Maintaining Architecture-Implementation Alignment

6 JUNE 2023

John Klein

Carnegie Mellon University
Software Engineering Institute

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.

DM23-0602

Outline

Context and Problem Definition

Early Solution – Reflexion Modeling

Automating Nonconformance Detection

State of the Practice

Toward a general solution

Context

Architecture

A system's architecture is defined by its significant design decisions, where in my experience, "significant" is measured by the cost of change.

- Grady Booch

A system's ability to meet its desired (or required) quality attributes is substantially determined by it's architecture. If you remember nothing else from this book, remember that.

- Len Bass, Paul Clements, and Rick Kazman

Grady Booch. Architectural Organizational Patterns. *IEEE Software*, 25(3):18–19, May/June 2008. doi:10.1109/MS.2008.56.
Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley, 4th edition, 2022.

Implementation

Source code is both the end deliverable and the medium in which solutions are expressed. Architecture models are not the end deliverable…"

- George Fairbanks

The code is the truth, but not the whole truth.

- Grady Booch

George Fairbanks, *Just Enough Software Architecture*, Marshall & Brainard, 2010.

Grady Booch. Architectural Organizational Patterns. *IEEE Software*, 25(3):18–19, May/June 2008. doi:10.1109/MS.2008.56.

Depending on Architecture to Deliver Qualities

“We need scalability”

- The architecture uses shared-nothing replicas to allow parallel processing

”We need low latency”

- The architecture pre-computes and caches at each node to avoid remote API calls

Architecture Design Concepts – Building Blocks for Creating Structures

Carnegie Mellon University Software Engineering Institute

Reusable realizations of architecture knowledge

Proven over time

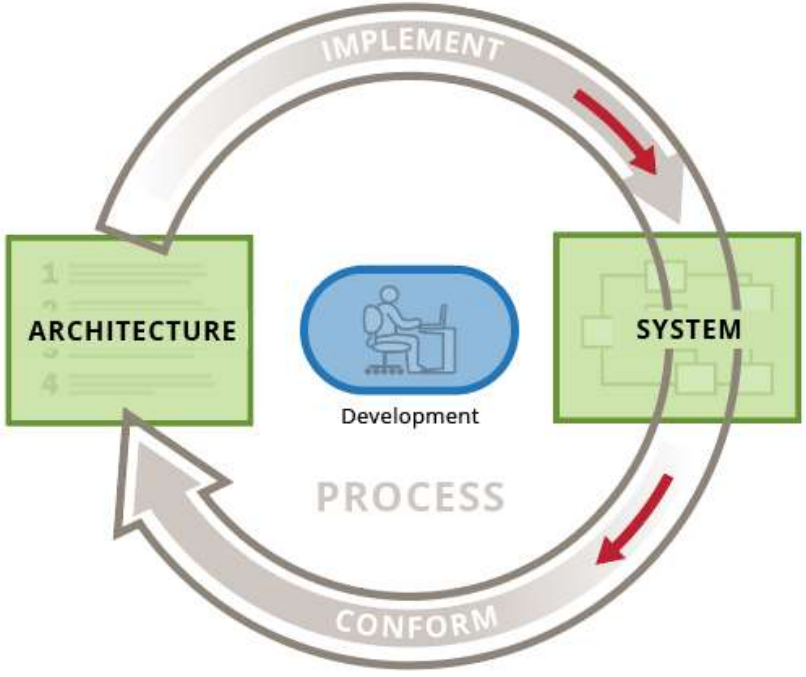
Well-understood properties and tradeoffs

- Reference Architectures
- Architectural Design Patterns
- Deployment Patterns
- Tactics
- Externally-Developed Components

Choosing design concepts is usually one of the first architecture design decisions

Cervantes and Kazman, *Designing Software Architecture*, Addison-Wesley, 2016.

Except…architecture models and not the real deliverable



Architecture can only permit, not guarantee, any quality attribute.

For the implementation to exhibit the quality attributes engineered at the architectural level, it must conform to the architecture.

