Users as Oracles: Semi-automatically Corroborating User Feedback

Andy Podgurski (with Vinay Augustine)
Electrical Eng. & Computer Science Dept.
Case Western Reserve University
Cleveland, Ohio

User Failure Reporting

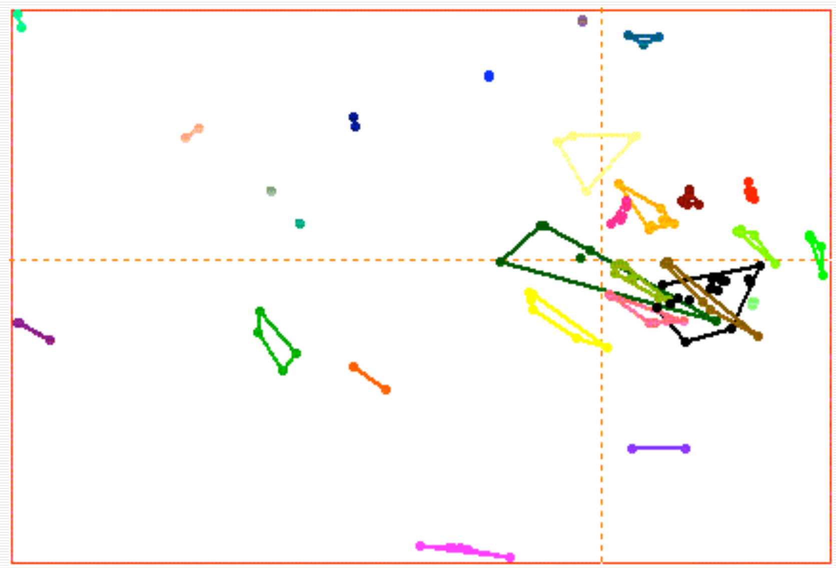
- Semi-automatic crash reporting is now commonplace
  - Report contains “mini-dump”
  - Facilitates grouping and prioritization
- Similar mechanisms for reporting “soft” failures are not
  - Would employ users as oracles
  - Would facilitate automatic failure classification and fault localization

Issue: Users Are Unreliable Oracles

- They overlook real failures
- They report spurious ones
  - Often misunderstand product functionality
- Developers don’t want to waste time investigating bogus reports

Handling Noisy User Labels: Corroboration-Based Filtering (CBF)

- Exploits user labels
- Seeks to corroborate them by pooling similar executions
  - Executions profiled and clustered
- Developers review only "suspect" executions:
  - Labeled FAILURE by users or
  - Close to confirmed failures or
  - Have unusual profile



Data Collection and Analysis

- Need four kinds of information about each beta execution:
  1. User label: SUCCESS or FAILURE
  2. Execution profile
  3. I/O history or capture/replay
  4. Diagnostic information, e.g.,
     - Internal event history
     - Capture/replay

Relevant Forms of Profiling

- Indicate or count runtime events that reflect causes/effects of failures, e.g.,
  - Function calls
  - Basic block executions
  - Conditional branches
  - Predicate outcomes
  - Information flows
  - Call sequences
  - States and state transitions

Filtering Rules

- All executions in small clusters ($|C| \leq T$) reviewed
- All executions with user label FAILURE reviewed
- All executions in clusters with confirmed failures reviewed

Empirical Evaluation of CBF

- Research issues:
  - How effective CBF is, as measured by
    - Number F_d of actual failures discovered
    - Number D_d of defects discovered
  - How costly CBF is, as measured by
    - Number R of executions reviewed by developers

Methodology

- CBF applied to test sets for three open source subject programs (actual failures known)
- Executions mislabeled randomly to simulate users
  - Mislabeling probability varied from 0 to 0.2
- For each subject program and test set, F_d, D_d, and R determined for
  - Three clusterings of the test executions:
    - 10%, 20%, 30% of test set size
  - Threshold $T = 1, 2, \ldots, 5$
- Same figures determined for three alternative techniques:
  - Cluster filtering with one-per-cluster (OPC) sampling
  - Review-all-failures (RAF) strategy
  - RAF+ extension of RAF
    - Additional executions selected for review randomly, until total is the same as for CBF

