ABSTRACT

DROSOS, IAN ZACHARIAH. HappyFace: Identifying and Predicting Frustrating Learning Obstacles at Scale. (Under the direction of Christopher Parnin.)

Unnecessary obstacles limit learning in cognitive-complex domains such as computer programming. When accompanied by a lack of appropriate feedback mechanisms, novice programmers can experience frustration and disengage from the learning experience. In large-scale educational settings, the struggles of learners are often invisible to the learning infrastructure and learners have limited ability to seek help. In this paper, we perform a large-scale collection of code snippets from a learning platform, Python Tutor, and collect a frustration rating through a light-weight feedback mechanism. We then devise a technique that is able to automatically identify sources of frustration based on participants labeling their frustration levels. We found 3 factors that best predicted novice programmers' frustration state: syntax errors, using niche language features, and understanding code with high-complexity. Additionally, we found evidence that we could predict sources of frustration. Based on these results, we believe an embedded feedback mechanism can lead to future intervention systems.

HappyFace: Identifying and Predicting Frustrating Learning Obstacles at Scale

by
Ian Zachariah Drosos

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, North Carolina

2017

APPROVED BY:

Tiffany Barnes	Kathryn Stolee

Christopher Parnin
Chair of Advisory Committee

DEDICATION

To Angela.

BIOGRAPHY

The author is a Masters of Science student in Computer Science at North Carolina State University. He will be continuing on as a PhD student in Cognitive Science at the University of California, San Diego.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Parnin, for his guidance.

TABLE OF CONTENTS

LIST OF TABLES . . . . . **vii**

LIST OF FIGURES . . . . . **viii**

Chapter 1 Introduction **1**
1.1 Motivating Example 3

Chapter 2 The HappyFace System **5**
2.1 Design Goals 6
2.1.1 Brief and Minimalist: 6
2.1.2 Problem and Language Agnostic: 7
2.2 Implementation 9
2.2.1 Affect Survey 9
2.2.2 Data Collection 9
2.2.3 Data-set Extraction 9
2.2.4 Code Analysis 9
2.2.5 Extended Feedback 10

Chapter 3 Online Deployment Studies **11**
3.1 Research Questions 11
3.2 Surveys 12
3.3 Participants 12
3.4 Analysis 13
3.4.1 Data Cleaning 13
3.4.2 System Refinement 13

Chapter 4 Results **14**
4.1 Research Question 1 14
4.2 Research Question 2 15
4.3 Research Question 3 17

Chapter 5 Discussion **19**
5.1 Implications 19
5.1.1 Interventions 19
5.1.2 Language Design 20
5.1.3 Feedback Mechanisms 20
5.1.4 Prediction of Categories 21
5.2 Challenges 22
5.2.1 Integrations 22
5.2.2 Extraction of Informative Features 22
5.2.3 Participation 23
5.3 Limitations 23
5.4 Related Work 24

5.4.1 Lightweight and Static Analysis of Code at Scale 24
5.4.2 Using the Crowd 25

Chapter 6 Conclusion **26**

BIBLIOGRAPHY **27**

LIST OF TABLES

Table 2.1 Extracted Features 8
Table 2.2 Selection Categories 8

Table 4.1 Odds Ratio 16

LIST OF FIGURES

Figure 1.1 Code example on Python Tutor with HappyFace affect survey 3

Figure 2.1 HappyFace Affect Survey 5

Figure 4.1 HappyFace vote distribution 15

Figure 5.1 Highlighted feedback to learner 20

CHAPTER

1

INTRODUCTION

Frustrating obstacles during programming [FOR15] can cause a learner to disconnect from the learning process and can even lower their self efficacy. In massive open online courses (MOOCs) and large-scale learning environments, such as Python Tutor, frustration can come from course difficulty, lack of support, lack of digital or learning skills, and failed expectations [OB14]. When students cannot overcome these obstacles, they often dropout as a result [Con13], with only 15% of learners completing the course [Jor]. There is a need to discover these frustrations felt by learners so that tools that provide interventions to assist learners in their studies can be effective.

In large-scale learning systems, traditional mechanisms for overcoming learning difficulties, such as peer feedback and tutor instruction, do not scale well. This results in having limited context about the problems being solved, limited knowledge about the level of learner, and a lack of insight into a learner's potential misconceptions. Learners from all around the world are editing, running, and visualizing code from introductory courses or may be trying to debug a more advanced programming assignment. While coding, learners inevitably run into frustrating events ranging from syntax errors to misunderstanding features of the code language. These frustrations can limit the effectiveness of the platforms the learners are using.

Several platforms have tried to help novice programmers at scale. For example, HelpMeOut suggests solutions to errors it collected from a crowd of programmers who had a similar error in

order to help inexperienced programmers understand and correct errors [BH10]. While crowd-based strategies can be effective for resolving certain types of obstacles, maintaining incentives and quality of a crowd-based systems can be difficult, especially for rapidly evolving technologies and for obstacles beyond interpretating an error message. Intelligent tutoring systems (ITS) have been used to automatically assist learners by providing adaptive and individualized feedback about a learner's work. In particular, JavaTutor has been used to predict how a learner's affect changes in response to questions from the automated tutor [Vai16]. Unfortunately, obtaining affect data required for prediction requires several physical sensors. For MOOCs, the population of learners is numerous so that it would be beneficial to not rely on each student having a webcam, Kinect, or any other sensor. Further, intelligent tutors rely on pre-written assignments, requiring structure that usually only exists in courses.

