

Intelligent Code Editor with Code Suggestion and Code Correction

DESIGN DOCUMENT

Team 29

Iowa State University (Ali Jannesari)

Ali Jannesari

Jacob Puetz, Ben Gonner, Cory Smith, Evan Christensen, Jordan
Silvers

sdmay21-29@iastate.edu

Team Website

<https://sdmay21-29.sd.ece.iastate.edu/>

Revised: 10/4/20

Executive Summary

Development Standards & Practices Used

We used standard industry practices as taught in our previous classes while working on the Intelligent Code Editor. We are planning on using Test-Driven Development as well for the project as it has been highly praised by professors throughout the Software Engineering and Computer Science departments. This will allow us to test the quality of the Intelligent Code Editor based on the number of bugs we discover from our process. We are also planning on implementing the design thinking process to allow us to be able to empathize with the users and clients on the Intelligent Code Editor.

Summary of Requirements

- Ability for user to input code and natural language into an editor
- Ability to store the natural language and expected code translation
- Ability for plugin to classify and translate natural language to code
- Ability for user to translate the natural language to code
- Ability for code to be executed once translated
- Ability for multiple natural language statements to be translated to code at once
- Fix any bugs that exist from previous versions

Applicable Courses from Iowa State University Curriculum

- COMS 227 (Introduction to Object-Oriented Programming)
- COMS 228 (Introduction to Data Structures)
- COMS 309 (Software Development Practices)
- COMS 230 (Discretes CMP Structures)
- COMS 311 (Algorithm Design and Analysis)
- SE 329 (Software Project Management)
- SE 339 (Software Architecture and Design)
- SE 362 (Object-Oriented Analysis and Design)

New Skills/Knowledge acquired that was not taught in courses

List all new skills/knowledge that your team acquired which was not part of your Iowa State curriculum in order to complete this project.

- IDE plugin Development
- Graph Neural Networks
- Semi-Supervised Learning for Neural Machine Translation
- Unsupervised Machine Translation
- Natural Language Processing
- Data Formatting and Cleansing

Table of Contents

1 Introduction	5
Acknowledgement	5
Problem and Project Statement	5
Operational Environment	5
Requirements	5
Intended Users and Uses	5
Assumptions and Limitations	6
Expected End Product and Deliverables	6
Project Plan	7
2.1 Task Decomposition	7
2.2 Risks And Risk Management/Mitigation	8
2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria	9
2.4 Project Timeline/Schedule	9
2.5 Project Tracking Procedures	10
2.6 Personnel Effort Requirements	10
2.7 Other Resource Requirements	11
2.8 Financial Requirements	11
3 Design	12
3.1 Previous Work And Literature	12
Design Thinking	12
Proposed Design	12
3.4 Technology Considerations	12
3.5 Design Analysis	12
Development Process	13
Design Plan	13
4 Testing	13
Unit Testing	13

Interface Testing	14
Acceptance Testing	14
Results	14
5 Implementation	14
6 Closing Material	14
6.1 Conclusion	14
6.2 References	14
6.3 Appendices	15

List of figures/tables/symbols/definitions (This should be the similar to the project plan)

Figure 1: Gantt Chart

8

1 Introduction

1.1 ACKNOWLEDGEMENT

This project was started by last year's senior design team sday 20-46.

1.2 PROBLEM AND PROJECT STATEMENT

Our project is an extension for a code editor that takes natural language statements and translates them into functional code. We plan to achieve this by implementing a graph neural network to classify words and phrases. This extension could change the way we approach programming. It offers a new way to think about software problems and a more natural transition from traditional solutions to syntactically correct solutions. We will expand upon this extension by implementing code corrections and suggestions as well. This will further improve the user experience for users new to programming and help them learn.

1.3 OPERATIONAL ENVIRONMENT

The end result of our project is a plugin for the IntelliJ IDEA Java IDE. As such, it will be able to run on any computer capable of running the IDE. Additionally, the computer will need to have Python and OpenNMT-py installed, and be actively running the OpenNMT-py server for the plugin to be operational.

1.4 REQUIREMENTS

- To create a plugin to IntelliJ that is able to convert natural language to code.
- Fix any bugs that exist from previous developers.
- Refine and add code corrections and suggestions to help developers make better programs.
- Users should be able to input both code and natural language into their editor.
- The plugin should store both natural language and its expected translation.
- The plugin should be able to translate natural language to equivalent code.
- The plugin should allow users to convert their natural language statement to code after the translation is complete.
- The code translation should run as-is, without modification by the user.
- The plugin should be able to translate multiple lines of natural language at once.
- The project will be entirely digital.
- All work on the project will be done remotely.

