UTSA Computer Science

UTSA

I·C·S The Institute for Cyber Security

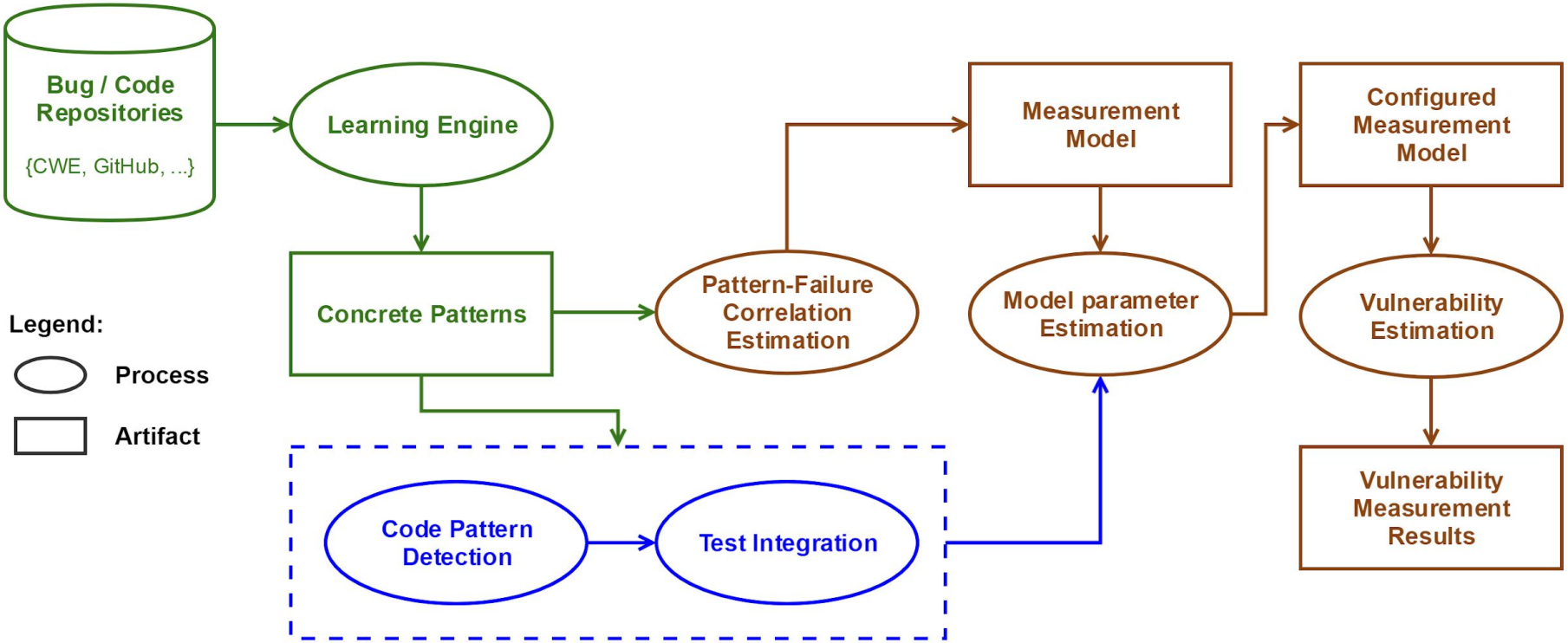
Toward a Pattern-Based Measurement Model for Improving Software Reliability

