Discovering and Representing Systematic Code Changes

Miryung Kim and David Notkin, ICSE 2009

Presented by: Sultan Almaghthawi

Outline

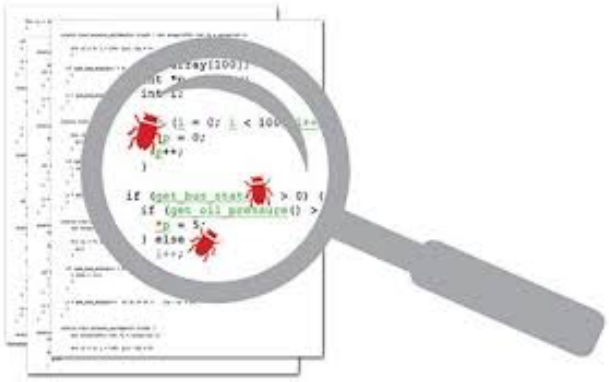
- Main Idea/Problem
- Related Works
- Approach/Algorithm
- Study Conducted
- Assessment
- Discussion

Main Idea

- Developer often want to inspect the code
- Certain tools are built for this purpose
  - Code change differences tool is one

Problem

- Why code changes differencing tools ???
- Inspecting code changes
  - ▸ Behavior changes after modifications
  - ▸ Code reviewing

