A Catalog of OO-inspired End-User Smells, and an Application to LEGO MINDSTORMS EV3 and Kodu

David Hoepelman, Katryn Stolee, Felienne Hermans

ABSTRACT

In the workforce today, millions of people program without degrees or professional training in software development. These end-user programmers write code to design hardware circuits, combine web information, and make business decisions. Software engineering research into refactoring has traditionally focused on professionally used object-oriented programming languages, yet other domains and languages also suffer from code smells in need of refactoring.

TODO: Update this In this work, we explore recent research in three end-user domains and languages, spreadsheets in Microsoft Excel, web mashups in Yahoo! Pipes, and system design in National Instruments' LabVIEW. Through exploring the commonalities and differences among the domains, we 1) show how these end-user domains benefit from prior research on refactoring object-oriented languages, 2) discuss unique smell detection and refactoring opportunities for these domains, and 3) identify future opportunities for smell detection and refactoring research in these studied domains as well as other end-user programming domains.

1. INTRODUCTION

End-user programmers are said to outnumber professional programmers three times over [1]. These end-user programmers perform a wide variety of tasks within their organizations, ranging from building or maintaining applications to simple data manipulation in a spreadsheet. While performing these tasks, end-user programmers face many of the challenges of professional developers, such as identifying faults, debugging, or understanding code written by someone else [2]. Similar to code written by professional developers, end-user artifacts may have a long life-span, the average lifespan of a corporate spreadsheet being 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*.

Smells in end-user programming have been a topic of research over the past few years. Most notable are structural smells in Yahoo! Pipes web mashups [4] and Excel spreadsheets [5] Experiments in all these areas have shown that end-user programmers understand smells and often prefer versions of their code that are non-smelly [6, 11, 15]. Alleviating those smells can be achieved with refactoring.

Refactoring was first introduced as a systematic way to restructure source code and facilitate software evolution and maintenance. Martin Fowler later introduced the concept of code smells [9]. Refactoring code is often motivated by noticing a code smell, which signals the opportunity for improvement.

The taxonomy of smells outlined in Fowler's text pertained to object-oriented (OO) code, and professional programming languages were the focus for at least the first decade of refactoring and code smell research [10]. Like smell definitions refactorings have too been adapted and extended to other end-user programming paradigms, including web mashups [4, 11], Excel spreadsheets [3, 5, 12], and LabVIEW programs [6].

Considering the large number of end-user programmers, the longevity of their artifacts and the preference for non-smelly code, supporting end-user programmers in code smell detection and refactoring is valuable. The applicability of smells, originally created to detect weaknesses in source code, to other domains shows how powerful the concept is. Furthermore, studying the smells and refactorings in a fresh context provides new insight on how to use smells in software engineering and even suggests new types of smells. The contributions of this work are:

- Synthesis and catalog of object-oriented-inspired code smells and refactoring in end-user programs

- Discussion of unique smell detection and refactoring opportunities in the end-user domains

- Identification of future opportunities for smell detection and refactoring in end-user programming domains

2. BACKGROUND

In this section we briefly explore each end-user language targeted by prior refactoring research before presenting the relevant code smells in Section 3 and refactorings in Section **??**.

Excel.

Spreadsheets are very commonly used in businesses, from inventory administration to educational applications and from

Figure 1: Microsoft Excel 2013 showing a spreadsheet. Column B and C shows the typical mix of data and calculation. The formula of the selected cell B17 is visible right above the spreadsheet.

Figure 2: Example of a program in Yahoo! Pipes. It has five RSS feed data sources, each in a *Fetch Feed* module, feeding to a *Union* module that concatenates the feeds, a *Truncate* module that limits the number of items to 15 prior to the final *Pipe Output*.

scientific modeling to financial systems. Winston [13] estimates that 90% of all analysts in industry perform calculations in spreadsheets. Microsoft Excel is by far the most used, and therefore most studied, spreadsheet program, but other implementations exists and are similar.

