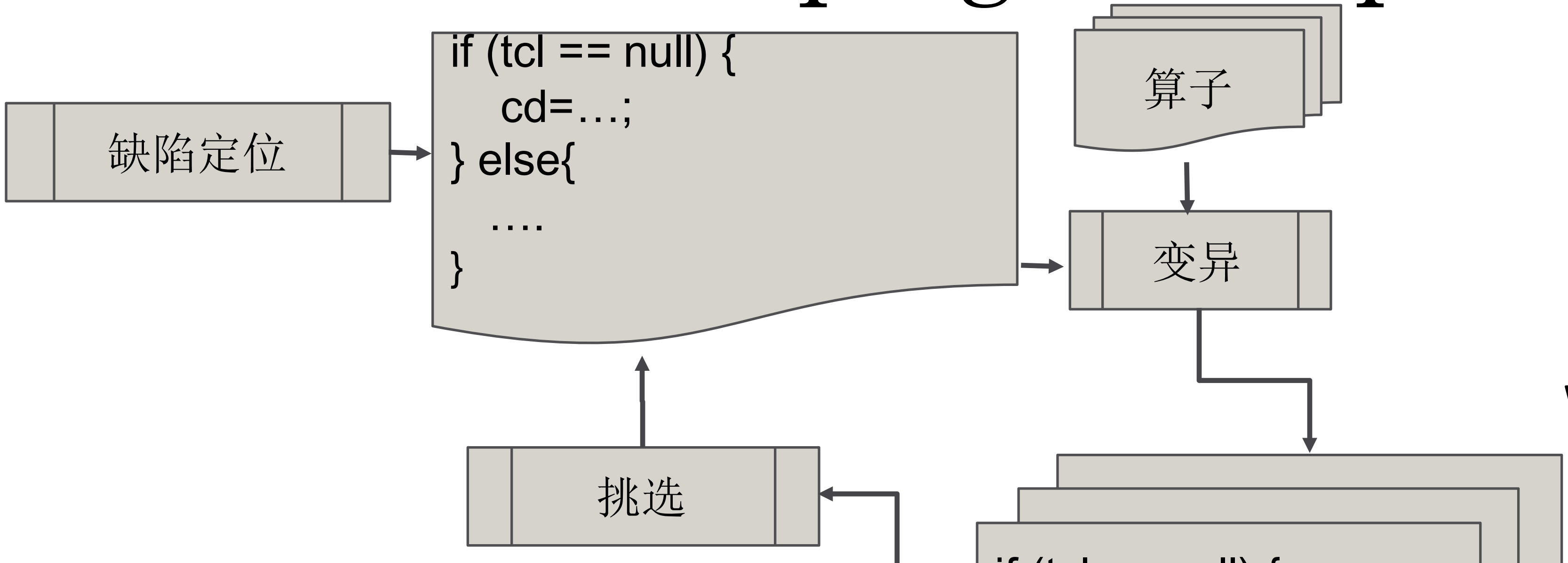
Boosting Complete-code Tools for Partial Program Analysis and its Applications

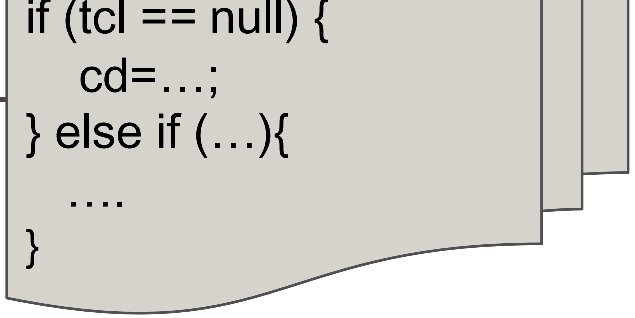
Hao Zhong

Shanghai Jiao Tong University

2017.8

Automatic program repair





Westley Weimer

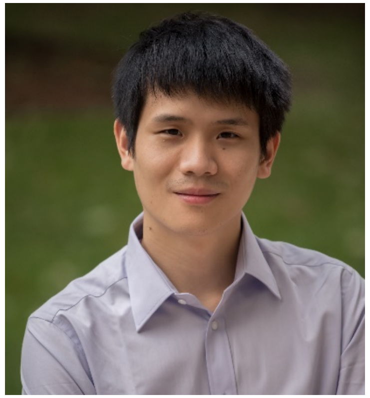
DISTINGUISHED PAPER AWARD

Sung Kim

DISTINGUISHED PAPER AWARD

- 自动化程度高
- 修复未知缺陷

Martin Monperrus



Fan Long

My previous work on bug fixes

- An empirical study on real bug fixes
  - **Hao Zhong** and Zhengdong Su. *In Proc. International Conference on Software Engineering (ICSE)*, pages 913-923, 2015.

Our empirical study

- Understanding the challenges and potential by analyzing manual bug fixes.
- Dataset

TABLE I
DATASET

Name	LOC	Bug			Commit			
		F	I	%	T	N	K	%
Aries	142,110	497	490	98.6%	5,318	839	226	20.0%
Cassandra	121,170	1,374	1,236	90.0%	6,825	1,708	281	29.1%
Derby	659,426	2,433	2,022	83.1%	10,285	4,624	346	48.3%
Lucene/Solr	677,873	3,145	2,226	70.8%	26,890	4,234	2,019	23.3%
Mahout	121,084	457	407	89.1%	3,632	553	289	23.2%
Total	1,721,663	7,906	6,381	80.7%	52,950	11,958	3,161	28.6%

F: fixed bugs in bug reports. I: bugs whose fixes are identified by issue number; T: total commits; N: bug-fix commits that are identified by issue number; K: bug-fix commits that are identified by keywords.

Reported bug

On-demand bug

Partial Program Analysis for Java

ChangeDistiller

Partial program analysis

- B. Dagenais and L. J. Hendren. Enabling static analysis for partial Java programs. In *Proc. 23rd OOPSLA*, pages 313–328, 2008.
  - We consider a partial program to be a subset of a program's source files.

