OkEclipse: Voice command based Software Development

Vismay Golwala
NC State University
vjgolwal@ncsu.edu

Dharmang Bhavsar
NC State University
dmbhavsa@ncsu.edu

Srija Ganguly
NC State University
sgangul2@ncsu.edu

Tushita Roychaudhury
NC State University
troycha@ncsu.edu

ABSTRACT

Natural Language Processing has impacted some of the major industries[6] resulting in some superlatively innovative changes. But the software development process has still been largely untouched by this powerful technique. Taking this into consideration, we try to incorporate some Natural Language Processing features into the already existing feature rich OkEclipse plug-in. We try to extend the use of NLP from just development phase in the Software Development Life Cycle to the Testing phase as well. New features helping programmers work with multiple languages are also planned to be added in our implementation of the project.

Keywords

Eclipse, Java, Python, voice command, testing, version control, code generation

1. INTRODUCTION

"Voice based softwares" are a thing right now. But what about "Voice based software development"? We ask. OkEclipse is an answer to that. The system worked in helping programmers code by giving them features like code searching, error searching etc directly in the development environment. Although the system is working effectively, we saw a lot of opportunities to make it better. So, our efforts aim to expand this work by integrating voice based commands throughout the software development life cycle. Also, some more work is put in to make programmer's life easier by making features such as code conversion, code generation and voice based version control. Previous efforts in integrating NLP in SDLC deemed to be successful.[1][2][3] The main aim is to start a "voice based innovation streak" in the all the phases of SDLC in the same way that that voice recognition technology has affected other domains.

2. EXISTING SYSTEM

In the first phase of this project, OkEclipse intended to enhance the user experience of the Eclipse Integrated Development environment by incorporating speech recognition. The current OkEclipse implementation has the following features:

- Listening Menus: Menu options, even when highly nested, can be invoked with the user reciting the option label text. Upon receiving voice input, the text-equivalent is matched with a lookup table to retrieve the corresponding Command ID. Then this request is passed to the eclipse command framework to invoke the said command.
- Loud Console: Whenever the user needs a recommendation regarding the issues listed in the console log, the user can invoke the "Find" command to get suggested solutions from crowd-sourced Q&A repositories and also relevant YouTube videos to troubleshoot the issue.
- Sound Programmer: This feature aims to reduce the time spent on certain frequent and mundane tasks like generating getters and setters, creating the main method, writing the toString function. These tasks have been implemented to be achieved by voice commands.

However, the existing implementation hit a few challenges:

- Constrained Dictionary: The large dictionary size of the CMU Sphinx Voice recognition library caused several voice commands to get misrecognized. Hence, a separate dictionary was created with application relevant keywords for use in the implementation thus improving command recognition and reducing the processing time involved.
- Disparate Keywords: The number of keywords in the voice command was limited to one and phonetically disparate commands were used.

3. USER SURVEY

This section describes the questions that were asked in the user surveys, the reason for asking those questions, the results and the conclusions that we derived from the results.

Question 1: **How do you feel about being able to manage your github repository directly from the Eclipse IDE using voice commands?**

The reason for asking that question was to get a feel of whether integrating Github version control support in the Eclipse plug-in. From the first iteration of the project, we personally found that it was inconvenient to open command line and then push the code to Github even if the project was already cloned in the existing directory.

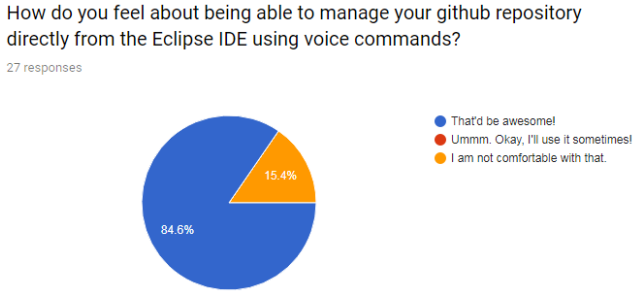

Figure 1: Responses for survey question 1

The results in Figure 1 are interesting as a majority of users ~85% wanted such a system. They might be going through the same inconvenience that we did go through in the project.

