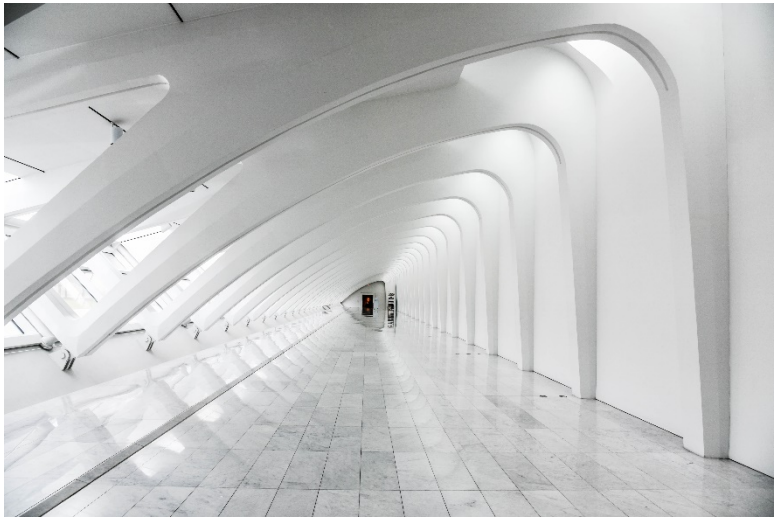


Industrial Experiences with

Automated Software Architecture Measurement

By Yuanfang Cai and Rick Kazman

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Software Health Management System

From research to practice

Metrics for comparison and contrast

Monitor the degradation, variation, and evolution of software projects

Hotspot detection

Pinpoint design flaws with ever-increasing maintenance costs

Quantification

Calculate the cost of each flaw and estimate the benefits of refactoring

Software Health Management System

Systems we have analyzed

- 252 Open Source Projects
- 50+ Industrial projects

softserve

SAMSUNG

BRIGHTSQUID

General Work Flow

Overview

Step 1: Data Collection

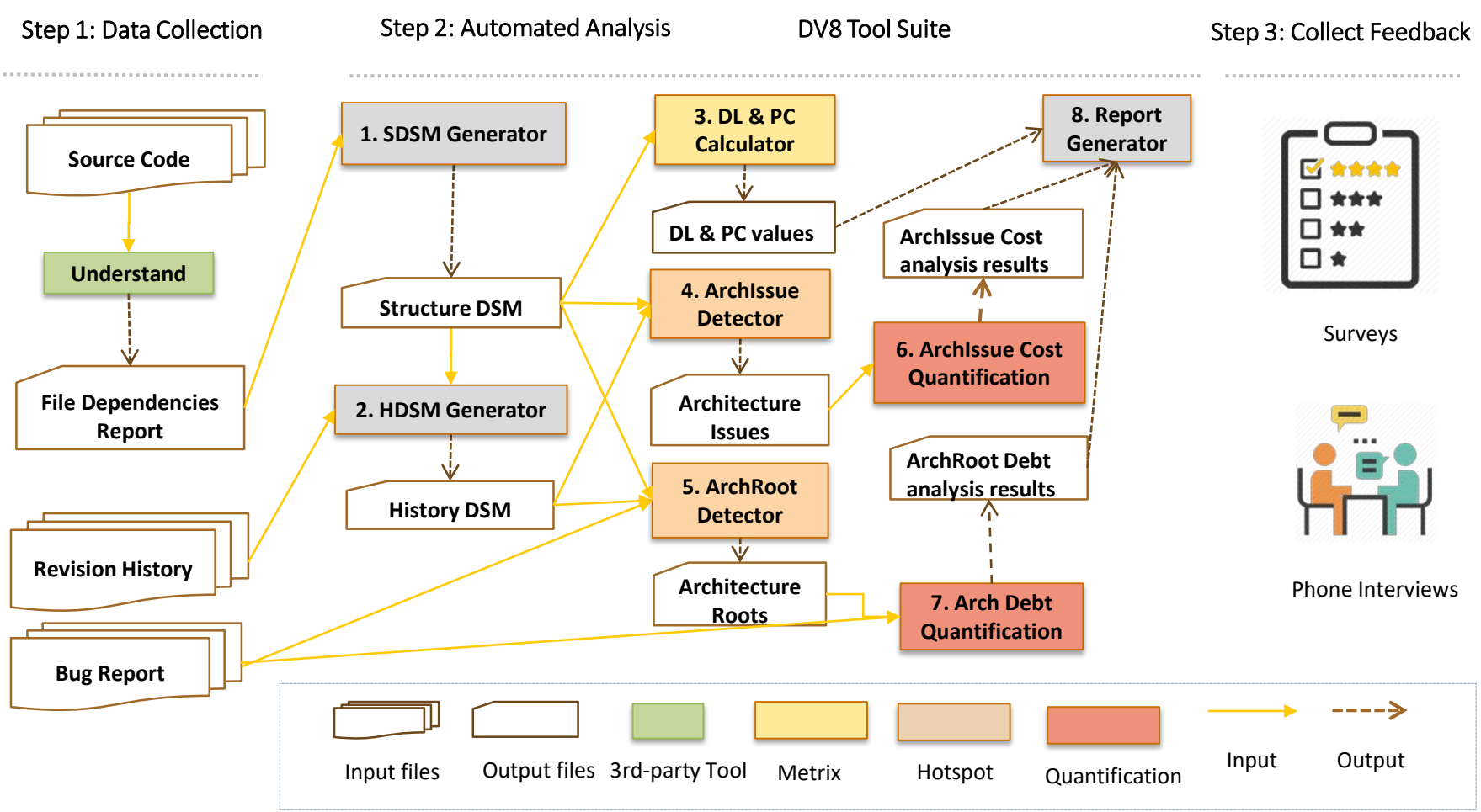
Code dependency, history, issue record

Step 2: Automated Analysis

Measurement, hotspot detection, cost calculation

Step: Collect Feedback

Surveys and Interviews



Step 1: Data Accepted by DV8 tool suite

Beyond source code

Dependency information

- Currently support Understand Cytoscape output
- Extensible to other dependency formats through standard interfaces

History information (optional)

- Commit history such as git record, svn record
- Extensible to other history record types through standard interfaces

Issue information (optional)

- Issue tracking info such as Jira record, MTF work item, Bugzilla record etc.
- Extensible to other records through standard interfaces

Design Rule Space (DRSpace) [Xiao et al. 2014]

A new architecture model supported by DV8

A DRSpace is composed of a **meaningful subset** of a system's files along with the **architectural connections** among these files.

- Any subset of files may form a design space
- Architectural connections
  - Structural couplings: call, inherit, aggregate, etc..
  - Evolutionary couplings
  - One or multiple types
  - Implicit or explicit

Design Rule Space (DRSpace)

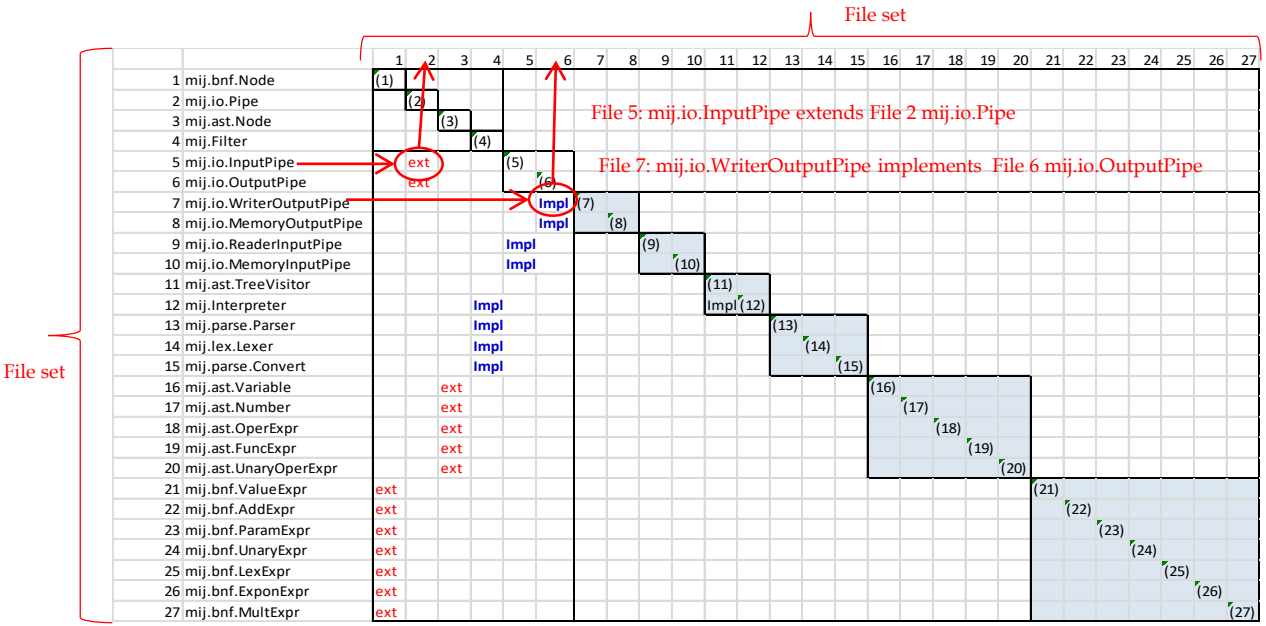
A new architecture model supported by DV8

