Smells in Block-Based Programming Languages

Felienne Hermans
Delft University of Technology
Delft, the Netherlands
f.f.j.hermans@tudelft.nl

Kathryn T. Stolee
North Carolina State University
Raleigh, NC, USA
ktstolee@ncsu.edu

David Hoepelman
Delft University of Technology
Delft, the Netherlands
D.J.Hoepelman@student.tudelft.nl

***Abstract*—Code smells were originally designed for object-oriented code, but in recent years, have been applied to end-user languages, including spreadsheets and Yahoo! Pipes. In this paper, we explore code smells in block-based end-user programming languages aimed at education. Specifically, we explore the occurrence of smells in two educational languages not previously targeted by smell detection and refactoring research: LEGO MINDSTORMS EV3 and Microsoft's Kodu. The results of this exploration show that object-oriented-inspired smells indeed occur in educational end-user languages and are present in 88% and 93% of the EV3 and Kodu programs, respectively. Most commonly we find that programs are plagued with lazy class, duplication, and dead code smells, with duplication smells being present in nearly two-thirds of programs in both languages.**

I. Introduction

End-user programmers are said to outnumber professional programmers three times over [1]. In their jobs, they face many of the challenges of professional developers, such as identifying faults, debugging, or understanding code written by someone else [2].

Similar to professional development is the longevity of the produced artifacts; the average lifespan of a corporate spreadsheet, for example, is five years [3]. During this long lifespan, end-user artifacts are modified, often by different people. These properties make them, like source code artifacts, vulnerable to *smells*.

Code smells are deficiencies or anti-patterns in source code. As outlined in the taxonomy of Fowler [4], smells pertained to object-oriented (OO) code. These OO professional programming languages were the focus for at least the first decade of code smell and refactoring research [5]. Recently, however, smells in end-user programming have also received attention in research, most notable structural smells in Yahoo! Pipes web mashups [6] and Excel spreadsheets [7]. Experiments in these and other end-user areas have shown that end-user programmers understand smells, and often prefer versions of their code that are non-smelly [8]–[10].

In this paper we broaden research on end-user code smells by examining block-based educational programming languages. Smells are particularly interesting in educational languages as the programs are often shared and remixed through online communities (e.g., Scratch, LEGO MINDSTORMS EV3, and Microsoft's Kodu all have online repositories for sharing). The presence of smells implies an opportunity for refactoring research to improve the quality of these programs and is especially important for shared programs.

To investigate the occurrence of smells in educational programs, we have gathered 44 programs written by children and online community members in two languages, EV3 and Kodu, and studied the occurrences of code smells. The results of this evaluation show that OO-inspired smells in fact occur in both end-user education languages: 88% of the EV3 and 93% of the Kodu programs contained at least one smell.

In EV3 and Kodu, the smells that we most commonly find are small abstractions (lazy class), duplication, and dead code, illustrating commonality across all end-user programming domains. The contributions of this work are as follows:

- Definitions for 11 end-user programming smells in LEGO MINDSTORMS EV3 and Kodu (Section III)
- Two case studies investigating end-user smells in educational programming languages: LEGO MINDSTORMS EV3 and Kodu (Section IV and Section V)

II. Background

In this work, we use smells previously explored in end-user programming languages to guide our exploration and analysis of programs written in the block-based educational languages. In previous research, code smells were applied to various other, end-user programming paradigms. Most prominently, research has focused on Yahoo! Pipes, a web mashup language and environment, and spreadsheets. Between those two domains, we observe few similarities in the code smells studied. Of the 11 smells covered between the two languages, only the *duplicate code*, *lazy class*, and *long method* smells were studied in both, illustrating the breadth of possible code smells, even in smaller, end-user languages.

Many languages have been developed with computer science education in mind, such as Scratch [11], Alice [12], Kodu [13], LEGO MINDSTORMS EV3 [14]. Each of these languages has its own unique structures, abstractions, and programming environment. One advantage of block-based languages, from the user's perspective, is that syntax errors are often prevented, allowing the user to concentrate on program design and logic. In this work, we focus on EV3 and Kodu, two popular block-based educational languages with public repositories for sharing programs, with which the authors have prior experience.

LEGO MINDSTORMS EV3: EV3 is the third iteration of the LEGO MINDSTORMS robotics line. It consists of several sensors, four motors and an ARM 9 "intelligent brick". The robotics kit comes with a control software package, which

978-1-5090-0252-8/16/$31.00

allows for visual programming of the brick, based on LabView. The software supports several basic constructs common to programming, including loops and conditionals, as well as more advanced features like parallel execution.

