

## Research Review 2021

# Automated Design Conformance during Continuous Integration

November 2021

Robert Nord, Lena Pons

# Document Markings

Copyright 2021 Carnegie Mellon University.

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.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

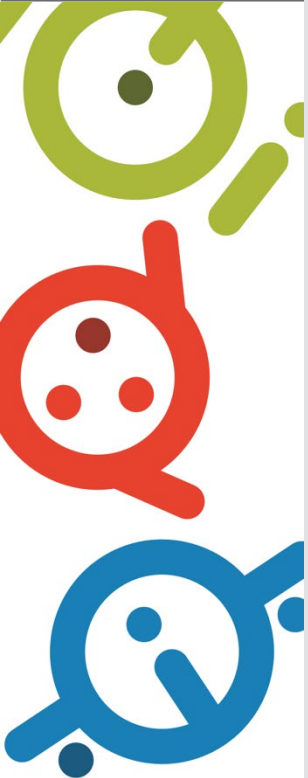
This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at [permission@sei.cmu.edu](mailto:permission@sei.cmu.edu).

DM21-0807

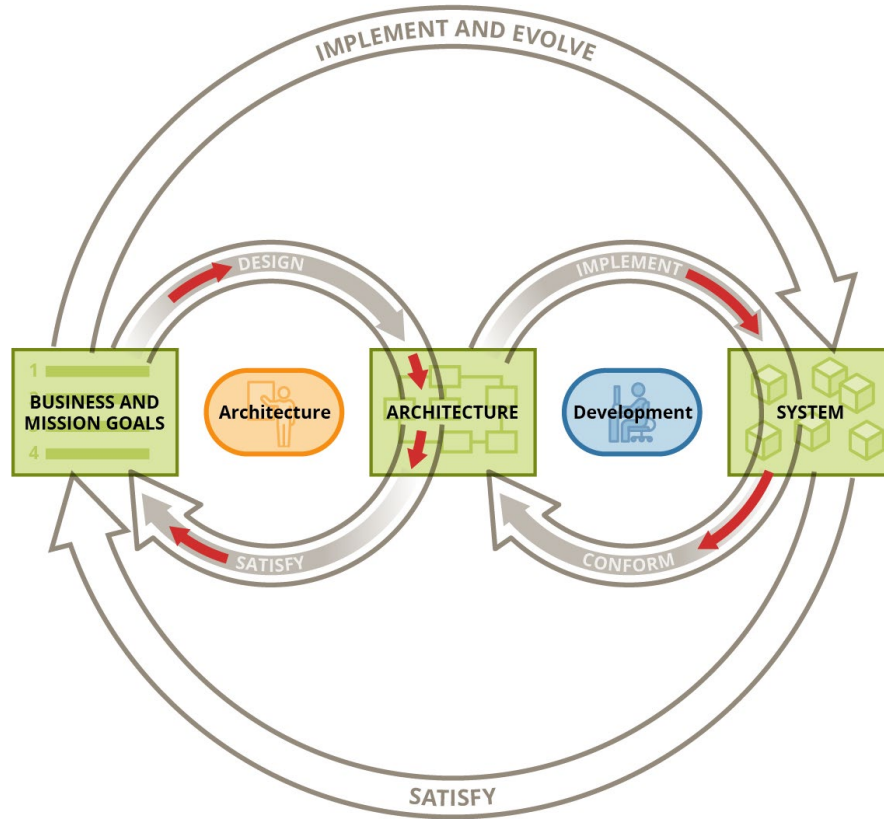