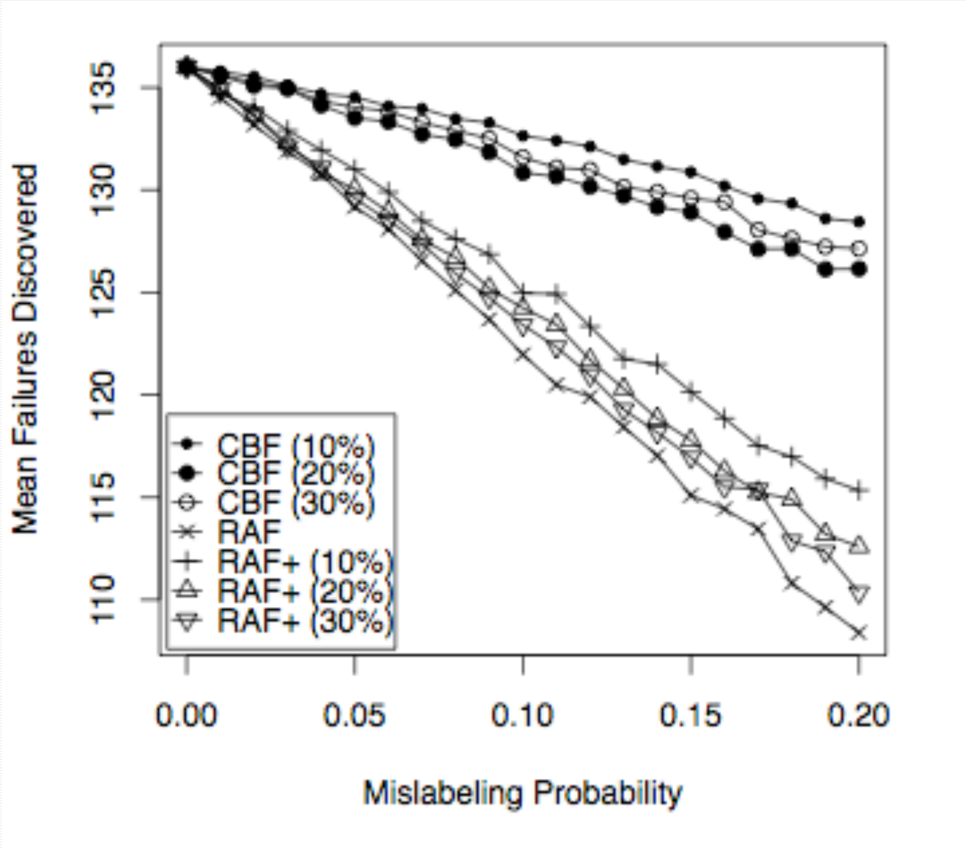
Subject Programs and Tests

- GCC compiler for *C* (version 2.45.2)
  - Ran GCC 3.0.2 tests that execute compiled code (3333 self-validating tests)
  - 136 failures due to 26 defects
- Javac compiler (build 1.3.1_02-b02)
  - Jacks test suite (3140 self-validating tests)
  - 233 failures due to 67 defects
- JTidy pretty printer (version 3)
  - 4000 HTML and XML files crawled from Web
  - Checked trigger conditions of known defects
  - 154 failures due to 8 defects
- Profiles: function call execution counts

