves the project
Inputs:	Selected Language, modified file
Source of input:	Editor of project
Outputs:	An updated document,
Destination of output:	Repository
Processing:	Save documents updated code, call Version Control
Requirements:	Identity of user modifying
Pre-condition:	Document exists
Post-condition:	Former document modified.
Side Effects:	None
Responses to Abnormal Behavior:	Display: “Not enough space”
Stability:	Stable
Necessity:	Required

Name of function:	Open Project
Description:	Opens a selected code source in an editor for the user to see.
Inputs:	File Name
Source of input:	User
Outputs:	The selected file
Destination of output:	Display device
Processing:	Search for the file, open the file
Requirements:	Database of files
Pre-condition:	File exists
Post-condition:	Contents of the file displayed
Side Effects:	None
Responses to Abnormal Behavior:	Display: “File does not exist” Display: “File not Supported”
Stability:	Stable
Necessity:	Required

Name of function:	Compile Project
Description:	Project translates code into commands for the computer to run,
Inputs:	Project application code
Source of input:	Repository
Outputs:	Translated Code
Destination of output:	Storage Device
Processing:	Open code, translate code, save translated code
Requirements:	Translation Instructions
Pre-condition:	The code is syntactically correct, the code source is saved
Post-condition:	A translated code is stored
Side Effects:	None
Responses to Abnormal Behavior:	Display: “Not Enough Space” Display: Syntax Errors Display: Compilation Unsuccessful
Stability:	Stable
Necessity:	Required

Name of function:	Execute
Description:	Runs an application
Inputs:	Compiled application
Source of input:	Storage Device
Outputs:	Execute Instructions
Destination of output:	Operating System
Processing:	Find executable file, tell system to run the file
Requirements:	Translated code
Pre-condition:	Code translated or able to be translated
Post-condition:	Code ran
Side Effects:	None
Responses to Abnormal Behavior:	Display: “Project needs to be compiled” Display: “Unable to execute”
Stability:	Stable
Necessity:	Required

Name of function:	Kill
Description:	Stops process during execution
Inputs:	Kill command
Source of input:	User
Outputs:	None
Destination of output:	Operating System
Processing:	System receives kill command, send instructions to end Processes for application
Requirements:	None
Pre-condition:	A process is running
Post-condition:	Processes has stopped
Side Effects:	None
Responses to Abnormal Behavior:	Display: “Operation not permitted”
Stability:	Stable
Necessity:	Required

Name of function:	Debug
Description:	Searches for possible errors and warns the user of it's location
Inputs:	Code
Source of input:	Memory
Outputs:	If errors, list of errors Else, display "No errors found"
Destination of output:	Display
Processing:	Run through code, find
Requirements:	Syntax Rules
Pre-condition:	A file exist to debug
Post-condition:	A list of bugs displayed
Side Effects:	None
Responses to Abnormal Behavior:	Display: "No file found"
Stability:	Stable
Necessity:	Required

Name of function:	Version Control
Description:	Keeps track of modifications to a source file
Inputs:	Modified File
Source of input:	Save Function
Outputs:	New file to version control
Destination of output:	Repository
Processing:	Take new File, Store Modification Date. Checks for merges, conflict, and changes.
Requirements:	Repository information
Pre-condition:	File is saved
Post-condition:	File added to version control
Side Effects:	None
Responses to Abnormal Behavior:	Display: “Program should have updated” Display: “Versions will be merged”
Stability:	Stable
Necessity:	Required

Name of function:	Customize
Description:	The IDE will be fully customizable by the users
Inputs:	Modification from the user
Source of input:	User
Outputs:	Modified system
Destination of output:	Storage device
Processing:	User opens an application, user modifies it, then saves it.
Requirements:	Compatible environment for modification
Pre-condition:	System accessible
Post-condition:	System upgraded
Side Effects:	None
Responses to Abnormal Behavior:	None
Stability:	Unstable
Necessity:	Required

Name of function:	Modify
Description:	Allows code source to be modified
Inputs:	Text, text commands
Source of input:	Users input device
Outputs:	Modified text
Destination of output:	Display
Processing:	User enters modifications, modifications put in memory
Requirements:	Modifiable file
Pre-condition:	File exists
Post-condition:	File modified
Side Effects:	None
Responses to Abnormal Behavior:	Display: “Unable to modify file”
Stability:	Stable
Necessity:	Required

