An Experience Report Related to Restructuring OODesigner: A CASE Tool for OMT

Taegyun Kim

Pusan University of Foreign Studies

Nacer Boudjlida

University Henri Poincare Nancy 1

Contents

1. Introduction

- In 1994, we started this project with two types of goals:
  - <u>Product goals:</u>

    to make a CASE tool for OMT
  - <u>Process goals:</u>

    to practice OO design and Implementation

    to learn about OO paradigm

Problems found in 1996

- We felt that class architecture is ill-designed.
- Maintaining OODesigner became hard.
- Enhancing functional modeler and dynamic modeler has two alternatives:
  - to continue to enhance it with version 1.x
  - to totally restructure version 1.x, and to enhance it with a better architecture

2. Restructuring OODesigner

- Restructuring Goals
- Restructured Items
- Restructuring Process
- Benefits Gained
- Metrics Comparisons
- Lessons

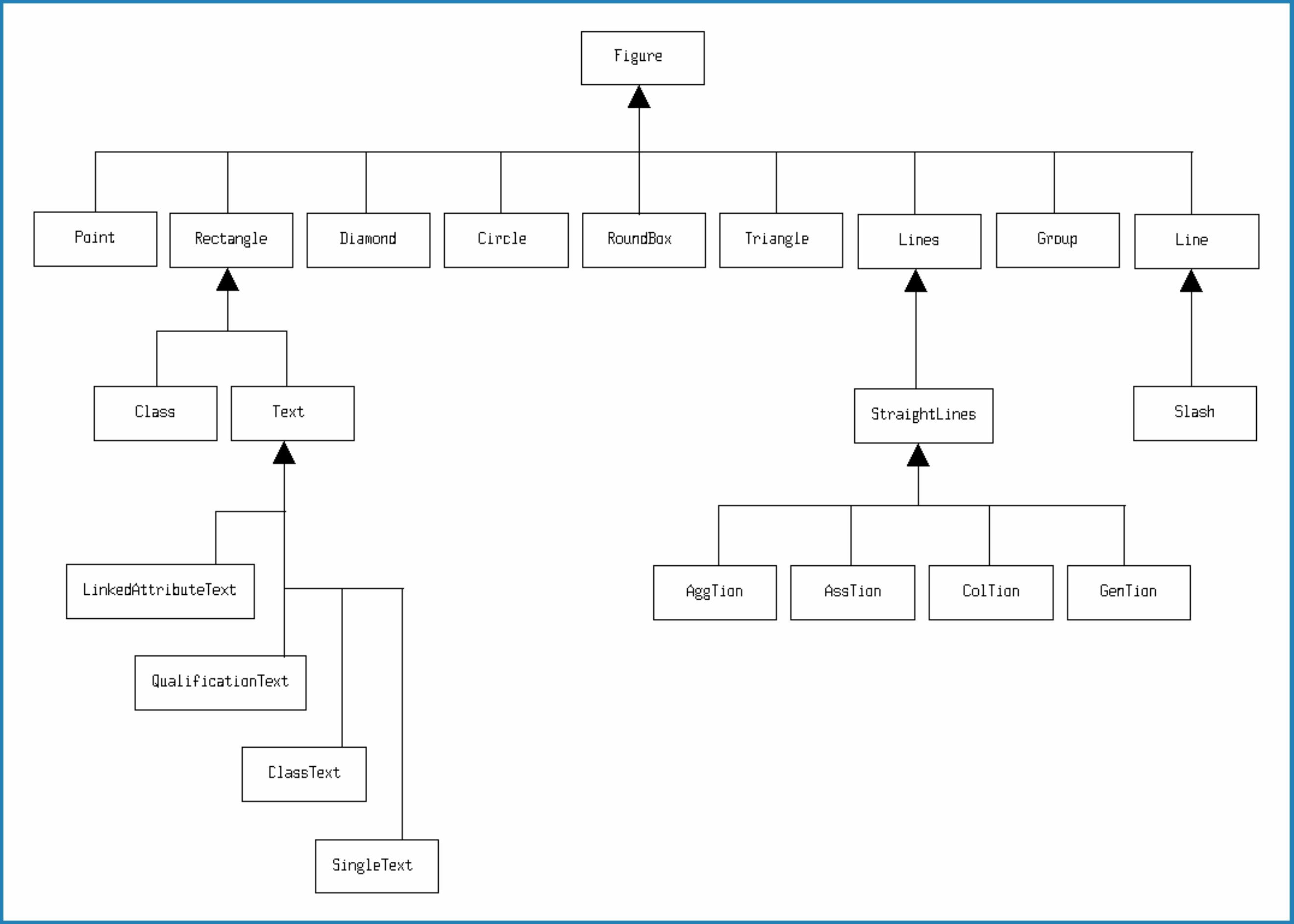
Restructuring Goals

- Make classes application independent
- Reorganize class inheritance tree
- Make control structure loosely coupled
- Localize platform dependencies
- Minimize duplication of code
- Minimize global members
- Increase robustness of code