In addition to these programming concepts, users have the possibility to define 'MY BLOCKS' which are basically subroutines. MY BLOCKS cannot be programmed from scratch; they can only be created by replacing existing blocks with a new MY BLOCK, similar to 'extract method' present in most modern IDEs.

Kodu: Microsoft Research's Kodu is a visual programming language [13] and environment that allows users to create video games. Users create a world, add characters and objects, and then program each character or object individually (e.g., a turtle character, an apple object; we use character and object interchangeably). Variables are game scores or character properties. Scores are integers and global variables, identified by a color or letter. Character properties, such as color, glow color, expression and health, are like local variables.

The programming environment treats each character or object as an autonomous agent. Each object has 12 pages that can be programmed where the current page defines the current behavior of the object. The object's behavior can be changed by switching between pages to modify state and control flow. Each page contains a set of rules, and each rule is in the form of a condition and an action, which form a `when -- do` clause. The `when` is defined by a sensor (e.g., see, hear, gamepad input) and filters (e.g., apple, gamepad A button). The `do` is defined by an actuator (e.g., movement, shoot) and modifiers (e.g., missile, toward). Kodu's unique language can be used to express many fundamental concepts in computer science, such as boolean logic [15].

III. Definitions of Smells

We map OO smells to EV3 and Kodu, limiting our exploration to the 11 code smells found in other end-user languages. As is common in the other approaches, we define a loose mapping of OO concepts to the languages, shown in Table I.

In EV3 programming, MY BLOCKS resemble methods in source code in that they abstract one or more blocks and can be called multiple times. We investigate whether and how each of the smells in our catalog apply to EV3 programs. In Kodu, we consider pages of programming as analogous to methods or classes and scores and character properties as analogous to variables. These analogies are used in the smell definitions.

IV. Study

The aim of this paper is to explore **to what extent do code smells occur in block-based educational programs?** We analyze and count code smells in programs created in two domains: a data-flow language for programming robots, LEGO MINDSTORMS EV3 software, and an event-driven language for building and playing video games, Kodu. In this section, we describe the study artifacts and analysis.

A. Artifacts

For each language, we sought to explore programs created by two communities, children (the intended audience of the languages) and the community at large, which often includes children and adults. One threat to validity is that programs may not be representative of what all users create.

1) LEGO MINDSTORMS EV3: We gathered 17 programs from two data sources. The first is a weekly robotics club for children aged 8 to 13, which yielded eight programs.

Second, we solicited members of the EV3 group on Facebook to share programs. We refer to these as programs from the *community*. Nine programs come from this source.

2) Kodu: We gathered 27 programs from two data sources. For the first, we ran a workshop that introduced children to Kodu Game Lab in three 3-hour sessions. Children between the ages of 9 and 12 volunteered to participate, with parental consent, yielding 17 programs. For the second, we randomly sampled 10 programs from the public Xbox Live community.

B. Analysis

At a high level, EV3 is a robotics language and Kodu is inspired by robotics. As such, they share some high-level concepts, such as performing actions based on sensor values. This allows us to similarly tune the smell detection thresholds. For both languages, we count *long method*, a smell was counted when the MY BLOCK had 10 or more blocks or rules per MY BLOCK or page, as this is the maximum number that fits on a screen.

1) LEGO MINDSTORMS EV3: For the *lazy class* smell, we defined smelly as three blocks or fewer. For the *many parameters* smell, the use of four or more input values is smelly; typical programs use only one or two.

2) Kodu: For the *many parameters* smell, the use of four or more game scores. Thus thresholds was chosen because only a couple games used more than three game scores.

V. Results

In this section we explore the smell detection analysis in EV3 and Kodu. Table II shows, for each language and each code smell, the percentage of programs affected by each smell among programs created by children (i.e., the *Kids* column), those created by the community (i.e., the *Com* column), and overall (i.e., the *All* column). For example, 63% of the EV3 programs created by children have the *dead code* smell whereas only 11% of the EV3 programs created by the community have this smell. Among all EV3 programs we collected, 35% have the *dead code* smell.

Overall, 88% of the EV3 and 93% of the Kodu programs contained at least one smell. Coupled with the 81% of Yahoo! Pipes programs [9] and 42% of Excel programs [8] that were previously found to be smelly, this provides further evidence of the prevalence of smells in end-user programs.

