ABSTRACT

Code smells codify poor coding patterns known to degrade software quality. Block-based languages have proven to be a viable educational and end-user programming paradigm with increasing adoption across a broad spectrum of users and domains. This rising popularity of this programming paradigm calls for a serious look at the program quality written in block-based languages. While code smells in the context of text-based languages have been studied extensively, the research community lacks a comprehensive understanding of code smells in block-based software.

To address this problem, we present the results of a large-scale study of code smells prevalent in programs written in the highly popular Scratch programming language. We analyzed programs submitted to the public Scratch repository in 2016, considering a million programs altogether. We discovered interesting relationships between the prevalence of certain smells and the levels of proficiency of the programmers commonly introducing them. Our findings not only can help block-based programmers improve the quality of their software, but also establish the requirements for refactoring support in this programming domain.

Keywords

Software quality; Code smells; Block-based programming; Scratch; Introductory computing; CS education

In this work, we study block-based programming languages, which have been growing in popularity, providing highly effective tools for pedagogical pursuits and end-user, domain-specific development. The resulting increase in block-based software calls for a serious look at its quality. Unfortunately, while smells in text-based program have been thoroughly studied, code smells in block-based software remain poorly understood. This understanding is required to properly inform students and end-users about how to improve software quality by avoiding bad designs. Finally, we plan to follow up on our results by prioritizing our efforts in providing refactoring support for block-based software.

2. BACKGROUND AND RELATED WORK

Previous works identify “bad practices and habits” in Scratch programs, without explicitly identifying them as code smells. Meerbaum-Salant et al. [2] identified scenario-based scripts though intuitive, can lead to poor readability and maintainability. A preliminary study by Moreno [3] uses static analysis to identify 2 bad programming habits (i.e., code repetition and bad object naming) in a 100 Scratch projects.

Aivaloglou and Hermans [1] study over 250,000 projects Scratch programs to understand which types of blocks are used most frequently as well as analyze the subject programs for three code smells: large scripts, dead code, and duplicate block codes. Our work differs by not only considering generic code smells but also block-based specific code smells. We intend to study code smells for block-based languages comprehensively in terms of the number of smells considered (12 smells), and the number of subjects in our study’s dataset (∼1M)

3. APPROACH AND UNIQUENESS

Adopting generic code smells as is might not be sufficient or readily applicable to block-based languages due to the unique differences of block-based languages (high-level and domain-specific nature) and the users (the majority of programmers are non-professionals). In this study, we identify code smells commonly found in block-based programming languages in general and Scratch in particular. We also investigate the relationship of projects containing code smells to the programmers’ levels of expertise.

Overall, our methodology for identifying a catalog of code smells relies on personal observation, Scratch discussion forums, and those of other researchers and practitioners. We identify a total of 12 distinct code smells with a brief description of each smell as well as its prevalence in Table 1. Our study subjects comprise a collection of 1,066,308
code smells, as trouble spots for the majority of program
provide such support should focus on the more prevalent
support for block-based programmers, while the efforts to
gardless of the PCT levels of the program authors.
easier to work with). Certain smells are prevalent (US) re-
have been more careful to avoid them to make the program
less prevalent as projects grow in size as programmers may
developing
sidered. Overall, a small projects authored by programmers
ure 1. First, code smell are rare for programs created by
authors’ PCT level, shown graphically as a heatmap in Fig-
study can be summarized as follows:
Scratch projects. We define a set of metrics necessary in the
study. We develop a code smell analyzer operating at the
AST level, and address the scalability challenge by making
use of the Hadoop MapReduce on an HPC cluster.
We investigate the relationship between “smelly” programs
and their programmers’ computational proficiency. We rank
programmers by their computational thinking (PCT) scores:
1:basic, 2:developing, 3:proficient. Our PCT score extends
the prior computational thinking (CT) metrics [4], which an-
alyzes block-based programs on 7 computational concepts
e.g., data abstraction, flow control, etc.). PCT considers
multiple projects written by the same programmer to in-
crease the CT score accuracy.