Presenter: John Heaps
University of Texas at San Antonio

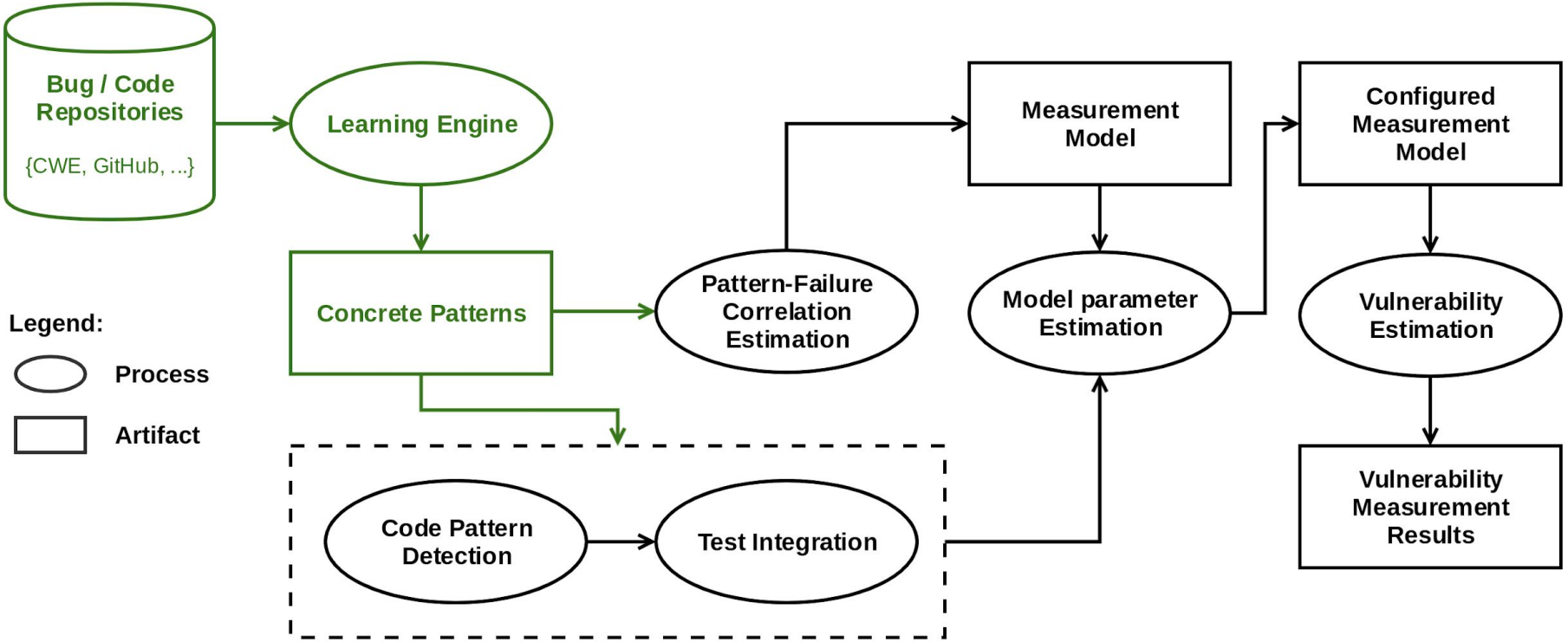
Researchers: John Heaps, Rodney Rodriguez, Xiaoyin Wang, and Jianwei Niu

Goal

- The reliability of software systems is essential in making decisions on real-world problems, but how do we determine the reliability of those systems?
- We propose a framework to detect bugs based on code pattern detection
- Our empirical analysis-based framework will mine and generate bug patterns, detect those patterns in code, and calculate a vulnerability measure
- Our framework will determine the level of reliability, affected by bugs, and allow stakeholders to make informed decisions about software


Bug / Code Repositories
{CWE, GitHub, ...}
Learning Engine
Concrete Patterns
Pattern-Failure Correlation Estimation
Measurement Model
Model parameter Estimation
Configured Measurement Model
Vulnerability Estimation
Vulnerability Measurement Results
Code Pattern Detection
Test Integration
Legend:
Process
Artifact

