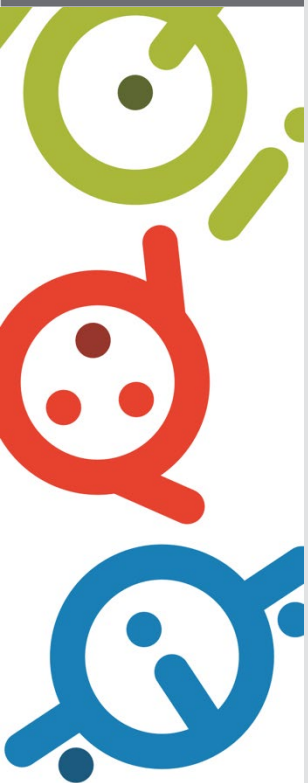
Carnegie Mellon University
Software Engineering Institute



Research Review 2021

Automated Design Conformance during Continuous Integration

November 2021

Robert Nord, Lena Pons

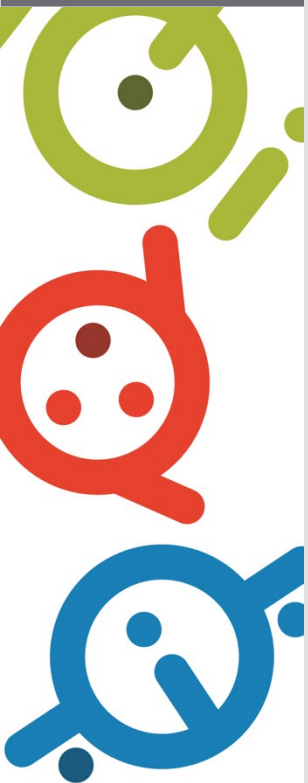
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Document Markings

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.

DM21-0807

Carnegie Mellon University
Software Engineering Institute

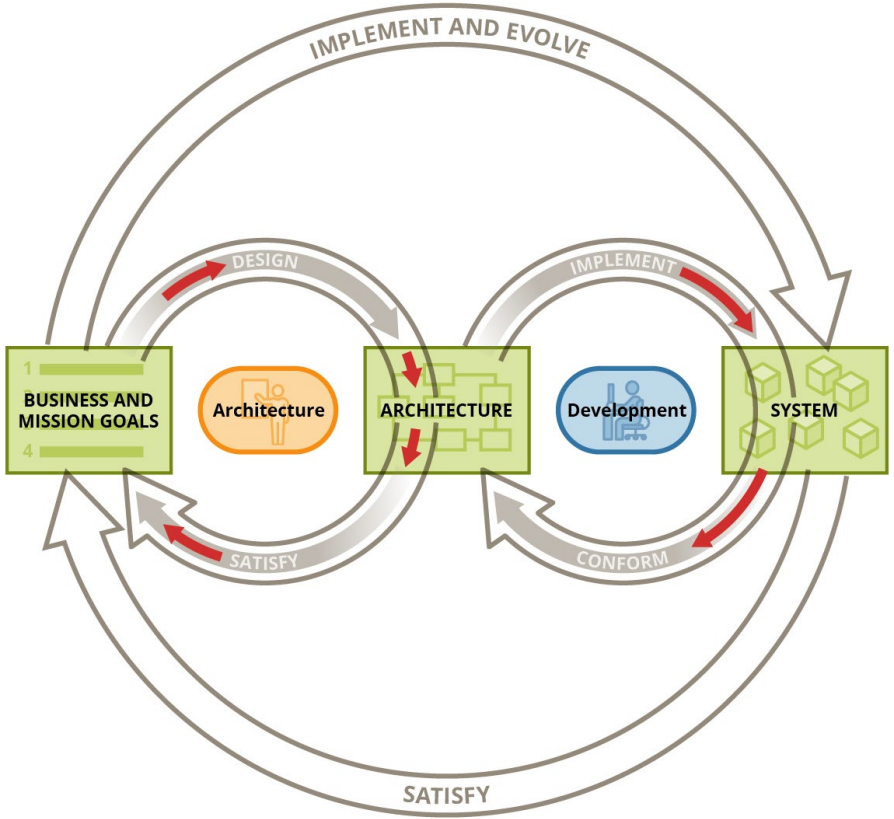


Research Review 2021

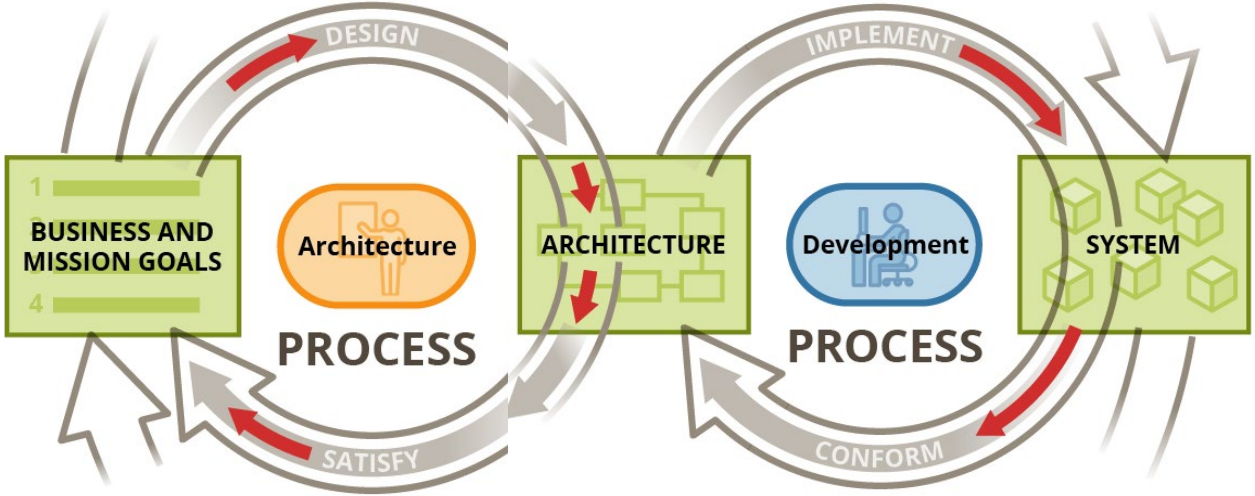
Automated Design Conformance during Continuous Integration

