Multi-level Automated Refactoring Using Design Exploration

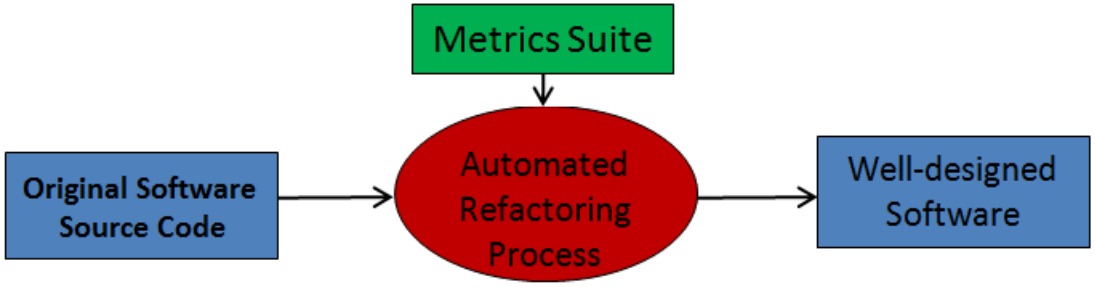
Iman Hemati Moghadam
Supervisor: Mel Ó Cinnéide

Agenda

- Introduction
- Motivations for a Multi-level Refactoring Approach
- Proposed Approach in Detail
- Research Questions
- Key Results to Date
- Conclusion

Introduction

- Primary Goal
  - Improve software quality as expressed by a metrics suite using automated refactoring
- Approach
  - Consider OO design improvement as a combinatorial optimization problem
  - Use search-based refactoring to direct the refactoring process
- Code-Imp
  - *Combinatorial Optimisation for Design Improvement*
  - Refactors automatically Java source code based on a metrics suite



Key results in previous work: Is automated improvement possible?

- Can search-based refactoring improve program design?
  - QMOOD metrics suite was used as fitness function
    - Understandability
    - Flexibility
    - Reusability

- **Results:**
  - Significant improvement in **Understandability**
  - Minimal improvement in **Flexibility**
  - **Reusability** function proved unsuitable
    - Resulted a large number of featureless classes

Key results in previous work : Which search technique works best?

- Which search technique is more effective?
  - Multiple-ascent hill-climbing
  - Genetic algorithm
  - Simulated annealing

- **Results:**
  - **Multiple-ascent hill-climbing** performed the best overall
    - Performed **surface**, rather than **radical**, design exploration
  - Single best design improvement was achieved using **simulated annealing**
    - Illustrates that more **radical** design improvement is probably possible

Motivations

Motivation for Multi-level Refactoring

- Drawbacks of Source-to-Source transformation
  - All low-level code details must be handled
  - Refactoring steps too small to effect major design change
- Both happen because of detailed information at source-code level
- Design-level refactoring can be used for a deeper and faster design exploration
  - However, code refactoring needs to be applied **manually**
  - Cannot be used in an Agile context

Motivation 1:

Perfoming a **radical** and **faster** design exploration rather than minor improvements

Motivation for Multi-level Refactoring

- Using a fully automated refactoring tool
  - It is possible to apply many refactorings without programmer intervention
  - Programmer only sees the end result of the refactoring process