- Non-trivial software system must contain multiple design spaces:
  - each feature implemented
  - each pattern applied
  - each concern addressed
- Each file can participate in multiple design spaces
- We propose that software architecture can, and should be modeled as **multiple, overlapping design spaces**
- Each design space can be visualized as a ***Design Structure Matrix*** (DSM)

Design Rule Space (DRSpace) in Design Structure Matrix (DSM)

A small calculator example

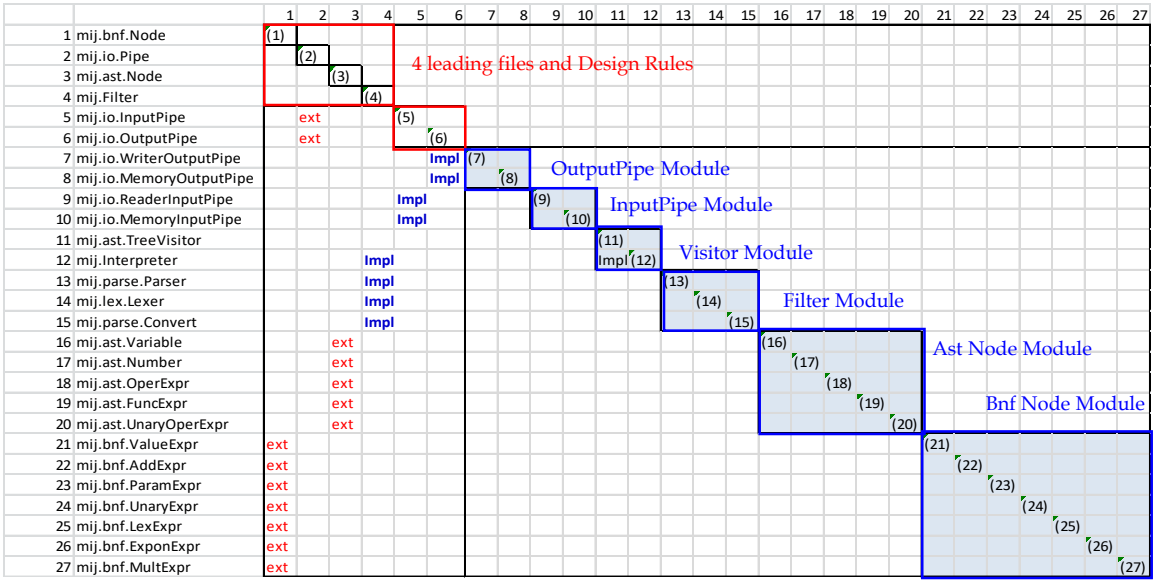
DRSpace 1: Polymorphism DRSpace : formed by “extend” and “implement”



Design Rule Space (DRSpace)

A small calculator example

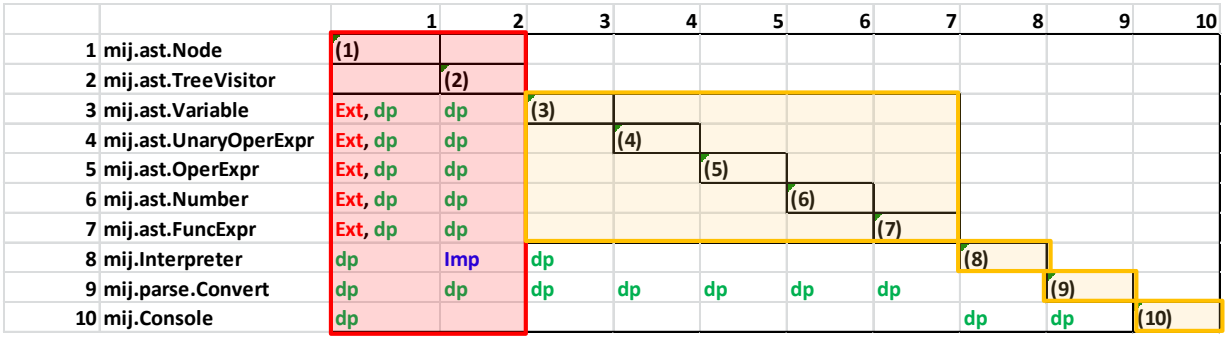
DRSpace 1: Polymorphism DRSpace : formed by “extend” and “implement”



Design Rule Space (DRSpace)

A small calculator example

DRSpace 2: Visitor Pattern DRSpace: formed by multiple dependency types



		1	2	3	4	5	6	7	8	9	10
1	mij.ast.Node	(1)									
2	mij.ast.TreeVisitor		(2)								
3	mij.ast.Variable	Ext, dp	dp	(3)							
4	mij.ast.UnaryOperExpr	Ext, dp	dp		(4)						
5	mij.ast.OperExpr	Ext, dp	dp			(5)					
6	mij.ast.Number	Ext, dp	dp				(6)				
7	mij.ast.FuncExpr	Ext, dp	dp					(7)			
8	mij.Interpreter	dp	Imp	dp					(8)		
9	mij.parse.Convert	dp	dp	dp	dp	dp	dp	dp		(9)	
10	mij.Console	dp							dp	dp	(10)

Ext: Extend **Impl**: Implement
dp: General dependencies such as call, use, and create

- Visitor interface: mij.ast.TreeVisitor
- Element interface: mij.ast.Node
- Concrete elements of the pattern: m(rc3-7)
- Concrete visitor role: Calculator

Design Rule Space (DRSpace)

A small calculator example

DRSpace 2: Visitor Pattern DRSpace: Viewing the *evolutionary coupling* and the *modular structure* of a DRSpace *simultaneously* helps to reveal flawed architectural connections.

		1	2	3	4	5	6	7	8	9	10
1	mij.ast.Node	(1)									
2	mij.ast.TreeVisitor		(2)								
3	mij.ast.Variable	Ext, dp, 5	dp	(3)							
4	mij.ast.UnaryOperExpr	Ext, dp, 10	dp		(4)						
5	mij.ast.OperExpr	Ext, dp, 25	dp		5	(5)					
6	mij.ast.Number	Ext, dp, 8	dp		20		(6)				
7	mij.ast.FuncExpr	Ext, dp, 6	dp		15			(7)			
8	mij.Interpreter	dp, 10	Imp	dp					(8)		
9	mij.parse.Convert	dp	dp	dp	dp	dp	dp	dp		(9)	
10	mij.Console	dp							dp	dp	(10)

Ext: Extend **Impl**: Implement
dp: General dependencies such as call, use, and create

- Visitor interface: mij.ast.TreeVisitor
- Element interface: mij.ast.Node
- Concrete elements of the pattern: m(rc3-7)
- Concrete visitor role: Calculator

Step 2: Automated Architecture Analysis

Overview

Step 2.1 Measure and Monitor

Compare, Contrast, and Monitor

Step 2.2: Pinpoint Hotspots

Detection and Visualization

Step 2.3: Quantify Design Debt

Costs and benefits

Step 2.1 Measure and Monitor

A new metric suite

- Decoupling Level (DL) [Ran et al. 2016]:
  an options-based metric, measuring the system's ability to generate options
- Propagation Cost (PC) [MacCormack 2006]:
  a DSM-based metric, measuring how tightly coupled a system is

The Concept of “Metrics”

What is a "real" metric?

x Software “metrics”:

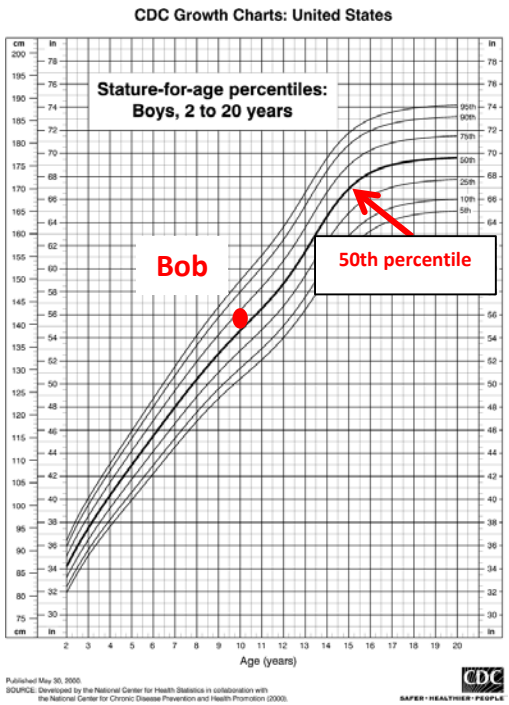
Lines of Code (LOC), CK metrics, McCabe complexity

✓ Real metrics

Meter, Pound, Kilogram

The Properties of a Real Metric

Toward a real metric



Support cross-project comparison

Precisely tell maintainability differences
Independent of project domains, sizes, ages, etc.

