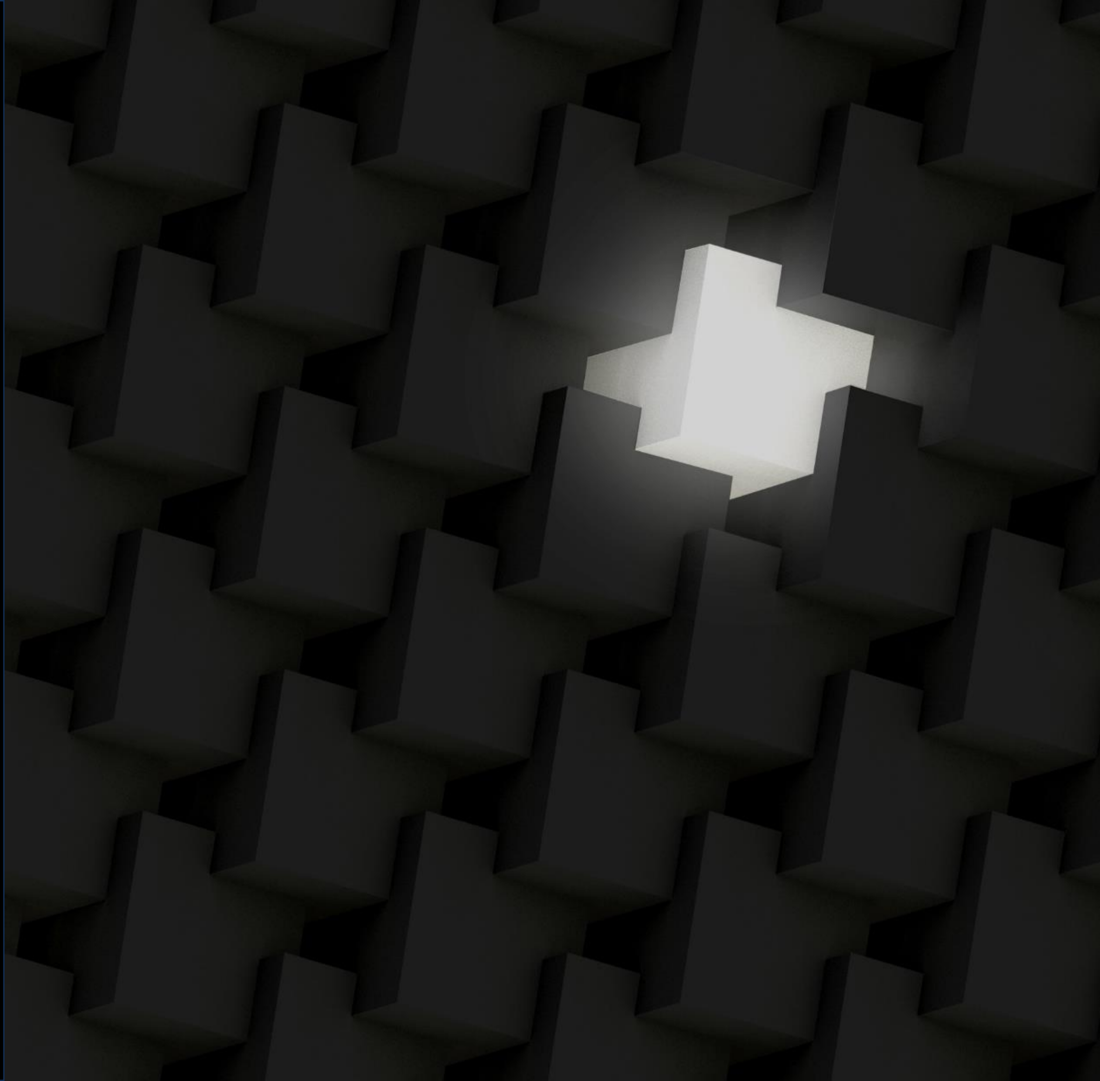
Carnegie Mellon University
Software Engineering Institute