Question 2: What do you think about Eclipse detecting and warning the user about unused code as a sanity check before pushing code into the version control system?

The reason for asking that question was to know whether people go through such a scenario where they have made functions or variable and have not used in the program. Pusing such subroutines and variables in the version control system leads to overhead in maintaining code. So it is better to let such subroutines and variables go.

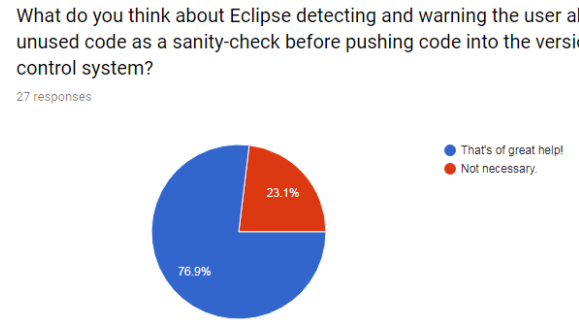

Figure 2: Responses for survey question 2

As shown in Figure 2, the results for this question were largely popular with ~76% of surveyors wanting such a feature.

Question 3: Would it help if Eclipse IDE generated code syntax of different constructs, depending on a chosen language, via a simple voice command?

The question was aimed at knowing whether people were interested in a system that generated code according to certain requirements. Code generation is no mean feat (if it was, why we would be studying CS).

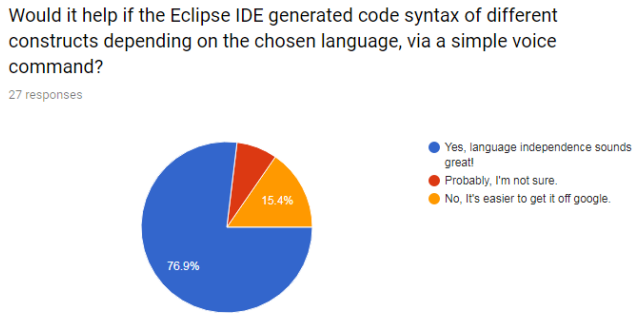

Figure 3: Responses for survey question 3

Again the results were largely positive (Figure 3), with some unsure and some negative reviews. This indicated that people would like such a feature to be present in the IDE itself rather than going to Google to search about that.

Question 4: How would you like voice commands that automate unit testing of your code?

As we said, we are trying to integrate voice command features into different SDLC life cycle phases. This was an attempt to get to know the sentiment of public for such a system. From Figure 4, it is evident that the response was totally positive and was the highest ranked feature that surveyors wanted.

