RESEARCH REVIEW 2022

Automated Design Conformance During Continuous Integration

NOVEMBER 14–16, 2022

Robert Nord
Principal Researcher

Carnegie Mellon University
Software Engineering Institute

Introduction

Software architecture enables our ability to innovate through extensible design. The end goal–to build systems that provide timely and cost-effective capability to users–is achieved only if the code conforms to the architecture.

This project developed an automated conformance checker prototype that can be used in a continuous integration workflow to discover nonconformances within minutes, instead of the months or years it takes today.

This work helps teams detect problems as they are introduced, allowing faster and more economical realignment of code and architecture and increasing confidence that sustainable code is being delivered.

Automated Design Conformance during Continuous Integration

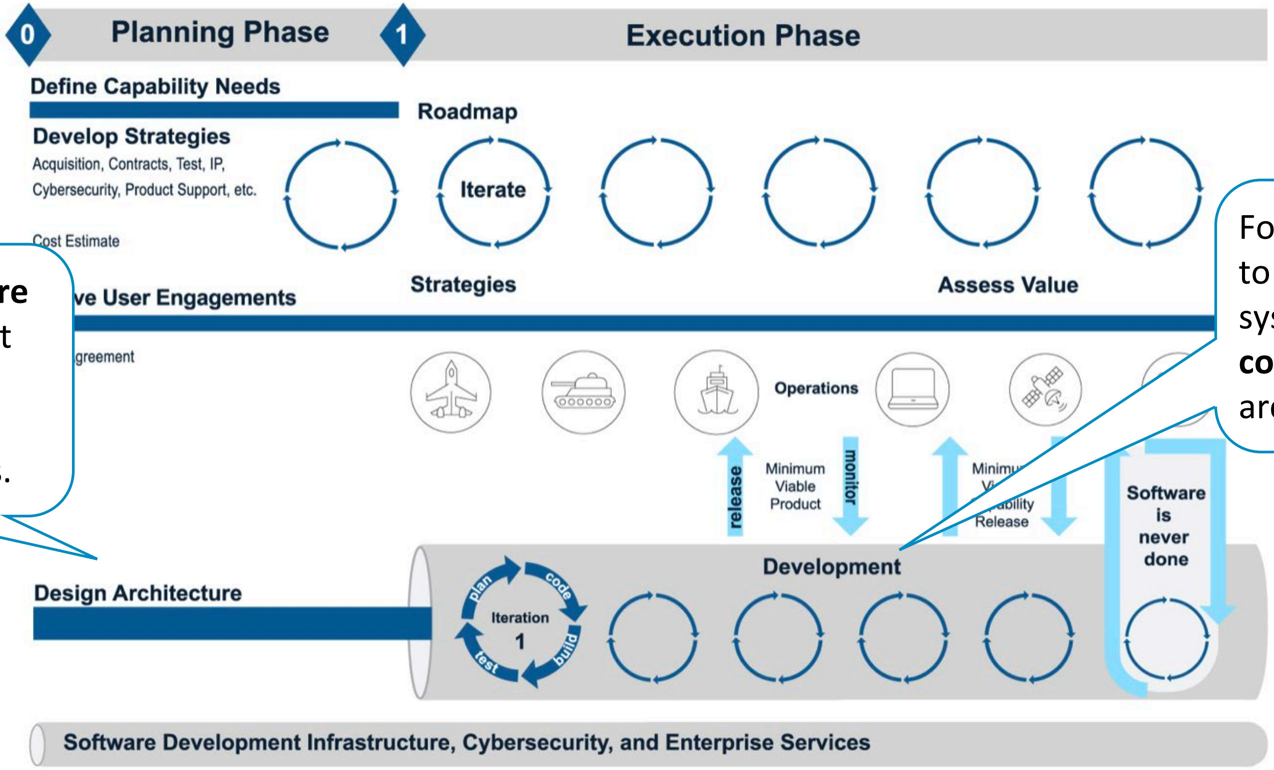
Why Conformance Matters for Open Software Systems

