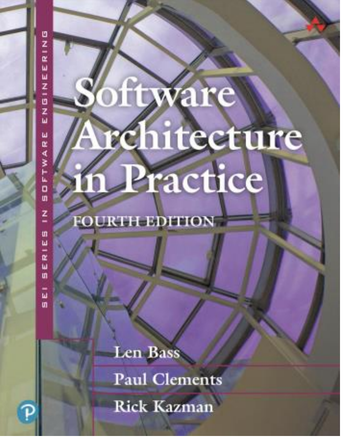
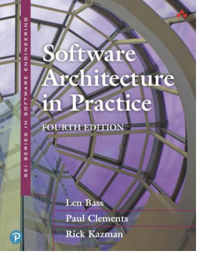


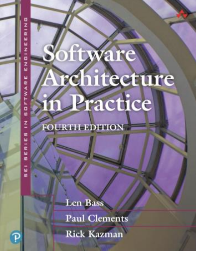
Chapter 23: Managing Architecture Debt

Some debts are fun when you are acquiring them, but none are fun when you set about retiring them.
—Ogden Nash



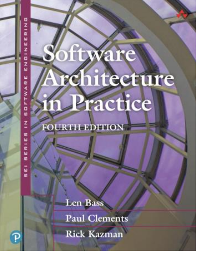
Chapter Outline

- Managing Architecture Debt
- Determining Whether You Have an Architecture Debt Problem
- Discovering Hotspots
- Example
- Automation
- Summary



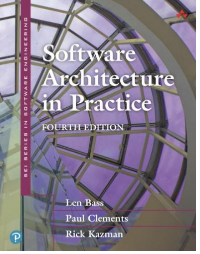
Architecture Debt

- Without careful attention, designs become harder to maintain and evolve over time.
- We call this form of entropy “architecture debt,” and it is an important and highly costly form of technical debt.
- Architecture debt can be removed through refactoring.



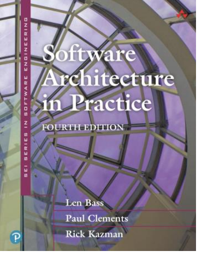
Architecture Debt

- The debt identification process requires three types of information:
  - *Source code*. This is used to determine structural dependencies.
  - *Revision history, as extracted from a project's version control system*. This is used to determine the co-evolution of code units.
  - *Issue information, as extracted from an issue control system*. This is used to determine the reason for changes.



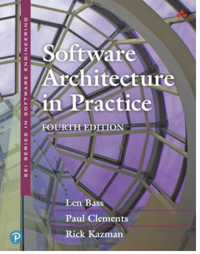
Determining Whether You Have an Architecture Debt Problem

- How to we determine if a group of files is architecturally connected?
  1. identify the static dependencies between the files in your project using a static code analysis tool.
  2. capture the evolutionary dependencies between files in a project. An *evolutionary dependency* occurs when two files change together, and you can extract this information from your revision control system.



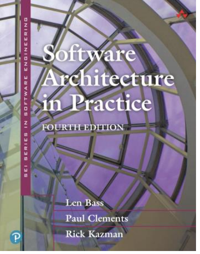
Determining Whether You Have an Architecture Debt Problem

- We represent file dependencies using a design structure matrix (DSM).
- Files are placed on the rows of the matrix and, in the same order, on the columns.
- The cells of the matrix are annotated to indicate the type of dependency.



