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

[Project Proposal - 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

Our proposal details the various essential aspects required to build and evaluate our conceived idea. Our goal is to improve user experience in the Integrated development environment(IDE) by incorporating a speech recognition system. We start by defining the problem and our motivation. We divide and translate the problem into manageable requirements that we intend to conquer. We further get into the technical aspects of building the same using industry standard plugin and MVC based architecture to promote parallelism. We also reason out why prototype model and Test-driven development (TDD) is suitable for our software development process. Lastly, we portray an evaluation plan that is aptly suited for this nature of application that we intend to build.

Keywords

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

1. PROBLEM

In the era of abundant voice libraries, we presume that IDEs are not reaping its benefits. Users typically rely on keyboard and mouse and most recently touch displays for a method of input. We see potential in the voice as an additional input method to amplify user experience and thereby productivity.Results show that strategies aligning with following categories could achieve following percentage savings on productivity [4]):

- 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 a gap for an inability to multitask given the voice avenues.

2. REQUIREMENTS & SOLUTION

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

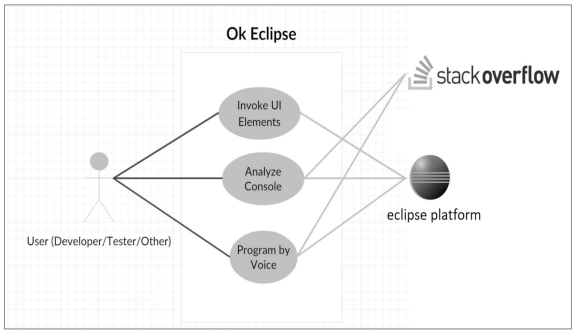

Figure 1: Ok Eclipse - Overall Use Case Diagram

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 required traversing multiple levels down the menu. One might argue the use of short-cut keys. However, to master this huge list of short-cut keys for all the options on the menus (IDE specific) is arduous. We believe, this voice-based feature will aid newbie users to utilize the features of IDE in-par with seasoned users.

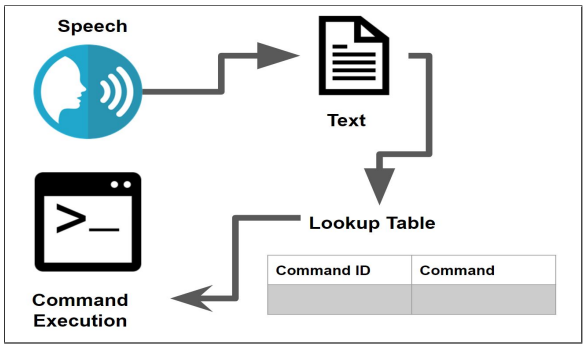

Figure 2: Overview

We identify every menu item in eclipse associated with a command. Commands have unique ID's with the IDE. Hence, we propose to create 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.

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 [11]) repositories for appropriate solutions.

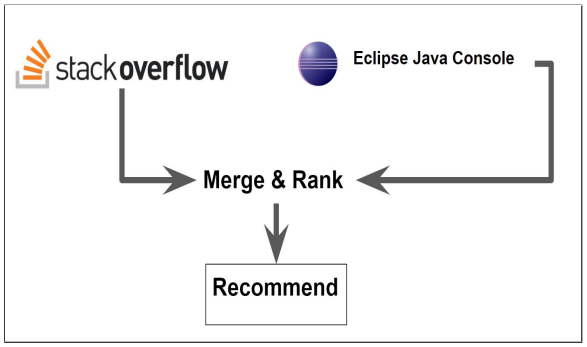

Figure 3: Overview

Whenever the user feels like he might need a recommendation, the user may invoke the "Analyze Console" 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 order of decreasing number of 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.

2.3 Sound Programmer

In this space, we wish to identify the frequently used and also mundane items 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.

