FINDING AND FIXING THE ARCHITECTURAL ROOTS OF BUGGINESS

RICK KAZMAN

KAZMAN@HAWAII.EDU

YOUR TYPICAL SOFTWARE PROJECT

YOUR TYPICAL SOFTWARE PROJECT

The boat is leaking but you keep paddling!

Why?

1. **The illusion of progress.**
2. **The lack of measurements.**

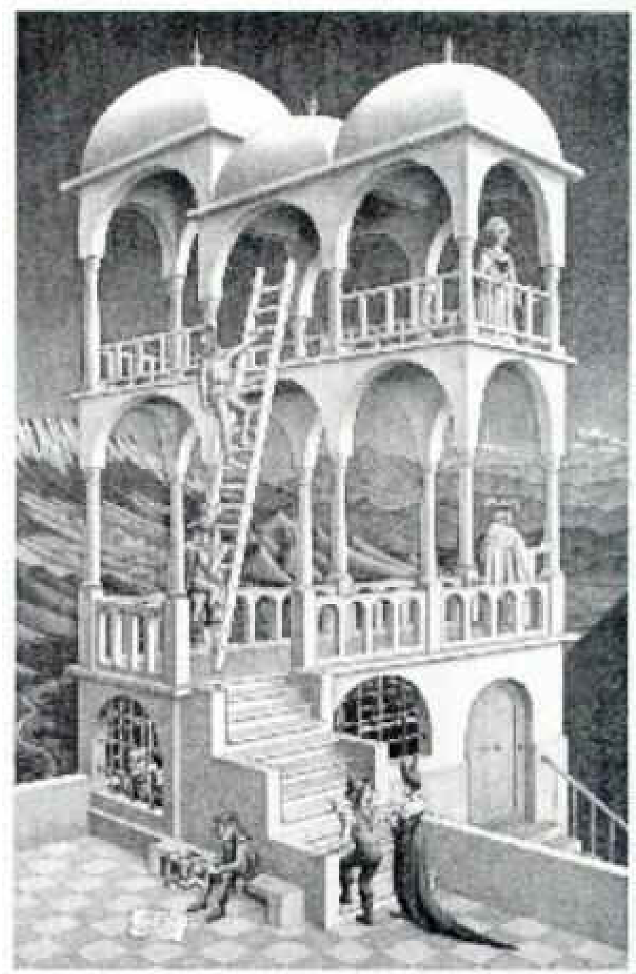
ARCHITECTURAL FLAWS

ARCHITECTURE MODELS

Do they help us understand an architecture?

Do they help us analyze an architecture?

Do they help us determine what is wrong with an architecture?



Mostly ….

No.

BRIDGING THE GAP BETWEEN ARCHITECTURE AND QUALITY

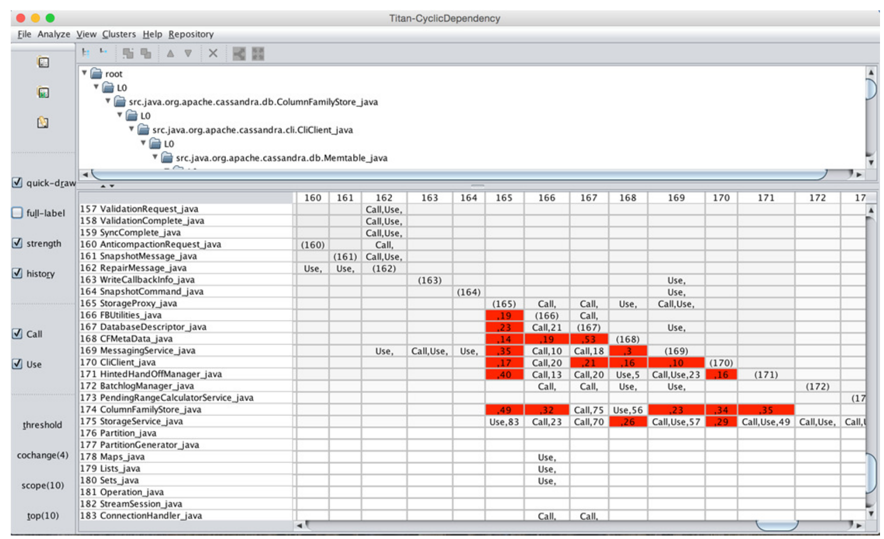
To bridge this gap we have created a design representation and associated tool:

Representation: Design Rule Space (DRSpace)

- Formed by selecting structural and/or evolutionary dependency relationships
- Clustered using the Design Rule Hierarchy algorithm

Tool: Titan

- Manipulates and visualizes DRSpaces
- Helps to *discover* architecture flaws



BRIDGING THE GAP BETWEEN ARCHITECTURE AND QUALITY

Using Titan we can find architecture design flaws:

- **cyclic class dependencies**
- **cyclic package dependencies**
- **improper inheritance**
- **modularity violations**
- **unstable interfaces**

Identifying such flaws allows us to:

- **assess technical debt and its economic impact**
- **predict the economic impact of repair strategies**

How does this work?

BACKGROUND CONCEPTS

Design Rule (DR)

- Design decisions that decouple the rest of the system into modules (typically a major interface or abstraction)

Design Structure Matrix (DSM)