- Michele Tufano, Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrea De Lucia, and Denys Poshyvanyk. There and back again: Can you compile that snapshot? Journal of Software: Evolution and Process (2016).
  - On average, only 38% of the change history of project systems is currently successfully compilable.
  - Space and time limit
- Existing tools
  - PPA、ChangeDistiller、Spoon、groums

Partial program analysis

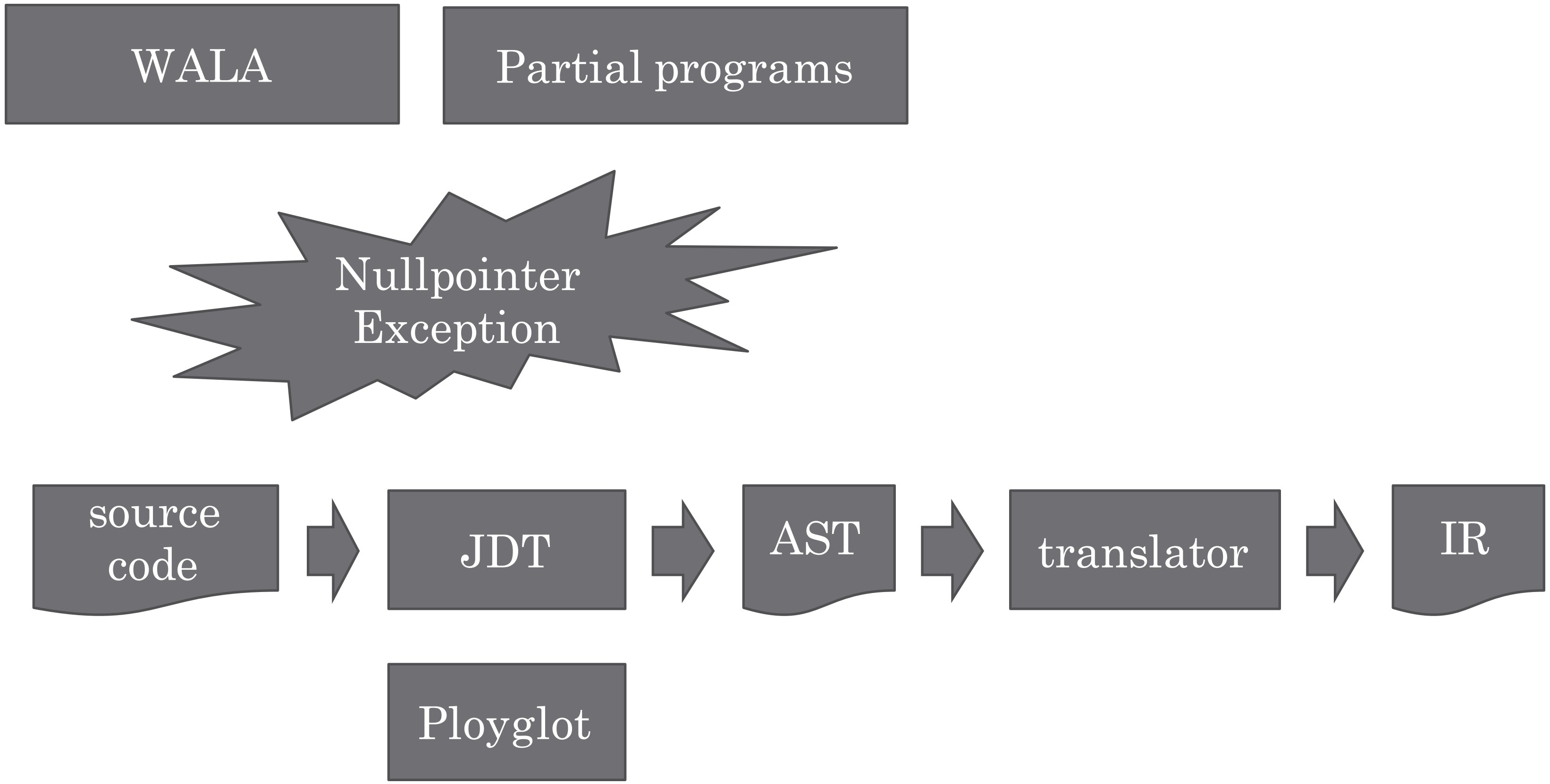
- A. Mishne, S. Shoham, and E. Yahav. Typestate-based semantic code search over partial programs. In Proc. OOPSLA, pages 997–1016, 2012.
  - our approach extracts a possibly partial temporal specification from each snippet using a **relatively** precise static analysis …
- D. Kim, J. Nam, J. Song, and S. Kim. Automatic patch generation learned from human-written patches. In Proc. ICSE, pages 802–811, 2013.
  - **To reduce manual inspection time**, we first gathered similar patches using groums [16] …
  - We could gather patches having the same differences into a group. Although a patch group is not necessarily a fix pattern, doing this can substantially **reduce manual inspection time**. …
  - We then **examined** the root causes of bugs and how the corresponding patches specifically resolved the bugs.

Partial program analysis

- Hao Zhong, Xiaoyin Wang, Analyzing partial programs using whole program static analysis tools. In Proc. ASE, to appear, 2017.

- Upon discussion, the reviewers agreed that the presented work provides a novel application of … However, the reviewers have serious concerns about the evaluation of the presented work. Given that the presented work is built on top of GRAPA's results, but GRAPA is not made available, it would be difficult to assess the impact of GRAPA and the value of the contribution of the paper. In addition, …

GRAPA



WALA
Partial programs
Nullpointer Exception
source code
JDT
AST
translator
IR
Ploygiot

