Ok Eclipse: Enabling Voice Input to augment User Experience in Integrated Development Environment

[March Report 1b - Team 'O']

Shrikanth N C
NC State University
CS department, NC 27606
snaraya7@ncsu.edu

Karthik M S
NC State University
ECE department, NC 27606
kmedidi@ncsu.edu

Kashyap S
NC State University
ECE department, NC 27606
ksivasu@ncsu.edu

Charan Ram V C
NC State University
ECE department, NC 27606
cvellai@ncsu.edu

ABSTRACT

The goal of the project is to improve user experience in the Integrated development environment(IDE) by incorporating a speech recognition system. The paper starts by outlining the problem statement that we aim to solve, and delves into the details of the methodologies adopted to develop the system. Then, the implementation details of each use case discussed in the problem statement are elucidated concisely. Furthermore, we present a complete evaluation of the system, revealing the actual usefulness of the voice-based system to an average IDE user. We finally delineate the challenges faced during the development process and possible future scope of this project. We clarified all the deviations from original proposal and discussed with reasoning in the respective sections.

Keywords

Speech to text, Integrated Development Environment, Q&A repositories, plugins

1. PROBLEM

In the era of abundant voice libraries, IDEs are not reaping its benefits. Users typically rely on keyboard and mouse and most recently touch displays as methods of input. We see potential in the voice as an additional input method to augment user experience and thereby increase productivity.Results show that strategies aligning with following categories could achieve corresponding percentage savings in productivity [3]:

- Working Faster – 8%,
- Working Smarter – 17%
- Work Avoidance – 47%

Our problem statement targets the first two categories by trying to achieve the goal of improving the user experience. Making the entire process of development (but not limited to) easier and faster would help reduce the amount of time required for development and ultimately have a positive impact on productivity. It's worth noting that we don't completely delve into programming by voice. An evaluation with expert Java developers showed that programming completely by voice is slower than typing [1]. Therefore, we use voice as an input for expediting simple tasks on the IDE and for achieving two other critical use cases (discussed in Section 2) which would act as useful tools for a programmer. This could be perceived as an effort to bridge the gap of an inability to multitask given the voice avenues.

2. REQUIREMENTS & SOLUTION : DEVIATIONS INCORPORATED

We approach to tackle the problem with the following three use-cases (Refer Figure 1):

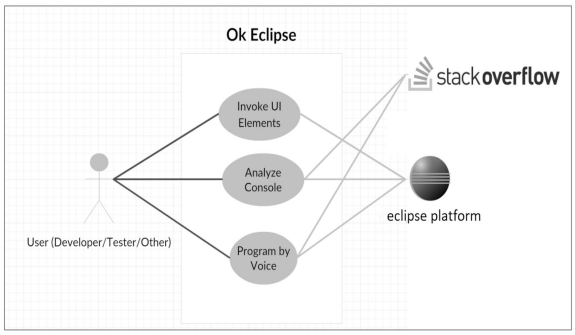

Figure 1: Ok Eclipse - Overall Use Case Diagram

2.1 Listening Menus

Eclipse [9] IDE as we know is composed of editors, views, and menus (driven by commands). All these components are built as plugins. Hence, we intend to approach this problem statement by introducing plugins to Eclipse IDE that take voice as input. These plugins simultaneously enhance both user experience and efficiency of a developer.

Navigating views with a mouse is cumbersome especially in scenarios where certain menus require traversing multiple levels down the menu. One might argue for the use of short-cut keys. However, to master the huge list of short-cut keys for all the options on the menus (IDE specific) is an arduous task. We believe, this voice-based feature will aid newbie

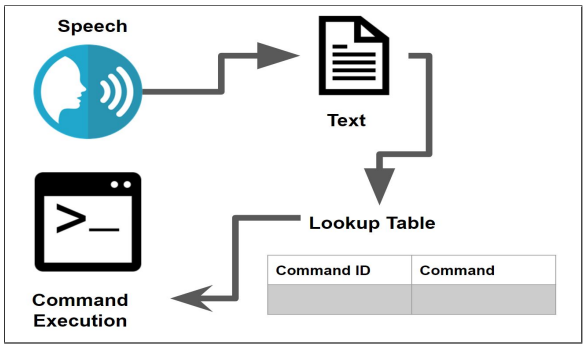

Figure 2: Overview

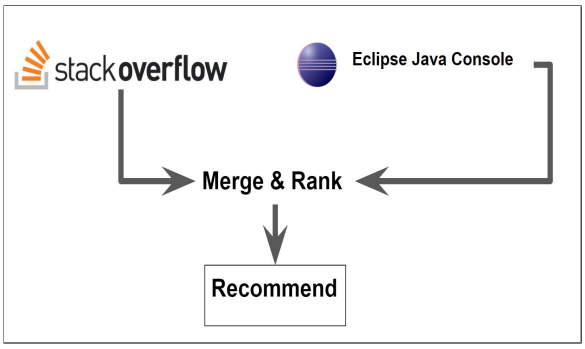

Figure 3: Overview

users to utilize the features of IDE in-par with seasoned users.

We identify every menu item in eclipse associated with a command. Commands have unique ID's with the IDE. Hence, we propose to create a look-up table (Map) that identifies the command ID given a menu label. The menu label is the visual text as seen in the IDE. We expect the user to recite the label text and our engine will invoke the appropriate command using the look-up table.

Creating this look-up table manually for all the commands in eclipse IDE requires a mammoth effort. However, we have discovered an easier way to automate this task. As mentioned earlier eclipse IDE is made up of plugins. Each plugin has a plugin.xml file that contains all the meta information about that plugin. This also includes all command extension and ids. This file along with the plugin is placed under eclipse/plugins folder. Hence, by mining the plugin.xml file we plan to extract (using regex or XML dom) all relevant ids and command labels to build the look-up table automatically.

Upon receiving the voice input, we match its text equivalent with the look-up table to retrieve the corresponding command ID. Then, we pass a request to the eclipse command framework to invoke the corresponding command given the ID as shown in Figure 2.

We are presently using *'shift + Z'* for 'Ok Eclipse' to hear the speech request for a short time. This can be changed in the plugin.xml part of the code base. Ok eclipse generates the dictionary and look up csv files in the ok_eclipse folder. This folder is usually generated inside the installed eclipse folder.

2.2 Loud Console

Console portrays the exact behavior of code at a given time - Errors, exceptions, and user-defined messages (commonly tagged as INFO, DEBUG, and ERROR in log files). We propose the recommended fixes by extracting the relevant text of interest from console log to form a query and mine Q&A (stackoverflow.com [12]) repositories for appropriate solutions.