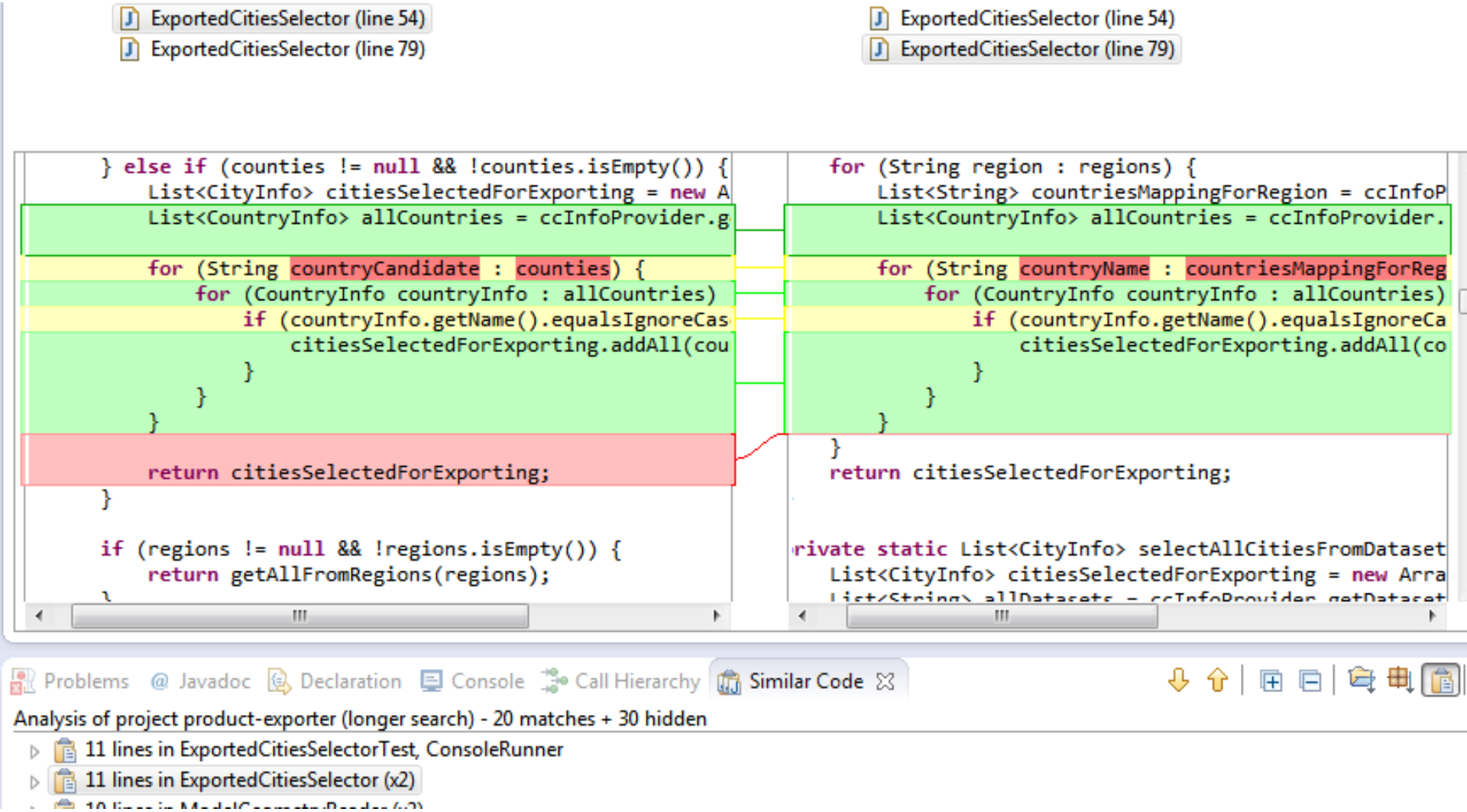


- Program differencing tool limitations:
  - Detecting inconsistency
  - The Ability to group related code changes
  - Focusing on lower-level changes but not the systematic ones
  - One may have to look file-by-file
  - Different levels of abstraction
  - Reports individual changes without structure
- ✕ generally do not capture systematic changes

