CARIAL: Cost-Aware Software Reliability Improvement with Active Learning

Boya Sun, Gang Shu, Andy Podgurski, Soumya Ray

Electrical Engineering and Computer Science
Case Western Reserve University
boya.sun@case.edu

Outline

- Motivation
- CARIAL Framework
  - Balancing cost and gain in operational testing
  - How to define and estimate **cost**?
  - How to define and estimate **gain**?
  - How to balance the two?
- Empirical study

Motivation

Synthetic and Operational Software Testing

- **Synthetic Testing**
  - Status of software: not ready for release
  - Test inputs: artificial tests; generated by programmers or tools
  - Goal: reveal defects
- **Operational Testing**
  - Status of software: almost ready for release
  - Test inputs: generated (and possibly captured) in field
  - Goal: reliability estimation and improvement

Problem

- **Scenario**
  - Software produces complex output, such as graphs, images, text
  - No *automatic* oracle available; much effort spent on test output evaluation
  - Thousands of operational tests available
  - Users are not “reliable oracles”
- **It is infeasible for programmers to run all tests and do test output evaluation**

Problem (continue)

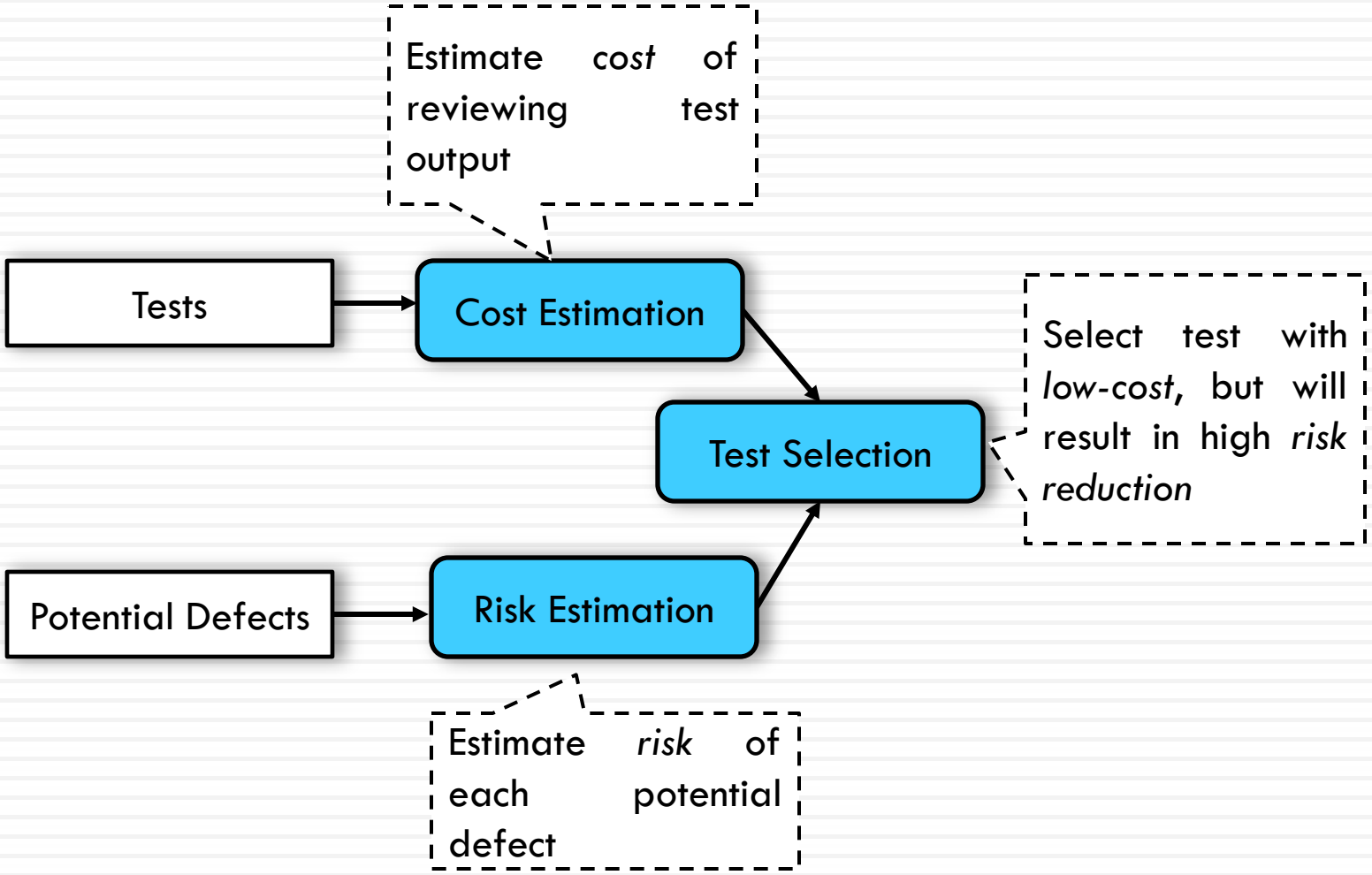
- **Cost**
  - Test output evaluation
- **Benefit**
  - Improvement of software reliability / reduction of risk
- **Problem**
  - How to get improvement of software reliability by evaluating fewer tests?
- **Solution**
  - CARIAL: Cost-Aware Reliability Improvement with Active Learning