Context – Expensive, long-lived, acquired systems

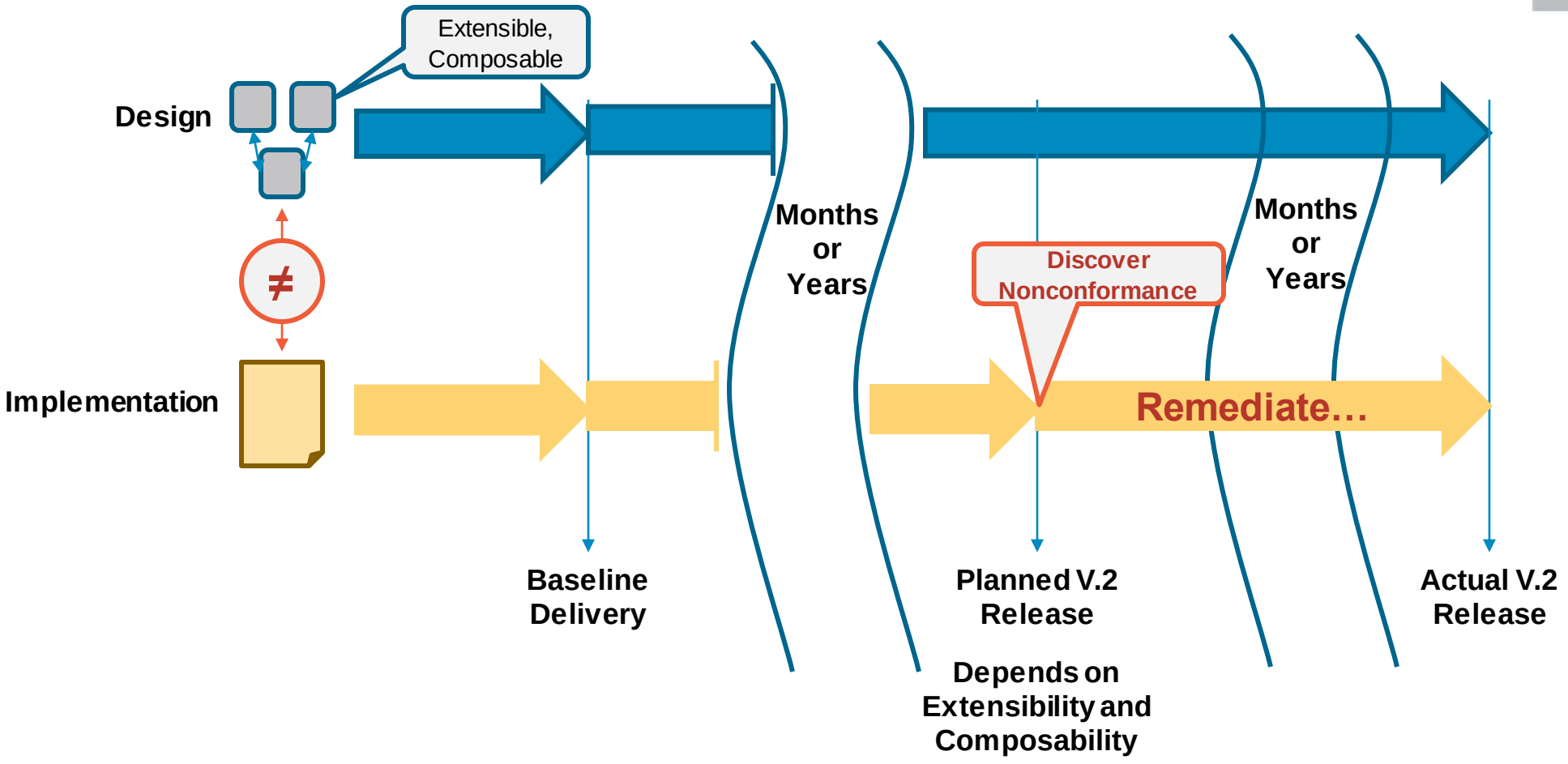
Carnegie Mellon University Software Engineering Institute

Expensive and long-lived systems,
commissioned by organizations,
to be designed and built under contract.

The acquirers are risk-conscious,
and are comfortable with big, up-front design,
but they are learning the value of incremental delivery.

They may not have enough technical expertise to monitor the construction,
and even the smart ones are challenged to understand exactly what is delivered.

Typical Nonconformance Scenario



Conformance Mechanisms in This Context

The team that develops V1 may not be the same team that develops V2

- There may be little incentive to maintain conformance vs. deliver features

Product owners need trusted mechanisms to monitor conformance

Conformance monitoring mechanisms must be

- Affordable and efficient for product owners and developers
- Operate continuously in the development process – detect nonconformance before it becomes expensive to remove

Context – DevSecOps team owns system

Carnegie Mellon University Software Engineering Institute

Systems built by technically-capable organizations,

with product owners and other stakeholders who have the skills to monitor the development.

A DevSecOps team owns the system throughout its lifetime,

and the architect is a fully-engaged member of the team.

Delivery iterations are short,

and architecture-implementation nonconformance is discovered and resolved quickly.

Conformance Mechanisms in This Context

In this context, conformance is still a concern but the mechanism is often more organic

Conformance monitoring is an invisible part of the development process, for example:

- Incremental design reviews at end of iterations
- Pull request reviews consider architecture conformance
- Architects guide development as technical leads
- Architects develop “scaffolding” used by the rest of the team

Conformance monitoring mechanisms that reduce the architect’s workload are valuable

Summary — Context

Investment in architecture is wasted if the implementation doesn’t conform

The mechanisms needed to detect nonconformance depend on the context

Maintaining Architecture-Implementation Alignment

Conformance Checking Approach– Reflexion Modeling

Carnegie Mellon University
Software Engineering Institute

Reflexion Modeling

Introduced by Murphy, Notkin, and Sullivan in 1995

Intuitive approach

- Hypothesize an architecture model for a software system
- Analyze the implementation for agreement with the hypothesis
- Iterate until the the agreement is *close enough* for your needs

Reflexion Modeling as a general approach has remained popular in both researchers and practitioners

> *"Reflexion Modelling is considered one of the more successful approaches to architecture reconciliation. Empirical studies strongly suggest that professional developers involved in real-life industrial projects find the information provided by variants of this approach useful and insightful…"*

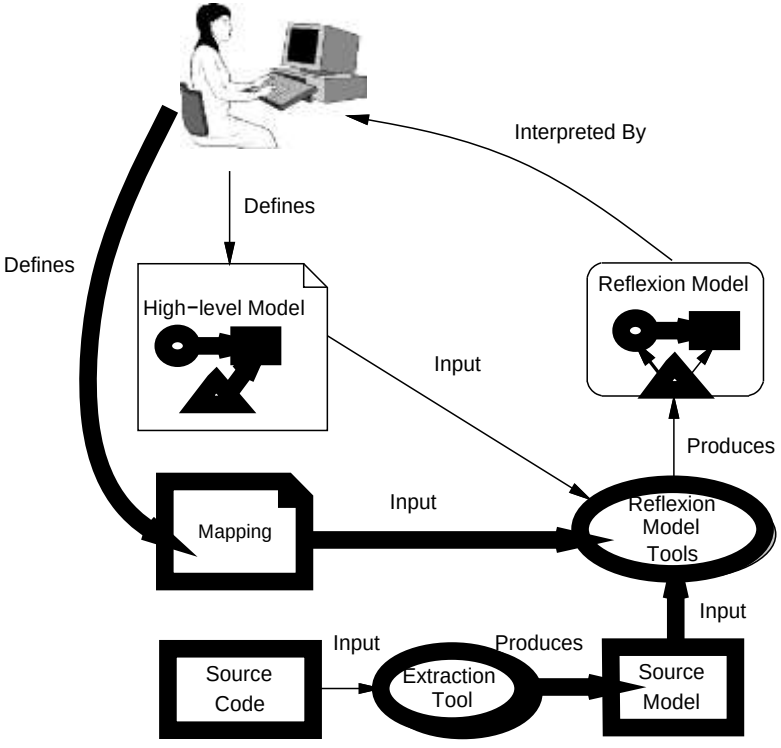
Jim Buckley, Nour Ali, Michael English, Jacek Rosik, Sebastian Herold, Real-Time Reflexion Modelling in architecture reconciliation: A multi case study, *Information and Software Technology*, Vol 61, 2015, Pages 107-123, https://doi.org/10.1016/j.infsof.2015.01.011.