In this paper we describe HappyFace, a system that allows a novice programmer to annotate their learning experience and then automatically infer what type of learning obstacle they are encountering. Even with low participation rates, HappyFace can discover types of frustrating experiences and predict categories of learner frustration. Our system is not restricted by pre-written assignments and relies on simple input from the learner. HappyFace is composed of four components:

- **Affect Survey**. Embedded in a host system, HappyFace collects the current affect of a learner when the learner selects the face that most represents their current state.
- **Feature Extraction**. HappyFace automatically analyzes the code the learner was working on when they reported their current affect. This analysis is done via static program analysis through the parsing of abstract syntax trees to extract features, and stylometric analysis through the extraction of features relating to the style of the code.
- **Correlation**. HappyFace correlates the extracted features with the affect vote to discover frustrating features.
- **Prediction**. HappyFace predicts the type of learning obstacle a learner is facing when they report their affect.

In a 2.5-month online study, we found that learners were willing to provide feedback about their current emotional state during a learning experience. We implemented HappyFace into an existing Web-based program visualization tool called Python Tutor [Guo13]. Python Tutor lets users to write code inside their browser and then visualize each execution step of their code, allowing the user to better understand what is going on inside their code. 2,385 Python users elected to use HappyFace during their coding session, 2.35% of Python Tutor Python users during the time span of the study.

From learner annotations we found that syntax errors, features that increase code complexity, and niche language features, such as the Python Slice Operator and Globals, correlated with frustration in learners. After an odds ratio analysis we found that syntax and indentation errors doubled the odds that a learner is frustrated. We also found that each occurrence of boolean comparators, library imports, and use of Globals increased the odds of frustration by 1.5 to 1.7 times. In a follow-up study, we found that annotations could be used to help identify and predict frustrating experiences. Using Logistic Regression, we predicted when learners did not understand their code with 80% precision.

Despite HappyFaces affect survey being a simple 5-choice survey, the data we collected yielded an abundance of useful data without using more complex methods of data collection such as physical sensors. Further we were able to use this data to discover features causing frustration with only lightweight static program analysis. Based on these results, we believe the approaches used in HappyFace can be used to support future learning interventions.

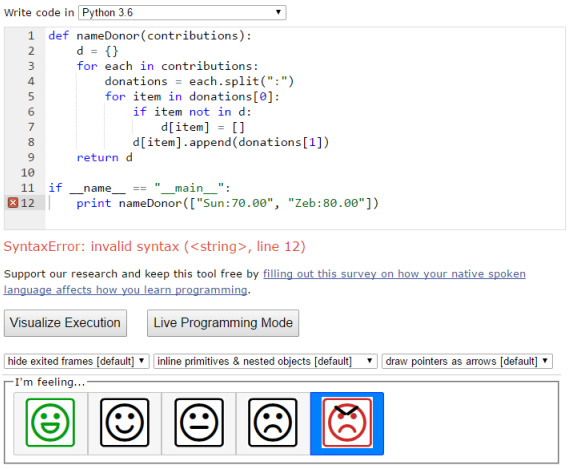

Figure 1.1 Code example on Python Tutor with HappyFace affect survey

1.1 Motivating Example

For example, take a professor who is teaching an introductory programming MOOC and has just given the 1000+ students in the course a new programming assignment involving splitting strings and printing results. Since this is an introductory course, many of these students are new to programming and thus inevitably run into various issues completing the assignment. In previous semesters,

the professor had to ask the TAs to obtain sources of frustration felt by the students and causes of errors in student code. The TAs attempted to obtain this information through interviews of the students via forum threads, but this requires a lot of manual effort. Also, while the TAs are able to find some categories of frustration, most of the students do not yet have the knowledge or experience to accurately convey the issues they faced in the assignment.

Instead, the professor has implemented HappyFace this semester so students can report when they are frustrated during an assignment. One student, Ada, is using Python Tutor for the assignment but when she executes her program she receives a SyntaxError (Fig 1.1). Ada becomes frustrated as she does not understand why she is getting this error as she is not aware of certain syntactic rules in Python 3.

Ada clicks a face that represents frustration on the HappyFace survey, located under the code editor, when she receives her error. HappyFace can then analyze her code and report valuable information to the professor that show some of the students are frustrated when dealing with SyntaxErrors. The professor can now adjust the lecture to include strategies to solving this error, alleviating any unnecessary frustration that may be felt by students like Ada attempting similar work.

CHAPTER

2

THE HAPPYFACE SYSTEM

HappyFace allows a learner to annotate their learning experience. We embed a survey to analyze these annotations by creating an abstract syntax tree (AST) from code snippets and process the code for language features and other stylometric features, such as types of whitespace, code complexity, and the standard deviation of line length. These scores are run through randomized logistic regression for feature selection to identify potentially frustrating learning experiences. HappyFace is then extended to allow for participants to select additional reasons for frustration. Using our code metrics, we then predict the problem categories a code snippet will fall into.

Figure 2.1 HappyFace Affect Survey

2.1 Design Goals

To define HappyFace's design goals, we examined multiple feedback surveys in use today. Further, we tested a few prototype designs to assess which style of survey implementation would elicit the most responses from learners on Python Tutor. Lastly, in order to gather and analyze data provided by learners, HappyFace must be adopted by learning frameworks. Based on these observations, we adhered to these design goals:

2.1.1 Brief and Minimalist:

To enable participation, the design must be brief and minimalist. If the survey component of HappyFace clashes with the learning platform it is embedded, it will serve as an unwanted distraction to the learning process. Thus, HappyFace is designed to be as brief as possible so a learner can use the platform as intended. HappyFace is also designed to be minimal. It is an in-line survey that takes only the amount of visual space it requires. When a learner decides to cast a frustration vote, any further parts of the survey expand to allow the learner to provide more information and then collapse once a learner has finished answering the survey. This allows HappyFace to provide a feedback tool that minimally disrupts the appearance of the learning platform it is embedded and allows learners to quickly return to their learning workflow.

