Discovering and Representing Systematic Code Changes

"What did Bob change? Did he implement the intended changes correctly?"

Miryung Kim
Electrical and Computer Engineering
University of Texas at Austin

David Notkin
Computer Science & Engineering
University of Washington

Motivating Scenarios

- “Did Bob implement the intended changes correctly?”
- “There’s a merge conflict. What did Alice change?”

What Changed?

Check-in comment:
"Common methods go in an abstract class. Easier to extend/maintain/fix" [Revision 429 of the *carol* project]

Changed Code		
File Name	Status	Lines
`DummyRegistry`	New	20 lines
`AbsRegistry`	New	133 lines
`JRMPRegistry`	Modified	123 lines
`JeremieRegistry`	Modified	52 lines
`JacORBCosNaming`	Modified	133 lines
`IIOPCosNaming`	Modified	50 lines
`CmiRegistry`	Modified	39 lines
`NameService`	Modified	197 lines
`NameServiceManager`	Modified	15 lines
Total Change: 9 files, 723 lines		

Was it indeed an ***extract superclass*** refactoring?
Were any parts of the refactoring missed?
Did Bob make some other changes along the way?

Diff Output

Changed Code		
File Name	Status	Lines
DummyRegistry	New	20 lines
AbsRegistry	New	133 lines
JRMPRegistry	Modified	123 lines
JeremieRegistry	Modified	52 lines
JacORBCosNaming	Modified	133 lines
IIOPCosNaming	Modified	50 lines
CmiRegistry	Modified	39 lines
NameService	Modified	197 lines
NameServiceManager	Modified	15 lines
Total Change: 9 files, 723 lines		

```
- public class CmiRegistry implements
NameService {
+ public class CmiRegistry extends
AbsRegistry implements NameService {
-     private int port = ...
-     private String host = null
-     public void setPort (int p) {
-        if (TraceCarol. isDebug()) { ...
-        }
-      }
-      public int getPort() {
-        return port;
-       }
-      public void setHost(String host)
{ ....
```

Existing Program Differencing Approaches

diff, Syntactic Diff (Cdiff), Semantic Diff, Jdiff, BMAT, Eclipse diff, UMLdiff, Change Distiller, etc.

Individually compare code elements
at particular granularities
using similarity measures

Systematic Changes

- Refactoring [Opdyke 92, Griswold 92, Fowler 99...]
- API update [Chow&Notkin 96, Henkel&Diwan 05, Dig&Johnson 05...]
- Crosscutting concerns [Kiczales et. al. 97, Tarr et. al. 99, Griswold 01...]
- Consistent updates on code clones [Miller&Myers 02, Toomim et. al. 04, Kim et. al. 05]

While high-level changes often consist of systematic transformations, existing program differencing tools ***do not identify systematic relationships.***