Assumptions

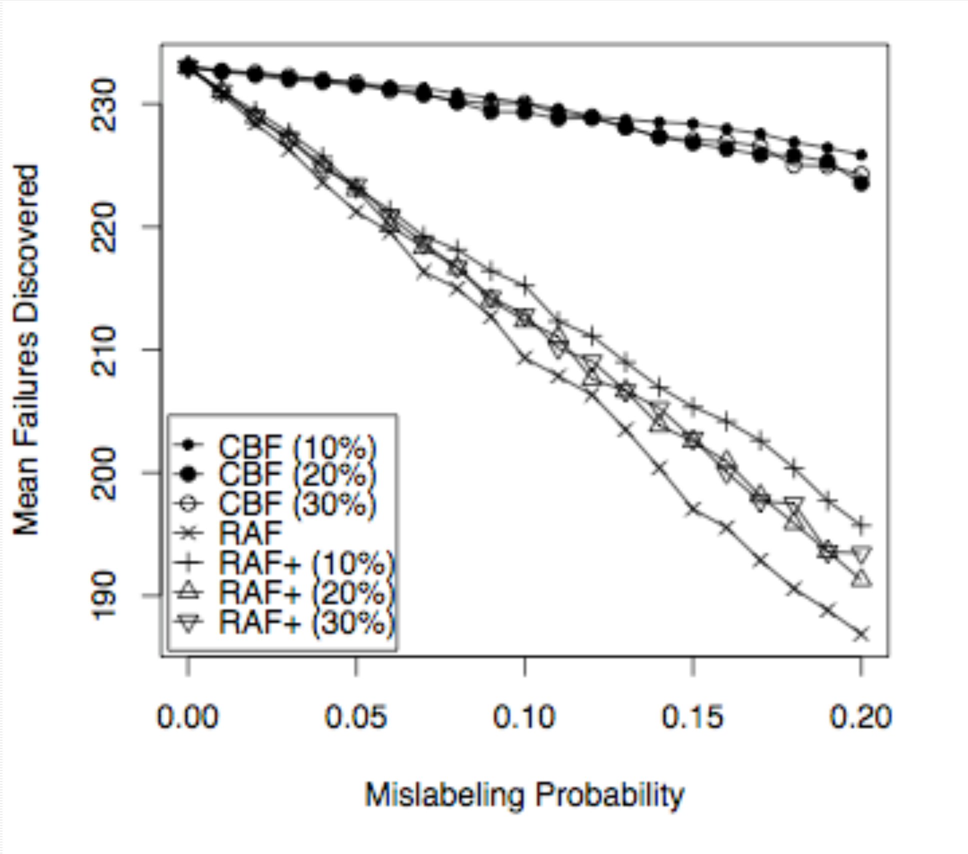
- Each actual failure selected would be recognized as such if reviewed
- The defect causing each such failure would be diagnosed with certainty

Mean Failures Discovered (b)



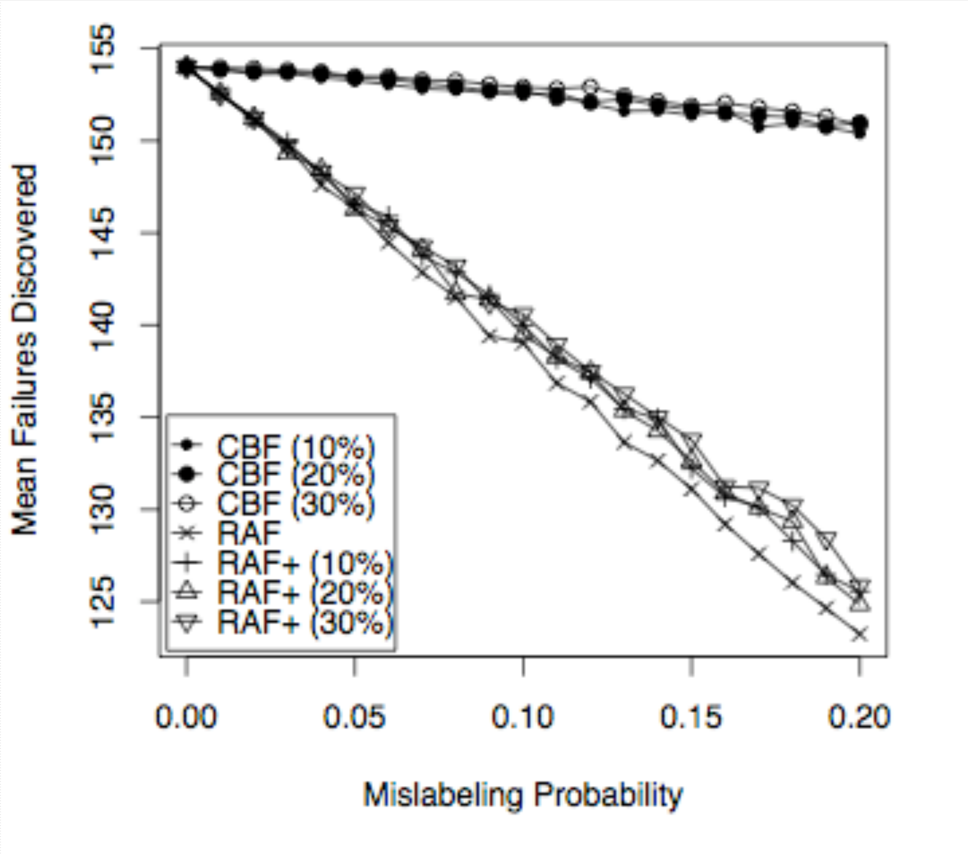
GCC (*T* = 1)

