Code Shaping

Iterative Code Editing with Free-form Sketching

Ryan Yen
University of Waterloo

Jian Zhao
University of Waterloo

Daniel Vogel
University of Waterloo

motivation

Ink-based sketching has long been used in programming tools to help programmers express ideas visually and intuitively. These sketches serve various purposes, such as enhancing code comprehension, communicating with collaborators, and specifying intended edits.

solution

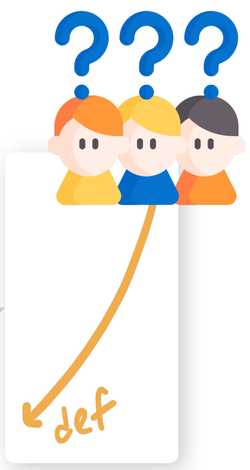
Recent advances in sketch recognition have brought up the possibility of a concept we termed, "**code shaping**," where sketched annotations are operationalized into actionable code edits. Unlike the current most prevalent tools that convert sketched visualizations or UI components into code, our research explores editing code directly through sketches on the code editor. This approach allows programmers to encapsulate their expectations of code functionality through sketches, connecting annotations with the syntactic structure of the code.

study

However, the ambiguous nature of sketches presents challenges, as similar annotations can have different meanings in various contexts. To address this, we conducted an exploratory study with six programmers using a prototype that converts free-form sketches into code edits, aiming to understand how programmers convey and how systems interpret these annotations.

Finding #1:
Programmers Developed their Personalized Workflow

Participants developed personalized workflows. Some found breaking tasks into too much detail ineffective for AI interpretation, while others preferred smaller task pieces for better understanding. High-level instructions were often used initially, followed by detailed annotations after evaluating the generated code.



Finding #2:
A Single Sketch can Represent Various Meanings

Similar sketches were used for different purposes. This ambiguity made it challenging for the system to accurately translate annotations into code edits, leading to clearer mappings between sketches and code edits, especially with multiple annotations.

3.1. Create a function to visualize the plot

3.2. Including visualization into the current function

3.3. Create a function refer to another existing function

```
31      plt.figure(figsize=(6, 4))
32      plt.hist(self.dataframe[column].dropna(), bins=10, density
33      plt.title(f'Distribution of {co}')
        plt.ylabel('Density')
        plt.xlabel(column)
        plt.grid(axis='y', alpha=0.75)
        plt.show()
```

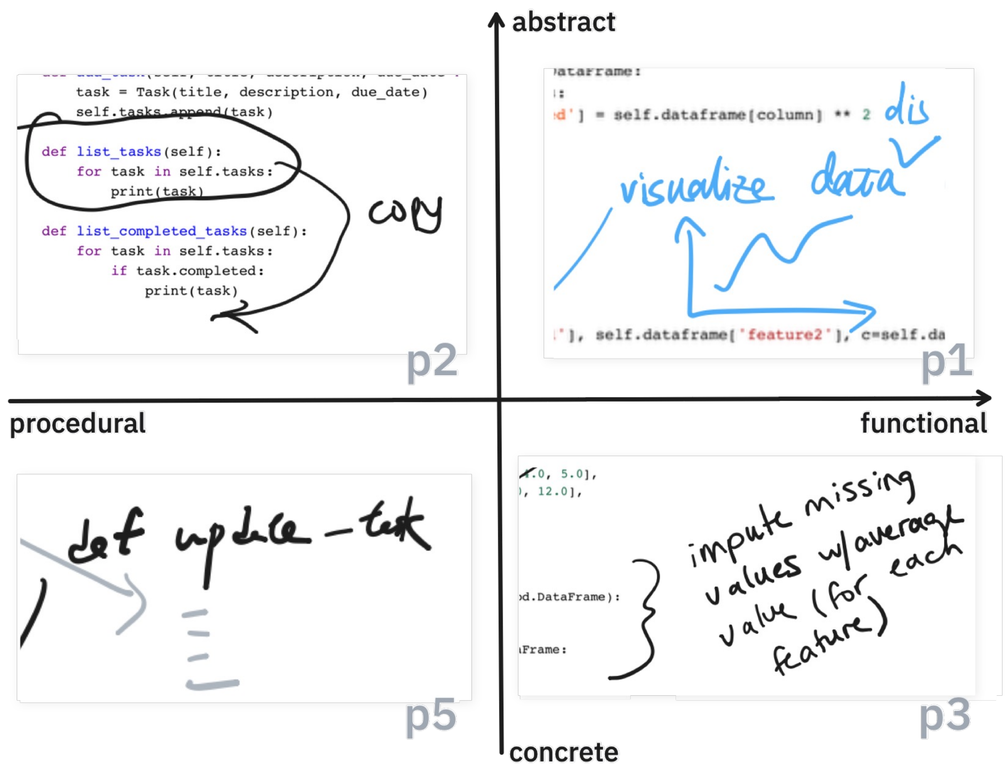
2. Plot distribution

```
9   data = {
10      'feature1': [1.0, 2.0, np.nan, 4.0, 5.0],
11      'feature2': [2.0, np.nan, 3.0, 4.0, 5.0],
        'feature3': [0.1, 0.2, 0.3, 0.4, 5.0],
        'label': [0, 1, 0, 1, 1]
    }
```

1. only first three features...