Whenever the user feels like he might need a recommendation, the user can invoke the "Find" command to get a suitable recommendation for the relevant console log displayed at a given time.

We choose crowd-sourced Q&A repositories as they are more likely to have multiple user responses to a question. Our basic criterion in selecting the answer is, first, we choose the accepted answer and in cases where the accepted answer is unsatisfactory, we offer the answer in the order of decreasing number of up-votes. In other words, internally we plan to build a small answer ranking algorithm from the available questions and present the best ranked one to the user. The user will have more than one recommendation (answer) to view. We also present relevant youtube.com videos for the relevant error/exceptions available in the console.

2.3 Sound Programmer

In this space, we wish to identify the frequently used and mundane tasks faced by programmers in IDE and we chose to activate the same using voice. Few examples such as generating toString, getter, and setter methods, main method, sorting, running, testing, code formatting, debugging and certain objected-oriented features.

In essence, we define this feature as an elegant way for a programmer to multitask on the IDE, achieving multiple tasks by voice inputs and as a result improving his efficiency is being proposed.

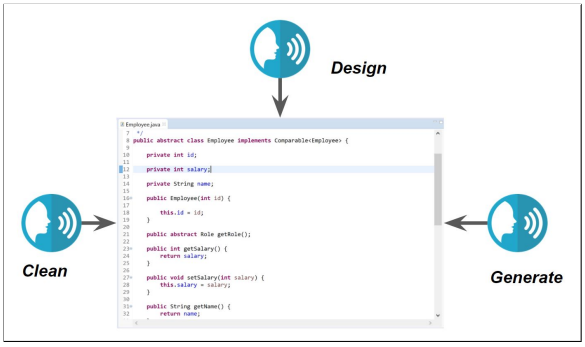

Figure 4: Overview

3. ARCHITECTURE & DESIGN : DEVIATIONS INCORPORATED

An architecture with three separate plugins for three different use cases (section 2) was initially conceptualized to allow us to do things in parallel. However, we bundled it in the same plugin. This is because we figured it was unnecessary, since there was no major need to decouple the three features. Also installing a single plugin looked far more simple.

Eclipse, as we know, is made up of plugins. These plugins are loosely coupled to the Eclipse platform. Eclipse nicely exposes extensions and extension points for developers to consume and build plugins to host custom features within eclipse IDE.

Type	Library
Speech to Text	CMU Sphinx [4]
User Interface	SWT and JFace
Development environment	Eclipse PDE
Platform	Eclipse Rich client platform

Table 1: Required Libraries

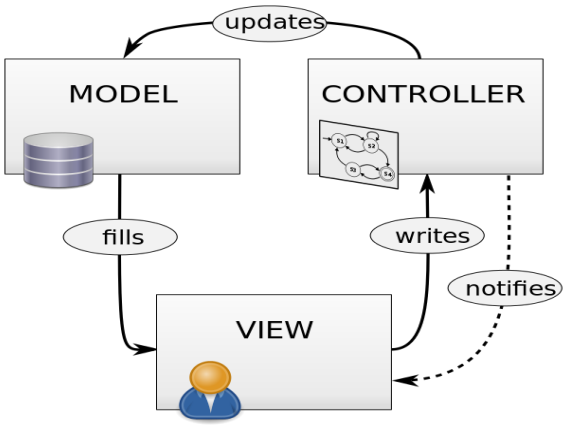

Figure 5: MVC Architecture [16]

Incorporating our plugin into eclipse is as simple as dropping the plugin jar file into the eclipse plugins folder.

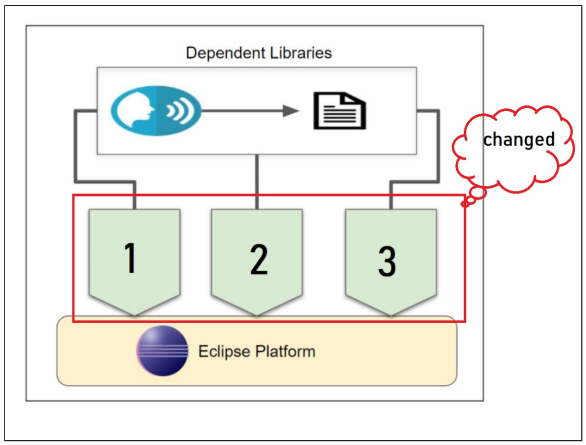

Figure 6: Proposed Architecture

However, the commonality between our three features is the CMU sphinx library (speech recognition) which we use as a speech to text converter shown symbolically in Figure 6. We propose to write a wrapper to this software such that each plugin can individually consume it to cater to the specific use case.

The eclipse plugin development environment implicitly enforces the users to build by Model-View-Controller (MVC) architecture which is shown in Figure 5. Further, the bulky code written within each of these plugins still need to be robust, extensible and maintainable. Hence, we chose to adhere to design by interface and apply object-oriented design such as SOLID principle in developing each of these plugins.

4. SOFTWARE DEVELOPMENT

A software development process is aimed at enabling swift development of software with a minimization of errors through substantial planning and management. It was crucial to adopt a software development process which accurately described the aspects of the project in hand considering the requirements specified and the resources available. Bearing these factors in mind, a Prototyping model of development was chosen given its simplicity in our case. Further, this model of development provided a means to deliver results early on in the time line of the project.

4.1 Prototyping Model

As with any software development practice, the requirements for our project were clearly defined and understood at the beginning of the project timeline. Three unique use cases were discussed and formulated under the motivation of improving the user experience of the Eclipse IDE. Several tasks and subtasks were then extrapolated from these use cases such that the most difficult tasks (that is, task with the most uncertainty/ambiguity in its implementation) were given priority. This allowed us to work without anxiety later on as we initially focused on and solved the tasks that could either make or break the project. The feasibility and scope of using an open source speech recognition software and deciding the actual tool that is to be employed was one such task. Adhering to the Prototyping model, development started early on and the initial focus was on integrating the Sphinx speech recognition software with the Eclipse IDE. We aimed to provide support for enabling voice based invocation of certain frequently used inbuilt Eclipse commands as a start. This however also came with its own challenges since we had to delve into a time intensive mining process to acquire the unique IDs associated with each of Eclipse's inbuilt commands which were not directly evident. With the command IDs and the Sphinx speech-to-text vocabulary in hand, we had to link the text output of the Sphinx vocabulary to the correct Eclipse command ID and write appropriate handlers for the same. This enabled us to solicit an Eclipse command through voice, thus wrapping up the core functionality for one of the use cases - Listening Menus.