Support Evolution Monitoring

Remain stable for non-refactored versions
Non-trivial changes should reflect architecture variation
Architectural debt thermometer

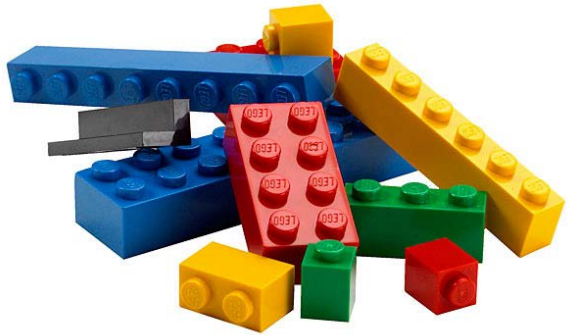
Form a Maintainability Chart

Similar to real health chart
An industrial benchmark

Decoupling Level (DL): Towards a Real Metric

Rationale

- Module: A true module (with high option value) should be
  - Small
  - Independent
- System: A highly modularized system should
  - Have large numbers of true modules…
  - connected by design rules



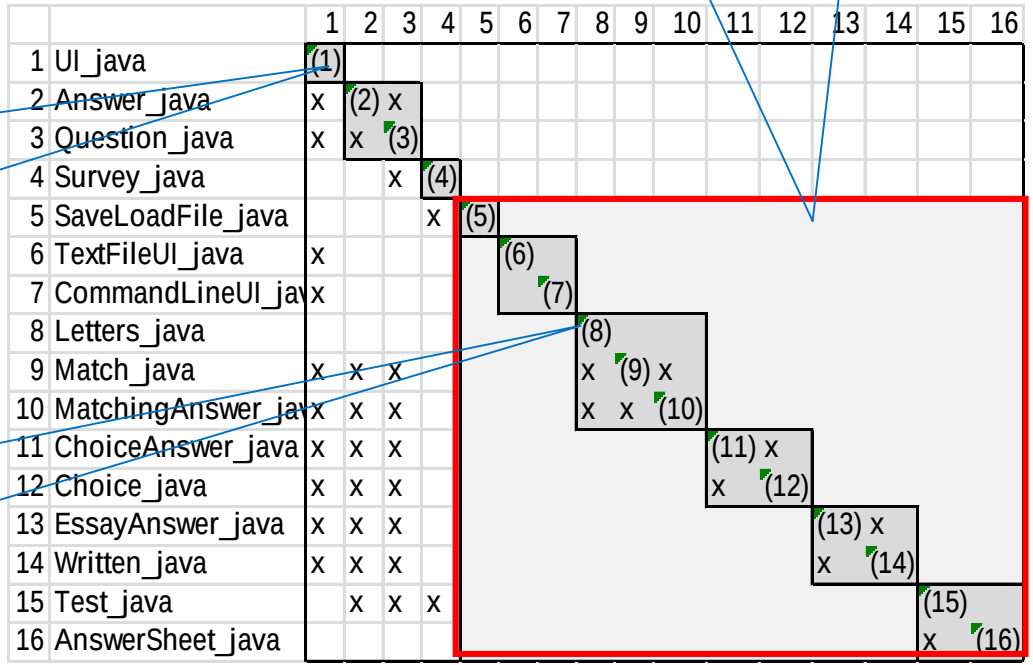
Decoupling Level (DL)

Algorithm

The more files are clustered into true module, the higher the value

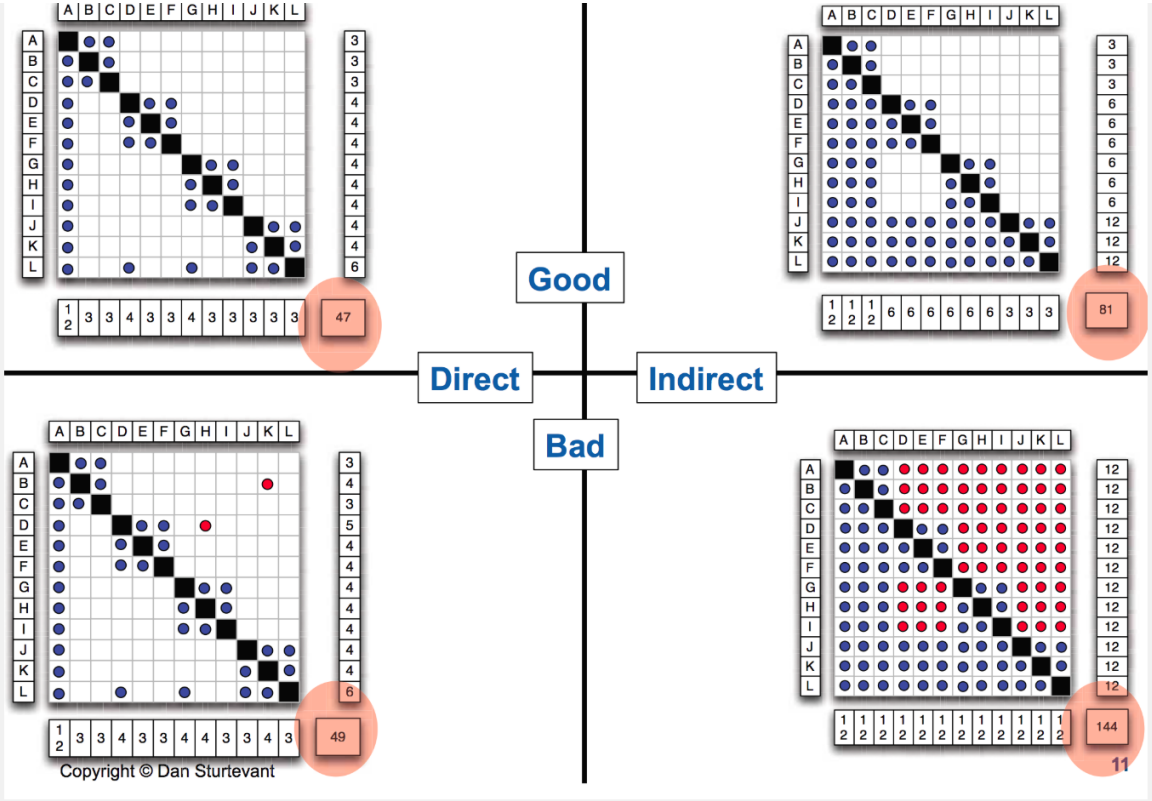
Upper Layer modules:
- The fewer dependents, the higher the value
- The smaller the module, the higher the value

True modules:
- The smaller a true module, the higher the value
- The more true modules, the higher the value



Propagation Cost (PC)

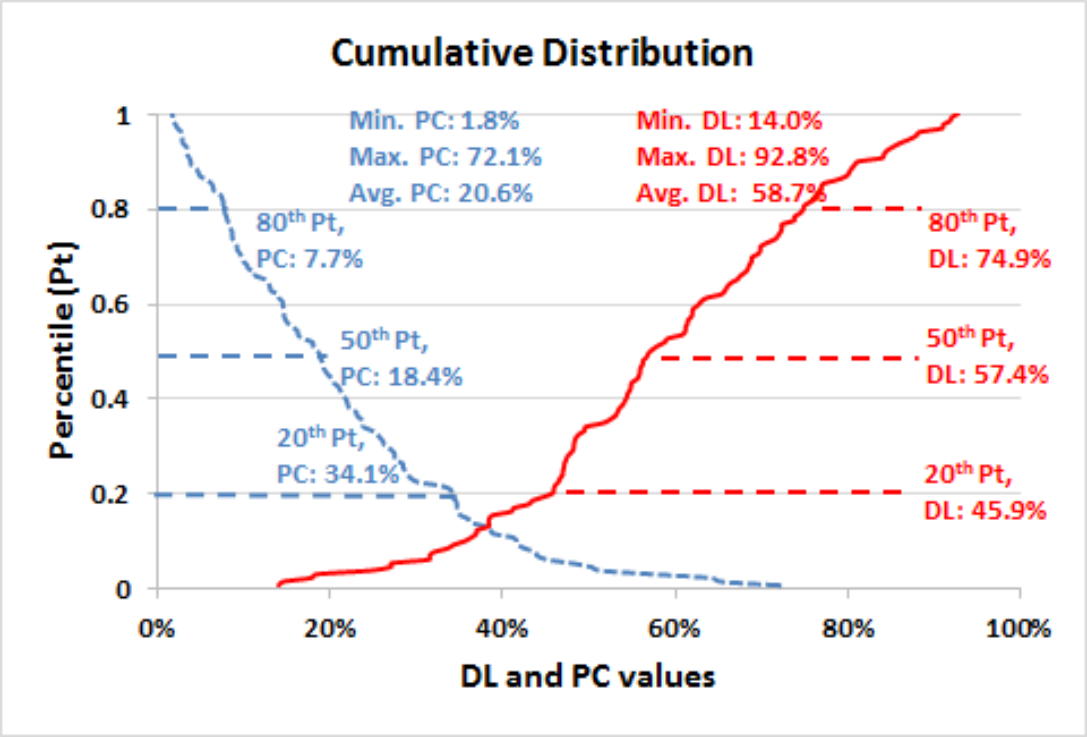
Rationale



Dan Sturtevant, Technical Debt in Large Systems, http://sdm.mit.edu/news/news_articles/webinar_050613/sturtevant_050613.pdf