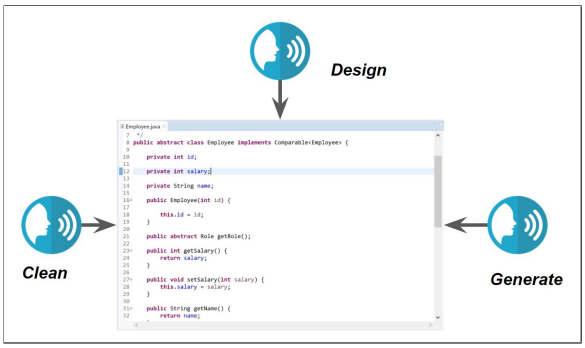

Figure 4: Overview

3. ARCHITECTURE & DESIGN

Having three separate plugins for three different use cases (section 2) allows us to do things in parallel.

Type	Library
Speech to Text	CMU Sphinx [5]
User Interface	SWT and JFace
Development environment	Eclipse PDE
Platform	Eclipse Rich client platform

Table 1: Required Libraries

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.

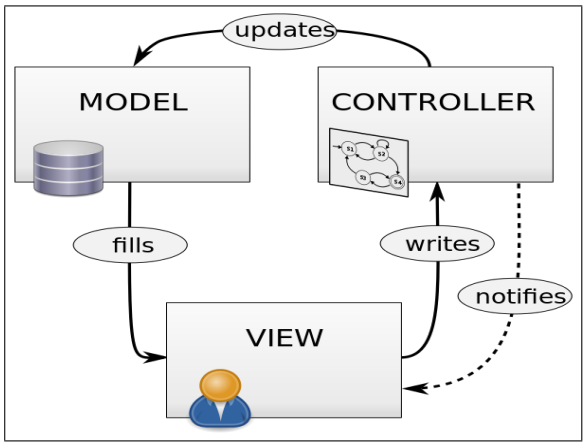

Figure 5: MVC Architecture [14]

We wish to plug our three plugins into eclipse, which is as simple as dropping the plugin jar file into the eclipse plugins folder.

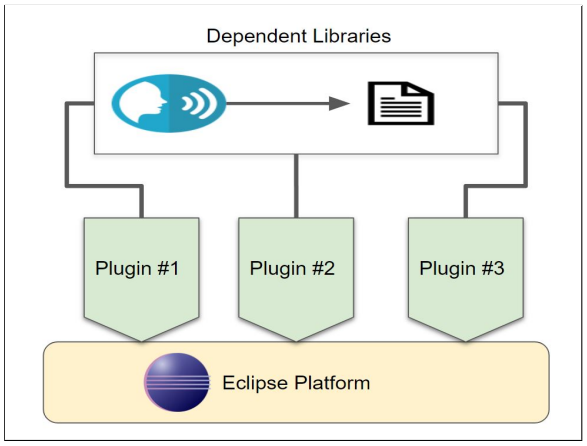

Figure 6: Proposed Architecture

However, the commonality between our 3 plugins is the CMU's 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 such that each plugin can individually consume it to cater to its 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 is 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. Also, it is crucial to adopt a software development process which accurately describes the aspects of the project in hand considering the requirements specified and resources available. Bearing these factors in mind, a Prototyping model of development has been chosen given its simplicity in our case. Further, this model of development provides a means to deliver results early in the timeline of the project.

Also, the Test Driven Development (TDD) methodology will be the primary framework that forms the basis for the entire coding process. This approach to developing code for the software will ensure that the probability of encountering errors in the final product will be kept to a minimum. On the other hand, a TDD based approach will streamline the coding objectives to strictly meet the requirements outlined in the design requirements document. The management and support tasks for this project will be met through software such as Git-Hub[10] for the purpose of maintaining source control, tracking the project milestones (subtasks) and Eclipse IDE for development, compilation and testing of the resulting code.