The remaining two use cases involved a significant amount of coding as it brought in functionalities external to Eclipse and was different from the previous use case in that they did not make use of inbuilt commands exclusively. The second use case namely, Loud Console, involved aiding the user in debugging any exception that he/she might encounter while writing a piece of code. We made use of online QA forums such as Stack Overflow to provide meaningful suggestions to the users thereby making it easier for them to understand where they went wrong. In order to facilitate this functionality, we used Stack Overflow APIs to fetch information relevant to the exception in question. Furthermore, we made sure to fetch and display only the accepted answers or the most upvoted ones (in the absence of any ac-

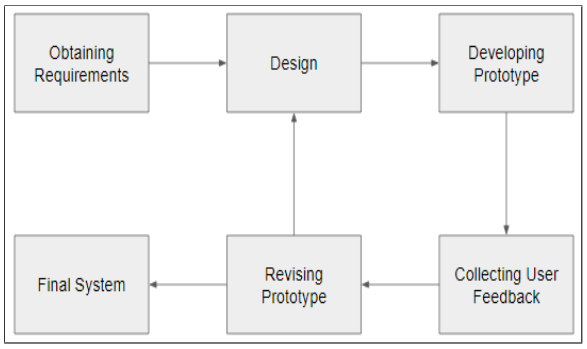

Figure 7: Prototyping Model

cepted answer) to the user. The underlying nature of the Prototyping model of development that revolved around an iterative approach to developing software not only enabled us to generate the final product within the targeted date of completion but also enabled us to make use of the final product as a means to help us envision and revise the design requirements. This allowed us to make minor enhancements to the requirement wherein we provided the user with an array of Youtube videos pertaining to the relevant exception by making use of the web browser plugin in Eclipse. Thus we sought to propel the productivity of the programmer by creating a conducive environment wherein the user had all the resources at his disposal.

Lastly, the Sound Programmer use case was all about making coding itself to be a fairly simple and effortless process. We planned to inject pieces of code directly onto the editor pane of Eclipse in response to the user's request. This functionality works by searching the Stack Overflow repository and fetching the appropriate code. In order to avoid entering Stack Overflow credentials periodically while using Stack Overflow APIs, we used Google's search engine instead and crawled Stack Overflow HTML pages to obtain the necessary code logic. This workaround enabled us to implement this feature efficiently which if otherwise would have been impractical.

We used the following tools during the process of development as shown in the table 2.

Type	Software
Build & Deployment	Eclipse PDE [5]
Testing	JUnit [13]
Issue tracking	Github [10]
Milestones life-cycle	Microsoft Excel/ Kanban[10]
SCM	Github[10]
Design	UML[8]
Wireframe	Pencil[6]

Table 2: Process Tools

Overall, the Prototyping model helped in detecting certain limitations in the system and even aided in discovering new requirements earlier on in the process while also leading to better solutions due to quicker feedback.

4.2 Test Driven Development Methodology : Deviations Incorporated

The Test Driven Development (TDD) methodology was the primary framework that formed the basis for the entire coding process. It allowed the developer to create a feedback system while maintaining a strict and self-imposed conformity to the design requirements.

Our initial objective was to obtain a comprehensive understanding of the underlying requirements for each of the use cases provided in section 2 and then build test cases for each of them. However instead of the developer writing a test case for his own functionality we made a slight alteration to TDD such that the developer of a functionality was not the owner of its test case. This made it mandatory for all of the team members to understand the functionality of their peers while the peers tried to break the developer's code through comprehensively written test cases.

Central to the TDD approach is the concept of the Red-Green-Refactor methodology. Adhering to this methodology, in the 'Red' portion of this model we created a suite of test cases that provided an overall representation of the abstract design requirements and wrote code that was designed to fail the test suite. In the 'Green' segment we started writing code that can satisfy these test cases to a basic degree without exceeding the functional specifications. This created a tangible framework that made the developers more accountable and prevented us from deviating from the given specifications. As the software underwent subsequent iterations of coding as well as functional updates, a process of refactoring on the entire scale of the software was needed so as to provide maintainability and sustainability of the code. The code was cleaned up through this refactor phase which involved removal of code and/or functional duplication and rearrangement of certain code segments among other things. Each of the functionalities viz. the Listening Menus, Loud console and the Sound Programmer were tested thorouhgly. Issues were logged in Github to help keep track of the bugs and were closed as soon as the bugs got fixed.

The Test Driven Development methodology proved to be advantageous in that it helped catch a few defects early on in the project timeline due to the frequent testing done throughout the development cycle. This served to eliminate the need for any tedious debugging at the end.

4.3 Effort Estimation

4.4 Expected

We used Use Case Points [2] to estimate our effort and we display the final results below Figure 8. We understand that this is not an ideal estimate. However, this activity helped us to dissect and view the factors that could affect delivery.

4.5 Actual

The actual effort required for the project was a total of *180* hours. That is, each team member has worked for 45 hours in this project. This involved the four developers to dedicate a little less than two hours per day ,on average, during the development phase of the project for a time span of one month. The actual effort requirement closely matches the result of the effort estimation model that was put forth in the initial design document.

5. INFRASTRUCTURE & TOOLS

Calculations From Other Tabs		
TCF	Technical Complexity Factor	0.635
EF	Environmental Factor	0.455
UUCP	Unadjusted Use Case Points	20
AW	Actor Weighting	1
Calculation of Use Case Points		
UCP	Use Case Points	6.1
Calculation of Estimated Effort		
Ratio	Hours of Effort per Use Case Point	28
	Hours of Effort	170

Figure 8: Effort Estimation - UCP

Our application has been built in an environment shown in the Software & Hardware requirement Tables 4 & 3.

Component	Configuration
Processor	2.5 GHz
RAM	4GB
Storage	1TB
System Type	64 bit OS
Network	50 Mbps Download ; 10 Mbps Upload

Table 3: Hardware requirements

Type	Software
Operating System	Windows 10
Virtual Machine	Java 8
IDE	Eclipse [9]
Build & Deployment	Eclipse PDE [5]
Testing	JUnit[13]

Table 4: Software requirements

Constrained by the low budget of our project, we were forced to zealously search for other open-source tools for the speech to text implementation. In our exploration, we were able to identify the CMU Open-source Sphinx library tool as the best performing within our constraints. However, the speech to text software presented its own limitations when integrated to our project's specifications. To address this concern within the constraints of time, we have deduced that using a dedicated microphone in a reasonably noise-free environment improves the operation significantly. Thus, these have been added to the requirements set out in the initial document.