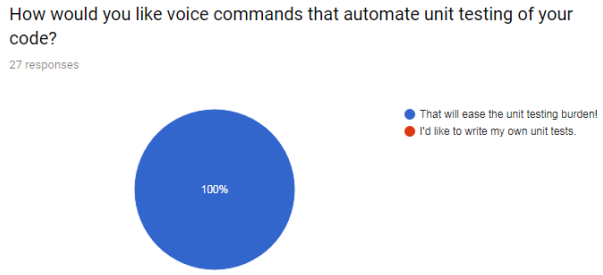

Figure 4: Responses for survey question 4

4. IMPLEMENTATION

4.1 Use Cases

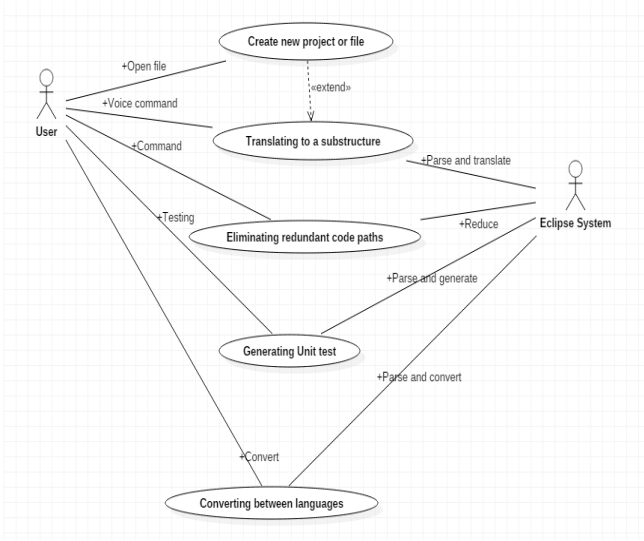

Figure 5: Use case diagram

4.1.1 Use case 1

Use Case Name	Translating to a substructure
Actors	User , Eclipse System
Description	User will voice command to generate a substructure. Eclipse system will parse the input and generate the structure.
Trigger	Use case is triggered when the user selects voice command feature and speaks out a command having similar syntax
Preconditions	1. User has a PC 2. User has active internet connection 3. User has Eclipse 4. Project has been created
Post conditions	1. Substructure syntax created

Normal Flow	1. User creates new project 2. User selects voice command feature and speaks 3. System interprets
Special Requirements	Sneeze and other noise needs to be controlled
Assumptions	User understands and speaks English
Notes and Issues	Interface is basic and provides a trigger to the main backend work of the software

Table 1: Use case 1

4.1.2 Use case 2

Use Case Name	Eliminating redundant code paths
Actors	User , Eclipse System
Description	The user would want to eliminate unreachable and redundant code, which can be achieved by this
Trigger	Use case is triggered when the user provides a command asking for code elimination
Preconditions	1. User has a PC 2. User has active internet connection 3. User has Eclipse
Post conditions	Eclipse updates the file with removed code
Normal Flow	1. User creates a project and starts coding 2. User invokes code elimination command
Special Requirements	Noise and sneeze needs to be kept in control
Assumptions	User understands and speaks English
Notes and Issues	Interface is basic and provides a trigger to the main backend work of the software

Table 2: Use case 2

4.1.3 Use case 3

Use Case Name	Generating unit test
Actors	User , Eclipse System
Description	Unit testing cases are generated automatically
Trigger	Use case is triggered when the user speaks a command that asks for generating unit tests
Preconditions	1. User has a PC 2. User has active internet connection 3. User has Eclipse
Post conditions	Eclipse shows an updated project with test cases
Normal Flow	1. User finishes developing the application and wants to test it 2. User selects voice command and speaks for unit testing 3. System generates unit test
Special Requirements	Voice commands require some specific sensitivity
Assumptions	User understands and speaks English
Notes and Issues	Interface is basic and provides a trigger to the main backend work of the software

Table 3: Use case 3

4.1.4 Use case 4

Use Case Name	Converting between languages
Actors	User , Eclipse System
Description	The user will be able to convert between languages like Java to Python for basic structure
Trigger	Use case is triggered when the user selects or chooses the command to convert to another language
Preconditions	1. User has a PC 2. User has active internet connection 3. User has Eclipse
Post conditions	Eclipse shows a new project with the substructure reflected in a different language
Normal Flow	1. User commands code conversion 2. User provides a new file name 3. System generates current code structure in the required language
Special Requirements	Voice commands can be susceptible to noise and code generation should be of simple code written
Use Case Name	Converting between languages
Actors	User , Eclipse System
Assumptions	User understands and speaks English
Notes and Issues	Interface is basic and provides a trigger to the main backend work of the software

Table 4: Use case 4

4.2 Proposed Features

The component features of the system are described below:

- **Code Generation** - The application will generate code according to the details specified by the user. For example, an input from the user to make a function with 3 parameters will create a function with a generic name and 3 void parameters. For code generation, we are hard coding the things that it will be able to generate. The generated code will be OS independent. Code generation will work for two languages namely Python and Java. If code generator is called in a .py file, Python code will be generated and if it's called from a .java file, Java code will be generated. If file is not saved, default will be set to Java code.
- **Code Conversion** - The application will be able to convert Java code to Python with satisfactory accuracy. For conversion, we are using java2python library. The library is able to handle some corner cases like different access modifiers in variables and different syntax for comments and is also OS independent, thus, making ours, the only Eclipse plug-in to do something like Java to Python conversion.
- **Unit Testing** - For unit testing, the application will use the AgitarOne library in Java which automatically generates test cases for given function. The library has an interface called JUnit Factory for managing the cases and asserting their correctness. The library usually provides a 30 days trial, however with it's university program, it is provided at no cost. The library basically finds all combinations of inputs that can be passed to a specific subroutine to test it. The results and outputs from the subroutine from these tests will be displayed to the user and they can make changes according to the results.
- **Github voice integration** - Programmers use version control for maintaining large codebases. Github is one such VCS. We are trying to ease the pain of opening command line every time some changes needs to be pushed. Instead, that will be replaced by basic voice commands such as pull, commit and push.
- **Removing unused code** - Unused code can be a big problem while maintenance of code. We are trying to tackle this problem by removing such unused subroutines and variables when the user pushes the code to Github. Permission will be asked before removing any code from the user.

4.3 Benefits

- The application tries to integrate another phase of software development life cycle into the mix through integration of testing along with development through voice commands.
- This will open new frontiers for a full fledged system being developed in the future that might be enough for the whole software development life cycle.
- Code Generation helps coders by giving basic syntaxes and saving their valuable time through generating basic code snippets according to the requirements.
- Code Conversion will help when an application is being migrated from one language to another.
- Unused code removal will be beneficial when maintaining code because it removes the overhead of rendezvous with unused code.
- Github integration will save valuable time because no more the programmer will have to open the command line to pull, commit and push changes to the version control system.

4.4 Technologies used

Our system uses simple and easily available technologies. The description of the languages, environments, API and libraries used are as follows. The application is an Eclipse plug-in so it is majorly built in Java. There are some components in the application that use some other languages such as Python.

- Languages – Java, Python
- Libraries – Agitar One (java unit testing)[8], java2python (java to python conversion)[9], Sphinx (voice detection).[7]
- Environment – Eclipse[4]
- API – GitHub integration API[5]

4.5 Plan of Action

- To start with the development, we will be implementing and hard coding the code generation bits of the application. As we have to do this for more than one language, it will take time.
- For code conversion, we will integrate the Python library java2python and include python calls in the Java code. Meanwhile, we will also be evaluating this library for correctness.
- Github API will be used for integrating version control in the Eclipse workspace through voice commands.
- Unit testing library Agitar One will be integrated.
- Coming up with a phonetically different voice command for the above functions will be the key for success for the project.
- Searching for different subroutines and variables that are not used and removing them when pushing to the version control system will be implemented. Permission will be asked when removing the code from the user.
- Integrating all these changes into the main plug-in file of OkEclipse and finally doing rigorous testing through evaluations to make sure the system works.

5. EVALUATION PLAN

The aim of the project is to provide easy to use interface for fast development of code using Eclipse. For this problem, we plan evaluate our system on two bases:

- Internal - Since a major chunk of our functionality is heavily dependent on the application recognizing voice commands, a log file will be maintained that records the failure or success of speech to command mapping and eventual text conversion. We will also record the response time of each of our feature implementation to evaluate if the functionality does indeed improve developer efficiency by cutting down significantly on the time taken to do those tasks manually otherwise.
- External - To evaluate the effectiveness of our implementation in improving the user experience working with Eclipse IDE, we plan to record responses of end-users using our application in a noise-controlled environment. We will ask participants to perform development tasks utilizing our features and observe them. We will conduct a structured interview for every participant to garner their comfort in using the features - whether it significantly improves their experience, correctly does the intended task, the extent to which it generates bug free code, if the features have a non-competitive response time etc.