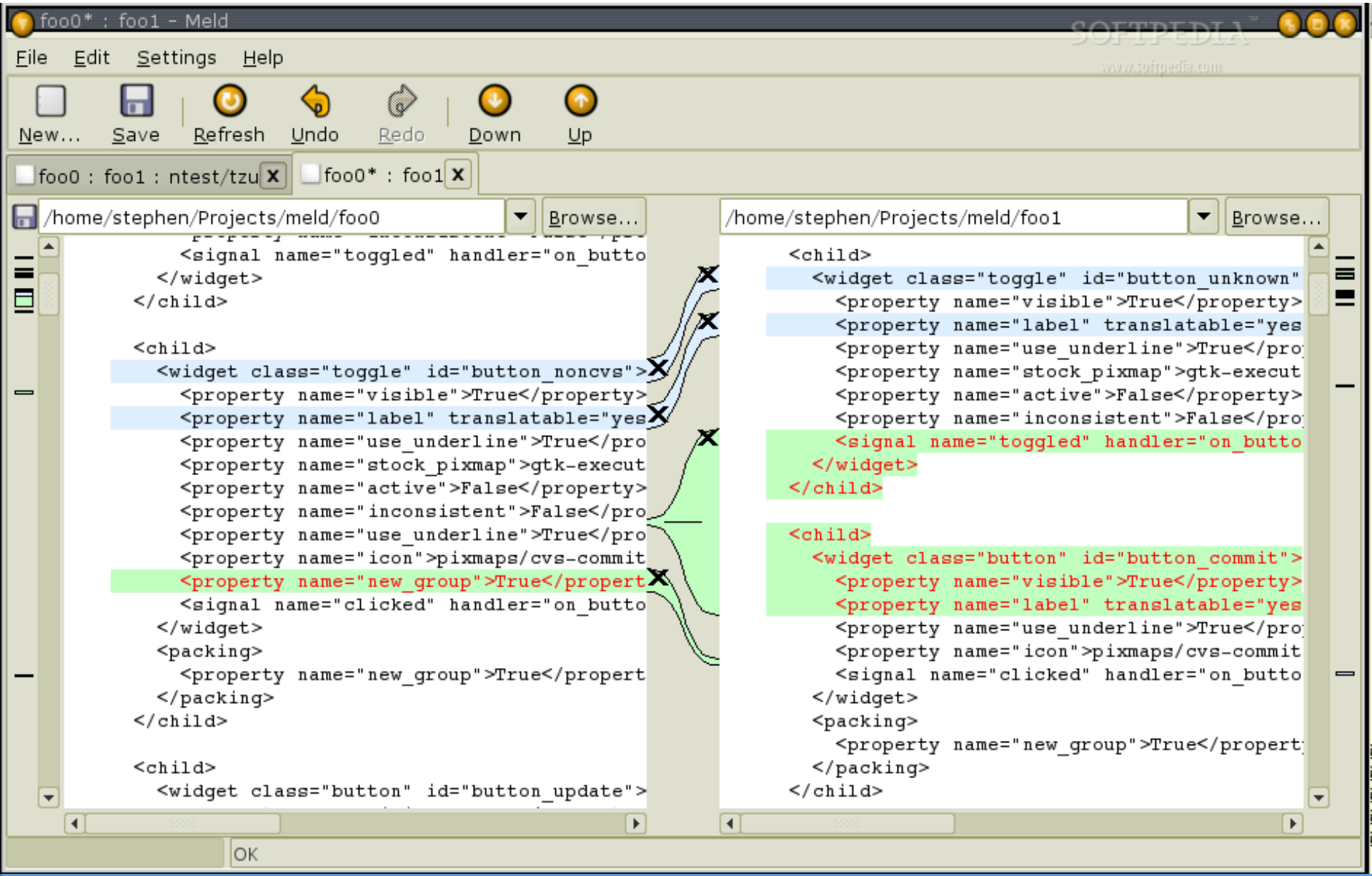
Problem

- UMLdiff and EclipseDiff
  - organize differences by logical locations
  - Not orthogonal to a program's containment hierarchy
- Others detect crosscutting concerns but don't find regularities
  - such as adding similar fields in the same class.
- Therefore: Developer is burdened to detect inconsistencies that leads to potential bugs.

EclipseDiff



Diff tool


foo0* : foo1 - Meld
File Edit Settings Help
New... Save Refresh Undo Redo Down Up
foo0 : foo1 : ntest/tzu
foo0* : foo1
/home/stephen/Projects/meld/foo0
Browse...
/home/stephen/Projects/meld/foo1
Browse...
OK

Main idea/Solution

- Developing a tool that will overcome these limitations
- **LSdiff: Logical Structural Diff**
- A tool that infers systematic structural differences as logic rules.
- Focuses on systematic changes
- Detects Exceptions to the logic rules

LSdiff

- Represents a program version as a set of predicates
- Described as : Structural information
  - Also called Fact-Based Representation
- LSdiff abstracts a program as code elements (e.g., methods and fields)
- And the structural dependencies (e.g., field-accesses and overriding)

This abstraction is used to identify systematic changes

1. package (packageFullName)
2. type (typeFullName, typeShortName, packageFullName)
3. method (methodFullName, methodShortName, typeFullName)
4. field (fieldFullName, fieldShortName, typeFullName)
5. return (methodFullName, returnTypeFullName)
6. fieldoftype (fieldFullName, declaredTypeFullName)
7. typeintype (innerTypeFullName, outerTypeFullName)
8. accesses (fieldFullName, accessorMethodFullName)
9. calls (callerMethodFullName, calleeMethodFullName)
10. subtype (superTypeFullName, subTypeFullName)
11. inheritedfield (fieldShortName, superTypeFullName, subTypeFullName)
12. inheritedmethod (methodShortName, superTypeFullName, subTypeFullName)

LSdiff

- LSdiff infers logic rules
- Any differences not represented by the inferred rules are retained as logic facts.

- Example: Cathy is trying to understand Alan's code

Fact 1. AbsRegistry is a new class.
Rule 1. All host fields in NameSvc's subtypes were deleted except LmiRegistry class.
Rule 2. All setHost methods in NameSvc's subtypes were deleted except LmiRegistry class.
Rule 3. All getHost methods in NameSvc's subtypes deleted calls to SQL.exec except LmiRegistry class.

LSdiff

- Alans added AbsRegistry
- By pulling up ‘host’ field and setHost
- Cathy’s can double check why Alan left LimiRegistry unchanged.

Fact 1. AbsRegistry is a new class.
Rule 1. All host fields in NameSvc’s subtypes were deleted except LmiRegistry class.
Rule 2. All setHost methods in NameSvc’s subtypes were deleted except LmiRegistry class.
Rule 3. All getHost methods in NameSvc’s subtypes deleted calls to SQL.exec except LmiRegistry class.

LSdiff

- This work is motivated by a previous work in 2007
  - M. Kim, D. Notkin, and D. Grossman. Automatic inference of structural changes for matching across program versions. In ICSE, pages 333–343, 2007.
- It extends its rule representation and inference algorithm.
- To describe changes within the method body and at the field level as well.
- It relies on observation and studies arises from clone code.