Software Nonconformance

Software Architecture Enables Our Ability to Innovate



Software Architecture Enables Our Ability to Innovate



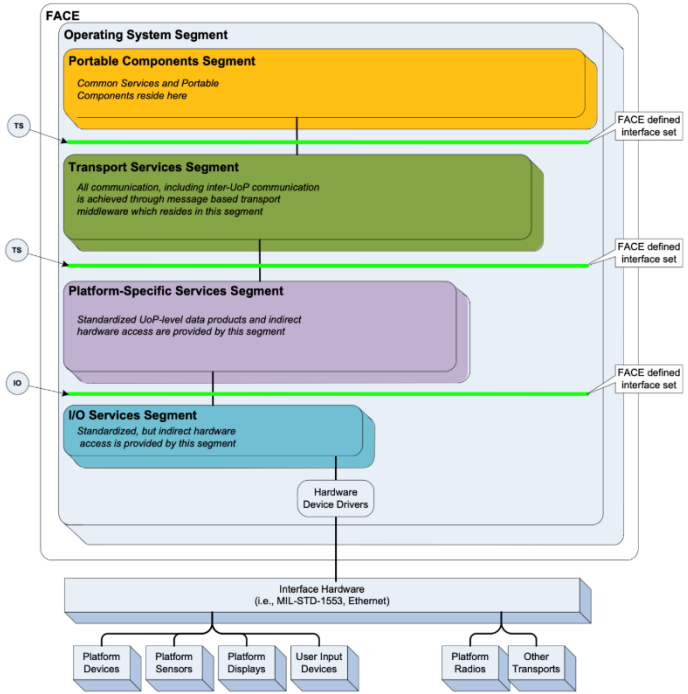
Software architecture is an abstraction that helps organizations satisfy business and mission goals.

The community has evolved a body of knowledge in the form of architecture styles that guides design and analysis.

The degree to which a system meets its goals is dependent on architectural decisions.

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

Challenges in Software Conformance



The Open Group (2017). **Reference Architecture,** *FACE (Future Airborne Capability Environment) Technical Standard, Edition 3.0*.

Modular Open Systems Approach (MOSA)

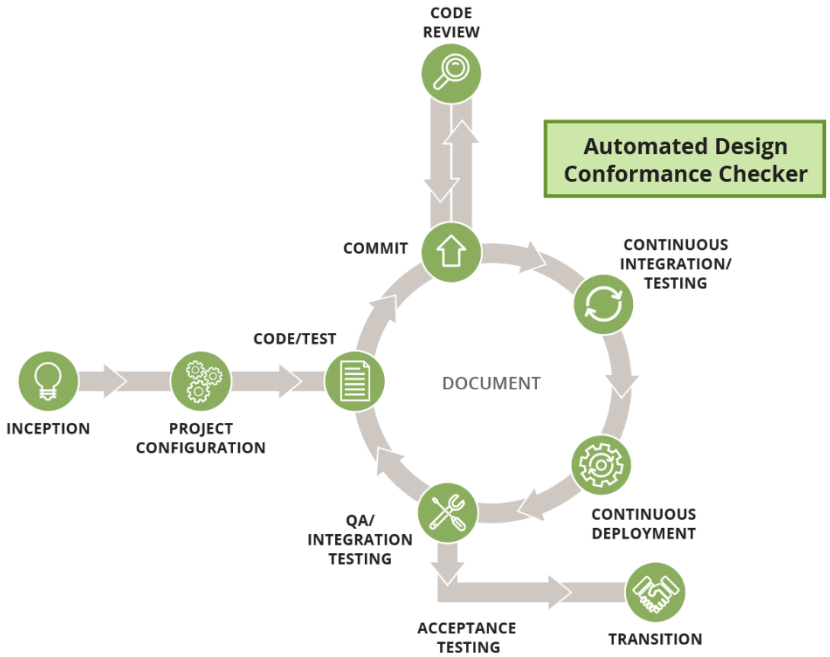
- technical and business strategy
- affordable and adaptable systems

FACE Technical Standard

- conformance verification matrix
  - 487 items
  - 194 are inspection of design
- component-level standard

FACE-compliant systems may encounter integration problems.