Carnegie Mellon University Software Engineering Institute

Software Architecture Enables Our Ability to Innovate

Carnegie Mellon University Software Engineering Institute

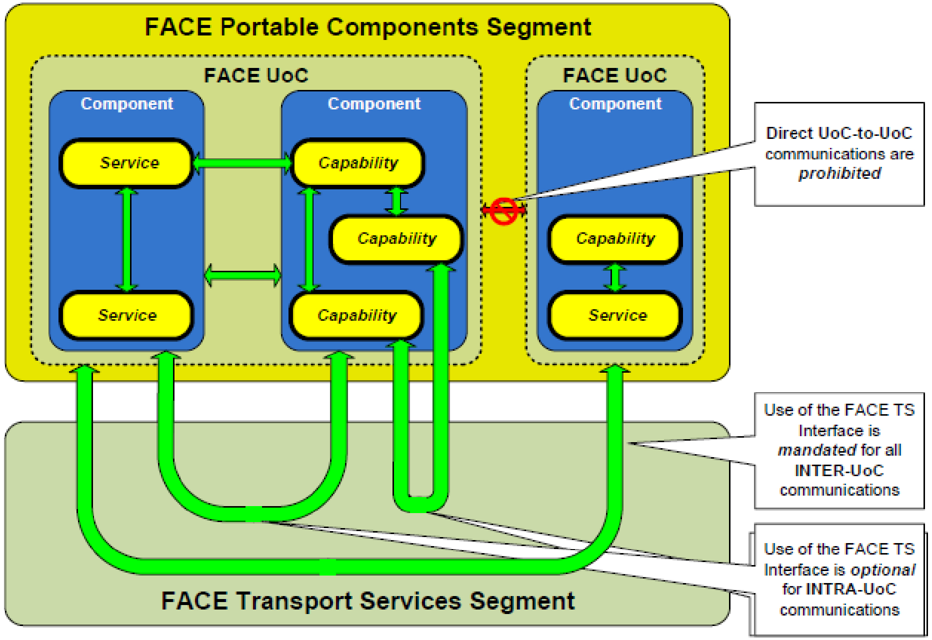
Software architecture is an abstraction that helps organizations satisfy mission goals and capability needs.



For the implementation to exhibit the desired system qualities, it must **conform** to the architecture.

Lifecycle View of Software Acquisition https://aaf.dau.edu/aaf/software/

Challenges in Conformance Checking



The Open Group (2017). **Example Inter- and Intra-UoC Communication,** *FACE (Future Airborne Capability Environment) Technical Standard, Edition 3.0.*

Conformance Checks

- *Inter-construct communication relations originate from a construct and end at infrastructure.*
- *The intended specification allows communication between construct A and construct B.*
- *The implemented design has all constructs listed in the intended specification.*

