# Understanding Automatically-Generated Patches Through Symbolic Invariant Differences

Padraic Cashin, Carianne Martinez, Westley Weimer, Stephanie Forrest

# The Problem

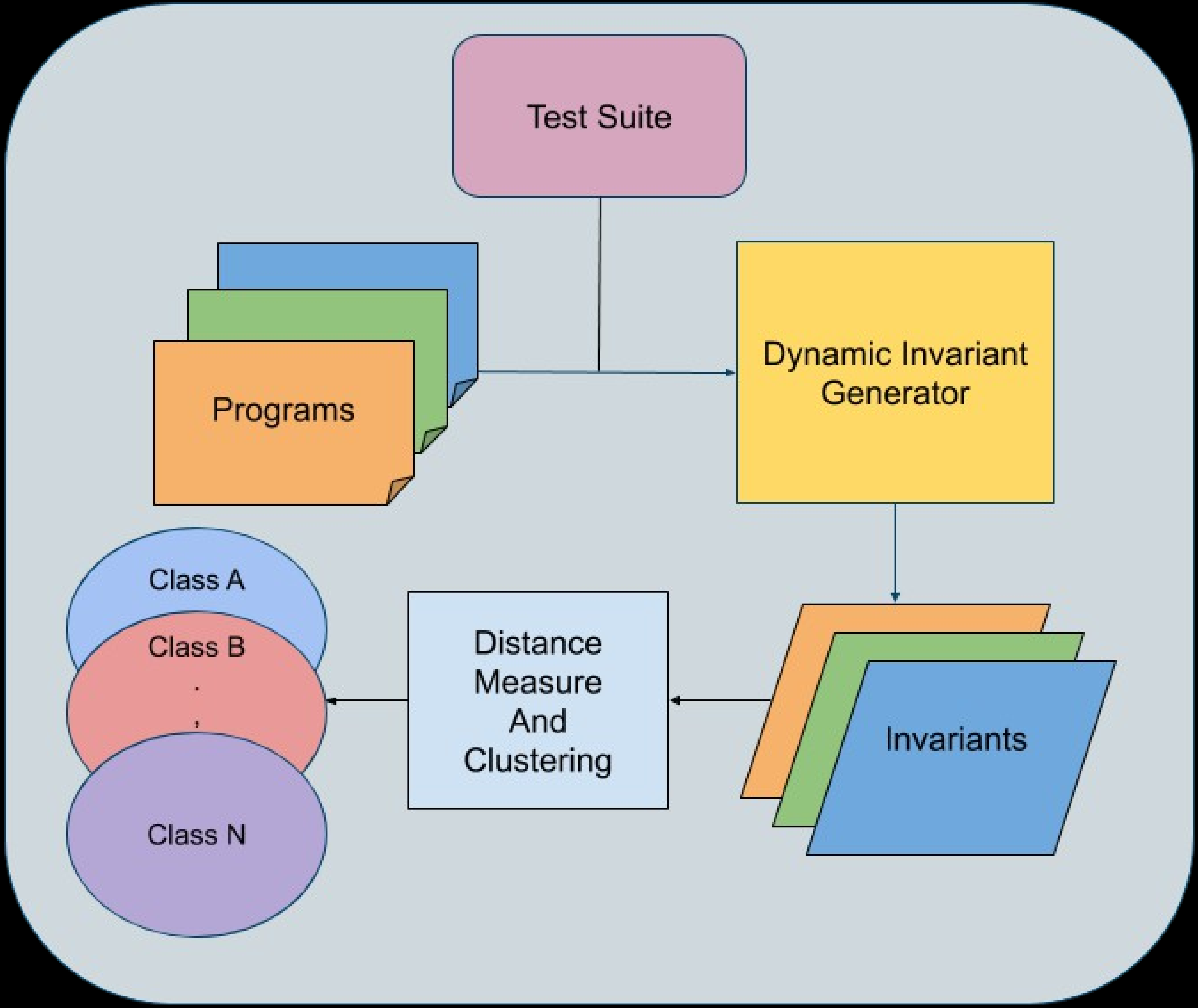
- Automated program repair may reduce software maintenance costs
  - Given a program and evidence of a bug, produce patches that fix that bug
  - SapFix, Angelix, Hercules, Prophet, Darjeeling, …
- A plausible patch passes local tests but *may or may not* be acceptable to developers
  - Assessing plausible patches takes time and effort
  - Can we reduce that manual analysis time?

# Patch Quality

- Many quality properties influence human decisions to adopt patches
  - Readability, maintainability, trust, style, …
- In addition, there are functional correctness concerns related to overfitting
- Repair algorithms may incorporate techniques to produce more acceptable patches
  - (e.g., templates, restricted operators, consolidation, etc.)