Name of function:	Makefile
Description:	Defines coding language supported in specified directory. Hooks WAK source into knit
Inputs:	WAK files
Source of input:	Storage Device
Outputs:	Sequence of commands for execution by Unix
Destination of output:	Operating System
Processing:	Takes dependencies provided by user, and automatically performs updating tasks.
Requirements:	Rules
Pre-condition:	Directory exists
Post-condition:	Updated system
Side Effects:	None
Responses to Abnormal Behavior:	Display: “Process unexecutable”
Stability:	Stable
Necessity:	Required

Name of function:	txt.mk
Description:	For use by Makefile for text files in knit
Inputs:	Makefile
Source of input:	Knit
Outputs:	.txt file support
Destination of output:	Knit
Processing:	Define selected languages supported in a given directory
Requirements:	Language files
Pre-condition:	Dependencies exist
Post-condition:	Text files supported
Side Effects:	None
Responses to Abnormal Behavior:	Display: “Unable to process”
Stability:	Stable
Necessity:	Required

Name of function:	python.mk
Description:	Used by Makefile for python support.
Inputs:	Dependencies
Source of input:	Storage device
Outputs:	python support
Destination of output:	Makefile
Processing:	Makefile utilizes to define language in directory
Requirements:	Language Files
Pre-condition:	Dependencies exist
Post-condition:	Python supported
Side Effects:	None
Responses to Abnormal Behavior:	Display: “Unable to process”
Stability:	Stable
Necessity:	Required

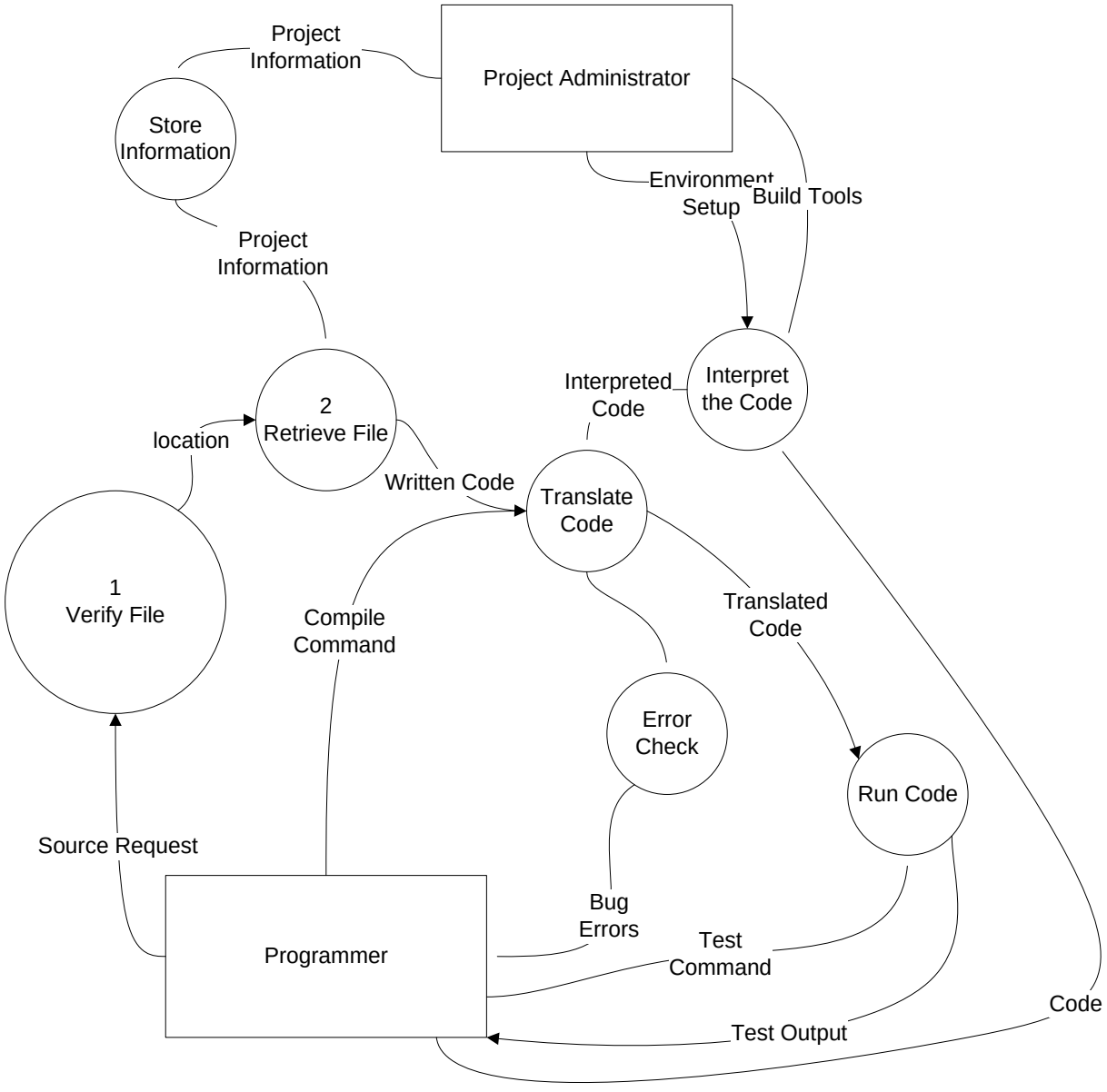
Name of function:	php.mk
Description:	Used by Makefile for php support.
Inputs:	Dependencies
Source of input:	Storage device
Outputs:	php support
Destination of output:	Makefile
Processing:	Makefile utilizes to define language in directory
Requirements:	Language Files
Pre-condition:	Dependencies exist
Post-condition:	php supported
Side Effects:	None
Responses to Abnormal Behavior:	Display: “Unable to process”
Stability:	Stable
Necessity:	Required

