Programming With Everybody: Tightening the Copy-Modify-Publish Feedback Loop

Tom Lieber
MIT CSAIL
32 Vassar St, Cambridge, MA
tl@csail.mit.edu

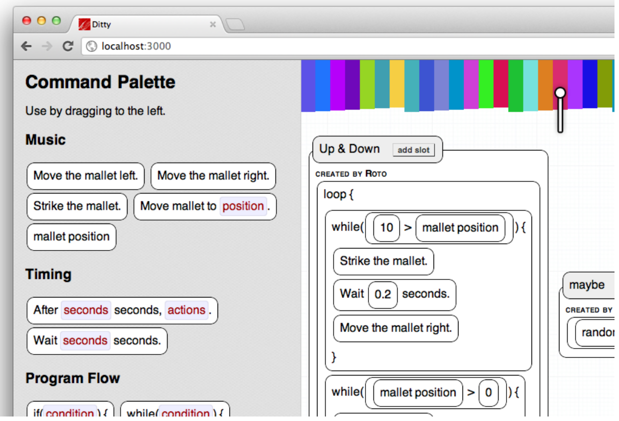
Robert C. Miller
MIT CSAIL
32 Vassar St, Cambridge, MA
rcm@mit.edu

ABSTRACT
People write more code than they ever share online. They also copy and tweak code more often than they contribute their modifications back to the public. These situations can lead to widespread duplication of effort. However, the copy-modify-publish feedback loop which could solve the problem is inhibited by the effort required to publish code online. In this paper we present our preliminary, ongoing effort to create Ditty, a programming environment that attacks the problem by sharing changes immediately, making all code public by default. Ditty tracks the changes users make to code they find and exposes the modified versions alongside the original so that commonly-used derivatives can eventually become canonical. Our work will examine mechanical and social methods to consolidate global effort on common code snippets, and the effects of designing a programming interface that inspires a feeling of the whole world programming together.

Author Keywords
Programming environments; collaboration; awareness; open source software development; social computing.

ACM Classification Keywords
H.5.2 [Information interfaces and presentation]: User Interfaces - Graphical user interfaces.

INTRODUCTION
The natural state of programming today is one person, one computer, and a local disk drive. Sharing code on blogs, forums, question-and-answer sites, and public source code repositories has become commonplace, but only when the user has an explicit intention to publish their work. On the other hand, it's easy to steal code, and copying is a common strategy to build programs efficiently [1]. Code that's published online by one person can be used by many—adapted, tweaked, and tucked away. Though some of those (potentially) improved versions make it online where they can be refined again by the next wave of scavengers, often they're simply forgotten because putting code online takes effort and doesn't help the user complete their current task any faster.



```
loop {
  while( 10 > mallet position ) {
    Strike the mallet.
    Wait 0.2 seconds.
    Move the mallet right.
  }
  while( mallet position > 0 ) {
```

Figure 1. Ditty is a collection of niche programming environments on the web. This one caters to generative music. Pictured above: the palette of frequently-used commands (left), interactive xylophone visualization (top), and code canvas (right).

Nevertheless, the modifications made during that process *can* be useful to others. Sometimes the code's behavior is changed so that it can work in a new environment. Sometimes the adaptation is simply to improve the code's design according to the new author's aesthetics. Those instances (and likely others) result in massively duplicated effort if the use cases are common or the original design was poor. If we can somehow harvest the effort people put into adapting code to aid others who would otherwise need to do the same, then there is the potential to save a lot of time for everybody.

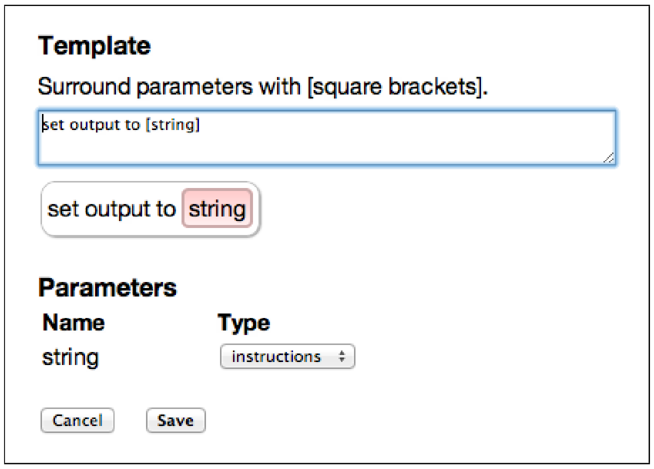
This abstract is about Ditty, our programming environment that attempts to address the problem by combining these two strategies:

- Make all code public by default, and
- Link snippet instances to their original sources so that derivatives can be easily discoverable.

What follows in the next section is a detailed description of the relevant portions of the interface and the reasoning behind their design.