Automated Design Conformance during CI

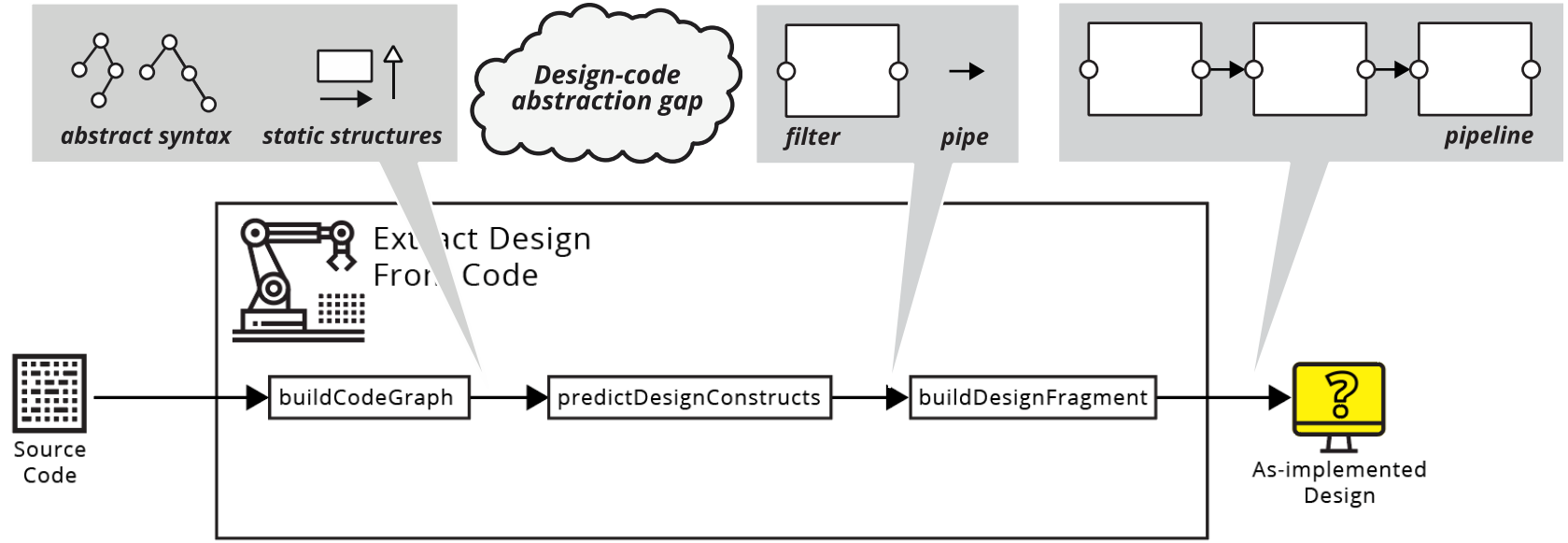


An automated design conformance checker integrated into a continuous integration (CI) workflow will reduce time to detect violations.

Automation enables early detection and allows remediation before the violation becomes a fixed feature of the implementation.

Detection of nonconformances allows program managers to hold developers (contractor or organic) accountable.

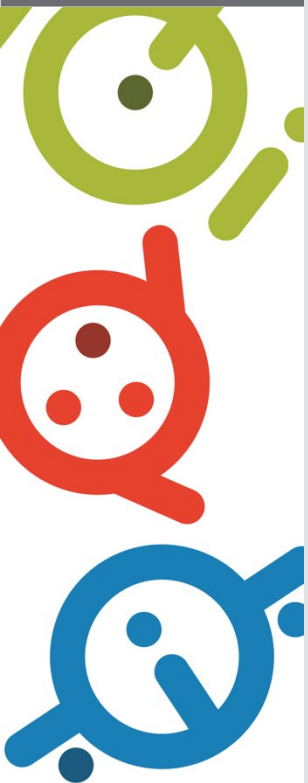
Extract Design From Code



We are motivated to create a new generation of automation for architects that helps bridge the gap between architecture abstractions and code.

Ivers et al. (2019). **Can AI Close the Design-Code Abstraction Gap?** *International Workshop on Software Engineering Intelligence, IEEE/ACM International Conference on Automated Software Engineering (ASE).*