Lsdiff

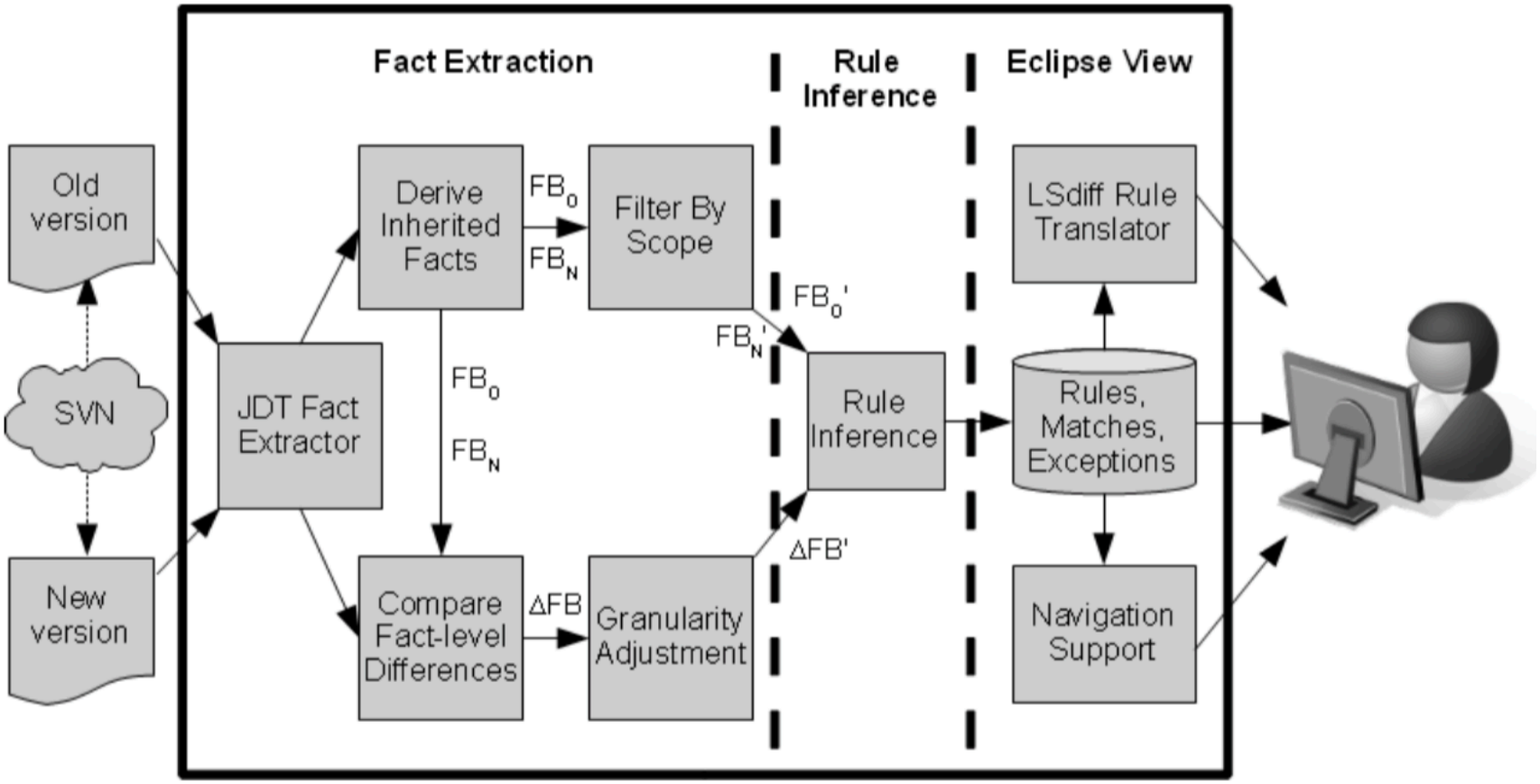