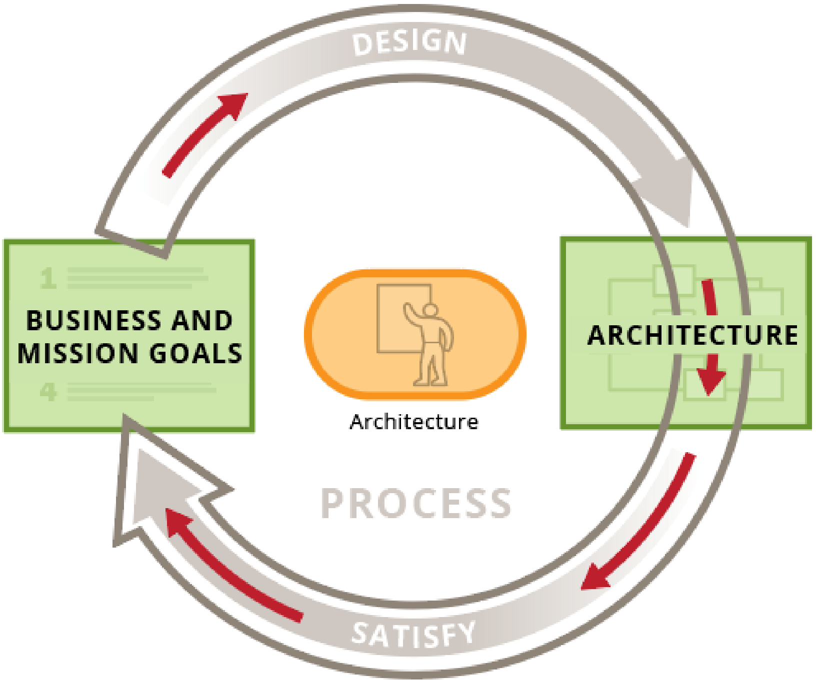
RESEARCH REVIEW 2020

Automated Design Conformance during Continuous Integration

Robert Nord, Ben Cohen, Shane Ficorilli, James Ivers, John Klein, Lena Pons, Chris Seifried



Software Architecture Enables Our Ability to Innovate

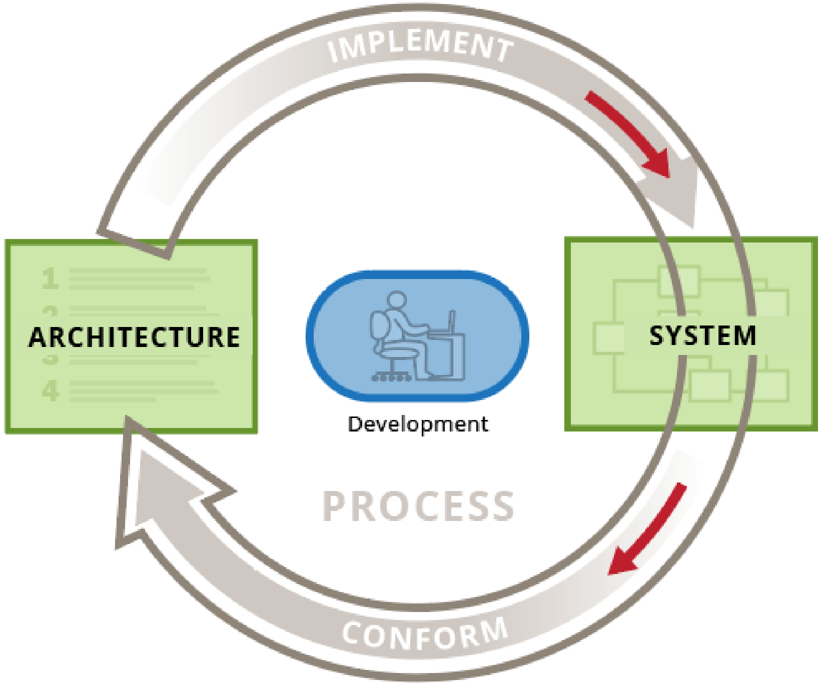


The software architecture community has evolved a body of knowledge that guides design and analysis.

This body of knowledge includes

- design principles
- reference architectures
- architectural design patterns
- deployment patterns
- tactics
- externally developed components

