Dark Programming and The Quantifying of Rationality and Understanding in Software

Isaac Griffith
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 answers to:

- Rationality of emergent programs
- The link between rationality and understanding of programs
- Analysis of metrics to measure these qualities over the scope of large 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

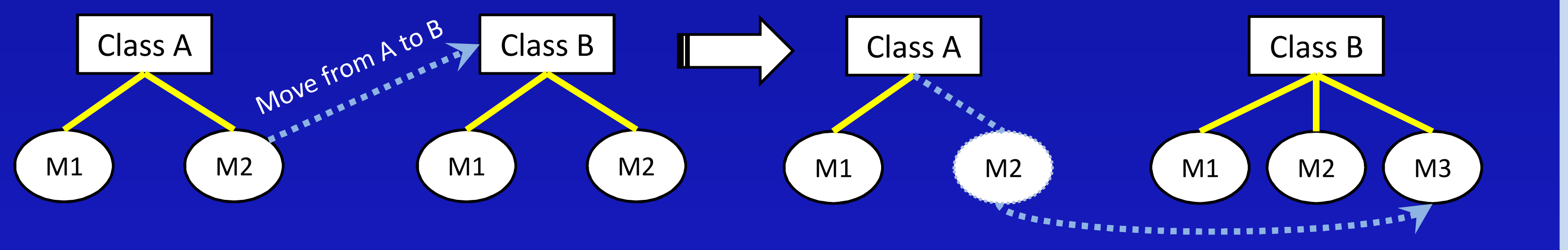
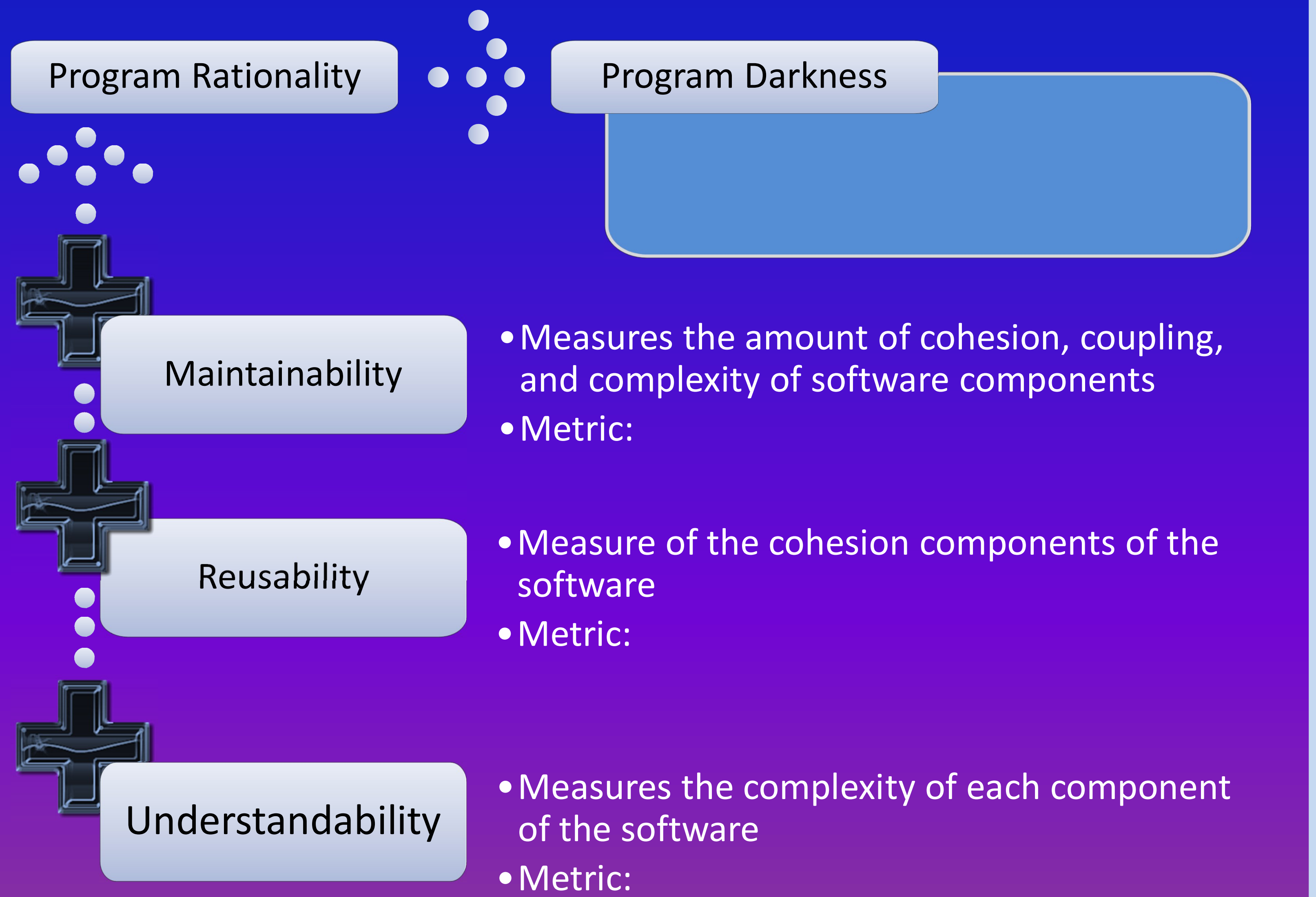