Name of function:	java.mk
Description:	Used by Makefile for java support
Inputs:	Dependencies
Source of input:	Storage device
Outputs:	java support
Destination of output:	Makefile
Processing:	Makefile utilizes to define language in directory
Requirements:	Language Files
Pre-condition:	Dependencies exist
Post-condition:	java supported
Side Effects:	None
Responses to Abnormal Behavior:	Display: “Unable to process”
Stability:	Stable
Necessity:	Required

3.6 Software Breakdown

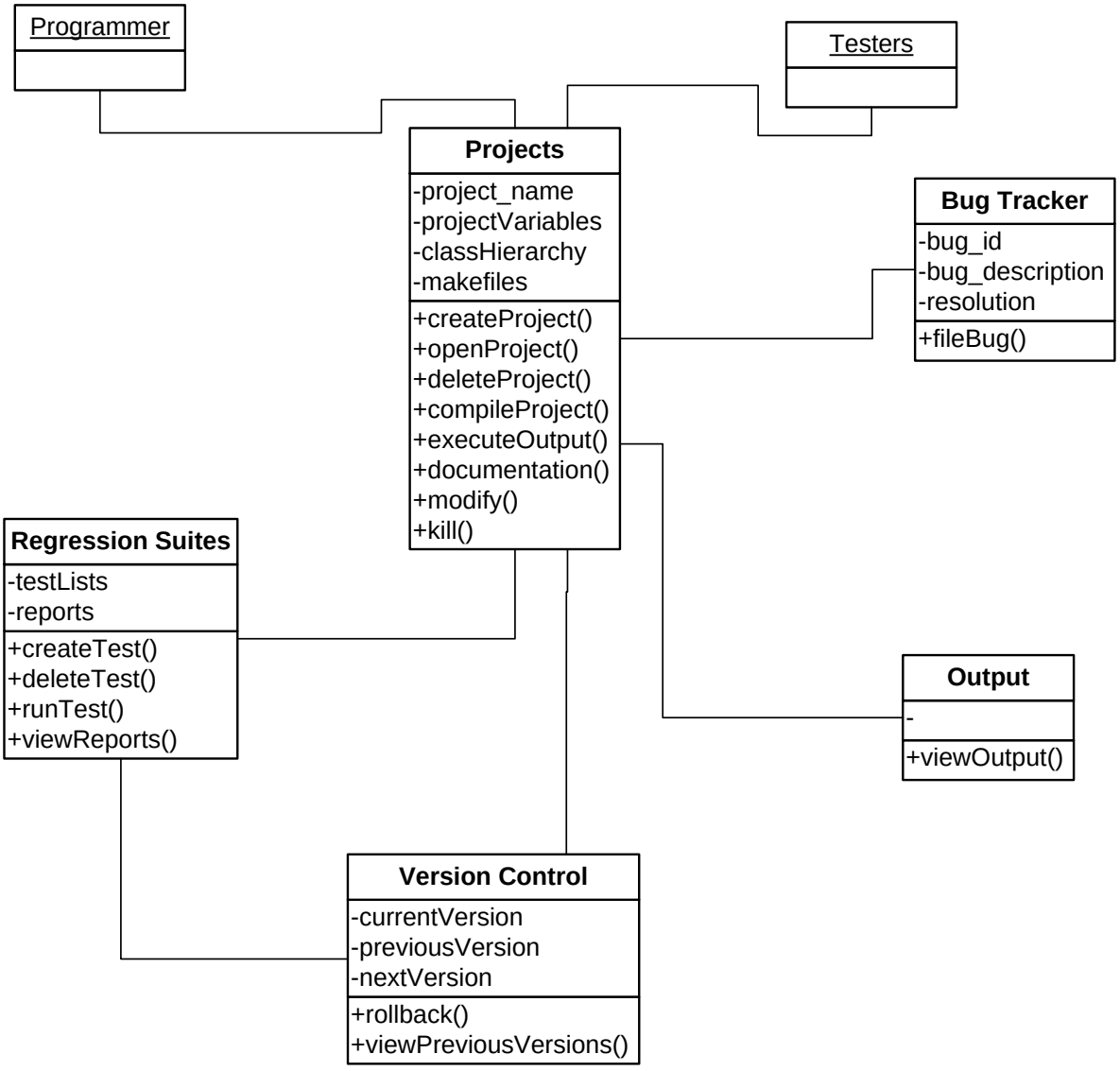
3.6.1 User Data Flow

The diagrams below show how data flows from the respective users and through the system.



3.7Database Breakdown

The following diagram shows the basic breakdown of the database that stores the code in the Smart IDE system. With the database, tracking of changes made on saved code is feasible.



3.8Acceptance Test

3.8.1 Create New Project

Description of input	Input	Expected Results
Blank project is created when no other project is open	Valid	Blank project is created
Blank project is created while a project is already open	Valid	Prompt user if current project should be saved, create blank project
Blank project is attempted to be created but fails	Invalid	Project doesn't get created and IDE stays in current state; ERROR

3.8.2 Save Project

Description of input	Input	Expected Results
User saves project from a dropdown menu of supported langages	Valid	Project is saved in the specified format
User saves project when disk space is full	Invalid	Project is not saved; error message