In modern spreadsheet programs, a *cell* can contain a single *formula* which performs a calculation, and a table of cells is bundled in a *worksheet*; an example is shown in Figure 1. A *workbook* consist of a collection of worksheets. Formulas can reference other cells in the same or in different workbooks and worksheets.
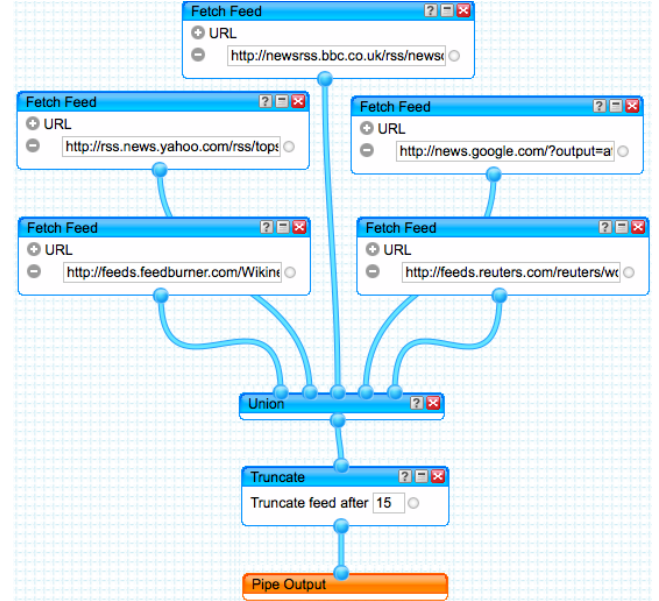
Yahoo! Pipes.

Yahoo! Pipes is a popular web mashup language and environment with which RSS feed information can be collected and combined from various sources. Figure 2 shows an example program. The boxes represent modules connected by wires. Abstraction is possible with *subpipe* modules, which allow a programmer to insert a different pipe as a subroutine, appearing like a standard module.

3. SMELLS IN END-USER PROGRAMS

Research into end-user language smells has had two approaches, which are not mutually exclusive. The first approach is to take existing smells for OO programming languages, usually those defined by Fowler [9], and transform them to be applicable to the end-user environment [4–6, 11, 15]. The second approach is to define smells tailored to the end-user environment. This can be done by interviewing experienced end-users to see which smells they perceive [6], by looking at user reports like forum or newsgroup posts [6, 16], or by analyzing publicly available repositories [4, 11].

This section provides an overview of different smells that researchers have found to be applicable to end-user artifacts

using both the above described approaches and proposes future directions for smell detection in these domains.

3.1 OO Smells in End-User Programs

We summarize the OO smells present in two end-user languages, Excel spreadsheets and Yahoo! Pipes mashups, in Table 1.

Overall, we often observe similarities in the code smells studied. For example, the *Duplicate Code* smell has been studied in both languages. Some smells, like the *Long Method* smell, have been studied in only one domain but are likely applicable in the other domain, these are marked with *. These smells present opportunities for future research, detailed in Section 3.2.

In this section we will detail the application of OO smells on specific end-user languages.

3.1.1 Excel

Hermans et al. [5, 15] analogize a workbook to a program, a worksheet to a class inside that program and a cell to a method and use this to transform nine of Fowlers smells. An example of this is the *Long Method* smell, which translates to the *Multiple Operations* smell because formulas with a large number of operations suffer from similar problems as long methods.

3.1.2 Yahoo! Pipes

Stolee and Elbaum [4, 11] treat a Yahoo! Pipes mashup as a class and each module as a method. Fields in a module are treated as parameters. Using this analogy, several OO smells were mapped to this language. The most common smell, appearing in nearly one-third of the 8,000 pipes studied, was *Duplicate Strings*, an instance of Fowler's *Duplicate Code*

Table 1: Code Smells in End-User Programs

OO Smell	Excel	Yahoo! Pipes
Feature Envy	Feature Envy [5]	Feature Envy *
Long Method	Multiple Operations [15]	Long Module *
Message Chain	Long Calculation Chain [15]	
Inappropriate Intimacy	Inappropriate Intimacy [5]	Inappropriate Intimacy *
Lazy class or Middle Man	Middle Man [5]	Unnecessary Abstraction [11]
Many Parameters	Multiple References [15]	
Duplicate Code	Duplicated Formulas [15]	Duplicate Modules, Duplicate String or Isomorphic Paths [11]
Dead Code	✗	Disconnected or Dangling Modules [11]
Unused Field	✗	Noisy Module [11]
No-op	Redundant Operations *	Unnecessary Module [11]
Use of Deprecated Interfaces	Deprecated Functions *	Deprecated Module or Invalid Source [11]

✗ : Not applicable due to the nature of the paradigm

* : Proposed smell, likely future opportunity not supported by prior work

⟨blank⟩ : Not discussed in this work, possible future opportunity

TODO: fill all empty spots, they should either be explicitly NA or possible

smell. *Duplicate Modules*, impacted nearly one-quarter of the pipes studied. Overall, 81% of the programs studied from the Yahoo! Pipes community had at least one smell.