## Research Review 2021

Automated Design Conformance during Continuous Integration

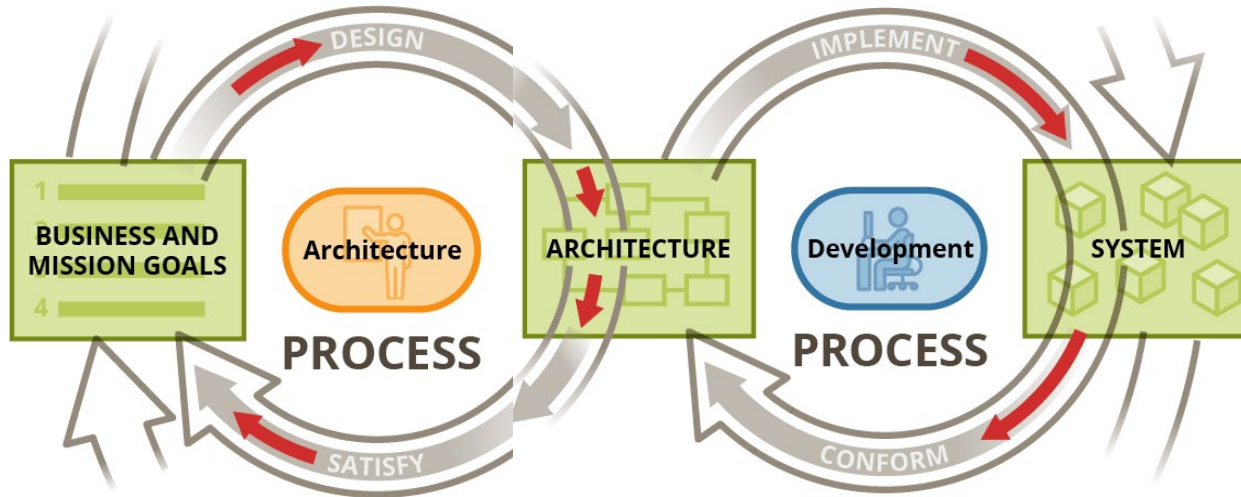
# Software Nonconformance



# Software Architecture Enables Our Ability to Innovate



# Software Architecture Enables Our Ability to Innovate



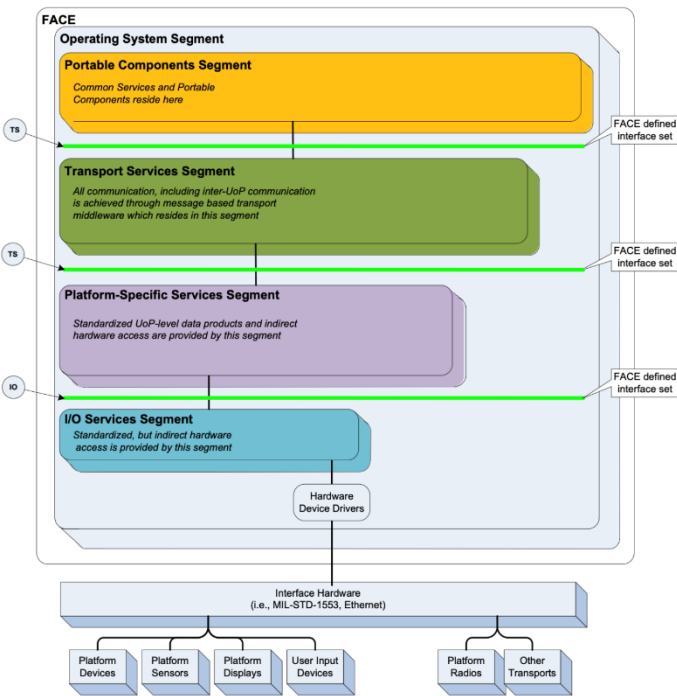
Software architecture is an abstraction that helps organizations satisfy business and mission goals.

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)

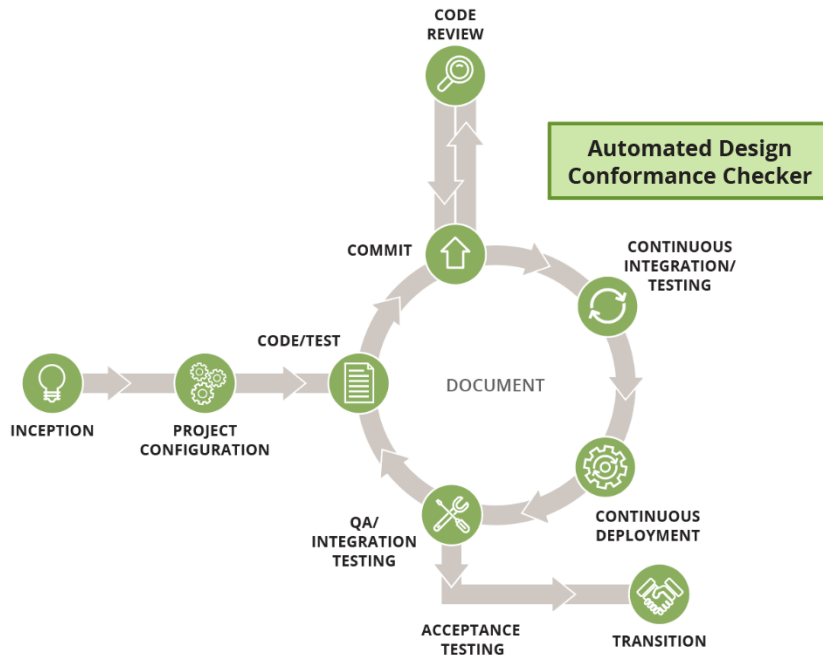
- technical and business strategy
- affordable and adaptable systems