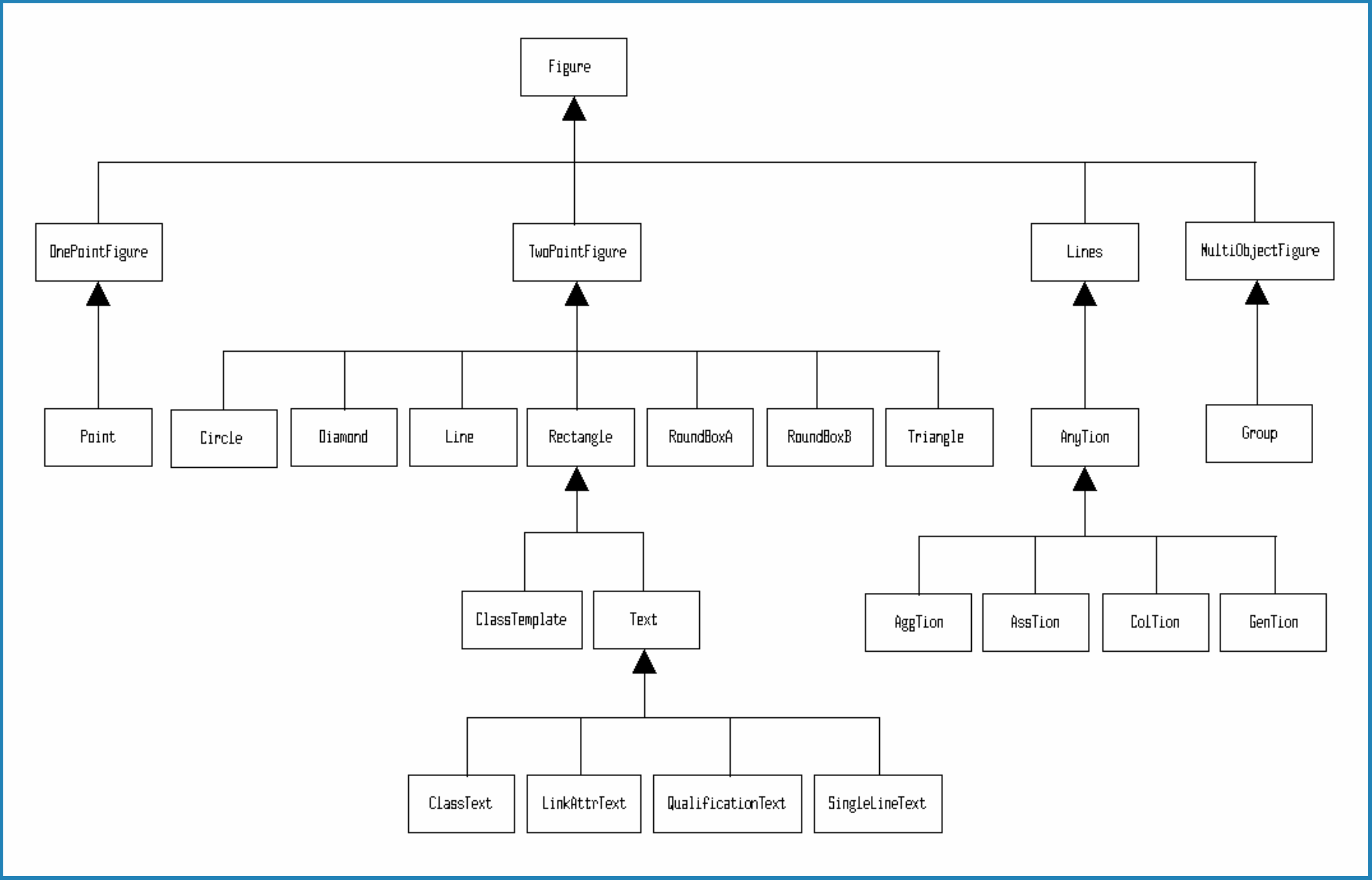
Restructured Items

- Reducing duplicated members
- Encapsulating GUI components
- Intensive use of dynamic binding
- Encapsulating global members
- Making destructors more complete
- Adopting our coding convention

