1 modifying existing ops

Sometimes, it would be useful to modify existing operators. However, it could also be very dangerous and make things very confusing. Is there a good way to make this possible in \English that avoids danger?

First a use case: I was trying to create a WebGL program in JavaScript for a CSE470 (Computer Graphics class) assignment. In this part of the assignment, I had to draw a line between any two points that the user clicked on. Because old lines were supposed to stay on the screen and for other complicated reasons, I was keeping track of a list of all the lines to draw and redrawing all the lines when a new line was introduced. The problem was that there was a `clearCanvas()` function in the pre-defined utility library, but when the canvas was cleared with that function, it wouldn't clear my list of lines. I had no way of knowing when the canvas was cleared without modifying the utility library... but that was against the "rules" of the assignment. In short, sometimes you need to listen to or even modify the functionality of an operator without modifying it. Should this be allowed? To what extent? Is it dangerous?

1.1 possible solutions

I figure there are a few different levels of power that could be implemented for such a system; in order of most powerful and more dangerous to least powerful and least dangerous, they are as follows:

1. We could include the ability to modify operators to the fullest extent. Modifiers already exist in \English, so we could make the system similar to those. (Modifiers like "`private`" can go at the beginning of operator declarations to change their functionality much like Python function decorators).
   - As example of possible use, imagine that you want to change the canvas-clearing functionality so that it can only clear the canvas when there are 3 or more lines (where the syntax of the example could be subject to change):

     ```
     change (clear the canvas):
             if Lines's size <= 3,
                     clear the canvas
     ```

   - Pros:
     - It allows for large amounts of flexibility, even when using an unmodifiable library or A.P.I.
   - Cons:
     - It's very dangerous. This is another prime example of "obfuscating" functionality, allowing functionality of an operator to be described far away from the operator. If someone reads that `clear the canvas` operator, they would see nothing about only doing it if there were three or more lines, so they would be quite confused.
     - It can make security holes in some kind of secure library, e.g. a library that works as an A.P.I. for building plugins for a program that handles sensitive data. What if someone used this to change a hash function in a security library do nothing?
2. We could have the ability to modify operators, but also include a modifier, e.g. "`secure`", that wouldn't allow the operator to be modified.
   - Pros:
     - It could allow libraries to secure certain essential parts as long as the authors of that library are aware of and smart enough to use the modifier.
   - Cons:
     - This is still very dangerous because of the code obfuscation issue (see above).
3. You can't modify operators, but you can listen to them and *add* functionality.
   - This example goes back to the WebGL example I mentioned before; you could clear the cache of lines whenever the canvas is cleared:

```
when (clear the canvas) is used,
        clear Lines
```

- Pros:
  - It's much more secure than either of the above options.
  - It reduces the issue of code obfuscation; with only listening, if you read the declaration of "`clear the canvas`", at least you know that everything in the body of that operator will be executed and not cancelled by code in some random file somewhere.
- Cons:
  - Some of the "obfuscation" problem still remains.

4. Neither listening nor modification of operations is allowed.
   - In the WebGL example... well, how would you solve that? You couldn't, really.
   - Pros:
     - No code obfuscation issue at all; when you read an operator declaration, you know exactly what it does, no more and no less.
   - Cons:
     - Problems like the WebGL example pop up and aren't really solvable.