Yahoo! can also *deprecate modules*, which can create similar problems to using deprecated or old versions of interfaces.

3.2 Future Opportunities for Smell Detection

Although not all OO smells have been studied in both domains, this does not mean they are not applicable, just that they have not been studies before. This is indicated in Table 1 with a *. For example, Feature Envy is not discussed in previous work on Yahoo! Pipes, but could apply when **TODO: Katie, how exactly?**. Similarly, redundant operations could occur in Excel, in fact, they do. A formula like SUM(A1+A2+A3), that we have seen in practice, exhibits the No-Op smell. While the occurrence of these smells falls outside of the scope of the current work, in this section we discuss the potential of generalizing some of the OO and domain-specific smell definitions to additional domains, thus addressing all *'s in Table 1.

3.2.1 Excel

Most of the smells studied in other end-user domains have been studied in Excel spreadsheets, but there is some future potential in the areas of redundancy and deprecation.

Currently all research in smells and spreadsheets has focused on Microsoft Excel. However, other spreadsheet software exists and operates on the same principles. Thus there is an opportunity to confirm that the identified smells apply in other spreadsheet software.

3.2.2 Yahoo! Pipes

The *Feature Envy* smell could apply when introducing abstraction. For example, if a pipe has several instances of the same subpipe module, this could be excessive use of another class.

When a program uses too much abstraction relative to the size of the pipe, it could suffer from *Inappropriate Intimacy* by depending too much on the implementation of the other class. In fact, in an empirical evaluation, programmers often preferred pipes without subpipe modules because they were easier to understand [11].

A *Long Module* smell could apply when a module has a large number of fields. For example, the *Fetch Feed* module, as in Figure 2, can hold one or more URLs. When the number of URLs makes the method so big it does not fit on the screen, this would likely impact the understandability of the pipe.

Drawing inspiration from the domain-specific *Inconsistent Formula* smell in Excel, identifying program patterns that are close, but not exactly the same, could identify missed opportunities for abstraction or errors in the mashup structure.

4. APPLICATION

In the above section, we have distilled a catalog of end-user smells from previous work on end-user smells. In order to demonstrate the applicability of our catalog, we apply it to a new domain: LEGO MINDSTORMS EV3software.

4.1 LEGO MINDSTORMS EV3

LEGO MINDSTORMS EV3is the third iteration of the LEGO MINDSTORMS robotics line. It consists of a number of sensors, four motors and a ARM 9 "intelligent brick". The robotics kit comes with a control software package, which

Figure 3: The interface of LEGO MINDSTORMS EV3 showing a simple line following program.
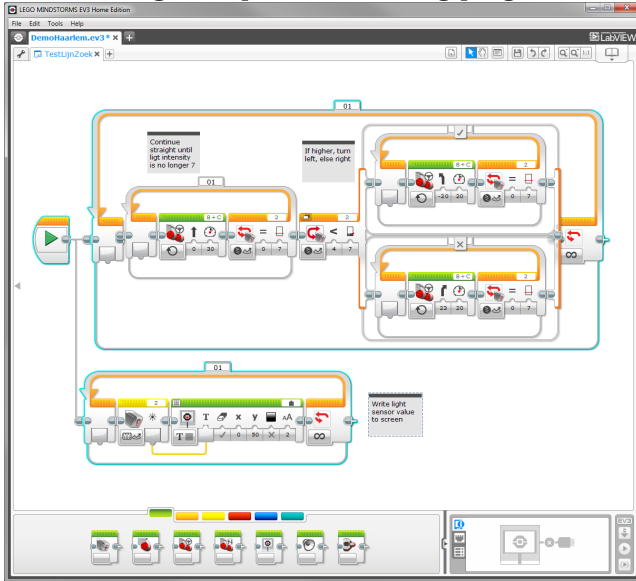
allows for visual programming of the brick. The software supports several basic constructs common to programming, including loops and conditionals, but also more advanced features like parallel execution. Figure 3 shows the user interface of the LEGO MINDSTORMS EV3 software with a program that makes the robot follow a black line, by steering left is the sensor value is too low and right if it is too high.

