STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

SSW 322: Software Engineering Design VI

Software Architecture
2020 Spring

Prof. Lu Xiao
lxiao6@stevens.edu

Acknowledge

- https://msdn.microsoft.com/en-us/library/ee658098.aspx

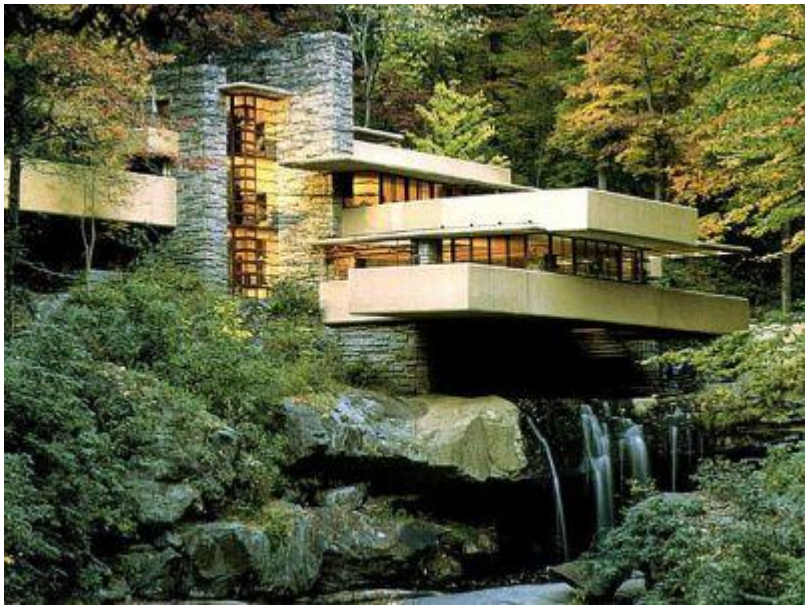
Today's Topic – Software Architecture

- Introduction to software architecture
- Key principles of architecture
- Architecture patterns and styles
- Architecture design representations

- My research in software architecture

What is software architecture?

- Software application architecture is the process of defining a structured solution that meets all of the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability.



Fallingwater—one of the greatest architectural triumphs of the 20th century.

What is software architecture?

"Software architecture encompasses the set of significant decisions about the organization of a software system including the selection of the structural elements and their interfaces by which the system is composed; behavior as specified in collaboration among those elements; composition of these structural and behavioral elements into larger subsystems; and an architectural style that guides this organization. Software architecture also involves functionality, usability, resilience, performance, reuse, comprehensibility, economic and technology constraints, tradeoffs and aesthetic concerns."

---Philippe Kruchten, Grady Booch, Kurt Bittner, and Rich Reitman derived and refined a definition of architecture based on work by Mary Shaw and David Garlan (Shaw and Garlan 1996).

What is software architecture?

“The highest-level breakdown of a system into its parts; the decisions that are hard to change; there are multiple architectures in a system; what is architecturally significant can change over a system's lifetime; and, in the end, architecture boils down to whatever the important stuff is.”

---Martin Fowler outlines some common recurring themes when explaining architecture In *Patterns of Enterprise Application Architecture.*

What is software architecture?

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. Architecture is concerned with the public side of interfaces; private details of elements—details having to do solely with internal implementation—are not architectural."

---Bass, Clements, and Kazman's definition in *Software Architecture in Practice (2nd edition)*,

Why is architecture important?

- Software must be built on a solid foundation: failing to consider key scenarios, failing to design for common problems, or failing to appreciate the long term consequences of key decisions can put your application at risk.
- Modern tools and platforms help to simplify the task of building applications, but they do not replace the need to design your application carefully.
- The risks exposed by poor architecture include software that is unstable, is unable to support existing or future business requirements, or is difficult to deploy or manage in a production environment.