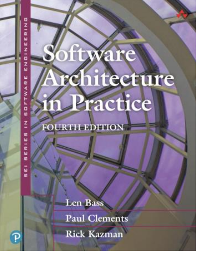
Example DSM from Apache Camel

		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	ExpressionDefinition_java	(1)																															
2	XMLTokenExpressionIterator_java		(2)																														
3	MethodNotFoundException_java			(3)																													
4	ExpressionBuilder_java	dp	dp		(4)					dp																							
5	BeanInfo_java			dp	dp	(5)																											
6	BeanProcessor_java					dp	(6)																										
7	RuntimeBeanExpressionException_java							(7)																									
8	BeanExpression_java						dp	dp	(8)																								
9	MethodCallExpression_java	ex,dp		dp		dp			dp	(9)																							
10	MockEndpoint_java										(10)	dp,im																					
11	AssertionClause_java	dp									dp	(11)																					
12	XMLTokenExpressionIteratorTest_java		dp										(12)																				
13	BeanDefinition_java			dp		dp	dp							(13)																			
14	ExpressionNode_java	dp													(14)	ex,dp																	
15	ProcessorDefinition_java	dp			dp									dp	dp	(15)																	
16	XmlGraphGenerator_java															dp	(16)																
17	MyDummyBean_java																	(17)															
18	BeanExplicitMethodAmbiguousTest_java																	dp	(18)														
19	BuilderSupport_java									dp										(19)	dp												
20	Builder_java				dp																(20)												
21	SplitTokenizerTest_java										dp									dp		(21)											
22	XMLTokenizerExpression_java	dp																					(22)	dp									
23	XMLTokenizeLanguage_java				dp																			(23)									
24	JsonPathTransformTest_java										dp														(24)								
25	XMLTokenizeLanguageTest_java										dp															(25)							
26	XMLTokenizeWrapLanguageTest_java										dp																(26)						
27	TokenizeLanguage_java				dp																							(27)					
28	TokenizerExpression_java	ex																										dp	(28)				
29	XQueryExpression_java	dp																												(29)			
30	XPathExpression_java	dp																													(30)		
31	SimpleExpression_java	ex,dp																														(31)	
32	JsonPathExpression_java	ex,dp																															(32)



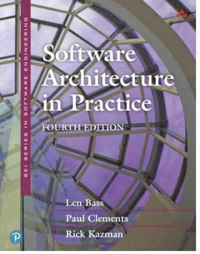
Example DSM with Co-Change Information

	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 ExpressionDefinition_java	(1)								,2	,2	,2			,4	,4							,3						,2	,4	,5	,3	,2
2 XMLTokenExpressionIterator_java		(2)		,3								,6										,3	,3		,3	,3						
3 MethodNotFoundException_java			(3)		,7	,2	,2	,4	,4				,4		,2		,2	,2														
4 ExpressionBuilder_java	dp	dp,3		(4)	,10			,4	dp	,5		,2							,16	,15		,3	,3		,2	,2	,4	,5				
5 BeanInfo_java			dp,7	dp,10	(5)	,16	,4	,7	,3	,2			,4		,2		,4	,5	,3	,2												
6 BeanProcessor_java			,2		dp,16	(6)	,3	,7	,3				,3		,3		,2	,3														
7 RuntimeBeanExpressionException_java			,2		,4	,3	(7)	,5	,2				,2		,2		,2	,2														
8 BeanExpression_java			,4	,4	,7	dp,7	dp,5	(8)	,6				,5		,2		,2	,2														
9 MethodCallExpression_java	ex,dp,2		dp,4		dp,3	,3	,2	dp,6	(9)				,6	,2	,2		,2	,2	,3									,2	,5	,6	,4	
10 MockEndpoint_java	,2			,5	,2					(10)	dp,im,6				,2				,3													
11 AssertionClause_java	dp,2									dp,6	(11)																					
12 XMLTokenExpressionIteratorTest_java		dp,6		,2								(12)										,2	,2		,4	,3						
13 BeanDefinition_java			dp,4		dp,4	dp,3	,2	,5	,6				(13)		,8		,2	,3														
14 ExpressionNode_java	dp,4								,2					(14)	ex,dp,8														,2	,3		
15 ProcessorDefinition_java	dp,4		,2	dp	,2	,3	,2	,2	,2	,2			dp,8	dp,8	(15)		,2	,3														
16 XmlGraphGenerator_java															dp	(16)																
17 MyDummyBean_java			,2		,4	,2	,2	,2	,2				,2		,2		(17)	,4														
18 BeanExplicitMethodAmbiguousTest_java			,2		,5	,3	,2	,2	,2				,3		,3		dp,4	(18)														
19 BuilderSupport_java				,16	,3				dp,3	,3									(19)	dp,19											,2	
20 Builder_java				dp,15	,2														,19	(20)												
21 SplitTokenizerTest_java										dp									dp		(21)											
22 XMLTokenizerExpression_java	dp,3	,3		,3								,2										(22)	dp,4		,2	,2			,2	,2		
23 XMLTokenizeLanguage_java		,3		dp,3								,2										,4	(23)		,2	,2						
24 JsonPathTransformTest_java										dp														(24)								
25 XMLTokenizeLanguageTest_java		,3		,2						dp		,4										,2	,2		(25)	,3						
26 XMLTokenizeWrapLanguageTest_java		,3		,2						dp		,3										,2	,2		,3	(26)						
27 TokenizeLanguage_java				dp,4																							(27)	,6				
28 TokenizerExpression_java	ex,2			,5					,2																		dp,6	(28)	,2	,2		
29 XQueryExpression_java	dp,4								,5					,2								,2						,2	(29)	,11	,3	
30 XPathExpression_java	dp,5								,6					,3								,2						,2	,11	(30)	,5	,2
31 SimpleExpression_java	ex,dp,3								,4										,2										,3	,5	(31)	,2
32 JsonPathExpression_java	ex,dp,2																													,2	,2	(32)



