Refactoring Practice:
How it is and How it Should be Supported

An Eclipse Case Study

Zhenchang Xing and Eleni Stroulia

Presented by: Sultan Almaghthawi

Outline

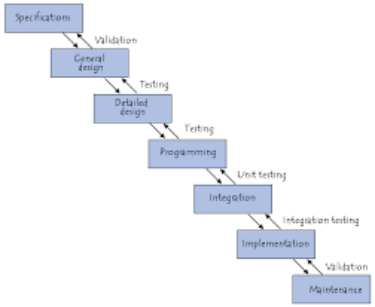
- Main Idea
- Related Works/Literature Alignment
- Overview of the Case Study
- Study Conducted
- Assessment
- Discussion

Literature Alignment

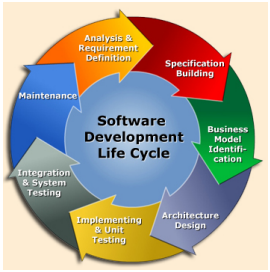
- Written in 2006 (before LSDiff)
- Emphasize:
  - Different types of refactorings
  - How refactoring should be done
  - The lack of support to high-level refactoring in modern tools
- Inspired other researchers to investigate high-level refactoring.

Introduction

- Most object-oriented software systems are developed using an evolutionary process model

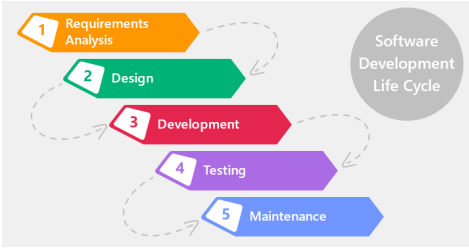


- **Change** is an integral part of the evolutionary development lifecycle.