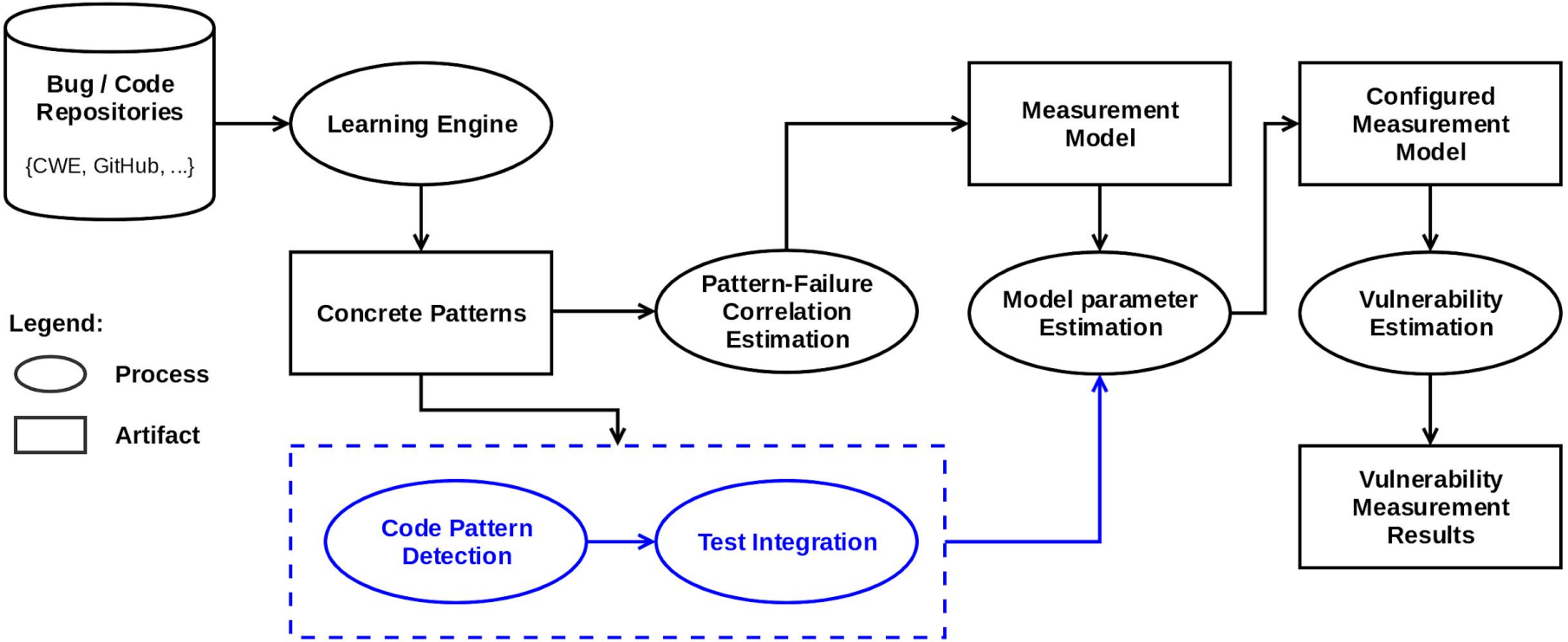
Learning Engine



Learning Engine

- Mine bug repositories (CWE, NVD, etc.) for bug categories, descriptions, and code snippet examples
- Create concrete patterns from mined bugs
- Link patterns to abstract qualities in order to perform reliability calculations
  - data, behavior, performance, security, and design

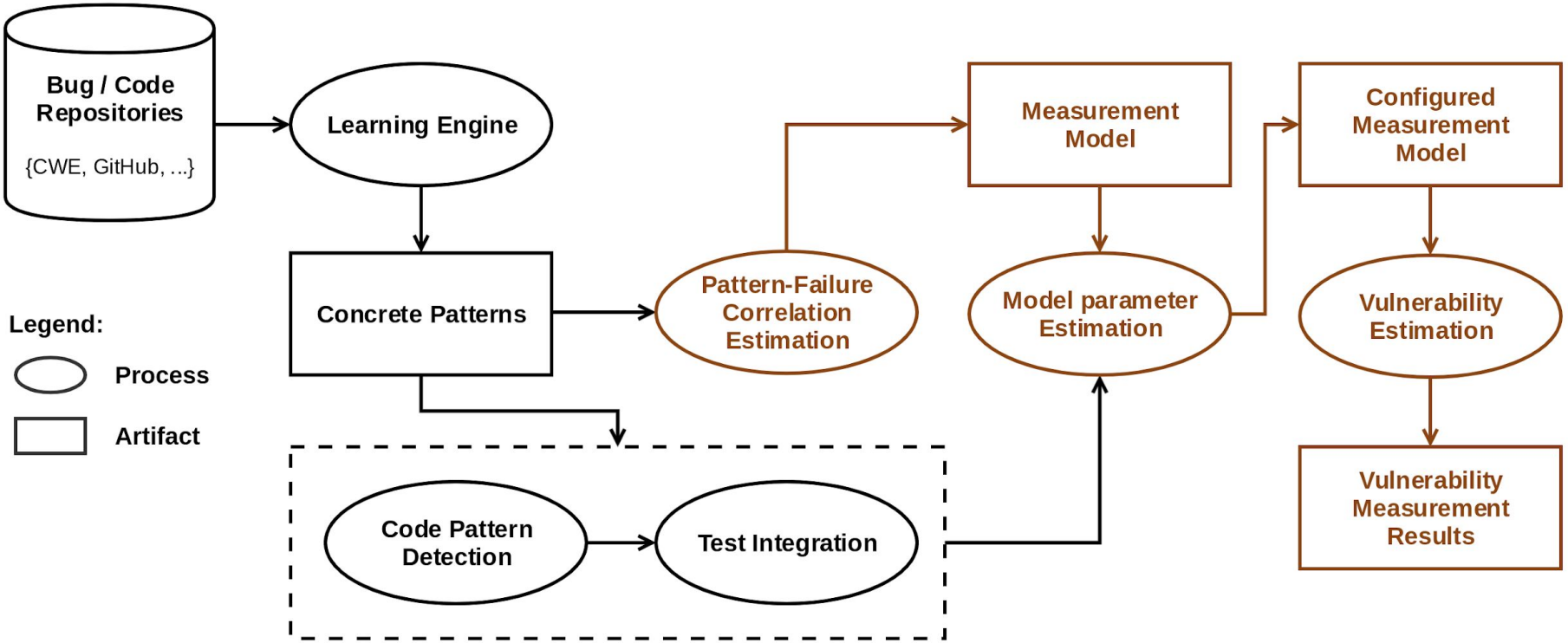
Pattern Detection and Test Integrator



Pattern Detection and Test Integrator

- Concrete bug patterns used to detect bugs in code using SpotBugs
- Identification of a bug pattern only indicates a strong **possibility** of a bug, not the existence of one
- Will use targeted testing on parts of code that contain bug patterns to determine if the bug can actually be triggered
  - If it can't, it is weighted less when calculating Reliability
  - If it can, it is weighted more when calculating Reliability