In addition to the programming concepts described above, users have the ability to define 'MY BLOCKS ' which basically are subroutines. MY BLOCKS may have up to 9 different input and one output parameter. MY BLOCKS cannot be created from scratch; they can only be created by selecting blocks in an existing program and creating a new block for them, together with a call to the newly created block, an action extremely similar to 'extract method' present in most modern IDEs. **TODO: can programs share the my blocks, have to look into that**

4.2 Smells and refactorings in LEGO MINDSTORMS EV3

In this subsection we describe how the smells would apply to LEGO MINDSTORMS EV3programs. As common in the other approaches, we define a loose mapping of OO concepts to the end-user language in question to be able to translate the smells. In LEGO MINDSTORMS EV3 programming, there are MY BLOCKS that contain a number of blocks, can be used multiple times and can use input and output. As such they resemble methods in source code, modules in Yahoo! Pipes and worksheets in spreadsheets. Based on this translation, we investigate whether OO smells could occur in LEGO MINDSTORMS EV3 programs too.

Feature envy While all defined variables within LEGO MINDSTORMS EV3 are global, they can be written in a certain MY BLOCK but read in a different one. If in a given MY BLOCK many variables are read that have

been written somewhere else, this might be an occurrence of the feature envy smell.

Long method If a MY BLOCK grows very large, it will no longer be easy to understand, diminishing the added value.

Inappropriate Intimacy As described above, variables might be read in one MY BLOCK but written somewhere else. If there are two MY BLOCKS sharing multiple variables this way, it might be better to combine them.

Lazy Class If a MY BLOCK is very small, for example, consisting of just one block, they do not add a lot of value, while making the program harder to understand, as a user has to navigate to the MY BLOCK to see what its function is.

Message Chain Because MY BLOCKS can have both input and output parameters, it is possible that they created a message chain in which values are continuously passed until they are used, while they could have been passed outside of the MY BLOCKS .

Many Parameters MY BLOCKS can have 9 different parameters, which could be considered too much for easy understandability.

Duplicate Code If the same, or very similar combinations of blocks occur, this would be the duplicate code smell.

Dead Code It is possible for programming blocks to be disconnected, but the interface clearly indicates this by making them gray. However, unused MY BLOCKS can be present in the project without a warning being issued. This is smelly as it makes the program unnecessarily large.

Unused Field As explained above, MY BLOCKS can define parameters. However, the user is not forced to use them, hence it is possible to define more parameters than used.

No-op It is possible to combine blocks in such a fashion that they do not actually contribute to the functionality of the program. For example, it a user stops the same motor twice, the second stop will be a no-op.

Use of Deprecated Interfaces This is a smell that does not apply, as, to date, there is only one version of the LEGO MINDSTORMS EV3 software and no blocks have been depreciated.

4.3 Study context

One of the authors of this paper runs a robotics club for kids where kids aged 8 to 13 program robots every week. Their programs have been collected as dataset for this paper. More specifically, we focus on two different projects within this set, RoboCup and Sumo. These two types of programs related to two different LEGO MINDSTORMS EV3 competitions: Sumo is a simple robot game in which robots have to 'sumo wrestle' each other: the robot that gets pushed out of the circular competition area first loses[1]. The Robocup programs were made to participate in the RoboCup Junior

---

[1] http://www.sugobot.com/

Rescue challenge[2], where robots have to first navigate part of the field by following a line and then look for a soda can and push it out of the field.

4.4 Findings

When investigating the programs, we found that they indeed sometimes suffered from smells. Table 2 presents an overview of the smells.

4.4.1 Duplicate Code

The most common smell we found in the nine programs is the Duplicate Code smell, which six out of nine program suffer from.

4.4.2 Dead Code

The second most common smell is Dead Code. In five out of nine programs we found MY BLOCKS no longer connected to the main program. This can pose a problem, as the EV3 environment compiles and transfers all programs and MY BLOCKS to the brick, causing the memory to be full quite quickly.

Looking at the LEGO MINDSTORMS EV3 interface, it is not that surprising that users forget about disconnected MY BLOCKS. The interface does not help users in understanding what blocks are used. If we look at the project properties screen, shown in Figure 5, one can see that there is no information about which MY BLOCK is used where. Even worse, we can delete connected MY BLOCKSwithout getting a warning, after which the program no longer compiles.