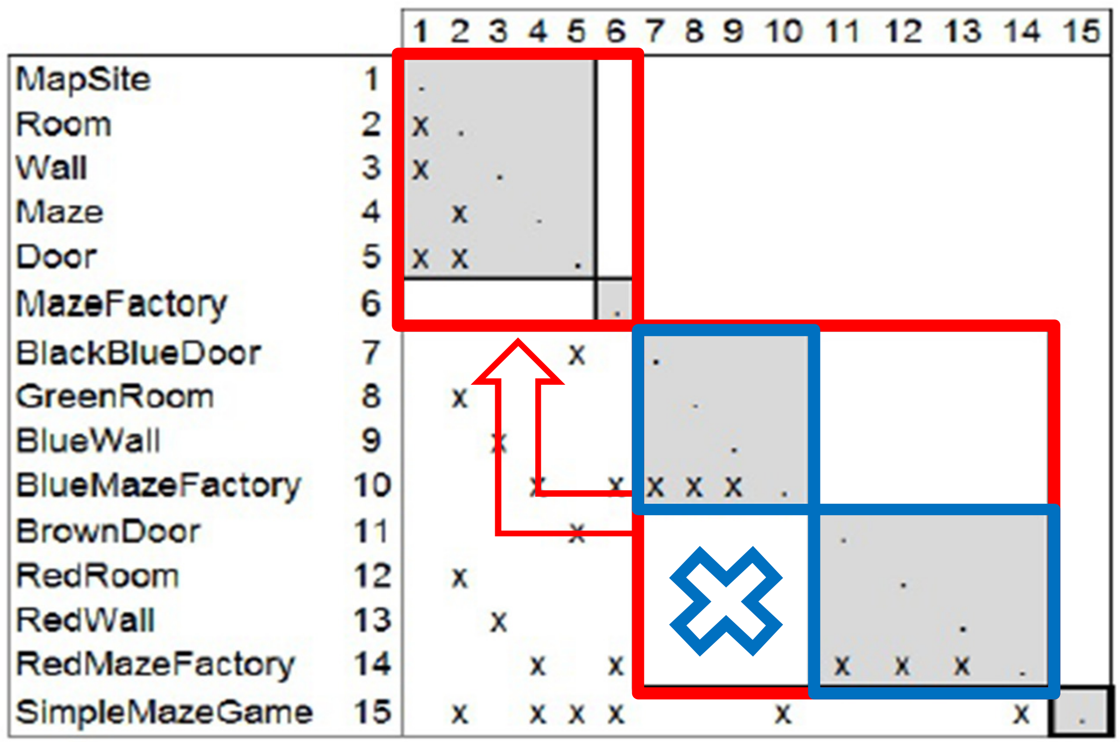
- Square matrix; rows and columns are classes (or files)
- Cell(i,j) shows a dependency relationship between the class in the ith row and the class in the jth column

Design Rule Hierarchy (DRH)

- A DSM clustering algorithm
- Lower layers only depend on higher layers
- Modules in the same layers are independent of each other

DSM IN DRH STRUCTURE

Factory pattern in a simple maze game



		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MapSite	1	.														
Room	2	x	.													
Wall	3	x		.												
Maze	4		x		.											
Door	5	x	x			.										
MazeFactory	6						.									
BlackBlueDoor	7					x		.								
GreenRoom	8		x						.							
BlueWall	9			x						.						
BlueMazeFactory	10				x		x	x	x	x	.					
BrownDoor	11					x						.				
RedRoom	12		x										.			
RedWall	13			x										.		
RedMazeFactory	14				x		x					x	x	x	.	
SimpleMazeGame	15		x		x	x	x				x				x	.

What is a DRSpace?

DESIGN RULE SPACE

- A set of classes
- A set of dependencies of different types revealing:
  - Structural relationships
  - Evolutionary relationships

1 mij.ast.TreeVisitor
2 mij.ast.Node
3 mij.ast.Number
4 mij.ast.OperExpr
5 mij.ast.FuncExpr
6 mij.ast.Variable
7 mij.ast.UnaryOperExpr
8 mij.Interpreter
9 mij.Interpreter$Calculator
10 mij.parse.Convert

	1	2	3	4	5	6	7	8	9	10
1	(1)									
2		(2)								
3	dp	ih	(3)							
4	dp	ih		(4)						
5	dp	ih			(5)					
6	dp	ih	8			(6)				
7	dp	ih		5			(7)			
8		dp						(8)	ag	
9	rl	dp	dp	dp	dp	dp	dp	9	(9)	
10		dp	dp	dp	dp	dp	dp			(10)

Realize/inherit

Aggregation

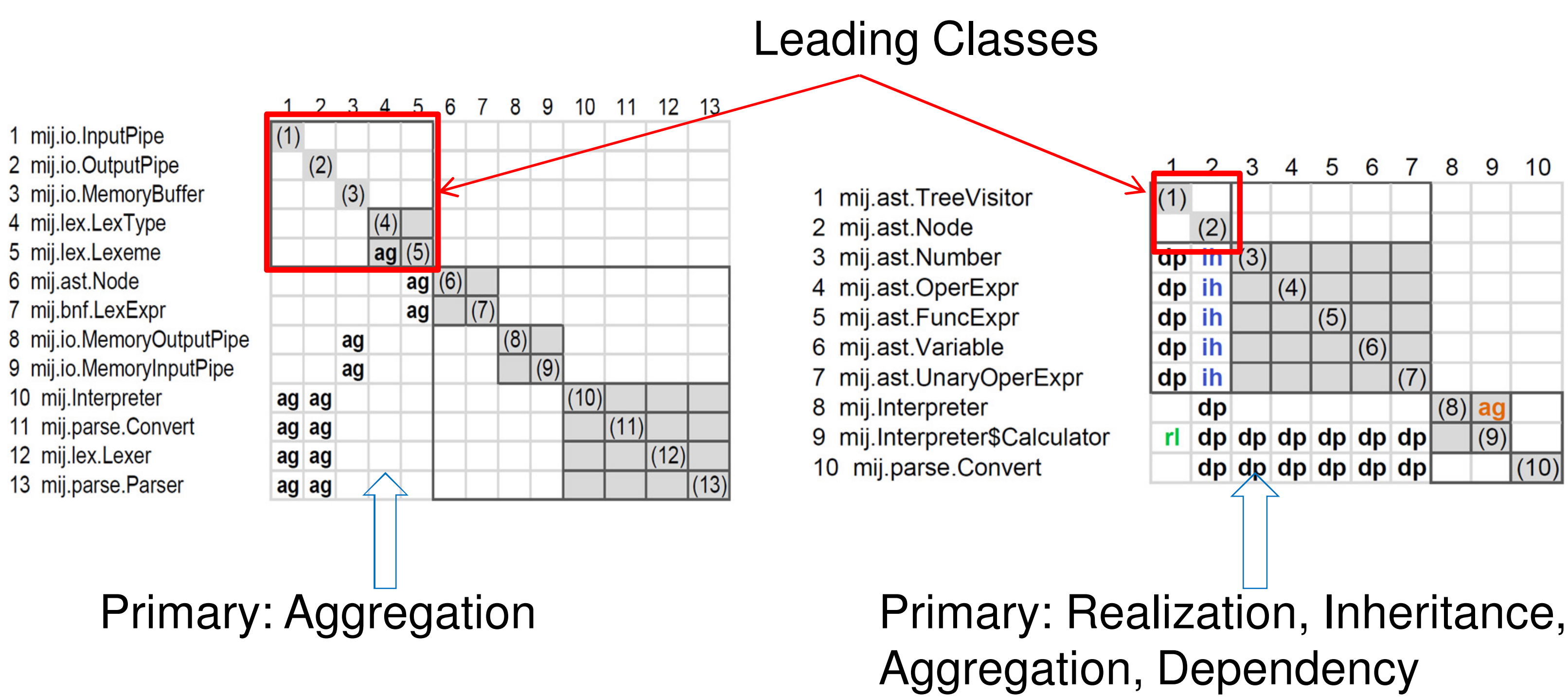
Dependency

Evolutionary

DESIGN RULE SPACE

A DRSpace is clustered using the DRH algorithm, based on selected *primary dependency* types.