## 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

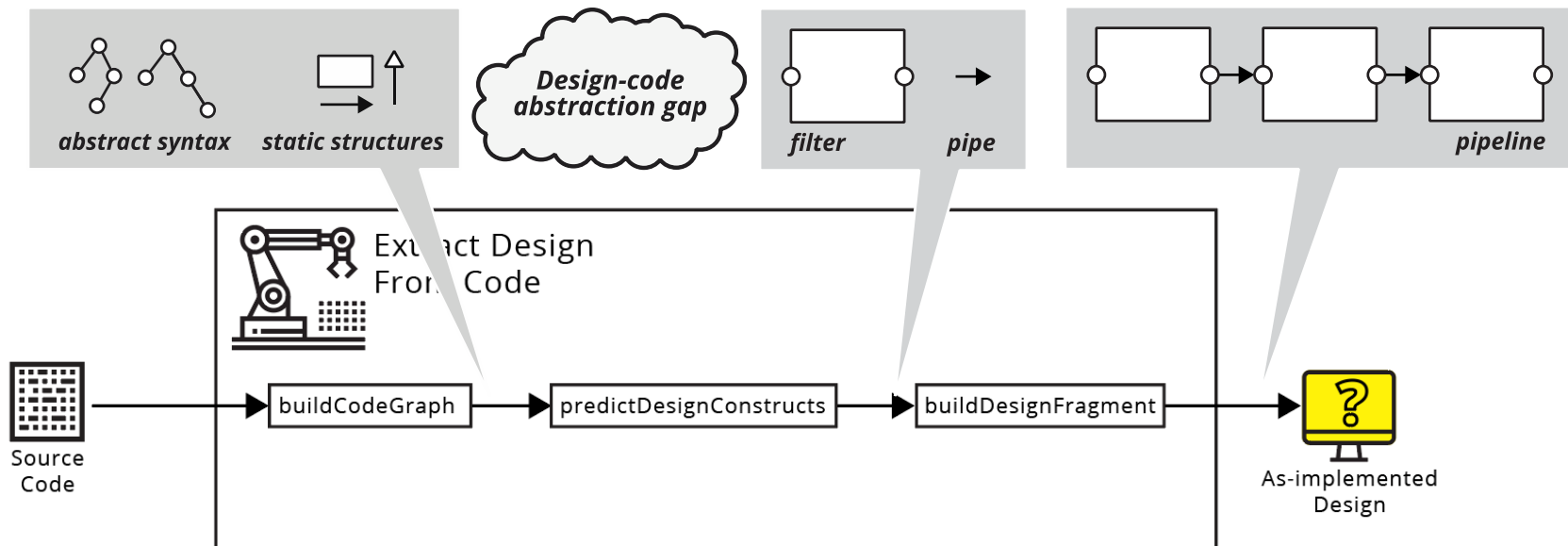


An automated design conformance checker integrated into a continuous integration (CI) workflow will reduce time to detect violations.

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.

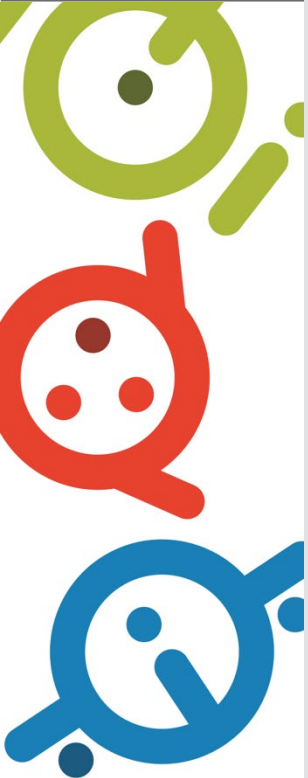
Ivers et al. (2019). **Can AI Close the Design-Code Abstraction Gap?** *International Workshop on Software Engineering Intelligence, IEEE/ACM International Conference on Automated Software Engineering (ASE)*.



