Behavior-based clustering of visual code

Sheela Surisetty\textsuperscript{1}, Catherine Law\textsuperscript{2}, Chris Scaffidi\textsuperscript{1,3}

\textsuperscript{1} School of Electrical Engineering and Computer Science
\textsuperscript{2} STEM Academy
\textsuperscript{3} Center for Applied Systems and Software
Oregon State University
Corvallis, OR USA
\{surisets,lawc,scaffidc\}@onid.oregonstate.edu

Abstract—A perennial problem with online repositories of end-user programmers’ code is the low level of reuse, including in situations where existing code might aid in learning. This paper presents a formative study of middle-schoolers learning the Scratch animation environment, which revealed that they struggled to find short pieces of code that they could reuse directly, or from which they could discover language primitives (language instructions) to implement desired behavior. In response, we present a model and supporting prototype tool for clustering behaviorally similar code together, as a basis for helping end-user programmers to locate code. We conducted an empirical study confirming that our tool’s model for estimating code similarity does correspond well with programmers’ perceptions of code’s behavioral similarity. Future work will expand on these results by providing new search engines that help end-user programmers to find and reuse visual code from online repositories.

Keywords—end-user programming; reuse; Scratch

I. INTRODUCTION

Online repositories have become a standard means of letting end-user programmers share code with one another. Examples include repositories for web macros [3], Yahoo! Pipe mashups [28][18], and Scratch animations [30][34]. The last of these is, perhaps, the biggest existing repository, as it contains over 9 million animations published by over 6 million users.

Sharing code online offers a variety of potential benefits. For example, one of the benefits claimed by the Scratch team is that having a large collection of reusable examples helps children learn to program [30]. This happens as “children are learning the practice of remix at the Scratch website” [19]. “Remixing” is a synonym for what software engineers refer to as “whitebox reuse”: adapting code for a new purpose.

However, despite the plethora of code shared online by end-user programmers, evidence shows that little is actually reused in practice. For example, one study of Scratch found that only 5% of animations were ever remixed by users other than the original author [11]. This finding dovetails with another study that examined the CoScripter web macro repository, where only 4% of programs were reused by anybody other than the original programmer [32]. (Rates of 41%-53% are reported for code-containing libraries published by professional programmers [13][40].)

Why is reuse of end-user programmers’ code so low?

One problem is that much code on the Scratch repository is experimental and not really meant for reuse—56% of the animations examined in the study above were “trial” code apparently intended to try out Scratch rather than intended to produce useful programs [11]. Such code has the potential to clutter the repository and make it hard to find reusable code.

To counteract this problem, the repository lists “featured projects” selected by the site curators and those frequently remixed. At least in other repositories, such attention appears to result in an information cascade: frequently-reused code tends to be used even more often in subsequent time periods [32]. Hence, such repository features aid in finding reusable code but may concentrate user interest toward fewer animations, doing little to promote reuse beyond those listed.

Aside from these curated lists, another source of information about potentially reusable animations is the online forums, where users can post longer discussions and links to animations. There are currently 700,000 posts in the English-centric collaboration/trouble/show-off Scratch-focused forums. Yet even this pales in comparison to the millions of animations on the site: most animations are never mentioned in the forums.

This leaves the venerable search engine as the main means whereby users can find and reuse the vast majority of code on the repository. This search engine, as with other repositories, is keyword-based: the query is matched with animation titles, descriptions, and tags. Only animations that have matching text appear in search results. This contrasts with repositories for code written in the textual languages favored by professional programmers, for which the search engines can search the actual text of the code (in addition to the textual metadata searched by the Scratch search engine). Moreover, textual code in these languages implicitly explains its own purpose, since it often includes calls to extensive APIs; for example, the Java method to put an application in full screen mode is “setFullScreenWindow,” making it possible for a search engine to match the text (and surrounding metadata) of code that shows how to do “java full screen” [38].

Recent efforts to improve the Scratch search engine have met with limited success. Motivated by users’ complaints, the Scratch team recently prototyped a new search interface that innovates primarily in providing thumbnail and preview play options for search results [6]. Yet, notwithstanding these incremental improvements, users still “emphasized the need to improve the relevance of [search] results” [6].
We feel that the problem’s core is not just the lack of an effective way to represent search results (e.g., with thumbnail or preview), but that for end-user programmers’ visual code, there is little text to search on, limiting the effectiveness of keyword-based search. Though with 700,000 forum posts it is reasonable to expect there is some text about each kind of animation behavior somewhere on the site (as we have seen substantial redundancy about types of animation behavior in past studies of the forums [33]), most animations themselves lack extensive textual explanations of their own. A new paradigm is required for searching repositories of visual code.

To address these issues, the primary contribution of this paper is a technique for clustering visual end-user programmers’ code. This technique provides a starting point for a new approach whereby tools gather together pieces of code that have a similar purpose, annotating each cluster of code with text, and providing a searchable interface that guides users to browseable clusters (rather than directly to individual pieces of code). For example, a user who wanted to learn how to implement a bouncing effect in her animation might search for “bouncing,” which would lead to a few clusters of code. Each cluster would demonstrate a distinct way of implementing a bouncing behavior. Clicking on a cluster would then take her not only to code that had explicitly been labelled with the word “bouncing,” but also other code that lacked such labels but that was placed into the same cluster because of its behavioral similarity to the explicitly labeled code. Thus, she could potentially discover and reuse code that happened to lack searchable labels.

We anticipate that this technique will generalize to other situations where programmers need to learn how to write short, simple code consisting of a small enough number of operations that they can be visually represented as a cluster. In other words, it is aimed at learners who are past conditionals and loops, who are now expanding their mastery over the language, but who are not yet ready for high-level concepts such as design patterns or system architectures. In other words, we are targeting novices, principally end-user programmers who might lack other structured support for learning.

As secondary contributions, we present two studies. The first was a formative study that pinpointed barriers children encounter while learning to program, which focused us on the need to help users find short pieces of code that they could reuse or learn from. Motivated by this finding, we developed a model and prototype tool for clustering code based on the primitives used by the code. The second study evaluated this prototype to confirm that the tool’s internal model for estimating code similarity did match programmers’ assessment of code’s behavioral similarity. Future efforts will build on these results to develop tools that help users find reusable visual code in online repositories.

See the IEEE website for the rest of the paper