The leading classes are the *design rules.*



ARCHITECTURE AS A SET OF DRSPACES

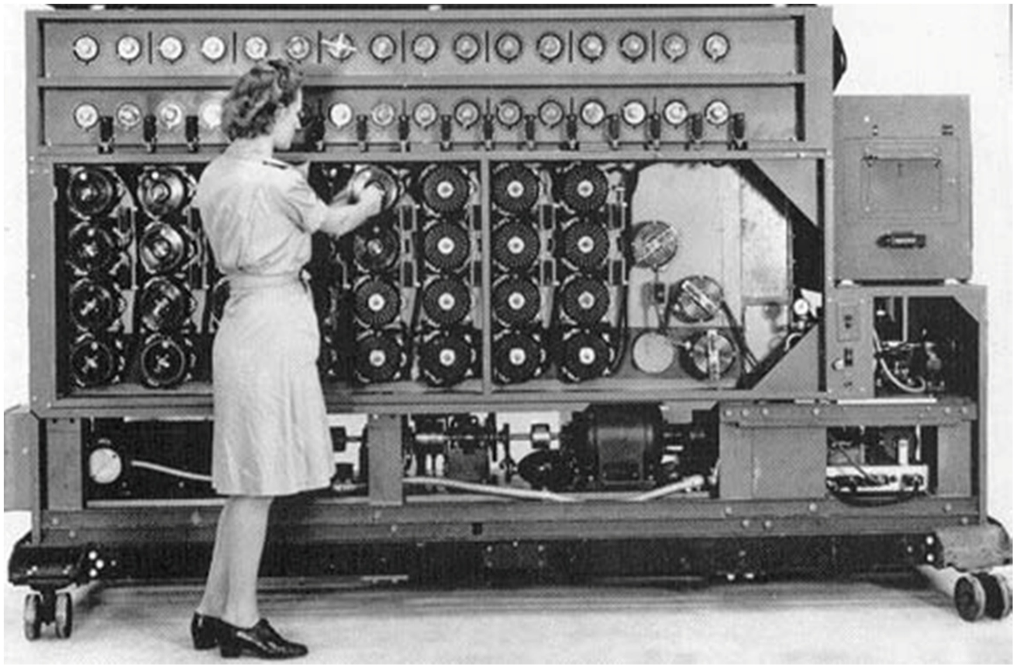
We can view and analyze architectures as a set of (overlapping) DRSpaces.

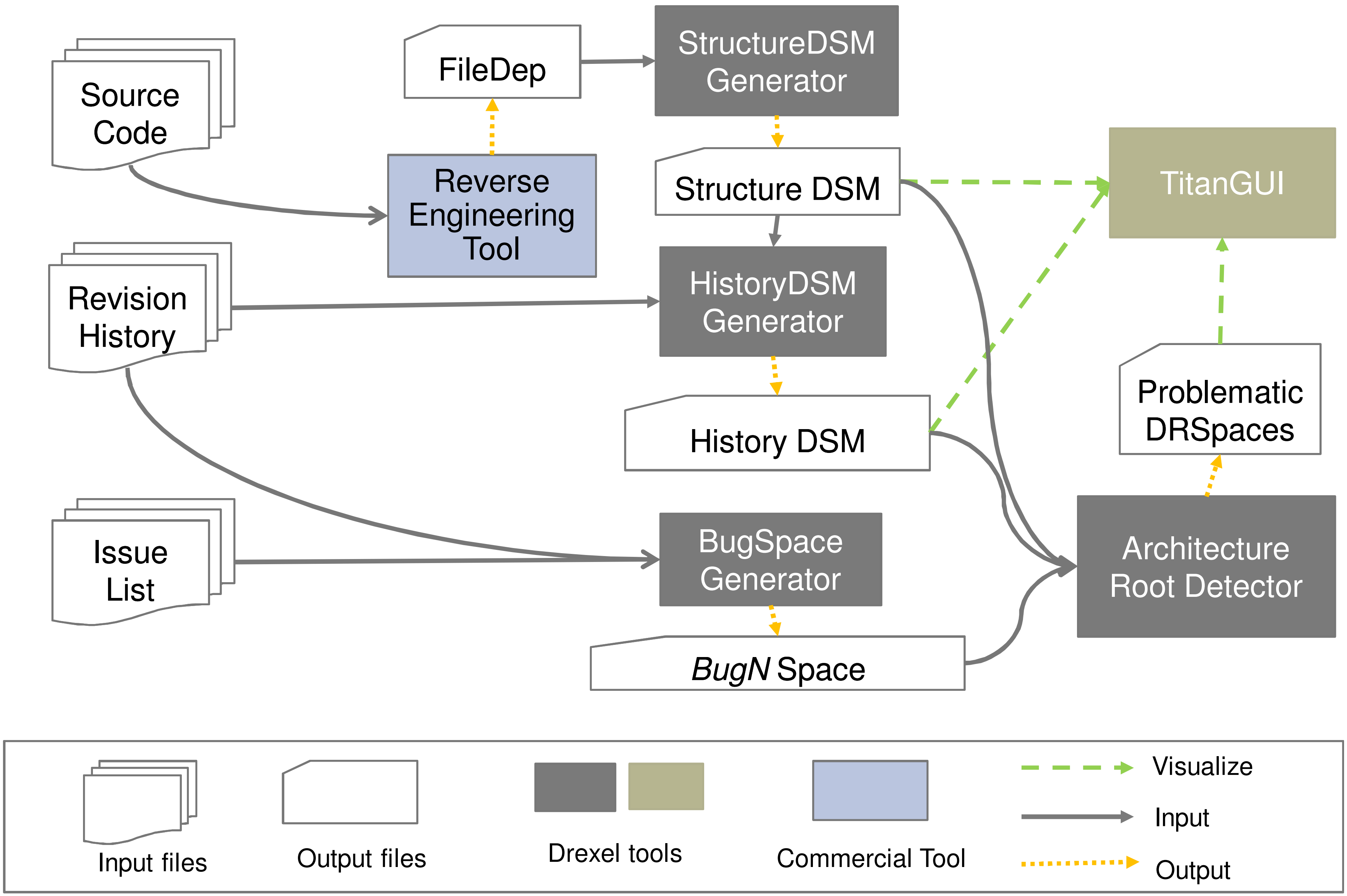
Each choice of primary relations reveals a different structure.