User saves project with valid filetype, file isn't saved or corrupted	Invalid	Project is not saved; error message

3.8.3 Open Project

Description of input	Input	Expected Results
User browses for project and selects a supported filetype to open	Valid	Project is loaded
User types filepath in manually of a supported filetype	Valid	Project is loaded
User types filepath of a file that doesn't exist	Invalid	No project is loaded, error message
User selects an unsupported filetype	Invalid	No project is loaded, error message

3.8.4 Compile Project

Description of input	Input	Expected Results
Project has correct syntax and a compile request is sent	Valid	Project is compiled
Project has incorrect syntax and a compile request is sent	Valid	Project shows syntax errorsa
Project has correct syntax but insufficient space for output file	Invalid	Project is not compiled, error message
Project has correct syntax and a compile request is sent; no output file	Invalid	Project was not compiled, error message

3.8.5 Execute

Description of input	Input	Expected Results
Project output is executed after a valid compilation	Valid	Project is executed
Attempt to execute a project before compilation	Invalid	Project is not executed, error message
Project is executed after a valid compilation but output is corrupted	Invalid	Project not executed, error message

3.8.6 Kill

Description of input	Input	Expected Results
Kill command is called, all processes involved with the IDE are halted	Valid	Processes are halted
Kill command is called, all processes are not killed	Invalid	Processes are not halted