Reliability Model



Reliability Model

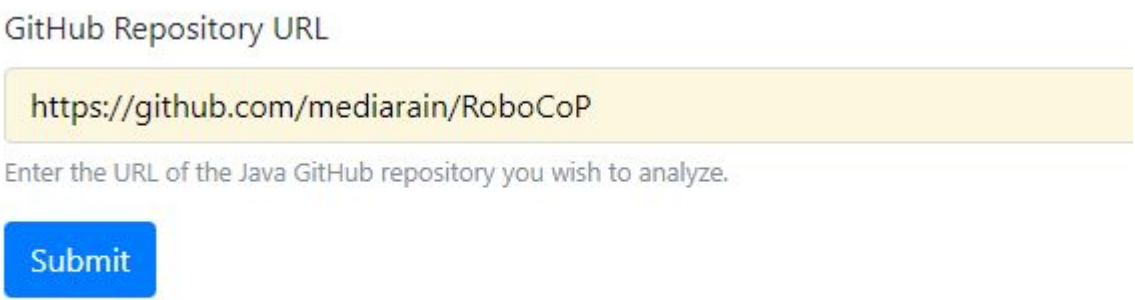
- Abstract qualities across all identified bugs are used to calculate Reliability
- *Impact* is a weighted sum of the abstract quality values of a bug
- Susceptibility is the weight determined by testing
- *R* is the average risk score of a software
  - Computed by taking the average risk of 763 top Java GitHub projects