Carnegie Mellon University
Software Engineering Institute

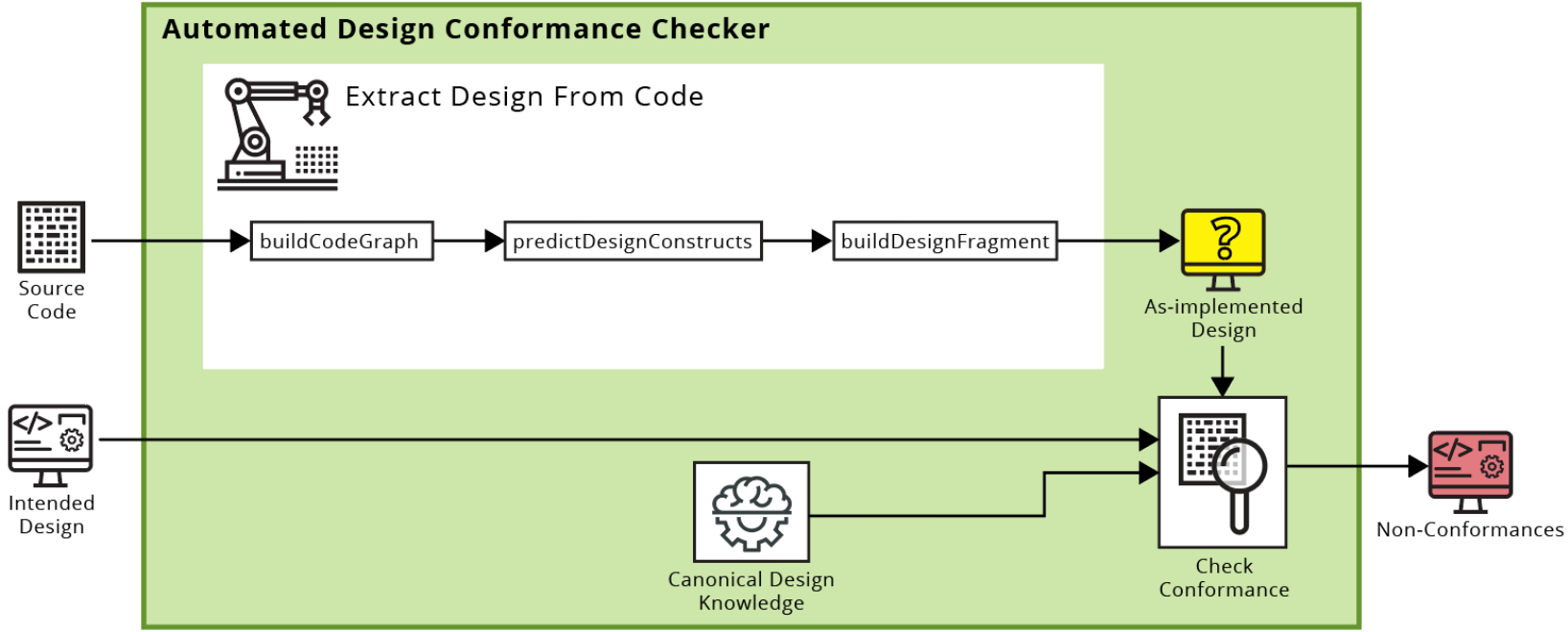


Research Review 2021

Automated Design Conformance during Continuous Integration

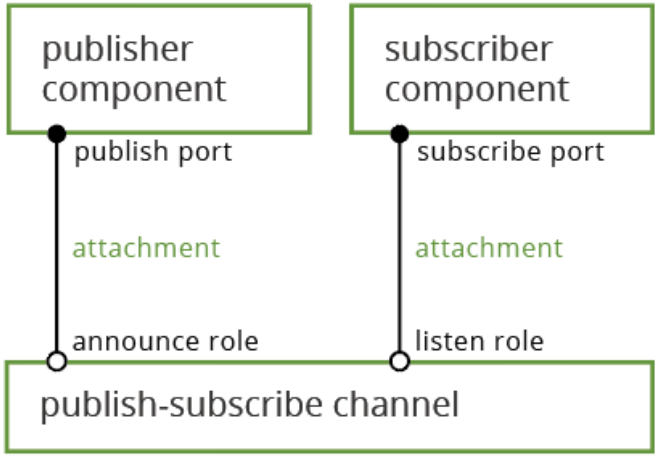
Conformance Checker

Prototype Design Conformance Checker



The approach builds on code analysis, software architecture, and machine learning.

Communication Styles in Software Systems



Publish-subscribe

Practitioner Vocabulary

- client-server
- N-tier
- service oriented architecture
- partitioning
- message passing
- distributed data service
- HTTP/HTTPS
- message queue
- shared memory
- sockets
- UDP/IP
- web services stack

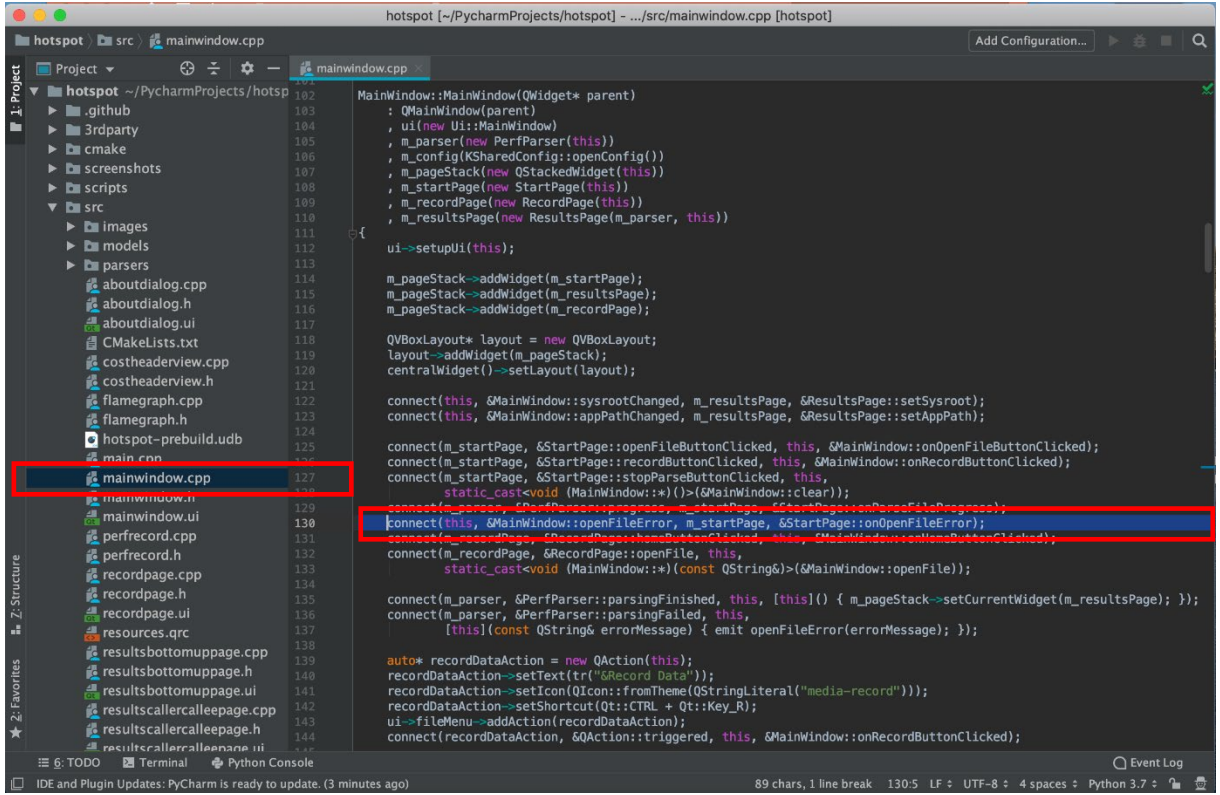
Canonical Design Knowledge

- synchronous publish-subscribe
- HTTP and message queue
- message-oriented middleware
- asynchronous point-to-point
- binary protocols

Software-reliant systems from 1998-2018, *Architecture Tradeoff Analysis Method (ATAM) Reports.*

Aksakalli et al. (2021). **Deployment and communication patterns in microservice architectures: A systematic literature review**, *Journal of Systems and Software*.