GRAPA

WALA

PPA

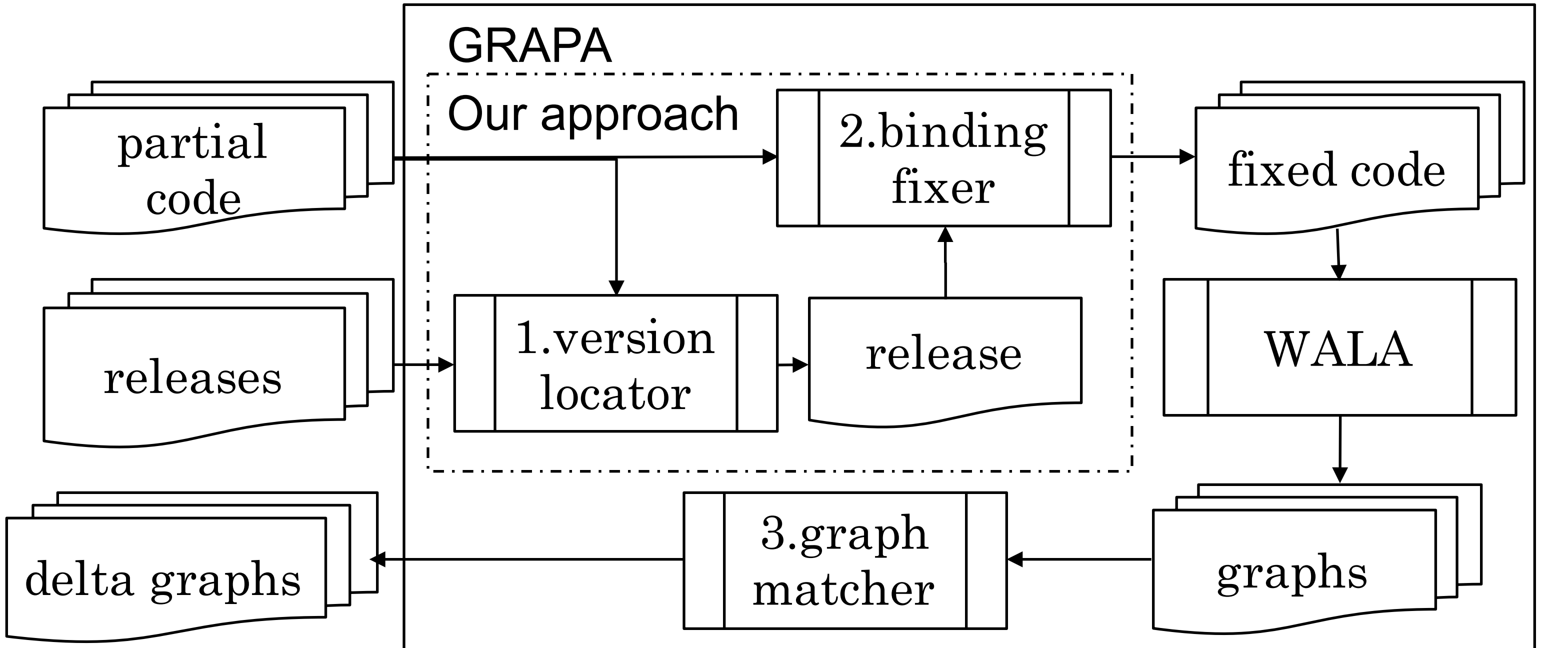
On average, the type inference and method binding passes correctly recovered 52% of the types that were previously unknown

Project	No Version		No PPA		No. Extended PPA	
	Success	%	Success	%	Success	%
Aries	339	61.90%	353	64.60%	435	79.6%
Cassandra	899	26.10%	1,546	44.90%	2,332	67.7%
Derby	637	24.90%	1,088	42.50%	1,882	73.5%
Mahout	189	33.80%	351	62.60%	442	79.0%
Total	2,041	28.70%	3,307	46.50%	5,113	71.9%

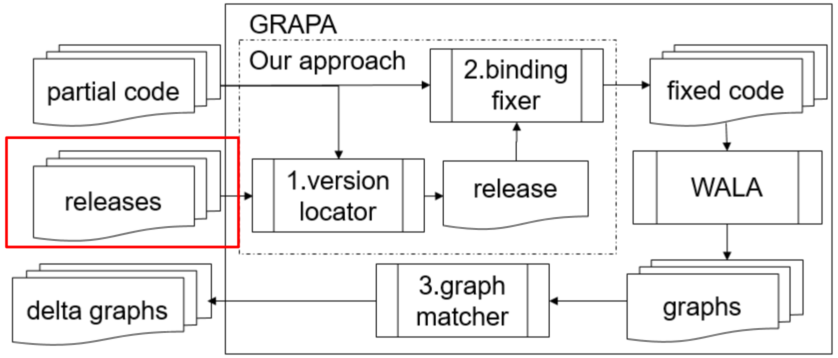
	Outcome	Lucene	JFreeChart	Jython	Spring
baseline	% correct	89.20	89.23	81.20	87.16
	% unknown	8.23	9.02	13.88	8.01
	% h. correct	0.29	1.12	1.91	1.12
	% erroneous	2.29	0.63	3.01	3.71
inf. no bind.	% correct	93.48	94.12	87.94	90.78
	% unknown	3.52	3.89	6.63	4.30
	% h. correct	0.34	1.16	2.36	1.20
	% erroneous	2.67	0.83	3.07	3.71
inf. bind.	% correct	93.80	94.40	88.22	90.97
	% unknown	2.46	3.56	6.21	4.07
	% h. correct	0.38	1.19	2.44	1.24
	% erroneous	2.71	0.85	3.13	3.72
	Total Facts	87706	250155	312907	325641

Table 4. Partial program analysis results

GRAPA


GRAPA
Our approach
partial code
releases
delta graphs
2.binding fixer
1.version locator
release
fixed code
WALA
graphs
3.graph matcher