	1	2	3	4	5	6	7	8	9	10
1 mij.io.Pipe	(1)									
2 mij.bnf.Node		(2)								
3 mij.ast.Node			(3)							
4 mij.Filter				(4)						
5 mij.io.InputPipe	rl				(5)					
6 mij.io.OutputPipe	rl					(6)				
7 mij.io.MemoryOutputPipe						rl	(7)			
8 mij.io.WriterOutputPipe						rl		(8)		
9 mij.io.ReaderInputPipe					rl				(9)	
10 mij.io.MemoryInputPipe					rl					(10)

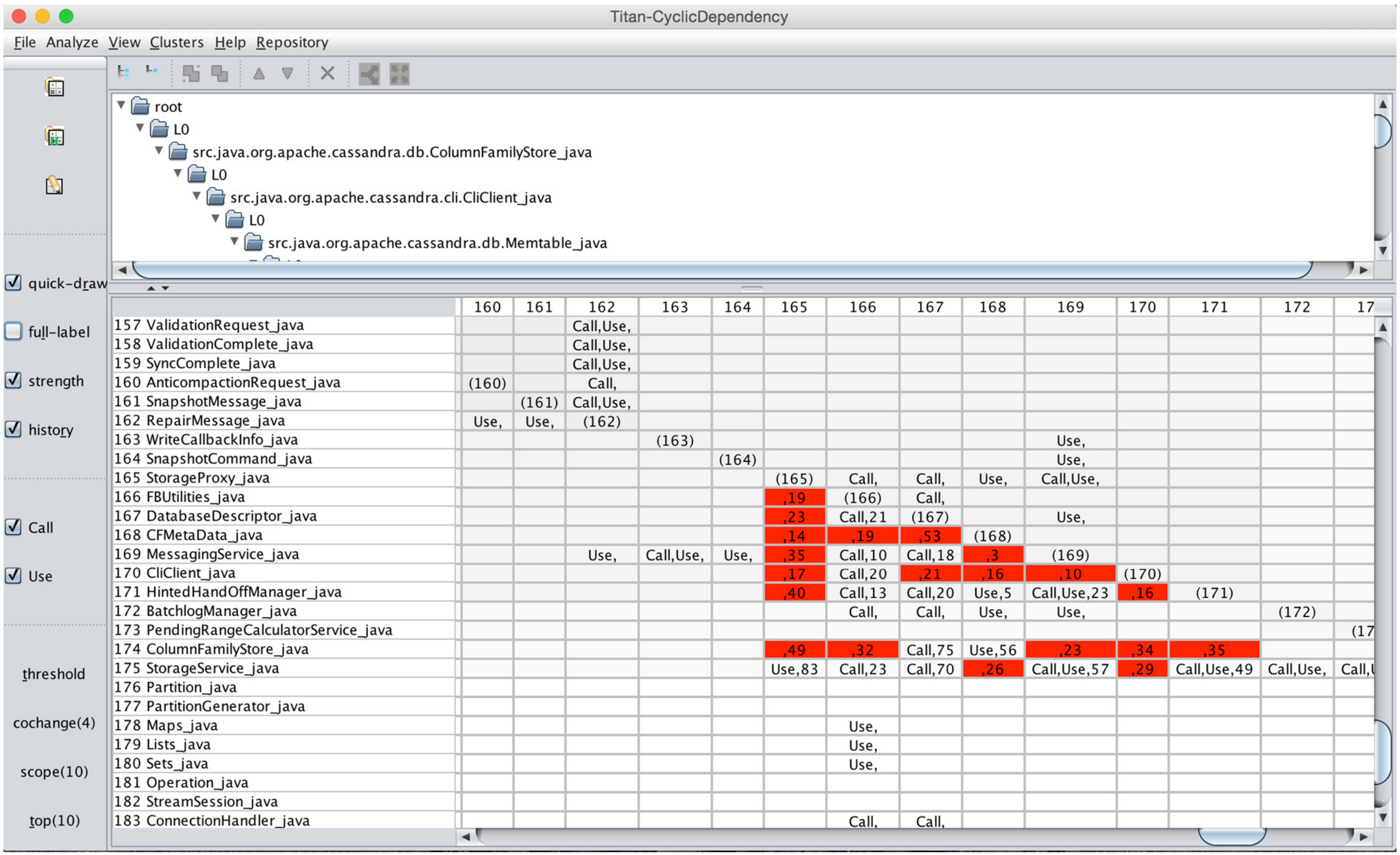
	1	2	3	4	5	6	7	8	9	10
1 mij.io.InputPipe	(1)									
2 mij.io.OutputPipe		(2)								
3 mij.io.MemoryBuffer			(3)							
4 mij.lex.LexType				(4)						
5 mij.lex.Lexeme				ag	(5)					
6 mij.ast.Node					ag	(6)				
7 mij.bnf.LexExpr					ag		(7)			
8 mij.io.MemoryOutputPipe			ag					(8)		
9 mij.io.MemoryInputPipe			ag						(9)	
10 mij.Interpreter	ag	ag								(10)

How do we compute DRSpaces?




Source Code
FileDep
StructureDSM Generator
Reverse Engineering Tool
Structure DSM
TitanGUI
Revision History
HistoryDSM Generator
History DSM
Problematic DRSpaces
Issue List
BugSpace Generator
Architecture Root Detector
BugN Space
Input files
Output files
Drexel tools
Commercial Tool
Visualize
Input
Output

TITAN GUI


Titan-CyclicDependency
File Analyze View Clusters Help Repository
root
L0
src.java.org.apache.cassandra.db.ColumnFamilyStore_java
L0
src.java.org.apache.cassandra.cli.CliClient_java
L0
src.java.org.apache.cassandra.db.Memtable_java
quick-draw
full-label
strength
history
Call
Use
threshold
cochange(4)
scope(10)
top(10)
160 161 162 163 164 165 166 167 168 169 170 171 172 17
157 ValidationRequest_java Call,Use,
158 ValidationComplete_java Call,Use,
159 SyncComplete_java Call,Use,
160 AnticompactionRequest_java (160) Call,
161 SnapshotMessage_java (161) Call,Use,
162 RepairMessage_java Use, Use, (162)
163 WriteCallbackInfo_java (163) Use,
164 SnapshotCommand_java (164) Use,
165 StorageProxy_java (165) Call, Call, Use, Call,Use,
166 FBUtilities_java ,19 (166) Call,
167 DatabaseDescriptor_java ,23 Call,21 (167) Use,
168 CFMetaData_java ,14 ,19 ,53 (168)
169 MessagingService_java Use, Call,Use, Use, ,35 Call,10 Call,18 ,3 (169)
170 CliClient_java ,17 Call,20 ,21 ,16 ,10 (170)
171 HintedHandOffManager_java ,40 Call,13 Call,20 Use,5 Call,Use,23 ,16 (171)
172 BatchlogManager_java Call, Call, Use, Use, (172)
173 PendingRangeCalculatorService_java (17
174 ColumnFamilyStore_java ,49 ,32 Call,75 Use,56 ,23 ,34 ,35
175 StorageService_java Use,83 Call,23 Call,70 ,26 Call,Use,57 ,29 Call,Use,49 Call,Use, Call,U
176 Partition_java
177 PartitionGenerator_java
178 Maps_java Use,
179 Lists_java Use,
180 Sets_java Use,
181 Operation_java
182 StreamSession_java
183 ConnectionHandler_java Call, Call,

What do different DRSpaces tell us?