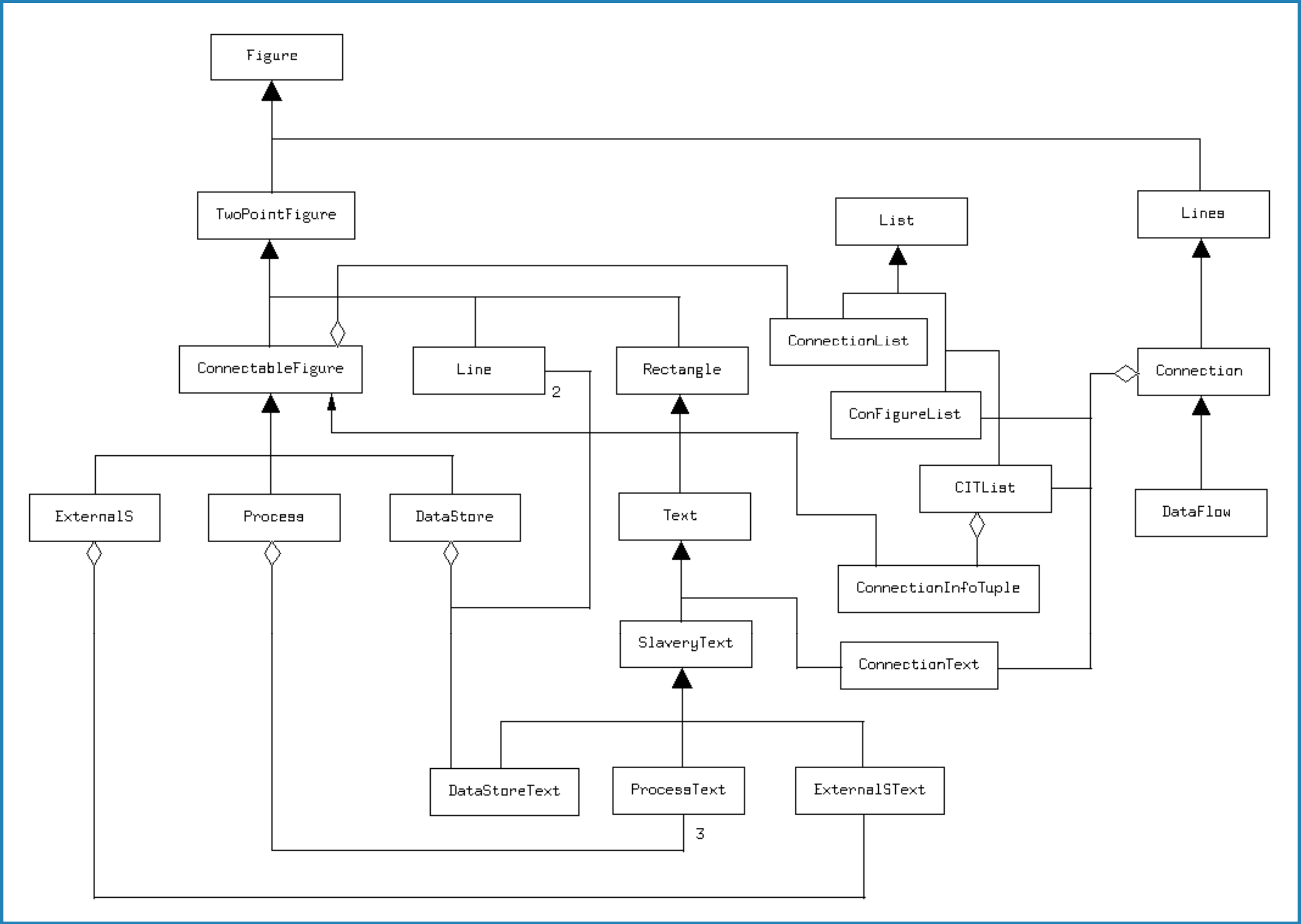
Typical Changes From (Figure 1)



