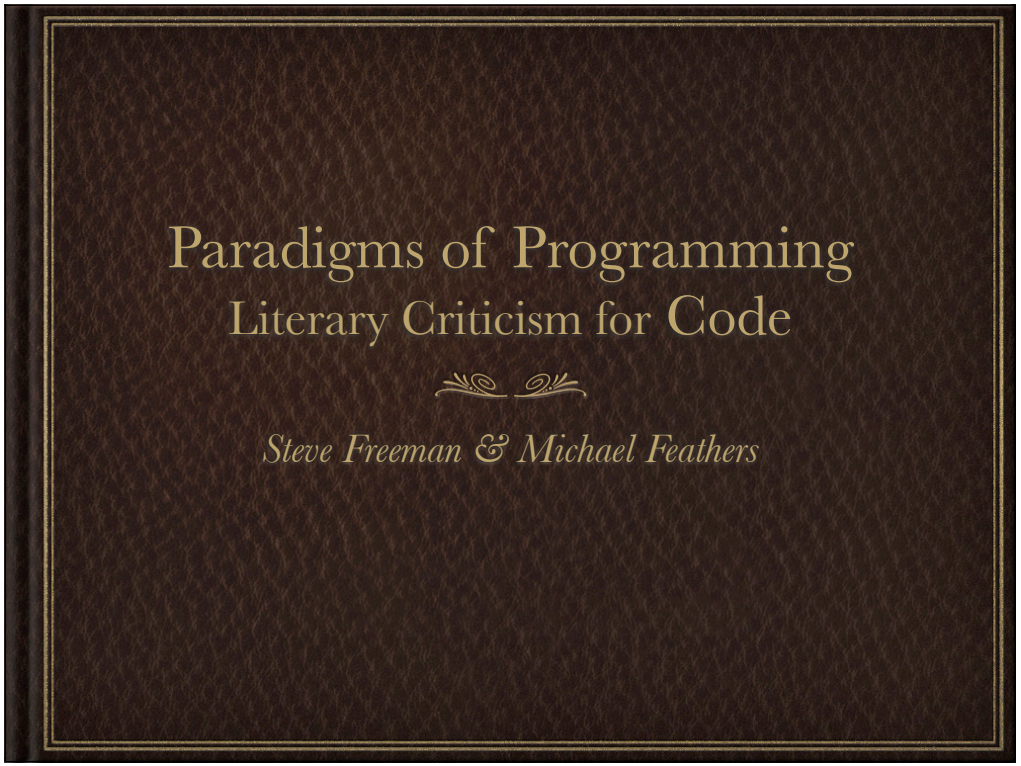
Paradigms of Programming
Literary Criticism for Code
Steve Freeman & Michael Feathers

Paradigms?

- Robert Floyd - ACM Turing Award acceptance: ‘Paradigms of Programming’ 1978
  - Programmatic ways of approaching problems
  - Languages can be changed to support paradigms (DSLs)
  - There is no one perfect language

Today

- Procedural
- Object Oriented
- Functional
- Rule-based

The Problem (I)

- The problem for this exercise is a listbox controller in a graphical user interface. You have to write code which accepts a list of items for the listbox and changes the currently selected item in the list.
- When the listbox is initialized, the current selection is the first element in the list. Two operations: arrowUp and arrowDown allow you to change the position of the current selection. If you arrowUp when the first element of the list is selected, it is a no-op. Likewise, it is a no-op if you arrowDown when the last element of the list is selected.
- If the controller is given an empty list, there is no currently selectioned item.
- In later iterations, we will expand this behavior to make it more

The Problem (II)

- Now that we have a rudimentary listbox, we have to start to make it a bit more realistic. In our listbox, we will introduce the concept of the "window." The window is the portion of the listbox which is currently displayed on the graphical user interface. Imagine a listbox with 100 elements. The window might be the set of elements in the range [10..19].

- In this iteration, you will have to change the system so that the current selection moves within this window. The arrowUp and arrowDown operations have special behavior at the top and bottom of the window -- they move it. For example, if the window is at [10..19] in a list, and the current selection is 10, an arrowDown operation will change the current selection to 11. On the other hand, an arrowUp operation will change the the current selection to 9 and shift the window up, giving it the range [9..18].

- In this iteration, it is sufficient to add behavior for windows in the center of the list. You do not have to deal with the cases where the window touches the top or bottom of the list yet.

The Problem (III)

- In this iteration, we introduce behavior at the edges. An arrowUp operation at the top of the window when the window is [0..x] for some x is a no op. An arrowDown operation at the bottom of the window when the window is [x..last] for some x is a no op also.
- In this iteration, we also calculate the size of the window. By default, the size of the window is 10 and it starts at the first element of the list (0). However, if the size of the list is less than 10, the size of the window becomes the size of the list.
- Take time in this iteration to refactor your solution. Become as comfortable with it as you can.

The Problem (IV)

- In this iteration, we add the operations pageUp and pageDown. The pageUp operation moves the window up windowSize elements so that it is just above it's previous first element. When the operation is complete, no element which was visible in the window before is visible in the window afterward. If pageUp can not go up windowSize elements because it hits the top of the list, it stops at its last possible move up.
- The current selection after a pageUp is the last element in the window.
- The pageDown operation has the exact opposite behavior. It moves the window downward. When a pageDown finishes, the current selection is the top element of of the window.