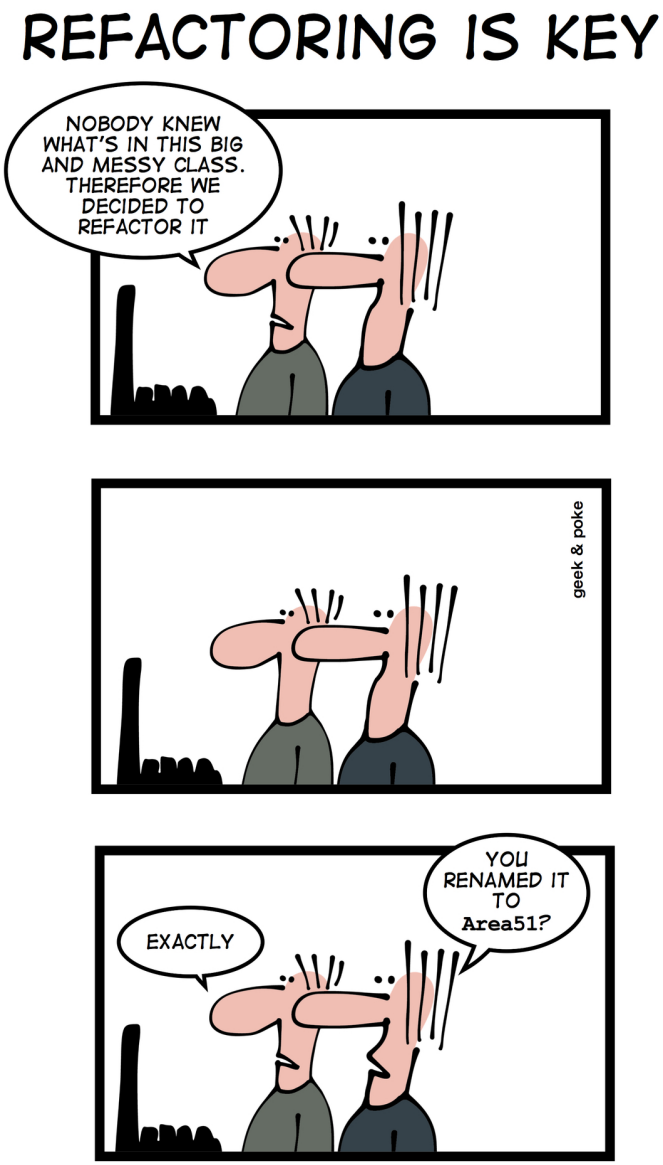


- An Important type of OO Software changes is called

Refactoring

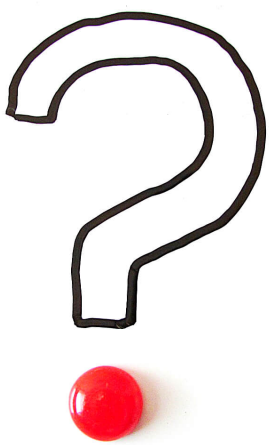


Introduction



Introduction

What is Refactoring



Refactoring

DefinitionOfRefactoring

Martin Fowler
1 September 2004

In my refactoring book, I gave a couple of definitions of refactoring.

Refactoring (noun): a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.

Refactoring (verb): to restructure software by applying a series of refactorings without changing its observable behavior.

Source: http://martinfowler.com/bliki/DefinitionOfRefactoring.html

***Refactoring: Improving the Design of Existing Code*, 1999**

Refactoring

- The goal of refactoring is to improve the quality of the software system, such as its :
  - Understandability
  - Extensibility
  - Maintainability .. Etc.
- Without changing its overall functionality and behavior.