Implications for the System

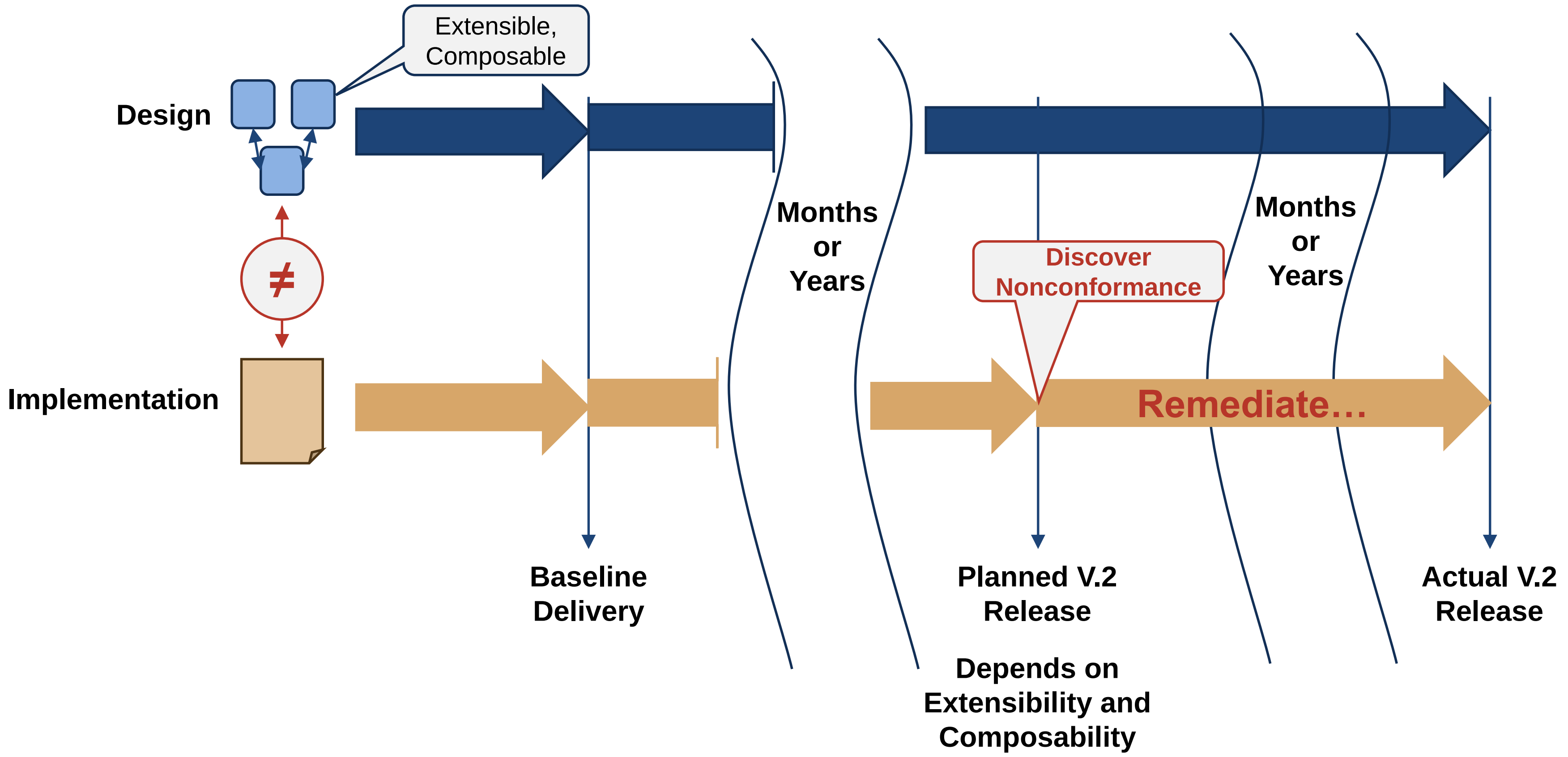


The degree to which a system meets its quality attribute requirements is dependent on architectural decisions.

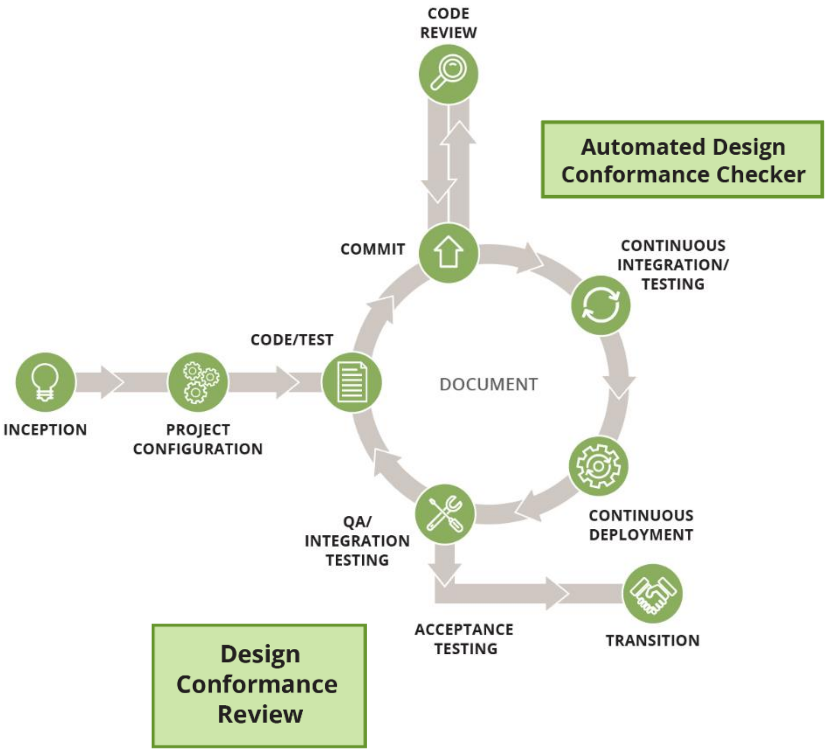
However, architecture can only permit, not guarantee, any quality attribute.