The Architectural Landscape

- Keep in mind that the architecture should:
  - Expose the structure of the system but hide the implementation details.
  - Realize all of the use cases and scenarios.
  - Try to address the requirements of various stakeholders.
  - Handle both functional and quality requirements.

Key Architecture Principles

Build to change instead of building to last. Consider how the application may need to change over time to address new requirements and challenges, and build in the flexibility to support this.

Key Architecture Principles

Model to analyze and reduce risk. Use design tools, modeling systems such as Unified Modeling Language (UML), and visualizations where appropriate to help you capture requirements and architectural and design decisions, and to analyze their impact. However, do not formalize the model to the extent that it suppresses the capability to iterate and adapt the design easily.

Key Architecture Principles

Use models and visualizations as a communication and collaboration tool. Efficient communication of the design, the decisions you make, and ongoing changes to the design, is critical to good architecture. Use models, views, and other visualizations of the architecture to communicate and share your design efficiently with all the stakeholders, and to enable rapid communication of changes to the design.

Key Architecture Principles

Identify key engineering decisions. Use the information in this guide to understand the key engineering decisions and the areas where mistakes are most often made. Invest in getting these key decisions right the first time so that the design is more flexible and less likely to be broken by changes.

Key Architecture Principles

Separation of concerns. Divide your application into distinct features with as little overlap in functionality as possible. The important factor is minimization of interaction points to achieve high cohesion and low coupling. However, separating functionality at the wrong boundaries can result in high coupling and complexity between features even though the contained functionality within a feature does not significantly overlap.

Key Architecture Principles

Single Responsibility principle. Each component or module should be responsible for only a specific feature or functionality, or aggregation of cohesive functionality.

Key Architecture Principles

Principle of Least Knowledge (also known as the Law of Demeter or LoD). A component or object should not know about internal details of other components or objects.