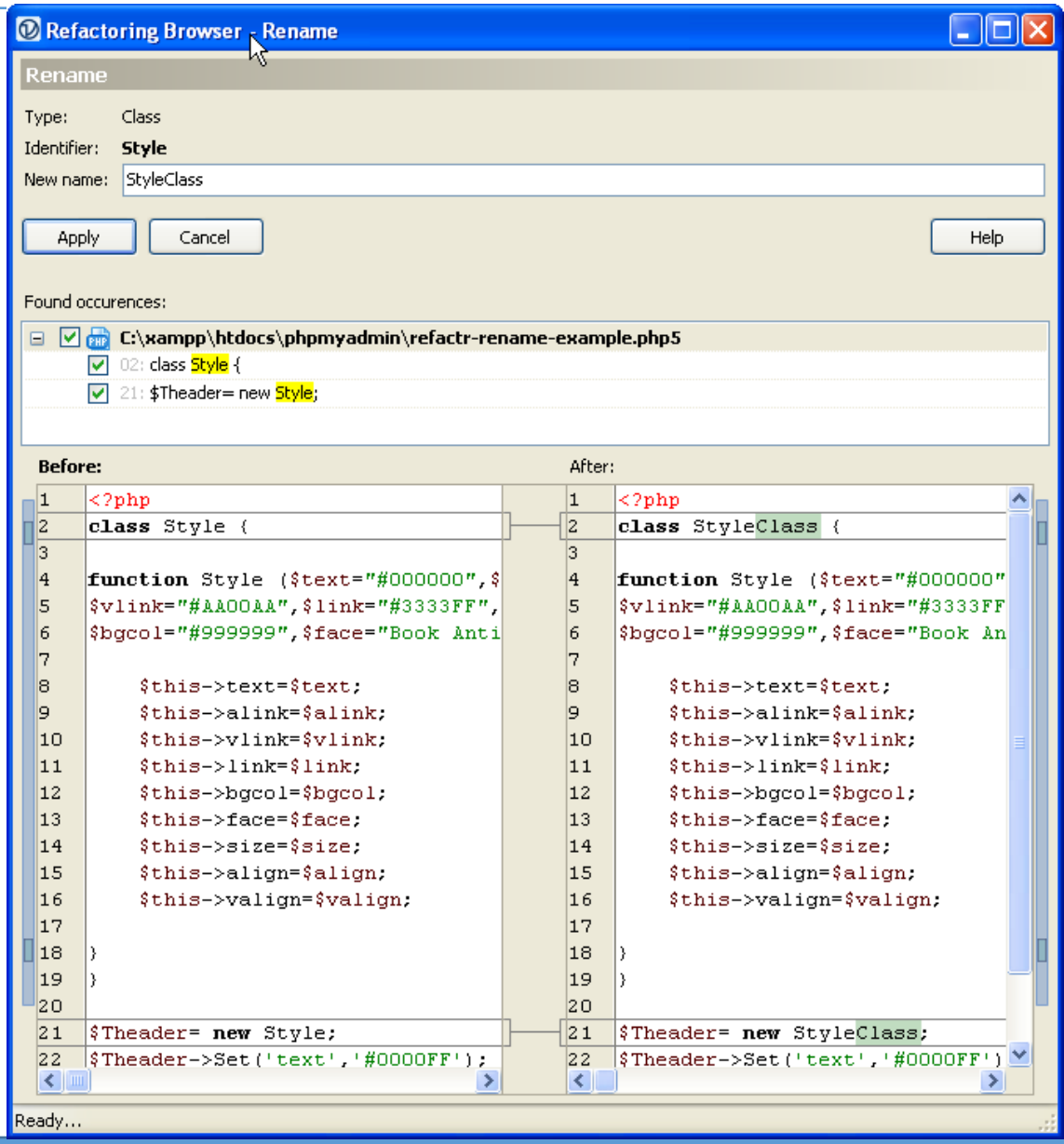
Refactoring

- Refactoring is very beneficial: IDE's Integration
  - Eclipse and Refactoring Browser :
    - "rename Field"
    - "rename Class"
    - "Move Method"
  - It has become less Expensive and more achievable.

Refactoring

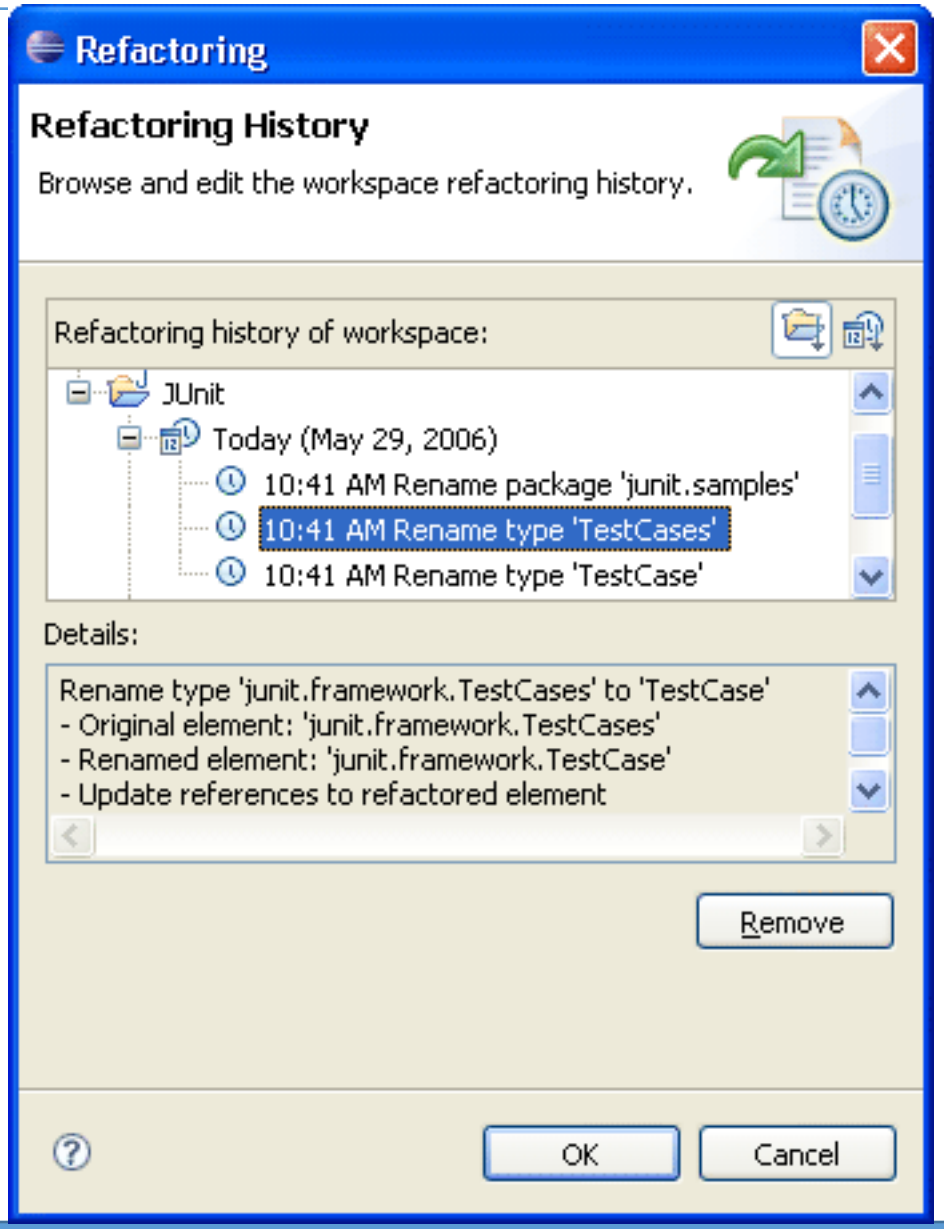
- Refactoring Browser
  renaming a class