4. RESULTS AND CONTRIBUTIONS

Table 1 shows how prevalent each smell is, ordered by the
most to least prevalent smells, while Table 2 shows the
distribution of the subject programs categorized by size and
PCT level. The insights we gained from the result of this
study can be summarized as follows:
The programming environment is partly responsible for
the top two Scratch specific code smells (BVS & UN), par-
ticularly, by having global as the default variable scope and
auto-generated generic name as default naming for program-
mapping elements. The lack of programming support to aid
programmers to identify Dead Code smells (UCB, UV, UC)
may have caused them moderately prevalent.
Block-based programs are plagued with Duplicate Code,
which confirmed the result of previous work[1].
We find interesting insights from the study of relation-
ship between projects containing smells and their project
authors’ PCT level, shown graphically as a heatmap in Fig-
ure 1. First, code smell are rare for programs created by
programmers with the PCT=1 regardless of project size
level. This indicates the programs these programmers cre-
ated are not complex enough to exhibit code smells we con-
considered. Overall, a small projects authored by programmers
of PCT=2 (developing status), are most prone to all code
smells. Certain smells (e.g. BVS, UN, and UC smells are
less prevalent as projects grow in size as programmers may
have been more careful to avoid them to make the program
easier to work with). Certain smells are prevalent (US) re-
gardless of the PCT levels of the program authors.
Overall, the findings suggest the need for refactoring tool
support for block-based programmers, while the efforts to
provide such support should focus on the more prevalent
code smells, as trouble spots for the majority of program-
mers. If block-based programmers can be better informed
about which smells to avoid when working with increasingly
complex projects, the overall quality of block-based software
is bound to improve.\footnote{This research is supported in part by the National Science
Foundation through the Grant 1134843.}

5. REFERENCES

we know: An exploratory study on the Scratch
repository. In Proceedings of the 2016 ACM Conference
on International Computing Education Research, ICER
’16, pages 53–61, New York, NY, USA, 2016. ACM.
Habits of programming in Scratch. In Proceedings of the
16th Innovation and technology in computer science
programming habits in Scratch: A preliminary study.
In Frontiers in Education Conference (FIE), 2014
thinking development assessment scores with software
complexity metrics. In 2016 IEEE Global Engineering
Education Conf. (EDUCON), pages 1040–1045. IEEE.

Table 1: Description of each smell and the percentage of code smells found in the sample program subjects

<table>
<thead>
<tr>
<th>Name</th>
<th>Abbrev</th>
<th>Freq (%)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broad Variable Scope</td>
<td>BVS</td>
<td>56.0</td>
<td>A variable with its scope broader than its usage does not tell which scriptable the variable belongs. Too many global variables clutter script palette and drop-down menus.</td>
</tr>
<tr>
<td>Uncommunicative Naming</td>
<td>UN</td>
<td>52.0</td>
<td>Generic naming started with Sprite or message (e.g. “SpriteZ” and “message1”) make programs unreadable</td>
</tr>
<tr>
<td>Long Script</td>
<td>LS</td>
<td>47.0</td>
<td>Long script (longer than 11 BLOCS) suggest inadequate decomposition and hinder code readability</td>
</tr>
<tr>
<td>Duplicate Code</td>
<td>DC</td>
<td>46.0</td>
<td>Repeated sequence of blocks regardless of block arguments is used as a way to reuse code</td>
</tr>
<tr>
<td>Unused Custom Block</td>
<td>UCB</td>
<td>29.0</td>
<td>A script definition of an unused custom block can be safely removed without affecting the program behavior</td>
</tr>
<tr>
<td>Unused Variable</td>
<td>UV</td>
<td>25.0</td>
<td>A variables is declared but never used anywhere in the program</td>
</tr>
<tr>
<td>Unreachable Code</td>
<td>UC</td>
<td>23.0</td>
<td>An unreachable script can be safely removed without affecting the program behavior</td>
</tr>
<tr>
<td>Hard-Coded Media Sequence</td>
<td>HCMS</td>
<td>13.0</td>
<td>A sequence of media elements are hard-coded as block arguments</td>
</tr>
<tr>
<td>Duplicate String</td>
<td>DS</td>
<td>10.0</td>
<td>Same string values are repeatedly used in multiple program locations</td>
</tr>
<tr>
<td>Unorganized Script</td>
<td>US</td>
<td>6.0</td>
<td>Similar event-based scripts are scattered around making the program hard to navigate</td>
</tr>
<tr>
<td>Unnecessary Workaround</td>
<td>UW</td>
<td>6.0</td>
<td>Use of polling of flag variables to direct control flow to recreate broadcast-receive mechanism</td>
</tr>
<tr>
<td>Extreme Fine-Grained Script</td>
<td>EFGS</td>
<td>2.0</td>
<td>Breaking up of functionally similar scripts into several small fine-grained event-based scripts</td>
</tr>
</tbody>
</table>

Table 2: Distribution of the program subjects

<table>
<thead>
<tr>
<th>Size</th>
<th>PCT=1</th>
<th>PCT=2</th>
<th>PCT=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size=small (10-100 blocks)</td>
<td>48,655</td>
<td>273,033</td>
<td>100,701</td>
</tr>
<tr>
<td>Size=medium (101-300 blocks)</td>
<td>1,082</td>
<td>36,059</td>
<td>40,584</td>
</tr>
<tr>
<td>Size=large (300-1,000 blocks)</td>
<td>103</td>
<td>5,901</td>
<td>13,673</td>
</tr>
</tbody>
</table>

Figure 1: The proportion of code smells across groups of programmers with different PCT levels (1-3) and project sizes(S:small, M:medium, L:large)