$$Reliability = \frac{R}{R + \sum_{Detected} Risk(b)}$$

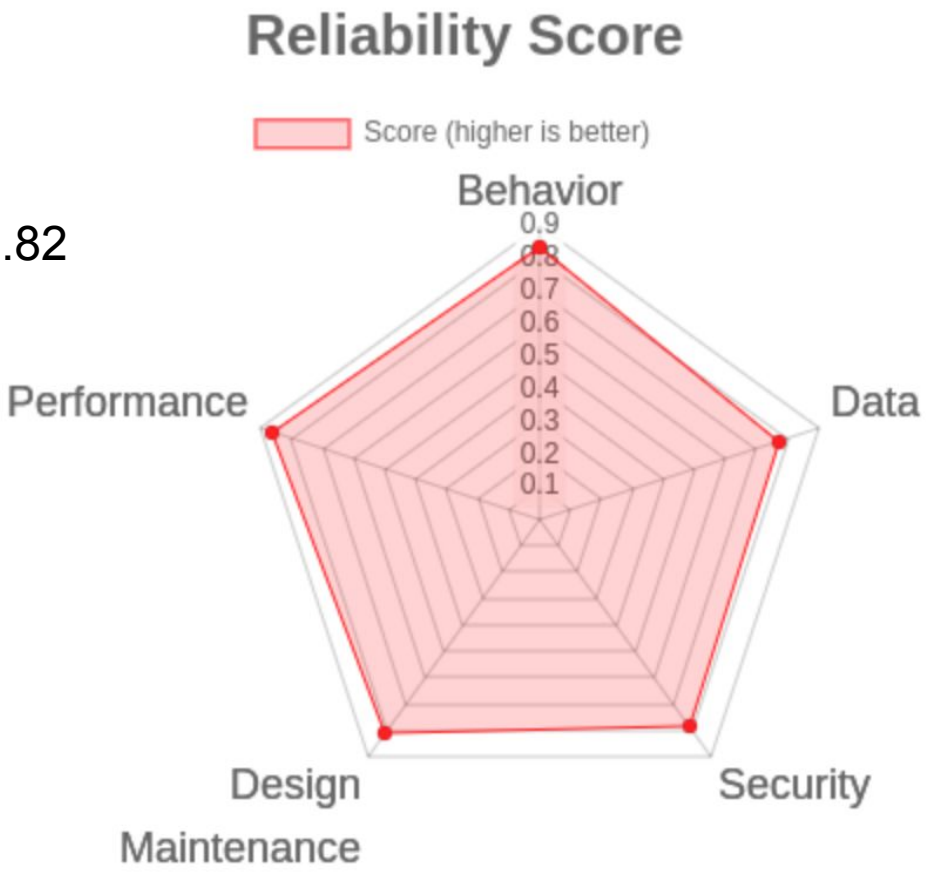
$$Risk = Impact * Susceptibility$$

Results

- http://galadriel.cs.utsa.edu:25666/



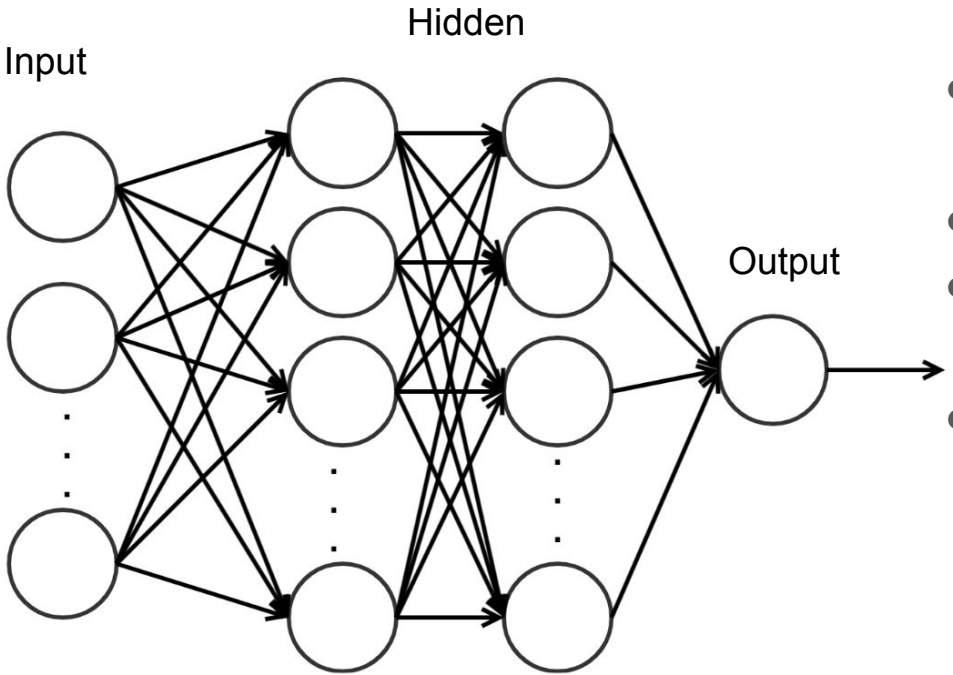
Reliability = .82



Observation of Practicality

- The biggest detriment to our approach (and every measurement model we found) is that many process must be done manually:
  - Creating concrete patterns, linking bugs to abstract qualities, etc.
- We want to use deep learning to help automate these manual tasks
- However, much of these manual process directly deal with code which cannot be learned over directly
- We will create word embeddings for code elements at which point we can perform learning tasks on code directly

Neural Networks



- Given an Input