Style -> StyleClass



Refactoring

- Eclipse Refactoring
  Ref History Viewer

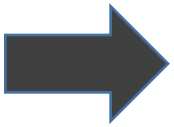


Refactoring

- **Refactoring Activities become challenging:**
  - Reuse-based development

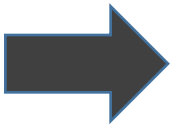
Developer should not remove or change existing parts of the API, or else they may cause the client applications to fail (which is dependent on that code).

- The more widely adopted the underlying component framework is



The higher the cost of breaking client code becomes

- These potential challenges rises because



There is a need for understanding how this activity is actually practiced

Refactoring – Research Question

Unanswered Questions

What proportion of the structural changes in the evolution of a system are the results of refactoring?

What are the typical refactorings applied in practice?

What aspects of a system's structural evolution can be automatically gathered?

Refactoring – Research Question

Unanswered Questions

Which of these types are "safe" to client applications that reuse the refactored system?

What type of support should modern IDEs provide and how might this support be implemented?

Goals

- Conduct a study on the structural evolution of Eclipse
  - Using JDEvAn
  - Implements a design-level structural differencing algorithm
    (which is UMLDiff)

- Understand the design requirements for refactoring-based development environment
  - from the perspectives of both the component developers and component users.

JDEvAn - "Java Design Evolution and Analysis"

- Analyzes the evolving structure of design-level software artifacts.
- Generates a Report of interesting trends and events during the evolution.
- JDEvAn focuses on the logical view of the OO system
  - Classes, Association and the services related they deliver.
- **INPUT**: source code from versioning system
- JDEvAn's UMLDiff is a domain-specific structural differencing algorithm

JDEvAn

API Proposals

[removed][com.jrefinery.chart.data.PlotFit.getFit()]

Message	Status	Support	#Heuristics
com.jrefinery.data.MovingAverage.createMovingAverage(XYDataset,String,long,long)	**added**	**0.5**	**1**
The problem entity and the proposed entity declare the same or compatible data type			
com.jrefinery.data.TimeSeriesCollection.TimeSeriesCollection()	**matched**	**0.6666667**	**1**
com.jrefinery.chart.demo.DemoDatasetFactory.createHighLowDataset()	**renamed**	**0.5**	**1**
com.jrefinery.data.JDBCXYDataset.JDBCXYDataset(Connection,String)	**renamed**	**0.16666667**	**1**
com.jrefinery.data.DatasetUtilities.sampleFunction2D(Function2D,double,double,int,String)	**matched**	**0.16666667**	**1**
The problem entity and the proposed entity declare the same or compatible data type			

Usage Examples

Usage examples for [added][com.jrefinery.data.MovingAverage.createMovingAverage(XYDataset,String,long,long)]