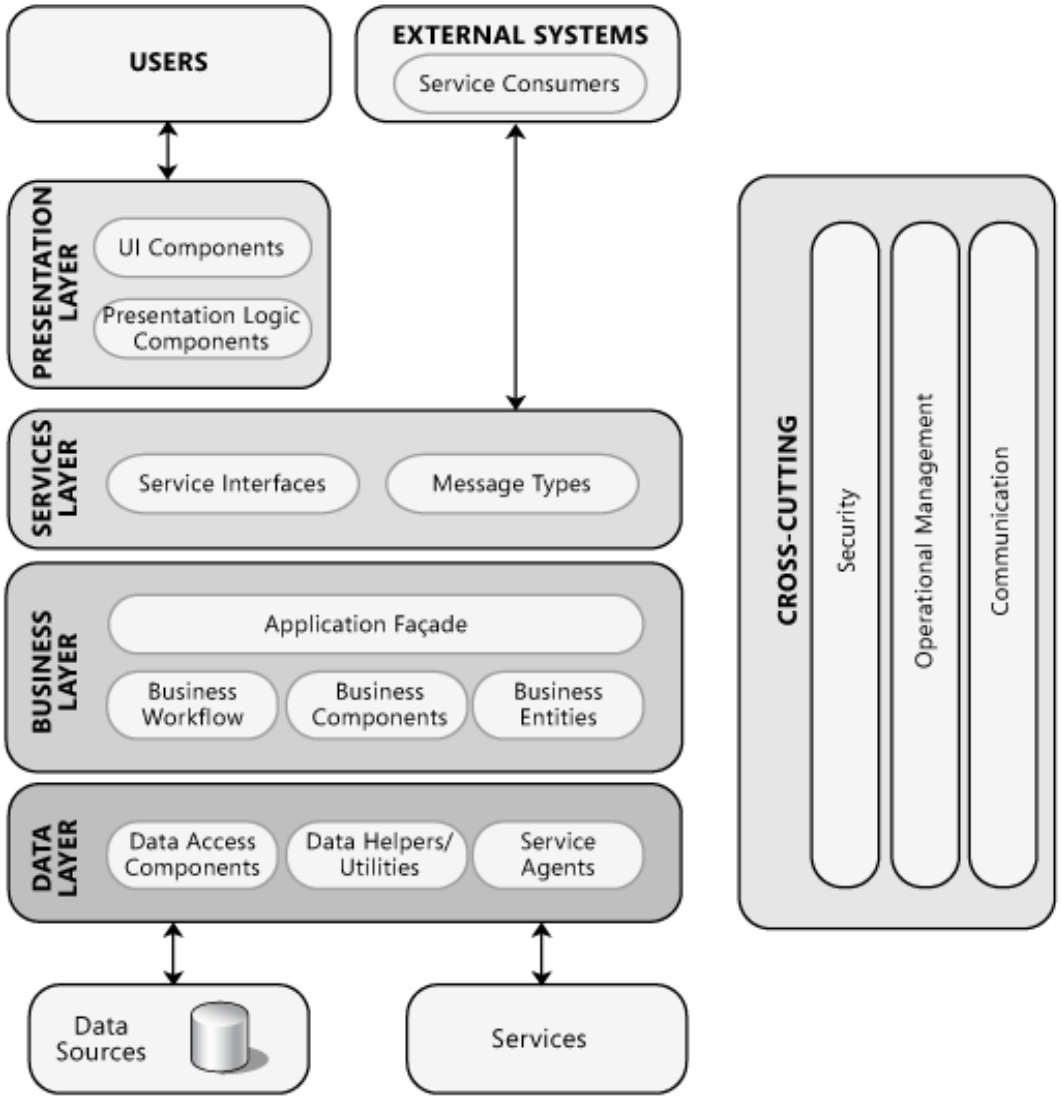
Key Architecture Principles

Don’t repeat yourself (DRY). You should only need to specify intent in one place. For example, in terms of application design, specific functionality should be implemented in only one component; the functionality should not be duplicated in any other component.

Key Architecture Principles

Minimize upfront design. Only design what is necessary. In some cases, you may require upfront comprehensive design and testing if the cost of development or a failure in the design is very high. In other cases, especially for agile development, you can avoid big design upfront (BDUF). If your application requirements are unclear, or if there is a possibility of the design evolving over time, avoid making a large design effort prematurely. This principle is sometimes known as YAGNI ("You ain’t gonna need it").

Common application architecture



- Software architecture is often described as the organization or structure of a system, where the *system* represents a collection of components that accomplish a specific function or set of functions.
- Architecture is focused on organizing components to support specific functionality. This organization of functionality is often referred to as grouping components into "*areas of concern*."

Architecture Key Considerations

- Determine the Application Type
  - Mobile application, client PC, web-based application, etc.
- Determine the Deployment Strategy
  - Physical separation of components across servers
- Determine the Appropriate Technologies
  - Technologies capabilities vs. application requirements
- Determine the Quality Attributes
  - Security, performance, usability, etc.
- Determine the Crosscutting Concerns
  - Logging mechanism, authentication and authorization, etc.

Architectural Styles

- An architectural style, sometimes called an architectural pattern, is a set of principles—a ***coarse grained pattern*** that provides an abstract framework for a family of systems.
- An architectural style improves partitioning and promotes design reuse by providing solutions to frequently recurring problems. You can think of architecture styles and patterns as sets of principles that shape an application.
- They also provide opportunities for conversations that are technology agnostic. This facilitates a higher level of conversation that is inclusive of patterns and principles, without getting into specifics.