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

Software Nonconformance Problem



Project Objective

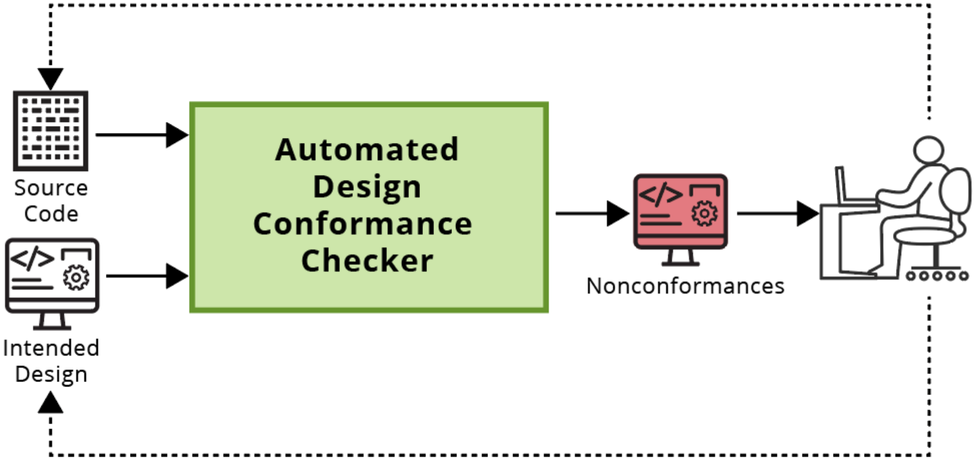


An automated design conformance checker integrated into a continuous integration workflow will reduce time to detect violations from months or years to hours.

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.

Create an Automated Design Conformance Checker

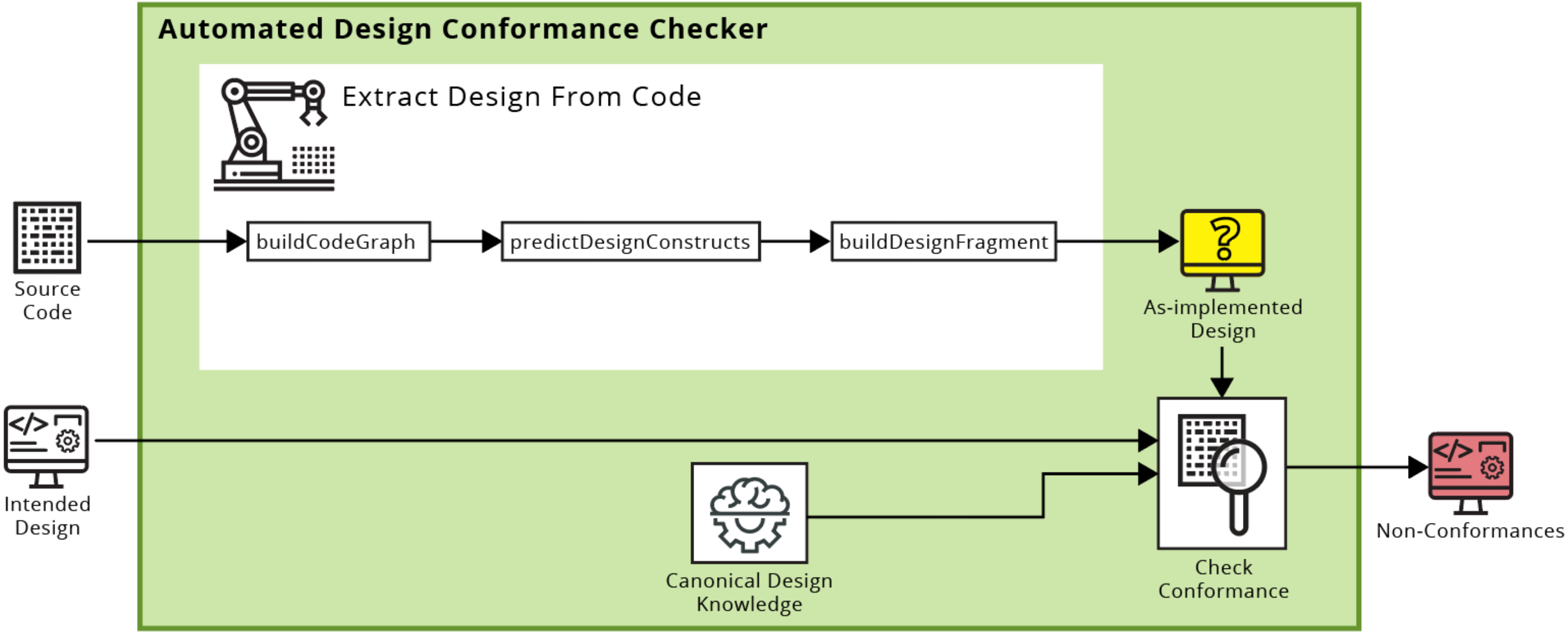


A design conformance checker automatically checks that the source code reflects the intended design and reports nonconformances.