GRAPA




Apache Software Foundation / Apache Cassandra / Download

Apache CASSANDRA

Home Download Documentation Community

Downloading Cassandra

Latest version

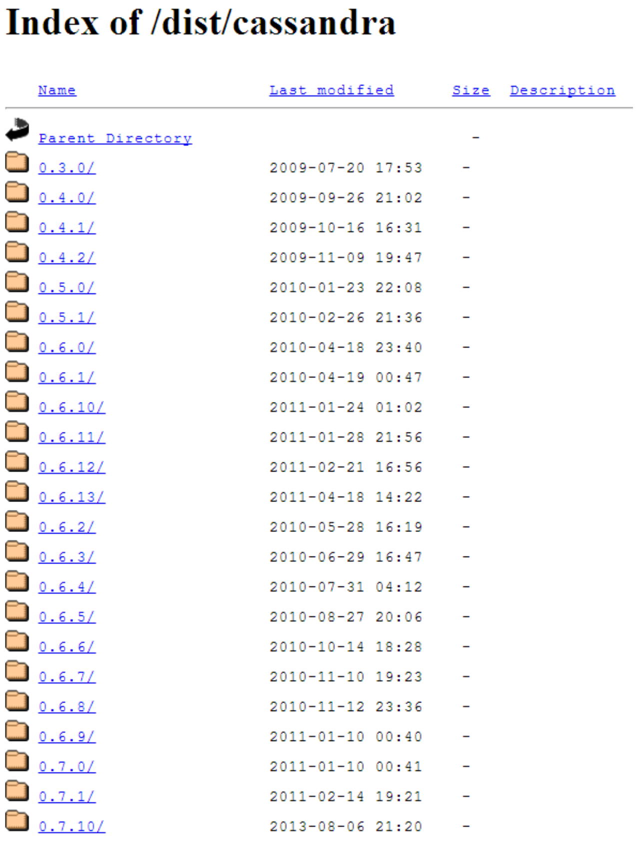
Download the latest Apache Cassandra 3.11 release: 3.11.0 (pgp, md5 and sha1), released on 2017-06-23.

Older supported releases

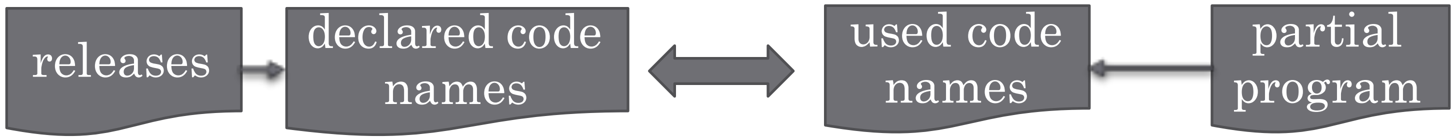
The following older Cassandra releases are still supported:

- Apache Cassandra 3.0 is supported until **6 months after 4.0 release (date TBD)**. The latest release is 3.0.14 (pgp, md5 and sha1), released on 2017-06-23.
- Apache Cassandra 2.2 is supported until **4.0 release (date TBD)**. The latest release is 2.2.10 (pgp, md5 and sha1), released on 2017-06-26.
- Apache Cassandra 2.1 is supported until **4.0 release (date TBD)** with **critical fixes only**. The latest release is 2.1.18 (pgp, md5 and sha1), released on 2017-06-26.

Older (unsupported) versions of Cassandra are archived here.


Index of /dist/cassandra

Name	Last modified	Size	Description
Parent Directory		-	
0.3.0/	2009-07-20 17:53	-	
0.4.0/	2009-09-26 21:02	-	
0.4.1/	2009-10-16 16:31	-	
0.4.2/	2009-11-09 19:47	-	
0.5.0/	2010-01-23 22:08	-	
0.5.1/	2010-02-26 21:36	-	
0.6.0/	2010-04-18 23:40	-	
0.6.1/	2010-04-19 00:47	-	
0.6.10/	2011-01-24 01:02	-	
0.6.11/	2011-01-28 21:56	-	
0.6.12/	2011-02-21 16:56	-	
0.6.13/	2011-04-18 14:22	-	
0.6.2/	2010-05-28 16:19	-	
0.6.3/	2010-06-29 16:47	-	
0.6.4/	2010-07-31 04:12	-	
0.6.5/	2010-08-27 20:06	-	
0.6.6/	2010-10-14 18:28	-	
0.6.7/	2010-11-10 19:23	-	
0.6.8/	2010-11-12 23:36	-	
0.6.9/	2011-01-10 00:40	-	
0.7.0/	2011-01-10 00:41	-	
0.7.1/	2011-02-14 19:21	-	
0.7.10/	2013-08-06 21:20	-	



GRAPA

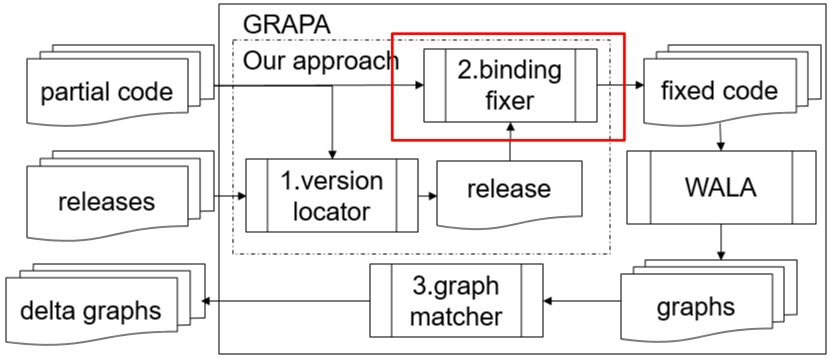


TABLE I: The inference strategies of PPA.

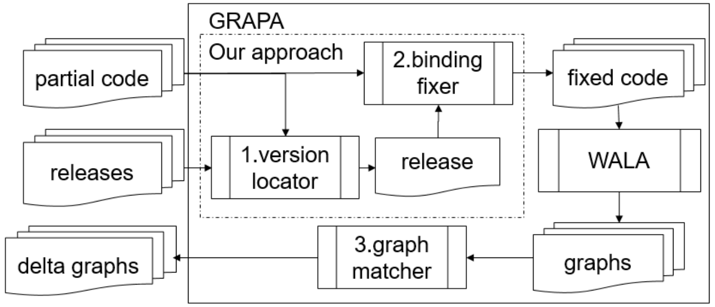
Strategy	Example
Assignment	`B.field=''Hello World'';`→{field,unknown, ≽ java.lang.String}
Return	`int m(){ return method()};`→{method,unknown, ≼ int}
Method	`f1=m1(f2); D m1(E p);`→{f1,unknown, ≽ D} and {f2,unknown, ≼ E}
Condition	`if(f){...};`→{f,unknown, boolean}
Binary and unary operators	`int i = f-10;`→{f,unknown, ≼ int}
Array	`f1 = f2[f3];`→{f3,unknown, ≼ int} and {f8,unknown,unknown[]}
Switch	`switch(f){...};`→{f,unknown, ≼ int}
Conditional	`int i=f1?1:f2;`→{f1,unknown,boolean} and {f2,unknown, ≼ int}