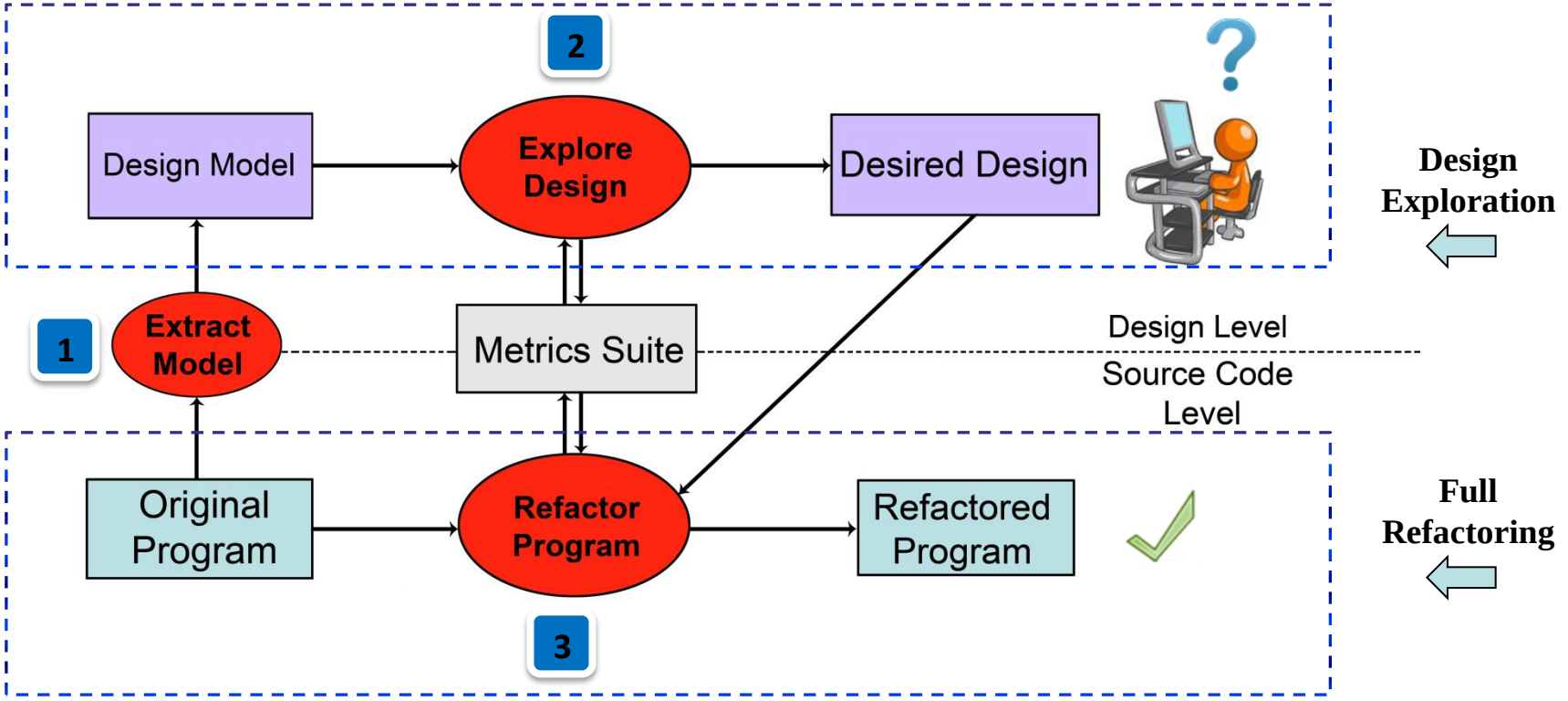
- Difficulties in preserving programmer understandability

Motivation 2:

Improve **User Understandability** of the refactored program

Proposed Approach

Multi-level Automated Refactoring



Research Questions

How to guide the search process efficiently ?

- The code refactoring process is informed by
  - The metrics suite
  - The user-selected design
  - The series of transformations that led to this program design
- How this combination is best used to guide the search is an open question
- What is the best trade-off when the series of transformations applied to the design
  - **Degrade** the quality of the source code?
  - **Cannot be mapped** easily to the existing source code?
  - **Cannot be executed** because of failing pre-conditions?

How to select suitable metrics?

- How metrics including in metrics suite should be selected?
  - **Off-the-shelf** metrics suite (e.g. QMOOD, MOOD, etc.)
  - **Tailored** one
- A recent experiment we performed to explore the metrics relationship
  - Five cohesion metrics were compared
    - LSCC, TCC, SCOM, CC, and LCOM5
  - Some of the metrics were strongly in conflict with each other
    - E.g. **LSCC** and **TCC**
- Is the same metrics suite appropriate for both levels?