6. EVALUATION PLAN

Our project is aimed at developing a piece of software that provides programmers an alternative way of executing Eclipse IDE options through voice-based input. It is aimed at reducing the hassle of the programmer by providing options such as executing options through categorized menu selection. It was set out in the initial report that since our project aims at improving user experience, quantifying the possible benefit, if any, in an objective manner is difficult. Thus, a two-pronged approach to the evaluation strategy based on quantitative and qualitative assessments was adopted for this project. The appraisal process for our project has two stages, namely, a formative and summative stage [17].

We faced a quandary over finalizing evaluation methodologies when the evaluation plan was presented in the initial report. However, as the features covered by the project were realized, our clarity over the evaluation mechanisms also heightened. In the following sections, we describe the processes that were undertaken in the final evaluation plan.

6.1 Formative Stage

The main intention of a formative stage evaluation plan is to create a well-defined framework of specific, identifiable and reasonable goals that shape the final deliverable.

Our project's framework is based on three key aspects, maintainability, sustainability and usability. With the aim of developing software that is maintainable, we incorporated the requirement of providing necessary documentation support, using understandable identifiers and following coding best practices. On the other hand, the objective of creating sustainable code has implications of allowing extensibility and providing a means for future improvements throughout the process of development. In this regard, we have provided a self-explanatory TDD-based approach and contacts of the software-creators for future development on this project.

For the third aspect of assessing usability, we had initially conceptualized a quantitative evaluation that involved a direct measurement of the change in delay for executing certain tasks with and without our software. However, since this study would not capture the benefits offered by all the features of the software, this was later modified to conduct a rigorous internal evaluation, the results of which have been surmised in Table 5).

Functionality	Command Recognition
1	
2	
...	
n	

Table 5: Internal Evaluation

6.2 Summative Stage

Upon completing the unit testing and integration testing on our project, we then moved to the Acceptance testing. At this stage, a system test to assess the accomplishment of goals set out in the initial design document was conducted. In addition to the self-evaluation conducted during the formative stages of development, which tried to quantify the usability of the software, a test appraising the software based on usability to the end-user using a survey to qualitatively specify the same was organized.

The industry standard **System Usability Scale** was adopted to measure User Experience with peers. In correspondence with experts, it was realized that the aspects of the generic System Usability Scale were inadequate in capturing the

user-experience for our system. In consequent iterations, the queries were fine-tuned to accurately express the benefits of our system. The results of this evaluation have been included in table (xxx) and the inferences drawn from these results have been summarized.

#	Plugin to evaluate	System Usability Scale
1.	Listening Menus	
2.	Loud Console	
3.	Sound Programmer	

Table 6: External Evaluation

7. PROJECT MANAGEMENT

We are a four member team. To avoid ambiguity and maintain clarity we divided large tasks into small chunks with short deadlines. We maintained a google sheet for task management with appropriate deadlines and owners. We created a Google drive folder dedicated to this project and all the assets pertaining to this project went into that folder.

As suggested by the instructors we maintained four branches in git for safe and parallel development.

Component	Owner(s)
Listening Menus	Shrikanth N C
Loud Console	Kashyap S, Karthik M S
Sound Programmer	Karthik M S, Charan Ram V C
Speech recognition Layer	Shrikanth N C
Eclipse UI	Charan Ram V C
Eclipse core support	Shrikanth N C
Acceptance Testing	Kashyap S
Evaluation execution	Karthik M S

Table 7: Divide & Conquer

In situations when members couldn't collaborate in person to address certain issues we used Team viewer software to share our screens to rectify the same. Git merge can be arduous sometimes, especially for newbies hence we used beyond compare tool for all local merges. Our overall goal and actual show stopper issues are recorded in git kanban and issues page.

Asset/Tool	About
Task Management spreadsheet	Manage tasks
Team viewer	Enable remote working
Beyond compare	Local code comparison
Git kanban and issues	milestone management
Google drive	Collaboration

Table 8: Internal Evaluation

7.1 Deployment

Ok Eclipse is basically a component that plugs itself to the eclipse architecture. The code base is an eclipse plugin project with java and maven nature. After you make any changes you can export the plugin (Refer Figure 10) using the plugin.xml file available in the 'edu.ncstate.csc510.okeclipse' project.

7.2 Testing

After you make any changes you can test the plugin project (Refer Figure 11) using the plugin.xml file available in the 'edu.ncstate.csc510.okeclipse' project.

This opens up a temporary eclipse instance with the loaded plugin.

7.3 Video Tutorial

We hosted a video[15] in youtube.com that exhibits most of our project's functionalities in a short yet comprehensive video.

8. FEATURE IMPLEMENTATION

This section outlines the implementation of each use case of the Ok Eclipse software.
It explains how each functionality is achieved, and details on classes that are responsible for achieving each of them. We further delve into the working of key methods in each of these classes.

8.1 Speech recognition Layer

The Speech recognition Layer is the pivot of the entire system. We use CMU Sphinx Open Source Library, which provides the dictionary for speech to text translation. A major chunk of Voice Recognition functionality is controlled by the VoiceRecognizer Class.The VoiceRecognizer Class is called whenever the system is prompted with a Voice Command. The class utilizes Sphinx Library to get the equivalent text for the Voice Command and return it to the appropriate function for further processing.

8.2 Listening Menus Implementation

The Listening Menus functionality is to control the options in Menus using voice inputs. The class that dictates this functionality is the SpeechHandler class. Execute, processSpokenText and executeCommand are the three methods inside this class that act as the cogwheels for this functionality. Firstly, the Execute method is invoked when the corresponding voice command for this functionality is given. The method uses the open source Voice Recognizer software (Sphinx Library) used to convert the speech command into the corresponding text. Then the clean method is used to filter any unwanted noise or silence in the converted text.
The processSpokenText method is then called which in turn invokes the executeCommand method, to find the corresponding Command ID for the String (spokenText) passed to it. The corresponding commandID is subsequently executed to perform the required action.
So, the core of this functionality is based on converting the Voice Command to text, finding the matching Command ID for the text input and finally executing the option associated with the Command ID.