- Matrix multiplication with Hidden layer (weights)
- Non-linear activation function (tanh, sigmoid, etc.)
- Predict Output
- Calculate loss by compare predicted label to actual label
- Update weights using backpropagation

Word Embeddings

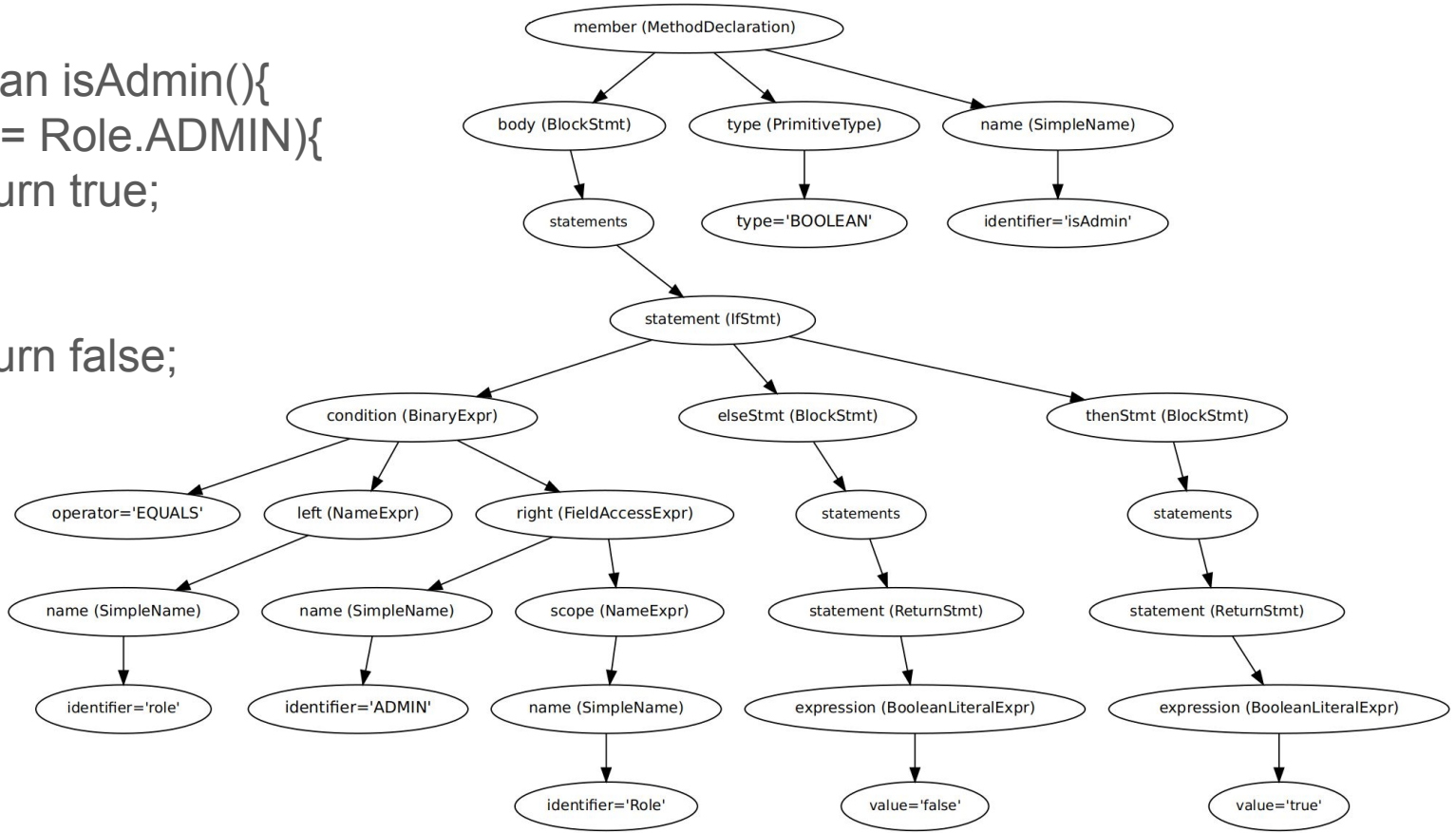
- Works very well for pictures since pixels have relevant values
- Text has no value to perform calculations with
- To solve this problem, we create real-valued vectors to represent each word in a corpus called word embeddings which can be used for calculations
- Given a *target* word in a corpus, we perform word prediction by using words surrounding the target, called *context*, to find relative semantic meaning between the words in the corpus
- When visualized, good word embeddings will have words with similar meaning grouped together