A Preliminary Experiment

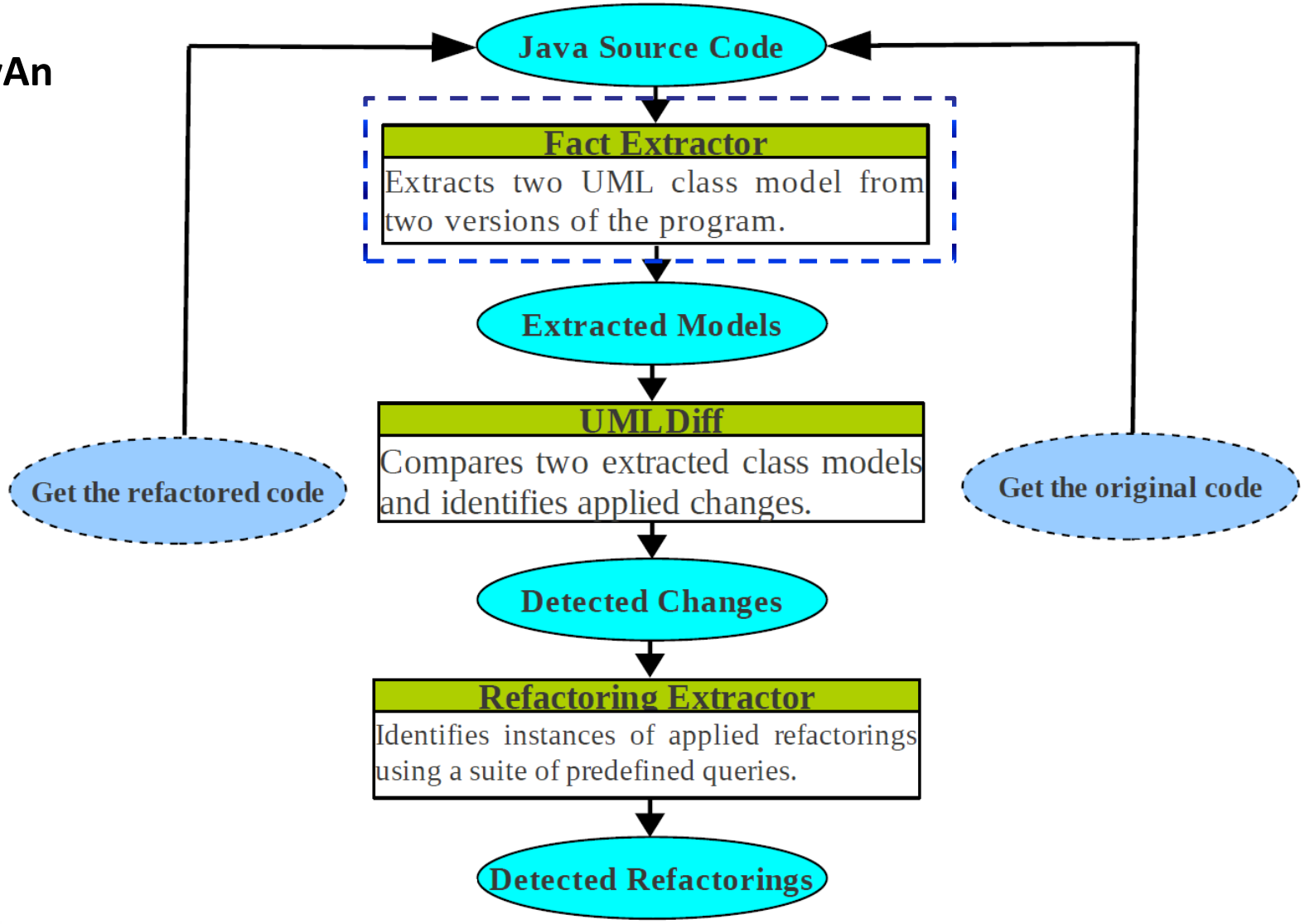
A preliminary experiment

- How precisely can a source program be refactored based on a desired design?