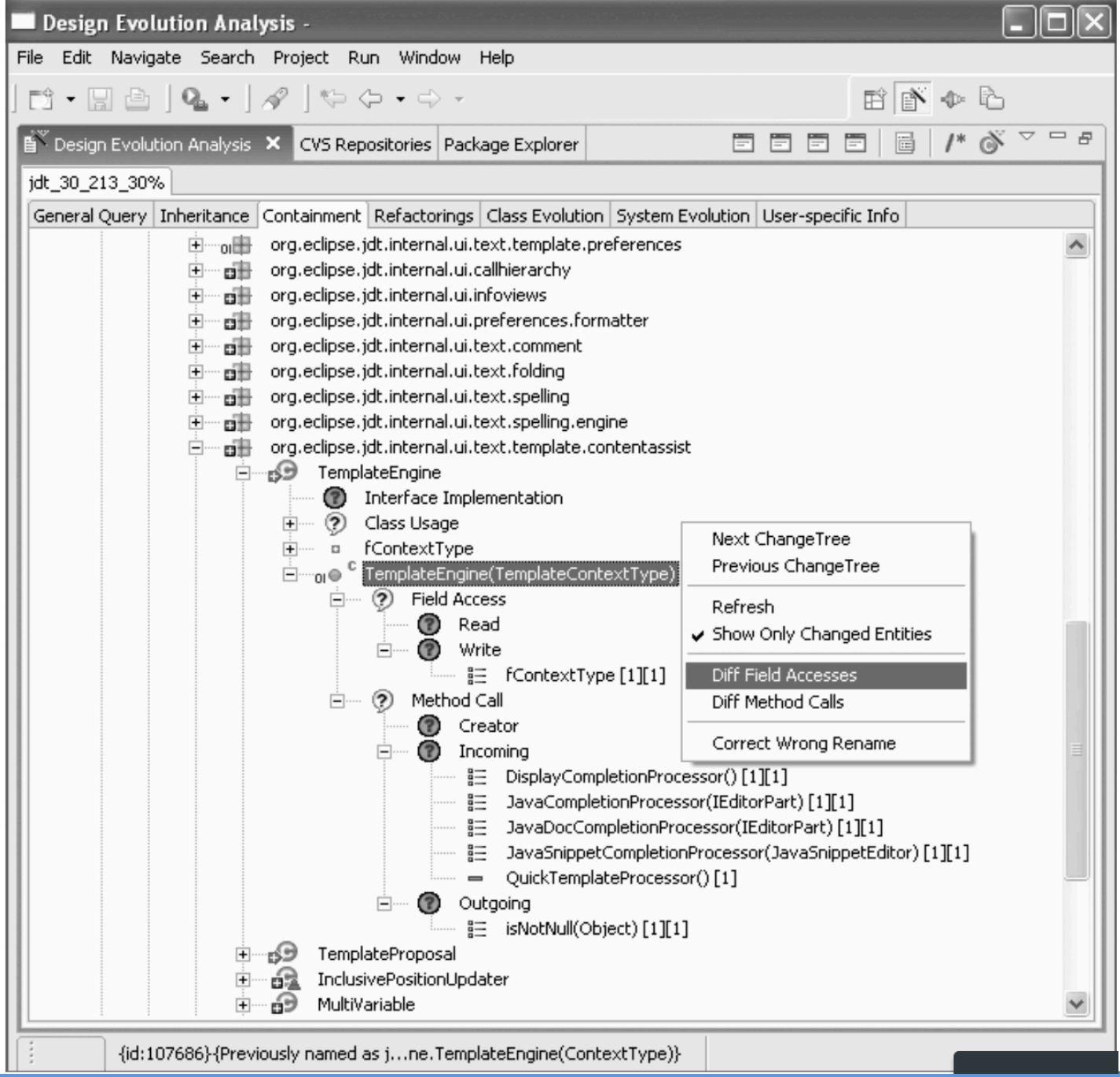
Message	Support	#Heuristics	Status
com.jrefinery.chart.demo.servlet.BaseImageServlet.createXYChart(int,String)	**2.0**	**1**	**matched**
For added parameter 'source' of the requested operation			
Invoke constructor of current type 'com.jrefinery.data.XYDataset' or its subtype			
com.jrefinery.data.JDBCXYDataset.JDBCXYDataset(Connection,String)			
Call method of current type 'com.jrefinery.data.XYDataset' or its subtype			
com.jrefinery.data.DatasetUtilities.sampleFunction2D(Function2D,double,double,int,Stri			
com.jrefinery.chart.demo.JFreeChartDemoBase.createCombinedAndOverlaidChart1()	**2.0**	**1**	**matched**
For added parameter 'source' of the requested operation			
Invoke constructor of current type 'com.jrefinery.data.XYDataset' or its subtype			
com.jrefinery.data.TimeSeriesCollection.TimeSeriesCollection()			
Call method of current type 'com.jrefinery.data.XYDataset' or its subtype			
com.jrefinery.chart.demo.DemoDatasetFactory.createHighLowDataset() : com.jrefinery.(			
com.jrefinery.chart.demo.JFreeChartDemoBase.createOverlaidChart()	**1.0**	**1**	**matched**