REALIZE & INHERIT RELATIONS

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1 mij.io.Pipe	(1)																									
2 mij.bnf.Node		(2)																								
3 mij.ast.Node			(3)																							
4 mij.Filter				(4)																						
5 mij.io.InputPipe	rl				(5)																					
6 mij.io.OutputPipe	rl					(6)																				
7 mij.io.MemoryOutputPipe						rl	(7)																			
8 mij.io.WriterOutputPipe						rl		(8)																		
9 mij.io.ReaderInputPipe					rl				(9)																	
10 mij.io.MemoryInputPipe					rl					(10																
11 mij.bnf.ExponExpr		ih									(11															
12 mij.bnf.UnaryExpr		ih										(12														
13 mij.bnf.LexExpr		ih											(13													
14 mij.bnf.MultExpr		ih												(14												
15 mij.bnf.AddExpr		ih													(15											
16 mij.bnf.ValueExpr		ih														(16										
17 mij.bnf.ParamExpr		ih															(17									
18 mij.Interpreter				rl														(18								
19 mij.parse.Convert				rl															(19							
20 mij.parse.Parser				rl																(20						
21 mij.lex.Lexer				rl																	(21					
22 mij.ast.OperExpr			ih																			(22				
23 mij.ast.Number			ih																				(23			
24 mij.ast.FuncExpr			ih																					(24		
25 mij.ast.Variable			ih																						(25	
26 mij.ast.UnaryOperExpr			ih																							(26)

OutputPipe Module

InputPipe Module

Bnf Module

Filter Module

Ast Module

Four leading classes in first layer

InputPipe and OutputPipe in the second layer

These decouple 5 independent modules in the third layer

AGGREGATION RELATION

		1	2	3	4	5	6	7	8	9	10	11	12	13
1	mij.io.InputPipe	(1)												
2	mij.io.OutputPipe		(2)											
3	mij.io.MemoryBuffer			(3)										
4	mij.lex.LexType				(4)									
5	mij.lex.Lexeme				ag	(5)								
6	mij.ast.Node					ag	(6)							
7	mij.bnf.LexExpr					ag		(7)						
8	mij.io.MemoryOutputPipe			ag					(8)					
9	mij.io.MemoryInputPipe			ag						(9)				
10	mij.Interpreter	ag	ag								(10)			
11	mij.parse.Convert	ag	ag									(11)		
12	mij.lex.Lexer	ag	ag										(12)	
13	mij.parse.Parser	ag	ag											(13)

Lex Module

Memory Buffer Module

Interpreter Module

Five leading classes in first layer

Three coherent modules in second layer

DEPENDENCY RELATION

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1 mij.FilterException	(1)																															
2 mij.lex.Lexeme		(2)																														
3 mij.ast.TreeVisitor			(3)																													
4 mij.ast.Node				(4)																												
5 mij.Interpreter	dp			dp	(5)																											
6 mij.ast.OperExpr			dp			(6)																										
7 mij.ast.Number			dp				(7)																									
8 mij.ast.FuncExpr			dp					(8)																								
9 mij.ast.Variable			dp						(9)																							
10 mij.ast.UnaryOperExpr			dp							(10																						
11 mij.bnf.LexExpr											(11																					
12 mij.bnf.ExponExpr												(12																				
13 mij.bnf.ValueExpr													(13																			
14 mij.bnf.UnaryExpr														(14																		
15 mij.bnf.MultExpr															(15																	
16 mij.bnf.AddExpr																(16																
17 mij.bnf.ParamExpr																	(17															
18 mij.bnf.Node																		(18														
19 mij.bnf.GrammarType																			(19													
20 mij.lex.LexType																				(20												
21 mij.parse.Parser	dp	dp									dp	dp	dp	dp	dp	dp	dp	dp	dp	dp	(21											
22 mij.parse.Parser$Entry																						(22										
23 mij.Interpreter$Calculator		dp		dp		dp	dp	dp	dp	dp													(23									
24 mij.io.MemoryOutputPipe																								(24								
25 mij.io.InputPipe																									(25							
26 mij.io.MemoryInputPipe																										(26						
27 mij.lex.Lexer	dp	dp																									(27					
28 mij.io.MemoryBuffer																												(28				
29 mij.parse.Convert		dp		dp		dp	dp	dp	dp	dp	dp							dp											(29			
30 mij.io.ReaderInputPipe																														(30		
31 mij.Filter																															(3	
32 mij.Console	dp				dp																dp			dp	dp	dp	dp	dp	dp	dp	dp	(32)