The CARIAL Framework

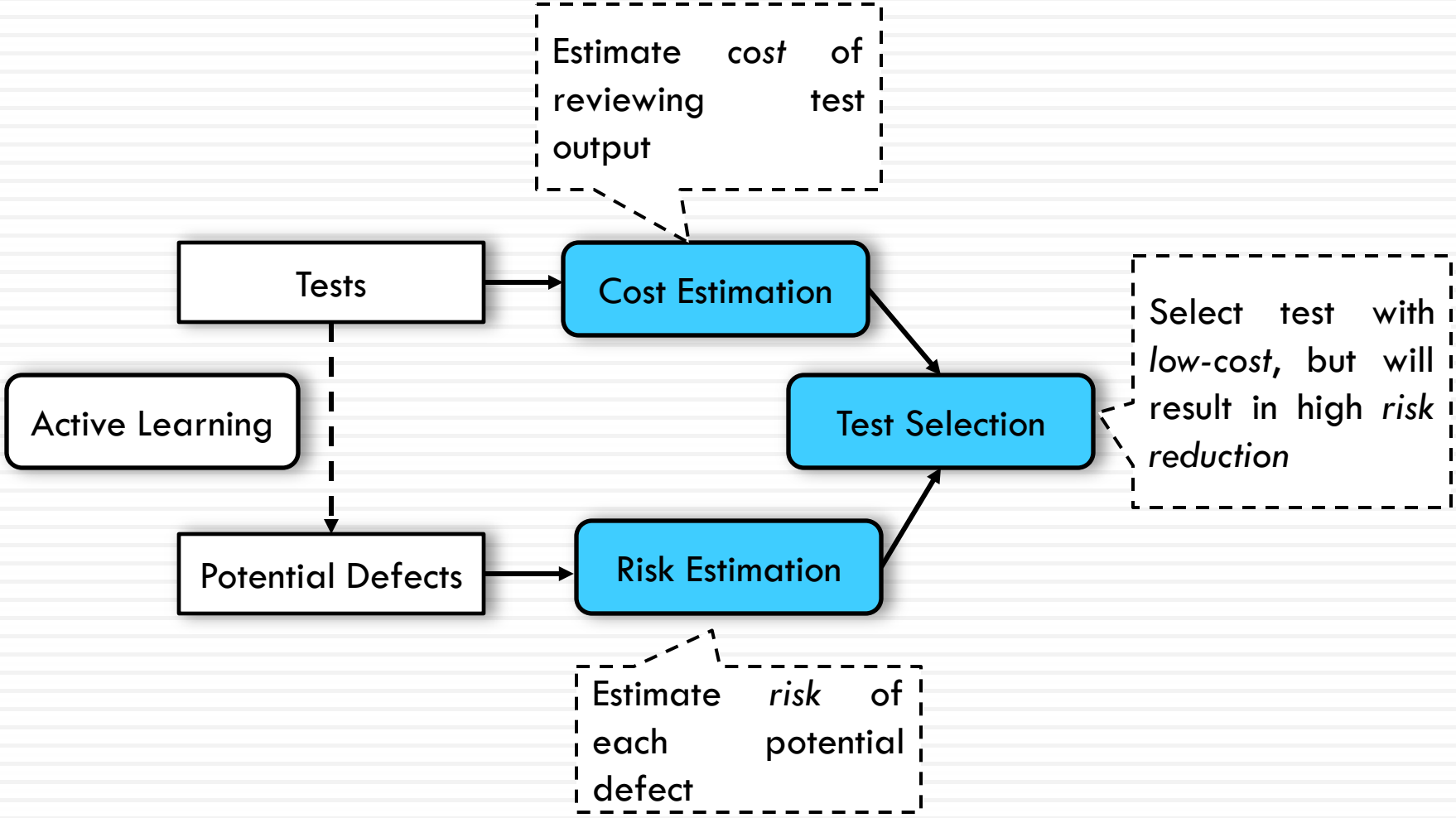
CARIAL

- **CARIAL: Cost-Aware Reliability Improvement with Active Learning**
  - **Goal:** balancing *cost* of testing and *benefit* in reliability improvement / risk reduction
  - **Main idea:** select tests that balance
    - Cost of Output Checking
    - Benefit of risk reduction

CARIAL Framework



CARIAL Framework



Cost Estimation

- **Cost**
  - Cost of test output evaluation is associated with the size or complexity of the output
  - Estimated by:

$$\alpha * \text{size of } t_i \text{ or } \alpha * \text{complexity of } t_i$$

Risk Estimation

- **Definition of Risk**