1. Variable inference strategy
2. Field inference strategy
3. Switch inference strategy

private ... String getTSIAQuotesOrderByChangeSQL =...
public MarketSummaryDataBean getMarketSummary() ... {
PreparedStatement stmt = getStatement(
getTSIAQuotesOrderByChangeSQL, ...);...}

ARIES-241

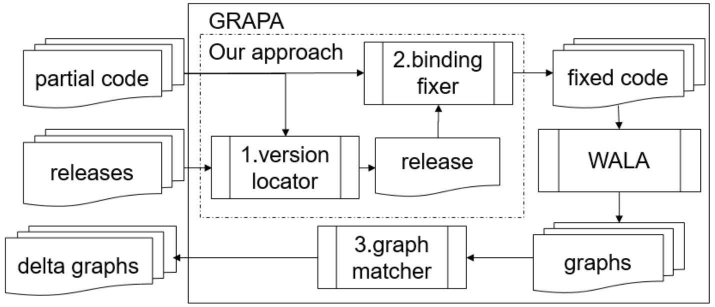
GRAPA



- RQ1. How effectively does GRAPA resolve unknown code names of partial programs?

Name	Version	Fix	Success	Failed			
				unsupported	not resolved	defect	%
Aries	4	547	533	1	10	0	97.4%
Cassandra	116	3,444	3,405	11	16	12	98.9%
Derby	20	2,560	2,538	8	12	0	99.1%
Mahout	13	560	494	2	61	3	88.2%
Total		7,111	6,970	22	99	15	98.0%

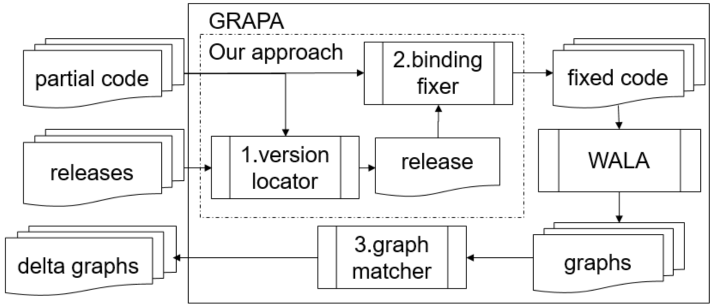
GRAPA



- RQ2. What is the accuracy of GRAPA to resolve unknown code names?

Name	Version	Bug Fix	File	Method	Same	%
Aries	1.0	24	38	456	452	99.1%
Cassandra	3.0.0	14	22	585	558	95.4%
Derby	10.11.1.1	37	88	2,088	1,995	95.5%
Mahout	0.10.0	16	25	661	636	96.2%
Total		91	173	3,790	3,641	96.1%

GRAPA



- RQ3. What is the effectiveness of GRAPA's internal techniques?

Project	No Version		No PPA		No. Extended PPA	
	Success	%	Success	%	Success	%
Aries	339	61.90%	353	64.60%	435	79.6%
Cassandra	899	26.10%	1,546	44.90%	2,332	67.7%
Derby	637	24.90%	1,088	42.50%	1,882	73.5%
Mahout	189	33.80%	351	62.60%	442	79.0%
Total	2,041	28.70%	3,307	46.50%	5,113	71.9%

More advanced empirical study

- M. Martinez, W. Weimer, and M. Monperrus. Do the fix ingredients already exist? an empirical inquiry into the redundancy assumptions of program repair approaches. In Proc. 36th ICSE, pages 492–495, 2014.
- E. T. Barr, Y. Brun, P. Devanbu, M. Harman, and F. Sarro. The plastic surgery hypothesis. In Proc. ESEC/FSE, pages 306–317, 2014.

- Le X B D, Lo D, Le Goues C. History driven program repair. In Proc. SANER, 2016, 1: 213-224.
- F. Long and M. Rinard. Automatic patch generation by learning correct code. In Proc. 43rd POPL, pages 298–312, 2016.

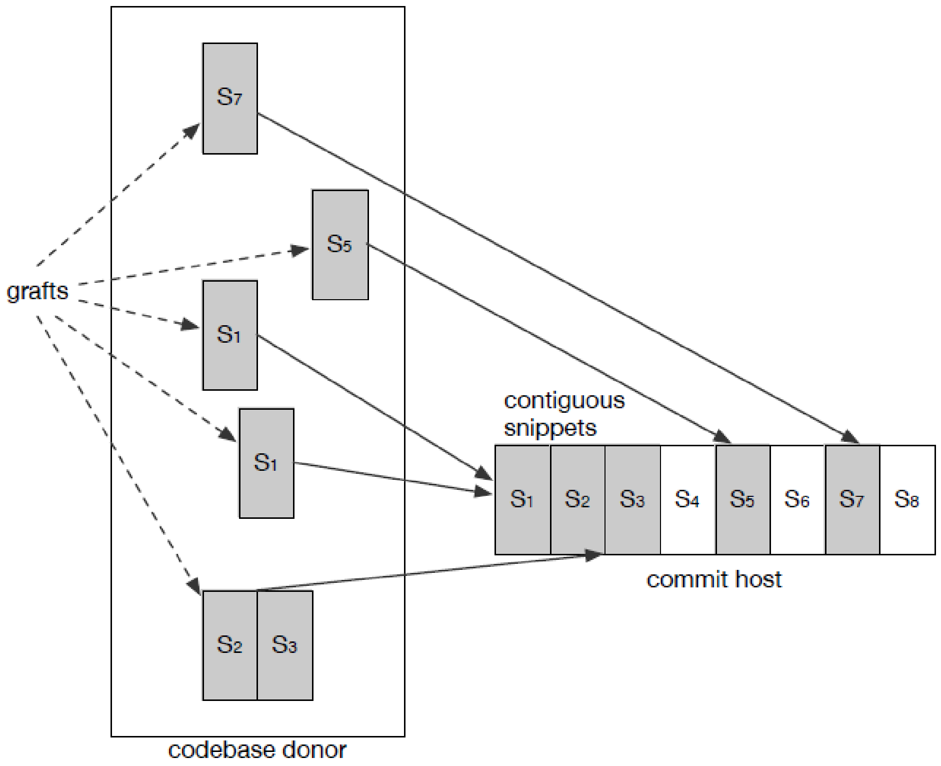
More advanced empirical study

- H. Zhong and N. Meng. Poster: An empirical study on using hints from past fixes. In Proc. ICSE, 2017.