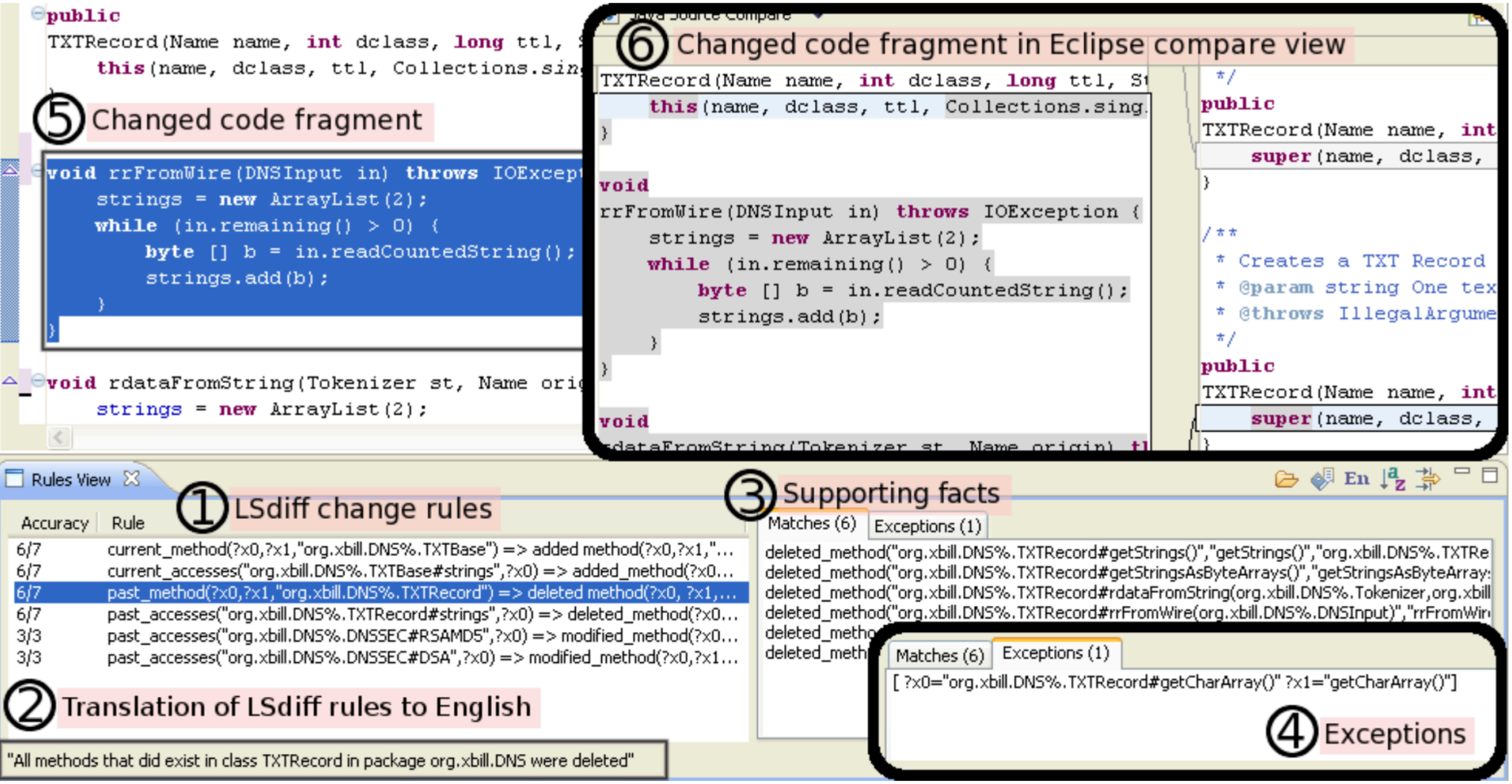