3.8.7 Debug

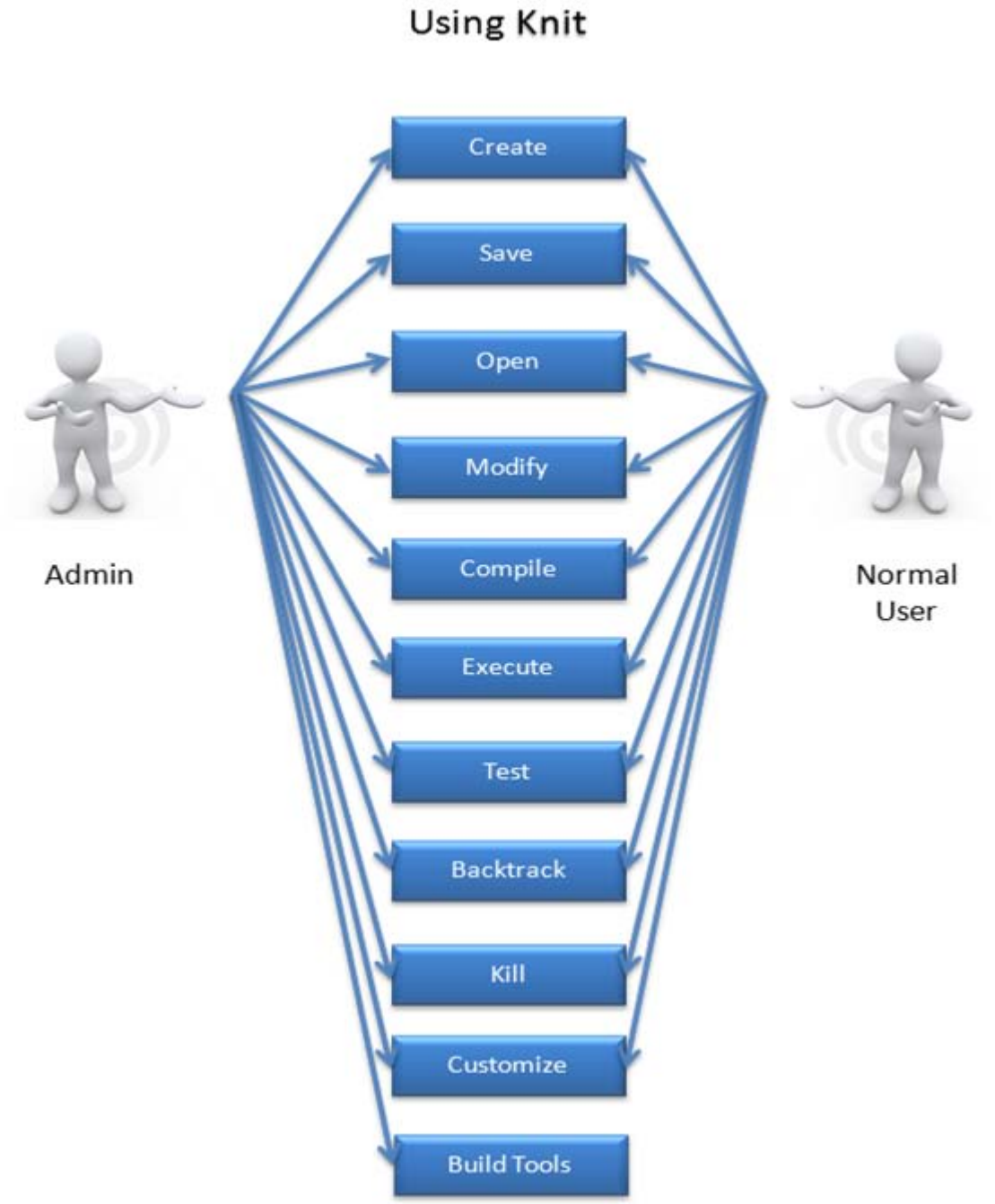
Description of input	Input	Expected Results
Debug command is called, allows use of stepping through lines of code	Valid	Program will step through the program
Debug command is called, allows use of stepping through lines of code	Invalid	Program does not step through the lines of code
Debug command is called without code being already compiled	Invalid	There is nothing to step through; error

3.8.8 Version Control

Description of input	Input	Expected Results
Code is modified and committed	Valid	Version updates
Code is modified and committed but not updated	Invalid	Version should have updated; error message
Code is modified and committed but a conflict is present with other modifications	Invalid	Versions must be merged before committ; error message

3.9 Use Cases

The Smart-IDE can either have an administrator and user or several users working on the same project at one time. Both the administrator and the user have same capability only in the case the administrator overlooks a project's progress. A use case diagram is illustrated and here are the use case descriptions;



Create Use-Case Description

Use-Case	**Create**
Actor	Admin, Normal User
Description	This use case occurs when the user creates a new project file in the system. It might be a copied template or a start from the scratch. The appropriate headers are applied to make the written program executable.
Stimulus	User creates a new file with the appropriate extension. A new makefile is then created from the provided templates to support the created file.
Response	The created file should appear on the user's directory.
Exceptions	No exceptions in this use-case.