Hotspot using the Qt Framework



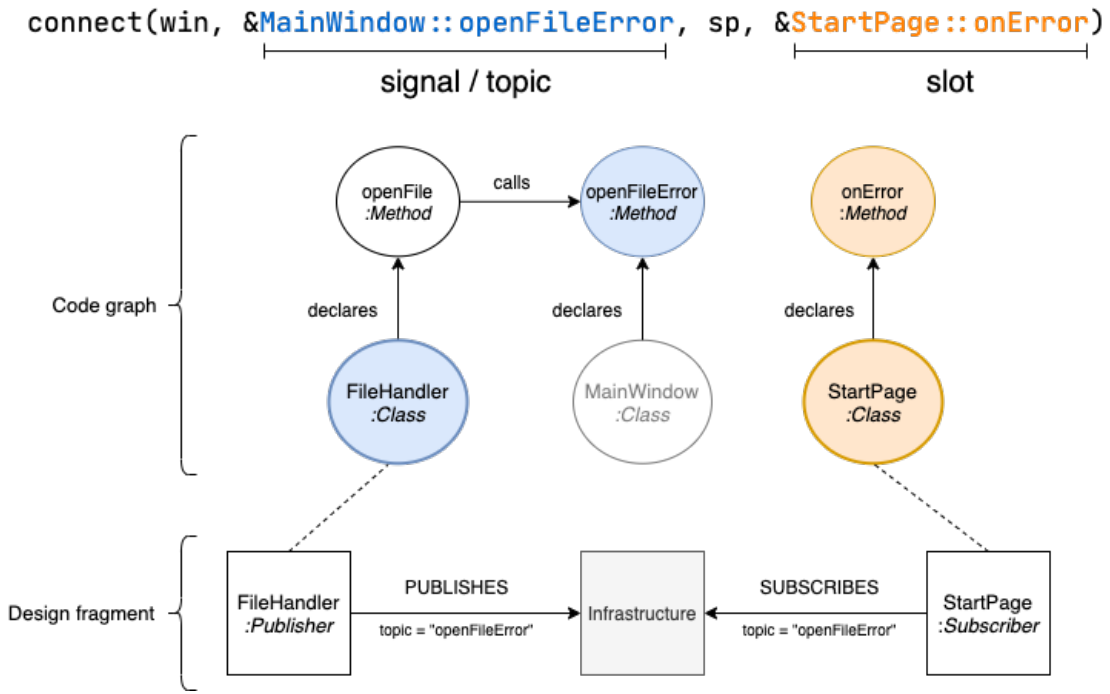
How do developers recognize design abstractions from code?

Hotspot

- performance analysis GUI
- 8K C++ code lines
- Qt framework
- 7 publishers
- 37 subscribers

github.com/KDAB/hotspot

From Code to Design Fragment



How would an automated technique recognize design abstractions from code?

- Rules or classifiers?
- Based on what data?
- How generalizable can you get?

Rules-Based Predictor


```
"MATCH (pub:Class)-[:DEFINES]->(:Method)-[:CALLS]->(signal:Method)-[:CALLS]->(a:Method) "
"WHERE a.longname = 'QMetaObject::activate' "
"RETURN DISTINCT pub.id "
```

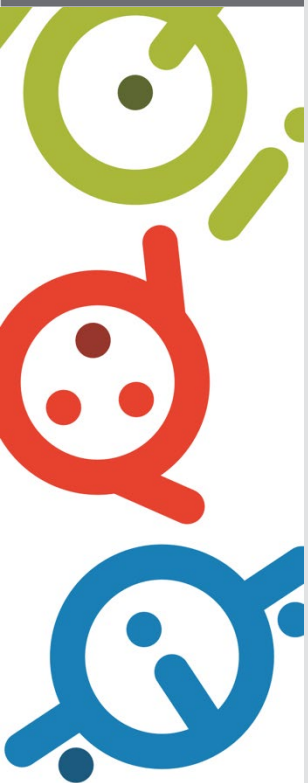
Cypher Query Language.

Rules are a reasonable approach for some abstractions in commonly used frameworks.

Work on rules-based predictors is work towards automating data labeling.

As we develop more precise definitions for communication styles, we will identify how different styles may be better characterized using different kinds of predictors.