Limitations of Existing Differencing Approaches

Do not group related changes

Kia.java

```
+ ...
- start();
+ begin();
```

GM.java

```
+ ...
- start();
+ begin();
```

BMW.java

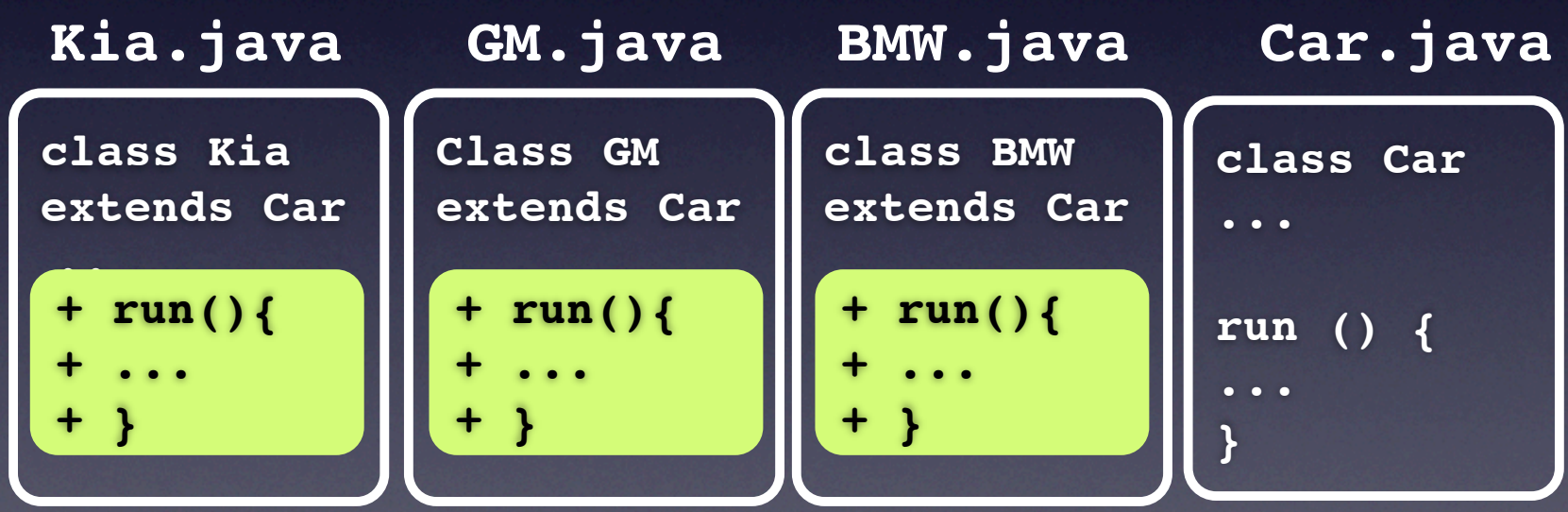
```
+ ...
- start();
+ begin();
```

Limitations of Existing Differencing Approaches

Difficult to notice missed changes

Limitations of Existing Differencing Approaches

Lack of contextual information



Outline

- Limitations of *diff*
- Rule-based program differencing approach
- LSdiff change-rule representation and inference algorithm
- Quantitative evaluation and focus group study

Our Logical Structural Diff Approach

- LSdiff computes structural differences between two versions using logic rules and facts.
- Each rule represents ***a group of transformations*** that share similar structural characteristics.
- Our inference algorithm ***automatically discovers*** such rules.

