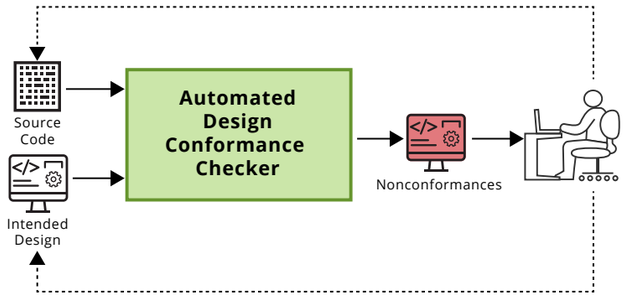
Automated Design Conformance during Continuous Integration

Problem
Code often does not conform to designs, undermining properties such as extensibility and composability. Late detection increases cost and delays delivering capability to the field.

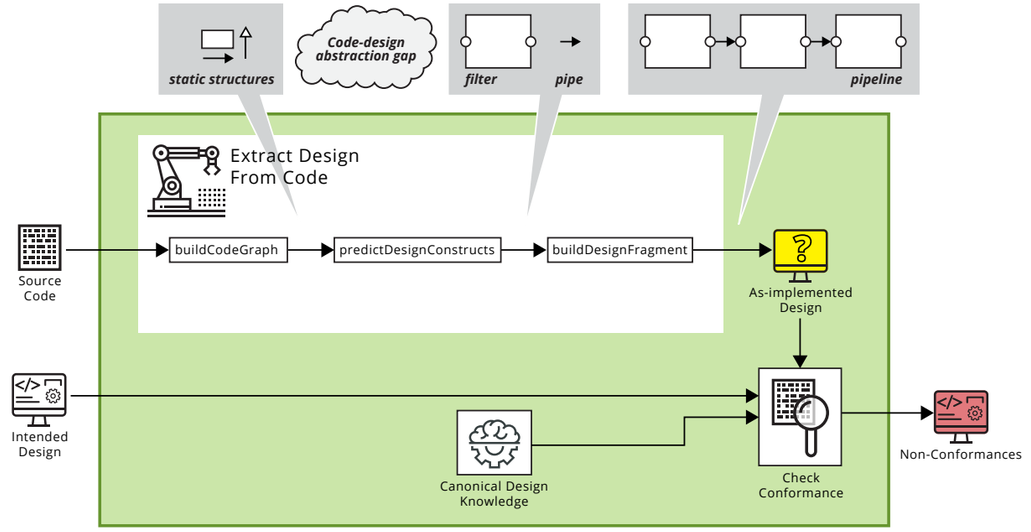
Solution
Use code analysis, software architecture knowledge, and machine learning to automatically extract design as implemented in the code and check conformance with the intended design.



Intended Impact (FY20–22)
- Recommendations correctly identify nonconformance and detect at the commit that introduces nonconformance.
- Automation enables early detection and allows remediation before the violation gets "baked in" to the implementation.
- Detection of nonconformances allows program managers to hold developers (contractor or organic) accountable.

Read more about our approach:
Nord (2020). *Using Machine Learning to Detect Design Patterns*, SEI Blog.

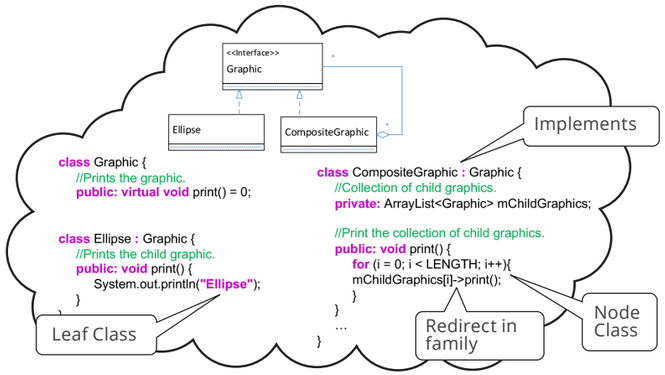
An **automated design conformance checker** integrated into a continuous integration workflow will reduce time to detect violations from months or years to hours.



Automated Design Conformance Checker

Approach
Our solution builds on code analysis, software architecture, machine learning, and continuous integration**.** We ingest a software repository and **build a graph** representation of the code structure based on code analysis. We apply machine learning to bridge the abstraction gap to **extract design constructs** from the code. We **build the design fragments** that comprise the as-implemented design. The as-implemented design can then be **checked for conformance** against the intended design at each code commit during **continuous integration.**

The central research of this project uses **machine learning** to extract features by recognizing abstractions commonly used in software architecture in C++ source code.



Code-Design Abstraction Gap

Feature engineering is key to extracting design and bridging the gap. Structural and behavioral features link elements (e.g., classes) though relations (e.g., inheritance, method call).

Our prototype advances the state of the art in applying machine learning to software engineering tasks and aligns with SEI strategic focus areas of timely and trustworthy software by introducing automation into the development and acquisition lifecycle.

Robert Nord, Ben Cohen, Shane Ficorilli,
James Ivers, John Klein, Lena Pons, Chris Seifried
info@sei.cmu.edu

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.

DM20-0866