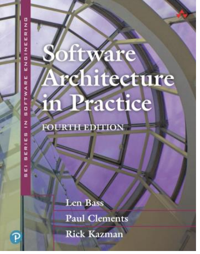
Discovering Hotspots

- To target architecture debt you need to identify the specific files and their flawed relationships.
- We call the sets of elements that make outsized contributions to the maintenance costs of a system *hotspots*.
- To identify hotspots, we look for anti-patterns that contribute to high coupling and low cohesion.



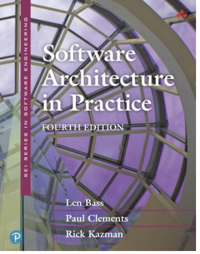
Discovering Hotspots

- Six common anti-patterns that lead to debt:
  - *Unstable interface*
  - *Modularity violation*
  - *Unhealthy inheritance*
  - *Cyclic dependency or clique*
  - *Package cycle*
  - *Crossing*



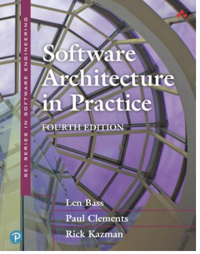
Unstable Interface

- An influential file—one representing an important service, resource, or abstraction—changes frequently with its dependents, as recorded in the revision history.
- The “interface” file is the entry point for other system elements to use the service or resource.
- It is frequently modified due to internal reasons, changes to its API, or both.
- To identify this anti-pattern, search for a file with a large number of dependents that is modified frequently with other files.



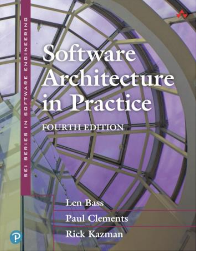
Modularity Violation

- Structurally decoupled modules frequently change together.
- To identify this anti-pattern, search for two or more structurally independent files—that is, files that have no structural dependency on each other—that change together frequently.



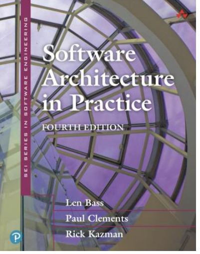
Unhealthy Inheritance

- A base class depends on its subclasses or a client class depends on both the base class and one or more of its subclasses.
- To determine unhealthy inheritance instances, search for either of the following in a DSM:
  - In an inheritance hierarchy, a parent depends on its child class.
  - In an inheritance hierarchy, a client of the class hierarchy depends on both the parent and one or more of its children.



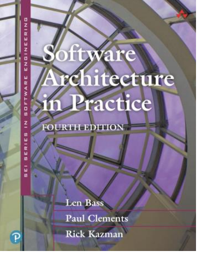
Cyclic Dependency or Clique

- A group of files is tightly connected.
- To identify this anti-pattern, search for sets of files that form a strongly connected graph, where there is a structural dependency path between any two elements of the graph.



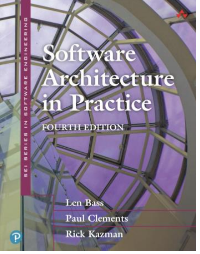
Package Cycle

- Two or more packages depend on each other, rather than forming a hierarchical structure, as they should.
- Detecting this anti-pattern is similar to detecting a clique: A package cycle is determined by discovering packages that form a strongly connected graph.



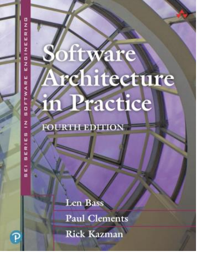
Crossing

- A file has both a high number of dependent files and a high number of files on which it depends, and it changes frequently with its dependents and the files it depends on.
- To determine the file at the center of a crossing, search for a file that has both high fan-in and fan-out and substantial co-change relations with other files.



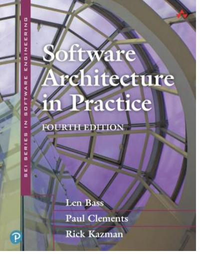
Clique in Apache Cassandra

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



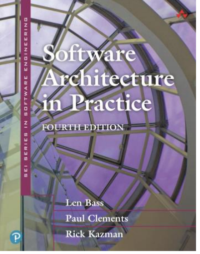
Unhealthy Inheritance in Apache Cassandra

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



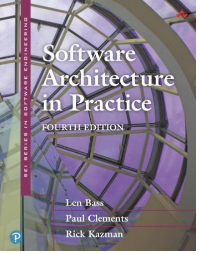
Fixing Hotspots

- Most issues in an issue tracking are either bug fixes or feature enhancements.
- Bug fixes and both bug-related and change-related churn are highly correlated with anti-patterns and hotspots.
- We can assign a *penalty* to each anti-pattern using this information.