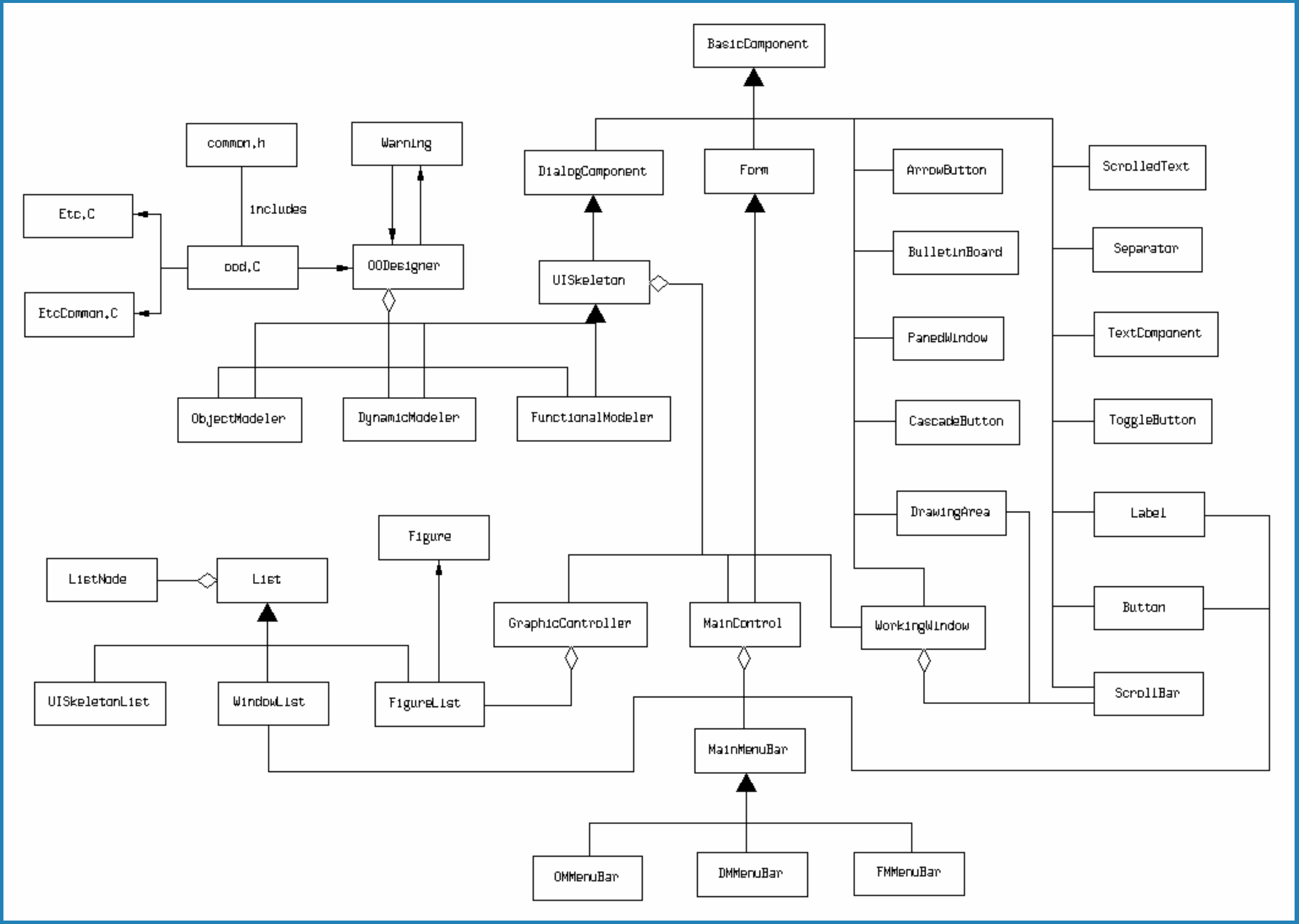
Typical Changes To (Figure 2)