Mean Failures Discovered (c)



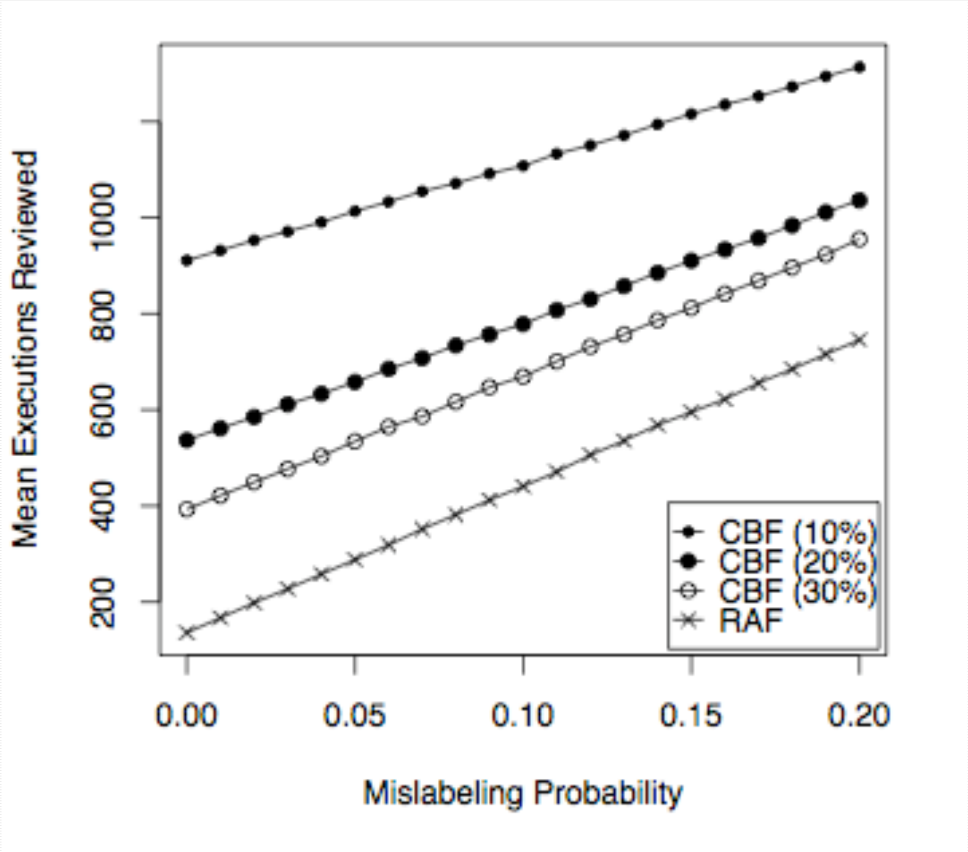
Javac (*T* = 1)