Figure X Example Refactoring Graph Operation

3. Metrics and Measures



- Measures the amount of cohesion, coupling, and complexity of software components
- Metric:

- Measure of the cohesion components of the software
- Metric:

- Measures the complexity of each component of the software
- Metric:

4. Basic Process

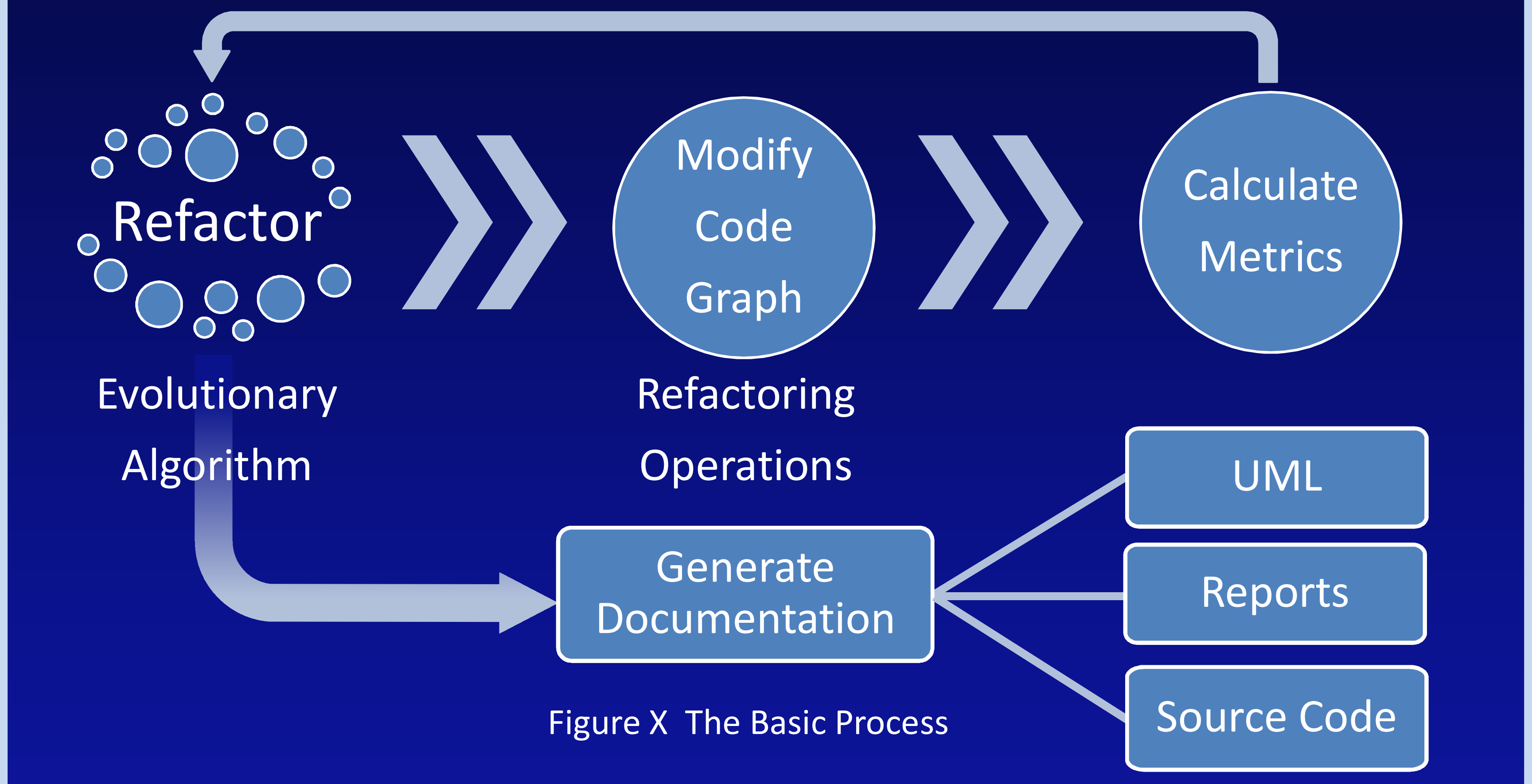
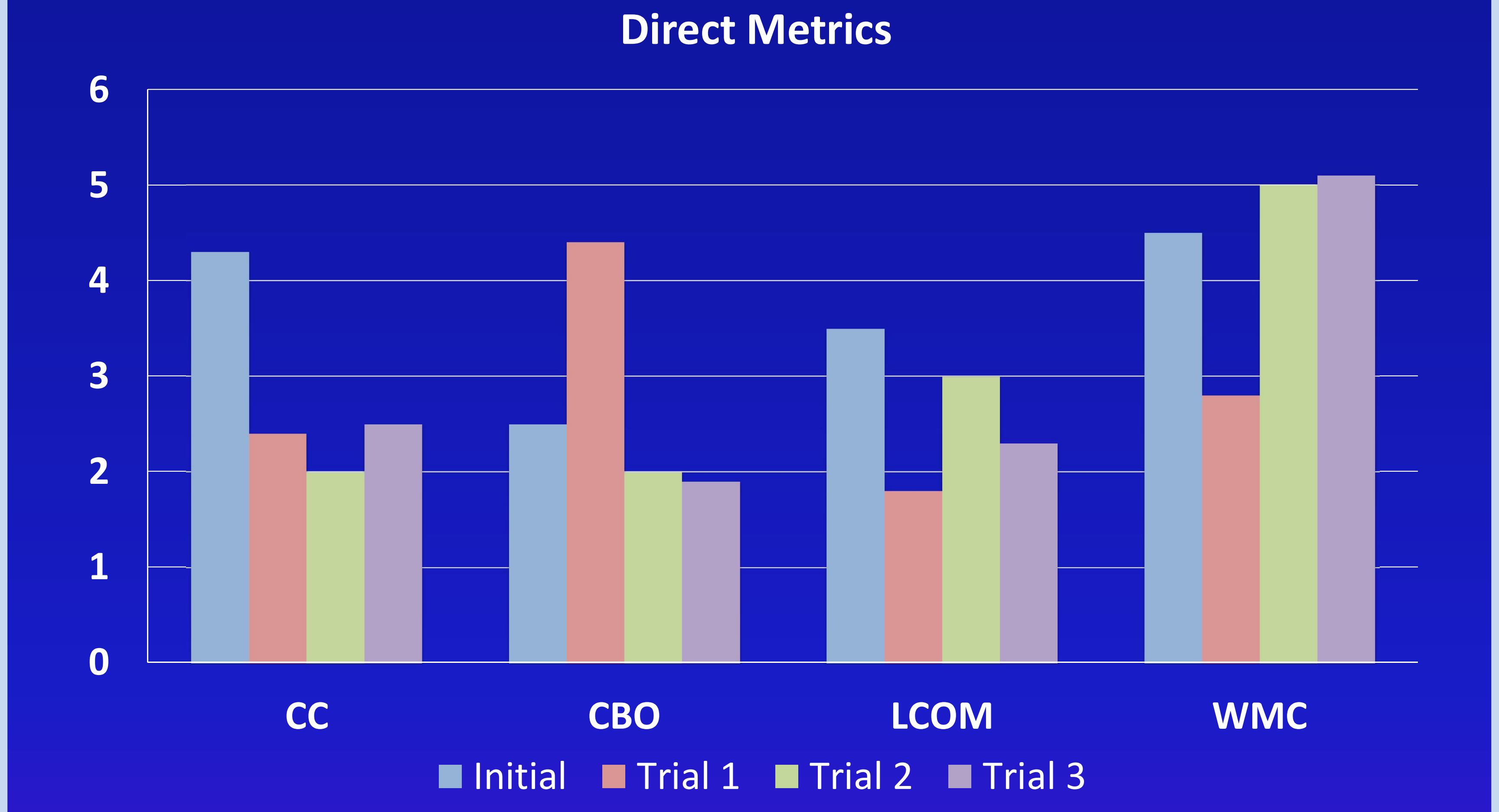
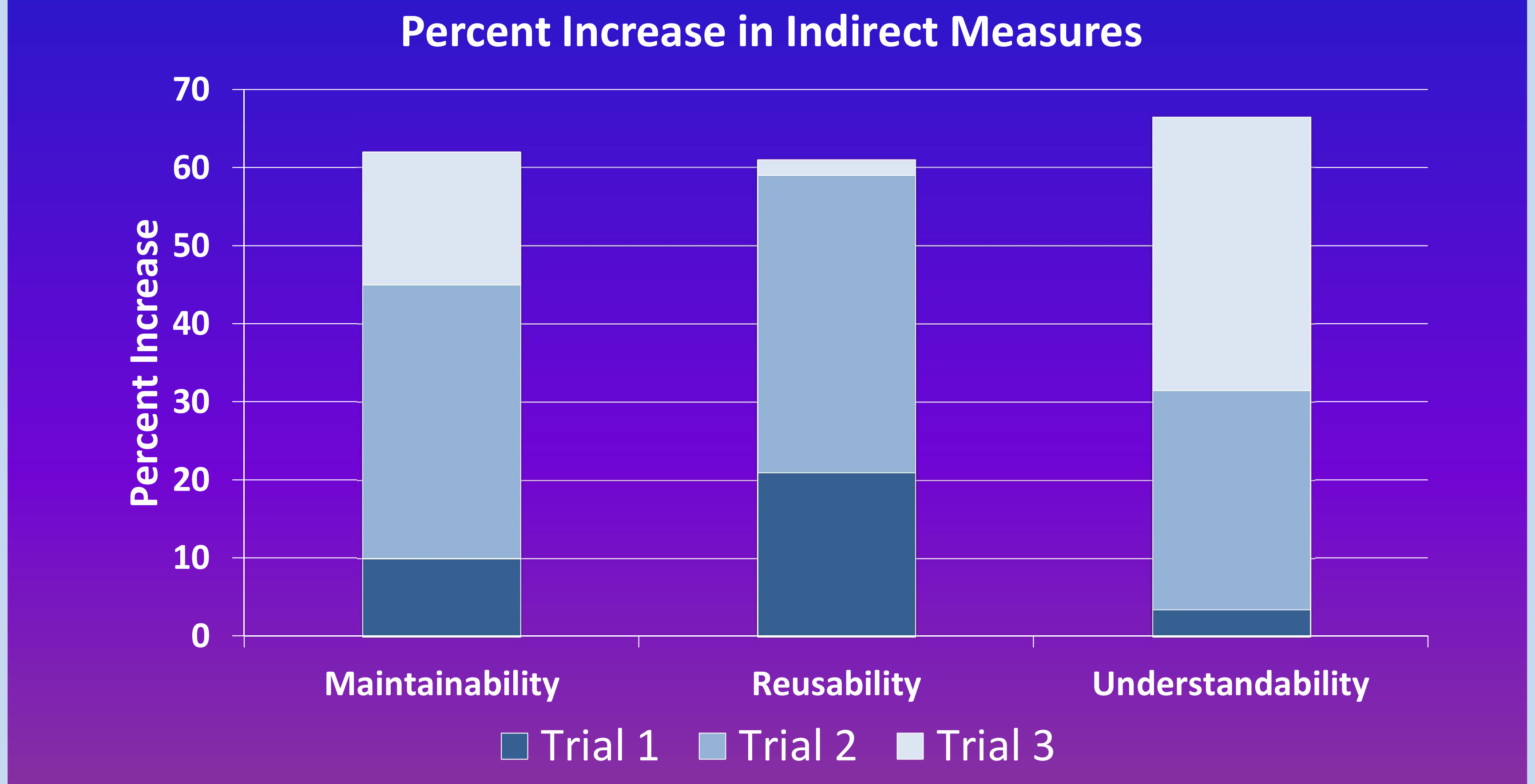