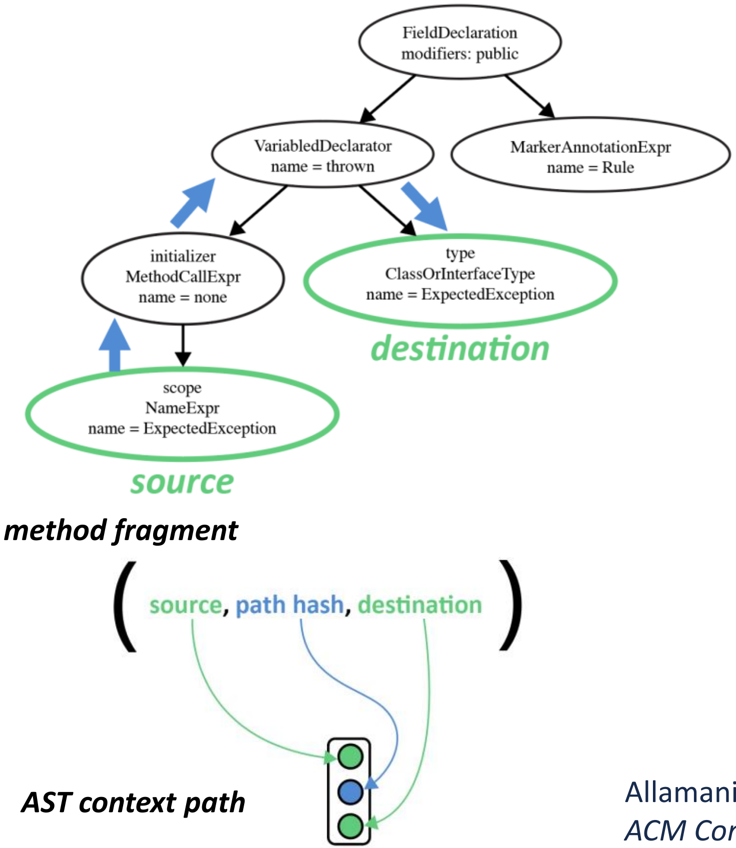
Recommendations correctly identify nonconformance, precision >.90 and detect at the commit that introduces nonconformance > 90%

Apply developer feedback to improve accuracy and significance within project contexts.

Building on Code Analysis, Software Architecture, Machine Learning, and Continuous Integration



Machine Learning for Source Code



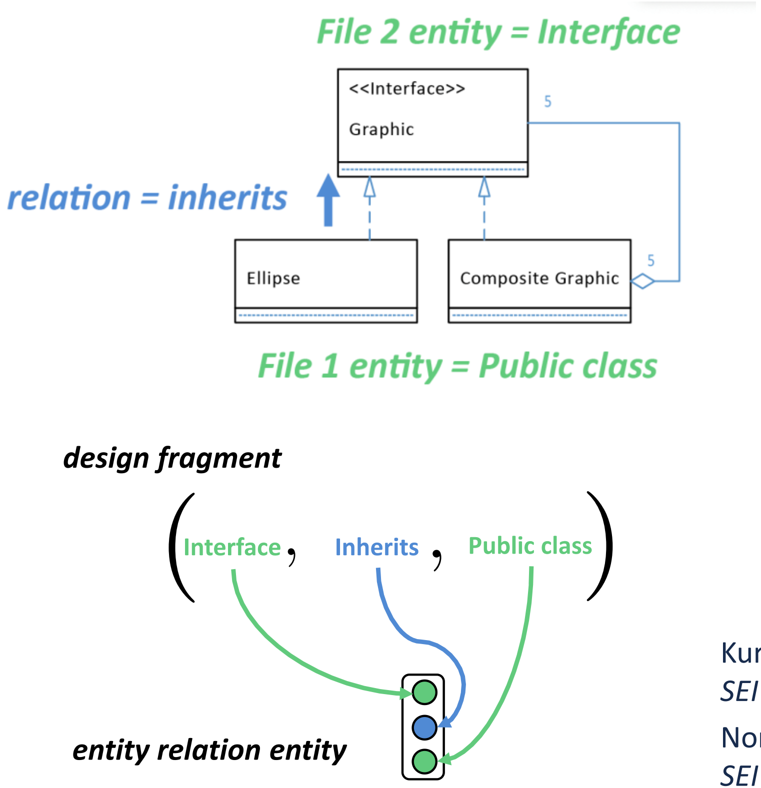
Initial applications applied off-the-shelf machine-learning tools with hand-extracted features.

Subsequent applications use the source code itself within machine learning drawing inspiration from natural language processing (NLP).

Current applications promise new machine-learning models informed by programming-language semantics.

Allamanis et al. (2018). **A Survey of Machine Learning for Big Code and Naturalness**. *ACM Computing Surveys, Article No 81.*

What Is a Good Feature Set for Architecture Design?