Mean Failures Discovered (d)



JTidy (*T* = 1)

Mean Executions Reviewed (b)



GCC (*T* = 1)

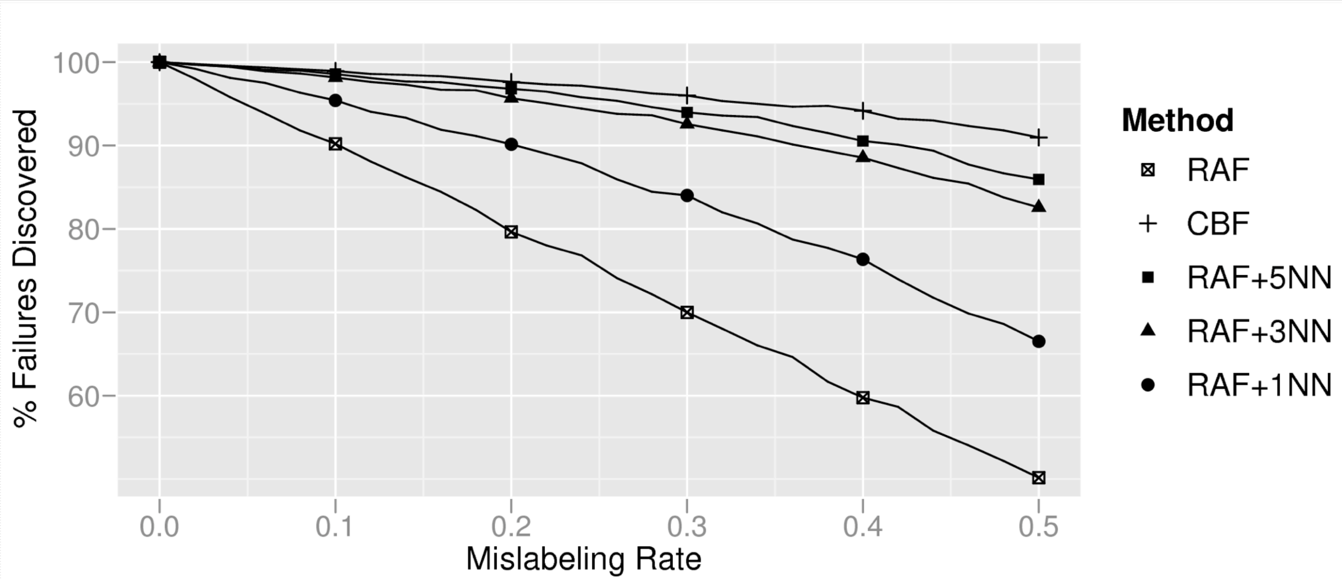
New Family of Techniques: RAF+*k*-Nearest-Neighbors (*k*NN)

- Compromise between low cost of RAF and power of CBF
- Require stronger evidence of failure than CBF
  - All executions with user label FAILURE reviewed
  - If actual failure confirmed, *k* nearest neighbors reviewed
  - Isolated SUCCESSes *not* reviewed

RAF+*k*NN: Executions Reviewed

Rome RSS/Atom Parser

RAF+*k*NN: Failures Discovered



JTidy

RAF+*k*NN: Defects Discovered

Subject	Method	10%	30%	50%
JTidy	CBF	7.99±.1	7.92±.27	7.73±.46
	RAF+3NN	7.91±.10	7.91±.29	7.73±.46
	RAF	7.91±.26	7.71±.46	7.55±.54
ROME	CBF	6±0	6±0	5.97±.17
	RAF+1NN	6±0	6±0	5.93±.26
	RAF	6±0	6±0	5.85±.36
Xerces	CBF	16.96±.28	16.80±.60	16.46±.89
	RAF+5NN	16.98±.20	16.62±.60	16.19±1.02
	RAF	16.96±.58	15.77±1.04	14.99±.89

