Intelligent Python programming environment to identify and follow up on struggling students

Project Plan

| Members:          | RAI, Adarsh           |
|                  | (3035366714)         |
|                  | NAMAKKAL RAGHAVENDRAN, Ajit Krishna |
|                  | (3035369443)         |
| Supervisors:     | Dr. H.F. Ting        |
|                  | Prof. T.W. Lam       |
| Date of Submission: | 4 October 2020     |

COMP4801 Final Year Project
Department of Computer Science
The University of Hong Kong
# Table of Contents

1. Introduction 3
   1.1 Background 3
      1.1.1 Current state of programming education 3
      1.1.2 Existing applications of static code analysis 4
   1.2 Objective 5

2. Methodology 6
   2.1 Scope 6
      2.1.1 Target Audience 6
      2.1.2 Potential Paths 6
   2.2 Potential Deliverables 6
      2.2.1. Research Paper 6
      2.2.2. Feature-rich Platform 7
   2.3 Data 7
      2.3.1 Data Collection 7
      2.3.2 Data Labelling 8
      2.3.3 Domain Models 8
   2.5 Analysis 9
      2.5.1 Syntax Trees 9
      2.5.2 Code Quality Metrics 13

3. Schedule and Milestones 15
   3.1 Estimated Timeline 15
   3.2 Division of Labor 16

4. Bibliography 17
1. Introduction

1.1 Background

This project was motivated by wanting to help more students learn programming, but from the perspective of instructors already teaching in educational institutions. Ideally, an instructor would be able to understand the needs of all their students and respond accordingly to their progress throughout the course. However, there is often difficulty in maintaining an active class engagement and individual support for students as a result of a larger class size (O’Neill, 2012).

When we consider an introductory course to programming, students’ concerns could include misunderstanding specific programming concepts, applying multiple programming techniques simultaneously and not getting tailored feedback on their assignments. These concerns are what this project hopes to tackle so that students will be able to succeed with their introduction to programming.

1.1.1 Current state of programming education

When we took the programming class ENGG1111 at our first year in HKU, there were at least 273 enrolled students, as seen from the below graphic.

![Marks distribution bar chart](image-url)
At this scale, it becomes very difficult to maintain a deep understanding of the needs and progress of each individual student.

1.1.2 Existing applications of static code analysis
Currently, the use of metrics has been led by companies wanting to better improve their codebases. Woodruff (2019) detailed in a blog post why Instagram developed the tool LibCST, which coincidentally this project will make use of. He stated that the large codebase lends itself easily to bugs being introduced with no easy way of fixing them or identifying the source of the issue. However, there is not much existing work on applying these analyses for educational purposes, specifically in conjunction with curriculums used by instructors. As a result, this project will take a slightly different approach to Instagram and apply these techniques to smaller codebases written by students unfamiliar with programming.
1.2 Objective

The main objective of this project is to save time for instructors to gauge and follow-up on the understanding of their students in large programming classes.

This project aims to ameliorate the teaching process by creating an intelligent system that analyses the activity and solutions provided by students and attempts to determine what the students are struggling with, if at all. Due to the educational context, this system will also be closely tied to the curriculum taught by the instructor.

What we ultimately envision is a system that, upon submission of code by a student, will do the following:

1. Identify what concepts the student is struggling with
2. Suggest follow-up action plans to reinforce said concepts
3. Inform instructors on the progress of all their students as they work on various programming tasks.
2. Methodology

2.1 Scope

This project limits itself specifically to the programming language Python. This choice is informed by recent decisions made by the University of Hong Kong in their introductory course to new students (Department of Computer Science, n.d.). Due to the close collaboration with University of Hong Kong professors, we will be focusing on the curriculum of COMP1117.

2.1.1 Target Audience

Our target audience includes first-year university students based in Hong Kong with little to no experience with programming and relevant concepts as well as their instructors. This project will be most beneficial in large class settings, however it should still have positive effects even on smaller classes (20-30 students).

2.1.2 Potential Paths

Depending on collaboration with organizations that want to promote programming for high school students, we may include that to our scope.

2.2 Potential Deliverables

This project can take two different directions depending on the actual progress and feasibility in later stages. They are outlined in the following two subsections.