4.4.3 Lazy Class

Lazy classes, which we counted as MY BLOCKS having three or fewer blocks in them, are also common. Many of 'lazy MY BLOCKS' were relatively small, consisting of two or three blocks, or even 1 in some cases. You could say this is smelly, as understanding the MY BLOCK requires clicking it and that might not be worth it for small MY BLOCKS . However, the LEGO MINDSTORMS EV3 interface does not allow regular blocks to be named, but it does allow this for MY BLOCKS. So by making a MY BLOCK, users can express the intent of a block. As an example, consider the two blocks shown in Figure 5, where the above a regular block moving a motor, and the second is a call to a MY BLOCK with the same functionality. The first one just expresses what the robot must do, but the second one expresses the intent ('weg' meaning flee). By using the MY BLOCK, even with one block, the program gets easier to read.

5. RELATED WORK

This paper builds upon the extensive body of work related to code smells and refactoring in the end-user programming domain [3–6, 11, 12, 15, 22, 25]. For an extensive overview in object-oriented research, we refer the reader to the work of Mens and Tourwé [26].

5.1 Smells based on user input

In end-user programming environments user input and logic are often more closely linked than they are in general purpose languages. As such analyzing the input data as opposed to the logic can also be used to detect problems.

Figure 4: Two blocks representing the same functionality, one in a regular block and one using a my block.
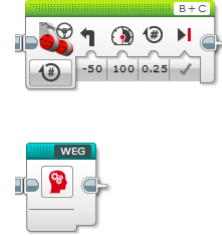
Figure 5: The project properties screen for the Sumo program 1, showing the three my blocks , but not indicating the my block 'draaien' is currently not called from the main program.
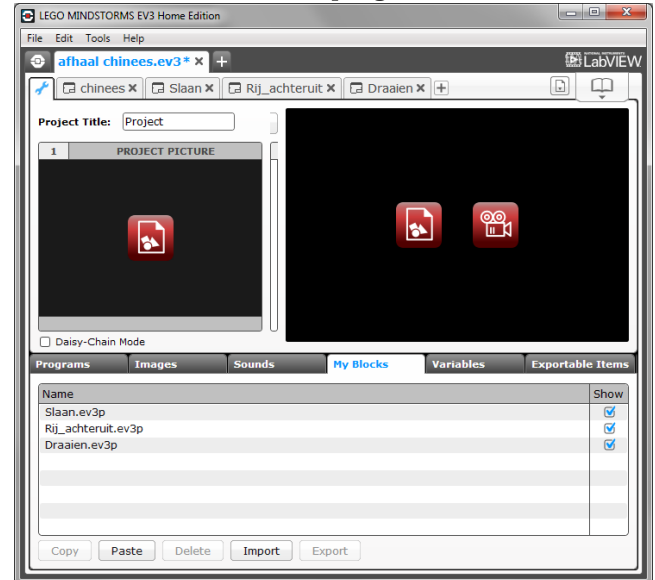
Table 2: Overview of the seventeen LEGO MINDSTORMS EV3 programs and the smells they exhibit

Name	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12	L13	L14	L15	L16	L17
Feature Envy	✓		✓	✓													
Long method					✓					✓				✓			✓
Inappropriate Intimacy																	
Lazy Class	✓		✓	✓		✓											
Message Chain																	
Many Parameters												✓					
Duplicate Code				✓	✓	✓	✓	✓	✓	✓	✓		✓			✓	✓
Dead Code	✓		✓	✓			✓	✓						✓			✓
Unused Field				✓										✓			
No-op			✓											✓			

Cunha et. al [18] claim to define code smells for spreadsheet, but for most of their smells they look at anomalies in the data and define these as smells. Examples of this are *Standard Deviation* which occurs if one assumes a normal distribution for a column in numeric values and the column contains values which fall outside two standard deviations. In more recent work, Barowy et. Al [27] take a more formalized approach which they label "Data Debugging". Their solution uses statistical analysis to find values with an unusually high impact on the calculated results in a spreadsheet, as such values are likely either very important or erroneous.

6. DISCUSSION

Based on the research and results for smell detection and refactoring in end-user programming domains, there are many directions for future work in the domains studied and other end-user domains.