Finding #3:
Types of Sketches

Sketches were categorized into a quadrant based on two spectrums: **Abstract-Concrete** and **Procedural-Functional.** Abstract sketches included symbols or graphs, while concrete sketches were written text. Procedural sketches described the program structure, and functional sketches specified how the program should work. Participants often combined these aspects in their annotations.

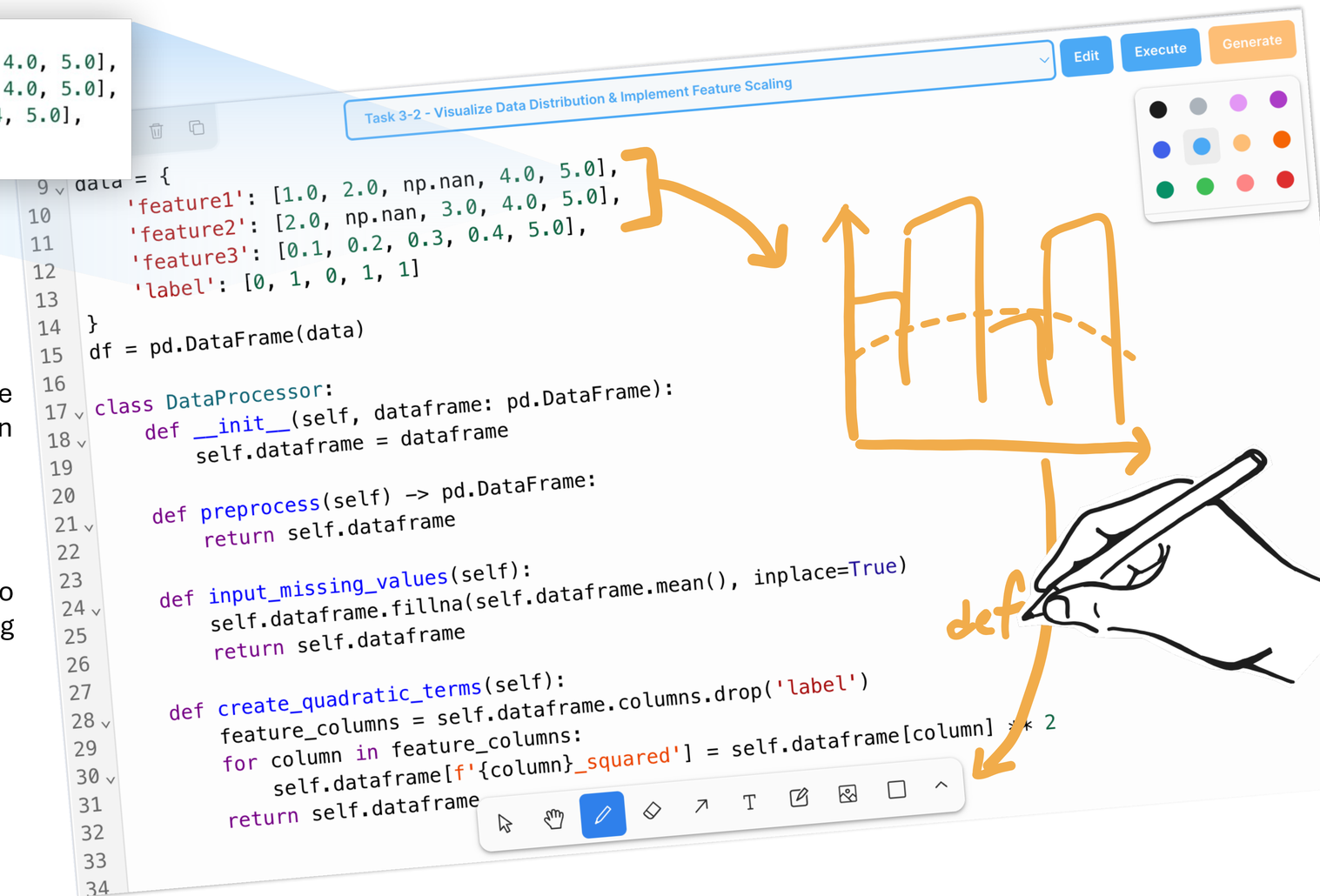


Finding #4:
Sketch as a Tool

Participants viewed sketches as functional tools that could be reused, not just static drawings. They chose sketches based on the environment and reused them to achieve similar effects.

Future Progress

We plan to iterate on the design based on the insights gained to support programmers in the iterative process of translating thought into sketches to code editing.



UNIVERSITY OF WATERLOO | HCI exii Waterloo Visualization Group