Current & Future Work

- Further empirical study
  - Additional subject programs
  - Operational inputs
  - Alternative mislabeling models
  - Other forms of profiling
- Prioritization of executions for review
- Use of supervised and semi-supervised learners
- Multiple failures classes
- Exploiting structured user feedback
- Handling missing labels

Related Work

- Podgurski et al:
  - Observation-based testing
  - Cluster filtering and failure pursuit
  - Failure classification
- Michail and Xie: *Stabilizer* tool for avoiding bugs
- Chen et al: *Pinpoint* tool for problem determination
- Liblit et al: bug isolation
- Liu and Han: *R-proximity* metric
- Mao and Lu: priority-ranked *n*-per cluster sampling
- Gruschke; Yemini et al; Bouloutas et al: event correlation in distributed systems

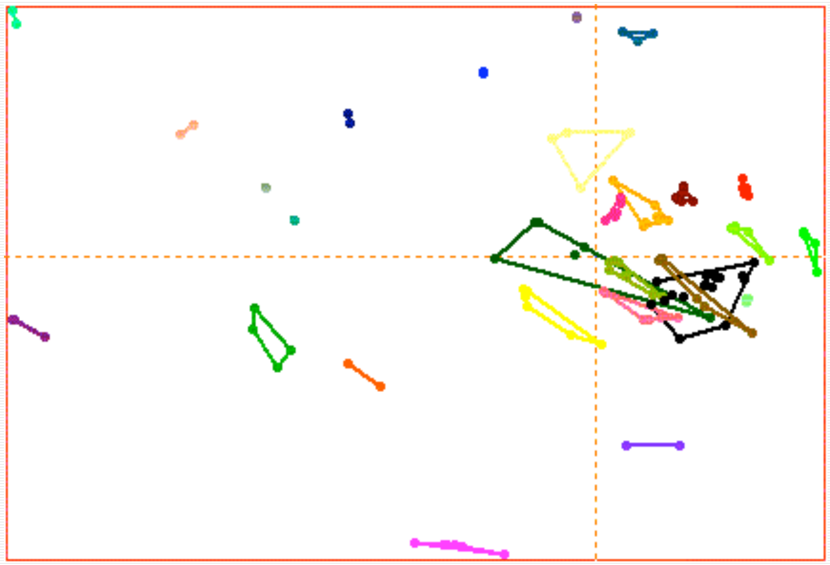
General Approach to Solution

- Record I/O online
  - Ideally with capture/replay tool
- Profile executions, online or offline
  - Capture/replay permits offline profiling
- Mine recorded data
- Provide guidance to developers concerning which executions to review

Approach #1: Cluster Filtering

[FSE 93, TOSEM 99, ICSE 01, … TSE 07]

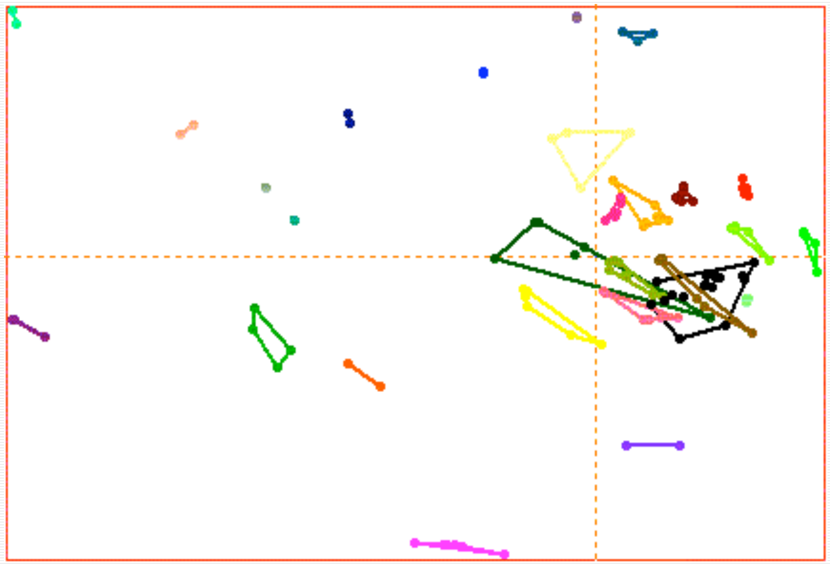
- Intended for beta testing
- Execution profiles automatically clustered
- 1+ are selected from each cluster or small clusters
- Developers replay and review sampled executions
- Empirical results:
  - Reveals more failures & defects than random sampling
  - Failures tend to be found in small clusters
  - Complements coverage maximization
  - Enables more accurate reliability estimation
- Not cheap
- Does not exploit user labels