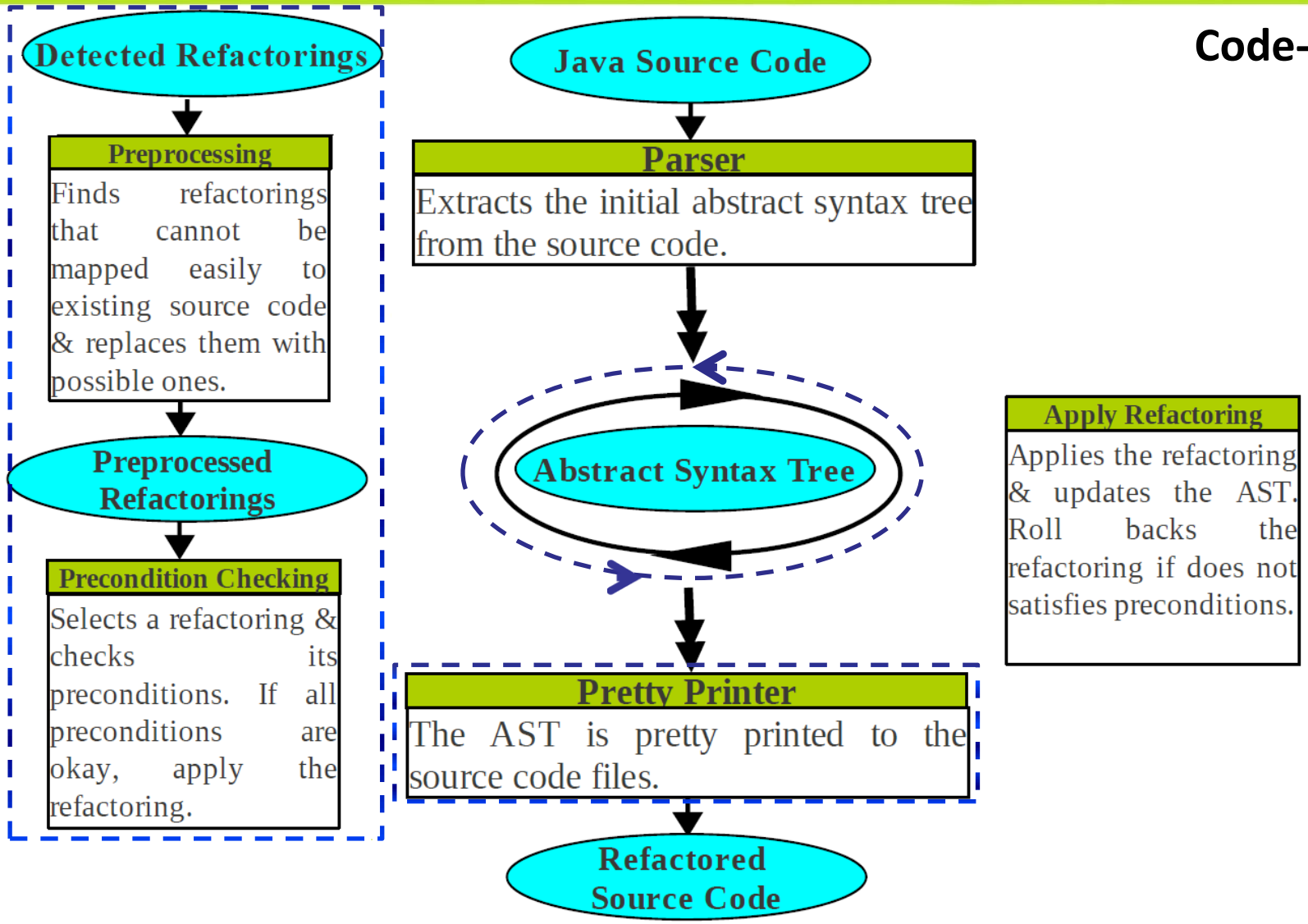
- Implemented Algorithm
  - A well-designed software system (JHotDraw) is randomly refactored
  - Two designs are compared using a design differencing tool (UMLDiff)
    - Applied changes are categorized as refactoring instances
  - Rebuild the original system based on the recovered refactorings
    - The initial program design is the desired design