Reflexion Model shows three types of agreement between architecture model and source model

Given implementation *I* and intended architecture *A*

- **convergence** - design elements and relations inferred in *I* are predicted in *A*
- **divergence** - *I* includes elements or relations not predicted in *A*
- **absence** - *I* does not include elements or relations predicted in *A*

Reflexion Modeling



Originally framed as architecture recovery/ reconstruction technology

Supported by a new tool set

Iterative approach

- Define architecture model and mapping to code (manual)
- Extract reflexion model (automated)
- Query model and iterate architecture model and mapping (manual)

Approach scales up

- MS Excel – “77,746 calls between the approximately 15,000 functions”

Gail C. Murphy, David Notkin, and Kevin Sullivan. Software reflexion models: Bridging the gap between source and high-level models. In Proc. 3rd ACM SIGSOFT Symp. on the Foundations of Software Eng., pages 18–28, 1995. doi:10.1145/222132.222136.

Summary — Reflexion Modeling

Reflexion Modelling is considered one of the more successful approaches to architecture reconciliation. Empirical studies strongly suggest that professional developers involved in real-life industrial projects find the information provided by variants of this approach useful and insightful, ***but the degree to which it resolves architecture conformance issues is still unclear***

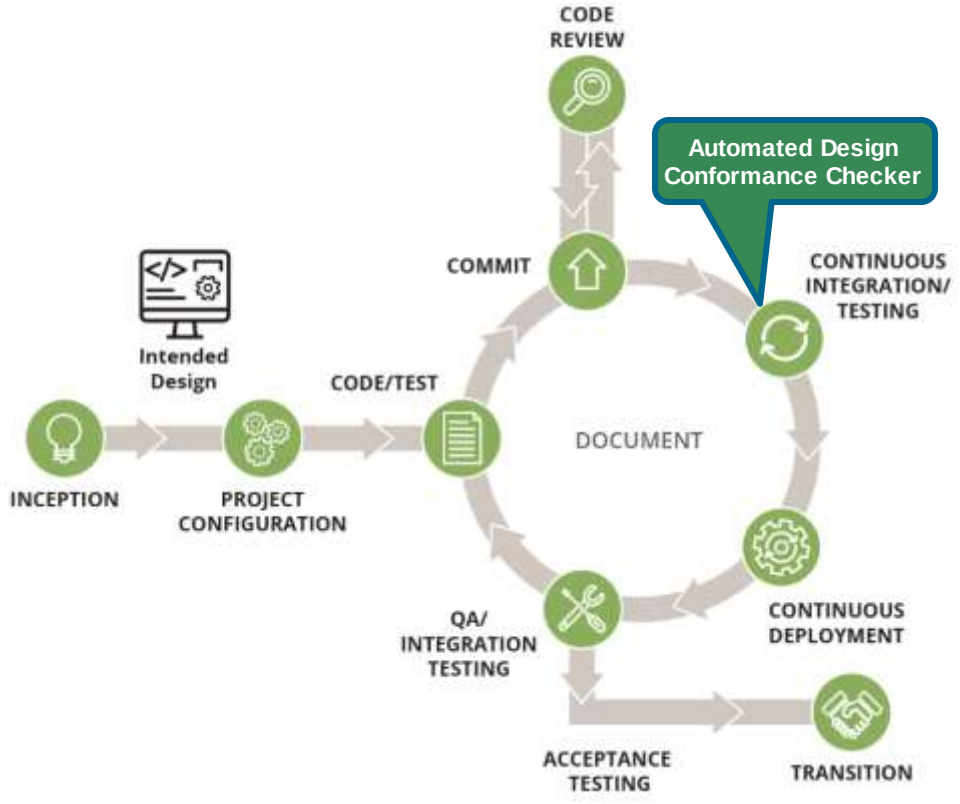
Jim Buckley, Nour Ali, Michael English, Jacek Rosik, Sebastian Herold, Real-Time Reflexion Modelling in architecture reconciliation: A multi case study, *Information and Software Technology*, Vol 61, 2015, Pages 107-123, https://doi.org/10.1016/j.infsof.2015.01.011.