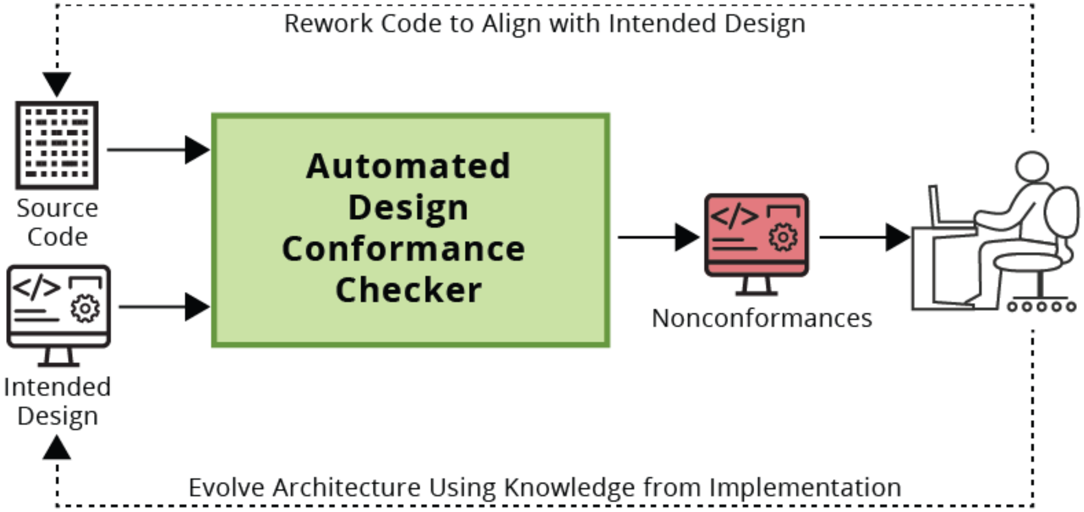
State of the Practice

- Component-level manual inspection
- ISO code quality standards, maintainability
- Modularity, dependencies, design paradigms

Challenges

- Automated inspection checks
- System-level checks: constructs and relations
- Conformance to architecture styles

Automated Conformance Checking during CI



Automating conformance checking and feeding back updates to maintain alignment

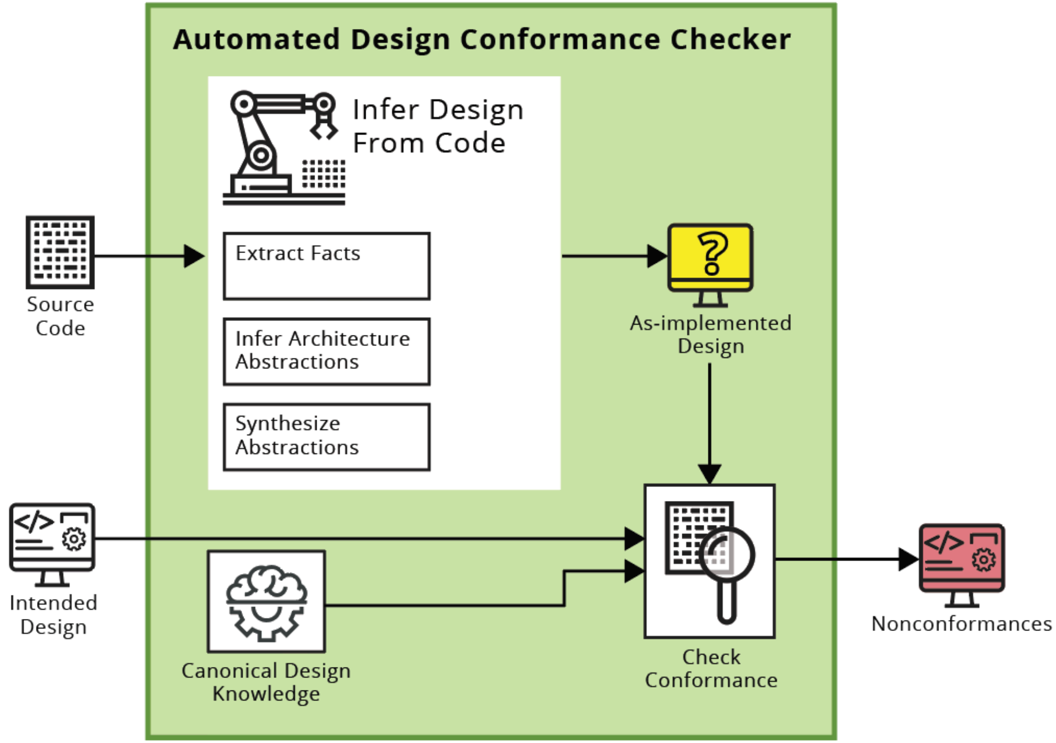
Conformance is the practice of keeping the architecture and code aligned.

Development teams check during **continuous integration** that implementation and architecture are aligned.