Approach #2: Failure Classification

[ICSE 2003, ISSRE 2004]

- Goal is to group related failures
  - Prioritize and assist debugging
- Does exploit user labels
- Assumes they are accurate
- Combines
  - Supervised feature selection
  - Clustering
  - Visualization (MDS)
- Only failing executions clustered & visualized
- Empirical results:
  - Often groups failures with same cause together
  - Clusters can be refined using dendrogram and heuristics
- Does not exploit user labels



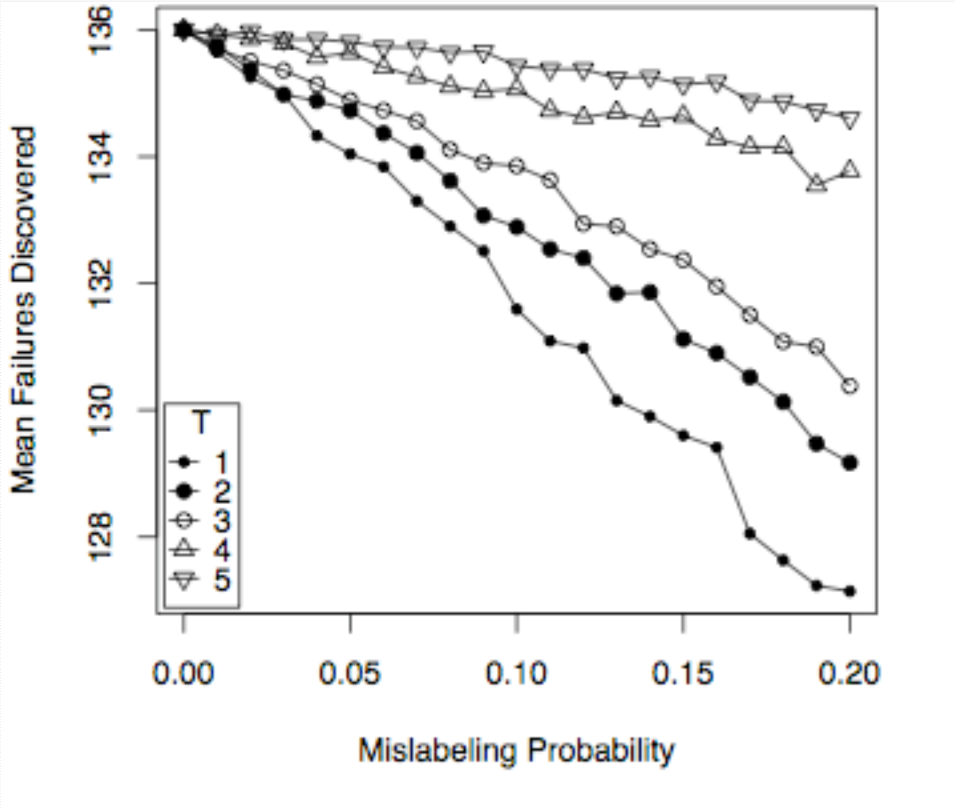
Data Analysis

- GNU *R* statistical package
- k-means clustering algorithm
  - Proportional binary dissimilarity metric

$$D_{n,m} = \sqrt{\sum_k \left(P_{n,k} - P_{m,k}\right)^2 + \left|B_{n,k} - B_{m,k}\right|}$$