Save Use-Case Description

Use-Case	**Save**
Actor	Admin, Normal User
Description	This use case occurs when user is ready to save the work that has been done.
Stimulus	User saves whatever file he/she was working on and updates the svn.
Response	The system will return an ok message with a log of all the changes that have been made on the file
Exceptions	No exceptions in this use-case.

Open Use-Case Description

Use-Case	**Open**
Actor	Admin, Normal User
Description	This use case occurs when user wants to open and existing file.
Stimulus	Using an editor of choice, the user goes to the file directory and opens the file.
Response	The file is loaded onto the editor and is ready for editing.
Exceptions	No exceptions in this use-case.

Modify Use-Case Description

Use-Case	**Modify**
Actor	Admin, Normal User
Description	This use case occurs when user wants to make changes on an existing file.
Stimulus	On an opened file in the user's editor of choice, user starts to make changes to the code. Adding functions and variables or even taking out some sections of code.
Response	The file opens up on an editor with the revision number attached to it.
Exceptions	No exceptions in this use-case.

Compile Use-Case Description

Use-Case	**Compile**
Actor	Admin, Normal User
Description	This use case occurs when user wants to compile the finished program file.
Stimulus	To compile, using the appropriate makfile, the user does a makebuild in the file's directory.
Response	On "ok" message is displayed when the system compiles successfully, otherwise an error message with the list of errors and line numbers where the errors occur is displayed.
Exceptions	No exceptions in this use-case.

Execute Use-Case Description

Use-Case	**Execute**
Actor	Admin, Normal User
Description	This use case occurs when user wants to run the compiled source code file
Stimulus	In a terminal, the user loads the compiled in supported environment.
Response	The output of the source code will be displayed, if not, an error message showing the line number where the error occurred is displayed
Exceptions	No exceptions in this use-case.

Test Use-Case Description

Use-Case	**Test**
Actor	Admin, Normal User
Description	This use case occurs when the user wants to make sure that the output got from the execute command is really what is expected out of the program.
Stimulus	Run a test command
Response	A log with both matching and un-matching files is printed out.
Exceptions	No exceptions in this use-case

Backtrack Use-Case Description

Use-Case	**Backtrack**
Actor	Admin, Normal User
Description	This use case occurs when the user wants to undo changes made to an already existing file
Stimulus	User enters a command that backtracks changes made on a file to one that was done previously.
Response	An earlier version of a file is opened.
Exceptions	No exceptions in this use-case.

Kill Use-Case Description

Use-Case	**Kill**
Actor	Admin, Normal User
Description	This use case occurs when the user is done and wants to end the process.
Stimulus	Depending on the editor, a command (eg. control-) will be used to halt the program
Response	Any open windows and operations should be terminated. A kill all and a kill just one process will be incorporated.
Exceptions	No exceptions in this us-case.