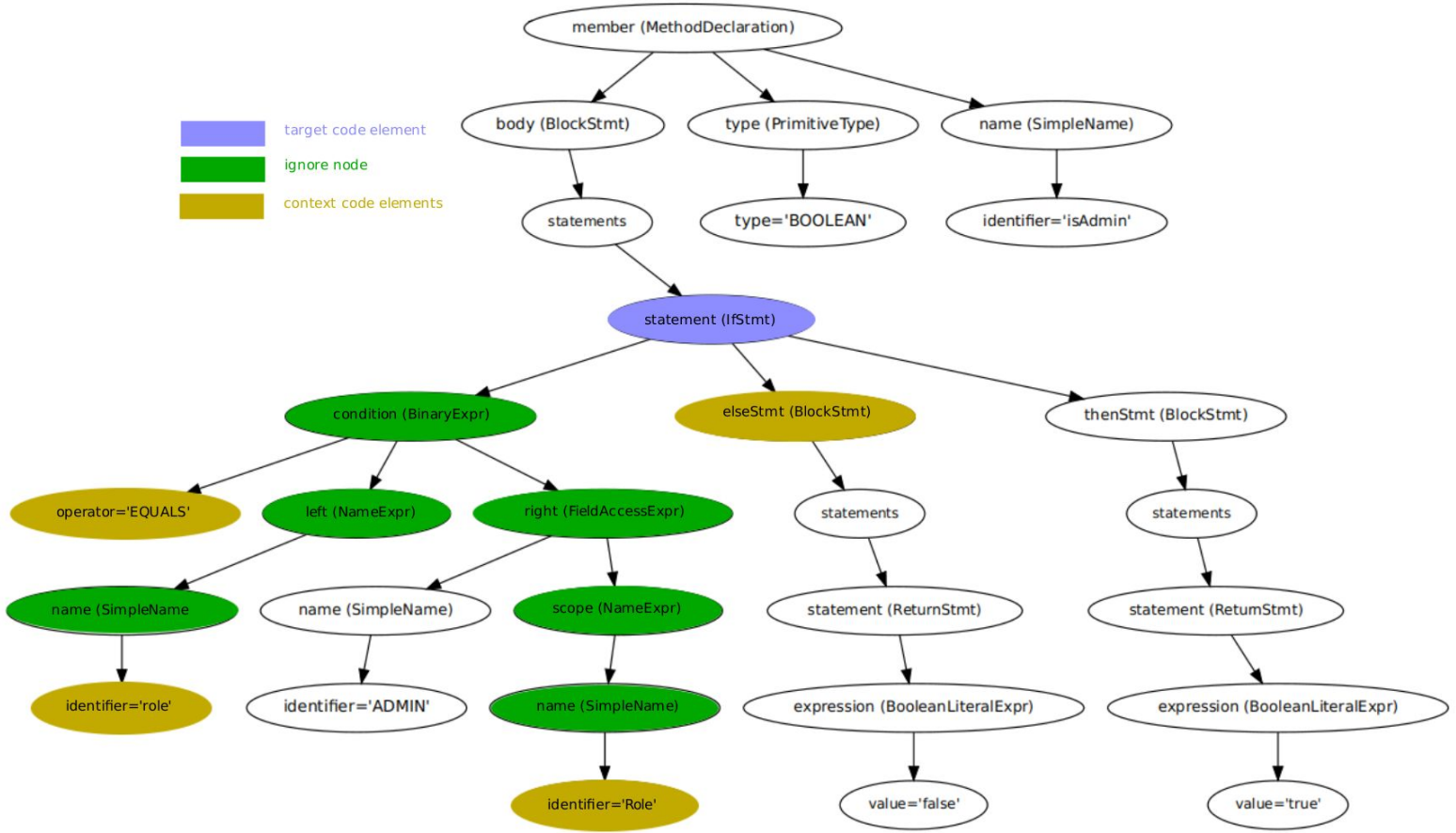
Word Embedding For Code

- We use the abstract syntax tree representation of code to make code exploration and analysis easier

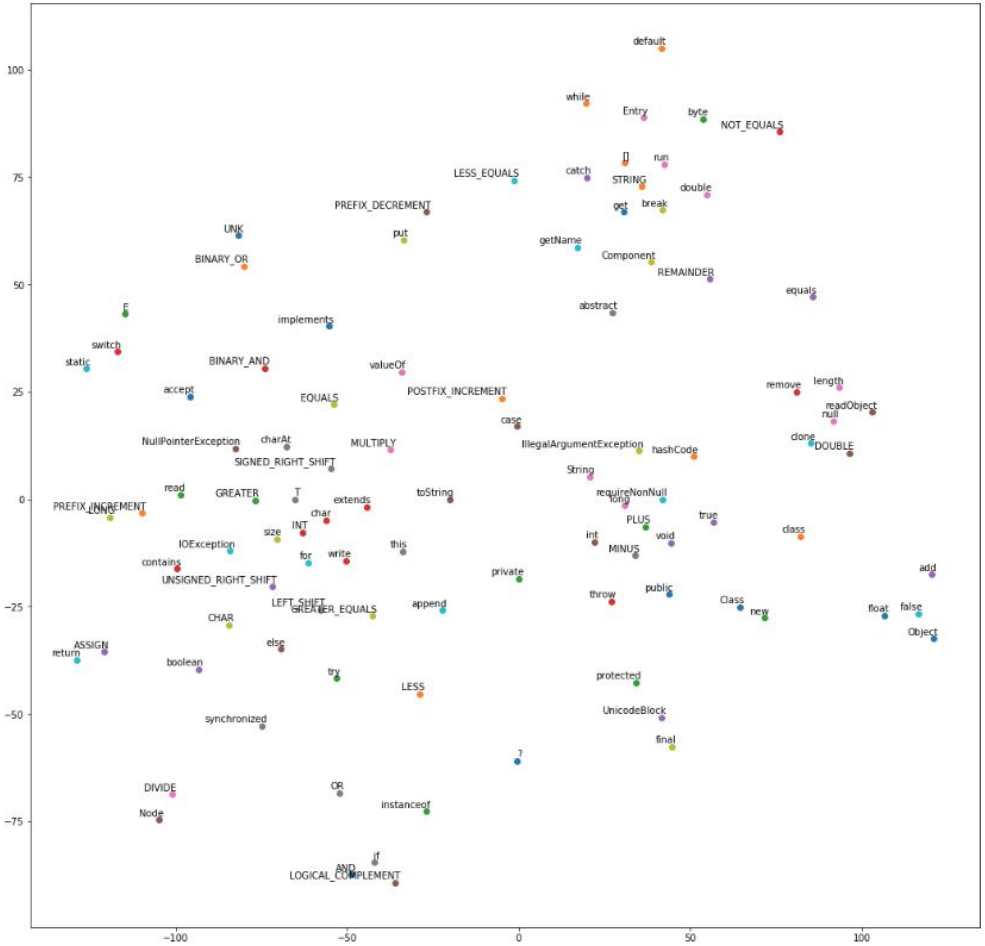
- We identify context based on asking the question “What other node types or code elements are needed to determine the meaning or perform the task of this node type?”

```
public boolean isAdmin(){
        if(role == Role.ADMIN){
                return true;
        }
        else{
                return false;
        }
}
```

