A Tool for Authoring Programs that Automatically Distribute Feedback to Novice Programmers

Michelle Ichinco, Yoanna Dosouto, Caitlin Kelleher

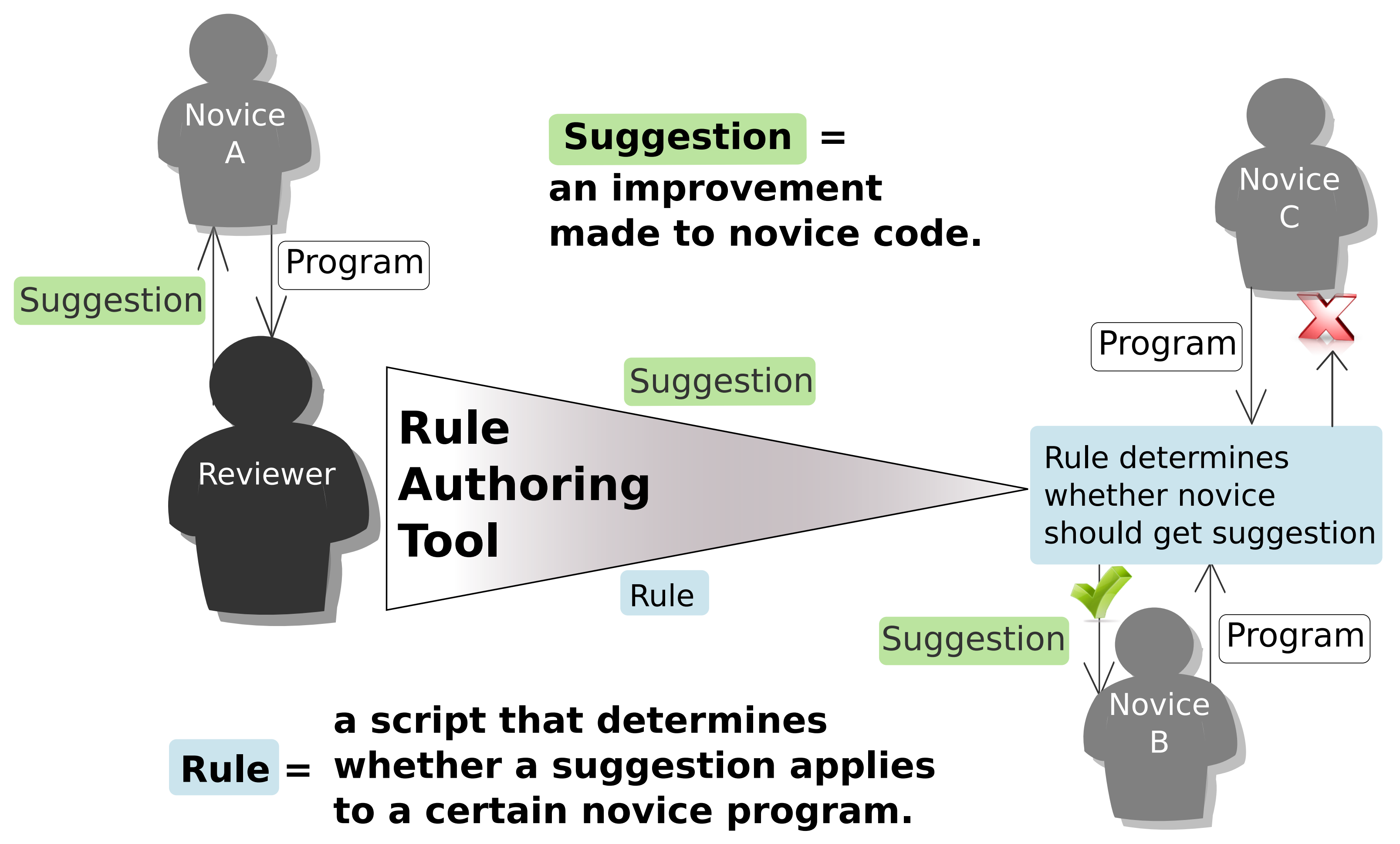
Washington University in St. Louis

Introduction

Novice programming environments can provide motivating contexts for learning programming. However, novices programming independently have no way to receive the type of unsolicited feedback a teacher provides in a classroom. Tools exist to help novices fix syntax errors and bugs, but they do not help novices recognize missed opportunities for improving their code when working on self-directed projects.