Parsing Function

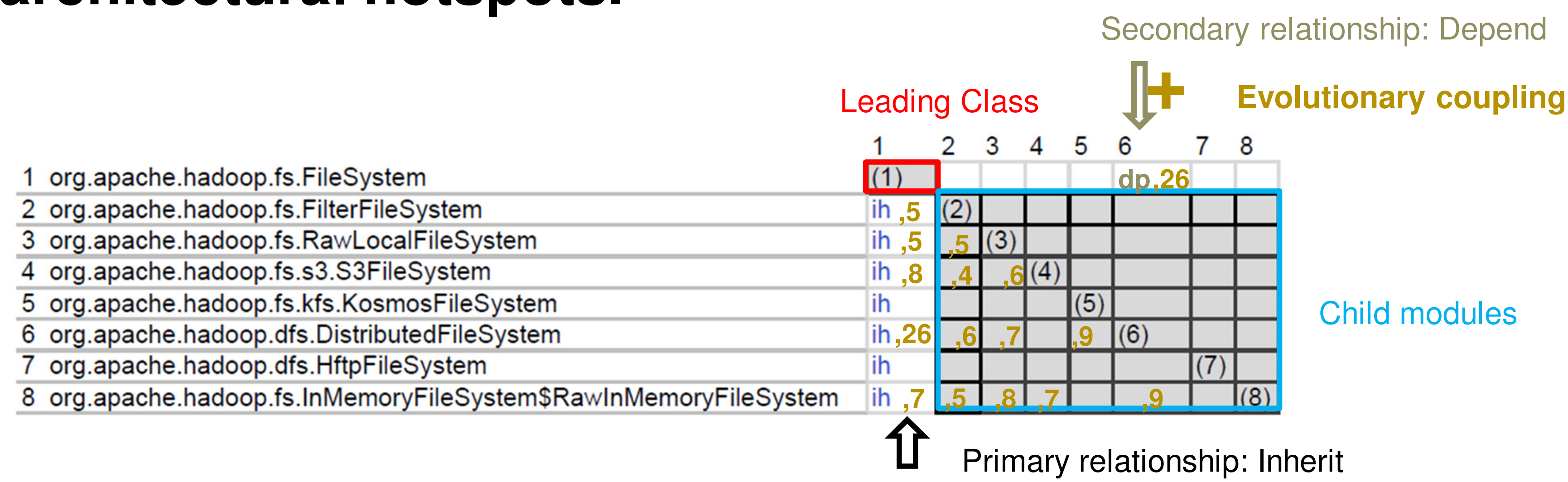
Console Function

How classes work together to accomplish a function

How do we use DRSpaces to reveal architecture hotspots?

REVEALING PROBLEMS

We add in *secondary relationships* to reveal architectural hotspots.



Issue 1: Parent class depends on child class
Issue 2: Unusual evolutionary coupling between parent class and child class
Issue 3: Modularity violation

Can we find meaningful relationships between architecture and quality?

INITIAL RESEARCH QUESTIONS

RQ1: If the leading class of a DRSpace is error-prone, are the files in the DRSpace also error-prone?

RQ2: Are the majority of the error-prone files concentrated in a few DRSpaces?

RQ3: By combining evolution and structure coupling, can we get more insight into architectural problems? Can this help us find not just the locations of errors, but also the reasons for them?

INITIAL EVALUATION DATA SET

Subject	History	#Files	#Releases	#Commits	#Issues
JBoss 3.2.4	Apr 00 – Jun 04	762	11	6458	550
Hadoop 0.15	Feb 06 – Oct 07	692	15	3001	490
Eclipse 3.0.2	May 01 – Mar 05	3,704	10	27,806	3,458

Conventions

- Bug Space: ***BugN***
- Design Space bugginess: ***dsb***
- Bug Space coverage: ***bsc***

RQ1

RQ1: If the leading class of a DRSpace is error-prone, are the files in the DRSpace also error-prone?

RQ1: ANSWER

We selected DRSpaces led by the top 30 most error-prone files as design rules.

We then sub-selected DRSpaces with at least 10 files.

We computed dsb and bsc for each project with respect to Bug2, Bug5 and Bug10.

Consider JBoss: 9 out of 30 DRSpaces have at least 10 files.

62% of the files in each selected DRSpace have more than 2 bug fixes. The average JBoss DRSpace contains 10% of the project's Bug2 files.

#TopDRS	Avg. in Bug2		Avg. in Bug5		Avg. in Bug10	
	dsb	bsc(#)	dsb	bsc(#)	dsb	bsc(#)
JBoss(9)	62%	10%(206)	50%	13%(129)	18%	30%(23)
Hadoop(11)	67%	11%(187)	53%	14%(106)	47%	18%(68)
Eclipse(11)	31%	7%(291)	15%	9%(98)	9%	13%(36)

RQ1: ANSWER

JBoss:

#DRS	#Bug (Rank)	#Bug2: 206		#Bug5: 129		#Bug10: 23	
		$\#(dsb)$	bsc	$\#(dsb)$	bsc	$\#(dsb)$	bsc
dr1:18	27(2nd)	10(56%)	5%	7(39%)	5%	4(22%)	17%
dr2:56	21(4th)	32(57%)	16%	26(46%)	20%	10(18%)	43%
dr3:28	18(7th)	22(79%)	11%	18(64%)	14%	5(18%)	22%
dr4:43	14(12th)	27(63%)	13%	20(47%)	16%	5(12%)	22%
dr5:76	11(21st)	43(57%)	21%	35(46%)	27%	12(16%)	52%

- JBoss's Bug2 space has 206 files
- Consider **org.jboss.ejb.Container**:
  - 21 bug fixes; 4^{th} most error-prone file
  - Its DRSpace contains 56 files
- 32 files out of the 56 files have more than 2 bug fixes
  - dsb = 32/56 = 57%
  - bsc = 32/206 = 16%

RQ2

RQ2: Are the majority of the error-prone files concentrated in a few DRSpaces?

RQ2: ANSWER

1) What portion of the bug space can the top 5 and top 10 largest DRSpaces cover?

Proj	Bug2		Bug5		Bug10	
	Top 5	Top 10	Top 5	Top 10	Top 5	Top 10
JBoss	57%	64%	52%	57%	78%	78%
Hadoop	59%	67%	68%	75%	76%	85%
Eclipse	71%	78%	83%	89%	89%	92%

✓The top 5 and top 10 largest DRSpaces can cover the majority of the files in each bug space.