Maintaining Architecture-Implementation Alignment

Automating Nonconformance Detection

Carnegie Mellon University
Software Engineering Institute

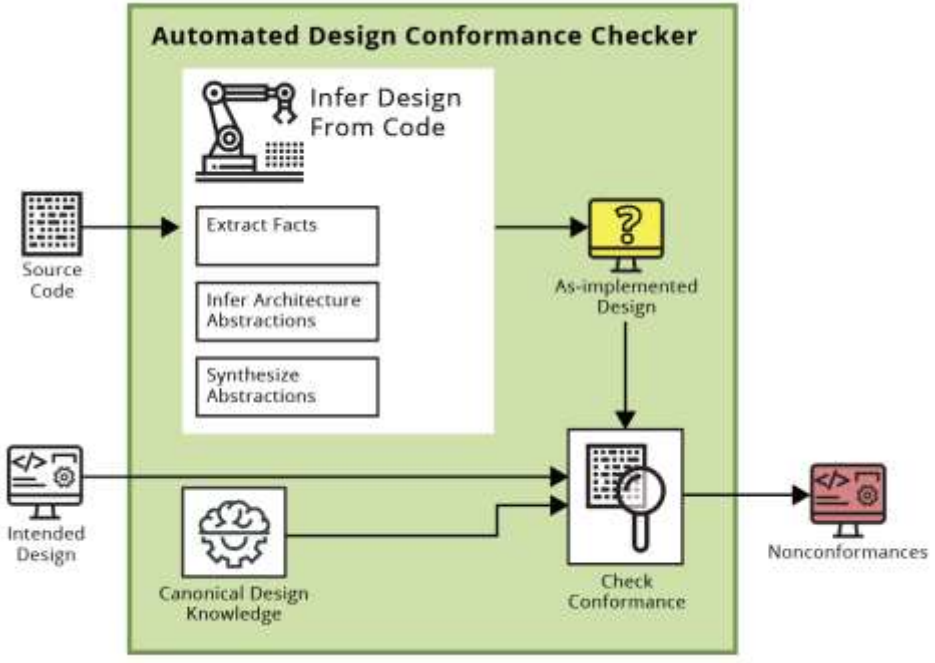
Improve Conformance Between Implementation and Architecture



An automated checker integrated into a continuous integration (CI) workflow

- exposes nonconformance at time of commit instead of months later
- promotes conversation whether code or architecture needs to change
- allows remediation before violations become fixed in the implementation
- enables program managers to hold developers accountable

Goal – Automatically Infer Design from Code



Need to bridge the model-code gap

Essential challenges to inferring design constructs

- imprecise definitions of abstractions
- variation in implementation
- limits of fact gathering analyses
- alignment or correlation of extracted design to intended design

James Ivers, Ipek Ozkaya, and Robert L. Nord. Can AI close the design-code abstraction gap? In *34th IEEE/ACM Int. Conf. on Automated Software Eng. Workshop (ASEW)*, pages 122–125, 2019. doi:10.1109/ASEW.2019.00041.

There is a Conceptual Gap Between an Architecture Model and the Code that Implements It

Dimension	Architecture Model	Code
Vocabulary	modules and components	classes and functions
Abstraction	Models omit details that are not needed to fulfill the purpose of the model	Contains all details needed to execute
Design Commitments	“use a message queue”	“use Kafka”
Intensional/ Extensional	Often intensional, e.g., “Filters communicate only using pipes”	Only extensional, e.g., “Component F1 is a filter that communicates to Component F2, which is also a filter, using pipe P1.”

George Fairbanks, *Just Enough Software Architecture*, Marshall & Brainard, 2010. Chapter 10.

Conformance Query

Does implementation I conform to architecture A?

Three types of agreement:

- **convergence** - design elements and relations inferred in *I* are predicted in *A*
- **divergence** - *I* includes elements or relations not predicted in *A*
- **absence** - *I* does not include elements or relations predicted in *A*

Conformance Versus Reconstruction

(Conformance Query)

Does *I* agree with *A* ?

≠

What is the architecture of *I* ?

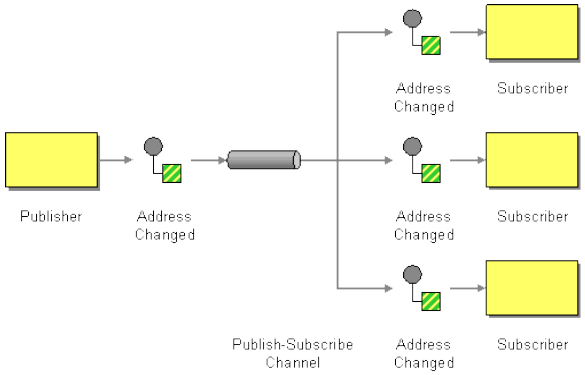
(Architecture Reconstruction)

Answering a Conformance Query – Inferring Implemented Design from Code

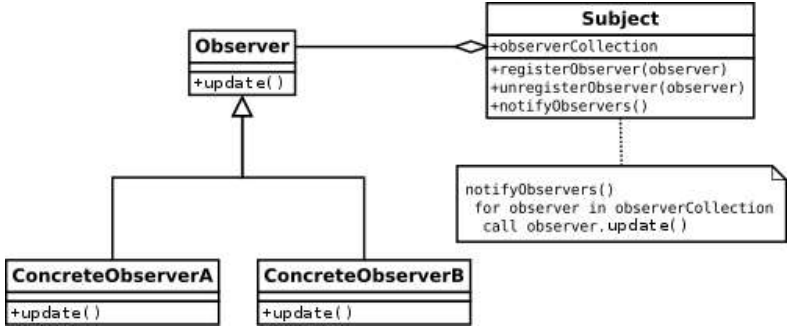
Imprecise definition of architecture abstractions	The architecture communities of research and practice both operate on imprecise definitions — variation in interpretation makes generalization challenging
Multiple sources of variation in how architecture abstractions are implemented	Developers within and across projects implement the same abstraction many different ways
Variation in and limits of fact extraction analyses	Different architecture views rely on different categories of facts, which in turn rely on different forms of analysis, many of which have known limitations
Relating an intended architecture to an inferred design	Aligning inferred structures to intended structures

Imprecise definition of architecture abstractions

Publish-Subscribe Pattern



Hohpe & Woolf
(https://www.enterpriseintegrationpatterns.com/patterns/messaging/PublishSubscribeChannel.html)



Gamma, Helm, Johnson, Vlissides
(https://en.wikipedia.org/wiki/Observer_pattern)

What does “Publish-Subscribe Pattern” mean?