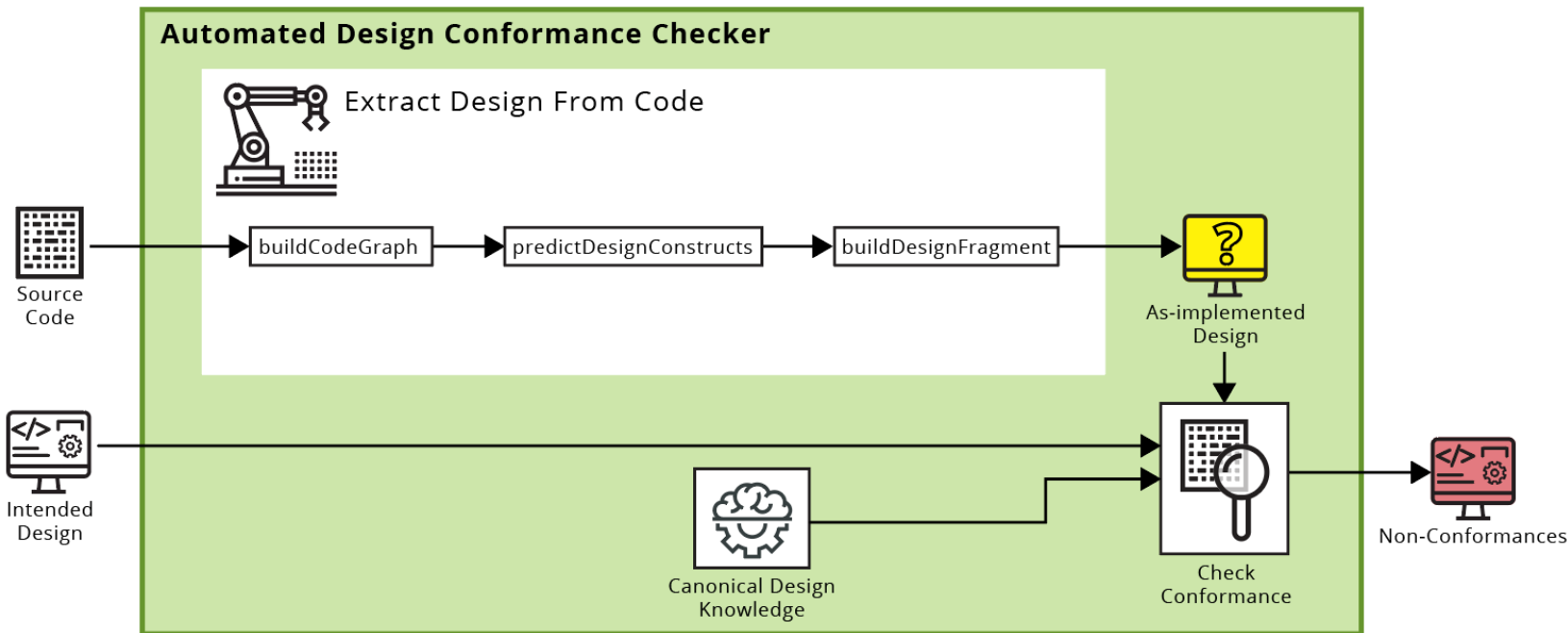
## Research Review 2021

Automated Design Conformance during Continuous Integration

# Conformance Checker

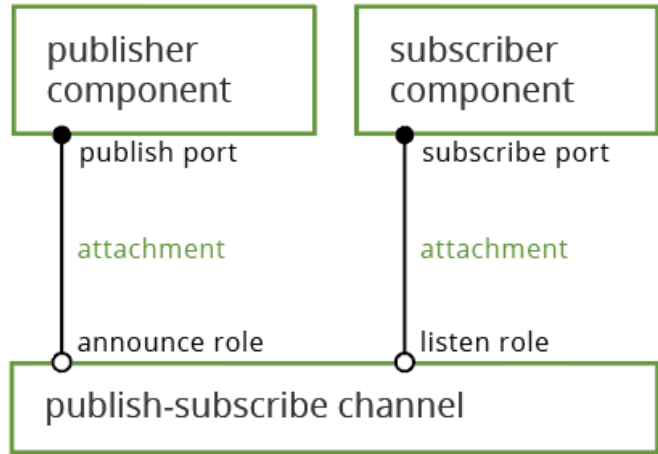


# Prototype Design Conformance Checker



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

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

### Canonical Design Knowledge

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

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

```

mainwindow.cpp
102 MainWindow:MainWindow(QWidget* parent)
103 : QMainWindow(parent)
104 , ui(new Ui:MainWindow)
105 , m_parser(new PerfParser(this))
106 , m_config(KSharedConfig::openConfig())
107 , m_pageStack(new QStackedWidget(this))
108 , m_startPage(new StartPage(this))
109 , m_recordPage(new RecordPage(this))
110 , m_resultsPage(new ResultsPage(m_parser, this))
111 {
112     ui->setupUi(this);
113
114     m_pageStack->addWidget(m_startPage);
115     m_pageStack->addWidget(m_resultsPage);
116     m_pageStack->addWidget(m_recordPage);
117
118     QVBoxLayout* layout = new QVBoxLayout;
119     layout->addWidget(m_pageStack);
120     centralWidget()->setLayout(layout);
121
122     connect(this, &MainWindow::sysrootChanged, m_resultsPage, &ResultsPage::setSysroot);
123     connect(this, &MainWindow::appPathChanged, m_resultsPage, &ResultsPage::setAppPath);
124
125     connect(m_startPage, &StartPage::openFileButtonClicked, this, &MainWindow::onOpenFileButtonClicked);
126     connect(m_startPage, &StartPage::recordButtonClicked, this, &MainWindow::onRecordButtonClicked);
127     connect(m_startPage, &StartPage::stopParseButtonClicked, this,
128             static_cast<void (MainWindow:*)()>(&MainWindow::clear));
129     connect(m_pageStack, &QStackedWidget::currentPageChanged, m_startPage, &StartPage::openFileProgress);
130     connect(this, &MainWindow::openFileError, m_startPage, &StartPage::onOpenFileError);
131     connect(m_recordPage, &RecordPage::openFileClicked, this, &MainWindow::onOpenFileClicked);
132     connect(m_recordPage, &RecordPage::openFile, this,
133             static_cast<void (MainWindow:*) (const QString&)>(&MainWindow::openFile));
134
135     connect(m_parser, &PerfParser::parsingFinished, this, [this]() { m_pageStack->setCurrentWidget(m_resultsPage); });
136     connect(m_parser, &PerfParser::parsingFailed, this,
137             [this](const QString& errorMessage) { emit openFileError(errorMessage); });
138
139     auto* recordDataAction = new QAction(this);
140     recordDataAction->setText(tr("&Record Data"));
141     recordDataAction->setIcon(QIcon::fromTheme(QStringLiteral("media-record")));
142     recordDataAction->setShortcut(Qt::CTRL + Qt::Key_R);
143     ui->fileMenu->addAction(recordDataAction);
144     connect(recordDataAction, &QAction::triggered, this, &MainWindow::onRecordButtonClicked);
  