1.5 INTENDED USERS AND USES

Our plugin is designed for inexperienced Java developers who are still learning how to convert their ideas into real code. With this in mind, our main users will be students in introductory programming classes or individuals who are learning to program on their own. These users will use

the plugin to convert their natural language statements into code and to find common errors in their code to make their programs run the way they want.

1.6 ASSUMPTIONS AND LIMITATIONS

Assumptions:

- The end user has an internet connection when they are programming.
- The end user will be able to pick the best code suggestions that fit their purpose.

Limitations:

- The final product will not work without an internet connection as it needs to access a code translation server.
- Currently only works with IntelliJ for functionality implementation.

1.7 EXPECTED END PRODUCT AND DELIVERABLES

We will improve upon the current system for translating natural language phrases into syntactically correct code snippets. November 2020.

We will write a document describing instructions for users to install the extension and other initial set-up steps. April 2021.

We will expand upon the extension to provide suggested corrections for common mistakes, and auto complete code snippets. April 2021.

2 Project Plan

2.1 TASK DECOMPOSITION

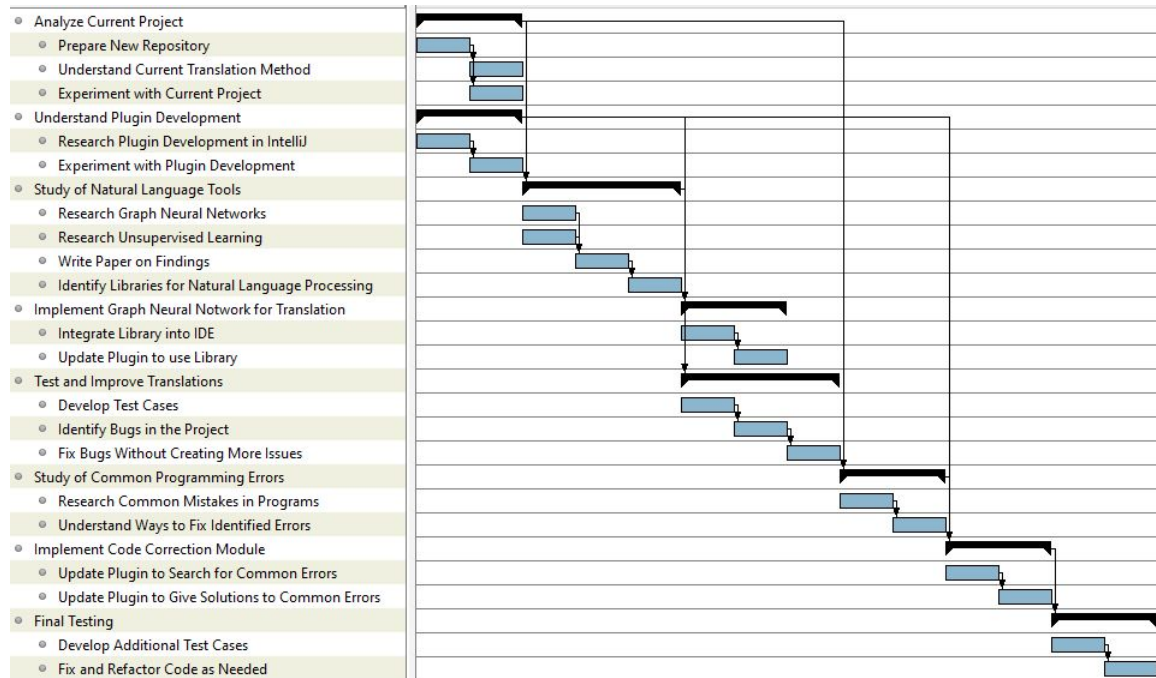


Figure 1: Gantt Chart

Analyze Current Project

- Prepare new repository: Move code from the old repository to the new repository.
- Understand Current Translation Method: Understand how the version one code translates natural language.
- Experiment with Current Project: Experiment with the version one code on our local machines.

Understand Plugin Development

- Research Plugin Development in IntelliJ: Research how plugin development works in our current IDE (IntelliJ).
- Experiment with Plugin Development: Experiment with plugin development on our local machines.

Study of Natural Language Tools

- Research Graph Neural Networks: Research how graph neural networks can be used in natural language processing.
- Research Unsupervised Learning: Research how unsupervised learning can be applied in the project.

- Write Paper on Findings: Write a paper to deliver to the client on how graph neural networks and unsupervised learning can be used.
- Identify Libraries for Natural Language Processing: Identify libraries that could be used within our project to translate natural language into code.

Implement Graph Neural Network for Translation

- Integrate Library into IDE: Download our chosen library and add it into our project.
- Update Plugin to Use Library: Update the plugin to make it use the graph neural network library instead of the old system.