Representing the code as a graph of entities provides insight into structure.

Modeling context in the form of relations can improve prediction performance.

Design constructs are traceable to a wider range or larger portion of the code base.

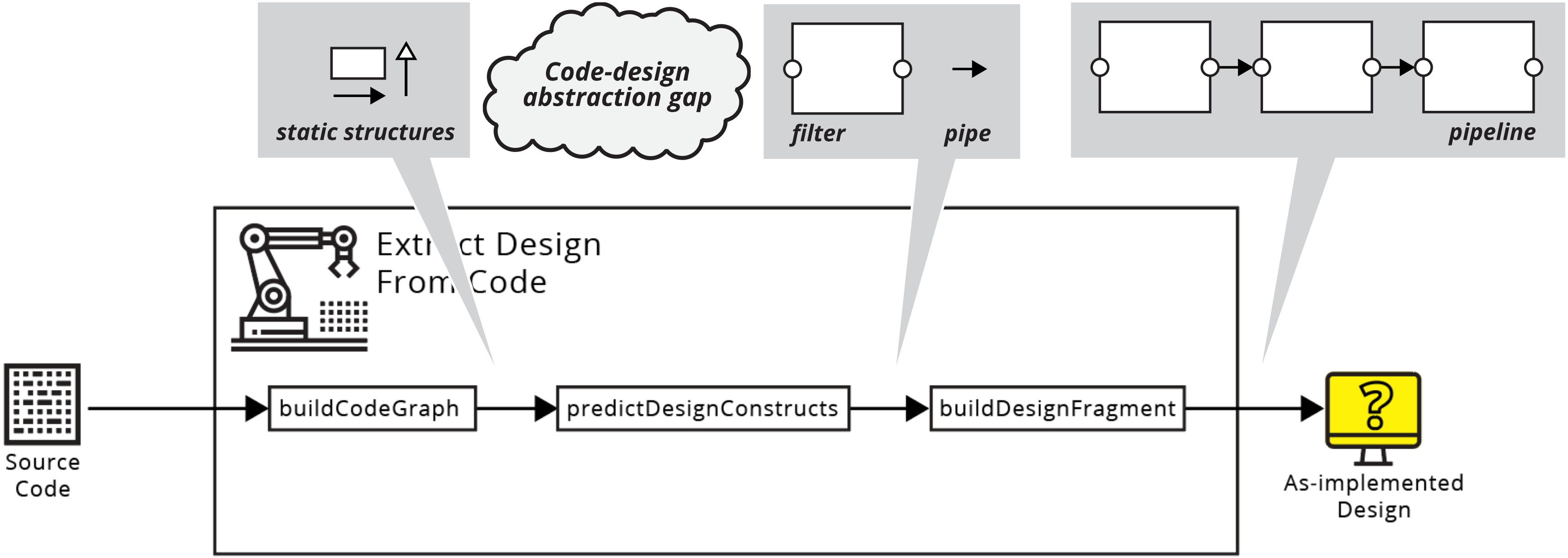
Common intermediate representation of object-oriented design.

Extensible representations allow data from different sources to be integrated.

Kurz (2019). **The Vectors of Code: On Machine Learning for Software.** *SEI Blog, Software Engineering Institute, Carnegie Mellon University.*

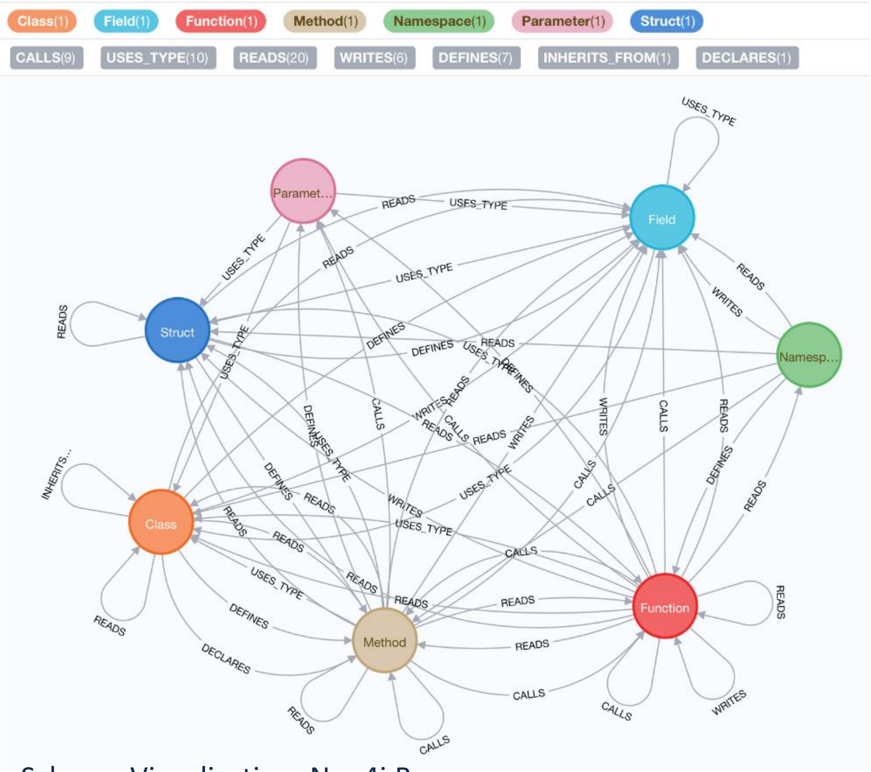
Nord (2020). **Using Machine Learning to Detect Design Patterns.** *SEI Blog, Software Engineering Institute, Carnegie Mellon University.*