```

How do developers recognize design abstractions from code?

## Hotspot

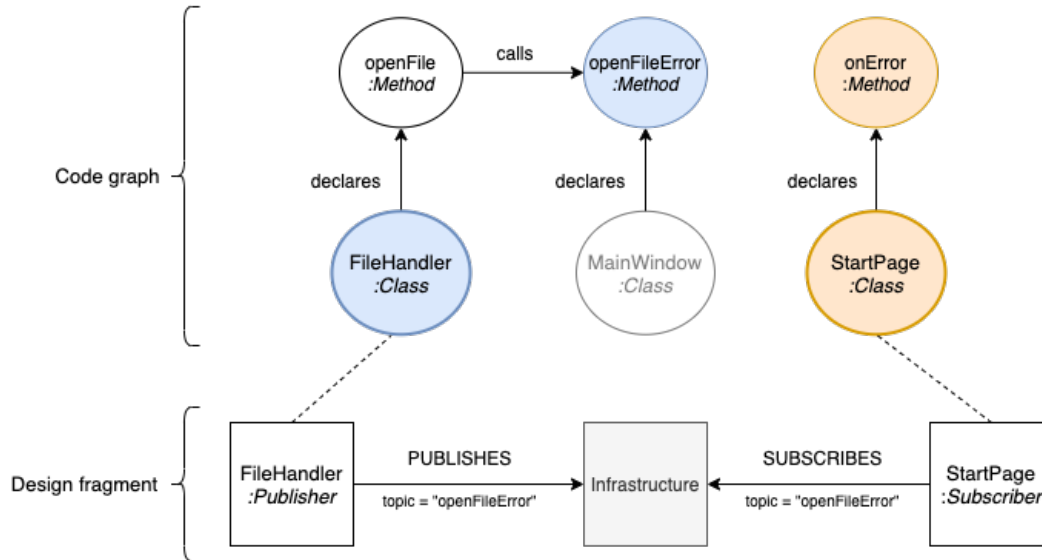
- performance analysis GUI
- 8K C++ code lines
- Qt framework
- 7 publishers
- 37 subscribers

[github.com/KDAB/hotspot](https://github.com/KDAB/hotspot)

# From Code to Design Fragment

```
connect(win, &MainWindow::openFileError, sp, &StartPage::onError)
```

-----
 signal / topic                      slot



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.

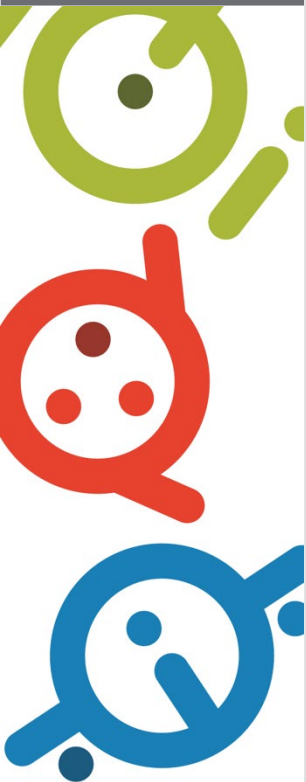
Work on rules-based predictors is work towards automating data labeling.

