Really Programming in Public

Tom Lieber
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02142
E-mail: dynamic@mit.edu

I. Introduction

Once online, code multiplies. One programmer asks a programming question, another person answers it with a snippet of code, and then the snippet is pasted into countless projects, adapted, tweaked, and tucked away. Some of those improved versions make it back online where they can be refined again by the next wave of scavengers, but often they're simply forgotten because putting code online takes effort.

Nevertheless, those modifications might be useful to others. Sometimes the behavior needs to change for it to work in a new environment. Sometimes the adaptation is simply to improve the snippet's design according to the new author's aesthetics. Both instances result in massively duplicated effort if the use cases are common or the original design was poor. That's true for entire libraries as well; it is common for people to fork entire projects on GitHub to track and share their personal modifications, but that's not possible for most snippets on the web.

I believe that an *efficient* snippet ecosystem requires a less formal and more automatic means of sharing changes. A first step would be to short-circuit the snippet adaptation feedback loop by:

- making all code public by default, and
- linking snippet instances to their original sources so that derivatives are easily discoverable.

This would allow users to benefit from the purely selfish process of adapting code even in the absence of the additional, altruistc step of publishing the changes afterward. Snippets could improve in quality over time whether users think of sharing them or not.

There are many ways to approach an interface for such collaboration and the work described here is one possible manifestation of the idea.

II. Really Programming in Ditty

One path to this goal would be to create an IDE plug-in like Blueprint that remembers the origin of pasted code [1] and automatically uploads modifications to a public server. Others who viewed the same snippet through the plug-in would also be able see common modifications and choose one of them instead. This path is complicated because, for most languages, it would require tracking changes to amorphous blobs of text in a sea of amorphous blobs of text, a problem which is orthogonal to the ideas I'd like to test.

Instead, I'm working on a simple language and web-based programming environment called *Ditty*. It is easily amenable to automatic change tracking because the language is tile-based (like Scratch, where the syntax tree is edited directly, typically using drag-and-drop gestures [2]), so all program edits are well-defined transformations. The decision to use tiles has the side benefits of being more usable for novices and being able to support natural language tile names, which may accommodate opportunistic searching even without additional metadata.

Ditty's minimum sharable unit is the *tile definition*. User-defined tiles (which are like functions or macros in other languages) are public by default and are immediately available for use by other programmers. They can be found while browsing the site, inspecting others' code, or serendipitously through textual auto-completion, which works as a global search interface.

When a user modifies someone else's tile, a fork is automatically created in their name. That two-way link to the original is maintained and exposed to the user in both directions. When that tile appears in a search result, both its ancestors and descendants can be viewed as well. This allows the user to find related versions that may be better suited to the their taste and the task at hand, and to compare with the original to determine what changed. It's a common browsing mechanism for source code repositories like GitHub, but in Ditty the units under consideration are as fine-grained as a function and the infrastructure doesn't require any action that the user wasn't already going to do to use the code.

III. Programming with Everybody

Public code is not a new idea; for example, CoScripter automatically publishes scripts to a public wiki where they can be copied and edited [3]. It's the fact that reading from and writing to the public code database is so tightly integrated with the programming environment that raises the following array of interesting research questions.

A. Programmer Behavior

Programming has typically been a lonely endeavor, especially for the hobbyist working from home. The authorship information displayed in Ditty provides opportunities to make connections with other programmers, and serves as a constant reminder of just how many shoulders one stands upon. Automatic publication and attribution may mean that such a user