4.1 Prototyping Model

The underlying objective of a Prototyping Model of development is to adopt an iterative approach to developing software that not only generates the final product well within the targeted date of completion but also uses this final product as a means to help the end-user to envision and revise the design requirements. Figure 7 provides an overview of the steps involved in this developmental strategy. Adhering to this model will help in shifting the focus on quick development while also ensuring that it actively involves the targeted end-user allowing a conclusive design specification document to emerge in the process. This approach is well suited to projects that have abstract design specifications because developing a working model of the system would enable users to get a better understanding of the envisioned system.

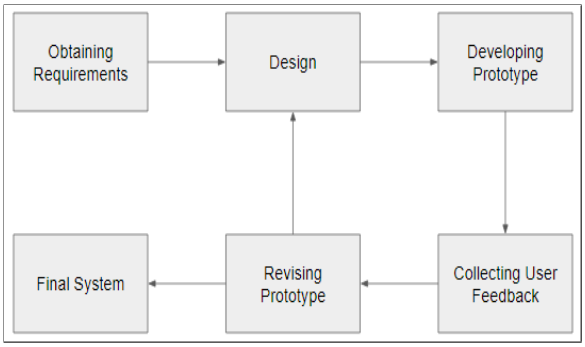

Figure 7: Prototyping Model

It is important to take into account the time and resources available while choosing the model of development. Given the scope of our project, this has a direct implication of producing quick results and also going through as many iterations of development as possible within the available timeframe. Therefore, a prototyping model has been concluded to be the best fit for the developmental stage. This would aid in detecting any limitations in the system or even the possibility of discovering new requirements earlier on in the process while also leading to better solutions due to quicker feedback.

The project has been proposed to implement various functionalities and each of them will be implemented individually and made to satisfy the requirements of the users. That

is, a good-enough prototype of the 'Okay Eclipse' interface shall be built in the first stage of development. In the second stage, this system will be subjected to a rigorous self-evaluation and the results of this process shall be translated into updated design specifications for the system. These considerations will shape the refactoring and refinement process of development. Given the esoteric nature of our project and the constraint of time, it is impossible to obtain a considerable external evaluation of the prototype system and thus, self-evaluation with a critical viewpoint should suffice in shaping the refinement process.

4.2 Test Driven Development Methodology

The Test Driven Development methodology is a novel approach designed to allow the developer to create a feedback system while maintaining a strict and self-imposed conformity to the design requirements. Central to this approach is the concept of the Red-Green-Refactor methodology. The 'Red' portion of this model is dedicated to creating a suite of test cases which provide an overall representation of the abstract design requirements and writing code designed to fail the test suite. This is based on the ideology of failing more and failing fast so as to rapidly learn about the system from them. The 'Green' segment refers to the stage of writing code that can satisfy these test cases to a basic degree without exceeding the functional specifications. This creates a tangible framework through which the developer is made accountable and prevented from deviating from the given specifications. As the software undergoes subsequent iterations of coding as well as functional updates, a process of refactoring on the entire scale of the software is needed so as to provide maintainability and sustainability of code.

We intend to use the following tools for the process of development as shown in the table 2.

Type	Software
Build & Deployment	Eclipse PDE [6]
Testing	JUnit [12]
Issue tracking	Github [10]
Milestones life-cycle	Microsoft Excel/ Kanban[10]
SCM	Github[10]
Design	UML[8]
Wireframe	Pencil[7]

Table 2: Process Tools

Since software engineering deals with the design and construction of 'good-enough' software, it is essential that this software be subject to testing before making it available for usage. This testing must not only aim at maintaining a high quality of code but also provide a mechanism to ensure that the user requirements have been satisfied accurately. Following a test-driven methodology has thus been adopted for our project. This allows maintaining simplicity, modularity, and flexibility in the code base while also providing a built-in process of evaluation against the design. Our initial objective will be to obtain a comprehensive understanding of the underlying requirements for each of the use cases provided in section 2 and then to build test cases for each of them. This further enables us to obtain the perspective of an end user as TDD makes the developer focus on the requirements before writing the code. Following the construction of these test cases, the actual software coding is done in an attempt to pass the test cases. Intuitively, this may result in failure but this provides crucial insight into the short-comings of the current code. Next, the code is improved with an aim of barely passing the existing test cases to satisfy the bare-bone design requirements. The written code is then validated against all the test cases and upon success, the code is deemed to meet the test requirements.

