Refactoring Code Smells in Google Spreadsheets: A User Response Study

- Introduction
- What are code smells?
- Refactoring as a Solution in Spreadsheets
- Multiple operations code smell and possible solutions
- Suggesting solutions to the user
- How we displayed our refactoring suggestions?
- Showing the suggested LET function
- What goes behind the back in our implementation?
- Conclusion

Introduction

This project aims to study the response of users when notified of a code smell in their Google Spreadsheet code and provide them with an option to refactor the formula into a better version.

A Google Spreadsheet add-on was created specifically for this purpose, which works on the 'multiple operations' smell and its refactoring.

What are Code Smells?

Code smells are indications of potential problems in software code. They are usually small design flaws or violations of good programming practices that can accumulate over time.

- Decreased readability and reusability
- Increased complexity
- Technical debt and reduced maintainability
- Slow development and maintenance
- Increased risk of bugs

Code smell example : Conditional complexity

codesmell.cpp

C: > Users > divya > OneDrive > Pictures > codesmell.cpp > ...

```cpp
#include <iostream>

int main() {
  int age;

  std::cout << "Enter your age: ";
  std::cin >> age;

  if (age < 18) {
    std::cout << "You are not yet an adult" << std::endl;
  } else if (age < 65) {
    std::cout << "You are an adult" << std::endl;
  } else {
    std::cout << "You are a senior citizen" << std::endl;
  }

  return 0;
}
```

refactored.cpp

C: > Users > divya > OneDrive > Pictures > refactored.cpp > ...

```cpp
#include <iostream>

void printAgeCategory(int age) {
  if (age < 18) {
    std::cout << "You are not yet an adult" << std::endl;
  } else if (age < 65) {
    std::cout << "You are an adult" << std::endl;
  } else {
    std::cout << "You are a senior citizen" << std::endl;
  }
}

int main() {
  int age;

  std::cout << "Enter your age: ";
  std::cin >> age;

  printAgeCategory(age);

  return 0;
}
```

Refactoring as a Solution in Spreadsheets

Refactoring means restructuring or rewriting code to improve its design, readability, and maintainability without changing its functionality.

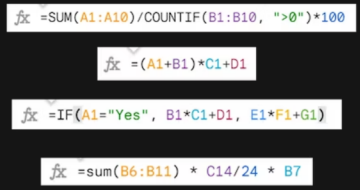
- Easier to read/understand and modify/debug
- Reduced cost of maintenance
- Better code quality, cleaner code

Multiple operations code smell

A formula with many different operations can be :

- Harder to understand
- Limited space to view a formula, causing long formulas to be cut off

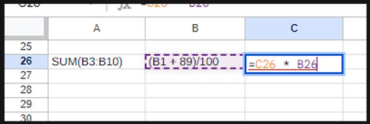
One of the most common code smell in Spreadsheets



Possible solutions:

- Conversion to a LET function
- Division of the expression over multiple cells

```
=LET(var_1, sum(B3:B10)  , var_2,  (B1 + 89) / 100, var_1 * var_2 )
```



Both of these solutions help in:

- Simplifying the expression into smaller parts
- Reducing the cognitive load while revisiting this formula expression for debugging/modifying it
- Mapping different parts of the expression to appropriate semantic labels

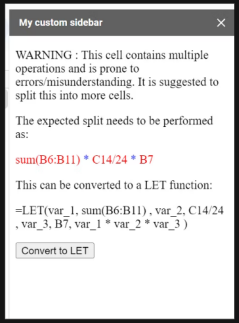
Suggesting solutions to the user

- Need to design appropriate algorithms which determine the units in which the formula must be divided
- These splits must be such that each part carries a significant semantic to it but, at the same time, is not over-simplified.
- Predicting a single such split accurately is a difficult task, and giving users multiple split suggestions to choose from can be a bad idea because :
  - Can be overwhelming and distracting
  - Higher cognitive overload and decision paralysis
  - Decision fatigue, decreased productivity and satisfaction

How we displayed our refactoring suggestions?

- Every time more than one major operation is detected in the currently active cell, the user is given a sidebar notification (a nearly un-obtrusive form of notification)

- Provided user with a single suggestion to consider
  - Described the problem and suggestive action in short
  - Highlighted the positions to split at in the formula
  - An option to convert their expression into a LET formula
  - The variable names in the LET formula are editable



Showing the suggested LET function

Working prototypes were implemented for displaying the LET function in a more readable, flexible manner

WARNING : This cell contains multiple operations and is prone to errors/misunderstanding. It is suggested to split this into more cells.

The expected split needs to be performed as:

sum(B6:B11) * C14 / 24 * B7

This can be converted to a LET function:

=LET(var_1, sum(B6:B11) , var_2, C14 / 24 , var_3, B7, var_1 * var_2 * var_3)

Convert to LET

WARNING : This cell contains multiple operations and is prone to errors/misunderstanding. It is suggested to split this into more cells.

The expected split needs to be performed as:

sum(B6:B11) * C14 / 24 * B7

This can be converted to a LET function:

```
=LET(
 variable 1 , sum(B6:B11),
 variable 2 , C14/24,
 variable 3 , B7,
 formula
)
```

Convert to LET

WARNING : This cell contains multiple operations and is prone to errors/misunderstanding. It is suggested to split this into more cells.

The expected split needs to be performed as:

sum(B6:B11) * C14 / 24 * B7

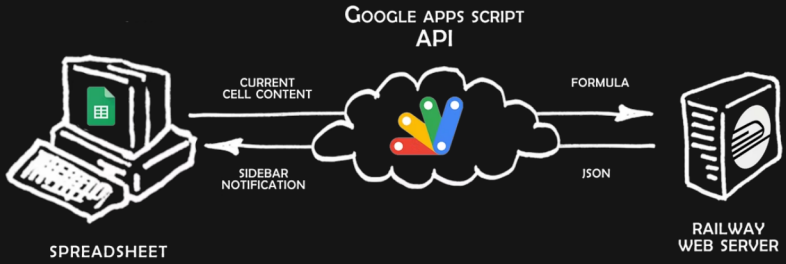
Map the following subexpressions to appropriate variable names

```
sum(B6:B11) --> variable 1
C14 / 24 --> variable 2
B7 --> variable 3
Final Formula:
```

Convert to LET

What goes behind the back in our implementation?

- Spreadsheet add-on made with Google Apps Script API.
- We hosted a nodeJs server on railway.app and connected it to the Apps Script API that reads the formula from the active spreadsheet cell and sends a request to the server.
- The server takes in the formula and outputs the code smell info, suggested split, and LET function.



Conclusion

This project can enhance code quality in Google Spreadsheets, saving time and reducing errors.

Our work's limitations:

- Only the "Multiple Operations" code smell was addressed in our implementation; other code smell types still require attention.
- Our add-on requires an internet connection and cannot be used offline due to calls to a web server.

Future project work will involve:

- Adding the ability to detect additional code smell types and present their refactoring suggestions.
- Comparing and evaluating ways to present refactoring suggestions.
- Creating ideal design rules for add-ons that are user-friendly and flexible.

Our project aims to inspire more research in real-time code smell detection and refactoring in various programming languages and platforms, including the design guidelines for such tools.

Thank you !