8.3 Loud Console Implementation

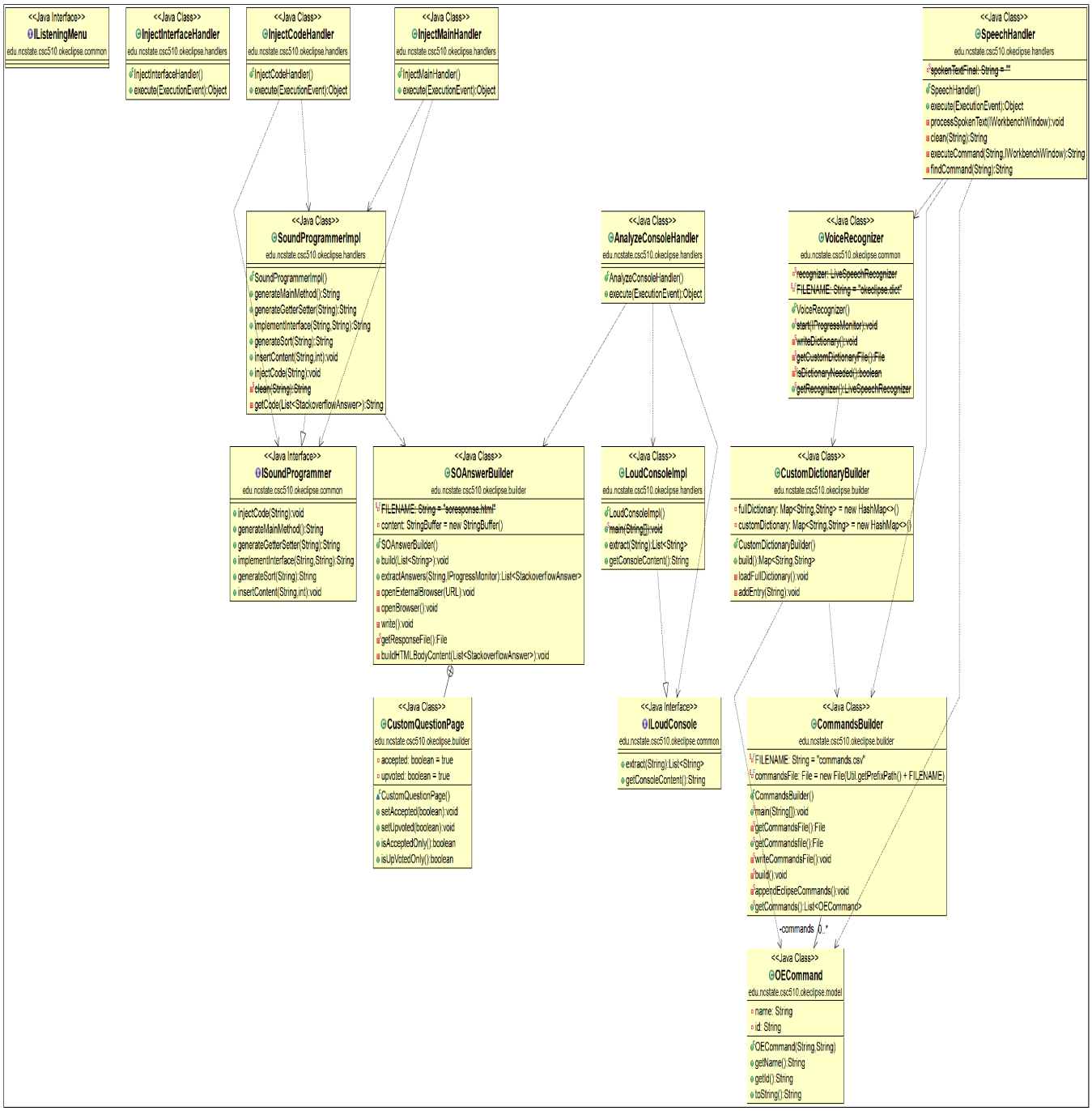

Figure 9: Ok Eclipse : Class diagram

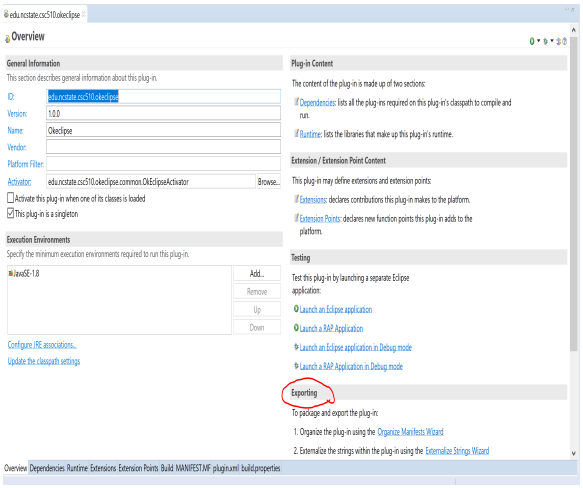

Figure 10: Plugin Export

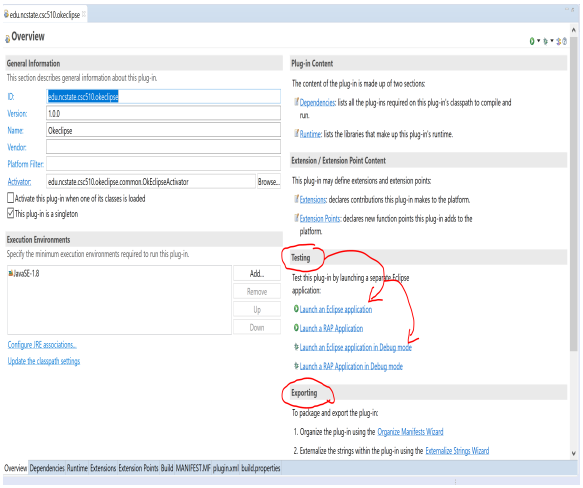

Figure 11: Plugin Testing

Consider the Class Diagram shown above (figure 9) for reference. The Loud Console Implementation has three main components: AnalyzeConsoleHandler, LoudConsoleImpl and ILoudConsole. There are two crucial methods in the LoudConsoleImpl which perform the crux of the entire functionality: extract() and getConsoleContent(). The getConsoleContent() function gets the content from the console whereas the extract() method extracts the errors and exceptions from the entire content and returns it one by one to the SOAnswerBuilder java class. The SOAnswerBuilder java class gets the answers for each of the passed Strings (Exceptions and Errors) and returns an HTML page with all the Answers.

8.4 Sound Programmer Implementation

The SoundProgrammer Implementation has the following classes intertwined to complete the functionality: InjectCodeHandler, InjectMainHandler, SoundProgrammerImpl (Refer to the Class Diagram above).