  - *Expected loss* incurred if a defect d_i remains in the released code
  - Depends on two factors
    - **defect severity:** trivia, normal, major, blocker..
      - $L(\text{severity}(d_i))$: loss incurred for a certain severity level
    - **defect failure rate (dfr):** probability of triggering a defect d_i in field
  - Formal definition

$$R(d_i) = L(severity(d_i)) * dfr(d_i)$$

Getting failure rate

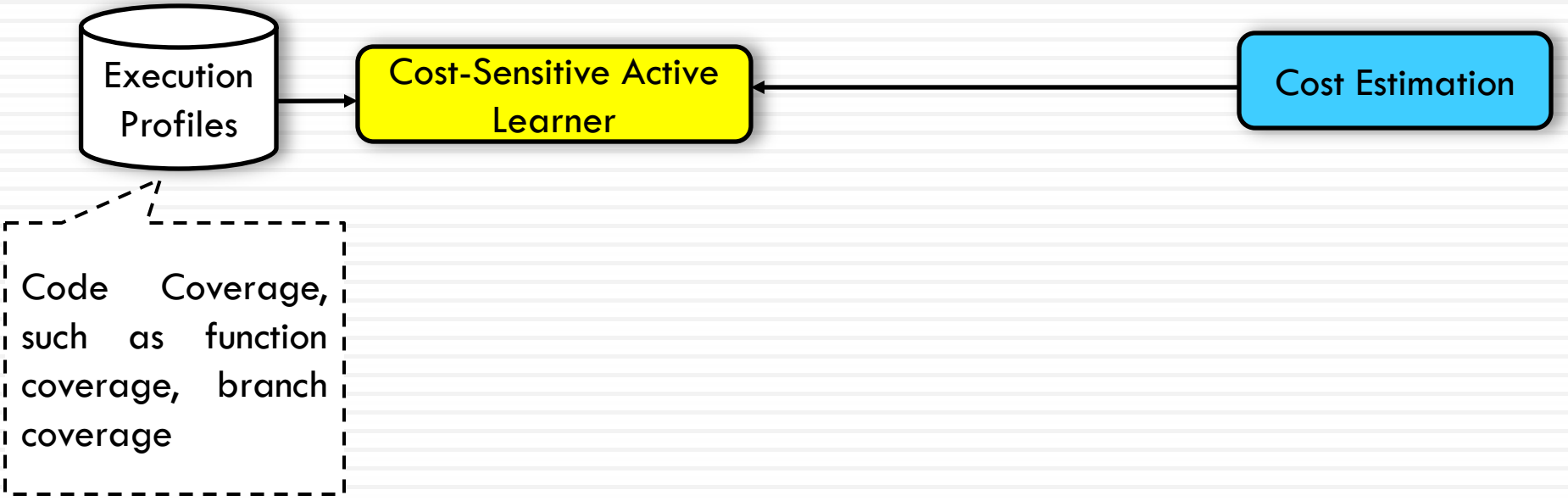
- **Partitioning of input corpus**
  - Successful region (*S*): test input that allow software to run normally
  - Failure regions (F_1, F_2, …, F_k): test input that cause the software to fail; each defect d_i has a corresponding F_i
  - Defect Failure Rate