TABLE I
SMELL DEFINITIONS FOR LEGO MINDSTORMS EV3 AND KODU

LEGO MINDSTORMS EV3	Kodu
Dead Code	
Unused MY BLOCKS can be present in the project without a warning being issued.	If there exists a page with programming such that there is no explicit path of control flow from Page 1 to it, it is unreachable and therefore dead.
Deprecated Interface	
This is a smell that does not apply, as there is only one version of the EV3 software.	Some language features may exist in early versions of Kodu, but as none of these features were ever deployed, this smell is not possible
Duplicate Code	
The same or very similar combinations of blocks occur	Two identical rules on a page, two identical pages for an object, or two rules on the same page with the same `when` clauses, but different actions.
Feature Envy	
While all variables within EV3 programs are global, they can be written in one MY BLOCK but read in a different one. If, in a MY BLOCK many variables are read that are written somewhere else, this the feature envy.	All global variables can be read and written by any character. If a certain character reads variables that have been written by another character, this could be an instance of the feature envy smell.
Inappropriate Intimacy	
Variables can be read in one MY BLOCK but written somewhere else. Two MY BLOCKS sharing multiple variables this way suffer from this smell.	If one character frequently checks the properties of another character, this could constitute inappropriate intimacy.
Lazy Class	
A MY BLOCK with few blocks can be considered lazy.	A character has no programming, it could be an instance of the lazy smell.
Long Method	
If a MY BLOCK has many blocks, it suffers from the long method smell.	A page with many rules may suffer from the long method smell.
Many Parameters	
MY BLOCKS with too many parameters are smelly, especially since parameters need to be connected with wires, leading to visual clutter.	A game can have 37 different global scores. Games that use many of these could be unnecessarily complex.
Message Chain	
Since MY BLOCKS can have both input and output parameters, so they can form a message chain, in which values are simply passed.	A chain of switches between pages without any logic on the page other than the jump.
No-op	
It is possible to combine blocks such that they do not actually contribute to the functionality of the program. For example, if a user stops the same motor twice, the second stop will be a no-op.	Jumping to a page with no logic is the logical equivalent of a null pointer. Alternatively, rules with `when` clauses but no `do` clauses do not perform any actions.
Unused Field	
MY BLOCKS can define parameters, but the user is not forced to use them, hence creating a code smell.	A global variable that is written to but not read is an instance of the the unused field smell.

A. LEGO MINDSTORMS EV3

On average, programs from the children had 2.5 smells each and programs from the community had 1.4 smells each, though the Facebook community most likely sent in nicely polished programs. Furthermore, the community programs were generally smaller (a median of 17 versus 23 total blocks) exhibiting more reuse as they more often used call to MY BLOCKS. Only three smells that do not occur in any of the EV3 programs: *deprecated interface* (not applicable), *inappropriate intimacy* and *message chain*.

Next, we discuss the most common smells within the EV3 programs, each appearing in at least one-third of the programs.

a) Duplicate Code: The most common smell we found is the *duplicate code* smell, which 65% of the programs suffer from. Duplication comes in various forms. Some of the programs use two motor blocks in a row, which could have been merged. Other programs exhibit duplication at a high level. For example, two MY BLOCKS were found that perform the exact same operation, but on a different motor. By adding the motor name as a parameter, this could have been implemented with one MY BLOCK.

b) Dead Code: The second most common smell is *dead code*. In 35% of the programs we found 'dead' MY BLOCKS. A large number of these smelly programs were created by children (63%) versus the community (11%).

Looking at the EV3 programming interface, it is not that surprising that users forget about dead MY BLOCKS, as the EV3 interface provided no information on which MY BLOCK is used and where. Even worse, users can delete MY BLOCKS that are still being called, without a warning being issued about this. After removal of a MY BLOCK in use, the program no longer compiles.

1) Summary: Overall, smells are found in 88% of the programs, with *duplicate code* and *dead code* appearing most frequently. Additionally, nearly one-fourth of the studied programs are affected by the *lazy class* (small abstractions) and *long method* smells, which may impact understandability. Hence we conclude that smells from OO are applicable to EV3.

TABLE II
SUMMARY OF SMELLS ACROSS EV3 AND KODU PROGRAMS

	EV3			Kodu		
	Kids	Com	All	Kids	Com	All
Dead Code	63%	11%	35%	6%	20%	11%
Deprecated	0%	0%	0%	0%	0%	0%
Duplicate Code	63%	67%	65%	53%	80%	63%
Feature Envy	38%	0%	18%	12%	40%	22%
Inappropriate	0%	0%	0%	6%	0%	4%
Lazy Class	50%	0%	24%	88%	70%	81%
Long method	13%	33%	24%	18%	40%	26%
Many Params	0%	11%	6%	0%	10%	4%
Message Chain	0%	0%	0%	0%	10%	4%
No-op	13%	11%	12%	35%	50%	41%
Unused Field	13%	11%	12%	24%	30%	26%
Any smell	88%	89%	88%	94%	90%	93%

B. Kodu

On average, the programs from children had 2.3 smells each and the programs from the community had 3.5 smells each, though we note that the programs from the community tend to be larger (e.g., an average of 101 rules vs. 42 rules).