6.1 Future Opportunities in Other Domains

End-user programming domains extend beyond spreadsheets, web mashups, and system designs. Stolee and Elbaum explore future opportunities for refactoring in educational programming languages [11]. Additional opportunities in educational languages exist in LEGO Mindstorms, which is based off the G language for LabVIEW, or in MAX/MSP, which are visual programming languages for music and multimedia. Other end-user programming domains that could benefit from smell analysis and refactoring are mathematical environments like MATLAB, Sage, and Mathematica.

In particular, the smells related to duplication and poor construction like *Long Method*, *Many Parameters* and *Dead Code* are prevalent in the three domains studied. These smells – and their respective refactorings – likely exist in other end-user programming domains, and likely hinder the understandability and maintainability of those programs. Worse even, these smells could lead to errors, and thus these smells are worthy of our attention.

6.2 Threats to Validity

The threats to validity of this work inherit the threats to validity of the original studies [3–6, 11, 12, 15, 16, 22, 25].

The three domains studied in this paper all happen to be dataflow languages, and the smells and refactorings may not generalize to other end-user programming domains (e.g., Scratch is OO-based).

In addition, we note that the authors of this work have pioneered smell detection and refactoring research in spreadsheets and Yahoo! Pipes, but are not involved with LabVIEW. Thus, the opportunities for future work in this area may not be complete.

7. CONCLUDING REMARKS

This paper presents an overview of the work in smell detection and refactoring for end-user programming languages. More specifically, it synthesizes work on Yahoo! Pipes, Excel and LabView. We explore commonalities between these works and identify opportunities for application to other end-user programming domains. As we move forward, we see many opportunities to explore additional smells and refactorings in the domains studied, in other end-user programming domains, and even in extending the findings to professional languages.

8. FUTURE WORK

8.1 Domain-Specific Smells

The end-user programming environments offer many opportunities to define smells based on user behavior or unique elements of the domain. In this section, we explore opportunities for new smells in end-user domains that extend beyond the OO-inspired smells in Section 3.1.

8.1.1 Excel

In spreadsheets data is often processed by having each record in a row and having a column with identical formulas to perform some calculation. This leads to the definition of the *Inconsistent Formula* smell, which occurs if a single, or small number, of cells contain a different formula while its neighbors or other cells in the row or column contain an identical formula. Interestingly this smell is already detected by Microsoft Excel which warns the user about it.

Another smell specific to spreadsheets is when a formula references an empty cell, which is often in error [18]. This is comparable to a null pointer in other languages and would be a runtime error, but because a spreadsheet contains both the input data and logic we can directly mark it as a smell.

8.1.2 Yahoo! Pipes

By exploring a large subset of the Yahoo! Pipes repository, Stolee and Elbaum identified a smell based on the

presence of broken data sources [11]. A reference to a broken data sources is similar to opening a non-existing file, which would be a runtime exception or error in most professional languages. Such exceptions are not in the Yahoo! Pipes language, which is the reason to mark it as a smell. Similar exploration was used to identify common programming practices, marking deviations from those practices as smells. This is similar to identifying smells as anti-patterns and could be extended to any language.

Acknowledgements

9. REFERENCES

[1] C. Scaffidi, M. Shaw, and B. A. Myers, "Estimating the numbers of end users and end user programmers," in *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2005)*, 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 Comput. Surv.*, vol. 43, no. 3, pp. 21:1–21:44, Apr. 2011. [Online]. Available: http://doi.acm.org/10.1145/1922649.1922658

[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] K. Stolee and S. Elbaum, "Refactoring pipe-like mashups for end-user programmers," in *33rd International Conference on Software Engineering*, 2011. [Online]. Available: http://www.neverworkintheory.org/?p=13

[5] 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.

[6] C. Chambers and C. Scaffidi, "Smell-driven performance analysis for end-user programmers," in *Visual Languages and Human-Centric Computing (VL/HCC), 2013 IEEE Symposium on*. IEEE, 2013, pp. 159–166.

[7] W. F. Opdyke, "Refactoring object-oriented frameworks," Ph.D. dissertation, Champaign, IL, USA, 1992, uMI Order No. GAX93-05645.