The SoundProgrammerImpl has the following methods which dish out some of the principle features of this functionality:

- generateMainMethod()
- generateGetterSetter(String)
- generateSort(String)
- InjectCode(String, Int)

The generateMainMethod() and generateGetterSetter(String) function as the name suggests generates main method, getters and setters for all the members of a class when the corresponding commands are issued.

Similarly, the generateSort(String) sorts the Input String when a corresponding input voice command is given.

The functionality of the InjectCode(String, Int) is to import code from StackOverflow for the corresponding query enclosed within "$ $" sign in the editor. The InjectCode(String, Int) pairs up with SOAnswerBuilder Class to get the Code from StackOverflow and inject it on to the editor just below the query.

9. SOFTWARE INSTALLATION AND USAGE

In the following sections we present to you two ways in which you can install the Ok Eclipse software.

9.1 Plugin install

If you are already using a latest version of eclipse to which you wish to add 'Ok Eclipse' support you can chose this option. All you have to do is download the plugin jar (see README.md for more details) and place a copy of the plugin jar in the dropins folder of your installed eclipse. Start eclipse and you should see the following screen (Figure 14 15).

Figure 12: Ok Eclipse Plugin

9.2 Bundle install

If you don't have a latest eclipse or wish to use a dedicated setup for 'Ok Eclipse' you should go ahead with this option. All you have to do is download the eclipse zip (see README.md for more details) and extract them to any destination and run eclipse.exe. 13

Figure 13: Ok Eclipse Plugin bundled with Eclipse

Post installation, on eclipse start/restart you will notice the following progress. This progress image (Figure 14) is shown during every start up. During this time the constrained dictionary for the commands listed in the csv file is being loaded in the speech recognition layer. Upon the dic-

tionary being loaded successfully you will see the following message (Figure 15).

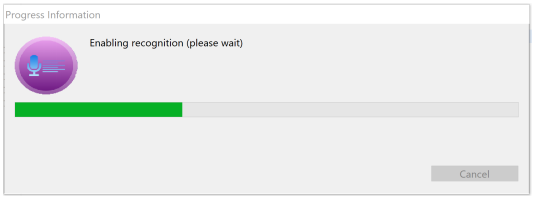

Figure 14: Loading dictionary

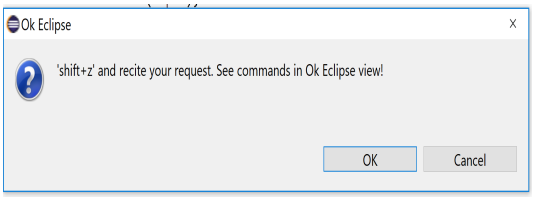

Figure 15: Success

You can always press the short cut key to utilize 'Ok Eclipse' features. In the eclipse workbench you will notice a dedicated view called 'Ok Eclipse view' (Figure 16) that lists all the eclipse commands that can be assigned a voice command. By default some of them have been already assigned.

You can edit these commands and provide custom dictionary keywords by pressing the button indicated by the red arrow.

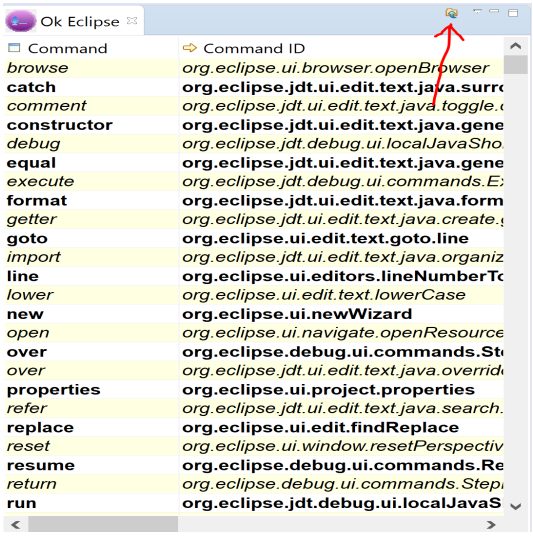

Figure 16: Ok Eclipse View - Lists all commands

9.3 Reuse

9.3.1 Technical assets

We adhered to *program to interface* principle which enables loose coupling. And eclipse's plugin development environment by itself strongly enforces this. For instance, our speech recognition (a wrapped to sphinx) can easily be used in similar projects to enable speech to text setup.

9.3.2 Functional assets

We recognized most of us were new to git and git initial setup and basic push, pull, merge and branch needed some orchestration of commands. We prepared a short tutorial and we refined it throughout our project.

10. EVALUATION RESULTS

The following sections provide the results from the evaluation conducted at the internal and external fronts. The internal assessment involved collecting the anonymous metadata regarding the success rate for the voice command recognition. On the other hand, the external evaluation involved recording the responses from end-users in a controlled environment survey.

10.1 Internal Evaluation - Log Analysis

The task of voice command recognition and translation was identified as the crucial link which determined the overall performance of the software and hence became the basis of our project's internal evaluation. A dedicated log file has been used to record the success and failure of the speech to text conversion software. The statistics in operational success based on this log file has been illustrated in the following pie chart (Refer Figure 17).

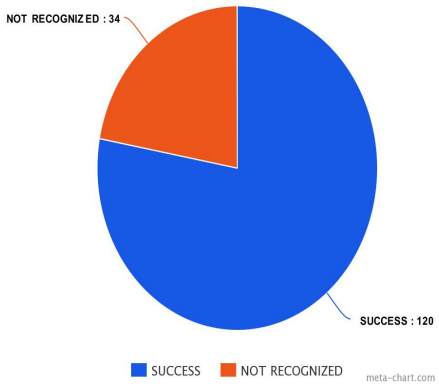

Figure 17: Voice Command Recognition Success Rate

10.2 External Evaluation - Real users

The System Usability Scale was used as the primary evaluation tool to record the end-user experience in using the software. A controlled environment survey methodology was used in collecting this information. This involved in actively requesting the participants perform evaluation of the software in a noise-controlled environment along with the usage of a dedicated microphone. Additionally, a software expert was at hand to assist the user in case of any confusion.

In the category of rating the ease of usage of the software, a majority of the participants voted that the software was easy or very easy to use (Refer Figure 18). Further, 92.3% of the participants felt that if the software was available to them, they would use it frequently (Refer Figure 19). Additionally, a little over 90% of the participant base felt that the software actively improves user experience (Refer

Figure 20). These survey results show that the software has achieved the goals that have been presented in the design document.