Figure 2. API Proposals and Usage Examples view

UMLDiff

- UMLDiff reports the structural changes between the two software versions in terms of:
  A. additions, removals, moves, renamings of packages, classes, interfaces, fields and methods,
  B. changes to their attributes, such data type, visibility, and modifiers
  C. changes of the dependencies among these entities
- It provides a detailed picture of the system changes
- This allows us to analyze and classify the structural changes that are **refactorings.**

UMLDiff and JDEvAn



Related Work

- **Opdyke's Ph.D. thesis**
  - catalogs a number of refactorings.
  - lists a set of invariants that must conform in order to behavior-preserving
- **The books of Fowler and Kerievsky**
  - Providing a good overview and definitions of refactoring
  - Good practice to accomplish design changes
- **Demeyer et al.**
  - Define four heuristics to identify refactorings.
- **Godfrey and Zou** use origin analysis to detect the merging and splitting of source-code entities.

Related Work

"Refactoring the reused components is often limited by the fear of breaking client code."

Automated refactoring support tool was needed.

- **Opdyke** developed first migrating tool in Refactoring Browser
- **Balaban et al** developed a tool that allow for mapping between old classes and their replacement. (Java Vector ➔ Java ArrayList)
- **CatchUp** is another attempt to relieve this burden by recording refactorings within IDEs such as Eclipse.
- **Dig and Johnson** conducted an empirical study on API migration
  - Similar Result, However, Changes were reported based documentation
  - Compared to this paper, This paper is using UMLDiff which is more detailed

Related Work

"Refactoring the reused components is often limited by the fear of breaking client code."

Automated refactoring support tool was needed.

- **Opdyke** developed first migrating tool in Refactoring Browser
- **Balaban et al** developed a tool that allow for mapping between old classes and their replacement. (Java Vector ➔ Java ArrayList)
- **CatchUp** is another attempt to relieve this burden by recording refactorings within IDEs such as Eclipse.
- **Dig and Johnson** conducted an empirical study on API migration
  - Similar Result, However, Changes were reported based documentation
  - Compared to this paper, This paper is using UMLDiff which is more detailed

Case Study – Eclipse Structural Changes

- Project: Eclipse (9 releases at the time)
- First version 2.0, released on June 27 2002
- Latest (at the time) 3.1 June 27 2005
- Comparison included 3 pairs:
- 2.0 and 2.1, 2.1.3 / 3.0, and 3.0.2 and 3.1
- Why Eclipse ?
  - The system has been undergoing substantial evolution
  - It is well documented
  - Widely used, potential impact of refactoring of reusable frameworks to their applications.
- Focus was on JDT subproject which has 40% of classes and interfaces.