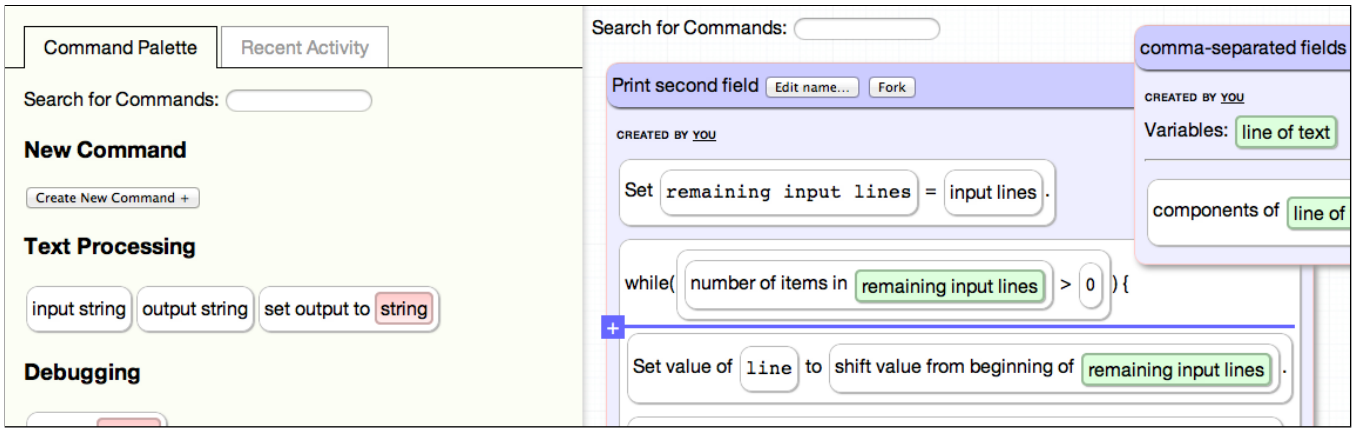

Fig. 1. There are several Ditty environments, each one supporting a different problem domain, but sharing a common language. This one caters to text processing tasks. Pictured are the built-in palette of frequently-used commands and the code canvas.

could log in one day to find that tiles they wrote have been used and adapted by many people that they've never met. How will the user react?

Some existing research in this vein comes from a study of GitHub users [4]. Most GitHub activity is public, as with Ditty, but users have the opportunity to carefully prepare everything they submit to the site. As a result, many people feel pressured to present a particular image of themselves. One interesting finding is that the motivation to share code comes more from noticing one's own reliance on others' code than realizing that one's published code is widely used. Using others' code is much more likely in Ditty because it's facilitated by the auto-completion interface, so the effect may be even more pronounced. I hypothesize that Ditty's interface will encourage the writing of more modular code because of the spotlight effect.

B. Attribution

Research on the Scratch user population [5] raises interesting questions about attribution in Ditty. Scratch projects (self-contained applets with games and animation) can be published to the web so that others may interact with them and copy their source material, including its code. One of the important lessons the authors learned from interviewing users (who were mostly children) was that automatic attribution was often inadequate. Some people felt that reuse was stealing unless a heartfelt 'thanks' was included as well. Some rejected the idea of reuse entirely. What kinds of attribution will be needed in Ditty, where reuse and derivation are integrated into the programming workflow and the definition of a single tile can rely upon dozens of other users' work?

C. Broken Code

It may become a challenge for users to sift through broken code. Ditty can vouch for popular tiles based on the assumption that they wouldn't be popular if they were broken, but it may be time-consuming for users to confirm the legitimacy of the rest. It remains to be seen what percentage of code is broken and whether good heuristics can be found to detect brokenness.

D. Interface Evolution

Built-in tiles are indistinguishable from user-defined tiles, and all tiles are public, so each one can be thought of as an addition to a single, community-curated language. The interface should reflect that. For example, the starting palette of commands is a categorized list that I designed to cover users' basic needs, but can it be automatically updated as better base abstractions are created and new coding patterns emerge?

IV. Ditty's Future

Results from preliminary user studies indicate that Ditty is usable in a lab setting by experienced programmers, and a field test demonstrated excitement about the new features I've described, but how well the model scales is a mystery I am still working to solve. In order to gauge the usefulness of the ideas presented here, I plan to launch Ditty as a public web site so that I can observe how it's used (or isn't used) in the wild. If successful, it has the potential to bring a world's worth of unpublished code improvements to light at last.

References

[1] J. Brandt, M. Dontcheva, M. Weskamp, and S. Klemmer, "Example-centric programming: integrating web search into the development environment," in *Proceedings of the 28th International Conference on Human Factors in Computing Systems*. ACM, 2010, pp. 513–522.

[2] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman *et al.*, "Scratch: programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[3] G. Leshed, E. Haber, T. Matthews, and T. Lau, "CoScripter: automating & sharing how-to knowledge in the enterprise," in *Proceedings of the twenty-sixth annual SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2008, pp. 1719–1728.

[4] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in GitHub: transparency and collaboration in an open software repository," in *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*. ACM, 2012, pp. 1277–1286.

[5] A. Monroy-Hernández, B. Hill, J. Gonzalez-Rivero *et al.*, "Computers can't give credit: how automatic attribution falls short in an online remixing community," in *Proceedings of the 2011 Annual Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 3421–3430.