Decoupling Level (DL) and Propagation Cost (PC)

From 129 open source projects: 108 open source, 21 industrial



Decoupling Level (DL) and Propagation Cost (PC)

Cross project comparison

	Open Source(%)			Commercial(%)		
	DL	PC	IL	DL	PC	IL
Avg	60	20	43	54	21	35
Median	58	18	41	56	20	28
Max	92	72	100	93	50	83
Min	14	2	12	15	2	9
20th Pt	47	8	28	36	6	24
40th Pt	55	14	37	46	17	26
60th Pt	66	21	45	59	24	38
80th Pt	75	34	55	65	35	46

Pt: Percentile

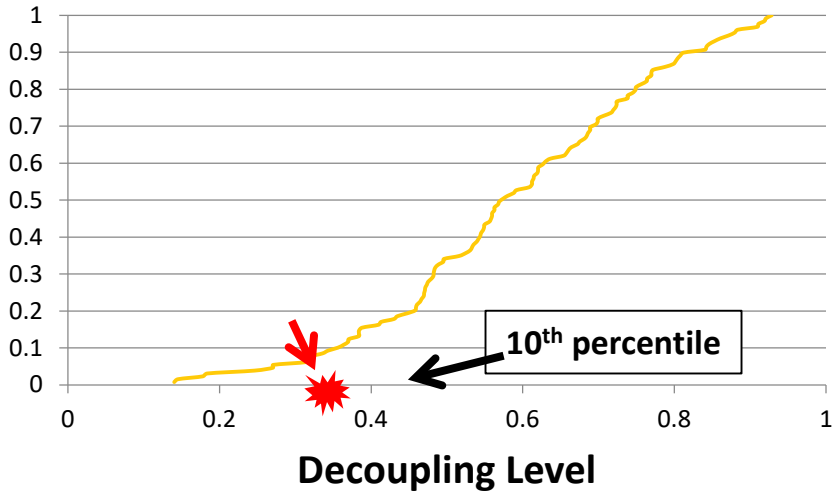
- Best DL (93%) is from industrial
- Worst DL (14%) is from open source

Decoupling Level (DL) and Propagation Cost (PC)

Towards a Heath Chart

An industrial project:

DL: 29%, 10th percentile: Confirmed to have severe maintenance difficulty

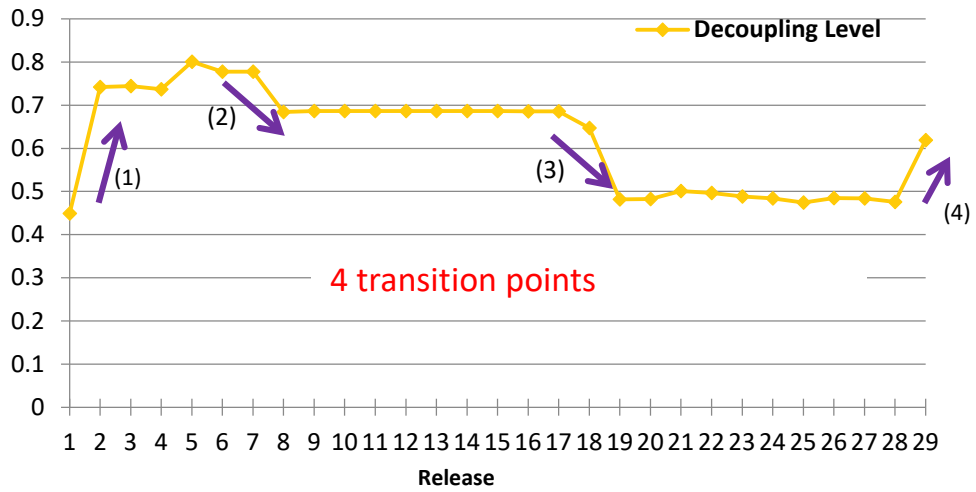


Evolutionary Monitoring

Towards a Heath Chart

Non-trivial variation in DL indicate major architecture variation

- An industrial project: Evolved for 6 year, 29 releases, 1082-1852 files

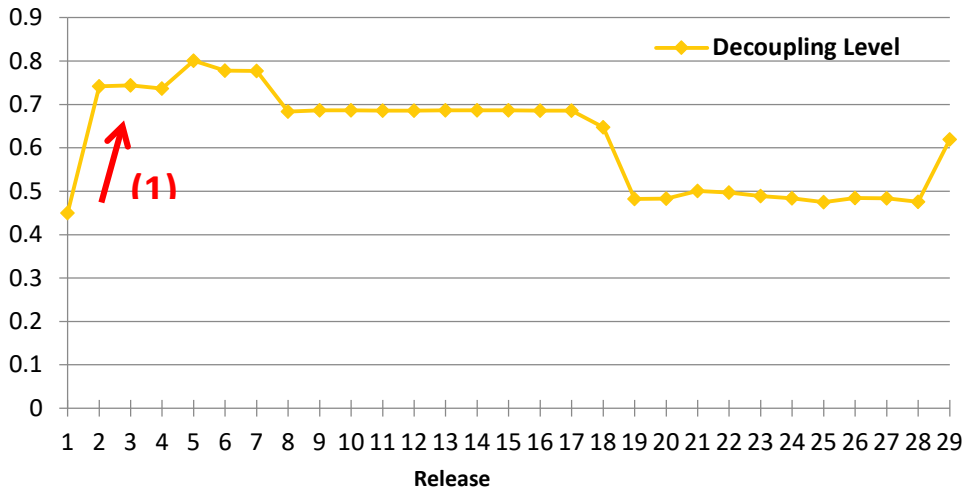


Evolutionary Monitoring

Non-trivial variation in DL faithfully indicate major architecture variation

1 DL increases from 45% to 74%:

Transforming from a prototype to a real product

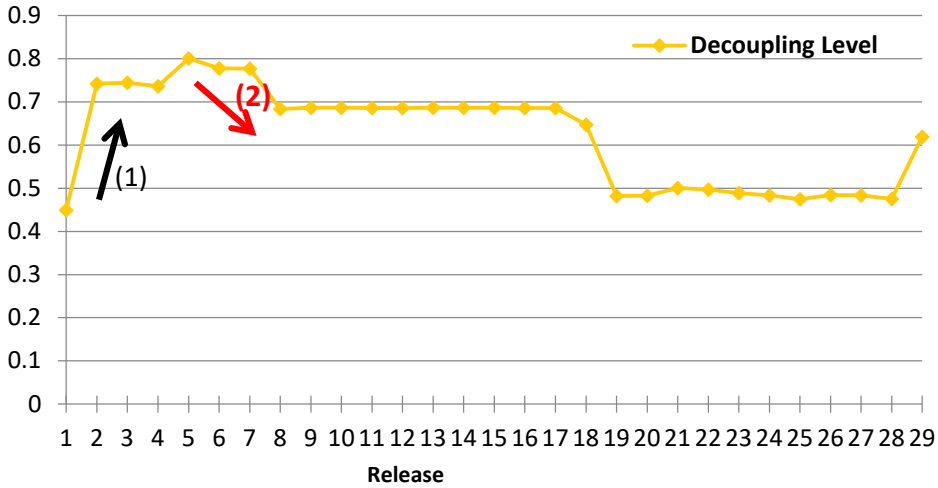


Evolutionary Monitoring

Non-trivial variation in DL faithfully indicate major architecture variation

2 DL decreases from 78% to 68%:

New features were added and Technical debt accumulates



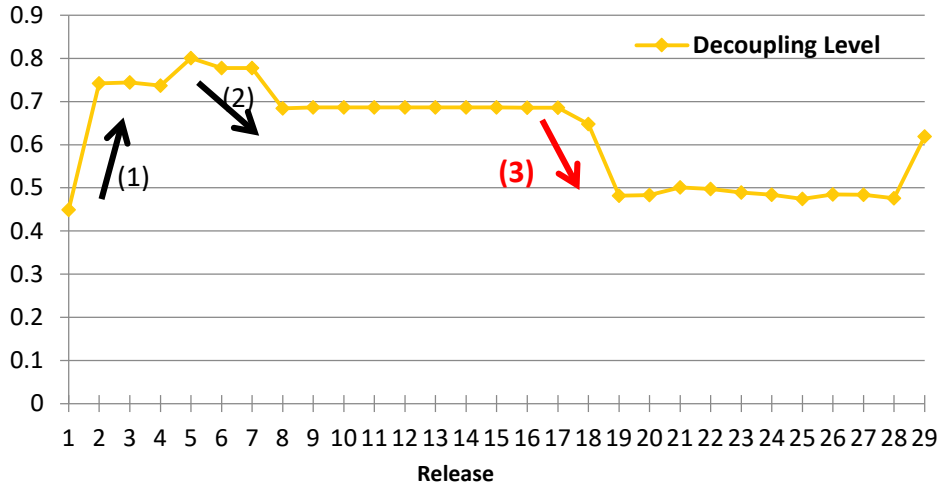
Evolutionary Monitoring

Non-trivial variation in DL faithfully indicate major architecture variation

3

DL decreases from 65% to 48%:

Unsuccessful refactoring



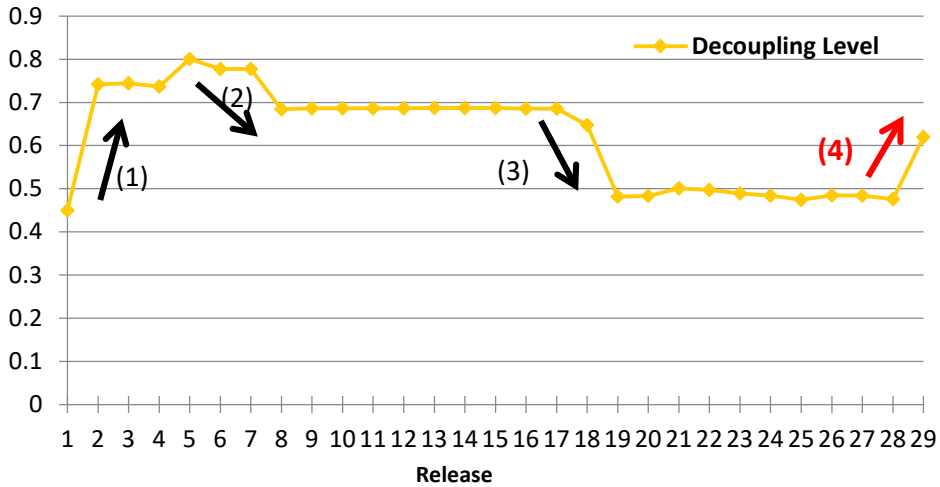
Evolutionary Monitoring

Non-trivial variation in DL faithfully indicate major architecture variation

4

DL increases from 48% to 62%:

"Cleaning up" technical debt



Step 2.2 Pinpoint Hotspots: (1) Flaw Detection

Flaws responsible for high-maintenance

We automatically identify 6 types of design flaws

1. Unstable interface
2. Modularity violation
3. Crossing
4. Improper inheritance
5. Cliques among files
6. Package cycles

These flaws are highly correlated with bugs, changes, and churn

Flaw 1: Unstable Interface

Sample Flaw

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	config.DatabaseDescriptor	(1)	lp,44	14	,10	,10	,6	,14	,36	,118	,12	,	,16	,12	,42	,52	,4	,18	,30
2	utils.FBUtilities	dp,44	2)	,40	,4	,6	,10	,6	,12	,38	,28	,12	,8	,14	,24	,46	,6	,18	,28
3	utils.ByteBufferUtil	,14	lp,40	3)	,			,	,	,4	,10	,20	,4	,4		,10	,26		,12	,4
4	service.WriteResponseHandler	,10	lp,4	2	(4)	,4	,6	,18	dp,22	,							,6	,		,
5	locator.TokenMetadata	,10	6		,4	(5)	,4	,10	dp,24	,	,8						,4	,6	,4	,
6	locator.NetworkTopologyStrategy	,6	lp,10	2	,6	dp,4	(6)	,10	ih,22	,4	,	,					,16	,		,8
7	service.DatacenterWriteResponseHandler	dp,14	lp,6	2	ih,18	,10	dp,10	(7)	,20		,	,					,6	,6		,
8	locator.AbstractReplicationStrategy	,36	lp,12	4	dp,22	ag,24	,22	dp,20	(8)	,6							,16	,10	,	,10
9	config.CFMetaData	,118	lp,38	dp,10	,	,	,4		,6	(9)			,16		,36	,46				,56
10	dht.RandomPartitioner	,12	lp,28	dp,20		,8	,	,			(10)	dp,4			,4	,16		,50		
11	utils.GuidGenerator	,	lp,12	4			,	,			,4	(11)				,4		,		
12	io.sstable.SSTable	,16	8	dp,4						ag,16			(12)	,4	dp,68	,10			,	
13	utils.CLibrary	,12	lp,14										,4	(13)	,12	,				
14	io.sstable.SSTableReader	dp,42	24	dp,10						,36	,4		ih,68	dp,12	(14)	,22	,4	,	,10	
15	cli.CliClient	,52	lp,46	dp,26	,6	,4	,16	,6	,16	,46	,16	,4	,10	,	,22	(15)	,6	,14	,48	
16	locator.PropertyFileSnitch	,4	lp,6		,	dp,6	,	,6	,10						,4	,6	(16)		,4	
17	dht.OrderPreservingPartitioner	dp,18	lp,18	dp,12		,4			,		,50	,			,	,14		(17)		
18	thrift.ThriftValidation	dp,30	28	dp,4	,	,	,8	,	dp,10	dp,56			,		,10	,48	,4		(18)	