Architectural Styles

Architecture style	Description
Client/Server	Segregates the system into two applications, where the client makes requests to the server. In many cases, the server is a database with application logic represented as stored procedures.
Component-Based Architecture	Decomposes application design into reusable functional or logical components that expose well-defined communication interfaces.
Domain Driven Design	An object-oriented architectural style focused on modeling a business domain and defining business objects based on entities within the business domain.
Layered Architecture	Partitions the concerns of the application into stacked groups (layers).
Message Bus	An architecture style that prescribes use of a software system that can receive and send messages using one or more communication channels, so that applications can interact without needing to know specific details about each other.
N-Tier / 3-Tier	Segregates functionality into separate segments in much the same way as the layered style, but with each segment being a tier located on a physically separate computer.
Object-Oriented	A design paradigm based on division of responsibilities for an application or system into individual reusable and self-sufficient objects, each containing the data and the behavior relevant to the object.
Service-Oriented Architecture (SOA)	Refers to applications that expose and consume functionality as a service using contracts and messages.

Architecture design representation

- **4+1**: the logical view, the process, the physical view, and the development view. A fifth view shows the scenarios and use cases for the software.
- **Agile Modeling:** content is more important than representation. The models created are simple and easy to understand, sufficiently accurate, and consistent.
- **IEEE 1471:** a standard formally known as ANSI/IEEE 1471-2000, which enhance the content of an architectural description.
- **Unified Modeling Language (UML)**: three views of a system model. The functional requirements view; the static structural; and the dynamic behavior view.