One way to provide this feedback is by crowdsourcing experienced programmers' suggestions and then enabling experienced programmers to author rule scripts to distribute the suggestions at a large scale.

Crowdsourced Review System



Suggestion = **an improvement made to novice code.**

Rule = **a script that determines whether a suggestion applies to a certain novice program.**

Formative Study

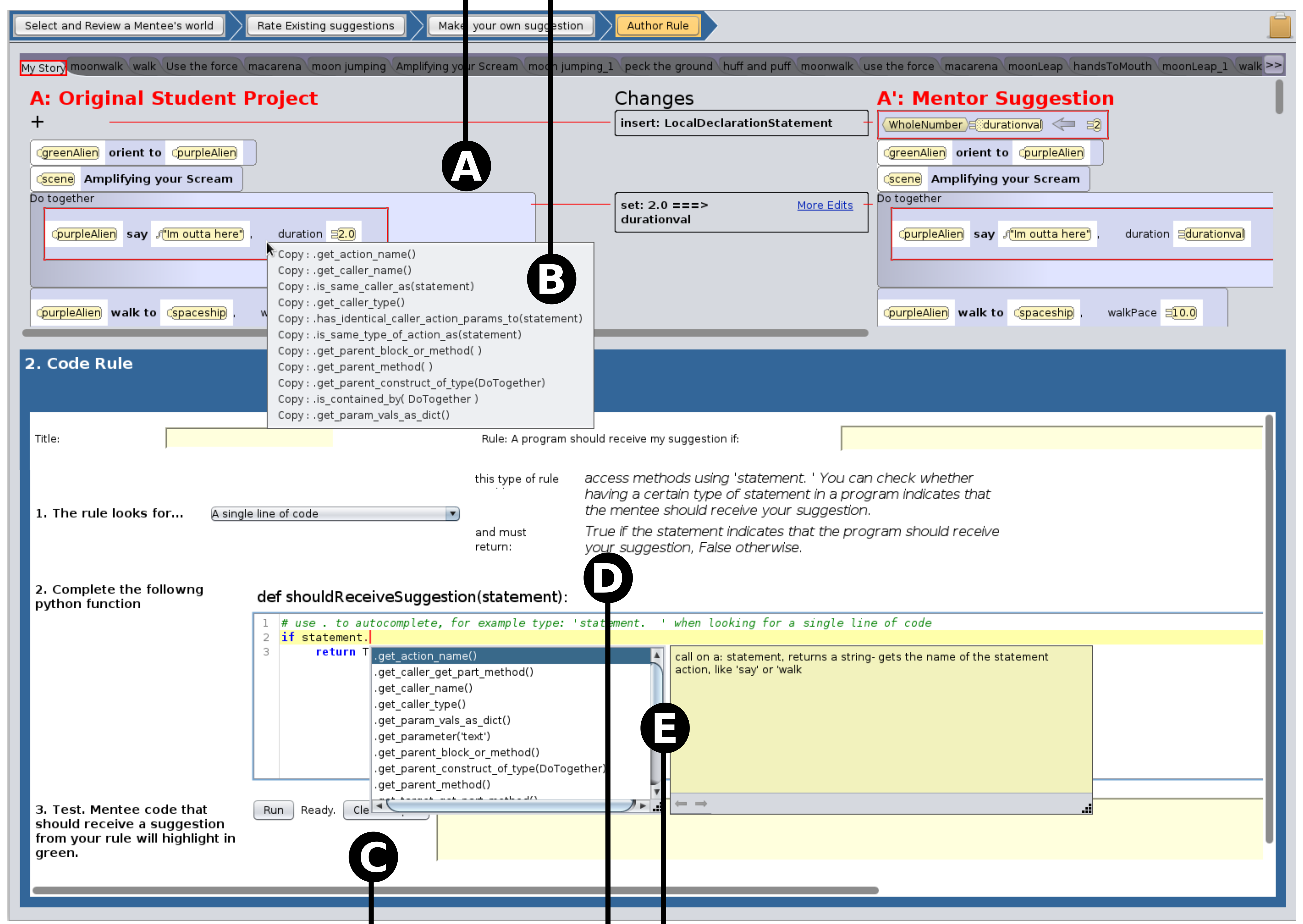
We ran an iterative, formative study to design and evaluate a rule authoring tool. We recruited 31 participants who had at least 2 years of programming experience. Participants made suggestions by improving code and used the reviewer tool to author rules. This work seeks to answer the following questions:

➡ *What barriers exist in rule authoring?*

➡ *How can we enable reviewers to access information about novice code naturally by providing API methods?*

Reviewer Rule Authoring Tool

After making changes to improve a novice program, the reviewer can use this tool to author a rule that checks when other novice programs should receive the same suggestion.


Select and Review a Mentee's world
Rate Existing suggestions
Make your own suggestion
Author Rule
My Story | moonwalk | walk | Use the force | macarena | moon jumping | Amplifying your Scream | moon jumping_1 | peck the ground | huff and puff | moonwalk | use the force | macarena | moonLeap | handsToMouth | moonLeap_1 | walk
A: Original Student Project
Changes
A': Mentor Suggestion
More Edits
2. Code Rule
A
B
C
D
E
Run
Ready.

A: Original Student Project

greenAlien orient to purpleAlien
scene Amplifying your Scream
Do together
purpleAlien say "Im outta here" , duration 2.0
purpleAlien walk to spaceship

Changes

insert: LocalDeclarationStatement

set: 2.0 ===> durationval

A': Mentor Suggestion

WholeNumber durationval ⇐ 2
greenAlien orient to purpleAlien
scene Amplifying your Scream
Do together
purpleAlien say "Im outta here" , duration durationval
purpleAlien walk to spaceship , walkPace 10.0

```
Copy : .get_action_name()
Copy : .get_caller_name()
Copy : .is_same_caller_as(statement)
Copy : .get_caller_type()
Copy : .has_identical_caller_action_params_to(statement)
Copy : .is_same_type_of_action_as(statement)
Copy : .get_parent_block_or_method( )
Copy : .get_parent_method( )
Copy : .get_parent_construct_of_type(DoTogether)
Copy : .is_contained_by( DoTogether )
Copy : .get_param_vals_as_dict()
```

Title:

Rule: A program should receive my suggestion if:

1. The rule looks for... A single line of code

this type of rule *access methods using 'statement. ' You can check whether having a certain type of statement in a program indicates that the mentee should receive your suggestion.*

and must return: *True if the statement indicates that the program should receive your suggestion, False otherwise.*

2. Complete the followng python function

```
def shouldReceiveSuggestion(statement):
# use . to autocomplete, for example type: 'statement.  ' when looking for a single line of code
if statement.
    return T
```

```
.get_action_name()
.get_caller_get_part_method()
.get_caller_name()
.get_caller_type()
.get_param_vals_as_dict()
.get_parameter('text')
.get_parent_block_or_method()
.get_parent_construct_of_type(DoTogether)
.get_parent_method()
```

call on a: statement, returns a string- gets the name of the statement action, like 'say' or 'walk

3. Test. Mentee code that should receive a suggestion from your rule will highlight in green.

Suggestion code change

Code API Access

- Unexpected way to access an API.
- Splitting up the API methods makes it difficult to understand the API as a whole. *For example, clicking on the 'say' code block only shows the API methods applicable for 'say' blocks.*
- Lacking details and examples for API methods.
- We added hints in the second iteration to direct users to this API documentation, which may have been a factor in the decrease in time to author a first rule from an average of 37 minutes in iteration 1 to 18 minutes in the following 3 iterations.

Testing and output console

Rule script editor

- In the first iteration of this tool, the rule script editor and the testing and output console were split between two panes and the 'testing and output console' pane was initially minimized.
- In the first iteration, participants spent on average 20 minutes authoring their rule before pressing the 'Run' button.
- For iterations 2-4, in which the two panes were combined (as seen in this screenshot), the average was about 6 minutes.

Autocomplete API Access

In iteration 4, we added explanations and examples of the API methods as a part of the autocomplete to better align with standard API documentation.