Automated Design Conformance during Continuous Integration

Automating Conformance Checking

Infer Design from Code



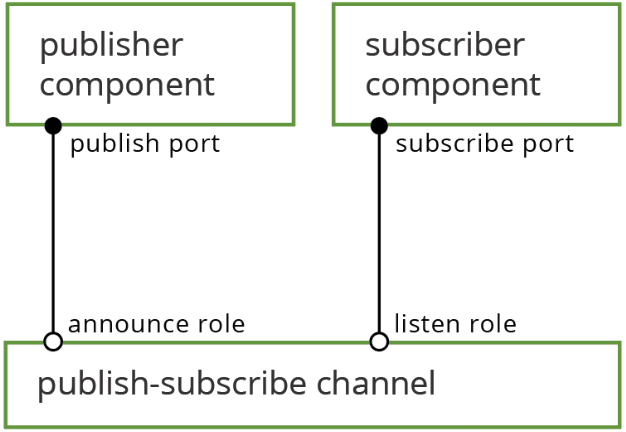
Conformance checker design

The key to this work is new research inferring **design** information from source code.

Detecting design constructs is challenging due to

- imprecise definitions of abstractions
- variation in implementation
- limits of fact gathering analyses

Using Frameworks to Infer Design



Publish-subscribe communication style

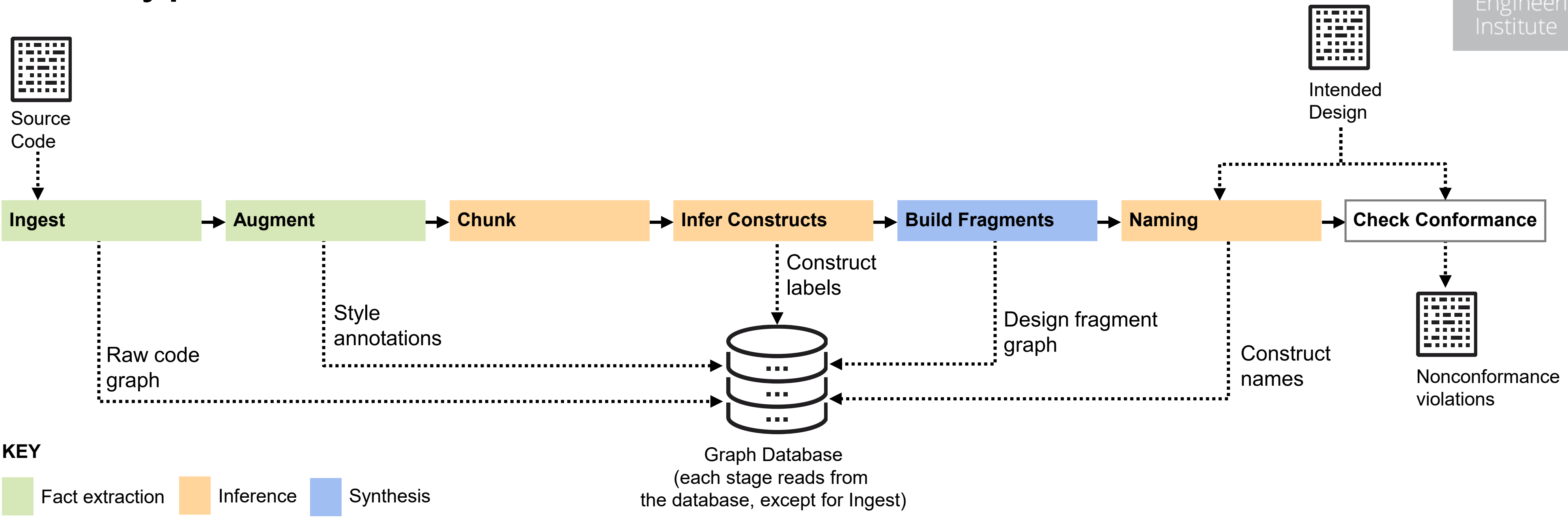
Concept	FACE	ROS
Publish *Intent to generate messages*	FACE::Create_Connection (*name, pattern, direction, conn_id);	NodeHandle::advertise <msg_type>(topic)
Update *Dissemination of messages*	FACE::Send_Message (conn_id, data);	M_statusPub.publish
Subscribe *Interest receiving messages*	FACE::Create_Connection (*name, pattern, direction, conn_id);	NodeHandle::subscribe (topic)
Reflect *Reception of messages*	FACE::Receive_Message (conn_id, data);	*not explicit in code*