Carnegie Mellon University Software Engineering Institute

Commonality

- Element Types: Publisher, Subscriber
- Basic Behavior:
  - Publisher components write units of information
  - The writer doesn’t know who (if anyone) will eventually receive the information, or when it will be received
  - Subscriber components can receive published information

Variability

- Element Types: Infrastructure?
- Behavior, including
  - Must a publisher register with the infrastructure before publishing an event?
  - Must events be explicitly labeled with a topic name?
  - Can subscribers select the events of interest by topic name or by event content?
  - Does a subscriber receive only events that were published after its subscription started, or can it receive events published before its subscription started?

Sources of Implementation Variation

Programming Language Paradigms

- Object-oriented, e.g., C++ and Java — classes and inheritance
- Not OO, e.g., C — functions
- Procedural or OO, e.g., Python
- Dynamic, e.g., Ruby — bind code to function names during execution

Coding or programming idioms

- E.g., represent pub-sub topic as literal, constant reference, or variable value

Frameworks and libraries

Distribution of responsibilities across code units

- Implementation of an abstraction may span code units
- Need to correlate facts across code units to reason about the abstraction

Fact extraction analysis

Carnegie Mellon University Software Engineering Institute

Module structures

Static analysis tools, e.g., Understand, Structure 101, Lattix

- Names and types
- Write, read, call or invoke, generalize/specialize

Limitations

- No semantics, e.g., module or layer, just relationships among chunks of code
- Unable to detect relationships when runtime binding is used