In combination with the Prototyping model, the system resulting at this stage is just the prototype and may be subject to further design changes and the addition of new functionalities as a result. In that case, the code also has to be cleaned up through a Refactor phase which involves removing any code or functional duplication and rearranging certain code segments to make it more readable and easy to maintain. The advantage of this methodology is that early and frequent testing helps to catch defects early in the development cycle thereby eliminating the need for tedious debugging later on and this is crucial for our project.

4.3 Effort Estimation

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.

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

5. INFRASTRUCTURE & TOOLS

We propose to build our application in an environment similar to the tables Software requirement Table 4 & Hardware requirement Table 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 [6]
Testing	JUnit[12]

Table 4: Software requirements

6. EVALUATION PLAN

Any software that has been developed must be evaluated with the end-user perspective in mind [13]. 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 executing options through categorized menu selection, by extending the idea of keyboard shortcuts. It is recognized that since the aim of this project is to improve user experience, quantifying the possible benefit in terms of a metric is difficult. Thus, a two-pronged approach to the evaluation strategy based on quantitative and qualitative assessments has been identified for this project. The appraisal process for our project has two stages, namely, a formative and summative stage [15].

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.

The proposed framework is based on three key aspects, maintainability, sustainability and usability. With the aim of developing software that is maintainable, we incorporate 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. For the third aspect of assessing usability at this stage, we plan to conduct a self-evaluation (refer table 5).

Functionality	IDE(sec)	Okay Eclipse IDE(sec)
1		
2		
...		
n		

Table 5: Internal Evaluation

6.2 Summative Stage

In the final stage, a system test to assess the accomplishment of goals set out in this document will be conducted. In addition to the self-evaluation conducted during the formative stages of development, a system-wide test to assess the project on these aspects will be conducted at this stage. Also, an additional test to appraise the software based on usability to the end-user, a survey or a similar scheme to qualitatively and quantitatively specify the same will be organized. On the quantifiable end, to evaluate usability, a possible evaluation tool in addition to the evaluation based on the measurement of delay experienced in accomplishing a certain task in the IDE, is an User Experience Survey.

Thus, an opinion survey or a behavioral study can be conducted to complete the qualitative aspect of evaluation of the system. We plan to adhere to the industry standard **System Usability Scale** to measure User Experience with peers and external mediums such as AMT[3] as applicable (refer Table 6). This is necessary to bolster the argument in favour of our system. That is, in cases where delays encountered in using keyboard shortcuts are indeed lesser (compared to menu selections), then a further assessment of how many users have the shortcut memorized, will determine the pitfall of the keyboard shortcut method upon which our system improves. The survey shall be a crucial tool to assess the system in terms of this metric. The results of this evaluation will be included in the final project report.

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

Table 6: External Evaluation

7. CONCLUSION

We elucidated in detail about the essential aspects to build our project[1] with the use of standard software engineering principles and practices. We identified the various processes, methodologies, architecture, design, tools, libraries and infrastructure suited to build this project. Finally, we devised a suitable evaluation plan that we presume will be the most effective way to measure the intended goal of our project.

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] Amazon. *Amazon Mechnaical Turk.* https://www.mturk.com. [Online; accessed 1/30/2018]. 2018.

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

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

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

[7] Evolus. *Pencil Project.* https://pencil.evolus.vn/. [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.

[1] CSC 510 - 001 Software Engineering - Spring 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] Jeff Atwood Joel Spolsky. *Stack Overflow - Where Developers Learn, Share, Build Careers*. https://stackoverflow.com/. [Online; accessed 1/30/2018]. 2018.

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

[13] *Software Evaluation Guide*. https://www.software.ac.uk/resources/guides-everything/software-evaluation-guide.

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

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