2.2.1. Research Paper

This route focuses on the machine learning aspect of this project, where the system can intelligently analyze user activity and identify struggling students. We anticipate the bulk of the effort to go into identifying the necessary data points to collect and a simple yet user-friendly interface dedicated to the narrow scope of identifying struggling students and providing help.
2.2.2. Feature-rich Platform
This route is to be taken only if the research route is considered infeasible with the current resources. This route focuses on creating a feature rich application where the machine learning aspect of the project is not yet fully realized but relevant heuristics are used to make the decisions.

Here, the system will provide many additional features such as action plans, supplementary materials, etc. The main weakness of this would be that the suggestions from the platform would not be as personalised.

2.3 Data
This project will need large volumes of data in the form of code snippets of varying qualities that achieves the same functionality.

2.3.1 Data Collection
For now, all of the source code used for analysis are manually generated. In the future, the project will diversify its sources to include code written by actual programming beginners, and from online sources.

Another possible candidate for data collection is the user log activity in a programming environment where we can define certain evaluation metrics, such as:

1. How many times a user interacts with the programming environment to complete a task
2. Time spent on a programming task
3. Time spent on a specific concept within a programming task
4. The sequence of interactions within a period of time
2.3.2 Data Labelling

Each code sample will be labeled as “good” or “bad”. The challenge will be to specify what it means for a code sample to be “good” or “bad”. One possible way is to simply label certain parts of the code that we identify as undesirable based on defined code quality metrics.

```
def filter_list(l):
    to_remove = []
    for i in range(len(l)):
        if isinstance(l[i], str):
            to_remove.append(l[i])
    for i in range(len(to_remove)):
        l.remove(to_remove[i])
    return l
```

Unnecessary variable

Two for loops

2.3.3 Domain Models

In the field of adaptive learning, there is a narrative approach to personalisation where concepts in a particular domain are related and woven into a coherent narrative for learners to navigate and understand (Conlan et al., 2013).

There, concepts are expressed as domain models that fulfil the objectives of a domain.

We will borrow that idea of domain models into this project. To illustrate how this can manifest in our project, we can consider the course COMP1117 Computer Programming in Python course as the domain.

The course objectives of COMP1117 from its course description is:

**CLO1 Computational Mind**

Able to identify possible solutions for problems based on computer programs

**CLO2 Program implementation**

Able to implement solutions for problems using Python
CLO3 Program Comprehension

Able to understand programs written by others and participate in larger scale system implementation

Given the course, let's take a concept: Basic Data Structures, and try to express it in a domain model:

1. Prerequisite concept: DataTypes
2. Objective: CLO1 Computational Mind
3. Resource: <Insert Resource>

It is a rudimentary example but we can expand this approach to include all relevant concepts in a course, and slowly find all relationships and properties for the concepts.

Where this project differs from the narrative approach is that we are not aiming to create an entire personalised adaptive learning path for each individual student. Rather, we are borrowing the ideas of relating concepts in a model and expressing them in domain models from this approach to help us in identifying points of difficulty in a pre-defined "narrative" and making it easier to follow up on struggling students.

2.5 Analysis

2.5.1 Syntax Trees

The primary means of analyzing source code will be by using Python libraries such as the ast and libcst modules to convert Python code into syntax trees.

There are two main types of syntax trees:

- Concrete Syntax Tree (CST)
- Abstract Syntax Tree (AST)

CSTs (Also known as parse trees) are a “concrete” representation of a given source code. It is a direct lossless mapping of an input source code. We will be using the Python module LibCST to generate CSTs. Below is an example of how libcst converts a Python expression into a CST.
Abstract Syntax Trees (ASTs) are a more simplified representation of an input source code, where details that do not contribute to program logic (whitespace, comments, etc.) are ignored. We will be using Python’s built in `ast` module for generating ASTs.
We can compress our parse tree and still maintain the correct nesting of nodes/tokens/terminals.

Compressing allows us to avoid redundancy.

An AST represents the exact same expression as our parse tree, but abstracts away from the concrete syntax.

Source: (Joshi, 2017)
Consider the following two code snippets. They both achieve the same functionality (filtering all strings in a given list). Yet, when we look at their ASTs, we can see a clear difference in code complexity.