Figure 3: Architecture of LSdiff

LSdiff: A Program Differencing Tool to Identify Systematic Structural Differences ,M. Kim, Alex Loh, 2010

Lsdiff Eclipse View



Related Work

- Diff tools varies in different ways:
  - abstraction level
  - Computing deltas
  - Representation
- Longest Common Sequence (Textual)
- Abstract Syntax Tree (AST) based
- Control Flow Graph
- Program Dependencies Graph
- Rule Based

- LSDiff is different by detecting systematic changes at the abstract level

Related Work

- Some types of **systematic changes** methods
- Refactoring
  - Abstract , systematic, follow similar patterns
- Cross-cutting concerns
  - Abstract, injecting logging at specific areas of code
  - Techniques: clone detection, language processing …etc
  - Ex: Dagenais et al automatically infer structural patterns

Related Work

- How is LSdiff different from:
- refactoring??
  - Both are at the abstract level
  - LSDiff identifies a broader class of systematic changes
  - Behavior changing modifications. …
- Cross-cutting concerns ??
  - LSDiff differs by inferring general kinds of systematic changes
  - detecting anomalies from systematic changes

Related Work

- Logic-based Program Representation
  - Jquery or CodeQuest
  - automatically evaluate logic queries
  - Mens et al.'s , use loging to determine design patterns
  - Eichberg et al. Datalog rules to continuously enforce constraints

- How is LSDiff different??
  - focusing on systematic differences between two versions
  - inferring rules without explicit specification

LSdiff

- Delta Representation

1. package (packageFullName)
2. type (typeFullName, typeShortName, packageFullName)
3. method (methodFullName, methodShortName, typeFullName)
4. field (fieldFullName, fieldShortName, typeFullName)

LSdiff

- Po: Old version
- Pn: new version
- FBn: a fact-based of the new version
- △FB: represents the differences,
  - computed by taking the set difference of FBo and FBn

Table 1. A fact-base representation of two program versions and their differences

P_o (an old version)	P_n (a new version)	FB_n (a fact-base of the new version)	ΔFB

`class BMW implements Car`
`void start (Key c) {`
`...`
	`class BMW implements Car`
	`void start (Key c) {`
	`Key.chk (null); ...`
		subtype("Car","BMW"), ...
		method("BMW.start", "start", "BMW")
		calls("BMW.start", "Key.chk")...
			+calls("BMW.start", "Key.chk")

`class GM implements Car`
`void start (Key c ) {`
`if (c.on) { ....`
`...`
	`class GM implements Car`
	`void start (Key c ) {`
	`Key.chk (c );`
	`...`
		subtype("Car","GM"), ...
		method("GM.start", "start", "GM")
		calls("GM.start", "Key.chk")
		...
			-accesses("Key.on", "GM.start")
			+calls("GM.start", "Key.chk")

`class Kia implements Car`
`void start (Key c ) {`
`c.on = true; ....`
	`class Kia implements Car`
	`void start (Key c ) {`
	`...`
		subtype("Car","Kia"), ...
		method("Kia,start", "start", "Kia")
		...
			-accesses("Key.on", "Kia.start")

`class Bus {`
`void start (Key c) {`
`c.on = false;} }`
	`class Bus {`
	`void start (Key c);`
	`log(); } }`
		type("Bus")
		method("Bus,start", "start", "Bus")
		calls("Bus.start", "log")
			-accesses("Key.on", "Bus.start")
			+calls("Bus.start", "log")

`class Key {`
`boolean on = false;`
`void chk (Key c) { ...`
`void out () { ...`
	`class Key {`
	`boolean on = false;`
	`void chk (Key c) {`
	`void output (Key c){ ...`
		type ("Key")
		field("Key.on", "on", "Key")
		method ("Key.chk", "chk", "Key")
		method ("Key.output", "output", "Key") ...	

• For presentation purposes, fully qualified names are shortened, and the added and deleted facts in ΔFB are noted with + and − sign respectively. FB_o is omitted to save space; it can be inferred based on FB_n and ΔFB.

LSdiff

- Computing deltas based on FBo and FBn has two weaknesses:
  - time-consuming to read and understand
  - Due to the unstructured list of a potentially large number of facts.
  - it does not capture the surrounding context of changed fragments.

- To overcome this LSDiff calculate the union of all three fact-bases: FBo, FBn, and ΔFB.
  - More concise
  - Finding contextual facts

LSdiff

- Rule Styles

Table 2. LSdiff rule styles

Rule Styles Antecedent ⇒ Consequent	Example Rule and Its Interpretation
past_* ⇒ deleted_*	past_calls(m, "DB.exec") ⇒ deleted_calls(m, "DB.exec") All methods that called `DB.exec` in the old version deleted calls to `DB.exec`.
past_* ⇒ added_*	past_accesses("Log.on",m) ⇒ added_calls(m, "Log.trace") All methods that accessed `Log.on` in the old version added calls to `Log.trace`.
current_* ⇒ added_*	current_method(m, "getHost", t) ∧ current_subtype("Svc", t) ⇒ added_method(m, "getHost", t) All `getHost` methods in the `Svc`'s subtypes are newly added ones.
deleted_* ⇒ added_* added_* ⇒ deleted_*	deleted_method(m, "getHost", t) ⇒ added_inheritedfield("getHost", "Service", t) All types that deleted `getHost` method inherit `getHost` from `Service` instead.