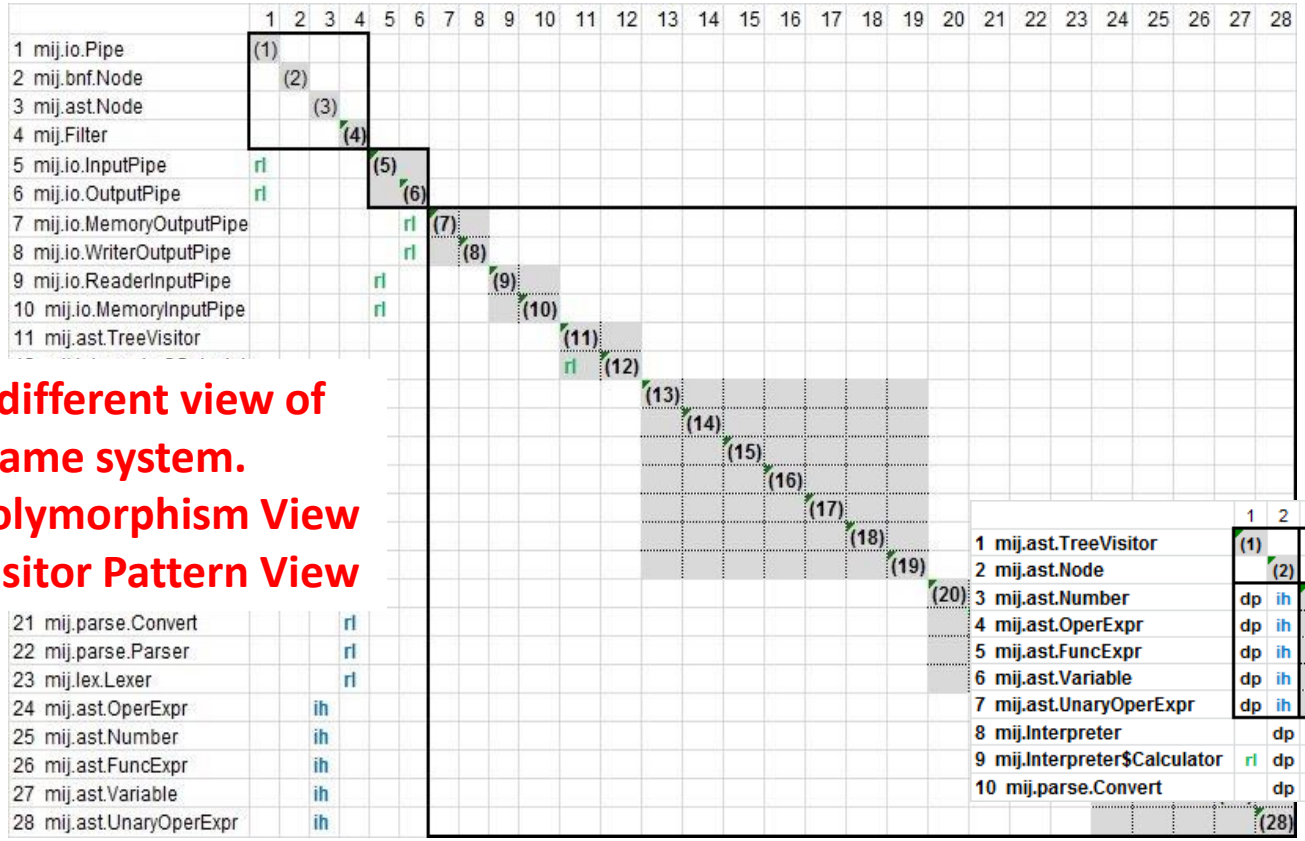
My Research in Software Architecture

My Ph.D. Dissertation

Bridging the Gap between Software Architecture and Maintenance Quality.

Architecture Modeling: Design Rule Spaces

Polymorphism DRSpace/View
rl: realize
ih: inherit



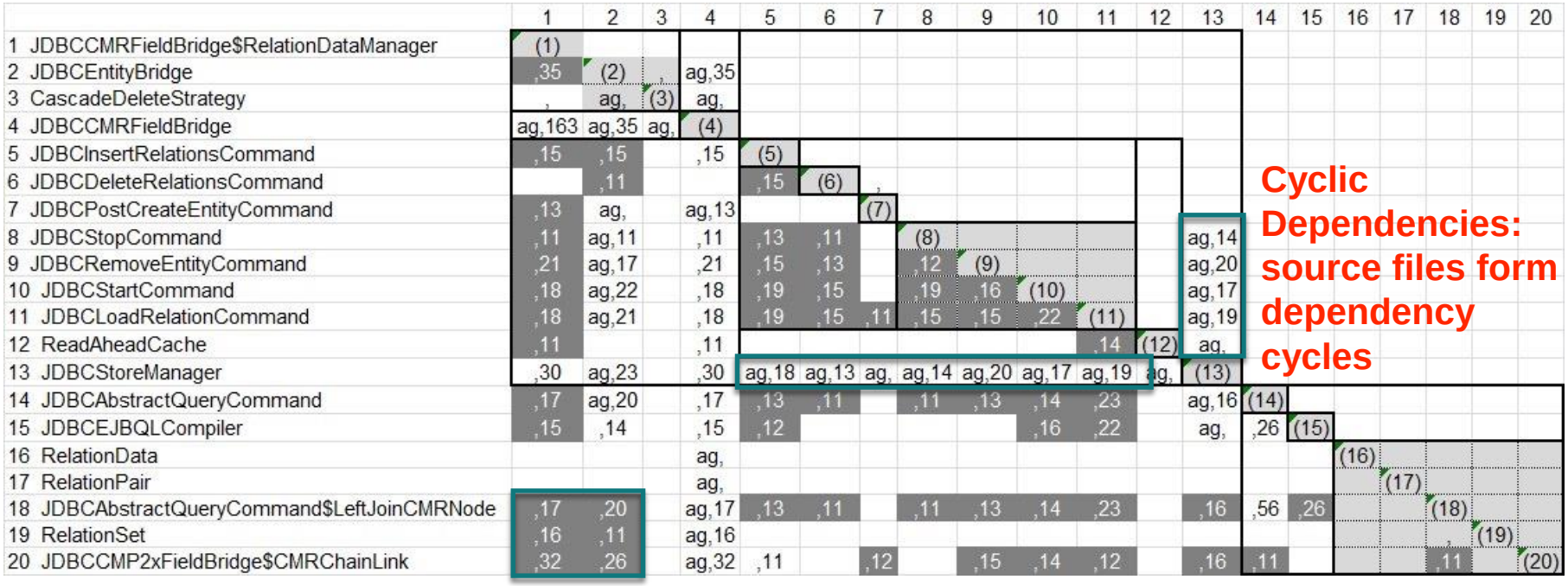
Two different view of the same system.