PROTOTYPE IMPLEMENTATION
Ditty is available as a web site that works in many modern desktop browsers when used with a keyboard and mouse.

Because we wanted to easily be able to track the types of modifications users make to the code blocks, and to increase the approachability of the interface, we decided to implement



Template

Surround parameters with [square brackets].

```
set output to [string]
```

set output to string

Parameters

Name	Type
string	instructions

Figure 2. A command block's name is a template where parameters are the segments surrounded by square brackets. Both terse ("substr([string], [pos], [length])") and natural language ("substring of [string], [length] characters long, starting at [position]") styles are supported.

Ditty as a tile-based language. Users construct programs by dragging together code blocks, manipulating the syntax tree of their program directly. It works much like in Scratch, Alice, and many others [8, 2, 4, 9, 10, 5].

Modularity is achieved through the creation of *command blocks* (analogous to functions and macros in other languages). The 'name' of a command block is just a free-form template, with sub-strings knocked out to create slots where parameters can go, similar to BYOB [3] (see Figure 2).

Searching for Public Commands

As mentioned before, all commands are public. When a user edits the definition of someone else's command, a new version is created with a link back to the original. Thus, we can easily track every command's ancestry. The user can navigate the family tree both ways in order to explore all of the available variations. People can also see all of the instances where a command is *used*, which is useful for finding examples. For the moment, sets of parents, children, and callers are displayed in simple lists, but we are exploring alternatives for finding useful commands quickly.

The primary means of accessing blocks created by other users is search. Ditty augments the drag-and-drop programming interface with keyboard-based auto-completion. If the user clicks a slot instead of dragging a block into it, a text box appears into which they can type the name of the block that they have in mind. As they type, a keyword search of every available command (even commands created by other users) is performed and the results are displayed in a drop-down menu. Closer matches and matches which have been used recently are given priority in the list.

When commands sharing a parent-child relationship are both in the results (one is a modified version of the other), we plan to adjust the rank depending on metrics like 'number of times used' and 'number of times modified,' which we expect to correlate well with the likelihood that the user would want to choose one command over another. We hope that over time this will allow the alternatives which are more generally useful to rise in the list for everyone.

CONCLUSION

Public-by-default is not a new idea. For example, CoScripter automatically publishes scripts to a public wiki where they can be copied and edited [6]. Tracking code provenance is also not new; source control systems are built to do that, and systems like Blueprint provide that functionality for snippets copied from the web [1]. Scratch also provides means of automatic and manual attribution for resources that were copied from another project [7].

What's unique about Ditty is the tightness of its copy-modify-publish feedback loop. Everything about the editing interface, such as global auto-complete and immediate publication, encourages users to act as if entire world shares a single, global namespace.

Ditty is still in the development stage, although we are nearing a public release. Initial user studies suggest the basic usability of the system for experienced programmers in a lab setting, but it remains to be seen how scalable and usable the ideas turn out to be in practice. Our hope is to work with real users in the wild to determine how best to employ this radical means of collaboration.

REFERENCES

1. Brandt, J., Dontcheva, M., Weskamp, M., and Klemmer, S. Example-centric programming: integrating web search into the development environment. In *Proc. SIGCHI*, ACM (2010), 513–522.
2. Cooper, S., Dann, W., and Pausch, R. Teaching objects-first in introductory computer science. In *ACM SIGCSE Bulletin*, vol. 35, ACM (2003), 191–195.
3. Harvey, B., and Mönig, J. Bringing no ceiling to Scratch: Can one language serve kids and computer scientists? *Proc. Constructionism* (2010).
4. Kay, A. Squeak Etoys authoring & media. *Viewpoints Research Institute* (2005).
5. Ko, A., and Myers, B. Barista: An implementation framework for enabling new tools, interaction techniques and views in code editors. In *Proc. SIGCHI*, ACM (2006), 387–396.
6. Leshed, G., Haber, E., Matthews, T., and Lau, T. CoScripter: automating & sharing how-to knowledge in the enterprise. In *Proc. SIGCHI*, ACM (2008), 1719–1728.
7. Monroy-Hernández, A., Hill, B., Gonzalez-Rivero, J., et al. Computers can't give credit: how automatic attribution falls short in an online remixing community. In *Proc. SIGCHI*, ACM (2011), 3421–3430.
8. Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., et al. Scratch: programming for all. *Communications of the ACM 52*, 11 (2009), 60–67.
9. Warth, A., Yamamiya, T., Ohshima, Y., and Wallace, S. Toward a more scalable end-user scripting language. In *Creating, Connecting and Collaborating through Computing*, IEEE (2008), 172–178.
10. Wolber, D. App inventor and real-world motivation. In *Proc. SIGCSE*, ACM (2011), 601–606.