The first prototype of HappyFace included a slider that manipulated the color and smile of a smiley face above the slider. When the user neared the maximum frustration rating the face became red and had a frown, when the user neared the minimum frustration rating the face became green and had a smile. This first prototype also had a free-form text area to allow the learner to explain why they were or were not frustrated in their own words. Once the frustration score was selected, the user could then press a button to submit their frustration score, frustration reason, and code snippet. This prototype survey was hosted on Python Tutor for a week but did not receive a satisfiable amount of votes. We attribute this to the fact that it took multiple steps to answer the survey, potentially adding frustration to an already frustrated learner. Thus, we sought to redesign the interface.

To implement this design goal, we drew inspiration from Bieri et al. work on the self-assessment of pain severity [DBZ90]. The authors asked children to look at a set of faces depicting different levels of pain and order them from no pain to the most pain. The aim of the authors was to create a scale with "minimal cognitive demands". The authors found that their face-based pain scale required little direction of the child as they are simply told to point to the face that matches the amount of pain they feel. The quick and simple face-based pain scale showed that it could be used reliably and validly for the self-reporting of pain in children.

Thus, the HappyFace survey became a simple five button design that took inspiration from the pain scales used by medical professionals (Fig **??**). The five buttons had faces with a range of emotions, from happy to angry. The only instruction provided to the learner is text saying "I'm feeling...", implying to the learner they should select the face that best represents their current feeling. Self-reporting of frustration on HappyFace is also very quick, one button click on the face that most represents the learner's frustration is all that is needed from the learner. Once a learner clicks a face, HappyFace collects the rest of the relevant data automatically.

2.1.2 Problem and Language Agnostic:

To ease adoption, the design of the system must be able to accommodate the many different programming problems as well as programming languages that a learning platform may support. For example, several approaches for automated feedback of student assignments require that the problem be known ahead of time [Vai16] and that it is accompanied by set of test cases [DKRR16]. Further, these techniques require heavier analysis, such as symbolic execution, which can work appropriately on few student assignments, but may not scale well when needing to do an analysis of millions of code snippets.

To implement this design goal, we drew inspiration from Caliskan-Islam et al. work on code stylometry [ACIG15]. Caliskan-Islam used submissions to a Google Code Jam as a dataset and extracted stylometric features from the code and attempted to de-anonymize programmers based on these features. Machine learning methods were run against the extracted feature sets and the paper's authors found that they could correctly attribute the author of a piece of source code at high accuracy (>90%). This result gave us confidence that properties extracted from the AST of the code could provide sufficient features without a more complex representation of problem state.

Thus, the analysis component of HappyFace was based on light-weight static analysis of the AST, using the same features as Caliskan-Islam et al. [ACIG15]. For example, some styles of programming, such as very long and complex lines of code, could frustrate a learner who is having difficulty understanding the meaning of a line of code. While the current analysis by HappyFace parses ASTs from Python code, the structure of most languages contain similar features like if statements, loops, and functions. Because of this, we maintain that HappyFace can be implemented for almost any language. Further, stylometric features, such as average line length, are language-agnostic.

Table 2.1 Extracted Features

Feature	**Notes**
Frustration score	given by user
AST nodes	defined by the Python standard library "ast"
Stylometric features	usage of python keywords (keyword and builtin libraries), line count, average length of lines, standard deviation and variance of line length, white space count, types of white space used, underscore usage, comment count, how many lines start with a comment or tab character, empty lines
Wordgrams	Wordgrams extracted using regex to detect commonly used words in the code snippets

Table 2.2 Selection Categories

Category	**Description**
Happy categories	"I fixed an error", "My output is right", "My code works", "I understand this code", "Other (user input required)"
Frustrated categories	"I get an error", "My output is wrong", "Some code is not running", "I don't understand this code", "My variable has the wrong value", "My loop doesnâĂŹt iterate correctly", "If statement does not work", "Other (user input required)"

2.2 Implementation

2.2.1 Affect Survey

HappyFace has an inline survey that can be embedded into a system to collect frustration data from its users. The current version of HappyFace was implemented on PythonTutor.com [Guo], a website that visualizes code execution. A user can use the HappyFace survey during their workflow on the website. When a user selects a face that best represents their current feeling, a vote is sent to checkbox.io [Par], a research survey site. This vote is comprised of data that includes their frustration score (1-5, happy to frustrated) and the code snippet they are programming on Python Tutor. In an extension of the affect survey, the user is also presented with a choice of tags they can select to better describe their vote in the form of a frustration class. This data is compiled in a database and extracted to a json file for analysis.

2.2.2 Data Collection

HappyFace was embedded on Python Tutor's visualize page under the code editor and "Visualize Execution" button (Fig 1.1). While users of the site were programming on this page they could select a face that most represents their current affect. Once a face was selected, HappyFace collects the code the user has written along with the affect rating and stores the vote in a database for analysis. The survey is constantly available and allows for multiple votes over the user's entire coding process. For the extended survey, after a user selects an affect they are presented with several categories to optionally annotate their affect with.

2.2.3 Data-set Extraction

The compiled data is then processed in Python for analysis. Each vote is run through a filter that removes invalid votes. Votes that are missing data (frustration score), votes that were cast missing code snippets, and spam votes (a multitude of votes cast by the same person in a short period of time) are removed.

2.2.4 Code Analysis

From the resulting data-set we process the feedback by creating an abstract syntax tree (AST) from code snippets attached to a frustration score. If an AST cannot be parsed from a snippet, the error produced by the compilation of code is caught and added as a feature. The frequencies of features, such as Set nodes, Str nodes, and If nodes, are compiled with the frustration score of the feedback.