Code-Design Abstraction Gap



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).*

Build Graph



Schema Visualization: Neo4j Browser

Static code analysis tool extracts structural information from C++ object-oriented code.

Sample graph sizes:

Blobby Warriors github.com/visusnet/Blobby-Warriors

- **48K** code lines
- **8,799** nodes and **50,411** relations

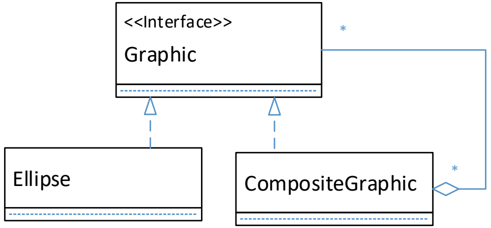
Hotspot (Qt framework) github.com/KDAB/hotspot

- **8K** code lines
- **2,648** nodes and **11,427** relations

Sample Industry System

- **1,587K** code lines
- **274,199** nodes and **1,057,595** relations

Engineer Features




```
class Graphic {
    //Prints the graphic.
    public: virtual void print();
}

class Ellipse : Graphic {
    public: void print() {
        System.out.println("Ellipse");
    } }

class CompositeGraphic : Graphic {
    //Collection of child graphics.
    private: ArrayList<Graphic> mChildGraphics;
    …
    public: void print() {
        for (i = 0; i < LENGTH; i++){
        mChildGraphics[i]->print();
        } } …
}
```

Interface

Implements

Leaf Class

Implements

Same Interface Container

Redirect in Family

composite pattern

Structural and behavioral features link elements though relations.

Structural

- *Degree of Accessibility, Virtuality*

  $\underline{interfaceMethods}(c) =$

  $|\{m \mid m \in M_{DEC}(c) \wedge Public(m) \wedge Abstract(m)\}|$

- *Association*

  $\underline{methodParam}(c1,c2) =$

  $true\ iff\ \exists p \in PAR(M_{DEC}(c1)) \wedge Type(p, c2)$

Behavioral

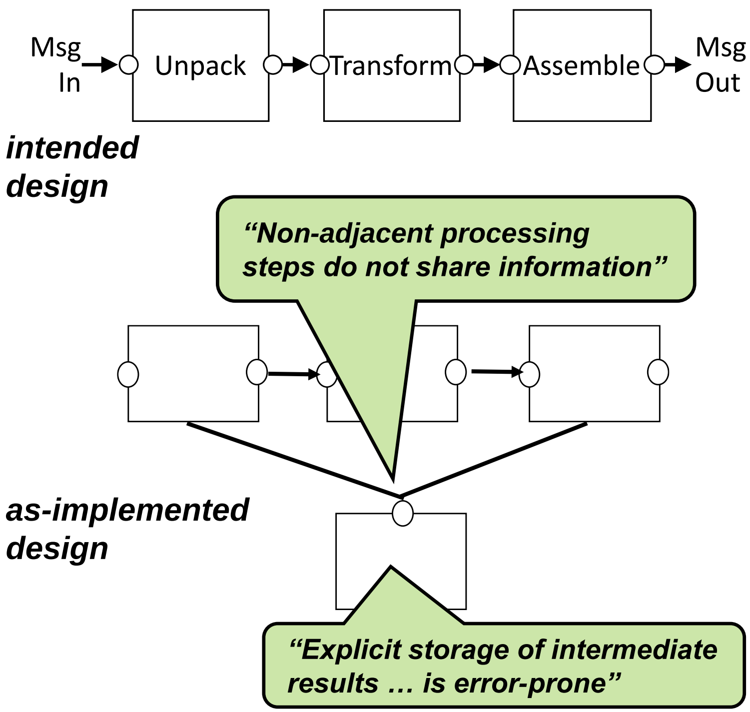
- *Invoker/invokee*

  toSibling$(c1, c2) =$

  $\{\exists c \in C \mid (Super(c1, c) \wedge Super(c2, c))\}$

Alhusain (2016). **Intelligent data-driven reverse engineering of software design patterns**. *PhD Thesis, De Montfort University.*