Test and Improve Translations

- Develop Test Cases: Develop some test cases to ensure that the project meets the client's requirements.
- Identify Bugs in the Project: Identify bugs in the current project based on test cases.
- Fix Bugs Without Creating More Issues: Fix the identified bugs so that the project meets its requirements without introducing new errors.

Study of Common Programming Errors

- Research Common Mistakes in Programs: Research common mistakes that developers make when working on programs.
- Understand Ways to Fix Identified Errors: Determine the best ways that the identified mistakes can be resolved.

Implement Code Correction Module

- Update Plugin to Search for Common Code Errors: The plugin should be capable of finding common code errors as well as natural language.
- Update Plugin to Give Solutions to Common Errors: Expand the plugin's capabilities to give suggestions for common code errors as well as natural language translations.

Final Testing

- Implement Tests: Create tests for many use cases, including corner cases for maximum coverage.
- Fix and Refactor the Code as Needed: As tests fail to pass, make any necessary changes to the code base to fix the problem.

2.2 RISKS AND RISK MANAGEMENT/MITIGATION

Analyze Current Project

- If the plugin's code is not understood by the team future development could be slowed. 10%
- error getting the team members machines to run the code might occur. 90%

- Mitigation plan: discuss issues running software with the client for insight.

Understand Plugin Development

- If the plugin development resources are not understood future development could be slowed. 5%

Study of Natural Language Tools

- Struggling to comprehend graph neural networks will slow or halt progress. 20%

Implement graph neural network for code translation

- An improperly trained neural network will have a low accuracy rate. 14%
- Build errors might occur with the implementation of the module. 30%

Test and Improve Translations

- Fixing a bug has a chance to reveal new bugs. 40%

Study of Common Programming Errors

- People always make errors and there will probably be a large number of different code errors people make. 15%

Implement Code Correction Module

- Build errors might occur with the implementation of the module. 30%

Final Testing

- Tests are not implemented correctly and bugs that exist are not found through testing. 10%

2.3 PROJECT PROPOSED MILESTONES, METRICS, AND EVALUATION CRITERIA

- The extension shall translate natural language into syntactically correct code snippets with 90% accuracy.
- The extension shall translate natural language phrases at a rate of 100 phrases per second.
- The extension shall provide auto-completion suggestions within 10 ms of the last keystroke.
- The extension shall identify code errors with 80% accuracy.

2.4 PROJECT TIMELINE/SCHEDULE

The first step of the project was the analysis phase. As shown in Figure 3, this phase consists of three parts: prepare, analyze, and experiment. We first move the old code into the new repository, then we analyze the code to gain a better understanding of it, and lastly we experiment with the code.

Secondly, the team started to research IntelliJ plugin development to get an understanding of how plugins are made and how to work on the existing project. The team also might experiment with plugin creation and implementation during the phase.

Third, we looked into Natural language tools. This phase includes researching graph neural networks along with both semi-supervised and unsupervised learning and identifying the current libraries to use. We will then write and present a paper for our client based on our findings.

After looking into natural language tools the team will be implementing a graph neural network to quickly and accurately translate the natural language into code.

We then are going to create tests for the previous implementation. We will develop specific test cases to cover as much area as we can to find bugs in the code. We will then try to fix any bugs that we find without compromising the existing code.

We will develop new features into version 2 of the code. New features added will include identification of common programming errors and suggested fixes for errors found.

Finally, we will add in test cases for the version 2 of the code. This will be a similar process as our test cases before but will be geared more towards the version 2 of the code. We will then identify and fix any bugs we find for version 2. We will also refactor the code as needed so other groups down the line do not have a problem identifying the parts of the program and can work more easily with it after we are gone.

2.5 PROJECT TRACKING PROCEDURES

For our project tracking we are using Git, Trello, and Discord. Git allows us to share the code between each other and our client. Trello allows us to keep track of tasks that are planned, in progress, and completed. The Trello is managed/maintained alongside the client. Discord is used as a way for the team to communicate with each other and share helpful information and links.