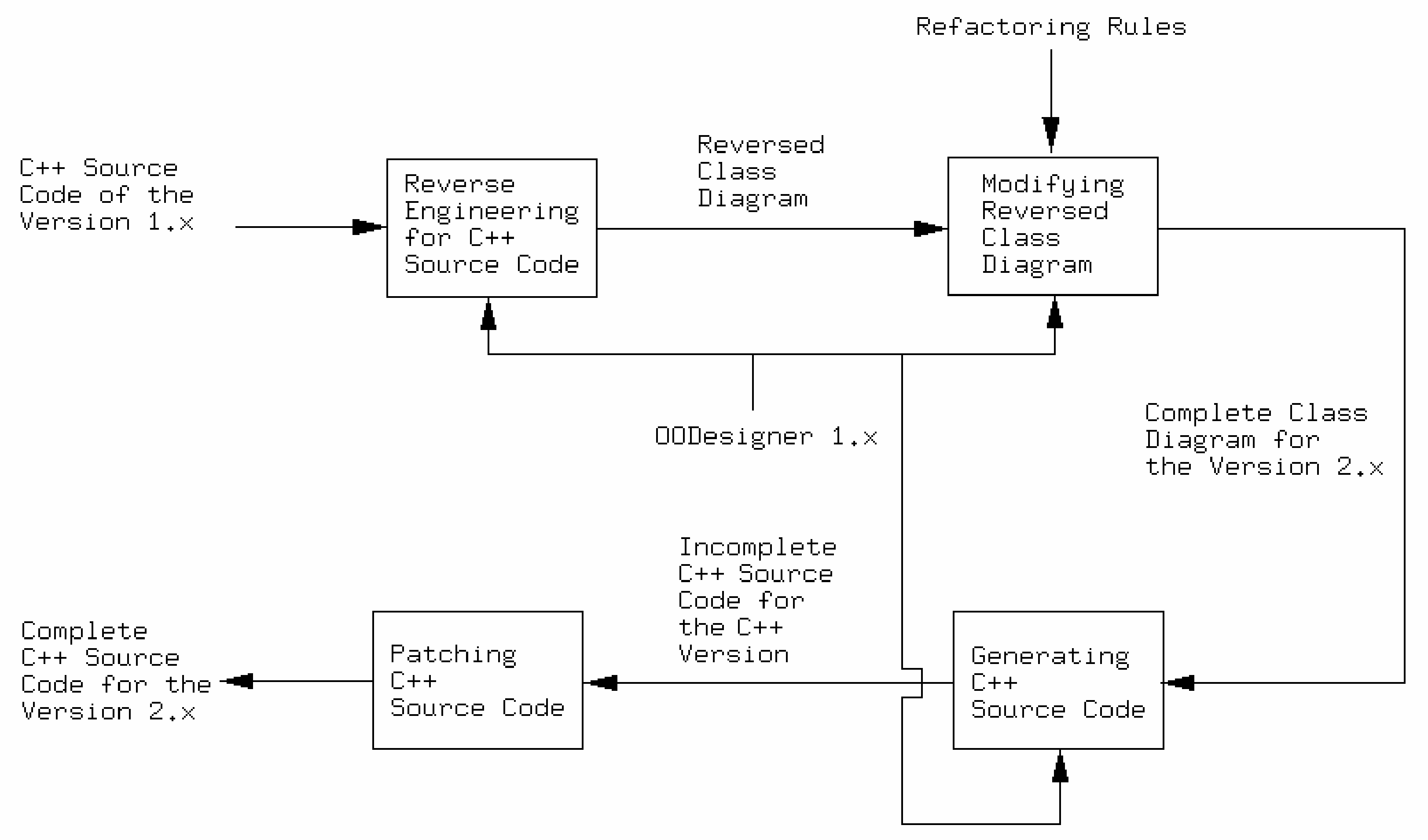
New Module for Functional Modeler



Main Module for OODesigner



Restructuring Process


Refactoring Rules
C++ Source Code of the Version 1.x
Reverse Engineering for C++ Source Code
Reversed Class Diagram
Modifying Reversed Class Diagram
OODesigner 1.x
Complete Class Diagram for the Version 2.x
Incomplete C++ Source Code for the C++ Version
Complete C++ Source Code for the Version 2.x
Patching C++ Source Code
Generating C++ Source Code

Benefits Gained

- Version 2.x became:
  - Easier to modify, enhance and understand
  - More flexible, stable and reliable
  - Platform independent
  - And finally easier to maintain
- We are currently developing Java version and PC version with minimal efforts.

Metrics Comparisons(Table 1)

- “Make classes as small as possible.”
  - Complexity is reduced.

    conditional statements and loop statements
  - Weighted Method per Class is decreased.

    less application specific
  - Depth of Inheritance Tree is increased.

    more reusable

Metrics Comparisons

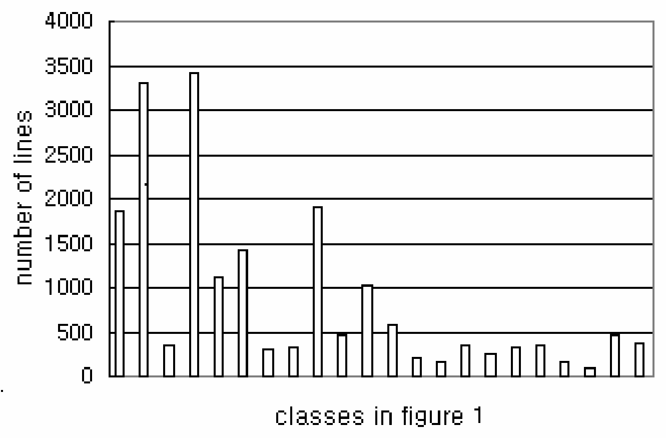
- Number of Children is increased.

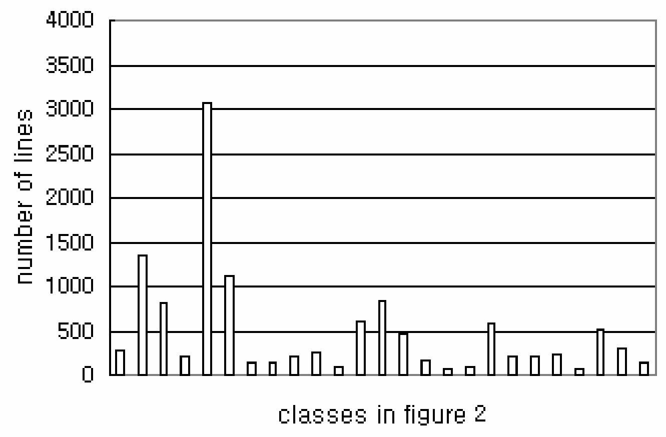
  more reusable
- Coupling between Object Classes is increased.