Project	Both				Structure				Code Name				Fix
	FI	%	PI	%	FS	%	PS	%	FN	%	PN	%	
aries	8	1.8%	10	2.3%	38	8.6%	144	32.6%	16	3.6%	37	8.4%	442
cassandra	68	2.8%	115	4.7%	383	15.6%	1,202	48.8%	126	5.1%	327	13.3%	2,463
derby	37	1.5%	44	1.8%	249	10.4%	865	36.2%	63	2.6%	169	7.1%	2,392
mahout	9	2.1%	14	3.2%	47	10.7%	155	35.4%	12	2.7%	29	6.6%	438
total	122	2.1%	183	3.2%	717	12.5%	2,366	41.3%	217	3.8%	562	9.8%	5,735

More advanced empirical study

- E. T. Barr, Y. Brun, P. Devanbu, M. Harman, and F. Sarro. The plastic surgery hypothesis. In Proc. ESEC/FSE, pages 306–317, 2014.



More advanced empirical study

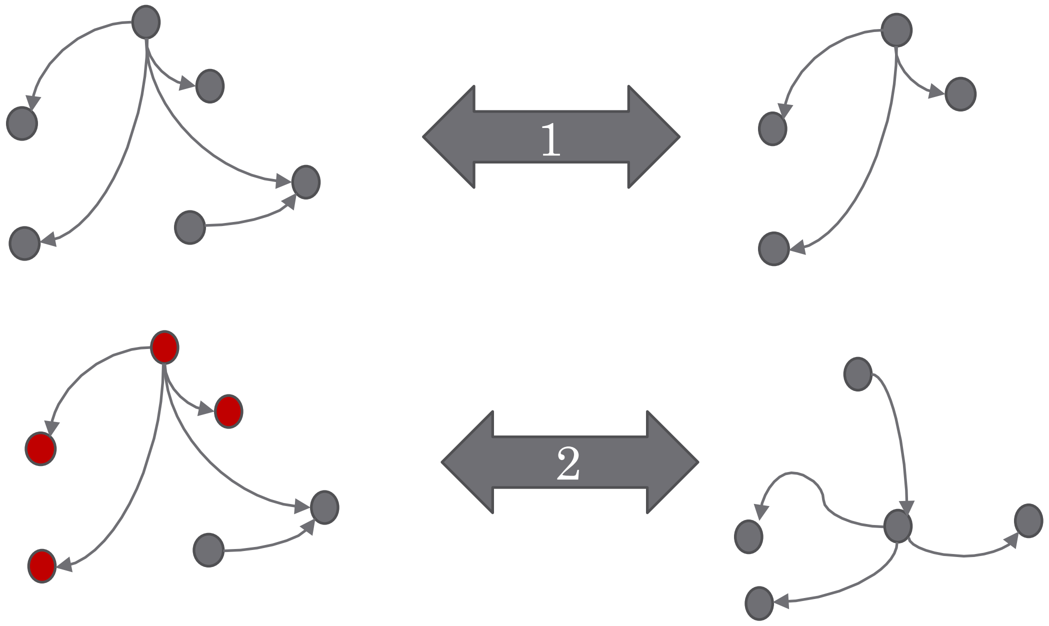
- M. Martinez, W. Weimer, and M. Monperrus. Do the fix ingredients already exist? an empirical inquiry into the redundancy assumptions of program repair approaches. In Proc. 36th ICSE, pages 492–495, 2014.

Table 1: The temporal redundancy of six open-source applications.

Program	Acceptable Commits	Line granularity				Token granularity			
		Global		Local		Global		Local	
		Temporal redundancy	Pool Size	Temporal redundancy	Pool Size	Temporal redundancy	Pool Size	Temporal redundancy	Pool Size
log4	1687	**9%**	43313	**6%**	57	**39%**	14294	**19%**	71
junit	713	**17%**	8855	**16%**	18	**43%**	3256	**29%**	72.5
pico	157	**3%**	16911	**2%**	22.5	**31%**	6273	**8%**	46
collections	1019	**7%**	25406	**4%**	35	**52%**	4163	**23%**	85.5
math	2210	**6%**	69943	**4%**	37	**45%**	20742	**18%**	100.5
lang	1290	**8%**	22330	**6%**	63	**50%**	6692	**29%**	98
C 1	C 2	C 3	C 4	C 5	C 6	C 7	C 8	C 9	C 10

More advanced empirical study

- No splitting



More advanced tools

- **Hui Liu**, Qiurong Liu, Cristian-Alexandru Staicu, Michael Pradel, Yue Luo. *Nomen est Omen: Exploring and Exploiting Similarities between Argument and Parameter Names.* In Proc. 38th ICSE, 1063-1073, 2016
- Hao Zhong, Lu Zhang, Tao Xie, and Hong Mei. Inferring resource specifications from natural language API documentation. In Proc. 24th ASE, pages 307–318, 2009.

