Automated Design Conformance during Continuous Integration

November 2021
Robert Nord, Lena Pons
Automated Design Conformance during Continuous Integration

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.

DM21-0807
Research Review 2021

Automated Design Conformance during Continuous Integration

Software Nonconformance
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

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.

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

Research Review 2021

Automated Design Conformance during Continuous Integration

Conformance Checker
The approach builds on code analysis, software architecture, and machine learning.
Communication Styles in Software Systems

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

Publish-subscribe


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
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.
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.

<table>
<thead>
<tr>
<th>Communication Style</th>
<th>Synchrony</th>
<th>Routing</th>
<th>Locality</th>
<th>Data Handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peer to peer</td>
<td>synchronous</td>
<td>indirect</td>
<td>remote</td>
<td>passing</td>
</tr>
<tr>
<td>Client-server</td>
<td>synchronous</td>
<td>direct</td>
<td>remote</td>
<td>passing</td>
</tr>
<tr>
<td>Publish-subscribe</td>
<td>asynchronous</td>
<td>indirect</td>
<td>remote</td>
<td>passing</td>
</tr>
<tr>
<td>Shared repository</td>
<td>asynchronous</td>
<td>direct</td>
<td>local</td>
<td>storing</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

A 2-node peer-to-peer configuration can look similar to client-server.
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.
Indicators of Properties Exist In Multiple Files

**APPLICATION**

\[ \text{app\_method}(p1, p2) \{
    \text{...}
    \text{app\_data\_meth}(p2)
    \text{...}
    \text{frame\_method}(p1)
    \text{...}
\} \]

- **data** = passing

**FRAMEWORK**

\[ \text{frame\_method}(p1,p2)\{
    \text{...}
    \text{stdlib\_method}(p1, block=true)
    \text{...}
    \text{frame\_indirect}(p2)
    \text{...}
\} \]

- **synchrony** = true

**LIBRARY**

\[ \text{stdlib\_method}(p1, block)\{
    \text{...}
    \text{open\_socket}(p1, block)
    \text{...}
\} \]

- **transport** = socket

- **routing** = indirect
Accumulating Indicators in One Representation

- **APPLICATION**
- **FRAMEWORK**
- **LIBRARY**

Extract Indicators → Mapping of Indictors to Properties → Annotate Code Graph → Synchronous Indirect Remote Passing
Bridging the Design – Code Abstraction Gap

APPLICATION
FRAMEWORK
LIBRARY

Extract Indicators

Mapping of Indicators to Properties

Annotate Code Graph

Formalization of Styles

Predict Styles

Detected Styles
Next Steps: More Projects, More Styles

ROADMAP

Generalizability
rules predictor
single architectural style
single framework

codify communication styles
extract properties from multiple layers of code
combine data from multiple sources into a single feature space

codify other style families
use properties mapping to characterize other styles
build multi-class ML based predictors

Maturity

What we’ve done

What we’re doing now

Where we would like to go