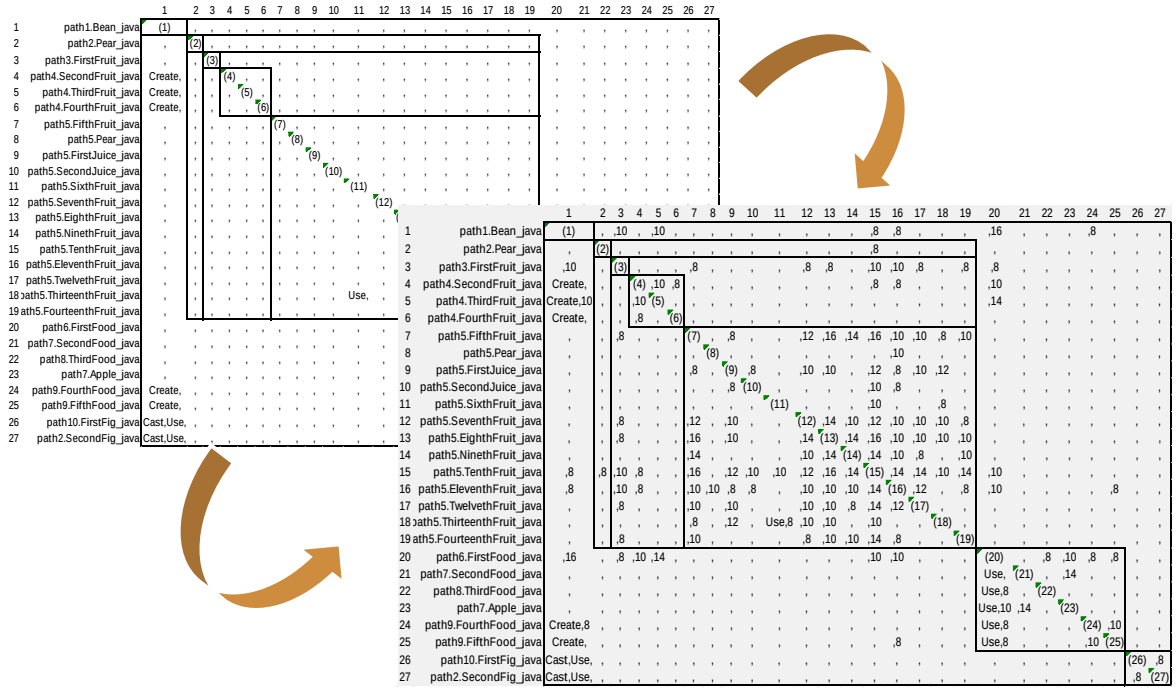
Flaw 2: Crossing

Sample Flaw

		1	2	3	4	5	6	7	8	9	10	11	12	13
1	path1.File1_h	(1)	, 2		, 6	, 11	, 3	, 2	, 8	, 6	, 5		, 2	
2	path1.File2_h	d, 2	(2)			, 2								
3	path1.File3_h	d		(3)		, 2			, 2				, 2	
4	path2.File1_h	, 6			(4)	, 5	, 2	, 2	, 5	, 5	, 4		, 2	
5	path2.File2_h	d, 11	d, 2	d, 2	d, 5	(5)	, 3	, 2	, 10	, 6	, 5	, 2	, 3	, 2
6	path2.File3_h	d, 3			, 2	d, 3	(6)	d, 2	, 2	, 3	, 2		, 2	
7	path3.File1_h	d, 2			, 2	d, 2	d, 2	(7)	, 2	, 2	, 2		, 2	
8	path3.File2_c	d, 8		, 2	, 5	d, 10	d, 2	d, 2	(8)	, 6	d, 5		, 3	
9	path3.File3_c	d, 6			, 5	d, 6	d, 3	d, 2	d, 6	(9)	, 8		, 2	
10	path4.File1_c	d, 5			, 4	d, 5	d, 2	d, 2	d, 5	d, 8	(10)		, 2	
11	path4.File2_c	d				d, 2						(11)		, 7
12	path4.File3_c	d, 2		, 2	, 2	d, 3	d, 2	d, 2	, 3	, 2	, 2		(12)	
13	path5.File1_c	d	d	d		d, 2	d	d	d			d, 7	d	(13)

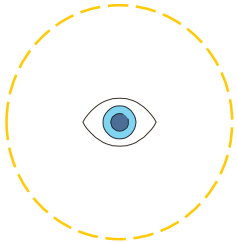
Flaw 3: Modularity Violation

Sample Flaw



Step 2.2 Pinpoint Hotspots: (2) Root Detection

Design Spaces that Grow and Propagate Errors



Architecture Roots

DRSpaces propagate errors among multiple files through flawed connections

The more error-prone files are, the more likely they are architecturally connected

Observations about Roots

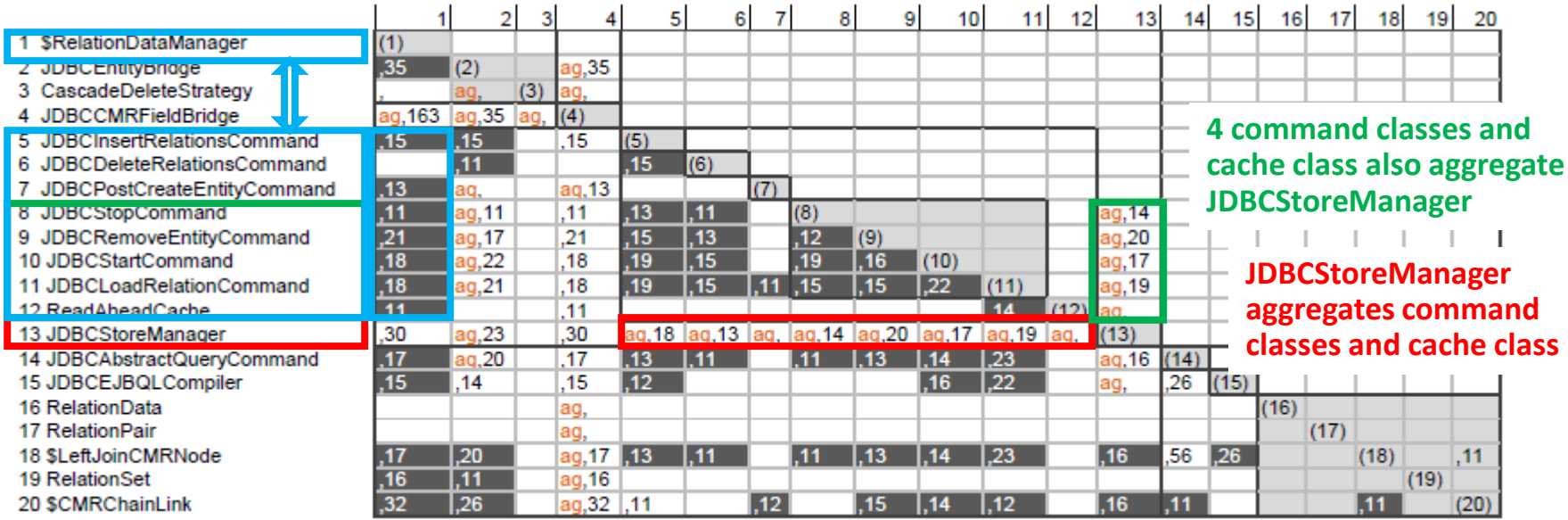
File in the DRSpaces led by an error prone file is also error-prone

A few (~5) DRSpaces capture more than half of the most error-prone files.

Error-prone DRSpaces often have architectural flaws

Sample Root from JBoss

Each root has multiple design problems



Jboss JDBCCMRFieldBridge DRSpace

The Impact of Roots in Open Source Projects

The observation is consistent with other projects

Consider JBoss: 5 Root spaces captured 52% of all files with more than 5 bugs

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%

Step 2.3 Quantification

Calculate the Costs and Benefits

- Calculate the costs of each root, each flaw and each types of flaws
- Calculate ROI (Return on Investment)

The Costs of Each Flaw and Each Type of Flaw

Sample output from DV8

	#Instances	#Files	Pt.	Flaw CF	Pt.	Flaw CC	Pt.	Flaw BF	Pt.	Flaw BC	Pt.
Clique	26	322	21%	1,790	28%	26,294	41%	643	34%	16,557	45%
Crossing	91	368	24%	3,146	50%	40,247	63%	1,051	55%	25,177	68%
ModularityViolation	667	588	38%	4,538	72%	46,224	72%	1,438	75%	27,648	74%
PackageCycle	175	499	32%	2,417	38%	29,906	47%	778	41%	18,889	51%
UnstableInterface	6	316	21%	1,669	26%	19,898	31%	388	20%	11,457	31%
UnhealthyInheritance	72	257	17%	1,528	24%	22,007	34%	480	25%	13,481	36%

Instance Name	Size	Tot. CF	Tot. CC	Tot. BF	Tot. BC
Clique1	99	226	7,847	112	4,673
Clique2	78	181	431	7	212
Clique3	28	181	1,686	49	897
Clique4	18	246	3,130	39	1,427
Clique5	16	168	3,553	89	2,662

Return on Investment Calculation

Experience with SoftServe [Kazman et al. 2015]

Penalty Caused by Architecture Debt **Refactoring Cost** **Expected Savings**

	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)**	**orm Exp efects/Yr**	**Norm Exp Changes/Yr**	**Norm Exp LOC Changed**
2	*Pear.java*	139	119.33	166	142.5	1068	839.2	49,171	42,213	5.5	39	346	20,281
3	*Apple.java*	158	133.83	63	53.4	607	451.7	25,603	21,686	7	44	388	22,745
4	*Bean.java*	65	37.83	72	41.9	429	207.2	17,807	10,364	1.5	12	110	6,429
5													
6	**DRSpace Total**		**290.99**		**237.8**		**1498**		**74,263**		**96.0**	**843.871**	**49,455**
7	**Project Total**	**797**		**265**		**2332**		**135,453**		**14**			
8	**Savings**										**142**	**654**	**24,808**
9													
10													
11	Base defect rates	0.33											
12	Base change rates	2.9										**Exp PM save**	**41.35**
13	Base LOC/file	169.95											
14	LOC/PM	600											

Result: ~300% ROI in the first year alone!

Analyzing 8 ABB Projects

Recent and Representative Experiences

- Using 3 complementary techniques:
  - Architecture-level maintainability metrics
  - Architecture flaw analysis and root detection
  - Cost and benefit analysis
- 8 projects developed at multiple locations (India, USA, and Switzerland) and differ in their ages, domains, and sizes.
- We reported the results back to each project and collected feedback

Research Questions

Collecting feedbacks of practitioners

? RQ1: does the tool suite help to close the gap between management and development? That is, does it help them to decide ***if***, ***when***, and ***where*** to refactor?

? RQ2: does the tool suite help practitioners understand the maintainability of their systems ***relative to other projects*** internal to the company, and relative to a more broad- based benchmark suite?

? RQ3: does the tool suite help developers pinpoint the ***hotspots*** of their systems, i.e., the groups of files with severe design flaws?