Categorize detected changes as refactoring instances

JDEvAn



Rebuild the original system based on the recovered refactorings



Code-Imp

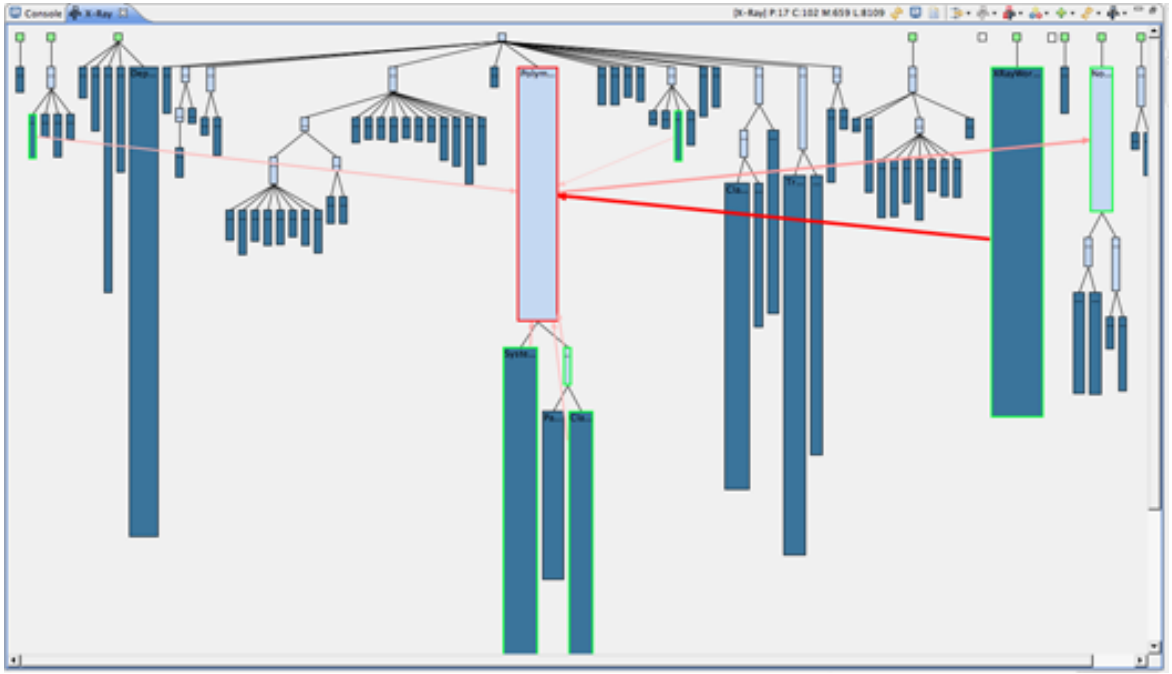
Conclusion

- The presentation covered:
  - The problem with current code improvement tools
    - Perform surface, rather than radical, design exploration
    - Difficulties in preserving user understandability
  - Multi-level Automated Refactoring using Design Exploration
    - Expand refactoring approach to perform radical, instead of surface, design exploration
    - Address the problem of comprehension of the refactored program
  - Progress to date
    - An automated search-based refactoring tool called Code-Imp
    - A preliminary experiment that shows the capability and the potential of the approach