This AST is traversed and, as each node name is visited, the node is counted as a feature. After AST features are extracted, the code snippet is parsed for stylometric and word frequency features. These extracted features are listed in Table 2.1 and include feature name, such as AST nodes, along with a description of the feature. The majority of stylometric features we chose to extract are derived from Caliskan-Islam's code stylometry feature set [ACIG15].

2.2.5 Extended Feedback

For a follow-up study, we extended HappyFace to allow for the annotation of the frustration felt by the learner. We presented the learner with several categories of frustration that the learner may feel their frustration can be placed. These categories were created after manually inspecting code reported by frustrated users and hypothesizing the issue faced. Also included is an Other category that elicits a free-form response from the user. These free-form responses can be used to tune categories presented to the learners once common categories become apparent. A list of these categories, separated into "Happy" and "Frustrated", are listed in Table 2.2.

CHAPTER

3

ONLINE DEPLOYMENT STUDIES

We developed a lightweight feedback mechanism called HappyFace that allows for the collection of learning experiences from programming learners. We then run two studies to gather learner feedback and use this feedback to discover frustrating factors in programming.

3.1 Research Questions

We ran a study with HappyFace, collecting frustration and code data from learners using Python-Tutor.com, a web-based automated program visualization tool. With the data collected we then conduct an empirical evaluation to answer the following research questions.

- **RQ1:** Are learners willing to provide feedback about their current emotional state during a learning experience?
- **RQ2:** Can we identify features of code that are related to frustration?
- **RQ3:** Can these features predict categories of problems faced when programming?

3.2 Surveys

For this paper we ran two implementations of HappyFace to gather frustration data from learners in two studies. The survey used in our first study included the face vote (Happy to Frustrated faces) and the extraction of the code the participant is working on. After finding several features that correlated with frustration we decided to run a follow-up study to see if these code features can predict the categories of the problems learners were facing. So, for the follow-up study we extended HappyFace to included a second section that allowed the participant to optionally select a problem category for the purpose of predicting user chosen categories using the extracted features present in learner code. For our two implementations of HappyFace we integrated our survey with Python Tutor for 2.5 months for the first study and 2 months for our follow-up study. Each survey collected thousands of votes from learners using Python Tutor which were then filtered down to extract usable data.

3.3 Participants

Any user of Python Tutor could then self-report their frustration level and, for the extended survey, annotate their frustration. Users of Python Tutor was a compelling population for discovering frustrating features of code since the user base is likely to be: (1) learners who are new to a language attempting to code small samples and (2) learners using the visualization features of Python Tutor to better understand the code snippets they are working on as one major source of users on Python Tutor are hundreds of thousands of learners taking introductory programming courses on MOOCs like Coursera, edX, and Udacity [Guo13]. Since it is unlikely experts with full featured IDEs and experience dealing with common frustrations of code are using Python Tutor, we can attribute frustrations reported through HappyFace to frustrations felt by learners due to features of the code they are working with. Frustrations differ between the coding frameworks being used as a fully featured IDE may have features that eliminate common frustrations of developers, as well as differing between the experience levels of the users being frustrated. A less experienced programmer will find certain features more frustrating than an experienced developer who has discovered solutions to common frustrating problems. Participants voluntarily responded to the HappyFace survey without any prompts or monetary rewards.

3.4 Analysis

3.4.1 Data Cleaning

An initial 7,450 valid votes were received for our first study. We then filtered out any reporting of frustration that was done before the user had started writing code, as we cannot extract code features without this data. After this pruning, we had 1,013 votes with code in Python 2 and 797 votes with code in Python 3. The resultant data contained 83 and 82 different AST node types, 1,025 and 805 word grams, and stylometric features as defined in Table I for Python 2 and 3 respectively. Despite the syntactic rules of both versions of Python being mostly similar, the changes done in Python 3 changed some features already existing in Python 2 as well as added new features, creating incompatibilities between the two versions. Because of these differences, we needed to separate the analysis of Python 2 and 3 code.

3.4.2 System Refinement

After we ran our first study we extend HappyFace's affect survey to take in an annotation by the learner on why they are frustrated. Common reasons for frustration. or non-frustration, are presented to the learner after a face vote. These categories range from the (mis)understanding of code to receiving or fixing of errors. Rather than correlate frustrating features of code, the frustration annotations the learner provides through the extended survey allows us to predict the categories of frustration a learner will feel given a particular set of code features. An initial 909 Python 2 and 1,304 Python 3 votes with code snippets were left after filtering thousands of votes cast by learners for the second study.

CHAPTER

4

RESULTS

4.1 Research Question 1

Are learners willing to provide feedback about their current emotional state during a learning experience?

Our first research question is to investigate the learners' willingness to inform a system on their current level of frustration while they are in the process of coding. Our goal was to make a survey that could quickly be taken without much interruption to the learner's work flow. If we failed learners would be unlikely to give us feedback on their frustration level, so a usable and streamlined survey was presented to learners for their feedback. Further, creating a survey that elicits feedback from the learner removes and necessity for more complex ways to gather frustration (physical reactions, typing characteristics, etc).