Next, we discuss the three most common smells found in the Kodu programs, appearing in one-third or more of the sample.

1) Lazy class: The *lazy class* smell was the most common, appearing in 81% of the Kodu programs. This smell is intended to capture when a character is placed in the world but has no programming, and thus no behavior. In some cases, this may not be a smell at all, such as when characters are used as decorations only (e.g., trees and rocks). If we tune the lazy class threshold to allow for up to five lazy objects, the frequency of this smell drops to 59% but 85% of all Kodu programs remain smelly.

2) Duplicate Code: This smell was present in 63% of the programs. All instances were that of duplicate `when` clauses within a page. The prevalence of this smell shows a missed opportunity for consolidating the code. A higher percentage of this smell appeared in the programs from the community (80%) compared to the programs from children (53%). This may be due to differences in the program sizes or complexity since the community programs were over twice as big as the children programs, on average (101 rules vs. 42 rules).

3) No-op: This smell is present in 41% of the programs. All instances were rules without `do` clauses. This is probably the product of the rapid cycling between testing and developing observed in Kodu development [15]; since the clauses have no actionable logic, keeping them in the code does not impact the semantics of the program, though it may impact performance.

4) Summary: Overall, smells are found in 93% of the programs, with *lazy class*, *duplicate code*, and *no-op* appearing most often. Additionally, approximately 26% of programs are impacted by the *long method* and *unused field* smells.

C. Smells in End-User Programming Languages

As with code written by other end-user programmers [9], *duplicate code* is prevalent in EV3 and Kodu programs, affecting over 60% of the samples in both languages. The other two smells that were studied in Excel and Yahoo! Pipes, *lazy class* and *long method*, are present in at least 24% of the studied EV3 and Kodu programs, illustrating commonality across all end-user programming domains.

Dead code is more frequently found in EV3 and *lazy class* smells are more common in Kodu. The *no-op* smell is present in 41% of the Kodu programs. *Feature envy* was also prevalent across both languages, appearing in 18% and 22% of the EV3 and Kodu programs, respectively, though in EV3, all instances of the *feature envy* smell are in the childrens' programs. We observe that the occurring smells seem to be language-independent anti-patterns.

VI. RELATED WORK

Related to the current research are efforts on code smells within traditional languages; we start with the work of Fowler [4]. Within end-user programming, this paper builds upon two lines of related work. First, the work of Stolee and Elbaum that studied smells [9] and refactorings [6] in Yahoo! Pipes. The second direction is the work by Hermans *et al.* that also took OO smells as an inspiration for a number of smells in spreadsheets [7], [8]. Subsequently they described corresponding refactorings [16] and a tool that applies refactorings [17]. This final work generalized previous work by Badame and Dig [18]. Other previous smells detection work on other end-user environments studied performance smells in LabView, a visual language for system-design [10], [19].

Prior EV3 and Kodu research primarily explores language design [15], [20]. Recent studies on Scratch have shown that smells and clones are harmful [27] and common [28].

VII. CONCLUSION

This paper studies code smells for an end-user programming domain not previously studied in code smell research: educational languages. Specifically, we focus on two block-based languages, LEGO MINDSTORMS EV3 and Kodu. We present an evaluation of these code smells in the form of two case studies with 44 end-user programs created by children and the community. The results show that indeed many of smells previously studied in end-user languages also apply in the new domains, and *lazy class* (small abstractions), *duplicate code* and *dead code* occur frequently in these educational languages.

We observe that the original concept of code smells is applicable to educational end-user languages, of a quite different character than the textual languages aimed at professional developers for which the smells were originally defined. This underlines the applicability of these smells, and also warrants further research into issues end-users encounter while programming. For example, exploring the impact of code smells on children who use the educational languages or designing user-friendly refactoring tools for visual languages are potential future directions.

ACKNOWLEDGEMENTS

Special thanks to Stephen Coy and 'Instituut het Centrum'. This work is supported in part by NSF SHF-EAGER-1446932.

REFERENCES

[1] C. Scaffidi, M. Shaw, and B. A. Myers, "Estimating the numbers of end users and end user programmers," in *Proc. of VL/HCC '05*, 2005, pp. 207–214.

[2] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, M. B. Rosson, G. Rothermel, M. Shaw, and S. Wiedenbeck, "The state of the art in end-user software engineering," *ACM Computing Surveys*, vol. 43, no. 3, pp. 21:1–21:44, Apr. 2011.