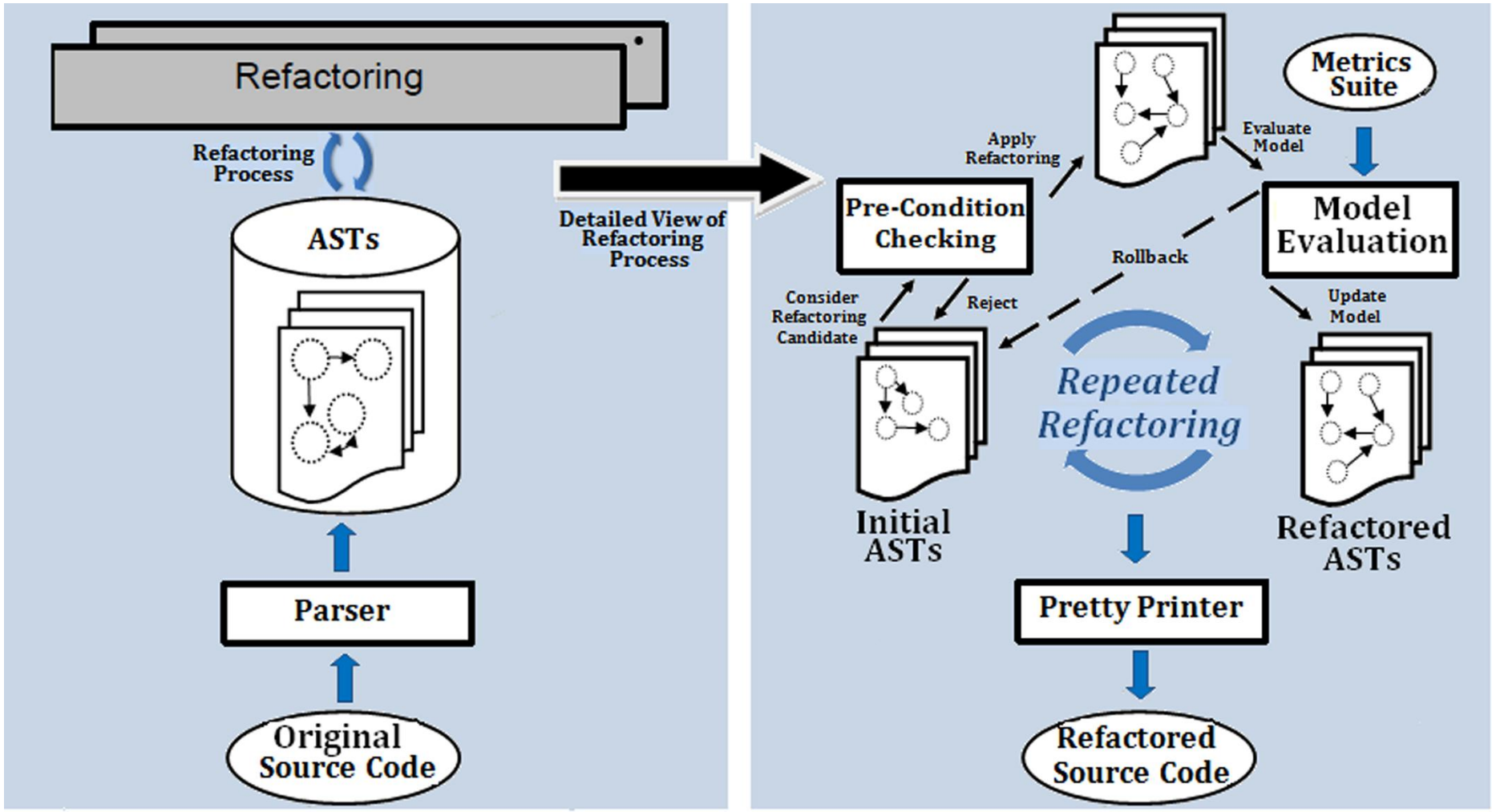
Thank You!

How to explain the new design?

- Reduce complexity using software visualization
  - Highlight design key points
    - Structure
    - History



Code-Imp Architecture



Design-level Refactorings

- **Implemented Refactorings:**
  - Method-level Refactorings
    - **Push Down Method**
    - **Pull Up Method**
    - **Decrease/Increase Method Visibility**
  - Field-level Refactorings
    - **Push Down Field**
    - **Pull Up Field**
    - **Decrease/Increase Field Visibility**
  - Class-level Refactorings
    - **Extract Hierarchy**
    - **Collapse Hierarchy**
    - **Make Superclass Abstract**
    - **Make Superclass Concrete**
    - **Replace Delegation with Inheritance**
    - **Replace Inheritance with Delegation**

Implemented Metrics

No.	Cohesion Metrics	acronym
1	Low-level Similarity Class Cohesion	LSCC
2	Similarity Class Cohesion	SCC
3	Normalized Hamming Distance	NHD
4	Tight Class Cohesion	TCC
5	Loose Class Cohesion	LCC
6	Lack of Cohesion in Methods1	LCOM1
7	Lack of Cohesion in Methods2	LCOM2
8, 9	Lack of Cohesion in Methods3 & 4	LCOM3 & 4
10	Lack of Cohesion in Methods5	LCOM5
11	Information-Flow-Based Cohesion	ICH
12	Cohesion Among Method of Class	CAMC
13	Class Cohesion	CC
14	Sensitive Class Cohesion Metrics	SCOM
	Coupling Metrics	
15	Response for Class	RFC
16	Direct Class Coupling	DCC
17	Data Abstraction Coupling	DAC
18	Coupling Factor	COF
19	Coupling Between Objects	CBO
20	Instability	Instability
21	Information-Flow-Based Coupling	ICP
22	Non-Inheritance ICP	NICP
23	Inheritance ICP	IICP
24	Message Passing Coupling	MPC
	Other Metrics	
25	Class Interface Size	CIS
26	Number of Methods	NOM
27	Data Access Metric	DAM
28	Design Size in Classes	DSC
29	Number of Polymorphic Methods	NOP
30	Average Number of Ancestors	ANA
31	Number Of Hierarchies	NOH