- **Polymorphism View**
- **Visitor Pattern View**

Visitor Pattern DRSpace/View
rl: realize
ih: inherit

Architecture Modeling: Design Rule Spaces (Cont)

Models the structural dependencies and ***historical couplings*** simultaneously.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1 JDBCCMRFieldBridge$RelationDataManager	(1)																			
2 JDBCEntityBridge	,35	(2)	,	ag,35																
3 CascadeDeleteStrategy	,	ag,	(3)	ag,																
4 JDBCCMRFieldBridge	ag,163	ag,35	ag,	(4)																
5 JDBCInsertRelationsCommand	,15	,15		,15	(5)															
6 JDBCDeleteRelationsCommand		,11			,15	(6)	,													
7 JDBCPostCreateEntityCommand	,13	ag,		ag,13			(7)													
8 JDBCStopCommand	,11	ag,11		,11	,13	,11		(8)					ag,14							
9 JDBCRemoveEntityCommand	,21	ag,17		,21	,15	,13		,12	(9)				ag,20							
10 JDBCStartCommand	,18	ag,22		,18	,19	,15		,19	,16	(10)			ag,17							
11 JDBCLoadRelationCommand	,18	ag,21		,18	,19	,15	,11	,15	,15	,22	(11)		ag,19							
12 ReadAheadCache	,11			,11							,14	(12)	ag,							
13 JDBCStoreManager	,30	ag,23		,30	ag,18	ag,13	ag,	ag,14	ag,20	ag,17	ag,19	ag,	(13)							
14 JDBCAbstractQueryCommand	,17	ag,20		,17	,13	,11		,11	,13	,14	,23		ag,16	(14)						
15 JDBCEJBQLCompiler	,15	,14		,15	,12					,16	,22		ag,	,26	(15)					
16 RelationData				ag,												(16)				
17 RelationPair				ag,													(17)			
18 JDBCAbstractQueryCommand$LeftJoinCMRNode	,17	,20		ag,17	,13	,11		,11	,13	,14	,23		,16	,56	,26			(18)		
19 RelationSet	,16	,11		ag,16														,	(19)	
20 JDBCCMP2xFieldBridge$CMRChainLink	,32	,26		ag,32	,11		,12		,15	,14	,12		,16	,11				,11		(20)

Cyclic Dependencies: source files form dependency cycles

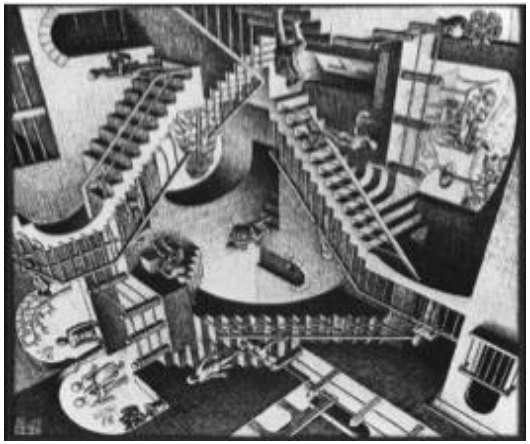
Hadoop DRSpace

Modularity Violations: source files independent from each other frequently coupled in history revisions

The largest 5 spaces usually capture 50% to 95% of all the error-prone files

Software Architectural Debts

- A group of architecturally connected files that incur high maintenance costs over time due to their flawed connections.