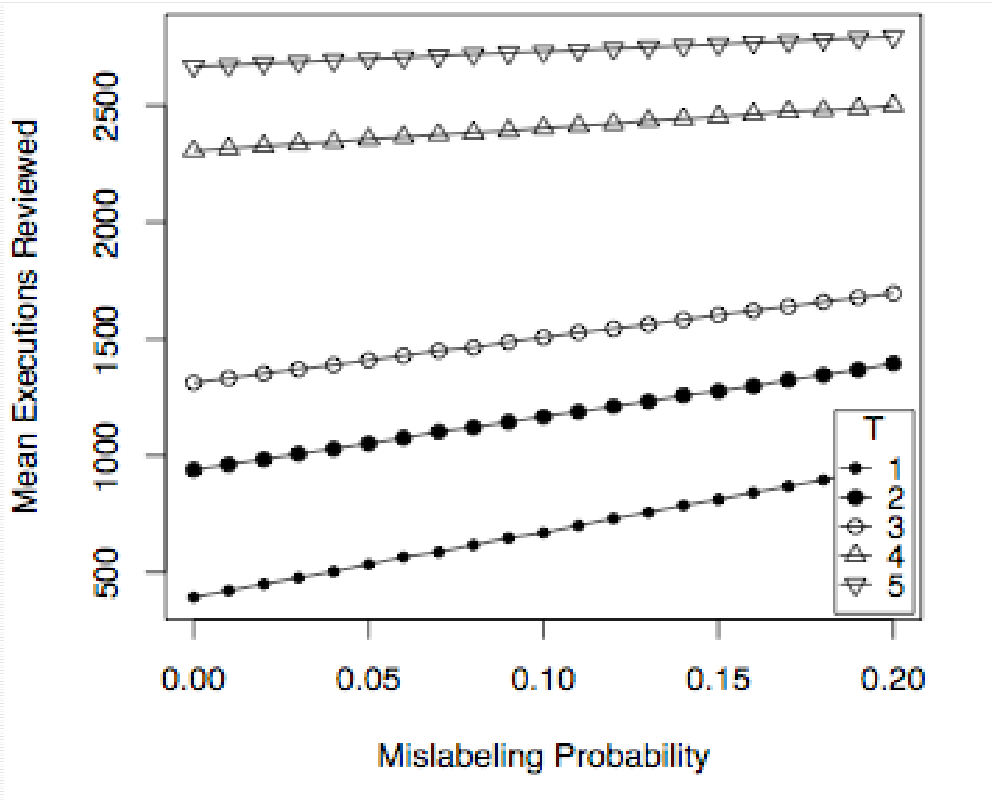
- CBF, RAF, RAF+ applied to 100 randomly generated mislabelings of test set
- OPC used to select 100 stratified random samples from each clustering
- Computed mean numbers of failures and defects discovered and executions reviewed

Mean Failures Discovered (a)



GCC (30% clustering)

Mean Executions Reviewed (a)



GCC (30% clustering)

Mean Failures Discovered with OPC Sampling

Program	Clustering		
	10%	*20%*	*30%*
GCC	13.98	26.04	48.16
Javac	30.36	58.51	88.85
JTidy	23.58	43.80	63.41

Analysis

- CBF with $T = 1$ revealed significantly more failures than RAF and OPC for all clusterings
  - Difference between CBF and RAF increased with mislabeling probability
- CBF entailed reviewing substantially more executions than RAF did
  - Held even with $T = 1$
  - Did not account for the additional failures discovered with CBF
- CBF and RAF each revealed most defects
  - OPC was less effective
  - RAF would not perform as well without "perfect" debugging