Metrics Scores and Rankings

Measure and Monitor

	DL	Percentile	PC	Percentile	#Files
Proj_EO	78%	85th	6%	85th	144
Proj_BM	77%	85th	2%	98th	371
Proj_CH	76%	81st	16%	54th	6,948
Proj_EP	72%	74th	7%	83th	1,541
Proj_SS	57%	49th	20%	45th	15,333
Proj_OP	57%	49th	21%	41th	7,754
Proj_CO	55%	43rd	17%	52th	491
Proj_EC	28%	5th	62%	2nd	4,125

Hotspots Detected

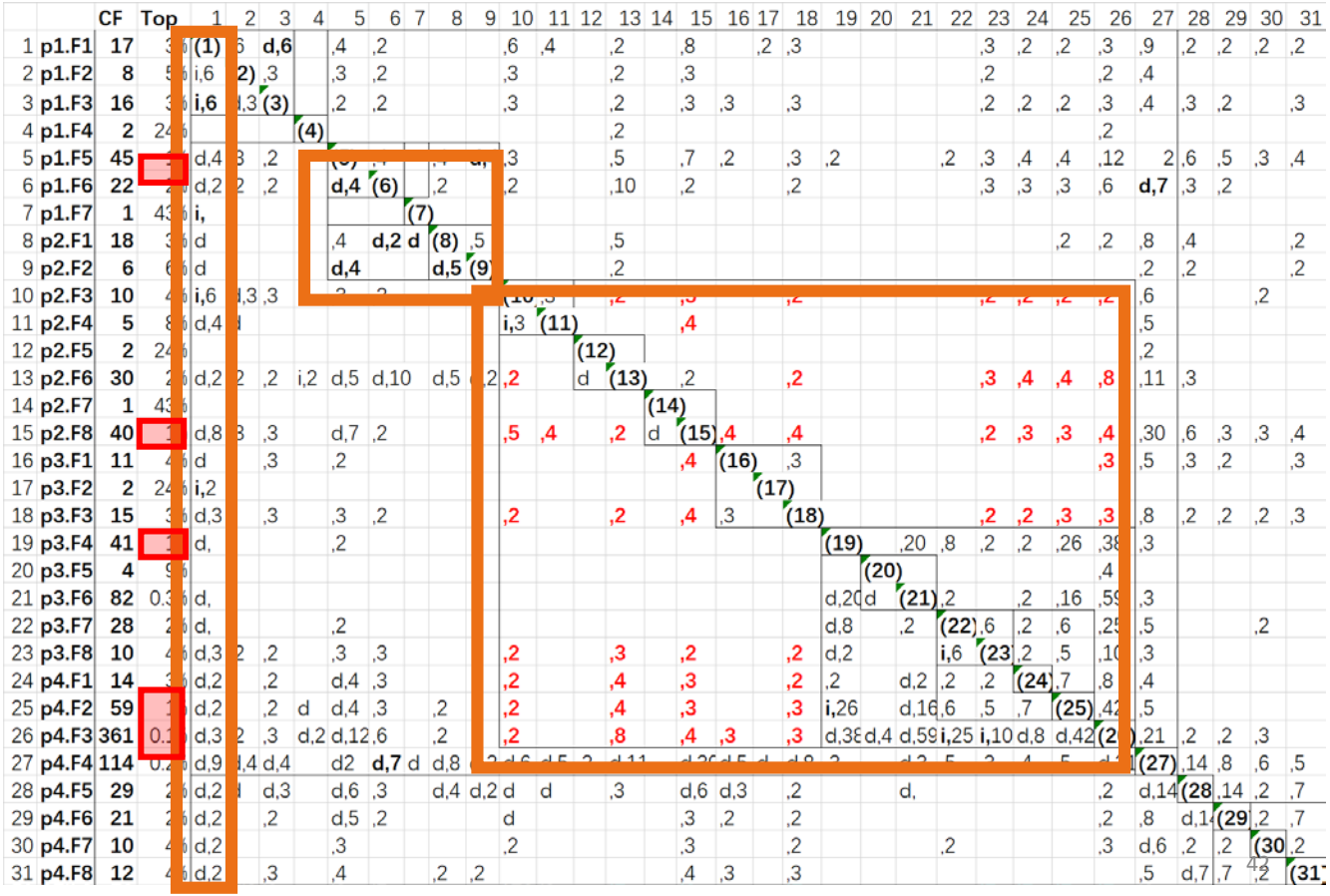
Pinpoint Hotspots

Unstable Interface

File Cliques

Modularity Violations

Most error-prone files



		CF	Top	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
1	p1.F1	17	3%	(1)	6	d,6		,4	,2				,6	,4		,2		,8		,2	,3					,3	,2	,2	,3	,9	,2	,2	,2	,2
2	p1.F2	8	5%	i,6	(2)	,3		,3	,2				,3			,2		,3								,2			,2	,4				
3	p1.F3	16	3%	i,6	d,3	(3)		,2	,2				,3			,2		,3	,3		,3					,2	,2	,2	,3	,4	,3	,2		,3
4	p1.F4	2	24%				(4)									,2													,2					
5	p1.F5	45	1%	d,4	3	,2							,3			,5		,7	,2		,3	,2			,2	,3	,4	,4	,12	2	,6	,5	,3	,4
6	p1.F6	22	2%	d,2	2	,2		d,4	(6)		,2		,2			,10		,2			,2					,3	,3	,3	,6	d,7	,3	,2		
7	p1.F7	1	43%	i,						(7)																								
8	p2.F1	18	3%	d				,4	d,2	d	(8)	,5				,5												,2	,2	,8	,4			,2
9	p2.F2	6	6%	d				d,4			d,5	(9)				,2														,2	,2			,2
10	p2.F3	10	4%	i,6	3	,3																								,6			,2	
11	p2.F4	5	8%	d,4	4								i,3	(11)				,4												,5				
12	p2.F5	2	24%												(12)															,2				
13	p2.F6	30	2%	d,2	2	,2	i,2	d,5	d,10		d,5	2	,2		d	(13)		,2			,2					,3	,4	,4	,8	,11	,3			
14	p2.F7	1	43%														(14)																	
15	p2.F8	40	1%	d,8	3	,3		d,7	,2				,5	,4		,2	d	(15)	,4		,4					,2	,3	,3	,4	,30	,6	,3	,3	,4
16	p3.F1	11	4%	d		,3		,2										,4	(16)		,3								,3	,5	,3	,2		,3
17	p3.F2	2	24%	i,2																(17)														
18	p3.F3	15	3%	d,3		,3		,3	,2				,2			,2		,4	,3		(18)					,2	,2	,3	,3	,8	,2	,2	,2	,3
19	p3.F4	41	1%	d,				,2														(19)	,20	,8	,2	,2	,26	,38	,3					
20	p3.F5	4	9%																				(20)					,4						
21	p3.F6	82	0.3%	d,																		d,20	d	(21)	,2		,2	,16	,59	,3				
22	p3.F7	28	2%	d,				,2														d,8		,2	(22)	,6	,2	,6	,25	,5			,2	
23	p3.F8	10	4%	d,3	2	,2		,3	,3				,2			,3		,2			,2	d,2			i,6	(23)	,2	,5	,10	,3				
24	p4.F1	14	3%	d,2		,2		d,4	,3				,2			,4		,3			,2	,2		d,2	,2	,2	(24)	,7	,8	,4				
25	p4.F2	59	1%	d,2		,2	d	d,4	,3		,2		,2			,4		,3			,3	i,26		d,16	,6	,5	,7	(25)	,42	,5				
26	p4.F3	361	0.1%	d,3	2	,3	d,2	d,12	,6		,2		,2			,8		,4	,3		,3	d,38	d,4	d,59	i,25	i,10	d,8	d,42	(2	,21	,2	,2	,3	
27	p4.F4	114	0.2%	d,9	4	d,4		d2	d,7	d	d,8		2	d,6	d,5	2	d,11		d,26	d,5	d	d,8	2		d,2	5	2	4	5	d	(27)	,14	,8	,6	,5
28	p4.F5	29	2%	d,2	4	d,3		d,6	,3		d,4	d,2	d	d		,3		d,6	d,3		,2			d,					,2	d,14	(28	,14	,2	,7
29	p4.F6	21	2%	d,2		,2		d,5	,2				d					,3	,2		,2								,2	,8	d,14	(29	,2	,7
30	p4.F7	10	4%	d,2				,3					,2					,3			,2				,2				,3	d,6	,2	,2	(30	,2
31	p4.F8	12	4%	d,2		,3		,4			,2	,2						,4	,3		,3									,5	d,7	,7	,2	(31	

Costs and Benefits of Architecture Debts

Costs and benefits

Debt Panelty

	Size (%)	Rt. CF (%)	Rt. CC (%)	Rt. BF (%)	Rt. BC (%)
root1	147 (10%)	1,109 (18%)	13,487 (21%)	414 (22%)	9,347 (25%)
root2	93 (6%)	1,050 (17%)	11,486 (18%)	452 (24%)	6,696 (18%)
root3	79 (5%)	601 (10%)	5,453 (9%)	183 (10%)	3,821 (10%)
root4	104 (7%)	486 (8%)	10,794 (17%)	166 (9%)	6,236 (17%)

			Coverage	
Root	Size	% Size	Change	Bug
root1	147	10%	24%	29%
root2	222	14%	38%	52%
root3	263	17%	47%	57%
root4	364	24%	55%	65%
	Penalty of Architecture Roots			
	Extra CF	Extra CC	Extra BF	Extra BC
root1	612	8,450	263	6,418
root2	1,332	16,601	615	11,175
root3	1,687	19,570	724	13,552
root4	1,754	26,110	763	17,314
Percentage	28%	41%	40%	47%

Expected Benefts

Step 3: Collecting User feedback

Answering research questions

- Surveys
- Interviews

Post-Mortem

Survey with practitioners

Q1: What did the report reveal that you didn’t know about your software?