Carnegie Mellon University
Software Engineering Institute

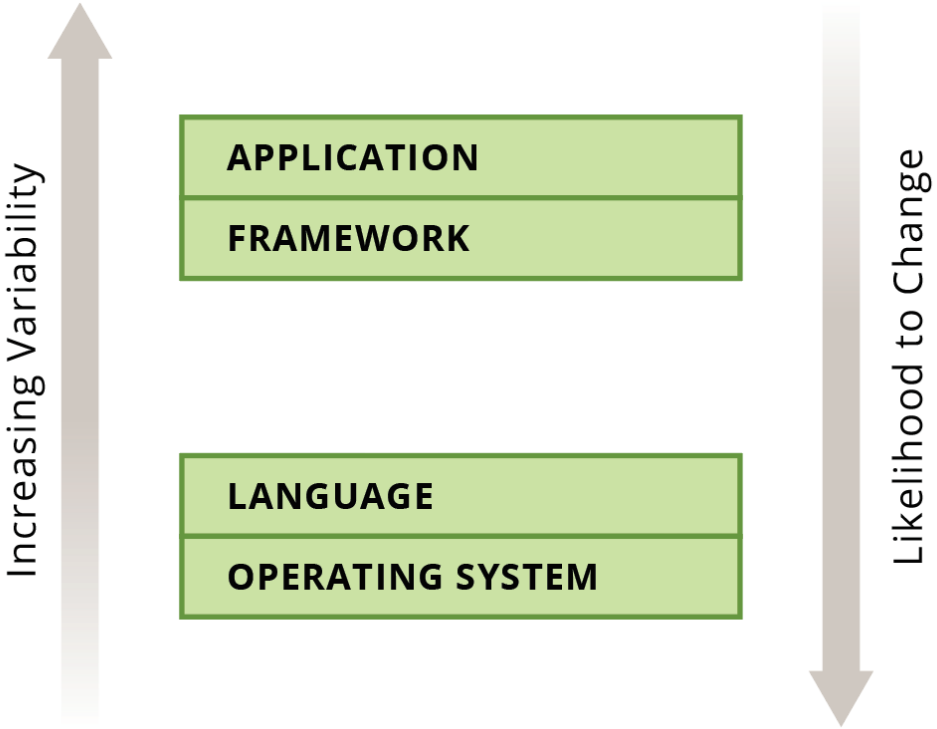


Research Review 2021

Automated Design Conformance during Continuous Integration

General Solution to the Design-Code Abstraction Gap

Expanding on the Conformance Checker Prototype

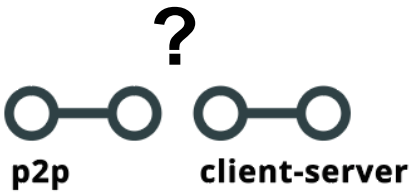


In the prototype, we used clues from frameworks to label constructs in multiple projects.

As we work towards generalizing techniques for finding architecture styles more broadly, we look to lower level realizations.

Formalizing and Differentiating Styles

Another challenge to predicting styles is that many styles can look similar, lack specificity and formalism in definition, or cannot be characterized by looking at only one source of information.



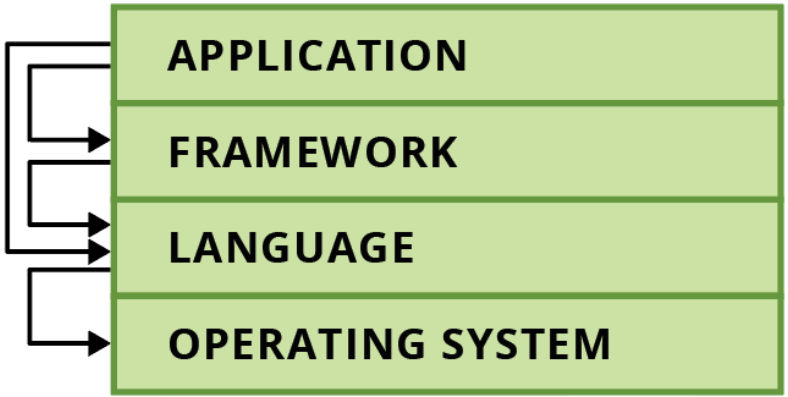
A 2-node peer-to-peer configuration can look similar to client-server.

Communication Style	Synchrony	Routing	Locality	Data Handling
Peer to peer	synchronous	indirect	remote	passing
Client-server	synchronous	direct	remote	passing
Publish-subscribe	asynchronous	indirect	remote	passing
Shared repository	asynchronous	direct	local	storing
…	…	…	…	…

Extracting and Connecting Information Across a Code Base

Calls from application source use services implemented in common frameworks.

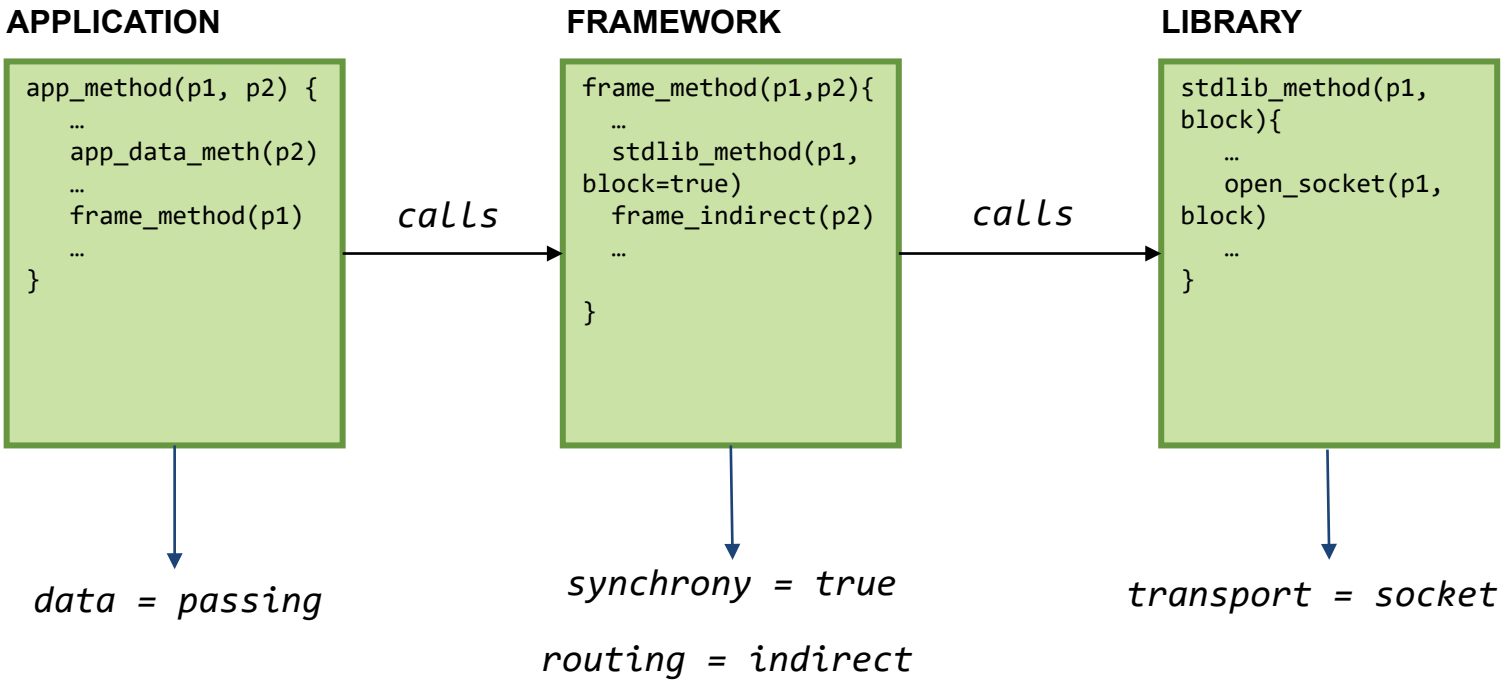
In turn, calls from frameworks are built on top of programming language libraries (e.g., C++ standard library) and operating system.