Study

- Focus Group Study
- Participants: professional SE from large Ecommerce company.
- Software engineers (incl. Testers), Technical manager and Architect.
- Targeting early stages of product design
- Seeking target user's feedback on:
  - New product
  - Concepts
  - Messages

- Motivations:
  - Assessing LSdiff potential benefits before investing in its development.
  - Low-cost

Study

- Goals of the study
- Answering these questions:

1. In which task contexts do programmers need to understand code changes?
2. What are difficulties of using program differencing tools such as diff?
3. How can LSdiff complement existing uses of program differencing tools?

Study

- Screening Questionnaire
- Knowing about:
- software industry experience
- familiarity with Java
- how frequently they use diff and diff-based version control systems
- the size of code bases that they regularly work with

- All 16 answered the questionnaire
- 5 attended the focus group

Study

- Focus Group
- Each had development responsibilities
- 6 – 30 yrs industrial experience
- Each used related tool at least weekly
- Each did code review daily except one did weekly

- For one hour: the author worked as a moderator
- Introduction, demonstration, Diff best practices
- A hand-on trial to evaluated LSDiff output

- Their key Findings were driven by research questions discussed

Study

- Focus Group
- Followed by in-depth evaluation of LSdiff

Carol Revision 430.
SVN check-in message: Common methods go in an abstract class. Easier to extend/maintain/fix
Author: benoif @ Thu Mar 10 12:21:46 2005 UTC
723 lines of changes across 9 files (2 new files and 7 modified files)

Inferred Rules		
1	(50/50)	By this change, six classes inherit many methods from AbsRegistry class.
2	(32/32)	By this change, six classes implement NameService interface.
3	(6/8)	All methods that are included in JacORBCosNaming class and NameService interface are deleted except start and stop methods.
4	(5/6)	All host fields in the classes that implement NameService interface got deleted except LmiRegistry class.
5	(5/6)	All port fields in the classes that implement NameService interface got deleted except LmiRegistry class.
6	(5/6)	All getHost methods in the classes that implement NameService interface got deleted except LmiRegistry class.
7	(5/6)	All getPort methods in the classes that implement NameService interface got deleted except LmiRegistry class.
8	(5/6)	All setConfigProperties methods in the classes that implement NameService interface got deleted except LmiRegistry class.
9	(5/6)	All setHost methods methods in the classes that implement NameService interface got deleted except LmiRegistry class.
10	(5/6)	All setPort methods in the classes that implement NameService interface got deleted except LmiRegistry class.
11	(3/3)	All configurationProperties fields got deleted.
12	(3/4)	All DEFAULT_PORT_NUMBER fields are added by this change except JacORBCosNaming class.

Remaining Change Facts	
Added Class	AbsRegistry
Added Class	DummyRegistry
Added Method	JRMPRegistry.getRegistry
Deleted Field	IIOPCosNaming.DEFAUL_PORT
Deleted Field	JacORBCosNaming.started
Added Field Access	CmiRegistry's constructor added accesses to ClusterRegistry.DEFAULT_PORT field.
Added Field Access	JacORBCosNaming's constructor added accesses to JacORBCosNaming.DEFAULT_PORT_NUMBER field.