Our Contribution 1. Conciseness

Rule

```
+ ...
- start();
+ begin();
```

```
+ ...
- start();
+ begin();
```

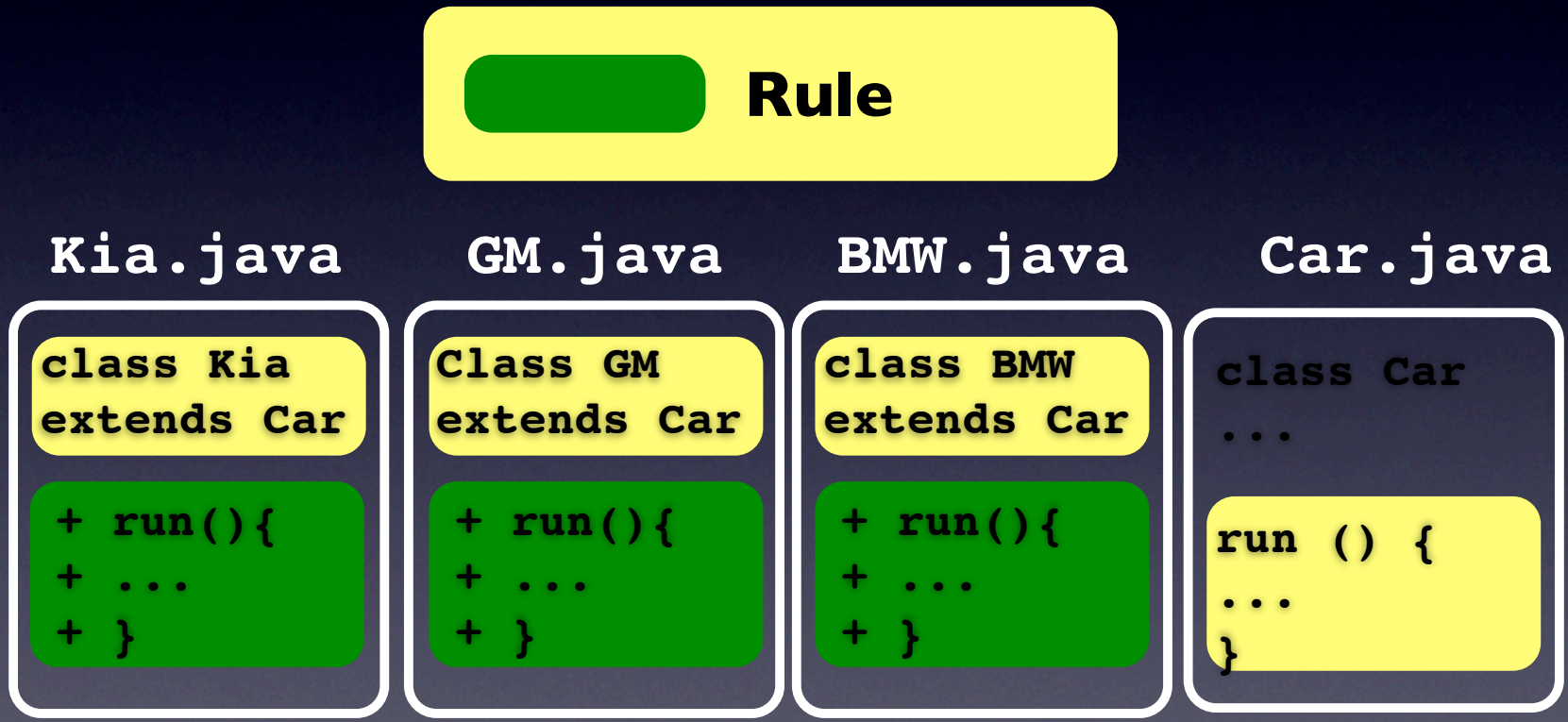
```
+ ...
- start();
+ begin();
```

Our Contribution 2.
Explicit Exceptions

Rule with an exception

in `GM.run()` method

Our Contribution 3. Additional Context



Outline

- Limitations of *diff*
- Rule-based program differencing approach
- LSdiff rule representation and inference algorithm
- Quantitative evaluation and focus group Study

Predicates in LSdiff

Code Elements
Structural Dependencies

```
package
type
method
field
return
fieldoftype
```

```
typeintype
accesses
calls
subtype
inheritedfield
inheritedmethod
```

Fact-base Representation

Old Program (FBo) **past_***

```
type("Bus",..)
method("Bus.start","start","Bus")
access("Key.on","Bus.start")
method("Key.out","out","Key")...
```

New Program (FBn) **current_***

```
type("Bus",..)
method("Bus.start","start","Bus")
calls("Bus.start","log")
method("Key.output","output","Key")...
```

Fact-Level Differences

Old Program (FBo) **past_***

```
type("Bus",..)
method("Bus.start","start","Bus")
access("Key.on","Bus.start")
method("Key.out","out","Key")...
```

\- ---------- set difference

New Program (FBn) **current_***

```
type("Bus",..)
method("Bus.start","start","Bus")
calls("Bus.start","log")
method("Key.output","output","Key")...
```

=

Differences (ΔFB) **added_* / deleted_***

```
deleted_access("Key.on","Bus.start")
added_calls("Bus.start","log")
deleted_method("Key.out","out","Key")
added_method("Key.output","output","Key")...
```

LSdiff Rule

Each rule represents ***systematic structural differences*** by relating groups of facts in the three fact-bases.

LSdiff Rule

Each rule represents ***systematic structural differences*** by relating groups of facts in the three fact-bases.

```
∀m ∀t method(m, “setHost”, t)
```

By binding some of a predicate’s arguments to universally quantified variables, a logic literal represents a group of similar facts at once.