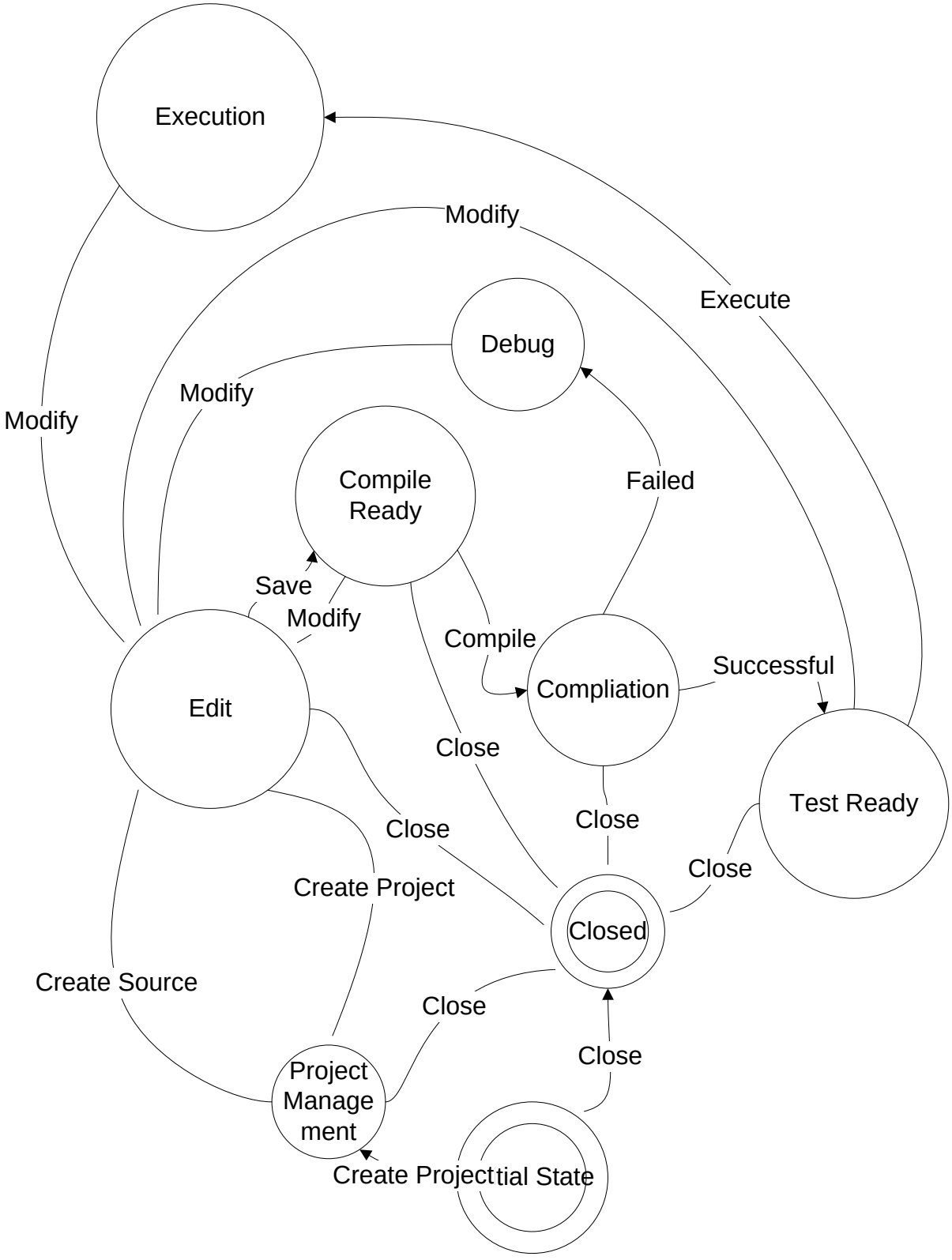
Customize Use-Case Description

Use-Case	**Customize**
Actor	Admin, Normal User
Description	This use case occurs when the user wants to customize the working environment.
Stimulus	Both the user and admin should be able to customize their working environment e.g. customize editor to automatically add headers to newly created files.
Response	A customized environment that's comfortable to work on by the user.
Exceptions	These customization options are optional but really important especially editor customization.

Build Tools Use-Case Description

Use-Case	**Build Tools**
Actor	Admin
Description	This use case occurs when an administrator creates tools to be used by other users.
Stimulus	The need for better tools to work with in the smart IDE system.
Response	The tools should be incorporated and applied by the regular user.
Exceptions	Only the admin can build tools to be used by other users.

3.10 State Diagram



4. Project Plan

4.1Project Management

4.1.1 Preliminary Work Breakdown

ID	Activity	Description	People	Resources
1	Write website documentation	Documentation for specific modules	ZC BB AO CC	HTML, PHP, WAK formatted files
2	Create language bindings	Design language bindings such that the website recognizes different languages	ZC BB AO CC	HTML, PHP, WAK formatted files, Javascript, AWK, Python
3	Create website to run KNIT modules	Design a website to take advantage of WAK extensions and markup fundamentals	ZC BB AO CC	HTML, PHP, WAK formatted files
4	Utilize community tools for IDE feedback	Make use of Google Code and other online community tools for auditioning the Smart IDE and Knit	ZC BB AO CC	Google Code, online communities
5	Utilize community tools for IDE feedback	Make use of Google Code and other online community tools for auditioning the Smart IDE and Knit	AO ZC	Google Code, online communities
6	Get a server	Find an online hosting solution to store code and downloads of the Smart IDE	AO CC	Computer servers, networking tools
6	Licensing	Determine legality and basic principles of the GNU license and how to utilize other works and/or redistribute our own works	ZC BB	GNU Public License, legal issues
7	Graphic Design	Create graphics for the Smart IDE software and/or website to better enhance the "look and feel" of our services	CC BB	Adobe Photoshop, Microsoft Paint, Adobe Flash, CSS