During the time period we ran our first study (April 2nd - June 14th), Python Tutor received 101,044 unique visitors who executed Python 2 or 3 code. Of these visitors, 2,385 voted on the HappyFace affect survey and gave us frustration values and code snippets. This represents a 2.36% participation rate for Python learners on Python Tutor. Our affect survey is meant to be unobtrusive below the coding interface and was completely optional for Python Tutor users to participate, with no reward offered. Despite the affect survey giving users a lack of motivation to report their

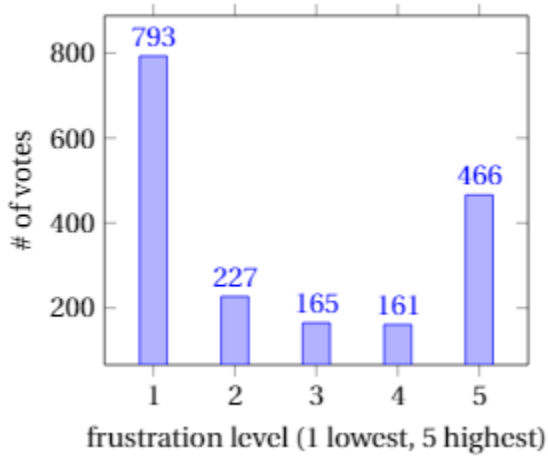

Figure 4.1 HappyFace vote distribution

frustration, we received enough data from participants for our analysis of frustrating code features. This showed that learners were willing to give us feedback on their current emotional state despite being deep into the coding process and potentially being frustrated.

4.2 Research Question 2

Can we identify features of code that are related to frustration?

One of our interests is in the correlation between features of programming languages, in this case Python, with frustration felt by learners. With learner frustration feedback along with features of the code snippets provided through our feedback mechanism we searched for supported features through randomized logistic regression feature selection.

Features were run through Randomized Logistic Regression for discovery of supported features. For Python 2 we found that use of Sets, Globals, Strs, Pow, Slices, and occurrences of SyntaxErrors were selected from the 83 extracted AST features. For Python 3 we found that use of Boolean comparators (is not, !, and !=), import statements, with statements, If statements, Load expressions, and occurrences of SyntaxErrors were selected from the 82 extracted AST features. The strongest correlating feature with frustration was SyntaxErrors. For stylometric features, the average length of the lines of code as well as the usage of python keywords were found to be supported, with average length having the strongest correlation with frustration. While many wordgrams were selected, the strongest correlations were already detected through AST and stylometric features. The use of for, if,

Table 4.1 Odds Ratio

Python 2	
Feature	**Odds Ratio**
SyntaxError	2.28
Str	1.01
Pow	0.59
Slice	1.10
Global	1.6
averageLength	1.02
uniqueKeywords	1.04
Python 3	
Feature	**Odds Ratio**
SyntaxError	2.17
For	1.11
If	1.05
Import	1.75
IsNot	1.78
Not	1.52
NotEq	1.10
averageLength	1.03
uniqueKeywords	1.09
False	1.47
IndexError	2.49
isinstance	2.55

and other python keywords correlated stronger than any other selected wordgram.

One example of a frustrating feature is the use of global variables. Python has unique rules for global variables [Fou]: If a variable is assigned a value inside of a function, it is considered a local variable unless the variable is declared as global. If the global variable is only referenced inside a function, this declaration is not needed. Not knowing that globals need to be explicitly declared in Python can cause frustration in learners when it violates their expectation. While the goal of this rule may have been to remove clutter caused by multiple global declarations, it also inserted a frustrating obstacle for learners.

As part of our analysis we extracted the odds ratios of our selected features. The odds ratio "is a measure of association between an exposure and an outcome" [Szu10]. That is, the occurrence of a specific feature will affect the odds of a certain outcome. In the case of HappyFace, the existence of certain AST and stylometric features will increase, decrease, or have no effect on the odds that the learner will be frustrated. If the odds ratio of an independent variable is greater than 1.00, an increase of a unit of the variable will increase the odds that the learner is frustrated. Inversely, when the odds ratio of an independent variable is less than 1.00, an increase in that variable decreases the odds of frustration. The larger the distance from 1.00, or neutral with no effect on odds, the larger effect on the odds of frustration. For example, if the odds ratio of SyntaxError is 2.00, the odds that a learner will be frustrated when a SyntaxError exists in their code is doubled (2.00/1.00). Table 4.1 lists the intercept and odds ratio of each selected feature for Python 2 and 3 we extracted for the first study. For Python 2, the feature that increased the odds of frustration was SyntaxErrors with an odds ratio of 2.28 For Python 3, SyntaxErrors also had a comparatively high odds ratio of 2.17. The highest odds ratio for Python 3 was the use of the isinstance function at 2.56.

4.3 Research Question 3

Can these features predict categories of problems faced when programming?

After extending our survey component to include annotations for the learner to select as a reason for their current emotional state, we ran a second experimental survey. The annotations presented to learners were categories we hypothesised as potential causes of frustration and happiness during the coding process. We also included another field so learners could write their own annotations to help us find frustrating experiences we might have missed. The extension only added a second, but optional, button click to select an annotation which did not disrupt the learners' willingness to provide feedback.

Since the main purpose of the extended survey was to discover if we could predict the category of the feature causing frustration, the votes of interests are the optional annotations the learners

could provide after casting a face vote. For votes with Python 2 code we received 215 reason votes, 23.65% of total votes. For votes with Python 3 code we received 303 reason votes, 23.24% of total votes. The most common reason for being frustrated was **"I get an error"** with 32 occurrences for Python 2 votes and 47 for Python 3 votes. Other commonly chosen reasons were **"My output is wrong"** and **"I don't understand this code"**, both showing that misunderstanding code is a cause for frustration in learners. The most common reason for being happy was **"I understand this code"** with 38 occurrences for Python 2 and 64 for Python 3. Other commonly chosen reasons for not being frustrated are **"My code works"** and **"I fixed an error"**. This was an expected result as the reasons that learners are not frustrated are antithetical to the reasons that learners are frustrated. We also allowed the learner to input their own reason in a free-form text box by selecting the "Other" reason.