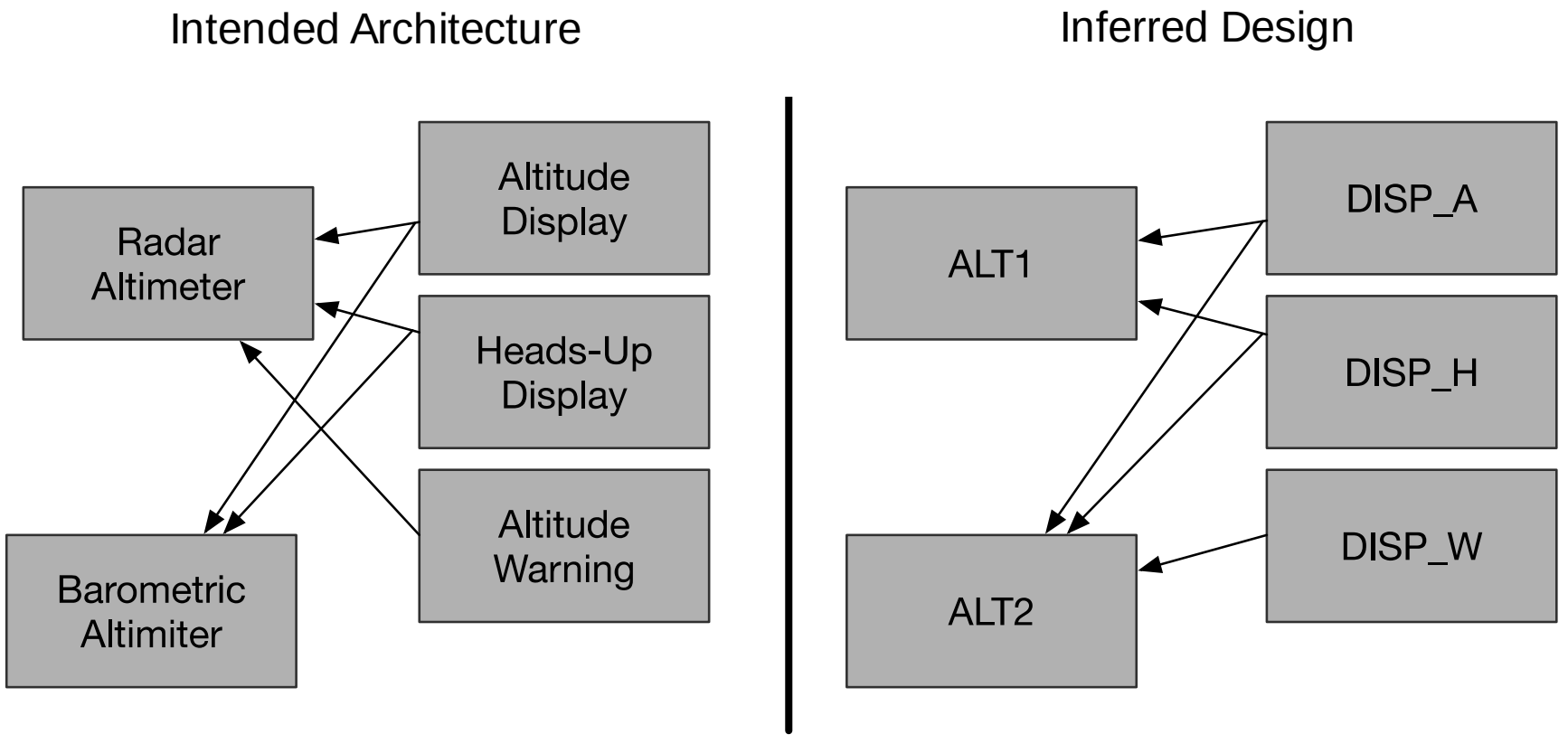
Component and Connector structures

Static analysis tools are limited

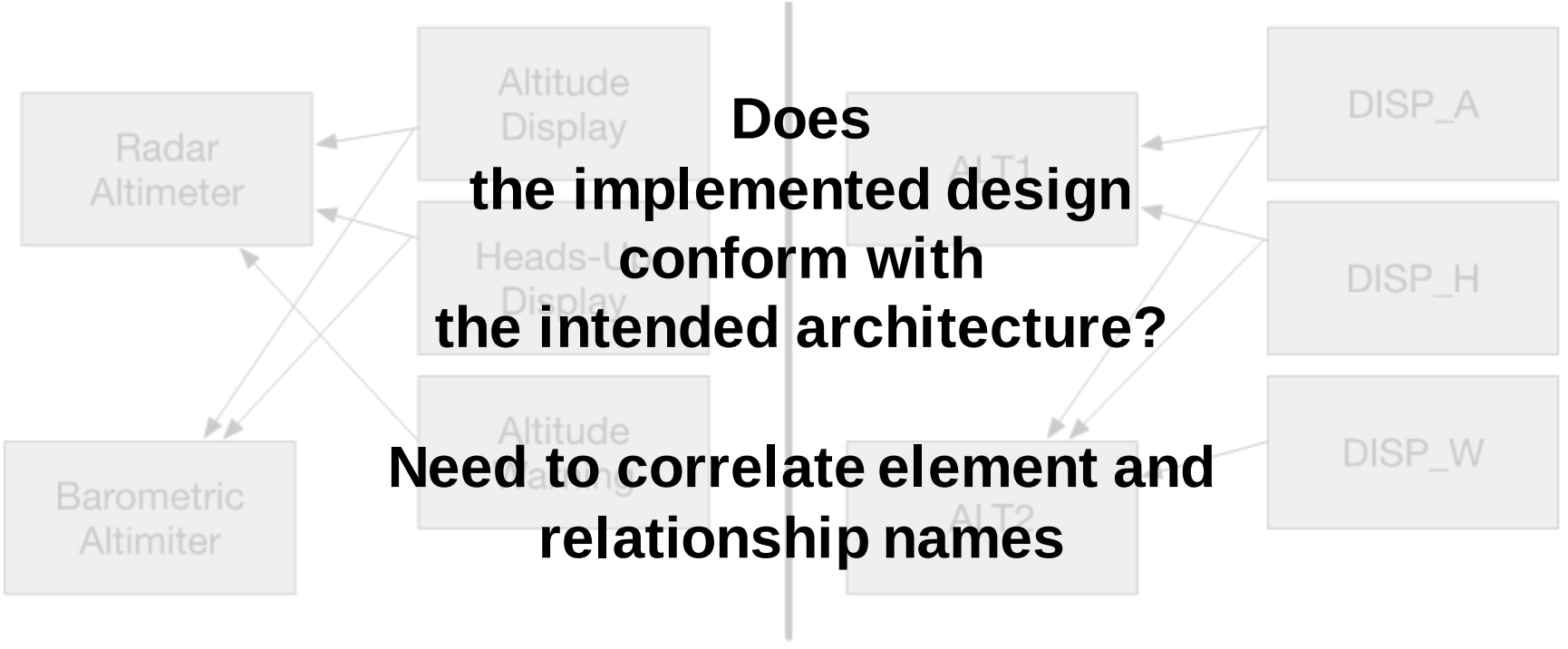
- Code represents classes, relationships are between executing instances
- Runtime instance creation and naming
- Multiple names can point to the same instance — aliasing

May need dynamic code analysis or trace analysis

Relating an Intended Architecture to an Inferred Design



Relating an Intended Architecture to an Inferred Design



Bridging the Model-Code Gap — (Some) Solutions and Tradeoffs

Some things that we tried while working to solve this challenge:

1. Define A New Style/Pattern-specific Language
2. Tailor Design Inference to a Specific Implementation Framework
3. Use Multiple Tools to Extract Facts
4. Focus on fragments of design and implementation

Bridging the Model-Code Gap — New Architecture Languages

Define A New Style/Pattern-specific Language — provide formal semantics to represent part of an architecture that uses a particular style or pattern

Tradeoff — architect productivity vs. breadth of representable style variations

- *Opinionated* language that with limited variability – easy to learn but can't represent everything
- Rich language that can represent anything but is hard to learn and use

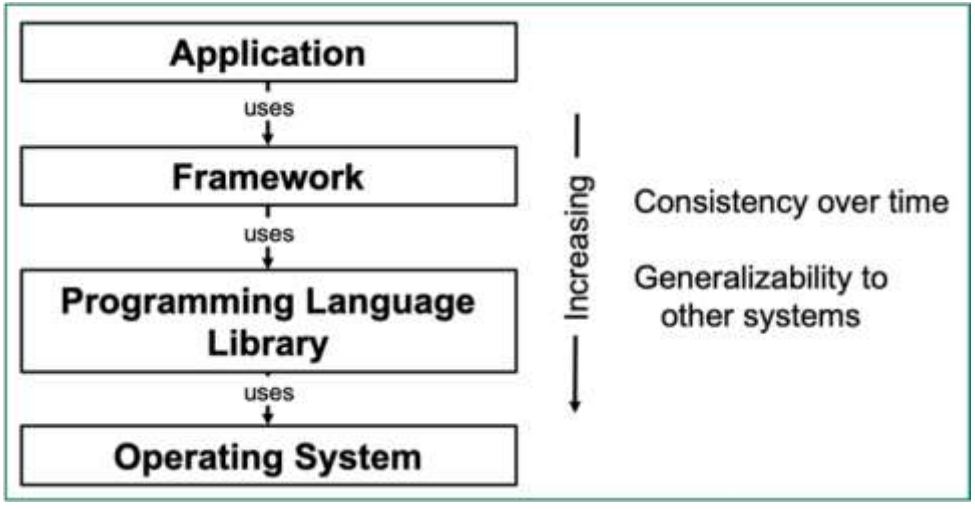
Bridging the Model-Code Gap — Tailor Design Inference to a Specific Implementation Framework

Framework — a reusable software component that addresses recurring concerns across a range of applications

Elements of most frameworks map to key design abstractions

Framework elements are already present in source code — developers don't need to to extra work like adding annotations to source code

Reduce variation due to coding or programming idioms — more efficient fact extraction



Using Framework Information to Infer Design Concepts — Publish-Subscribe Example

Design Concept	Framework		
	Qt	***FACE***	***ROS***
topic	signal (a method)	constant	constant
publish	`connect(obj1,signal,obj2,slot)`	`Create_Connection(name,pattern,dir,id)`	`NodeHandle::advertise<msg_type>(topic)`
update	call to signal	`Send_Message(id, data)`	call to publish
subscribe	`connect(obj1,signal,obj2,slot)`	`Create_Connection(name,pattern,dir,id)`	`NodeHandle::subscribe(topic)`
reflect	*not explicit in code*	`Receive_Message(id, data)`	*not explicit in code*

Note: Qt, FACE, and ROS are three frameworks appeared in our data sets.