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

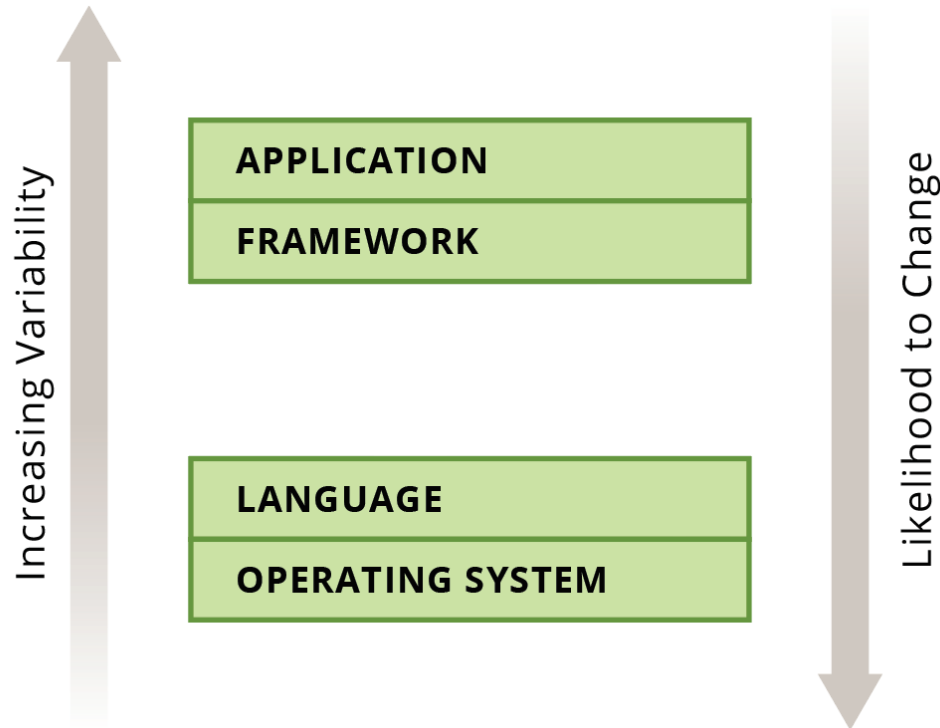
## Research Review 2021

Automated Design Conformance during Continuous Integration

# General Solution to the Design-Code Abstraction Gap



# Expanding on the Conformance Checker Prototype



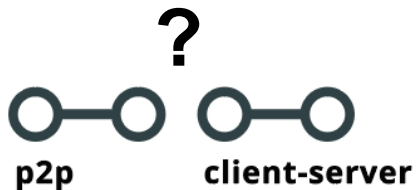
In the prototype, we used clues from frameworks to label constructs in multiple projects.

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.



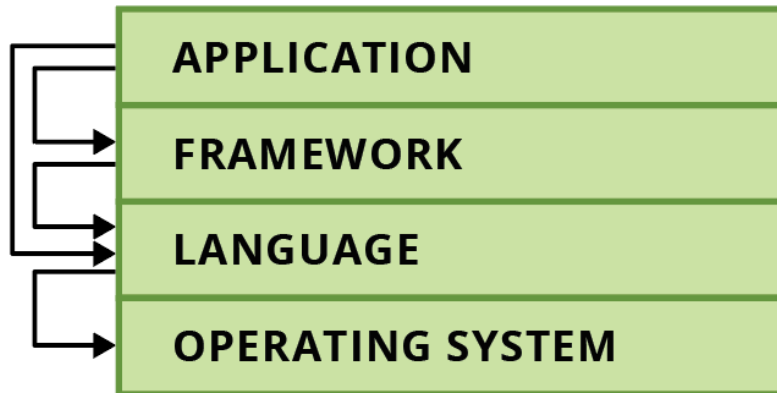
*A 2-node peer-to-peer configuration can look similar to client-server.*