Check Conformance

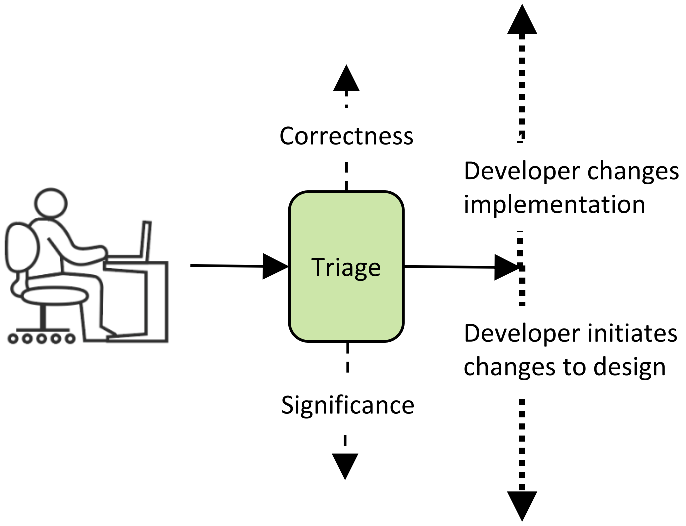


Design fragments represented as graphs enable automatic detection.

Detect nonconformances

- Check graph of extracted design fragment for agreement with intended design fragment to locate inconsistencies.
- Augment with checks against canonical design knowledge relevant to design fragment.

Continuous Integration Workflow



Detecting nonconformances produces greatest value when issues are exposed close to the time of injection.

Integrate with Jenkins CI tool to enable an empirical evaluation of the use of automation.

Use developer feedback in rating each nonconformance to improve results.

- *Correctness*—Improve design extraction by providing new labeled data.
- *Significance*—Improve adaptive filtering by capturing context-specific rules and exceptions.

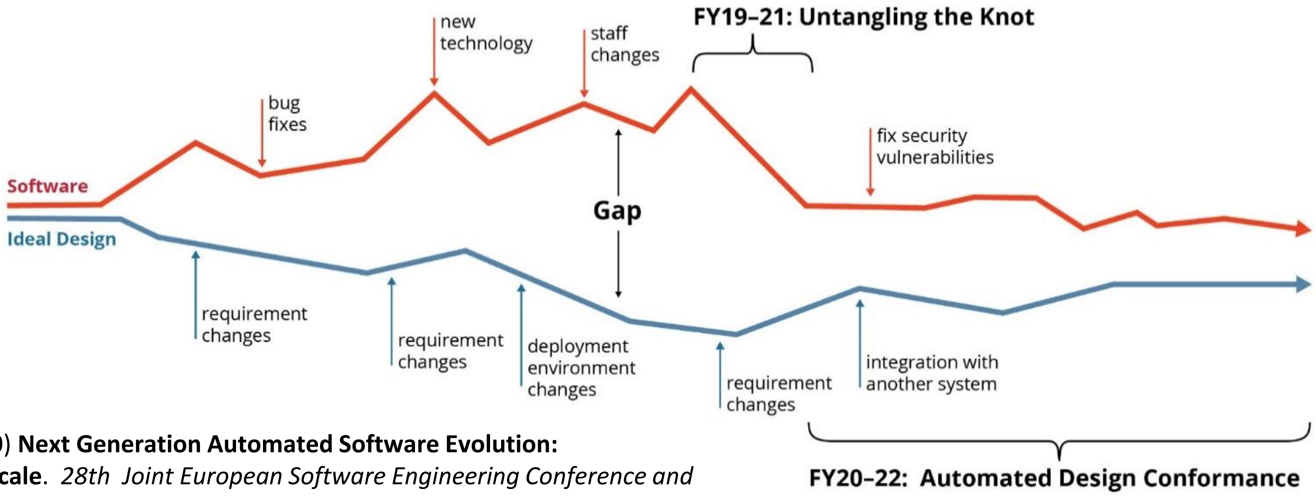
Looking Ahead

FY20	FY21	FY22
• Build out infrastructure: representation, features, design knowledge, and conformance. • Assemble open source data and initial analyses.	• Broaden the palette: more design knowledge and conformance checks. • Implement adaptive filtering.	• Fine-tune conformance checking. • Validate with experienced developers. • Ready to pilot conformance checking for C++ software.

Next Generation Automation for Software Evolution

Advance the state of the art where automation can

- keep software aligned with needs
- bring projects back into alignment
- realize changes sketched by developers in the language of design



Ivers et al. (2020) **Next Generation Automated Software Evolution: Refactoring at Scale**. *28th Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE ‘20). ACM.*

Project Team Members







Robert Nord
Ben Cohen
Shane Ficorilli
James Ivers

John Klein
Lena Pons
Chris Seifried