Publish-subscribe concept to framework map

Choosing a framework to realize an architecture style

- constrains code to use framework's interfaces to realize the style
- supplies structure for implementing the styles chosen for an application

Prototype Conformance Checker



Example Input: Source Code and Intended Design


```
# element declarations

CameraAutoBalance : Publisher (im);
XBeeNode : Publisher (runstop, gpsBaseRTCM3, pose_estimate);

XBeeCoordinator : Subscriber (pose_estimate, runstop, gpsBaseRTCM3);
StatusMonitor: Subscriber (mppi_controller/mppiStatus);

AutoRallyChassis :
    Publisher (chassisCommand, chassisState, wheelSpeeds),
    Subscriber (chassisCommand);

Runstop :
    Publisher (runstopData),
    Subscriber (runstop);

StateEstimator :
    Publisher (point, ptAcc, ptGyro, poseNew, delays, statusMsgs),
    Subscriber (gps,imu, wheel_odom);
```

Intended Design Excerpt

Source Code

- AutoRally Project
- Software for AutoRally Platform
- ~200K C++ code lines
- ROS-framework

Intended Design

- instances of publisher and subscriber design constructs
- message publication and subscription

Carnegie Mellon University Software Engineering Institute

Example Output: Nonconformances Found

```
[INTENDED] Construct CameraAutoBalance does not exist in the as implemented fragment.
[INTENDED] Construct StatusMonitor does not exist in the as implemented fragment.
[INTENDED] Construct Runstop does not have the correct type of <EntityType:Subscriber>.
[INTENDED] Construct ConstantSpeedController does not exist in the as implemented fragment.
[INTENDED] F_PUBLISHES relation from CameraAutoBalance to Infrastructure is not in the as implemented fragment.
[INTENDED] F_SUBSCRIBES relation from StatusMonitor to Infrastructure is not in the as implemented fragment.
[INTENDED] F_SUBSCRIBES relation from Runstop to Infrastructure is not in the as implemented fragment.
[INTENDED] F_PUBLISHES relation from ConstantSpeedController to Infrastructure is not in the as implemented fragment.
[INTENDED] F_SUBSCRIBES relation from ConstantSpeedController to Infrastructure is not in the as implemented fragment.
[INTENDED] F_SUBSCRIBES relation from XBeeNode to Infrastructure is not allowed.
10 nonconformances found.
```

Nonconformances Found

```
if(transmitPositionRate > 0)
{
  m_transmitPositionTimer = m_nh.createTimer(
                                ros::Duration(1.0/transmitPositionRate),
                                &XbeeNode::transmitPosition,
                                this);
  m_poseSubscriber = nh.subscribe("pose_estimate", 1,
                                   &XbeeNode::odomCallback,
                                   this);
}
```

Tracing Nonconformance to Code

```

# element declarations

CameraAutoBalance : Publisher (im);
XBeeNode : Publisher (runstop, gpsBaseRTCM3, pose_estimate);