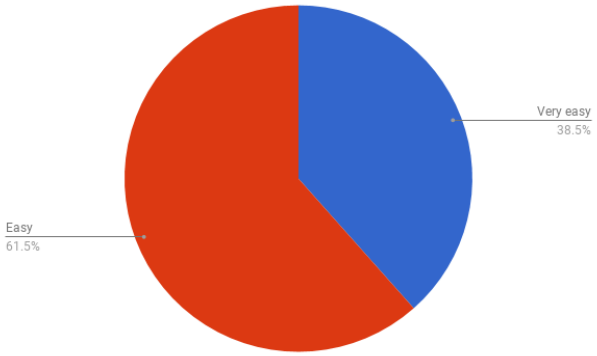

Figure 18: User Experience - Ease of Usage

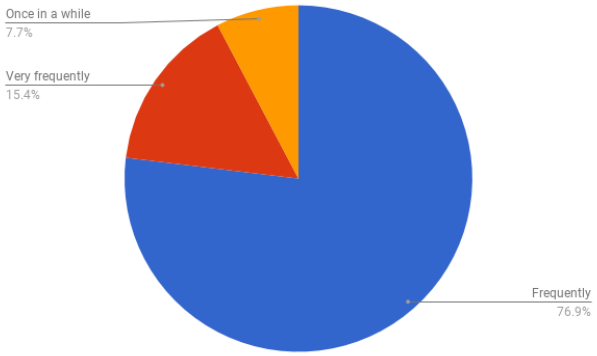

Figure 19: User Experience - Software Usefulness

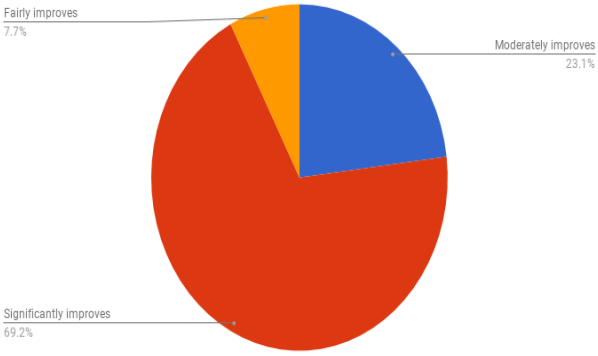

Figure 20: User Experience - Improvement

Secondly, the software's coding and architecture reflects in the eventual user experience as they have a direct performance impact. In order to estimate the same, the following survey results came in handy. Users rated that the features offered by the software were not disparate but actually work with each other providing a consistently unified approach to improving user experience (Refer Figure 21).

As mentioned previously, a software expert was at hand to assist the end-user in case of any confusion. Even so, most users did not feel the need for technical assistance in corroboration of the fact that the software's usage is self-explanatory (Refer Figure 22). This corresponds to a phenomenal 84.3% of the participants.

In the process of development of the software, we had identified specific features that were annoying to the programmer and had focused our efforts in improving these major trouble points. In evidence to the fact that these points have been successfully identified , all participants chose one of the offered features and did not exercise the 'other' option at all (Refer Figure 23).

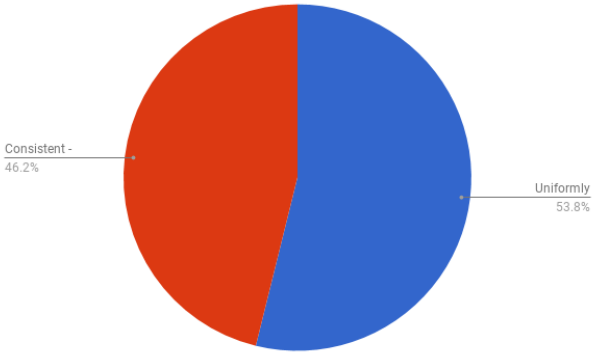

Figure 21: Feel of Software - Feature Consistency

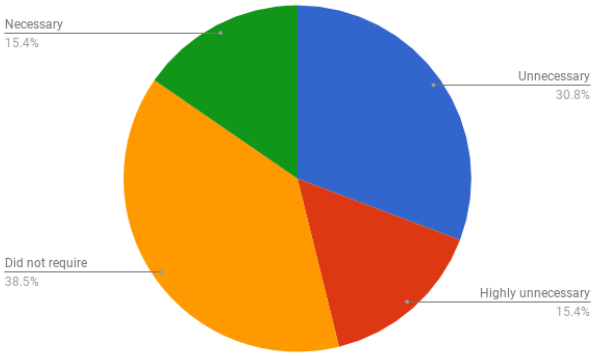

Figure 22: Feel of Software - Technical Assistance

In order to sufficiently bolster the survey results, it is absolutely necessary to provide detailed information about the participants. Thirteen participants took part in the Ok Eclipse Software Evaluation Survey and all of the participants had experience working with the Eclipse IDE platform upon which our software was based (Refer Figure 24). Since the users were all technically well-versed and have specific experience in using the base platform, their feedback regarding the software, is so much more valuable.

10.3 In a nutshell

Broadly, the results indicate that *Ok Eclipse* did augment User Experience in Integrated Development Environment. On the basis of personal choice, interestingly, some reviewers

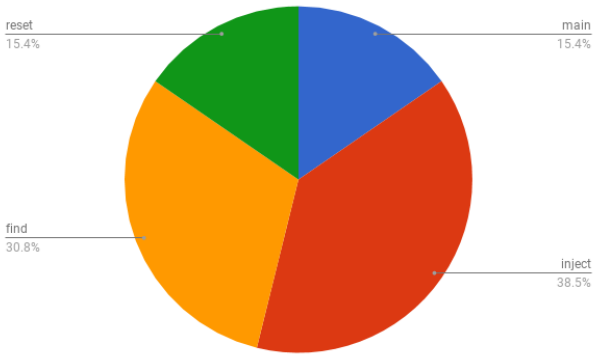

Figure 23: Feel of Software - Useful commands

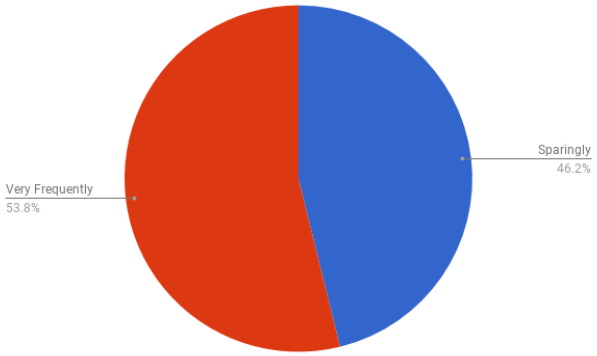

Figure 24: Participant Experience in Eclipse IDE

did debate on having some of the features keyboard based. Nevertheless, they appreciated the feature.