Case Study – Data

Data extracted using JDEvaAn

Table 1. The number of program entities

	2.0	2.1	2.1.3	3.0	3.0.2	3.1	Total
Package	138	144	144	177	177	188	968
Class	3546	4326	4332	5610	5612	6466	29892
Interface	692	768	769	935	935	1024	5123
Array Type	562	294	296	383	383	439	2357
Field	11440	14213	14245	18812	18862	29029	106601
Method	27623	33829	33878	42923	42927	49187	230367
Constructor	3929	4737	4751	6025	6027	6943	32412
Total	47930	58311	58415	74865	74923	93276	**407720**

Number of entities is increasing

Case Study – Data

Data extracted using JDEvaAn

Table 2. The number of entity relations

	2.0	2.1	2.1.3	3.0	3.0.2	3.1	Total
Contains	53623	65925	66034	84963	85076	105162	460783
Extends	3253	4003	4009	5134	5135	5921	27455
Implements	1449	1790	1792	2298	2300	2596	12225
Reads	44583	54842	54954	73754	73827	98120	400080
Writes	17781	21597	21638	27755	27815	32648	149234
Calls	90924	117813	117815	151629	151858	179775	809814
Class usage	31658	39284	39362	51700	51830	61594	275428
Class instantiation	9915	12273	12315	16025	16123	19037	85688
Total	253186	317527	317919	413258	413964	504853	**2220707**

Number of relations isincreasing as Eclipse evolves

The project is substantially evolving.

Case Study – Data

Data extracted using JDEvaAn

Table 3. Eclipse's changes

Type of change	2.1 – 2.0	3.0 – 2.1.3	3.1-3.0.2	Total
Entity renaming	809	2285	1488	4582
Entity move[#3]	387	1244	684	2315
Visibility change	435	857	550	1842
Data (return) type change	245	718	561	1524
Non-access modifier change	167	484	425	1076
Interface implementation change	190	391	274	855
Class inheritance change	33	109	162	304
Entity addition	7127	14095	17343	38565
Entity removal	1298	4157	2455	7910
Total	10691	24340	23942	**58973**

Case Study – Empirical Assessment of Eclipse

- **Elementary structural changes: Examined 7**

1. First started with “renaming”
2. then proceed to examine increasingly “suspect” modifications

   such as moves, modifier and visibility changes, data-type changes, inheritance-relation changes, and entity additions and removals.

- **Entity Renaming,** motivation behind it: 4891 renamings
- Conformance to a consistent naming scheme , isOnBuildPath()
- Reflecting the semantics of a particular change
- Concept merging or splitting RootCache() + pkgCache() ➔ ProjectCache()
- backward compatibility , oldParser()

Case Study – Empirical Assessment of Eclipse

- **Entity moves:** 2315 move instances
- Motivation :

1. Reorganizing
2. Moving responsibilities to eliminate Law-of-Demeter violations;
   "only talk to your friends" a method should only call method within the class (low-Coupling)
3. Maintaining backward compatibility
4. Deprecation plus delegation

Case Study – Empirical Assessment of Eclipse

- **Modifier changes:** 1088 modifier changes in 1064 entity
- **Visibility changes** 1842 program entities, either to restrictive or less.
- **Data-type changes** found 1524 data-type changes
- **Generalization and abstraction changes** 304 changes in total
- **Program-entity additions and removals**

Case Study – Empirical Assessment of Eclipse

- **Bigger Refactoring !!**
- Small series of refactoring leads to a big change in the system
- Such as introducing a design pattern.
- Other type of refactoring :
  - Containment-hierarchy refactoring
  - Inheritance-hierarchy refactoring
  - Class-relationship refactoring
  - Internal class refactoring

- **Structural Changes sequence**
- It seems that entity that undergo a refactoring process is most likely to be modified later.
- 27% (2104/7851) of entities have undergone a series of changes

Case Study – Analysis - Findings

- Refactoring is a advocated as a good and frequent practice
- It is not known how prevalent in practice it is
- UMLdiff reported 58973 changes in total.
- Most of the implementation changes occurred in the major release 2.1, 3.0, and 3.1
- 16% of all the changes can be called “standard” refactoring
- ➔ Idicator that a lot of effort is spent on restructuring the system

Therefore, providing (semi-)automatic refactoring support for developer is very beneficial.