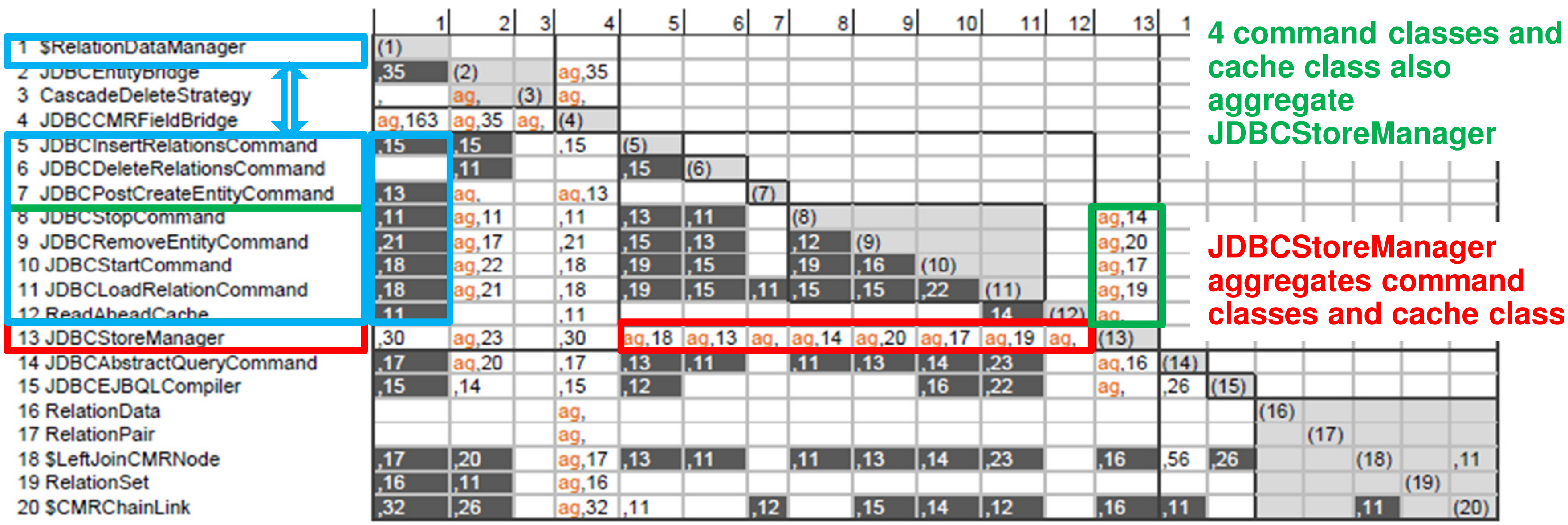
RQ3

RQ3: By combining evolution and structure coupling, can we get more insight into architectural problems? Can this help us find not just the locations of errors, but also the reasons for them?

RQ3: ANSWER

Aggregation cycles

Shared Secrets



	1	2	3	4	5	6	7	8	9	10	11	12	13	1						
1 $RelationDataManager	(1)																			
2 JDBCEntityBridge	,35	(2)		ag,35																
3 CascadeDeleteStrategy	,	ag,	(3)	ag,																
4 JDBCCMRFieldBridge	ag,163	ag,35	ag,	(4)																
5 JDBCInsertRelationsCommand	,15	,15		,15	(5)															
6 JDBCDeleteRelationsCommand		,11			,15	(6)														
7 JDBCPostCreateEntityCommand	,13	ag,		ag,13			(7)													
8 JDBCStopCommand	,11	ag,11		,11	,13	,11		(8)					ag,14							
9 JDBCRemoveEntityCommand	,21	ag,17		,21	,15	,13		,12	(9)				ag,20							
10 JDBCStartCommand	,18	ag,22		,18	,19	,15		,19	,16	(10)			ag,17							
11 JDBCLoadRelationCommand	,18	ag,21		,18	,19	,15	,11	,15	,15	,22	(11)		ag,19							
12 ReadAheadCache	,11			,11							,14	(12)	ag,							
13 JDBCStoreManager	,30	ag,23		,30	ag,18	ag,13	ag,	ag,14	ag,20	ag,17	ag,19	ag,	(13)							
14 JDBCAbstractQueryCommand	,17	ag,20		,17	,13	,11		,11	,13	,14	,23		ag,16	(14)						
15 JDBCEJBQLCompiler	,15	,14		,15	,12					,16	,22		ag,	,26	(15)					
16 RelationData				ag,												(16)				
17 RelationPair				ag,													(17)			
18 $LeftJoinCMRNode	,17	,20		ag,17	,13	,11		,11	,13	,14	,23		,16	,56	,26			(18)		,11
19 RelationSet	,16	,11		ag,16															(19)	
20 $CMRChainLink	,32	,26		ag,32	,11		,12		,15	,14	,12		,16	,11				,11		(20)

Jboss JDBCCMRFieldBridge DRSpace

RQ3: ANSWER

Problematic inheritance

Unusual evolutionary coupling

		1	2	3	4	5	6	7	8
1	org.apache.hadoop.fs.FileSystem	(1)					dp,26		
2	org.apache.hadoop.fs.FilterFileSystem	ih,5	(2)						
3	org.apache.hadoop.fs.RawLocalFileSystem	ih,5	,5	(3)					
4	org.apache.hadoop.fs.s3.S3FileSystem	ih,8	,4	,6	(4)				
5	org.apache.hadoop.fs.kfs.KosmosFileSystem	ih,				(5)			
6	org.apache.hadoop.dfs.DistributedFileSystem	ih,26	,6	,7	,9		(6)		
7	org.apache.hadoop.dfs.HftpFileSystem	ih,						(7)	
8	org.apache.hadoop.fs.InMemoryFileSystem$RawInMemoryFileSystem	ih,7	,5	,8	,7		,9		(8)

Hadoop FileSystem Inherit DRSpace

ANSWERS TO INITIAL RQS

RQ1: A significant portion of the DRSpaces led by an error prone class are also error-prone. ✓

RQ2: The 5 largest DRSpaces always captured more than half of the files in the bug space. ✓

RQ3: Error-prone DRSpaces also have structural problems and modularity violations. ✓

FOLLOW-ON RQ

RQ4: If a file is involved in greater numbers of architecture issues, it is more error-prone/change-prone than average files?

SECOND DATA SET

Subject	#Month	#Snap	#Comm	#Issue	#File
Avro 1.7.6	47	22	1480	630	145-298
Camel 2.11.1	53	46	17706	2326	528-1203
Cassandra 1.0.7	24	46	6738	3645	419-786
CXF 2.7.10	70	92	27247	3400	1426-3073
Hbase 0.94.16	70	21	14858	5032	347-2142
Ivy 2.3.0	52	11	3799	839	418-607
OpenJPA 2.2.2	68	17	6736	1574	1216-1761
PDFBox 1.8.4	46	13	1798	1279	458-589
Wicket 1.5.5	57	55	18004	3359	1099-1549
Commercial	9	13	6000	800	137-599

RQ4 ANSWER

We counted the architecture issues in the 10 projects and compared these to:

- Bug frequency
- Bug churn
- Change frequency
- Change churn

RQ4 ANSWER

Avro-1.7.6				
#AI	BF_avg	BC_avg	CF_avg	CC_avg
0	0.1	3.7	0.5	29.0
1	0.4	3.9	0.9	26.2
2	1.6	12.6	5.2	376.7
3	7.9	124.5	21.6	628.5
4	16.5	255.0	33.5	1220.0
PC	0.91	0.89	0.94	0.95

Camel-2.11.1				
#AI	BF_avg	BC_avg	CF_avg	CC_avg
0	0.5	7.9	2.2	58.2
1	1.2	18.5	5.6	131.5
2	3.7	56.6	14.4	304.7
3	8.4	141.5	33.9	681.3
4	13.9	204.7	50.9	1043.5
PC	0.96	0.96	0.97	0.97