10.4 Threats to validity

Its important to note that projects similar to ours heavily depend on the underlying speech to text library/platform, as underlined in the challenges section. We wish to highlight here that other proprietary/open source speech to text libraries are available which the readers may be interested to experiment this project with.

10.5 Chits

The chits collected from the NC State users who had provided feedback for our project have been listed below. Although, the online evaluations yielded similar results. External Evaluation section was completely based on in-person evaluation. We are using online evaluation only to observe any anomalies and gather user feedback. The chits listed from 14 to 21 are online evaluations.

No.	Chit
1.	GYR
2.	KPY
3.	MMK
4.	FAQ
5.	QKR
6.	GAR
7.	JOT
8.	LPR
9.	TDW
10.	LVB
11.	FOJ
12.	MMF
13.	UWK
14.	RUU
15.	JCF
16.	TOE
17.	QDR
18.	ZMK
19.	NQX
20.	TPQ
21.	WOT

Table 9: Chits

11. CHALLENGES

11.1 Constrained dictionary

We have made use of the Sphinx system as our speech recognition software for this project. CMU Sphinx is a set of speech recognition development libraries and tools that can be linked in to speech-enable applications. The large size of the Sphinx dictionary along with the open source nature of these libraries imposed certain limitations on the performance of the product as a whole. This caused us quite a few problems initially as several voice commands got misrecognized which resulted in the execution of a different action other than what the user had intended.
After a significant amount of testing and experimenting with the Sphinx dictionary we came up with a way to circumvent this problem. We had concluded that the problem was due to the dictionary hosting a huge number of words that did not hold any relevance within the scope of our project and realized that these irrelevant words oftentimes caused the misrecognition of certain voice commands. Hence, we decided to create our own dictionary restricting its contents to only those words that fell within the scope of eclipse. Upon further testing, this “constrained dictionary” was found to perform more accurately while also reducing the processing time involved.

11.2 Disparate keywords

In order improve performance time by overcoming the limitations of the Sphinx speech to text conversion software in terms of identification of separate words in the voice command, we have limited the number of keywords in the voice command to one. Moreover, through our experimentation, it has been concluded that the identification of the voice command works best when the voice commands are not homophones. Thus, given the constraints of time, phonetically different sounding words are recommended for usage as keywords.

12. FUTURE SCOPE

Assuming we have some more time with same amount of resources.

We may wish to attempt the following :

12.1 Verbose programming

We would look to solve the challenges elucidated above. Solving the above challenges would enable a flexible speech recognition layer that can tackle structured sentences rather than just a word. Ambitious use-cases such as refactoring, UML design and much more is possible with such power. Imagine a system that can tackle most of the Software development life cycle phases with speech, with emphasis on Software development.

12.2 Other IDE's & Beyond

Presently, this support is limited to eclipse IDE. We may extend this to other ide's such as Netbeans [14], IntelliJ IDEA[11] and much more. Further, in the era of cloud based IDE's like eclipse che [7] one may have to enable such support to browsers as well. Interestingly, this may give rise to a fruitful by-product to navigate web pages using speech.

13. CONCLUSION

Thus, we have presented the entire scope of the project along with the details of software development methodologies that were adopted. We have cataloged the implementation details of each use case, which would act as a guide for external developers during future improvements. Then, a formal evaluation of the system with internal and external feedback reveal that the system indeed augments user experience in Eclipse IDE and creates a positive impact on the overall efficiency of the programmer. Finally, we outline some of the key challenges that were faced during development and some possible extensions that could be added on to the existing implementation as future scope. [1]

References

[1] S.L. Graham A.Begel. *An assessment of a Speech-BAsed Programming Environment*. Visual Languages and Human-Centric Computing, 2006, VL/HCC, IEEE Symposium.

[2] S.L. Graham A.Begel. *Estimating With Use Case Points Use Case Points*. http://www.cs.cmu.edu/~jhm/DMS/\%202011/Presentations/Cohn\%20-\%20Estimating\%20with\%20Use\%20Case\%20Points_v2.pdf. [Online; accessed 1/30/2018]. 2018.

[3] B.Boehm. *Software Productivity and Reuse*. IEEE Computer Society. Sept. 1999.

[4] CMU. *CMU Sphinx*. https://cmusphinx.github.io/wiki/. [Online; accessed 1/30/2018]. 2018.

[5] Eclipse. *Eclipse Plugin Development Environment*. http://www.eclipse.org/pde/. [Online; accessed 1/30/2018]. 2018.

[6] Evolus. *Pencil Project*. https://pencil.evolus.vn/. [Online; accessed 1/30/2018]. 2018.

[7] Eclipse Foundation. *Eclipse Next-Generation IDE*. https://www.eclipse.org/che/. [Online; accessed 1/30/2018]. 2018.

[8] Ivar Jacobson Grady Booch and James Rumbaugh. *Unified Modeling Language*. http://www.uml.org. [Online; accessed 1/30/2018]. 2018.

[9] IBM. *Eclipse - The Eclipse Foundation open source community website*. https://www.eclipse.org/. [Online; accessed 1/30/2018]. 2018.

[10] GitHub Inc. *Software development platform · GitHub*. https://github.com. [Online; accessed 1/30/2018]. 2018.

[11] jetbrains. *IntelliJ IDEA: The Java IDE for Professional Developers by JetBrains*. https://www.jetbrains.com/idea/. [Online; accessed 1/30/2018]. 2018.

[12] Jeff Atwood Joel Spolsky. *Stack Overflow - Where Developers Learn, Share, Build Careers*. https://stackoverflow.com/. [Online; accessed 1/30/2018]. 2018.

[13] University of Calgary Kent Beck et al. *JUnit - Unit testing framework*. http://junit.org. [Online; accessed 1/30/2018]. 2018.

[14] oracle. *Netbeans*. https://netbeans.org/. [Online; accessed 1/30/2018]. 2018.

[15] Ok Eclipse Team. *ok eclipse demo - speech recognition within eclipse IDE*. http://tiny.cc/okeclipsedemo. [Online; accessed 1/30/2018]. 2018.

[16] Wikipedia. *MVC Architecture Image*. https://commons.wikimedia.org/wiki/File:MVC_Diagram_(Model-View-Controller).svg. [Online; accessed 1/30/2018]. 2018.

[17] *Writing an Evaluation Plan*. https://www.brown.edu/research/conducting-research-brown/preparing-proposal/proposal-development-services/writing-evaluation-plan.

[1] CSC 510 - 001 Software Engineering - Spring 2018