Using Logistic Regression to predict the learner-selected category based on features extracted from the user's code snippets, we were able to obtain a precision, recall, and f1-scores for predicting categories of frustration selected by learners. For Python 2 we were able to predict the choice of "I don't understand this code" with a precision of 0.80, recall of 0.33, and f1-score of 0.47 with a support of 12 out of 41 annotations. "I get an error" was predicted with precision of 0.39, recall of 0.85, and f1-score of 0.54 with support of 13 of 41. For Python 3 we were able to predict the choice of "I don't understand this code" with a precision, recall, and f1-score of 0.30 with a support of 10 out of 46 annotations. Selection of "I get an error" was predicted with precision of 0.52, recall of 0.76, and f1-score of 0.62 with support of 17 of 46. The other frustration categories could not be predicted as there was not enough support to do so.

CHAPTER

5

DISCUSSION

We discuss several implications and challenges for future research on learner frustration.

5.1 Implications

5.1.1 Interventions

We believe HappyFace can be used to inform intervention systems, both automatic and manual. In a large-scale learning framework like a MOOC, a frustrated student can report their frustration. HappyFace then can predict the category of frustration felt by a learner working on the particular code snippet. Interventions to this category of frustration can then be automatically presented to the learner.

For example, when HappyFace predicts that the frustration of the learner is from improper indentation, HappyFace can offer to perform "automatic repair" on the learner's code in order to fix the frustrating code. In the case of our learner Ada from the motivational example, HappyFace can inform intervention systems on when she is facing a frustrating SyntaxError. The intervention system can then highlight a suggested fix, for Ada's SyntaxError the system can highlight missing the parenthesis in her print statement (Fig 5.1). HappyFace can also help in a traditional setting where automatic tools are not available to help the learner. In courses that have human assistants,

HappyFace can inform them when a learner is facing frustrations along with the likely features causing frustration. Assistants more suited to helping learners understand code may prioritize helping frustrated learners whose code is difficult to understand.

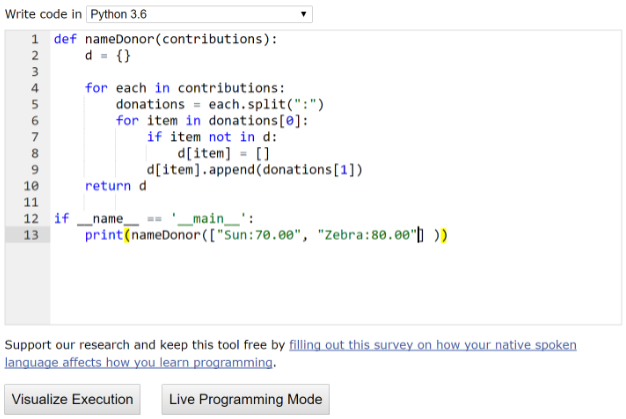

Figure 5.1 Highlighted feedback to learner

5.1.2 Language Design

Our findings support the need for better feature design in programming languages. Our analysis reinforced the common knowledge that SyntaxErrors are a frustrating obstacle that learners face, and potentially unnecessary in languages targeting novice programmers. Further, code complexity can be monitored and even prevented by the language when a learner's code starts to get too complex, frustrating learners when they do not understand their code. Lastly, certain features and the rules to use them may cause frustration in their implementation, such as global variable usage in Python mentioned in RQ2. HappyFace can discover these frustrating experiences in a programming language so that better designs can be created to remove barriers to programming education and aid learners in their understanding.

5.1.3 Feedback Mechanisms

With HappyFace we created a light-weight feedback mechanism that collects the affect of the voter and data describing the current actions of the learner. The design of HappyFace is purposefully simple, a single button click is all that is required of the voter. We believe this increases participation rate by learners by lowering the barrier to providing information to the surveys stakeholders. This

also lowers the impact on the voter by minimizing the interruption caused by the act of providing feedback. HappyFace shows you can gather meaningful data for discovering results, such as what features in Python cause learner frustration, by leveraging the actions of the voter while only requiring the voter to self-report their affect.

Carter et al. has investigated the automated detection of when a programmer is having difficulty or is "stuck" [JC10]. Researchers found that it was not efficient to allow programmers to manually change their status to stuck, as developers delay asking for help when they need it. Instead detecting difficulty by monitoring developers can inform helpful interventions earlier. While predicting affect is an important aspect in getting learners help, we are also interested in detecting why learners need help.

HappyFace allows for the discovery of issue-categories so that interventions can be done to alleviate the frustration of learners at both the education level and the learning platform level. While learning platforms do usually take in feedback and suggestions for improvement, most users may not know what their actual issue is and what improvements need to be made to deal with it. HappyFace uses the self-reported level of frustration and correlates this score with the learners' code to predict the category of the issue that is causing frustration. This reveals issue categories from code complexity and errors to missing features and bugs of the learning framework which teachers and platform developers can use to provide interventions that ease learner frustrations. For example, during our second study a few learners stated that they were receiving an indentation error as the reason for their frustration. This could inform the learning platform that it need to extend its capabilities and add auto-indentation or PEP-8 warnings to assist a learner in properly indenting their Python code.

5.1.4 Prediction of Categories

We were able to predict when a learner did not understand the code because AST feature extraction give us metrics on how complex the code is, which can create a difficulty in understanding the code for learners. However, while ASTs can be useful for some categories, some AST features cannot give us insight on expected behaviors or values and require more analysis.