Study

- Focus Group
- Rule description

"All `host` fields in the classes that implement `NameService` interface were deleted except in `LmiRegistry`'s `host` field."

past_field(f,"host",t) ∧ past_subtype("NameService",t)
⇒ deleted_field(f,"host",t) except t="LmiRegistry"
Accuracy: (5/6)
deleted_field("CmiRegistry.host","host","CmiRegistry")
deleted_field("IIOPCosNaming.host","host","IIOPCosNaming")
deleted_field("JRMPRegistry.host","host","JRMPRegistry")
deleted_field("JacCosNaming.host","host","JacCosNaming")
deleted_field("JeremieRegistry.host","host","JeremieRegistry")
Exception: [t="LmiRegistry", f="LmiRegistry.host"]

Figure 2. Rule description

Study

- Focus Group
- HTML diff augmented with rules using CSDiff

40: public class CmiRegistry extends AbsRegistry implements NameService {
~~41:~~
~~42: /**~~
~~43: * URL~~
~~44: */~~
All port fields in the classes that implement NameService interface got deleted except LmiRegistry class.
~~45: private int port = ClusterRegistry.DEFAULT_PORT;~~
~~46:~~
~~47: /**~~
~~48: * Hostname to use~~
~~49: */~~
All host fields in the classes that implement NameService interface got deleted except LmiRegistry class.
~~50: private String host = null;~~
51:

Figure 3. HTML diff augmented with rules

Study

- Focus Group : Questions
- When do programmers use diff ?

- *The one that comes up the most frequently is a code review. . . Multiple times a day, someone makes changes and sends them out so that everybody can see it."*

- ". . . You need to see generational changes; not just this file and that file but how they went through a series of change motivations. . . "

Study

- Focus Group: Questions
- What would you like to have in an ideal program differencing tool?

The diff tools that I use, they are all file-oriented. They don't have notions, which I think you are trying to address is that, they don't have semantic relationships between different files. I want to say 'What did I change due to this problem?' It might have changed over 300 different files. I'd like to see not just one file but all 300 files that were included as a part of that. It is scaling up from a single source file to into spacing in which correlated change took place."

Study

- Focus Group: Summary
- Participant were positive about LSDiff
- *Asked to use it in work*
- They believed that LSdiff can help programmers reason about related changes effectively
- This study shows how LSdiff can complement existing tools but not how to assess it.

Assesment

- Comparison with structural deltas
- How often do individual changes form systematic change patterns? By measuring coverage
  - 90% coverage of facts 90/100 were inferred.
- How concisely does LSdiff describe structural differences in comparison other diff tools?
  - Lsdiff imporove conciseness by factor of 5.
- How much contextual information does LSdiff find from unchanged code fragments?
  - Measuring additional facts included from fact-bases.
  - LSdiff finds an average of 9.7 additional facts than ΔFB.

- Comparison with textual deltas
- LSdiff rules help programmers quickly understand the systematic changes and focus on other changes instead

Assesment

△FB: represents the existing code differences approach

Table 4. Comparison with △FB

	FB_o	FB_n	△FB	Rule	Fact	Cvrg.	Csc.	Ad'l.
10 revision pairs in carol (carol.objectweb.org)								
Min	3080	3452	15	1	3	59%	2.3	0.0
Max	10746	10610	1812	36	71	98%	27.5	19.0
Med	9615	9635	97	5	16	87%	5.8	4.0
Avg	8913	8959	426	10	20	85%	9.9	5.5
29 release pairs in dnsjava (www.dnsjava.org)								
Min	3109	3159	4	0	2	0%	1.0	0.0
Max	7200	7204	1500	36	201	98%	36.1	91.0
Med	4817	5096	168	3	24	88%	4.8	0.0
Avg	5144	5287	340	8	37	73%	8.4	14.9
10 version pairs in LSdiff								
Min	8315	8500	2	0	2	0%	1.0	0.0
Max	9042	9042	396	6	54	97%	28.9	12.0
Med	8732	8756	142	1	11	91%	9.8	0.0
Avg	8712	8783	172	2	17	68%	11.2	2.3
three data sets above								
Med	6650	6712	132	2	17	89%	7.3	0.0
Avg	6632	6732	302	7	27	75%	9.3	9.7

(m=3, a=0.75, k=2)

Threat to validity

- Participant view may be biased to practices in their organization.
- Internal Validity:

  the inferred rules are incomplete in nature as they depend on both input parameter settings

- The findings may not generalize to other data set. In terms of how it complements other diff tools

Conclusion

- LSdiff discovers and represents systematic structural differences as logic rules
- Participant thought LSdiff may complement diff tools by grouping sys Changes.
- It is not fair to compare tools targeting different level of abstraction
- It can helps detecting missed update
- 9.3 more concise results
- 9.7 additional facts that cannot be found by looking at the code.

Discussion