Q2: Are the metrics useful for reflecting the architecture of your software?

Q3: What did the architecture design flaws reveal about your software?

Q4: What actions have you planned as a result of the architecture design flaws report?

Q5: What did the architecture roots reveal about your software?

Q6: What actions do you plan to take to address architecture roots?

Results

RQ review

RQ1: does the tool suite help to close the gap between management and development? That is, does it help them to decide if, when, and where to refactor?

5 Participants of all 8 projects verified that the information provided was useful in closing the understanding gap with management. They have begun the refactoring process.

Results

RQ review

RQ2: does the tool suite help practitioners understand the maintainability of their systems relative to other projects internal to the company, and relative to a more broad- based benchmark suite?

All participants said the report gave them quantifiable results with which to judge their project. The comparison with industrial benchmarks made it clear that maintenance difficulty caused by degrading architecture is common.

Results

RQ review

RQ3: does the tool suite help developers pinpoint the *hotspots* of their systems—that is, the groups of files with severe design flaws?

✓ Six of the eight projects planned to or already started refactoring to address the detected flaws. The project with the lowest DL score is undergoing a major rewrite.

Lessons Learned

Feedback Summary

- It is possible to automatically and objectively assess and quantify architecture quality.
- These results were enthusiastically received by the projects.
- 6 of the 8 projects are now embarking on major refactorings.
- ABB is incorporating DV8 into its development process

Conclusion

RQ review

- You can't manage it if you don't measure it.
  - DV8 tool suite make measurement of design complexity, design flaws and design debt less labor-intensive and easily repeatable.
- Incorporating these techniques into the build process ensures rapid feedback with supporting data.
- This measurement, detection, and quantification practice leads to improved architectures.

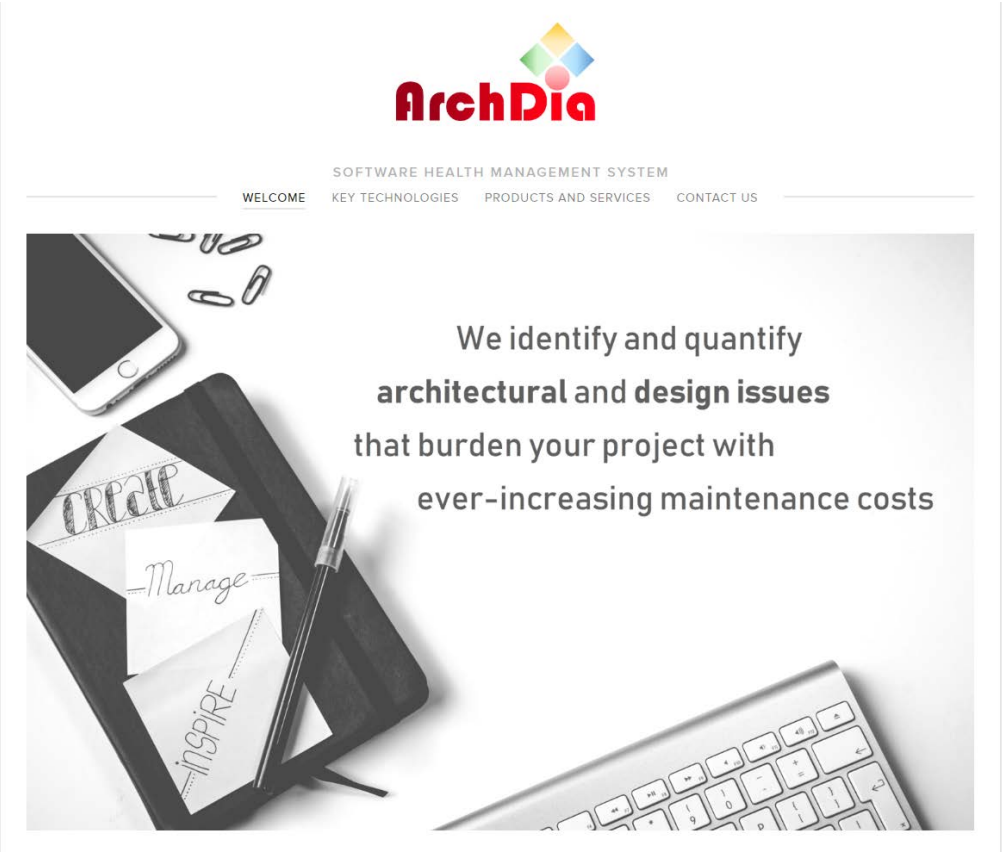
Questions?

Office hour:

Thursday

1:30-2:30

Room E



www.archdia.net (Under Constuction)

References

[Conway 1968] Melvin Conway, (April 1968), "How do Committees Invent?", Datamation, 14 (5): 28–31.

[Baldwin and Clark 1999] Carliss Baldwin, Kim Clark, 1999. Design Rules: The Power of Modularity Volume 1. MIT Press, Cambridge, MA, USA.

[MacCormack 2006] Alan MacCormack, John Rusnak, and Carliss Y. Baldwin: Exploring the structure of complex software designs: An empirical study of open source and proprietary code. Management Science, 52(7):1015–1030, July 2006.

[Parnas 1972] David Parnas, On the criteria to be used in decomposing system into modules, Communications ACM, Vol. 15, No. 12, December 1972 pp. 1053–1058.

References

[Sullivan et al 2001] Kevin J. Sullivan, William G. Griswold, Yuanfang Cai, Ben Hallen, "The structure and value of modularity in software design", ESEC/FSE 2001: 99-108.

[Cai 2006] Yuanfang Cai, Kevin Sullivan, "Modularity Analysis of Logical Design Models", ASE 2006: 91-102

[Wong et al. 2009] Sunny Wong, Yuanfang Cai, Giuseppe Valetto, Georgi Simeonov, Kanwarpreet Sethi, "Design Rule Hierarchies and Parallelism in Software Development Tasks", ASE 2009: 197-208

[Wong et al. 2011] Sunny Wong, Yuanfang Cai, Miryung Kim, Michael Dalton, "Detecting software modularity violations", ICSE 2011: 411-420

References

[Cai et al. 2013] Yuanfang Cai, Rick Kazman, Ciera Jaspan, Jonathan Aldrich, "Introducing tool-supported architecture review into software design education", CSEE&T 2013: 70-79

[Xiao et al. 2014] Lu Xiao, Yuanfang Cai, Rick Kazman, "Titan: a toolset that connects software architecture with quality analysis", SIGSOFT FSE 2014: 763-766

[Xiao et al. 2014] Lu Xiao, Yuanfang Cai, Rick Kazman, "Design rule spaces: a new form of architecture insight", ICSE 2014: 967-977

References

[Naedele et al. 2014] Martin Naedele, Rick Kazman, Yuanfang Cai, "Making the case for a "manufacturing execution system" for software development", Communations of the ACM 57(12): 33-36 (2014)

[Mo et al. 2015] Ran Mo, Yuanfang Cai, Rick Kazman, Lu Xiao, "Hotspot Patterns: The Formal Definition and Automatic Detection of Architecture Smells", WICSA 2015: 51-60

[Kazman et al. 2015] Rick Kazman, Yuanfang Cai, Ran Mo, Qiong Feng, Lu Xiao, Serge Haziyev, Volodymyr Fedak, Andriy Shapochka, "A Case Study in Locating the Architectural Roots of Technical Debt", ICSE 2015: 179-188

References

[Xiao et al. 2016] Lu Xiao, Yuanfang Cai, Rick Kazman, Ran Mo, Qiong Feng, "Identifying and quantifying architectural debt", ICSE 2016: 488-498

[Mo et al. 2016] Ran Mo, Yuanfang Cai, Rick Kazman, Lu Xiao, Qiong Feng, "Decoupling level: a new metric for architectural maintenance complexity", ICSE 2016: 499-510

[Feng et al. 2016] Qiong Feng, Rick Kazman, Yuanfang Cai, Ran Mo, Lu Xiao, "Towards an Architecture-Centric Approach to Security Analysis", WICSA 2016: 221-230