Another accuracy-disruptive attribute of the categories is that they can overlap in meaning. Code with an error might also be confusing to the learner, meaning both annotations could possibly be chosen. Once HappyFace discovers these frustration categories caused by confusing code and errors, intervention systems can be informed by HappyFace to produce and present interventions. One potential system for HappyFace integration is the Apex system, which automatically provides students explanations for their runtime errors by presenting the learners with the root cause and

an explanation for why the cause produces an error [DKRR16]. Similarly, Singh et al. presented a method for providing feedback automatically for learner errors by using an error model to present corrections to a learner's code [RSSL13]. Both of these methods can be helpful interventions to aid learner understanding of the errors encountered in the execution of code.

Finally, our current extracted features may not be good predictors for the categories, so we must further investigate features that can be extracted from learners' code that could be used to improve category prediction. Despite these current weaknesses, the ability to predict that a learner does not understand the code relates to our finding that complex features cause frustration in learners.

5.1.4.1 Programmer Frustration

Ford et al. investigated causes of frustration for software developers via a survey [FOR15]. The reported causes of frustration were then grouped into several categories ranging from frustrations caused by the skill of the developer, the complexity of the code, and the features of the programming tools in use. Rather than manual surveys to elicit frustrating features in programming, HappyFace analyzes code with reported frustration to discover frustrating features automatically. Further, HappyFace can predict the problem categories a user is facing based on the feature of the code, removing the necessity of manual categorization of user frustrations previously done and lessening the effort required to discover frustrations felt by programmers.

5.2 Challenges

5.2.1 Integrations

While Python Tutor is an excellent platform to integrate HappyFace into, we would like to expand HappyFace's reach into stand-alone IDEs. The survey component of HappyFace's current implementation allows it to be integrated into any web-based IDE as it is written in HTML, but many IDEs like Eclipse or Visual Studio cannot implement the survey. In order to increase the breadth of HappyFace's ability to report learner frustrations, it must have the ability to be implemented into the tools that learners use. For this a plug-in must be developed that replicates the HTML survey component implementation of HappyFace.

5.2.2 Extraction of Informative Features

Some of the stylometric and AST features we extracted from learner code correlated with frustration felt by the learner, but it is possible we did not discover and collect features that more strongly

correlated with frustration. Further investigation into what frustrates a learner can help discover new features that would be beneficial to extract. One method, implemented in our extended survey, is to elicit causes of frustration from the learner through free-form text fields. After learners report other causes of frustration, we can extract features we believe correlate with that cause to analyze the strength of correlation the new features have with frustration.

5.2.3 Participation

Increasing learner participation will give our analysis more data on features that correlate with frustration. We can increase participation rate by better survey design, prompts to the learner, or even automated detection of frustration. Our implementation of HappyFace in Python Tutor was a passive survey that did not prompt the learner to provide a vote. It is possible that knowing automated help could be given by the system could increase participation by learners. We did not want to prompt the learner through a pop-up notification as to limit the interruption the survey would cause the learner, potentially becoming a frustrating feature itself. In a classroom and MOOC setting, learners can be prompted by the course instructor to take the HappyFace survey, increasing participation to provide more data for analysis. Despite challenges to participation, the scale of the environments HappyFace can be implemented, such as millions of learners taking a MOOC course, means that even low percentages of participation can mean thousands of participants providing valuable data to analyze.

5.3 Limitations

HappyFace and our studies have the following limitations:

While HappyFace can collect any code language, the capability of our analysis is limited to Python 2 and 3 code. In order to increase the breadth of our analysis we would need to parse each language's code into ASTs through libraries like Esprima for JavaScript code. This will give us insight on the differences in frustrating features in code languages. For example, the metric of semicolons per line may affect Java learners much more than Python learners as semicolons are required in Java to terminate lines. Another example is indentation in Python which affects functionality and execution paths, so it may correlate with frustration in Python but not in Java. In order to investigate these differences HappyFace must be extended.

HappyFace only receives data on a vote by a learner, but frustration is a continuous experience. Frustration may change as the learner is programming, rising and falling throughout the process. For example, a syntax error may cause little frustration when the learner has just started programming,

but the frustration felt by receiving the error may increase exponentially the longer the learner has been coding. Since HappyFace only collects the feedback at the discrete time of the vote, we cannot analyze the entire learning experience. In order to handle this limitation, it may be beneficial to conduct a controlled experiment where the learner is told to report their current emotion periodically so we can analyze changes over time. Further, rather than self-reported feedback we could detect frustration through bodily reactions similarly to JavaTutor [Vai16] over an entire programming session. Thus, we see the need to integrate HappyFace with existing continuous affect systems or future studies to more finely understand the time course of frustration.

Another issue is that different learners experience frustration differently. An impatient learner may find certain obstacles much more frustrating than most. It is also possible that some votes by learners represented frustration with external factors and may already be frustrated before coding. These external factors can play a role in learner frustration that we cannot detect. Perhaps a learner was having a bad day and receiving a (usually) non-frustrating obstacle affected them greatly, causing them to report they were frustrated. For example, during the follow-up study one learner selected the "Other" annotation and wrote "I have a midterm today" as the reason they were frustrated. This may falsely affect the correlation between certain features and frustration.