6. TIMELINE

- Week 1 - Developing codebase for code generation and code conversion. Code conversion will be done with a Python script so a Python file call will be needed through the plug-in. Code Generation will be hard-coded for certain test cases. Adding new keywords in the Sphinx Voice detection library.
- Week 2 - Github Integration, Unit Testing and unused subroutines and variables searching and removing to be implemented.
- Week 3 - Integrating these changes with the prior implementation of OkEclipse.

7. CHALLENGES

We expect certain challenges to come our way. Some already comprehended ones are:-

- Integrating Python calls into the Java codebase for the Eclipse plugin will be a challenge because Eclipse Plugins are predominantly made in Java, but, our application uses Python library named java2python library for code conversion.
- Code Generation is an active area of research and understanding the context in which user wants the code in is a big challenge as well. For that we have made a small pool of commands which will generate codes for some predetermined structures such as classes, subroutines, variables etc.
- Code Conversion has been a big problem in the software industry for decades and trying to attempt something of such a huge scale in such less time was not sensible. So, we are using a Python library java2python for conversion.
- Adding new voice commands to the Sphinx voice detection library will be a challenge as well because the commands that we will be adding will have to be phonetically different from the commands already present.

8. CONCLUSION

In this report, we describe the base implementation that we are working with and then go on to present detailed the use cases of the ideas we wish to implement to enhance the existing OkEclipse implementation. We carefully evaluated the challenges and limitations of the existing implementation and solicited popular opinion on what would make the application more developer friendly and most importantly what features would significantly ease the burden of certain mundane development tasks. Our efforts, in this second phase, would be directed towards enhancing the already existing functionality and also add additional functionality that we found to be popular in the survey we conducted. Finally, the outlined evaluation plan will best help us determine the impact of our application and if we succeeded in enhancing the user experience even further.

9. ACKNOWLEDGMENTS

Our special thanks to our Teaching Assistants, Amritanshu Agrawal and Ken Tu along with Prof. Timothy Menzies for helping us on ways to expand the current system and to develop a work plan to successfully execute the advances.

10. REFERENCES

[1] Prashant Yalla, Nakul Sharma. *Integrating Natural Language Processing and Software Engineering.* International Journal of Software Engineering and Its Applications, Vol. 9, No. 11 (2015), pp. 127-136 http://dx.doi.org/10.14257/ijseia.2015.9.11.12

[2] Imran Sarwar Bajwa, Ali Samad, Shahzad Mumtaz. *Object Oriented Software Modeling Using NLP Based Knowledge Extraction.* European Journal of Scientific Research ISSN 1450-216X Vol.35 No.1 (2009), pp 22-33 http://www.eurojournals.com/ejsr.htm

[3] Aman Chandra, Sakthivasan. *Enhancing SDLC traceability using NLP*. https://www.wipro.com/holmes/enhancing-sdlc-traceability-using-nlp/ [Online; accessed 3/21/2018]. 2018

[4] Eclipse. *Eclipse Plugin Development Environment*. http://www.eclipse.org/pde/. [Online; accessed 3/23/2018]. 2018

[5] GitHub Inc. *Software development platform · GitHub*. https://github.com. [Online; accessed 3/24/2018]. 2018

[6] Parameshachari B D, Sawan Kumar Gopy, Gooneshwaree Hurry, Tulsirai T. Gopaul. *A Study on Smart Home Control System through Speech*. International Journal of Computer Applications (0975 – 8887) Volume 69– No.19, May 2013

[7] CMU. *CMU Sphinx*. https://cmusphinx.github.io/wiki/. [Online; accessed 3/26/2018]. 2018.

[8] Agitar. *Agitar One.* http://www.agitar.com/solutions/products/agitarone.html . [Online; accessed 3/27/2018]. 2018.

[9] java2python. *Simple but effective tool to translate Java source code into Python*. https://pypi.python.org/pypi/java2python [Online; accessed 3/27/2018]. 2018.