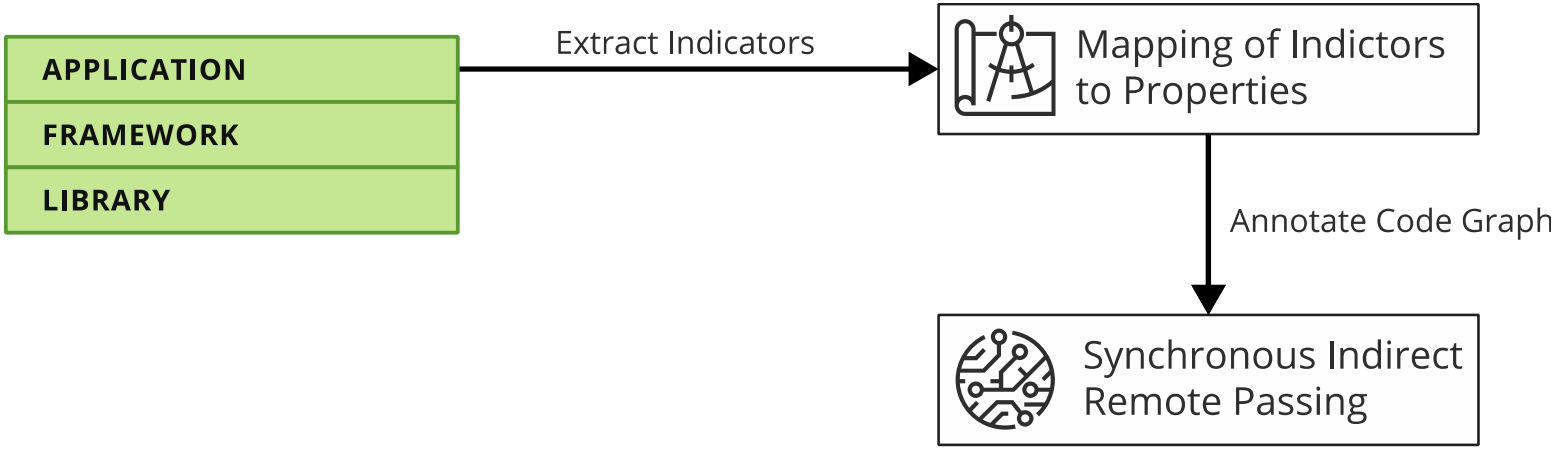

```
app_method(p1, p2) {
   …
   frame_method(p1)
…
}

frame_method(p1) {
   …
   stdlib_method(p1)
…
}
```

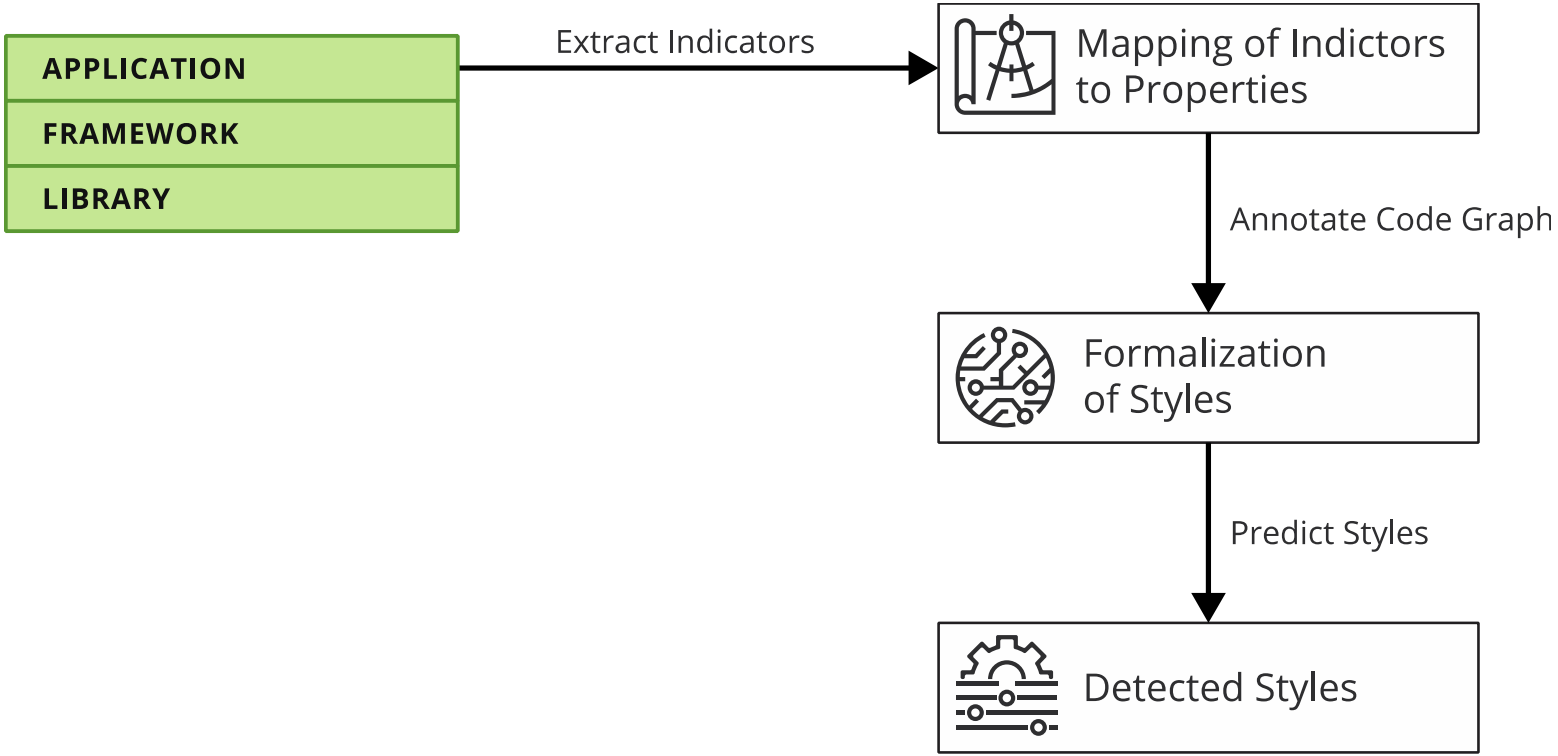
Indicators of Properties Exist In Multiple Files