5.4 Related Work

5.4.1 Lightweight and Static Analysis of Code at Scale

OverCode is a system that uses static analysis of code to cluster solutions written by learners [Gla15]. Glassman et al. found that OverCode gave teachers a high-level overview that allowed them to discern students' understanding and misconceptions about the code they are working on. OverCode accomplished this by reformatting and "cleaning" (standardizing) raw code, then creating stacks of resulting code that became identical. Teachers or teaching assistants can then observe these stacks for misconceptions and holes in learners' knowledge. HappyFace is also a system that uses static analysis of code, but instead of presenting an expert with stacks of solutions it extracts code features and correlates them with user-reported frustration levels. Further, HappyFace performs its analysis on raw code rather than cleaned code to keep stylometric features and word-unigrams intact. There can be valuable information in these features that standardizing the code can miss. For example, the occurrence of the word "Fibonacci" in the frustrating code could show that learners have frustration with implementing that algorithm. While OverCode offers analysis at scale by clustering solutions, these are solutions to the same problems faced by some group of learners as part of a course. HappyFace instead analyzes all code, no matter the problem being solved, to discover frustrating

features in programming. These discovered features, possibly the result of learner misconceptions or complexities beyond the learner's current understanding, can be obtained from code at a scale beyond a single problem space.

5.4.2 Using the Crowd

Codepourri uses a volunteer crowd of learners to create code tutorials and annotate each step of the tutorials [GG15]. The best annotations for use in a tutorial were chosen through a vote by the learners. These tutorials were judged by experts to be near the quality of their own expert-created tutorials. The importance of this finding to HappyFace is that a crowd of learners can provide a similar quality of observation that experts can and can even provide insights that experts missed. HappyFace uses a crowd of learners to discover and predict causes of frustration in programming. Rather than expert analysis of features that may cause frustration, learners report when they are frustrated and the code causing frustration. We can then correlate the features of the code with frustration scores to identify frustrating experiences in programming. In the extended survey, HappyFace allows learners to annotate their frustration and code to experiment with HappyFace's ability to predict these annotations based on frustration and code metrics.

CHAPTER

6

CONCLUSION

Frustrating obstacles during programming caused by lack of support, course difficulty, and even lack of skills can cause learners to drop out of large-scale MOOCs. In this paper we present HappyFace, a light-weight feedback mechanism and automated analysis and prediction engine that can discover these frustrations felt by learners. HappyFace can gather and discover frustrating features at scale. This allows an efficient method of data collection from thousands of students quickly and painlessly by collecting learner affect and code in one button click. We believe that HappyFace's automated analysis and prediction can be used to inform intervention systems built into learning frameworks in order to provide learners with solutions to frustrations. When these systems lack a specific intervention for a frustration, HappyFace can discover common frustrating features that intervention systems can be extended to solve. Once informed interventions are in place, we believe the completion rates of MOOCs will rise.

BIBLIOGRAPHY

[ACIG15] Aylin Caliskan-Islam Richard Harang, A. L.A.N.C.V.F. Y. & Greenstadt, R. "De-anonymizing programmers via code stylometry". *SEC'15 Proceedings of the 24th USENIX Conference on Security Symposium* (2015).

[BH10] BjÃűrn Hartmann Daniel MacDougall, J. B.S.R. K. "What Would Other Programmers Do? Suggesting Solutions to Error Messages". *CHI 2010* (2010).

[Con13] Conole, G. *MOOCs as Disruptive Technologies: Strategies For Enhancing The Learner Experience and Quality of MOOCs, http://www.um.es/ead/red/39/conole.pdf*. 2013.

[DBZ90] Daiva Bieri Robert Reeve, G. C. & Ziegler, J. B. "The Faces Pain Scale for the self-assessment of the severity of pain experienced by children: Development, initial validation, and preliminary investigation for ratio scale properties". *Pain* (1990), pp. 139–150.

[DKRR16] Dohyeong Kim Yonghwi Kwon, P. L.I.L.K.D.M.P.X. Z. & Rodriguez-Rivera, G. "Apex: automatic programming assignment error explanation". *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2016)* (2016).

[FOR15] FORD D., P. C. "Exploring the Causes of Frustration for Software Developers". *IEEE ICSE 8th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)* (2015).

[Fou] Foundation, P. S. *Programming FAQ, https://docs.python.org/3/faq/programming.htmlwhat-are-the-rules-for-local-and-global-variables-in-python*.

[Gla15] Glassman, E. L. et al. "OverCode: Visualizing Variation in Student Solutions to Programming Problems at Scale". *ACM Trans. Comput.-Hum. Interact.* **22**.2 (2015), 7:1–7:35.

[GG15] Gordon, M. & Guo, P. J. "Codepourri: Creating visual coding tutorials using a volunteer crowd of learners". *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. VL/HCC '15. 2015, pp. 13–21.

[Guo] Guo, P. *Python Tutor, http://pythontutor.com/*.

[Guo13] Guo, P. J. "Online Python Tutor: Embeddable Web-based Program Visualization for CS Education". *SIGCSE ACM, 579-584* (2013).

[JC10] Jason Carter, P. D. "Design, implementation, and evaluation of an approach for determining when programmers are having difficulty". *GROUP '10 Proceedings of the 16th ACM international conference on Supporting group work* (2010).

[Jor] Jordan, K. *MOOC Completion Rates: The Data, http://www.katyjordan.com/MOOCproject.html*.

[OB14] Onah Daniel F. O., S. J. & Boyatt, R. “Dropout rates of massive open online courses : behavioural patterns”. *EDULEARN14 Proceedings pp. 5825-5834* (2014).

[Par] Parnin, C. *checkbox.io, http://checkbox.io/*.

[RSSL13] Rishabh Singh, S. G. & Solar-Lezama, A. “Automated feedback generation for introductory programming assignments”. *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2013)* (2013).

[Szu10] Szumilas, M. “Explaining Odds Ratios”. *J Can Acad Child Adolesc Psychiatry* (2010), pp. 227–229.

[Vai16] Vail, A. K. et al. “The Affective Impact of Tutor Questions: Predicting Frustration and Engagement”. *Proceedings of the 9th International Conference on Educational Data Mining* (2016).