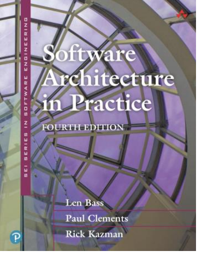
Fixing Hotspots

- With this analysis a debt-reduction strategy (through refactoring) is straightforward.
- Knowing the files implicated in the debt, along with their flawed relationships (as determined by the identified anti-patterns), allows the architect to fashion and justify a refactoring plan.
  - If a clique exists, for example, a dependency needs to be removed or reversed, to break the cycle of dependencies.
  - If unhealthy inheritance is present, some functionality needs to be moved, typically from a child class to a parent class.
  - If a modularity violation is identified, the unencapsulated “secret” shared among files needs to be encapsulated.
  - And so forth.



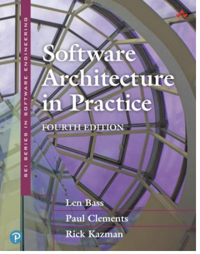
Example – SS1

- This case study, which we call SS1, was done with SoftServe, a multinational software outsourcing company.
- At the time of the analysis, SS1 had 797 source files, and we captured its revision history and issues over a two-year period.
- SS1 was maintained by six full-time developers and many more occasional contributors.



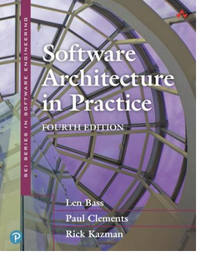
SS1 Analysis

- 2,756 issues were recorded in SS1's Jira issue-tracker (1,079 of which were bugs)
- 3,262 commits were recorded in the Git version control repository.
- Three clusters of files (291 files out of 797 in the project) were identified as containing the most harmful anti-patterns and hence the most debt.
- The number of defects associated with these three clusters covered 89 percent of the project's total defects (265 in a year).



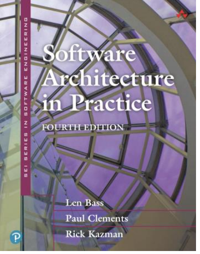
SS1 Analysis

- We calculate the *cost* of these debts in terms of the lines of code committed for bug fixes as follows:
  - The architect estimated the effort required to refactor the three hotspots as 14 person-months.
  - We calculated the average bug fixes per file annually for the project as 0.33.
  - We calculated the average number of annual bug fixes for files in hotspots as 237.8.
  - Based on these results, we estimated that the annual number of bug fixes for the files in the hotspots, after refactoring, would be 96.
  - The difference between the actual churn associated with the hotspot files and the expected amount of churn after refactoring is the expected savings.
- The estimated annual *savings* for the refactored files (using company average productivity numbers) was 41.35 person-months.
- Considering the calculations in steps 1–5, we see that for a cost of 14 person-months, the project can expect to save more than 41 person-months annually.



Tool Support

- This analysis process requires the following tools:
  - A tool to extract a set of issues from an issue tracker
  - A tool to extract a log from a revision control system
  - A tool to reverse-engineer the code base, to determine the syntactic dependencies among files
  - A tool to build DSMs from the extracted information and walk through the DSM looking for the anti-patterns
  - A tool that calculates the debt associated with each hotspot
- The only specialized tools needed for this process are the ones to build the DSM and analyze the DSM. We used DV8 (www.archdia.com).
- Projects likely already have issue tracking systems and revision histories, and plenty of reverse-engineering tools are available, including open source options.



Summary

- Architecture debt is an important and costly form of technical debt. Compared to code-based technical debt, architecture debt is harder to identify because its root causes are distributed among several files and their relationships.
- The process involves gathering information from the project’s issue tracker, its revision control system, and its source code.
- Architecture anti-patterns can be identified and grouped into hotspots, and the impact of these hotspots can be quantified.
- This architecture debt monitoring process can be automated and built into a system’s continuous integration tool suite.
- xOnce architecture debt has been identified, if it is bad enough, it should be removed through refactoring.
- The output of this process provides the quantitative data necessary to make the business case for refactoring to project management.