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

## APPLICATION

```
app_method(p1, p2) {
  ...
  app_data_meth(p2)
  ...
  frame_method(p1)
  ...
}
```

*calls*

## FRAMEWORK

```
frame_method(p1,p2){
  ...
  stdlib_method(p1,
  block=true)
  frame_indirect(p2)
  ...
}
```

*calls*

## LIBRARY

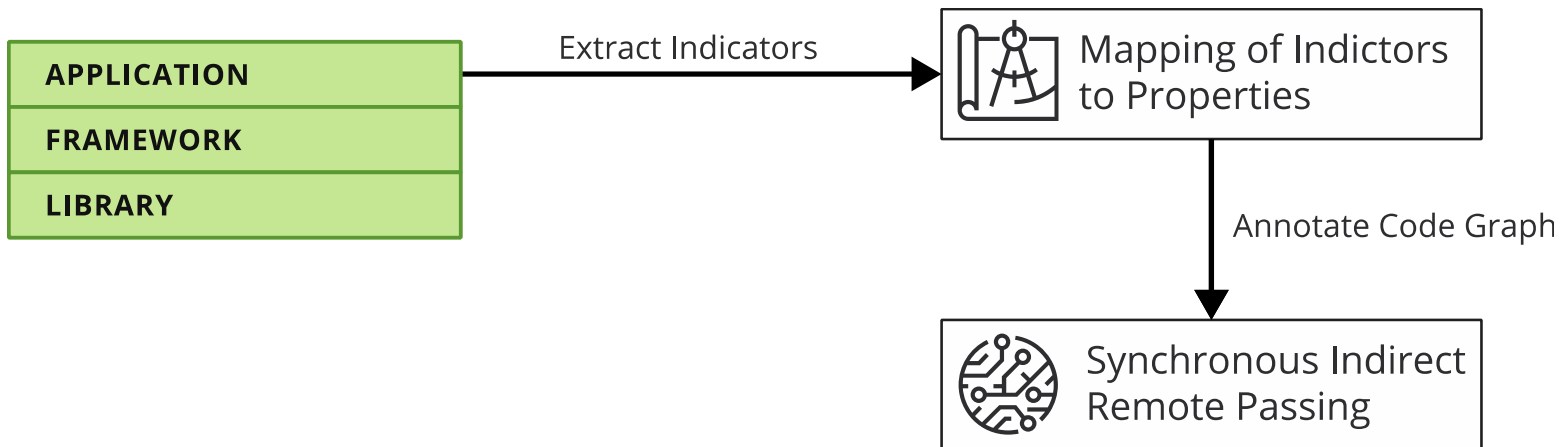
```
stdlib_method(p1,
  block){
  ...
  open_socket(p1,
  block)
  ...
}
```

*data = passing*

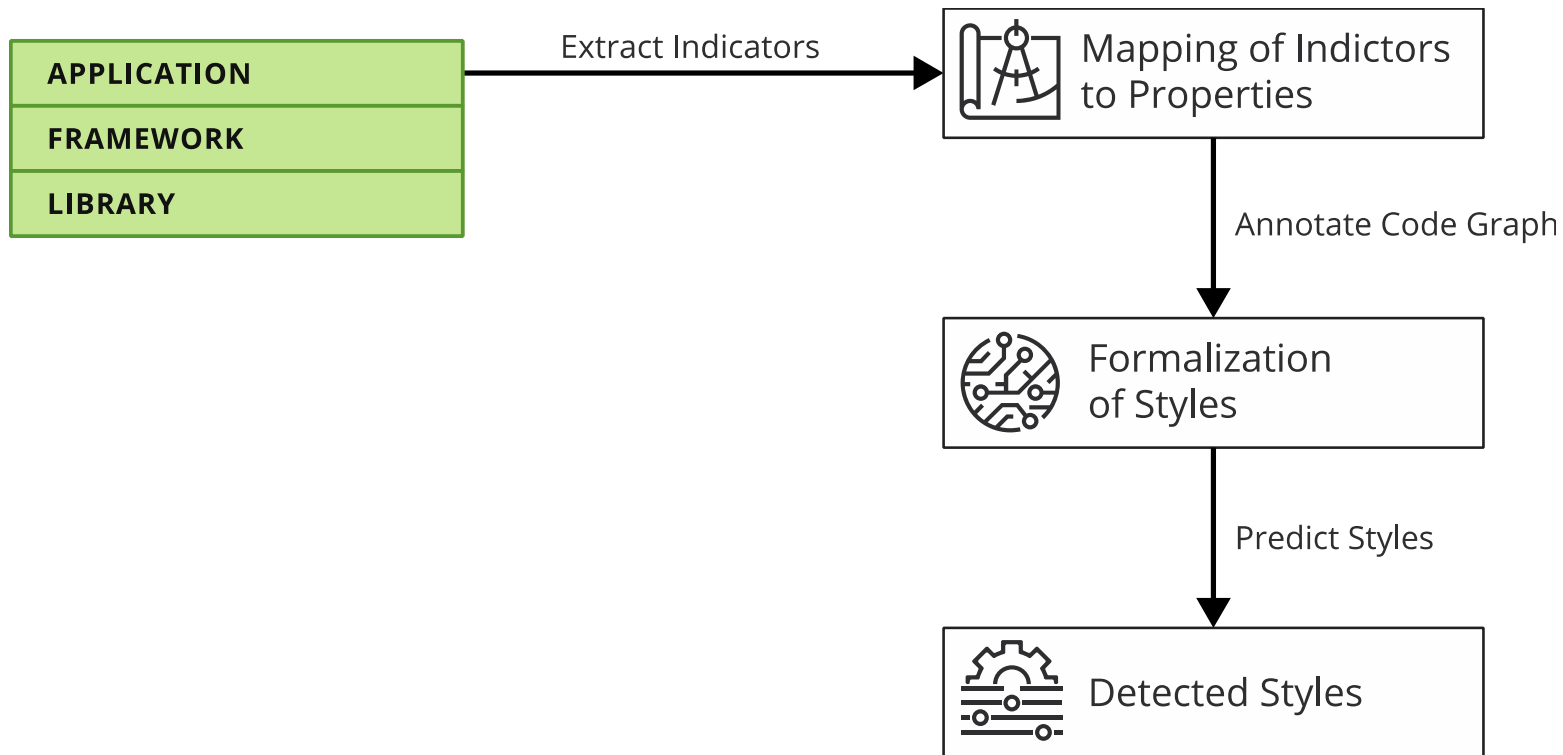
*synchrony = true*  
*routing = indirect*