LSdiff Rule

Each rule represents ***systematic structural differences*** by relating groups of facts in the three fact-bases.

```
∀m ∀t method(m, “setHost”, t)
∀t subtype(“Service”, t)
```

By binding some of a predicate’s arguments to universally quantified variables, a logic literal represents a group of similar facts at once.

LSdiff Rule

Each rule represents ***systematic structural differences*** by relating groups of facts in the three fact-bases.

```
∀m ∀t method(m, “setHost”, t)
∀t subtype(“Service”, t)
∀m calls(m, “SQL.exec”)
```

By binding some of a predicate’s arguments to universally quantified variables, a logic literal represents a group of similar facts at once.

LSdiff Rule

Each rule represents ***systematic structural differences*** by relating groups of facts in the three fact-bases.

```
∀m ∀t method(m, "setHost", t) ∧
subtype("Service", t)
⇒ calls(m, "SQL.exec")
```

Rules are horn clauses where a conjunction of logic literals implies a single consequent literal.

LSdiff Rule

Each rule represents ***systematic structural differences*** by relating groups of facts in the three fact-bases.

```
∀m ∀t past_method(m, “setHost”, t) ∧
past_subtype(“Service”, t)
⇒ deleted_calls(m, “SQL.exec”)
```

Rule styles are restricted to represent regularities about changes between two versions.

LSdiff Rule

Each rule represents ***systematic changes*** by relating groups of facts in the three fact-bases.

```
∀m ∀t past_method(m, “setHost”, t) ∧
past_subtype(“Service”, t)
⇒ deleted_calls(m, “SQL.exec”)
[except t=“NameSvc” m=”NameSvc.setHost”]
```

Rules explicitly note exceptions.

LSdiff Rule Example

```
∀m ∀t past_method(m, "setHost", t) ∧
past_subtype("Service", t)
⇒ deleted_calls(m, "SQL.exec")
[except t="NameSvc" m="NameSvc.setHost"]
```

"All `setHost` methods in `Service`'s subclasses in the old version deleted calls to `SQL.exec` except the `setHost` method in the `NameSvc` class."

```
deleted_calls("CmiSvc.setHost","SQL.exec")
deleted_calls("RmiSvc.setHost","SQL.exec")
deleted_calls("LmiSvc.setHost","SQL.exec")
exception [t="NameSvc" m="NameSvc.setHost"]
```

3 matches, 1 exception, accuracy 0.75

LSdiff Algorithm Overview

input: two program versions

1. Extract a set of logic facts from programs using JQuery [Jensen & DeVolder 03] and compute fact-level differences
2. Learn rules by using our customized inductive logic programming algorithm
3. Select a subset of rules and then winnow out the facts in ΔFB using the learned rules

output: logic rules and facts that explain structural differences

Step 1. Fact-base Preparation

Old Program (FBo) **past_***

```
type("Bus",..)
method("Bus.start","start","Bus")
access("Key.on","Bus.start")
method("Key.out","out","Key")...
```

-

New Program (FBn) **current_***

```
type("Bus",..)
method("Bus.start","start","Bus")
calls("Bus.start","log")
method("Key.output","output","Key")...
```

A fact-base program representation approach has been used by many tools such as JQuery [Jenzen&DeVolder 03], CodeQuest [Hajiev et. al. 06], Grok [Holt et. al.] , etc.