Example 1

```python
def filter_list(list):
    return [i for i in list if not isinstance(i, str)]

filter_list([1, 2, 3, 'a', 't', 5])
```

Generated by Python AST Visualizer
Example 2

def filter_list(list):
    to_remove = []
    for i in range(len(list)):
        if isinstance(list[i], str):
            to_remove.append(list[i])

    for i in range(len(to_remove)):
        list.remove(to_remove[i])
    return list

filter_list([1, 2, 3, 'a', 't', 5])

Generated with Python AST Visualizer
2.5.2 Code Quality Metrics

The following code quality metrics will also be used as features for training the machine learning algorithms.

There are at least three main categories for code quality metrics:

1. Complexity
2. Redundancy
3. Style

**Complexity** can be measured by analyzing the cyclomatic complexity of a given program. It is a measurement of the number of decisions or paths a program takes (Malhotra et al., 2015). This analysis can be implemented with the Python module *Radon*.

**Redundancy** can be measured by analyzing the entropy of a source code’s AST. Entropy is a measurement of how “random looking” a given source code is. It can give an idea of how much repetition and hence, redundant code, there is. (Tóth, V., 2018 June 21).

**Style** metrics refers to how well a given source code adheres to coding style standards. In this project, this would be whether a source code is “Pythonic” with standards written by the Python Software Foundation, namely PEP8 (2013).
3. **Schedule and Milestones**

Considering our approach to developing this project, we can only have a well-defined timeline up to the point where we enter one of the two aforementioned directions in section 2.1

3.1 **Estimated Timeline**

<table>
<thead>
<tr>
<th>Date</th>
<th>Scheduled work / Deliverables</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 Sept 2020</td>
<td>Test Prototype</td>
</tr>
<tr>
<td>4 Oct 2020</td>
<td><em>Phase 1 Deliverables: Project plan &amp; webpage</em></td>
</tr>
<tr>
<td>Oct - Nov 2020</td>
<td>Manually labelling and analyzing data sets</td>
</tr>
<tr>
<td>Oct - Dec 2020</td>
<td>Data Collection</td>
</tr>
<tr>
<td>1 Jan 2021</td>
<td>Determine feasibility of research paper deliverable</td>
</tr>
<tr>
<td>11-15 Jan 2021</td>
<td><em>First Presentation</em></td>
</tr>
<tr>
<td>24 Jan 2021</td>
<td><em>Phase 2 Deliverables: Preliminary Implementation &amp; Detailed Interim Report</em></td>
</tr>
<tr>
<td>18 Apr 2021</td>
<td><em>Phase 3 Deliverables: Finalized tested implementation &amp; Final report</em></td>
</tr>
<tr>
<td>19-23 April 2021</td>
<td><em>Final Presentation</em></td>
</tr>
<tr>
<td>4 May 2021</td>
<td><em>Project exhibition</em></td>
</tr>
</tbody>
</table>
### 3.2 Division of Labor

<table>
<thead>
<tr>
<th>Roles</th>
<th>RAI Adarsh</th>
<th>N.R. Ajit Krishna</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontend</td>
<td>Support</td>
<td>Lead</td>
</tr>
<tr>
<td>Backend</td>
<td>Support</td>
<td>Lead</td>
</tr>
<tr>
<td>Infrastructure</td>
<td>Support</td>
<td>Lead</td>
</tr>
<tr>
<td>Static Code Analysis</td>
<td>Lead</td>
<td>Support</td>
</tr>
<tr>
<td>Data Engineering</td>
<td>Lead</td>
<td>Support</td>
</tr>
<tr>
<td>Data Science</td>
<td>Lead</td>
<td>Support</td>
</tr>
</tbody>
</table>
4. Bibliography


https://medium.com/basecs/leveling-up-ones-parsing-game-with-asts-d7a6fc2400ff

LibCST. (n.d.). *Visualised graph of LibCST’s parsed nodes* [Online Image]. Instagram.

https://libcst.readthedocs.io/en/latest/_images/graphviz-d27e3495fa9bb130d76879db599060e8039a9fc5.png


https://medium.com/mindsoft/abstract-syntax-tree-entropy-e2047af4906f


https://slacker.ro/2019/08/16/static-analysis-at-scale-an-instagram-story/