# Patch Assessment

- Ultimately, generate-and-validate program repair may produce dozens of syntactically-unique patches for the same defect
- We propose to reduce this inspection burden
  - Characterize patches by their sets of formal invariants (i.e., their behavior)
  - Calculate a distance metric on invariant sets
  - Cluster invariant sets (and thus patches) into equivalence classes
  - Only inspect one patch of each equivalence class


Test Suite
Programs
Dynamic Invariant Generator
Class A
Class B
Class N
Distance Measure And Clustering
Invariants

# Comparing Invariant Sets

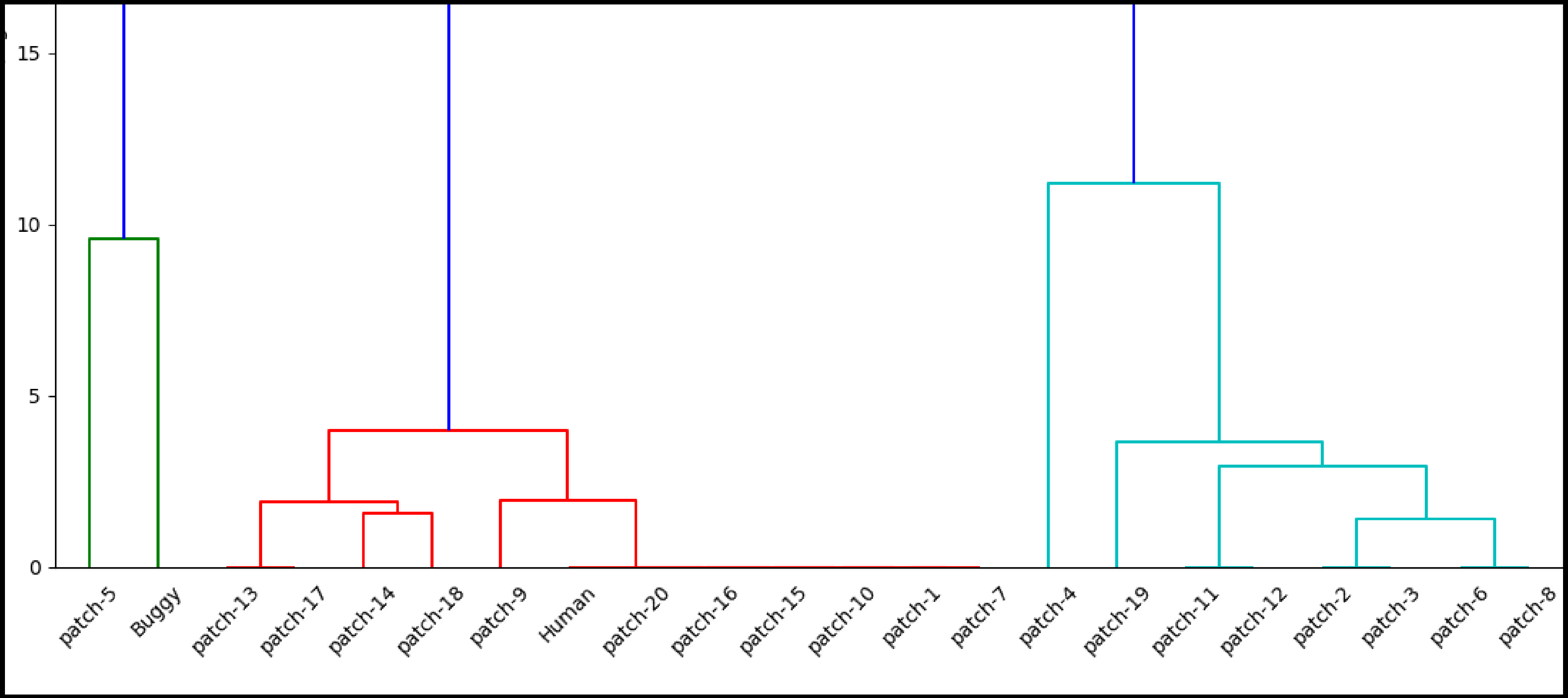
- Relaxes standard set difference from requiring equivalence to requiring logical implication
- Given programs A and B, tests T and invariant sets AI and BI
- We define the implication distance to be the cardinality of the subset of invariants in BI that are *not implied* by any invariant in AI
  - This definition admits hierarchical clustering
  - Optimization: consider only minterms from AI

# Efficient Invariant Comparison

- We also consider a more syntactic notion of distance on invariant sets
- We map syntactically-identical invariants to the same logical alphabet symbol
  - "X=2" is A, "X=2" is A, "X=1+1" is B, etc.
- And then calculate the Levenshtein edit distance on the induced strings
  - Efficient polytime computation (cf. Z3)

# Results & Conclusion

- Applied to 7 Defects4J and 5 ManyBugs bugs
  - 20-50 patches each from multiple tools




- Reduces manual inspection burden by 40-50%
  - Fast string-based distance has 95% accuracy