Figure X The Basic Process

5. Results





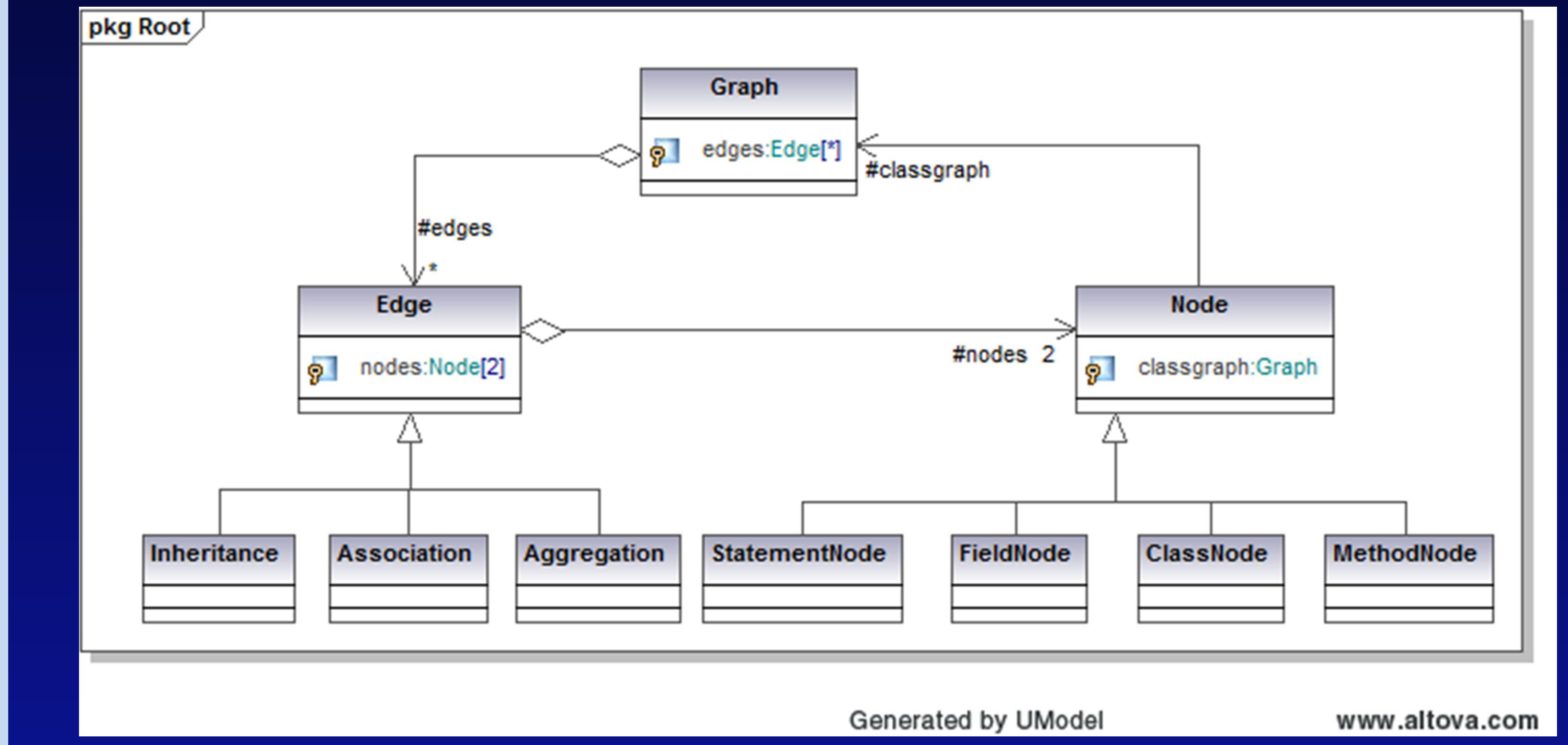


Figure X Example Resultant UML Documentation

6. Mathematical Implications

7. Philosophical Implications

8. Conclusions