Automated Refactoring of Legacy Software Systems to Current Best Practices

Isaac Griffifth
isaacgriffith@gmail.com

Stephani Scheilke
stephani.scheilke@gmail.com

Objective: To restructure software in order to increase the understandability, reusability, and maintainability as a means to quantify the rationality of a program. In effect, unveiling the potentially lost subjective knowledge and processes embedded into the original code by the Software Engineers.

1. Introduction

This work presents an attempt to provide empirical approach to:

- Analysis of metrics that indirectly measure the behavior concepts of maintainability, reusability, and understandability of software projects.
- Software Engineering issues dealing with automated refactoring the understandability of software by multiple engineers

2. Refactoring

- Refactoring is provided by manipulating entities within a graph structure representing the content of an entire source code base of an application
- These operations implement a defined technique which modifies the structure of the software without changing its overall function.
- The refactoring operations used are:
  - Move Method
  - Move Field
  - Pull Up Method
  - Pull Up Field
  - Collapse Hierarchy
  - Push Down Field
  - Push Down Method
  - Move Class

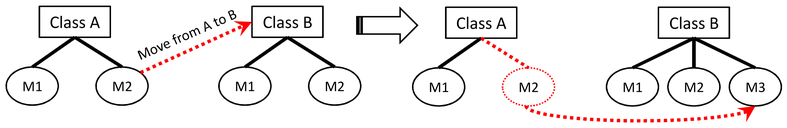


Figure 1. Move Method Refactoring Technique

- Code Smells: Provide a set of heuristics by which we can measure the effectiveness of refactoring over an entire software project's source code base.
- Combining the Quantified Heuristics of Code Smells with the metrics below we can adequately refactor a system and verify it meets current best practices.

3. Metrics and Measures

Maintainability	• Measures the amount of cohesion, coupling, and complexity of software components • Metric: Coupling Between Objects, Cyclomatic Complexity and Lack of Cohesion in Classes
Reusability	• Measure of the cohesion components of the software • Metric: Coupling between Objects and Lack of Cohesion in Classes
Understandability	• Measures the complexity of each component of the software • Metric: Cyclomatic Complexity

4. Basic Process

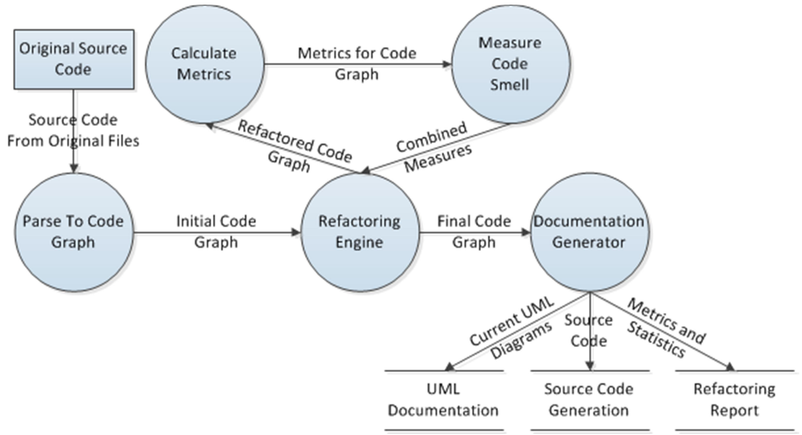
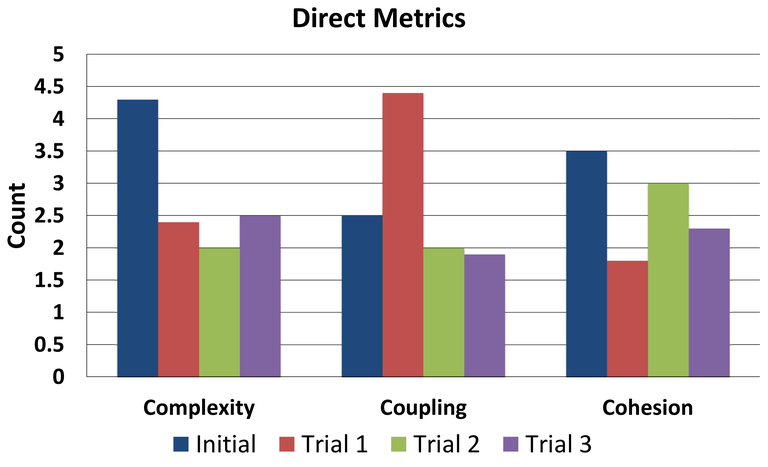
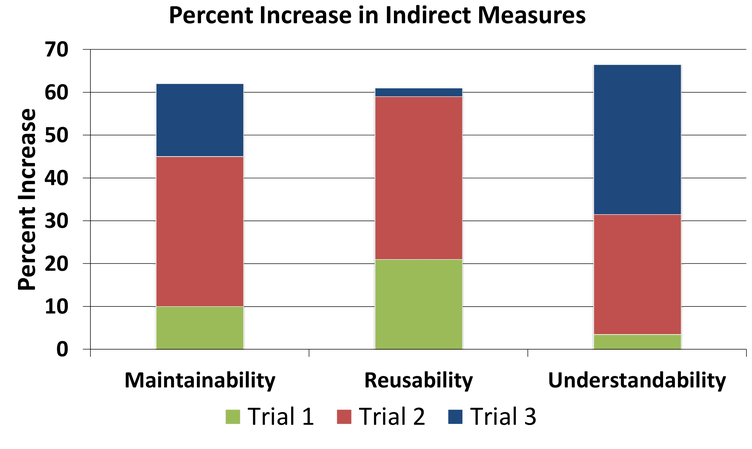


Figure 4. Basic Process Data Flow Diagram

5. Results





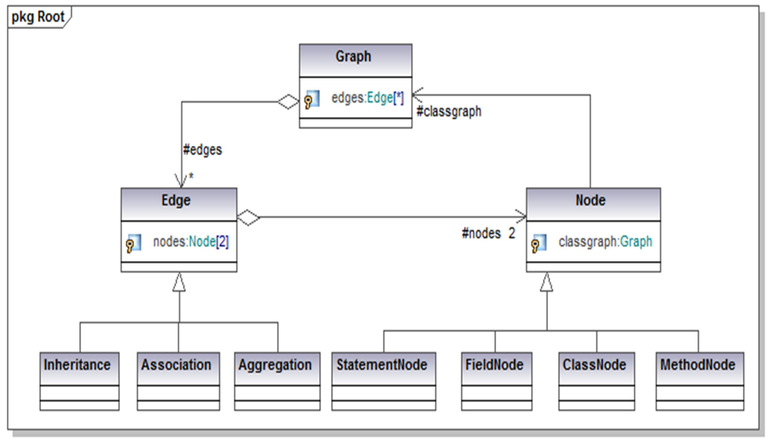


Figure 3. Example Resultant UML Documentation

6. Mathematical Implications

- Provides the quantification of the behavior of program execution in the form of metrics, which provide a tool to analyze the necessary qualities of software to ensure that it complies with current best practices
- Quantification of the notion of Code Smells providing an overall measure of the effect of refactoring on source code across a project.

7. Philosophical Implications

- Introduces an example of a new paradigm in Computer Science
- Produces an example of Dark Programing
- Provides empirical evidence towards a more defined understanding of program darkness

8. Conclusions