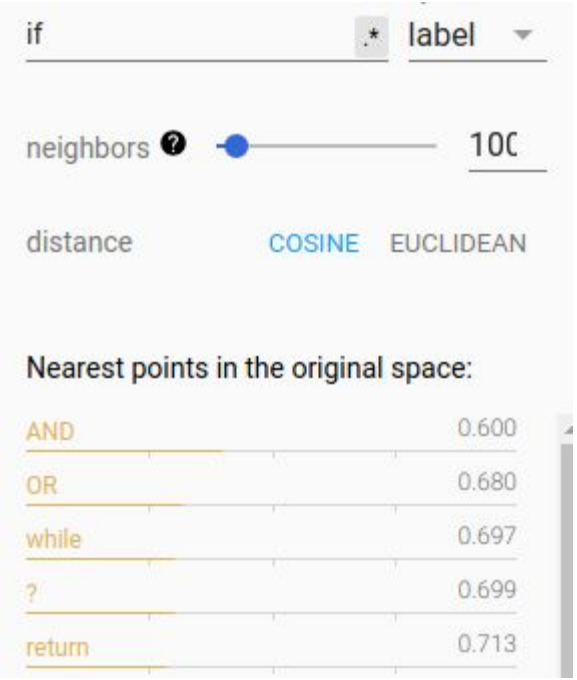



target code element
ignore node
context code elements
member (MethodDeclaration)
body (BlockStmt)
type (PrimitiveType)
name (SimpleName)
statements
type='BOOLEAN'
identifier='isAdmin'
statement (IfStmt)
condition (BinaryExpr)
elseStmt (BlockStmt)
thenStmt (BlockStmt)
operator='EQUALS'
left (NameExpr)
right (FieldAccessExpr)
statements
statements
name (SimpleName
name (SimpleName)
scope (NameExpr)
statement (ReturnStmt)
statement (ReturnStmt)
identifier='role'
identifier='ADMIN'
name (SimpleName)
expression (BooleanLiteralExpr)
expression (BooleanLiteralExpr)
identifier='Role'
value='false'
value='true'

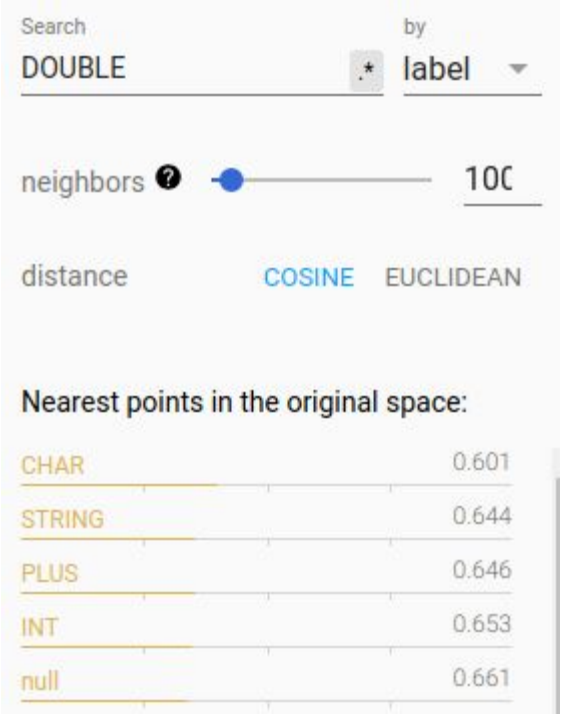
Lowest Average Loss: 2.69
Perplexity: 14.80



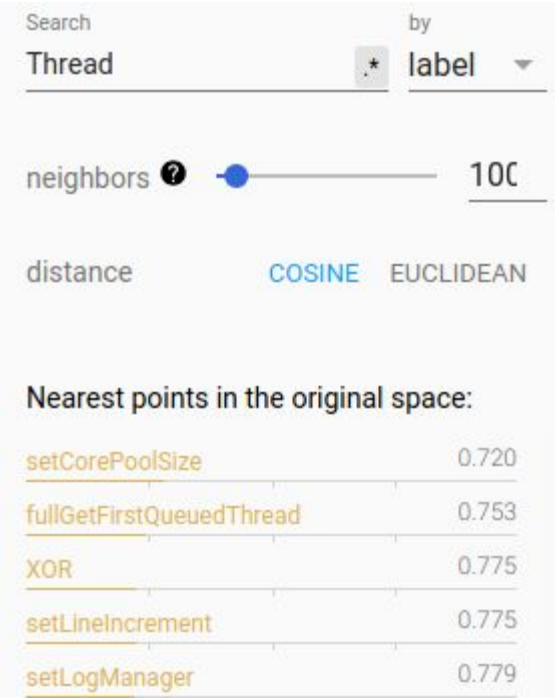
Top 5 closest embeddings to “if”



Top 5 closest embeddings to “DOUBLE”



Top 5 closest embeddings to “Thread”



Summary

- We proposed our framework for calculating a Reliability score on software by:
  - Collecting and creating bug patterns
  - Identifying those patterns in code
  - Calculating the reliability score based on the patterns identified

- We also explained our preliminary approach to mitigate the heavier manual process of the framework by using deep learning on code

Future Work

- Refine bug testing as it is still fairly preliminary
- Refine our word embedding approach
  - Especially how methods are processed
- Use the embeddings to perform learning tasks:
  - Classifying bugs to abstract concepts
  - Creating concrete patterns from bug code snippet examples