Accumulating Indicators in One Representation

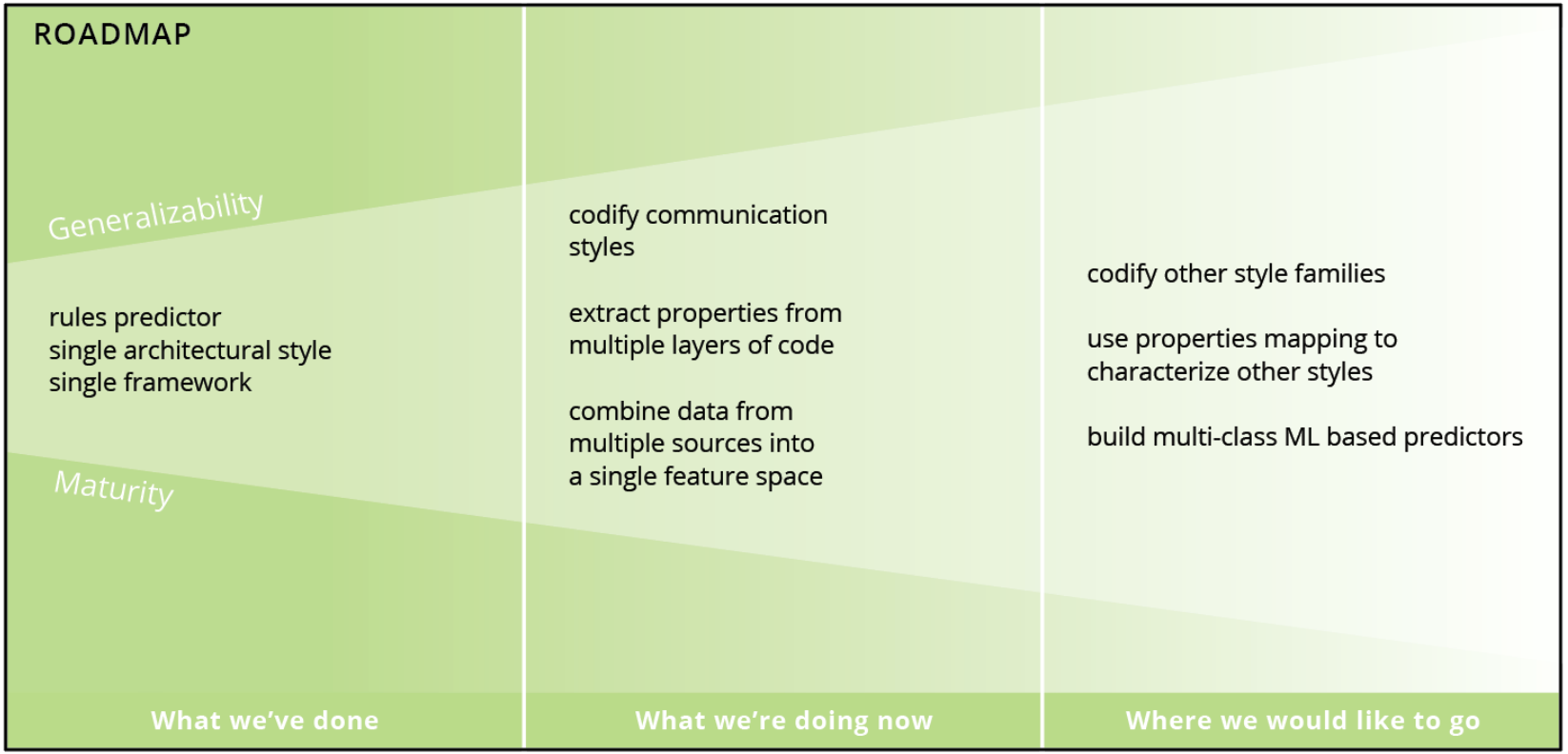


Bridging the Design – Code Abstraction Gap



Next Steps: More Projects, More Styles

Carnegie Mellon University
Software Engineering Institute



Project Team Members

Robert Nord
Principal Member of the Technical Staff, CMU / SEI

James Ivers
Principal Engineer, CMU / SEI

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

Lena Pons
Software Architecture and AI Researcher, CMU / SEI

Chris Seifried
Associate Engineer, CMU / SEI