Step 1. Fact-base Preparation

Old Program (FBo) past_*

```
type("Bus",..)
method("Bus.start","start","Bus")
access("Key.on","Bus.start")
method("Key.out","out","Key")...
```

\- ..... set difference

New Program (FBn) current_*

```
type("Bus",..)
method("Bus.start","start","Bus")
calls("Bus.start","log")
method("Key.output","output","Key")...
```

=

Differences (ΔFB) added_* / deleted_*

```
deleted_access("Key.on","Bus.start")
added_calls("Bus.start","log")
deleted_method("Key.out","out","Key")
added_method("Key.output","output","Key")
```

Step 1. Fact-base Preparation

Old Program (FBo) past_*

```
type("Bus",..)
method("Bus.start","start","Bus")
access("Key.on","Bus.start")
method("Key.out","out","Key")...
```

− ..... set difference

New Program (FBn) current_*

```
type("Bus",..)
method("Bus.start","start","Bus")
calls("Bus.start","log")
method("Key.output","output","Key")...
```

=

Differences (ΔFB) added_* / deleted_*

```
deleted_access("Key.on","Bus.start")
added_calls("Bus.start","log")
deleted_method("Key.out","out","Key")
added_method("Key.output","output","Key")
```

Remove spurious facts using inferred renamings [Kim et al.'s ICSE 2007]

Step 1. Fact-base Preparation

Old Program (FBo) past_*

```
type("Bus",..)
method("Bus.start","start","Bus")
access("Key.on","Bus.start")
method("Key.out","out","Key")...
```

\- ..... set difference

New Program (FBn) current_*

```
type("Bus",..)
method("Bus.start","start","Bus")
calls("Bus.start","log")
method("Key.output","output","Key")...
```

=

Differences (ΔFB) added_* / deleted_*

```
deleted_access("Key.on","Bus.start")
added_calls("Bus.start","log")
```

Remove spurious facts using inferred renamings [Kim et al.'s ICSE 2007]

Step 2. Learn Rules

- Our rule learner uses a ***bounded depth search*** algorithm with beam search heuristics to find rules
- We have input parameters that determine the ***validity*** of a rule.
  - *a*: the minimum accuracy of a rule
  - *m*: the minimum # of facts a rule must match
  - k : the maximum # of literals in an antecedent
  - β: the window size for beam search

Step 2. Learn Rules

```
R := {} // a set of ungrounded rules.
D := reduced ΔFB using default winnowing rules
L := {} // a set of valid learned rules.
for each antecedent size, i = 0...k :
    R := extend all rules in R by adding all
    possible literals.
    for each ungrounded rule, r:
         for each possible grounded rule g of r:

              if (g is valid) L:= L ∪ g.

    R := select the best β rules in R.
      D := D - { facts covered by L}
```

Step 3. Post Processing

- Select a subset of `L` that cover the same set of facts covered by `L` using the `SET-COVER` algorithm.
- Output the selected rules and remaining uncovered facts in `ΔFB`.

LSdiff Output

- "All methods that removed calls to the `SQL.exec` method added calls to the `SafeSQL.exec` method "

```
deleted_calls(m,"SQL.exec")=>
added_calls(m,"SafeSQL.exec")
```

- "All `setHost` methods in `Service`'s subclasses in the old version deleted calls to `SQL.exec` except the `setHost` method in the `NameSvc` class.

```
past_subtype("Service", t) ∧
past_method(m, "setHost", t)
⇒ deleted calls(m, "SQL.exec")

except t="NameSvc"
```

Outline

- Limitations of *diff*
- Rule-based program differencing approach
- LSdiff change-rule representation and inference algorithm
- Quantitative evaluation and focus group study

LSdiff Quantitative Evaluation

1. How often do individual changes form systematic change patterns? Measure *coverage*, # of facts in ΔFB matched by inferred rules
2. How concisely does LSdiff describe structural differences in comparison to existing differencing approach at the same abstraction level? Measure *conciseness*, ΔFB / (# rules + # facts)
3. How much contextual information does LSdiff find from unchanged code fragments? Measure *the number of facts mentioned by rules but* are not contained in ΔFB