Bridging the Model-Code Gap — Use Multiple Tools to Extract Facts

Use tools like SciTools Understand to quickly extract dependency relationships among data types

- Can infer that class `Publisher` publishes a `Message`
- Will not detect that `Controller` publishes `StartMessage`

Add other tools, e.g., Clang

- Add facts about literal and constant names and values

```
1   class Message {};
2   class StartMessage: public Message  {};
3   ...
4
5   class Publisher {
6       ...
7       void publish(Message m);
8       ...
9   };
10
11  class Controller: public Publisher {
12      ...
13      sm = new StartMessage;
14      publish(sm);
15      ...
16  }
```

Argument type ambiguity – Lines 7, 14

Focus on fragments of design and implementation

Don’t represent the entire intended architecture — focus on the structures needed to achieve the most important quality attributes

Limits the scope of the conformance query, simplifies other challenges like name correlation

Summary — Automating Nonconformance Detection

Model-Code Gap

Essential challenges to inferring design constructs

- imprecise definitions of abstractions
- variation in implementation
- limits of fact gathering analyses
- alignment or correlation of extracted design fragment to intended design fragment

Possible Solutions:

- Define A New Style/Pattern-specific Language
- Tailor Design Inference to a Specific Implementation Framework
- Use Multiple Tools to Extract Facts
- Focus on fragments of design and implementation

Maintaining Architecture-Implementation Alignment

State of the Practice for Automated Conformance Checking

Carnegie Mellon University
Software Engineering Institute

State of the Art in Conformance Checking

Tools based on static analysis

- E.g., Understand™, Structure101, Lattix

Code quality metrics are weakly correlated to nonconformance, but may be indicative

Dependency structure analysis can detect nonconformance for some architecture styles, e.g., Layers

- Requires mapping from code chunks to architecture elements like modules or layers

Effective use requires tuning of out-of-the-box configuration

https://www.scitools.com
https://structure101.com
https://www.lattix.com

System-specific ad hoc tests

- E.g., fitness functions

Implement using unit test framework like CppUnit or Junit

Tests could be developed to check conformance to any architecture style

Can post-process static analysis tool output

As the architecture and/or the implementation evolves, the tests related to those changes may become brittle over time and need to be maintained

Neal Ford, Mark Richards, Pramod Sadalage, and Zhamak Dehghani. *Software Architecture: The Hard Parts*. O'Reilly Media, 2021.

Steps You Can Take Today to Improve Conformance

Capture (parts of) the intended design – structures and observable principles that allow the system to satisfy the high priority quality attribute requirements

Adopt (and refactor to) an *architecturally-evident coding style* to bridge the model-code gap in the parts of the intended design that you have captured

Check conformance in the key parts of your intended architecture at commit time or incremental release or major release or gate review…but do it!

- Manual checks – in peer reviews
- Automated checks – tailor rules in off-the-shelf tools and/or ad hoc tests

John Klein and Robert L. Nord. Why architecture conformance matters for evolvable systems. *CrossTalk,* pages 19–24, July 2022. https://community.apan.org/wg/crosstalk/m/documents/420393

George Fairbanks. *Just Enough Software Architecture: A Risk-Driven Approach*. Marshall & Brainerd, 2010. (Ch. 10 – Arch-evident coding style)

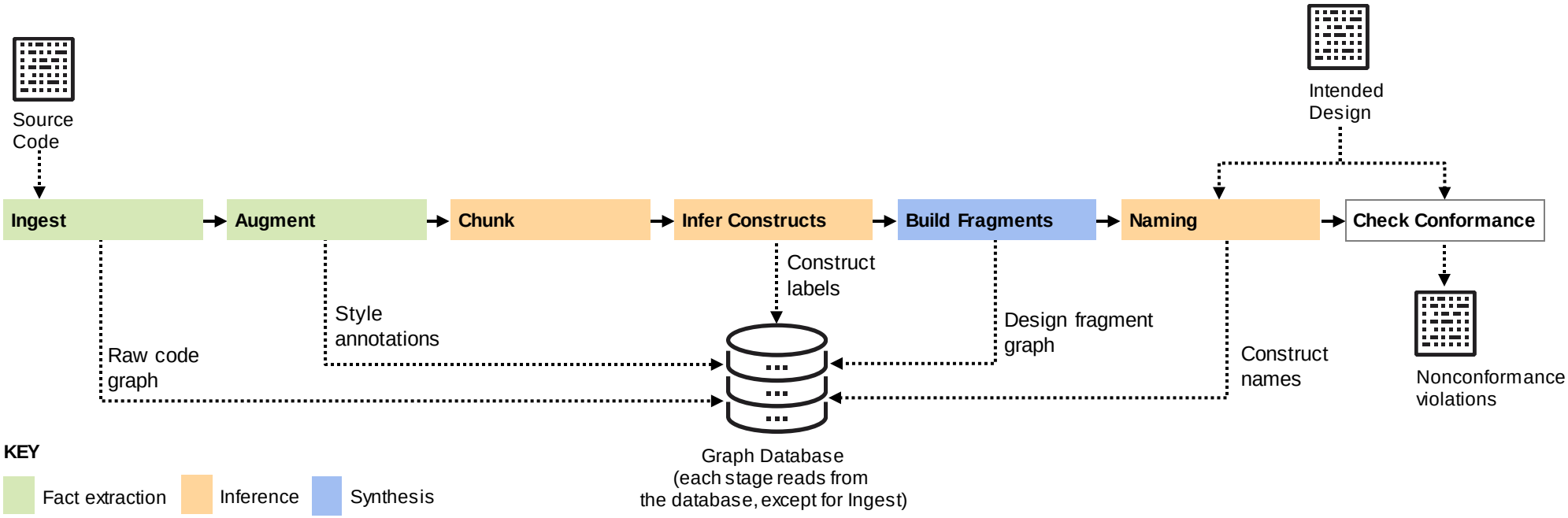
Maintaining Architecture-Implementation Alignment