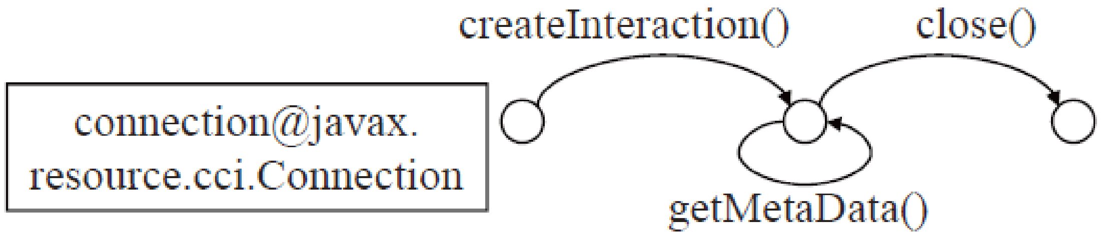
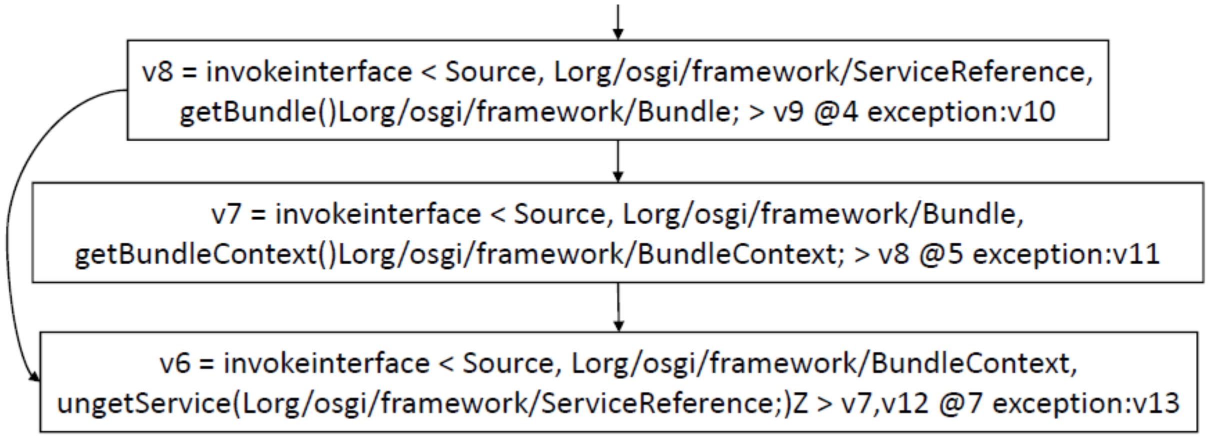


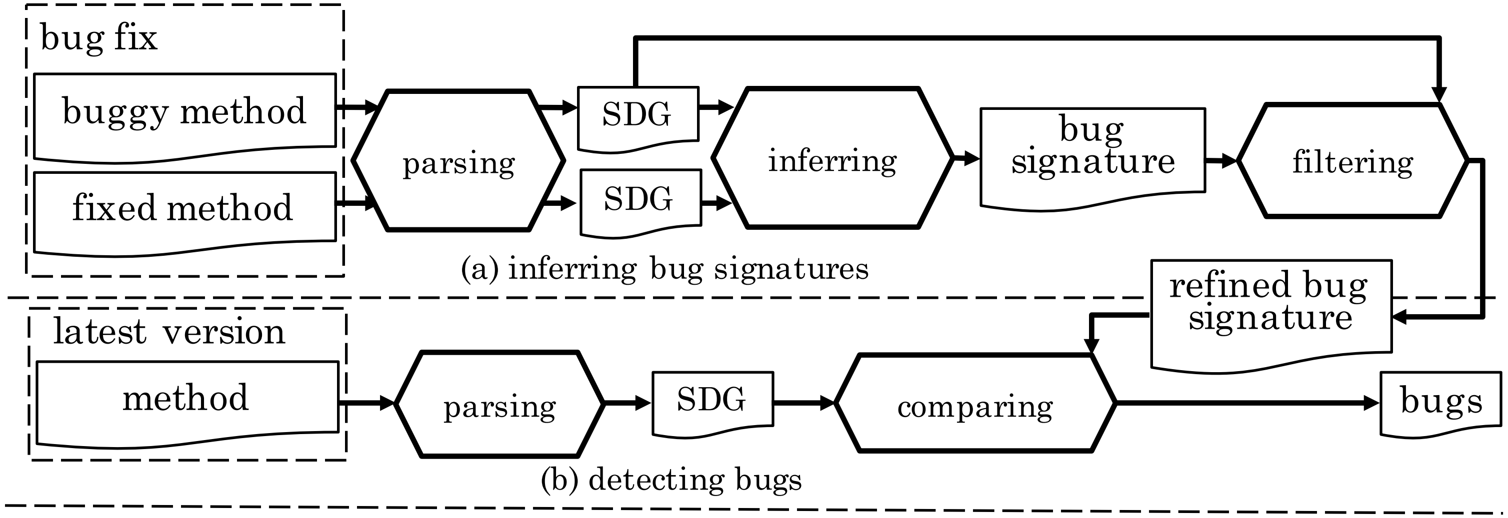
Figure 2: Resource specifications



(a) The inferred bug signature

More advanced tools

- Hao Zhong, Xiaoyin Wang, Detecting bugs with past fixes. Draft.



DEPA

More advanced tools

- Jingyue Li and Michael D Ernst. 2012. CBCD: Cloned buggy code detector. In Proc. ICSE. 310–320.
  - After excluding such cases, the evaluation used 5 Git bugs, 14 PostgreSQL bugs, and 34 Linux bugs.

- Chengnian Sun and Siau-Cheng Khoo. 2013. Mining succinct predicated bug signatures. In Proc. ESEC/FSE. 576–586.

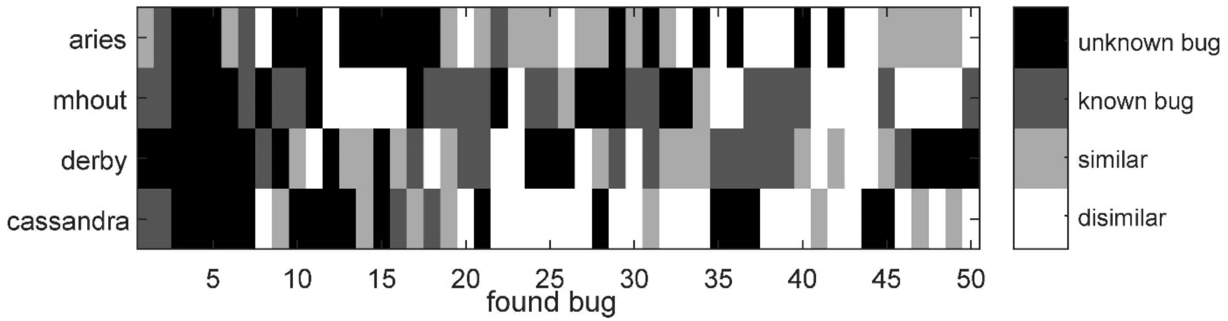
Subject	LoC	#Test Cases	#Faults
print_tokens	726	4130	7
grep	10,068	199	12
gzip	5,680	214	16
sed	14,427	360	9
space	6,199	13585	31
total			75

No unknown bugs are ever found!