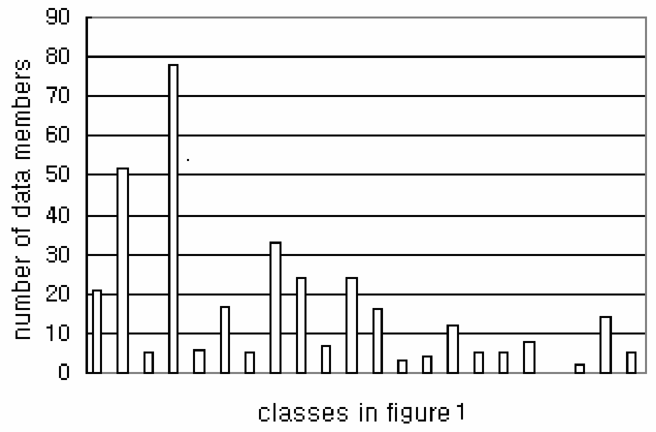
  strange result
- Violating the Demeter's Law is increased.

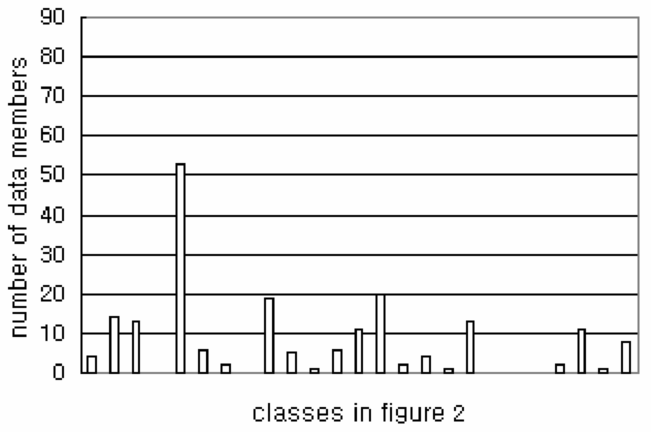
  We tried to keep from increasing the number of member functions.

Metrics Comparisons between Figure 1 and Figure 2









Lessons Learned

- Technical perspective:
  - keep the class size and member size small
  - use inheritance “aggressively”
  - use dynamic binding “aggressively”
  - use good naming convention

Lessons Learned

- Managerial perspective:
  - Inevitable failure is expected for the first OO project.
  - Synergistic effect of combining OO methodology, language and tool is great.
  - Don’t hesitate to restructure any troublesome OO legacy system.
  - Well-designed OO software makes us happy.

3. Current State of OODesigner

- UNIX version:

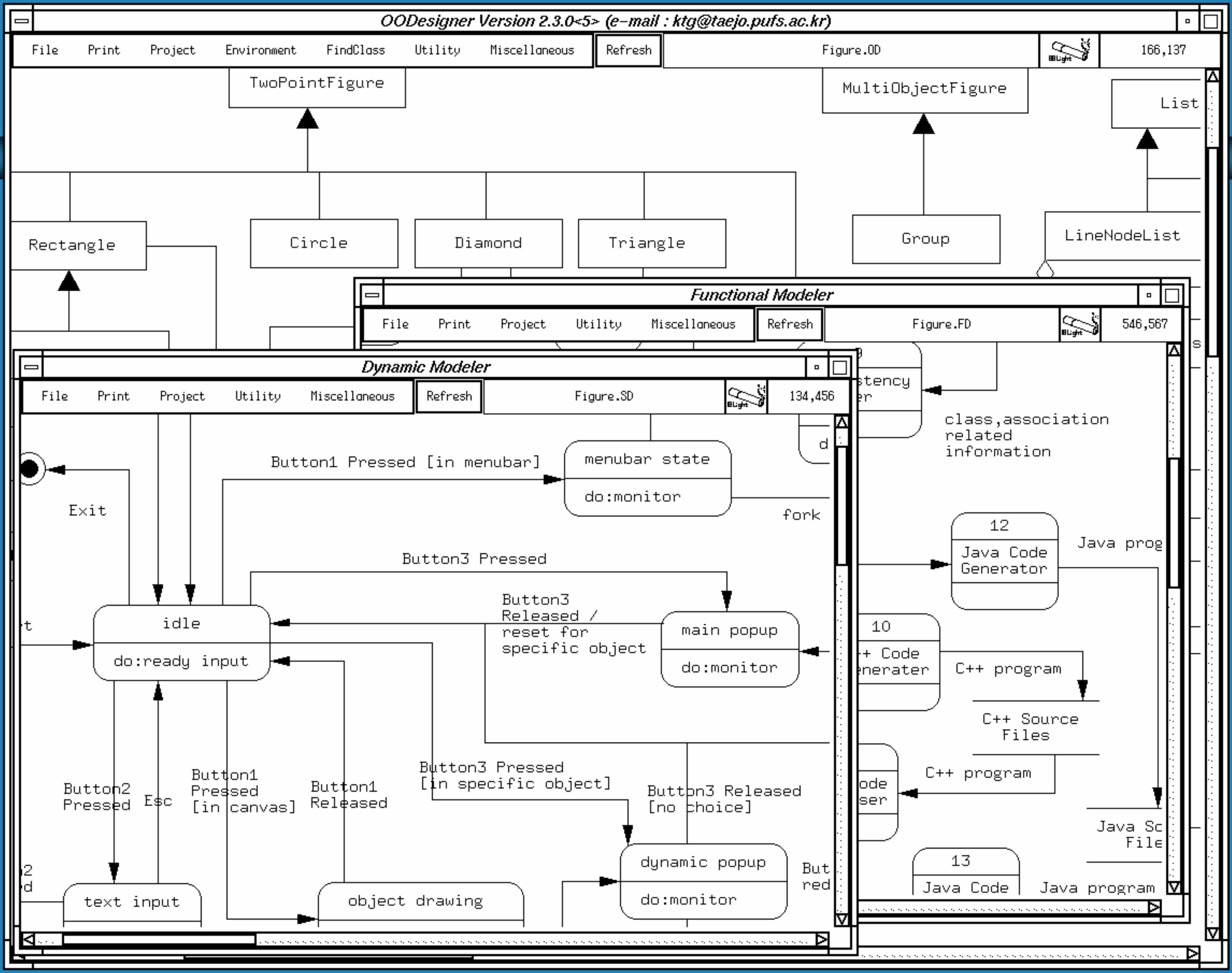
  OS-4.1.x, X11-R5, Motif 1.2, C++ 2.0

  ftp://203.230.73.24/pub/ood or ASSET
- Java version: under development

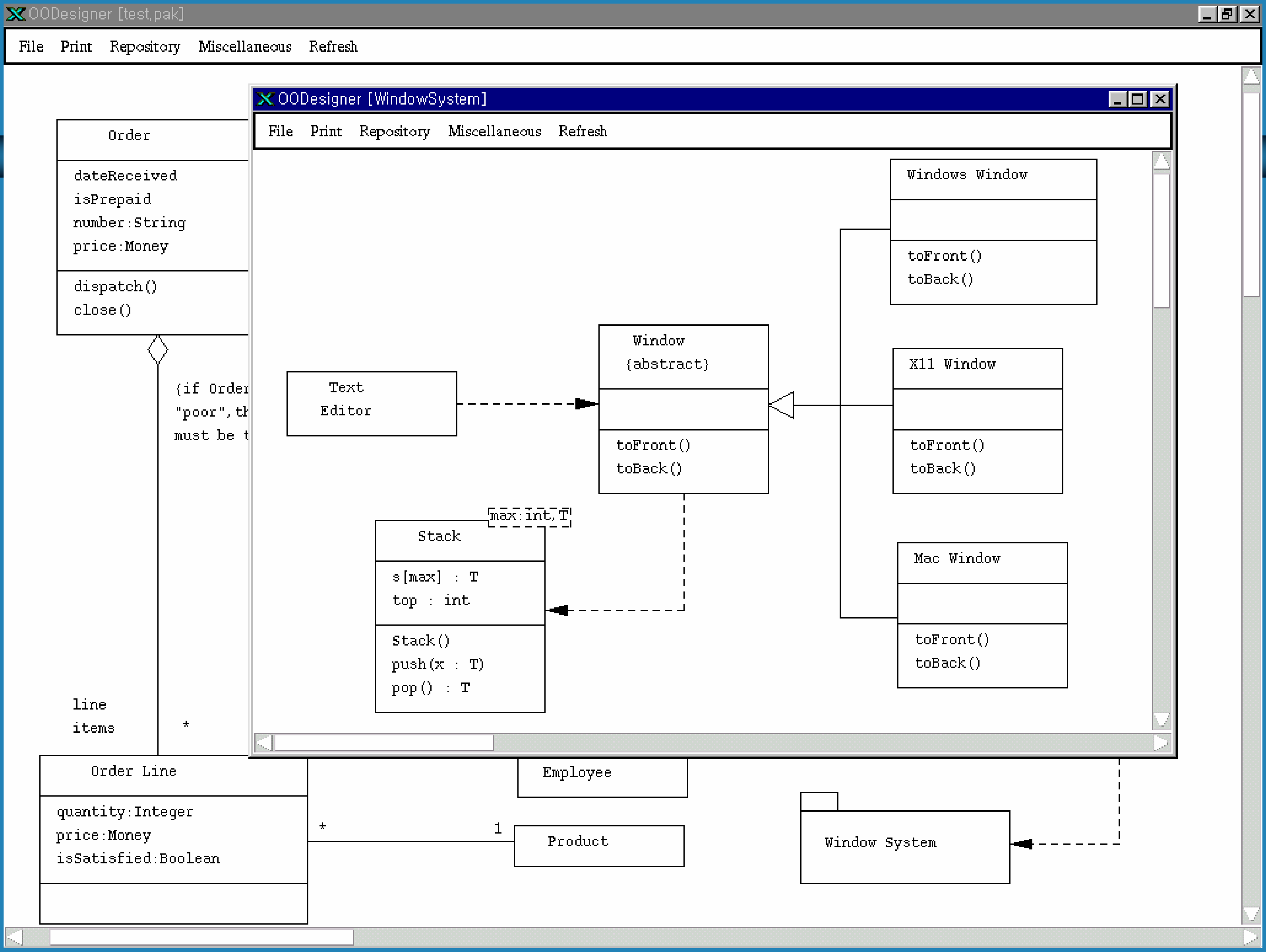
  Java application, JDK 1.1.4
- PC version: under development