```

Tracing Nonconformance to Intended Design

Automated Design Conformance during Continuous Integration

Looking Forward

Carnegie Mellon University Software Engineering Institute

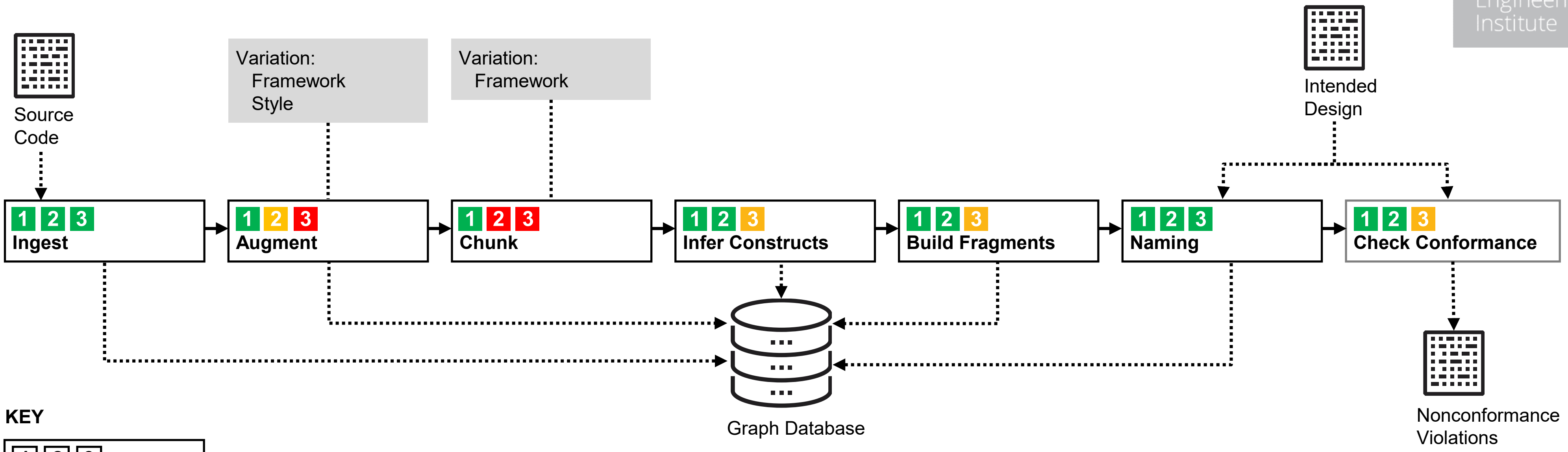
What Practical Problems Does the Approach Solve?

	State of Practice		Design Conformance
	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 under conformance.

What Is Involved in Applying the Approach



KEY

1 2 3
Stage

Type of Change to Prototype

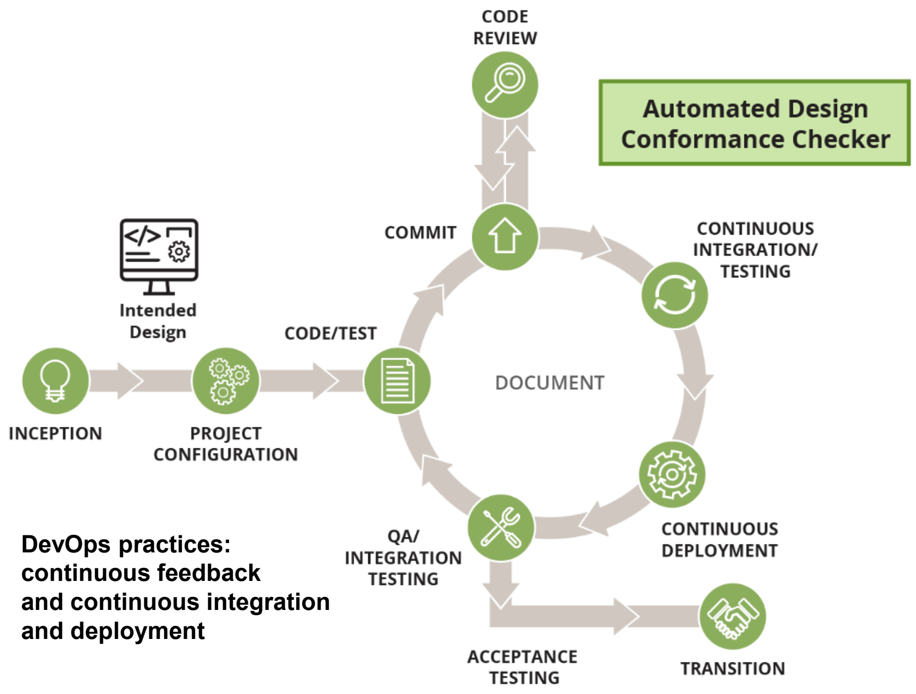
1. New system for known framework
2. New framework for known style
3. New style

Degree of change to prototype

reusable easier harder

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

Improve Conformance of Implementations to Architectures



DevOps practices: continuous feedback and continuous integration and deployment

An automated design conformance checker integrated into a CI workflow

- exposes nonconformances 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

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

Josh Fallon
Defense Network Analyst
CMU / SEI

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.

DM22-0796