LSdiff Quantitative Evaluation

	FBo/FBn	ΔFB	Rule	Fact	Cvrg.	Conc.	Ad'tl.
carol 10 revisions	3080~10746	15~1812	1~36	3~71	59~98%	2.3 ~27.5	0~19
dnsjava 29 releases	3109~7204	4~1500	0~36	2~201	0~98%	1.0 ~36.1	0~91
LSdiff 10 versions	8315~9042	2~396	0~6	2~54	0~97%	1.0 ~28.9	0~12

a=0.75, m=3, k=2, β=100

LSdiff Quantitative Evaluation

	FBo/FBn	ΔFB	Rule	Fact	Cvrg.	Conc.	Ad'tl.
carol 10 revisions							0~19
dnsjava 29 releases							0~91
LSdiff 10 versions	8315~9042	2~396	0~6	2~54	0~97%	1.0 ~28.9	0~12

On average, 75% coverage, 9.3 times conciseness improvement, and 9.7 additional contextual facts

a=0.75, m=3, k=2, β=100

Textual Delta vs. LSdiff

	Textual Delta				LSdiff	
	Changed Files	Changed Lines	Hunk	% Touched	Rule	Fact
carol 10 revisions	1~35	67~4313	9~132	1~19%	1~36	3~71
dnsjava 29 releases	1~117	5~15915	1~344	2~100%	0~36	2~201
LSdiff 10 versions	2~11	9~747	2~39	2~9%	0~6	2~54

a=0.75, m=3, k=2, β=100

Textual Delta vs. LSdiff

	Textual Delta				LSdiff	
	Changed Files	Changed Lines	Hunk	% Touched	Rule	Fact
carol 10 revisions						3~71
dnsjava 29 releases						2~201
LSdiff 10 versions	2~11	9~747	2~39	2~9%	0~6	2~54

When an average TD consists of 997 lines across 16 files, LSdiff outputs an average of 7 rules and 27 facts.

a=0.75, m=3, k=2, β=100

Focus Group Study

- Pre-screener survey
- Participants: five professional software engineers
  - industry experience ranging from 6 to over 30 years
  - use *diff* and *diff*-based version control system daily
  - review code changes daily except one who did weekly
- One hour structured discussion
  - I worked as the moderator. We also had a note-taker transcribe the discussion. Discussion was audio-taped and transcribed.

Focus Group Hands-On Trial

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).

Overview

Generated based on LSDiff output.

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.

A hand-generated html based on LSdiff output

http://users.ece.utexas.edu/~miryung/LSDiff/carol429-430.htm

Focus Group Hands-On Trial

```
46: public class IIOPCosNaming extends AbsRegistry implements NameService {
47:
48:     /**
49:      * Default port number ( 12350 for default)
50:      */
All DEFAULT PORT NUMBER fields are added fields except JacORBCosNaming class.
51:     private static final int DEFAUL_PORT DEFAULT_PORT_NUMBER = 12350;
52:
53:     /**
54:      * Sleep time to wait
55:      */
56:     private static final int SLEEP_TIME = 2000;
57:
58:     /**
59:      * port number
60:      */
All port fields in the classes that implement NameService interface got deleted except LmiRegistry class.
 61:     private int port = DEFAUL_PORT;
62:
63:     /**
64:      * Hostname to use
65:      */
All host fields in the classes that implement NameService interface got deleted except LmiRegistry class.
 66:     private String host = null;
```

Show related changes

http://users.ece.utexas.edu/~miryung/LSDiff/carol429-430.htm

Focus Group Participants' Comments

"You can't infer the intent of a programmer, but this is pretty close."

"This 'except' thing is great!"

"You can start with the summary of changes and dive down to details using a tool like *diff*."

Focus Group Participants' Comments

"This looks great for big architectural changes, but I wonder what it would give you if you had lots of random changes."

"This wouldn't be used if you were just working with one file."

"This will look for relationships that do not exist."

Other Related Work

- Identification of related changes
- Logic-based program representation
- Source transformation languages and tools
- Framework evolution

Conclusions

- LSdiff ***automatically identifies*** systematic structural differences as logic ***rules***.
- LSdiff represents 75% structural differences as rules on average, improving conciseness measure by 9.3 times on average.
- Our focus group study shows that LSdiff is promising as a complement to *diff*'s file-based approach and can help programmers discover potential bugs.

Acknowledgment: Special thanks to Marius Nita and Jonathan Beall