Case Study – Analysis - Findings

- Support is still missing for higher-level refactoring

- Most IDE such as Eclipse supports low-level refactoring
  - Renaming , moving instance field, push up a method to its own type but rather should be to any type.
- **An Effective refactoring tool should support the following**

1. information hiding refactoring, such as “hide a group of method in a class”
2. more flexible move of instance field and method
3. a refactoring user interface to collect the information about more complex refactoring task

Case Study – Analysis - Findings

- **Tools should implement refactorings using the command and composite patterns**
- A good possible implementation of refactoring
- ➔ would be to view a structural change as a command object
- For two reason:
  1. simple refactoring commands could be composed into larger ones
  2. and they could also be done, undone and replayed

Example: a method was moved and removed one of its parameter

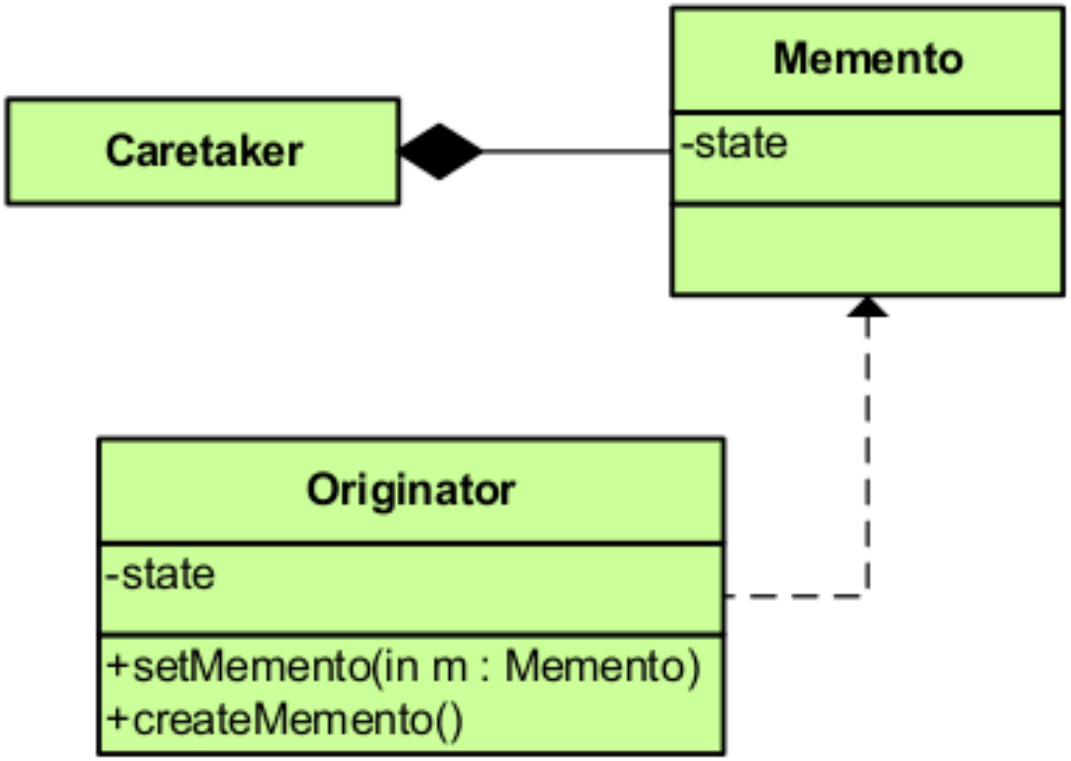
A memento object may be used to record which parameter is removed

Case Study – Analysis - Findings

Memento

Type: Behavioral

What it is:
Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.



Conclusion

- Refactoring is an activity crucial for evolutionary-development processes
- With the right refactoring the design will become:
  - more cohesive,
  - less coupled
  - and maintainable
- Eclipse is not typical but has interesting evolution history to study
- Although Refactoring is behavior-preserving, They might still affect the client behavior.
- Current tools are limited in terms of Compositional requirement and refactoring migration

Discussion