[8] W. G. Griswold and D. Notkin, "Automated assistance for program restructuring," *ACM Trans. Softw. Eng. Methodol.*, vol. 2, no. 3, pp. 228–269, Jul. 1993. [Online]. Available: http://doi.acm.org/10.1145/152388.152389

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

[10] T. Mens and T. Tourwé, "A survey of software refactoring," *IEEE Trans. Softw. Eng.*, vol. 30, no. 2, pp. 126–139, Feb. 2004. [Online]. Available: http://dx.doi.org/10.1109/TSE.2004.1265817

[11] K. T. Stolee and S. Elbaum, "Identification, impact, and refactoring of smells in pipe-like web mashups,"

[12] F. Hermans and D. Dig, "Bumblebee: a refactoring environment for spreadsheet formulas," in *Proceedings of the International Symposium on Foundations of Software Engineering '14*, 2014, pp. 747–750.

[13] W. Winston, "Executive education opportunities," *OR/MS Today*, vol. 28, no. 4, 2001.

[14] R. Panko, "Facing the problem of spreadsheet errors," *Decision Line*, vol. 37, no. 5, 2006.

[15] F. Hermans, M. Pinzger, and A. van Deursen, "Detecting code smells in spreadsheet formulas," in *Proc. of ICSM '12*, 2012.

[16] S. Badame and D. Dig, "Refactoring meets spreadsheet formulas," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 2012, pp. 399–409.

[17] M. Fisher and G. Rothermel, "The euses spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms," in *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4. ACM, 2005, pp. 1–5.

[18] J. Cunha, J. P. Fernandes, H. Ribeiro, and J. Saraiva, "Towards a catalog of spreadsheet smells," in *Computational Science and Its Applications–ICCSA 2012*. Springer, 2012, pp. 202–216.

[19] W. Tansey and E. Tilevich, "Annotation refactoring: Inferring upgrade transformations for legacy applications," *SIGPLAN Not.*, vol. 43, no. 10, pp. 295–312, Oct. 2008. [Online]. Available: http://doi.acm.org/10.1145/1449955.1449788

[20] T. v. Enckevort, "Refactoring uml models: Using openarchitectureware to measure uml model quality and perform pattern matching on uml models with ocl queries," in *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, ser. OOPSLA '09. New York, NY, USA: ACM, 2009, pp. 635–646. [Online]. Available: http://doi.acm.org/10.1145/1639950.1639959

[21] F. Hermans, M. Pinzger, and A. van Deursen, "Detecting and refactoring code smells in spreadsheet formulas," *Empirical Software Engineering*, pp. 1–27, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10664-013-9296-2

[22] 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, to appear.

[23] Y. Y. Sui, J. Lin, and X. T. Zhang, "An automated refactoring tool for dataflow visual programming language," *ACM Sigplan Notices*, vol. 43, no. 4, pp. 21–28, 2008.

[24] I. Balaban, F. Tip, and R. Fuhrer, "Refactoring support for class library migration," *SIGPLAN Not.*, vol. 40, no. 10, pp. 265–279, Oct. 2005. [Online]. Available: http://doi.acm.org/10.1145/1103845.1094832

[25] K. T. Stolee, J. Saylor, and T. Lund, "Exploring the benefits of using redundant responses in crowdsourced evaluations," in *Proceedings of the 2nd International*

Workshop on CrowdSourcing in Software Engineering, ser. CSI-SE 2015, 2015, to appear.

[26] T. Mens and T. Tourwé, "A survey of software refactoring," *Software Engineering, IEEE Transactions on*, vol. 30, no. 2, pp. 126–139, 2004.

[27] D. W. Barowy, D. Gochev, and E. D. Berger, "Checkcell: data debugging for spreadsheets," in *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*. ACM, 2014, pp. 507–523.

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

[29] 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.

[30] K. T. Stolee and T. Fristoe, "Expressing computer science concepts through kodu game lab," in *Proceedings of the 42nd ACM technical symposium on*

Computer science education, ser. SIGCSE '11, 2011, pp. 99–104.

[31] G. Leshed, E. M. 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*, ser. CHI '08, 2008, pp. 1719–1728.

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

[33] J. Schumacher, N. Zazworka, F. Shull, C. Seaman, and M. Shaw, "Building empirical support for automated code smell detection," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '10. New York, NY, USA: ACM, 2010, pp. 8:1–8:10. [Online]. Available: http://doi.acm.org/10.1145/1852786.1852797