*transport = socket*

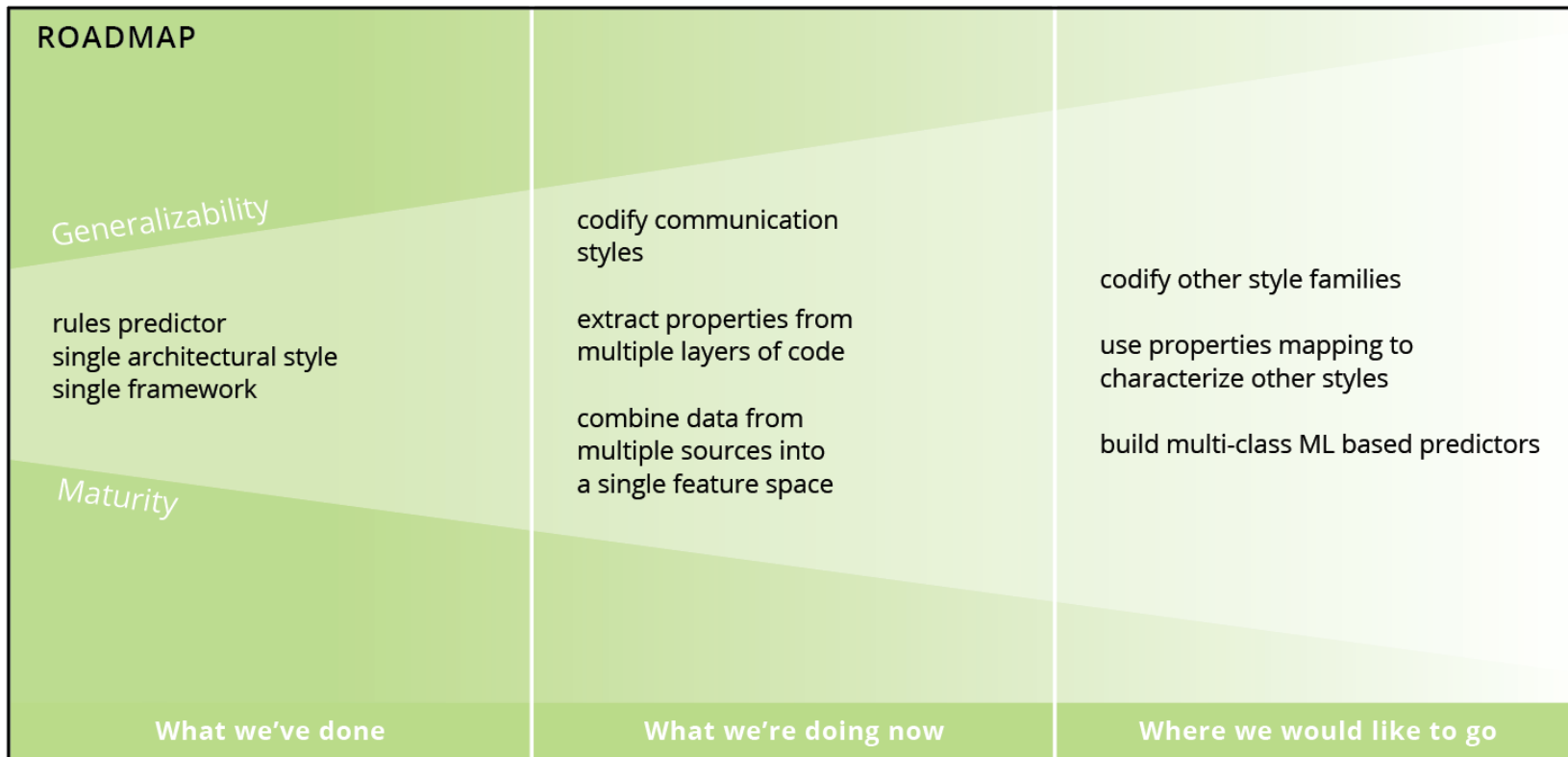
# Accumulating Indicators in One Representation



# Bridging the Design – Code Abstraction Gap



# Next Steps: More Projects, More Styles



# 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