  Window95, Visual C++ 5.0

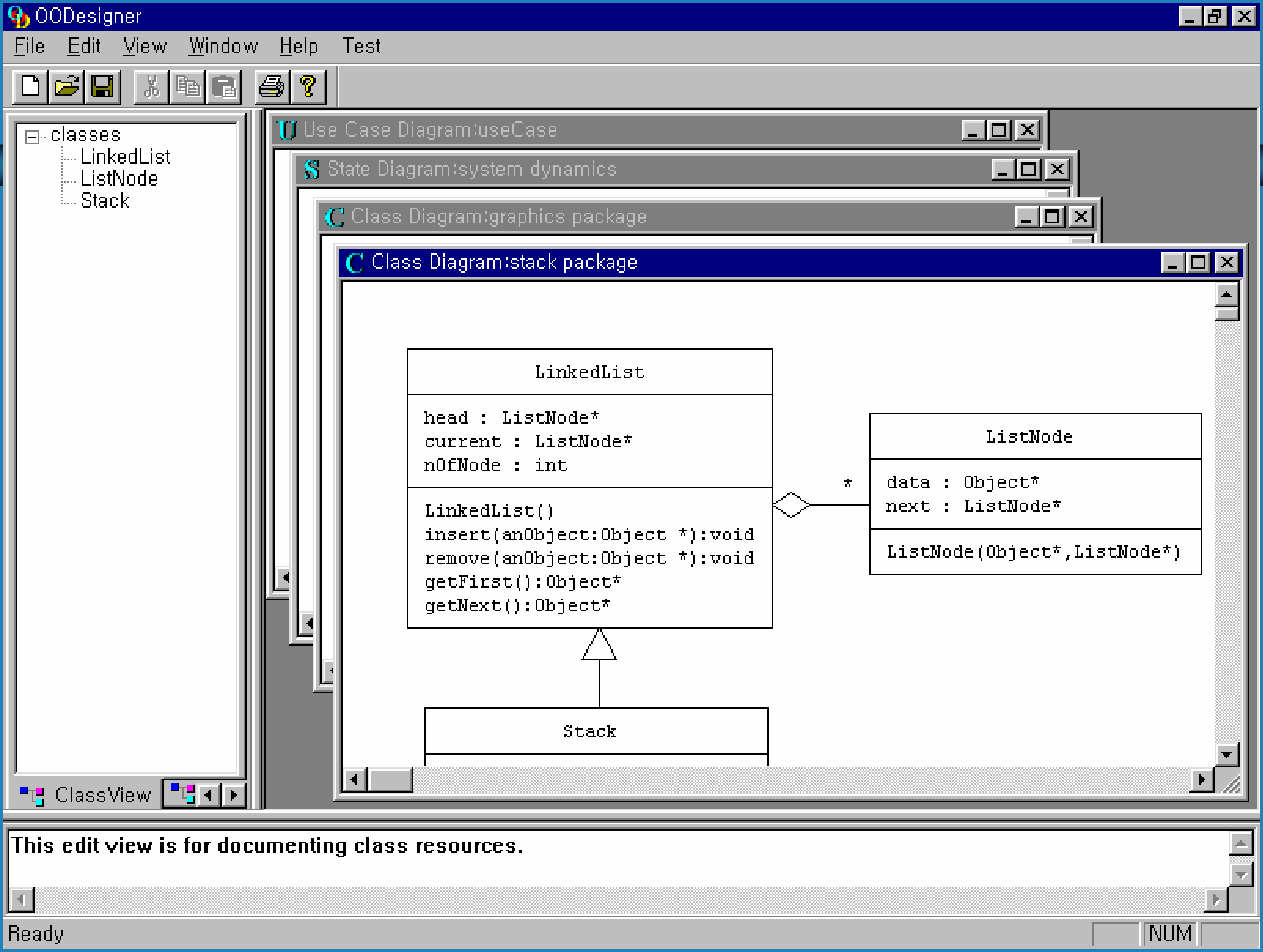
• Unix Version



• Java Version



• PC Version



This edit view is for documenting class resources.

4. Conclusion

- We presented:
  - Restructuring OODesigner

    Problems, Goals, Process, Benefits
  - Metrics Comparisons
  - Current State of OODesigner
- Further Research:
  - Full implementation OO development environment for UML