The Evolution of an ArchDebt Space (Camel)

		1	2	3	4	5	6	7	8	9	10	11
1	ProcessorDefinition	(1)	dp,	dp,	dp,	dp,	dp,	dp,	dp,dp,	dp,	dp,	dp,
2	LoadBalanceDefinition	Extend,dp,100%	(2)			dp,						
3	ChoiceDefinition	Extend,dp,100%		(3)		dp,			,100%			
4	RollbackDefinition	Extend,dp,100%			(4)	dp,						
5	RouteContext	dp,67%				(5)	,33%	,67%		,33%	,33%	dp,33%

R-2.2.0, 20 Files, +379 lines

R-2.12.4, 28 Files, +1742 lines

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	ProcessorDefinition	(1)	dp,	dp,	dp,	dp,	dp,	dp,	dp,	dp,	dp,	dp,	dp,	dp,		dp,	dp,	dp,	dp,	dp,	dp,
2	ChoiceDefinition	Ext,dp,100%	(2)											dp,			,100%				
3	LoadBalanceDefinition	Ext,dp,100%		(3)										dp,							
4	RollbackDefinition	Ext,dp,100%			(4)									dp,					,33%		
5	OnCompletionDefinition	Ext,dp,67%				(5)		,33%	,33%		,33%	,33%		dp,		,33%				,33%	
6	RouteDefinition	Ext,dp,33%					(6)							dp,					dp,		,33%
7	OnExceptionDefinition	Ext,dp,100%				,100%		(7)	,33%		,33%	,33%		dp,		,100%				,33%	
8	Channel	dp,50%				,50%			(8)	,50%	,50%	,50%		dp,						,50%	dp,
9	DefaultChannel	dp,44%				,33%			Implt,dp,33%	(9)	,33%	,33%		dp,						,33%	dp,
10	ToDefinition	dp,100%			,33%	,100%		,33%	,100%	,100%	(10)	,100%		dp,40%		,40%				,100%	
11	ThreadsDefinition	dp,100%			,33%	,100%		,33%	,100%	,100%	,100%	(11)		dp,40%		,40%				,100%	
12	RecipientListDefinition	dp,100%											(12)	dp,							
13	RouteContext	dp,60%					dp,							(13)		,50%					dp,
14	MarshalDefinition	dp,100%					,50%							dp,40%	(14)	,100%		,100%	,50%		,100%
15	PolicyDefinition	dp,75%												dp,		(15)					
16	TryDefinition	dp,100%	,100%											dp,			(16)				
17	UnmarshalDefinition	dp,100%					,50%							dp,40%	,100%	,100%		(17)	,50%		,100%
18	ErrorHandlerBuilderRef	dp,40%					dp,	dp,						dp,					(18)		
19	MulticastDefinition	dp,100%				50%			50%	50%	50%	50%		dp						(19)	
20	InterceptStrategy																				(20)
25	SamplingDefinition	dp,3																			
26	MulticastDefinition	dp,4																			
27	FinallyDefinition	dp,6																			
28	InterceptStrategy	,50%																			

Each ArchDebt is a refactoring opportunity to reduce the maintenance cost in the long run.

Ongoing Work (1)

- Architectural modeling for addressing collaboration/social issues
  - Past research has been focusing on social-technical congruence [3][4].
  - The misalignments/incongruences may link to deeper problems in architecture design and project governance.

[3] Melvin Conway. How do committees invent? In Datamation, Vol. 14, pages 28–31, 1968.

[4] Marcelo Cataldo, James D. Herbsleb, and Kathleen M. Carley. Socio-technical congruence: A framework for assessing the impact of technical and work dependencies on software development productivity. ESEM 08.

Ongoing Work (2)

- Constructing test-aware architectural modeling for improving test effectiveness/efficiency
  - How software artifacts form independent test modules?
  - How to improve test selection and prioritization based architectural importance?
  - How to maximize test parallelization?
  - How to guide regression testing?

Ongoing Work (3)

- Architecture modeling and analysis for performance bugs
  - How to model software architecture to address performance problems?
  - How improving performance impact other quality aspects?


thank you