4.1.2Gantt Chart

Task ID	Task	Group Member				Week														
						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		Zach	Arthur	Ben	Casey															
1	TXT bindings				4						1	3								
2	Java bindings			3								3								
3	Python bindings	3										3								
4	Knit documentation		3									3								
5	Bug tracking / Resolution	2	2	2	2			1					1	1	1	1		1	1	1
6	Website creation (Sr Design)		4																	
7	Artofawk / TinyTim sites	2	2	2	2			1	1	2			2			1		1		
8	Interface design																			
9	Final documentation	11	10	13	12															
10	Weekly meetings (totalled)	13	13	13	13			4	4	4	4	4	4	4	4	4	4	4	4	4
11	Other documents	20	20	20	20			4	16			16		4	16		4		4	16
	TOTAL	51	54	53	53	0	0	10	21	6	5	32	7	9	21	6	8	6	9	21
	Grand Total	211																		

4.2Qualifications

4.2.1Arthur Otieno

Arthur Otieno is currently majoring in Computer Science at West Virginia University and has also received technical certificates in Graphic and Web Design. Programming languages he has familiarized himself with are Java, C, VB, PHP, ASP, HTML, XML, SQL, Smalltalk, AWK and Lisp. At the moment, Arthur works as a private web designer and software developer. His previous work experience and skill makes him a great asset to this project.

4.2.2 Zach Cowell

Zach Cowell is majoring in Computer Science at West Virginia University. He has had experience with Java, Javascript, C, C++, Python, AWK, Smalltalk, LISP, HTML, and XML. Zach workedin an internship in 2008 for a software engineering company and was exposed to many techniques and fundamentals of used in software development. Zach currently works at Ruby Memorial Hospital as a Cardiac Monitor Technician and monitors patient heart rhythms and maintains various hospital equipment.

‘4.2.3 Ben Butler

Ben Butler is majoring in Computer Science at West Virginia University. He has worked on programs using C, C++, PHP, HTML, Visual Basic, Java, Awk, Lisp, Python, Smalltalk, and Prolog.

5. References

Denis Sureau (2002). *Timeline of general-purpose programming languages.* Retrieved March 12, 2010, from:http://www.scriptol.com/programming/history.php

Netbeans Inc. Brief History of NetBeans. Retrieved March 12, 2010, from Netbeans official website: http://netbeans.org/about/history.html

Tim O'Reilly (2002, July 21). *Creativity, Flow, and Joy in Programming.* Retrieved February 17, 2010, from the Open Source Web Platform: http://www.oreillynet.com/pub/wlg/1784

Tim Menzies (2010). *Mastering the Art of AWK.* Retrieved February 23, 2010, from Mastering the Art of Awk: http://artofawk.net/

Tim Menzies (2010). *Software Engineering and Creativity.* Retrieved February 23, 2010, from Tim Menzies News Page: http://menzies.us/index.php?news=7143

Wikipedia contributors. *Integrated Development Environment*. Wikipedia, The Free Encyclopedia. Available at: http://en.wikipedia.org/wiki/Integrated_development_environment#History. Accessed February 22, 2010

Wikipedia contributors. *Active State Komodo*. Wikipedia, The Free Encyclopedia. Available at: http://en.wikipedia.org/wiki/ActiveState_Komodo. Accessed February 22, 2010

Wikipedia contributors. *Crowd Sourcing.* Wikipedia, The Free Encyclopedia. Available at: http://en.wikipedia.org/wiki/Crowd_sourcing. Accessed February 22, 2010

Sandra Henry-Stocker. (2006, July 26). *Library Dependencies*. Retrieved February 17, 2010, from IT World, An Open Exchange:http://www.itworld.com/nls_unix_lib060727

Ronald P. Loui. (2008, July). *In Praise of Scripting: Real Programming Pragmatism.* Retrieved February 22, 2010, from Awk.Info: http://awk.info/?doc/praise.html

Brooks, Fred P. (1986). *No Silver Bullet - Essence and Accident in Software Engineering.* Retrieved February 17, 2010, from Proceedings of the IFIP Tenth World Computing Conference: 1069-1076: http://en.wikipedia.org/wiki/No_Silver_Bullet