2.6 PERSONNEL EFFORT REQUIREMENTS

Task	Number of Hours
ANALYZE	11 hours
Move Old Code to New Repository	1 hour
Understand Version 1 Code	5 hours
Experiment with Version 1 Code	5 hours
RESEARCH	35 hours

Research IDE Plug-in Development	5 hours
Research Semi-Supervised Learning	10 hours
Research Unsupervised Learning	10 hours
Research Natural Language Processing	10 hours
VERSION 1 CODE	70 hours
Bug Fixes from Version 1 Code	20 hours
Improvements on Natural Learning from Version 1 Code	50 hours
VERSION 2 CODE	85 hours
Research Common Types of Erroneous Code	10 hours
Implement Code Correction Model	75 hours
TESTING	100 hours
Write Unit and Integration Tests	25 hours
Run Usability Tests	25 hours
Fix and Refactor Code	50 hours

2.7 OTHER RESOURCE REQUIREMENTS

Given that our project is a plug-in for Intelli-j, we currently do not have any other needs for physical parts or materials. We are also not expecting to need any physical parts as our project is solely based on the plug-in and the integrated development environment. Our only other dedicated requirement outside of physical is a dedicated GPU server to train our classification model on the dataset.

2.8 FINANCIAL REQUIREMENTS

There are no financial resources required as everything we need to use is free/given to us. We are also not expecting any surprise finances to appear throughout the course of the project.

3 Design

3.1 PREVIOUS WORK AND LITERATURE

Include relevant background/literature review for the project

- If similar products exist in the market, describe what has already been done
- If you are following previous work, cite that and discuss the **advantages/shortcomings**
- Note that while you are not expected to “compete” with other existing products / research groups, you should be able to differentiate your project from what is available

Detail any similar products or research done on this topic previously. Please cite your sources and include them in your references. All figures must be captioned and referenced in your text.

3.2 DESIGN THINKING

Detail any design thinking driven design “define” aspects that shape your design. Enumerate some of the other design choices that came up in your design thinking “ideate” phase.

3.3 PROPOSED DESIGN

Include any/all possible methods of approach to solving the problem:

- Discuss what you have done so far – what have you tried/implemented/tested?
- Some discussion of how this design satisfies the **functional and non-functional requirements** of the project.
- If any **standards** are relevant to your project (e.g. IEEE standards, NIST standards) discuss the applicability of those standards here
- This design description should be in **sufficient detail** that another team of engineers can look through it and implement it.

3.4 TECHNOLOGY CONSIDERATIONS

Highlight the strengths, weakness, and trade-offs made in technology available.

Discuss possible solutions and design alternatives

3.5 DESIGN ANALYSIS

- Did your proposed design from 3.3 work? Why or why not?
- What are your observations, thoughts, and ideas to modify or iterate over the design?

3.6 DEVELOPMENT PROCESS

Discuss what development process you are following with a rationale for it – Waterfall, TDD, Agile. Note that this is not necessarily only for software projects. Development processes are applicable for all design projects.

3.7 DESIGN PLAN

Describe a design plan with respect to use-cases within the context of requirements, modules in your design (dependency/concurrency of modules through a module diagram, interfaces, architectural overview), module constraints tied to requirements.

4 Testing

Testing is an **extremely** important component of most projects, whether it involves a circuit, a process, or software.

1. Define the needed types of tests (unit testing for modules, integrity testing for interfaces, user-study or acceptance testing for functional and non-functional requirements).
2. Define/identify the individual items/units and interfaces to be tested.
3. Define, design, and develop the actual test cases.
4. Determine the anticipated test results for each test case
5. Perform the actual tests.
6. Evaluate the actual test results.
7. Make the necessary changes to the product being tested
8. Perform any necessary retesting
9. Document the entire testing process and its results

Include Functional and Non-Functional Testing, Modeling and Simulations, challenges you have determined.

4.1 UNIT TESTING

– Discuss any hardware/software units being tested in isolation

4.2 INTERFACE TESTING

– Discuss how the composition of two or more units (interfaces) are to be tested. Enumerate all the relevant interfaces in your design.

4.3 ACCEPTANCE TESTING

How will you demonstrate that the design requirements, both functional and non-functional are being met? How would you involve your client in the acceptance testing?

4.4 RESULTS

- List and explain any and all results obtained so far during the testing phase
 - Include failures and successes
 - Explain what you learned and how you are planning to change the design iteratively as you progress with your project
 - If you are including figures, please include captions and cite it in the text

5 Implementation

Describe any (preliminary) implementation plan for the next semester for your proposed design in 3-3.

6 Closing Material

6.1 CONCLUSION

Summarize the work you have done so far. Briefly re-iterate your goals. Then, re-iterate the best plan of action (or solution) to achieving your goals and indicate why this surpasses all other possible solutions tested.

6.2 REFERENCES

List technical references and related work / market survey references. Do professional citation style (ex. IEEE).

6.3 APPENDICES

Any additional information that would be helpful to the evaluation of your design document.

If you have any large graphs, tables, or similar data that does not directly pertain to the problem but helps support it, include it here. This would also be a good area to include hardware/software manuals used. May include CAD files, circuit schematics, layout etc., PCB testing issues etc., Software bugs etc.