More advanced tools

- Overall results

Project	past fix		lastest version		
	fix	method	file	method	LOC
aries	542	1,582	2,027	4,717	264,364
mahout	494	2,042	1,181	5,077	172,146
derby	1,604	4,966	2,764	18,568	1,207,970
cassandra	3,408	11,283	2,068	8,046	448,317
total	6,048	19,873	8,040	36,408	2,092,797



More advanced tools

Aries / ARIES-1701

A Possible NPE

Agile Board

Export

Details

Type:	Bug	Status:	RESOLVED
Priority:	Major	Resolution:	Fixed
Affects Version/s:	blueprint-core-1.7.0	Fix Version/s:	blueprint-cm-1.1.1
Component/s:	Blueprint		
Labels:	None		

People

Assignee: Guillaume Nodet

Reporter: Hao Zhong

Votes: 0 Vote for this issue

Watchers: 2 Start watching this issue

Dates

Created: 17/Mar/17 00:45

Updated: 20/Mar/17 14:33

Resolved: 20/Mar/17 14:33

Description

I found that the CmUtils.getProperties method has a buggy code as follow:

```
try {
ConfigurationAdmin ca = bc.getService(caRef);
Configuration config = getConfiguration(ca, persistentId);
if (config != null)
{ Dictionary<String, Object> props = new CaseInsensitiveDictionary(config.getProperties()); BundleContext caBc = caRef.getBundle().getBundleContext(); callPlugins(caBc, props, service, persistentId, null); return props; }

else
{ return null; }

} finally
{ bc.ungetService(caRef); }
```

Here, before calling ungetService, the call chain BundleContext caBc = caRef.getBundle().getBundleContext() can throw NPE.

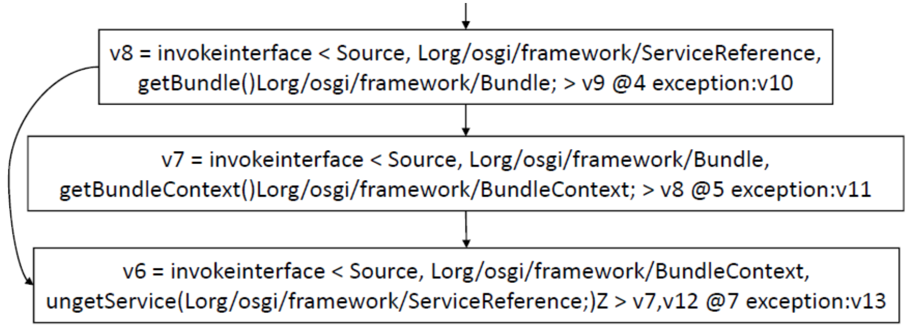
Indeed, a similar bug is fixed in ~~ARIES-788~~. Please check this bug at https://issues.apache.org/jira/browse/ARIES-788?jql=project%20%3D%20ARIES

Developn

1 commi

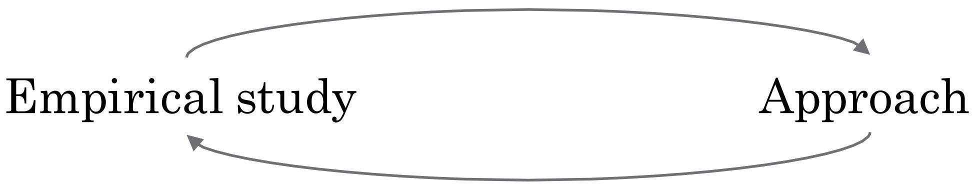
Agile

View on

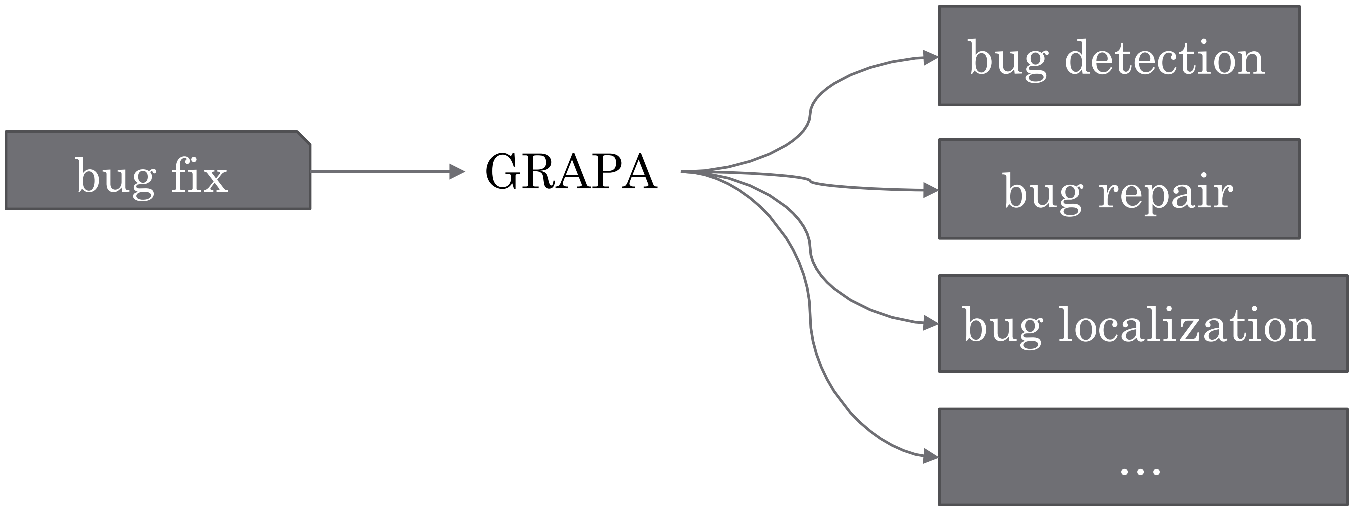


(a) The inferred bug signature

Conclusion



Empirical study
Approach



bug fix
GRAPA
bug detection
bug repair
bug localization
…

Questions