Cassandra-1.0.7				
#AI	BF_avg	BC_avg	CF_avg	CC_avg
0	0.4	7.1	1.0	32.6
1	1.1	17.4	4.8	106.4
2	5.3	84.5	21.2	559.1
3	12.8	245.8	45.7	1202.0
4	18.8	364.9	65.7	1909.4
PC	0.97	0.96	0.98	0.97

CXF-2.7.10				
#AI	BF_avg	BC_avg	CF_avg	CC_avg
0	0.8	21.0	2.8	86.9
1	2.9	62.3	9.4	262.5
2	8.6	164.8	23.1	592.0
3	20.2	390.9	52.5	1232.4
4	54.1	890.2	142.3	3326.0
PC	0.90	0.92	0.89	0.89

Hadoop-2.2.0				
#AI	BF_avg	BC_avg	CF_avg	CC_avg
0	0.4	12.7	1.0	56.8
1	1.5	24.8	4.2	167.7
2	5.3	173.6	13.8	558.3
3	26.0	725.1	58.0	1959.6
4	13.7	237.9	26.8	1252.0
PC	0.76	0.63	0.72	0.83

HBase-0.94.16				
#AI	BF_avg	BC_avg	CF_avg	CC_avg
0	0.7	10.4	0.9	53.0
1	4.8	236.7	8.3	614.6
2	9.9	418.5	17.2	2083.6
3	47.8	1335.1	87.6	3158.7
4	76.7	2370.4	135.1	6019.0
PC	0.93	0.94	0.93	0.97

Ivy-2.3.0				
#AI	BF_avg	BC_avg	CF_avg	CC_avg
0	0.2	4.5	1.1	31.8
1	1.1	22.8	3.3	79.6
2	2.9	54.6	8.4	251.9
3	7.0	119.9	20.9	646.2
4	6.4	204.6	18.6	792.3
PC	0.94	0.96	0.93	0.97

OpenJPA-2.2.2				
#AI	BF_avg	BC_avg	CF_avg	CC_avg
0	1.8	10.0	1.1	36.8
1	3.2	31.1	3.7	111.5
2	4.6	64.5	7.5	229.8
3	10.8	408.6	22.4	862.5
4	25.1	981.0	52.5	2301.1
PC	0.90	0.88	0.90	0.88

Pdfbox-1.8.4				
#AI	BF_avg	BC_avg	CF_avg	CC_avg
0	0.5	27.1	1.1	92.0
1	1.4	35.9	2.9	136.5
2	1.5	64.1	3.4	259.9
3	8.1	495.0	13.7	861.3
4	12.2	669.5	18.4	1254.4
PC	0.92	0.92	0.94	0.94

Commercial Project				
#AI	BF_avg	BC_avg	CF_avg	CC_avg
0	0.1	2.25	2.7	102.5
1	0.2	4.6	5.9	200.4
2	0.8	3.24	10.3	372.0
3	2.8	36.8	19.8	884.7
4	6.0	21	29.0	549.0
PC	0.91	0.73	0.98	0.81

AVRO-1.7.6

#AI	BF_avg	BC_avg	CF_avg	CC_avg
0	0.1	3.7	0.5	29.0
1	0.4	3.9	0.9	26.2
2	1.6	12.6	5.2	376.7
3	7.9	124.5	21.6	628.5
4	16.5	255.0	33.5	1220.0
PC	**0.91**	**0.89**	**0.94**	**0.95**

FOLLOW-ON RQ

RQ4: If a file is involved in greater numbers of architecture issues, it is more error-prone/change-prone than average files?

ECONOMIC ANALYSIS

Based on the identified DRSpaces and a knowledge of their architecture issues, we can determine refactoring strategies.

And we can make decisions about whether to refactor based on ROI.

Consider the following analysis of a commercial project:

ECONOMIC ANALYSIS

	A	B	C	D	E	F	G	I	J	K	L	M	N
1	***DRSpace Leading File***	**DRSpace Size**	**Norm Size**	**Current Defects/Yr**	**Norm Defects**	**Current Changes/Yr**	**Norm Changes/Yr**	**Tot LOC Changed**	**Norm LOC Changed**	**Refactor Cost (PM)**	**Norm Exp Defects/Yr**	**Norm Exp Changes/Yr**	**Norm Exp LOC Changed**
2	*Pear.java*	139	103.1	166	123.1	1068	839.2	49,171	36,471	5.5	34	299	17,522
3	*Apple.java*	158	103.5	63	41.3	607	451.7	25,603	16,772	7	34	300	17,590
4	*Bean.java*	65	30.4	72	33.7	429	207.2	17,807	8,328	1.5	10	88	5,167
5													
6	**DRSpace Total**		**237**		**198.1**		**1498**		**61,571**		**78.2**	**687.3**	**40,279**
7	**Project Total**	**797**		**265**		**2332**		**135,453**		**14**			
8	**Savings**										**120**	**811**	**21,292**
9													
10													
11	Base defect rates	0.33											
12	Base change rates	2.9										**Exp PM saved**	**35.49**
13	Base LOC/file	169.95											
14	SLOC/PM	600											
15													

Result: >250% ROI in the first year alone!

TAKE-AWAYS

Architectural problems lead to quality issues.

We can locate these problems!

We can not fix the quality problems without fixing the underlying architecture problems!

To do this we view and analyze architectures as a set of overlapping DRSpaces.

TAKE-AWAYS

You can not find these problems by code-based analysis alone.

This analysis allows us to plan refactoring strategies and make informed, *economics-based* decisions about if and how to refactor.

This work was supported in part by the U.S. National Science Foundation under grants CCF-0916891, CCF- 1065189, CCF-1116980 and DUE-0837665.

FURTHER READING

R. Kazman, Y. Cai, R. Mo, Q. Feng, L. Xiao, S. Haziyev, V. Fedak, A. Shapochka, “A Case Study in Locating the Architectural Roots of Technical Debt”, *Proceedings of the International Conference on Software Engineering (ICSE) 2015*, (Florence, Italy), May 2015.

R. Mo, Y. Cai, R. Kazman, L. Xiao, “Hotspot Patterns: The Formal Definition and Automatic Detection of Architecture Smells”, *Proceedings of The Working IEEE/IFIP Conference on Software Architecture (WICSA 2015*), (Montreal, Canada), May 2015.

L. Xiao, Y. Cai, R. Kazman, "Design Rule Spaces: A New Form of Architecture Insight", *Proceedings of the International Conference on Software Engineering (ICSE) 2014*, (Hyderabad, India), June 2014.