[3] F. Hermans, M. Pinzger, and A. van Deursen, "Supporting professional spreadsheet users by generating leveled dataflow diagrams," in *Proc. of ICSE '11*, 2011, pp. 451–460.

[4] M. Fowler, *Refactoring: improving the design of existing code*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[5] T. Mens and T. Tourwé, "A survey of software refactoring," *IEEE Trans. Soft. Eng.*, vol. 30, no. 2, pp. 126–139, Feb. 2004.

[6] K. Stolee and S. Elbaum, "Refactoring pipe-like mashups for end-user programmers," in *Proc. of ICSE '11*, 2011, pp. 81–90.

[7] F. Hermans, M. Pinzger, and A. van Deursen, "Detecting and visualizing inter-worksheet smells in spreadsheets," in *Proc. of ICSE '12*, 2012, pp. 441–451.

[8] ——, "Detecting code smells in spreadsheet formulas," in *Proc. of ICSM '12*, 2012, pp. 409–418.

[9] K. T. Stolee and S. Elbaum, "Identification, impact, and refactoring of smells in pipe-like web mashups," *IEEE Trans. Soft. Eng.*, vol. 39, no. 12, pp. 1654–1679, 2013.

[10] C. Chambers and C. Scaffidi, "Smell-driven performance analysis for end-user programmers," in *Proc. of VLH/CC '13*, 2013, pp. 159–166.

[11] "Scratch," http://scratch.mit.edu/, February 2011.

[12] S. Cooper, W. Dann, and R. Pausch, "Alice: a 3-d tool for introductory programming concepts," in *CCSC '00: northeastern conference on The journal of computing in small colleges*, 2000, pp. 107–116.

[13] "Kodu language and grammar specification," http://research.microsoft.com/en-us/projects/kodu/grammar.pdf, August 2010.

[14] "LEGO Mindstorms," http://mindstorms.lego.com/, April 2015.

[15] K. T. Stolee and T. Fristoe, "Expressing computer science concepts through kodu game lab," in *Proc. of SIGCSE '11*, 2011, pp. 99–104.

[16] F. Hermans, M. Pinzger, and A. van Deursen, "Detecting and refactoring code smells in spreadsheet formulas," *Empirical Software Engineering*, pp. 1–27, 2014.

[17] F. Hermans and D. Dig, "Bumblebee: a refactoring environment for spreadsheet formulas," in *Proc. of FSE '14*, 2014, pp. 747–750.

[18] S. Badame and D. Dig, "Refactoring meets spreadsheet formulas," in *Proc. of ICSM '12*, 2012, pp. 399–409.

[19] C. Chambers and C. Scaffidi, "Impact and utility of smell-driven performance tuning for end-user programmers," *Journal of Visual Languages & Computing*, vol. 28, pp. 176–194, 2015.

[20] T. Fristoe, J. Denner, M. MacLaurin, M. Mateas, and N. Wardrip-Fruin, "Say it with systems: Expanding kodu's expressive power through gender-inclusive mechanics," in *Proc. of FDG '11*, 2011, pp. 227–234.

[21] M. MacLaurin, "Kodu: End-user programming and design for games," in *Proc. of FDG '09*, 2009, pp. 2:xviii–2:xix.

[22] M. B. MacLaurin, "The design of kodu: A tiny visual programming language for children on the xbox 360," *SIGPLAN Not.*, vol. 46, no. 1, pp. 241–246, Jan. 2011.

[23] A. Fowler and B. Cusack, "Kodu game lab: Improving the motivation for learning programming concepts," in *Proc. of FDG '11*, 2011, pp. 238–240.

[24] D. S. Touretzky, D. Marghitu, S. Ludi, D. Bernstein, and L. Ni, "Accelerating k-12 computational thinking using scaffolding, staging, and abstraction," in *Proc. of SIGCSE '13*, 2013, pp. 609–614.

[25] D. J. Barnes, "Teaching introductory java through lego mindstorms models," in *Proc. of SIGCSE' 02*, 2002, pp. 147–151.

[26] C. S. Hood and D. J. Hood, "Teaching programming and language concepts using legos," in *Proc. of ITiCSE '05*, 2005, pp. 19–23.

[27] F. Hermans and E. Aivaloglou, "Do code smells hamper novice programming: A controlled experiment on scratch programs," in *Proc. of ICPC '16*, 2016, to appear.

[28] E. Aivaloglou and F. Hermans, "How kids code and how we know: An exploratory study on the scratch repository," in *Proc. of ICER '16*, 2016, to appear.