Research Prototype of an Automated Conformance Checker

Carnegie Mellon University
Software Engineering Institute

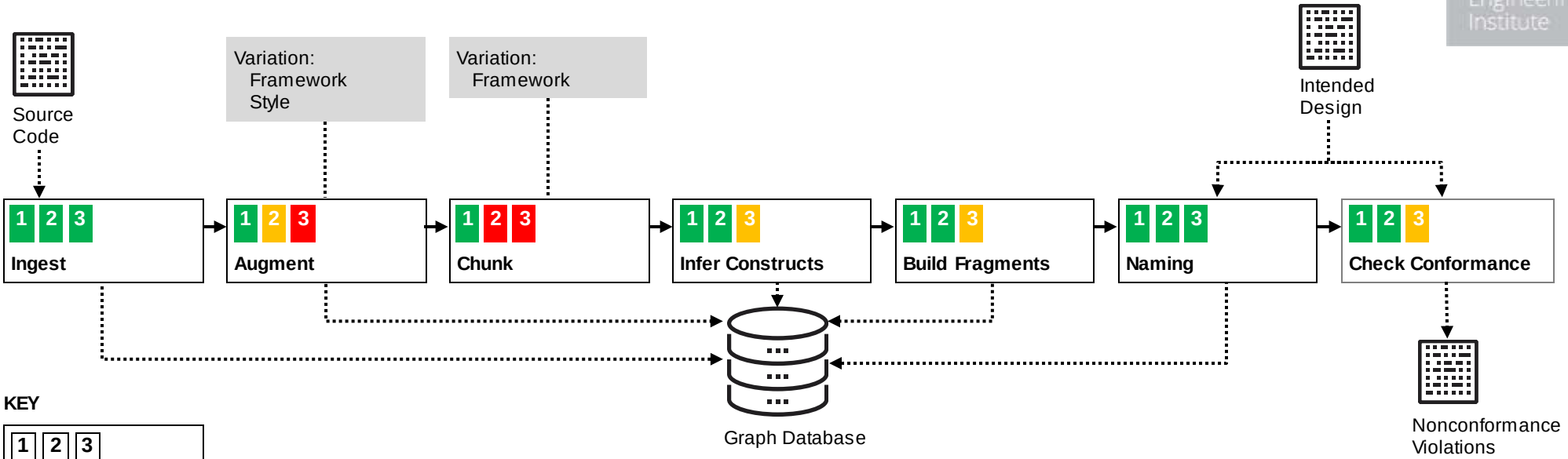
Towards Generalized Automation: Prototype Conformance Checker

Developed at SEI in 2022



https://www.youtube.com/watch?v=6svndCKuaSw

What Is Involved in Applying the Approach



We have learned how to **customize** the approach for a particular **framework-**based system and architecture **communication style**.

What Practical Problems Does the Approach Solve?

	State of Practice		Conformance Checker Prototype
	Code Quality	**Architecture Quality**	
Design concepts	Classes, packages, files	Modules, dependencies	Architecture **communication styles**
Bridging code and design	Logical and physical element composition	Dependency clusters (semi-automated)	**Automated** rules (framework-based systems)
Conformance	ISO standards, maintainability	Modularity, dependencies, design paradigms	**Intended** architecture and canonical design knowledge

Conformance checking is **feasible** today using a **rules-based** approach to extract design information from **framework-based** systems.

The approach recovers a broader range of architecture views and supports checking a broader range of criteria for conformance.

Challenges to Conducting Research in Architecture ~~Conformance Checking~~

Finding suitable examples in open source software projects

- Too simple, too specific (e.g., ROS projects), fragments

Buildability of open source software projects

- Analysis generally requires building (compiling and linking) the source code.
- Challenges included lack of documentation, tools, environment, package dependencies.

Lack of architecture documentation

- No documentation to use as basis for intended design specification.
- Both in open source and commercial projects

Existing research data sets are not applicable

- Limited labeled data for ML-based approaches

Summary — Research Prototype of an Automated Conformance Checker

Prototype demonstrated feasibility and some generalizability

Heuristic-based approach

Multi-tool fact extraction

Variation points:

- Architecture pattern/style
- Framework

Open challenges remain

Final Take-aways

- Investment in architecture is wasted if the implementation doesn't conform
- The type of mechanisms needed to detect nonconformance depend on the context
- Reflexion Modeling is one mature approach, but doesn't solve the whole problem
- There are essential challenges to bridging the model-code gap to infer design from code
- Static analysis tools and ad hoc checks can provide some value today
- Automation using heuristic-based approaches may be cost-effective

SEI Conformance Checker Prototype Project Team

Robert Nord
Principal Member of the Technical Staff (Ret.),
CMU / SEI

James Ivers
Principal Engineer,
CMU / SEI

Lena Pons
Software Architecture and AI Researcher,
CMU / SEI

Chris Seifried
Associate Engineer,
CMU / SEI

Josh Fallon
Defense Network Analyst,
CMU / SEI

John Klein
Principal Member of the Technical Staff,
CMU / SEI