$$dfr(d_i) = \frac{|F_i|}{|Q|}$$

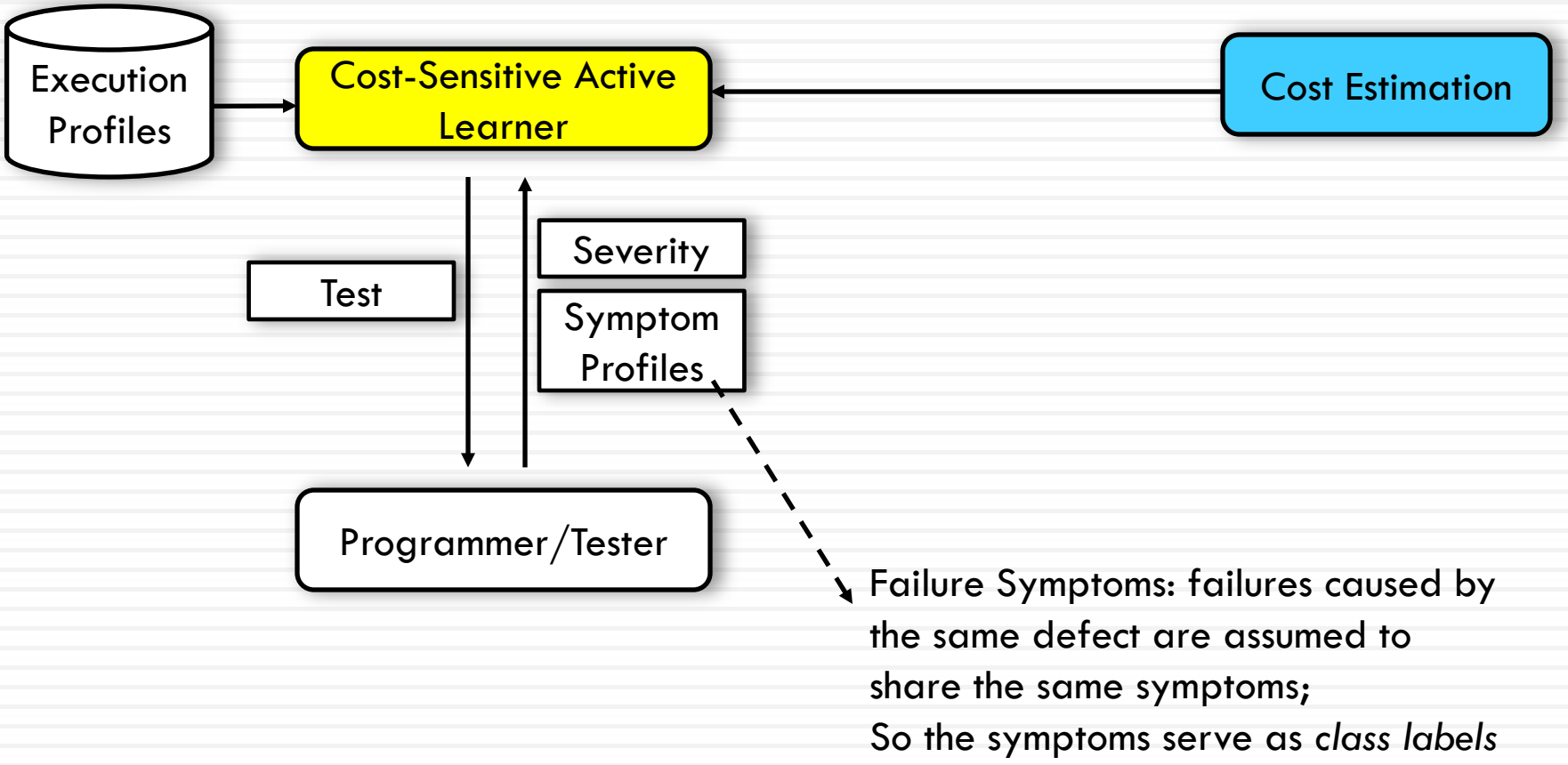
Cost-Sensitive Active learning for Risk Estimation

- **Goal**
  - Approximate failure regions by examining a sample of tests
  - Active Learning
    - Interactively get labels from users
  - Cost-Sensitive Analysis
    - Allow programmers to minimize labeling cost

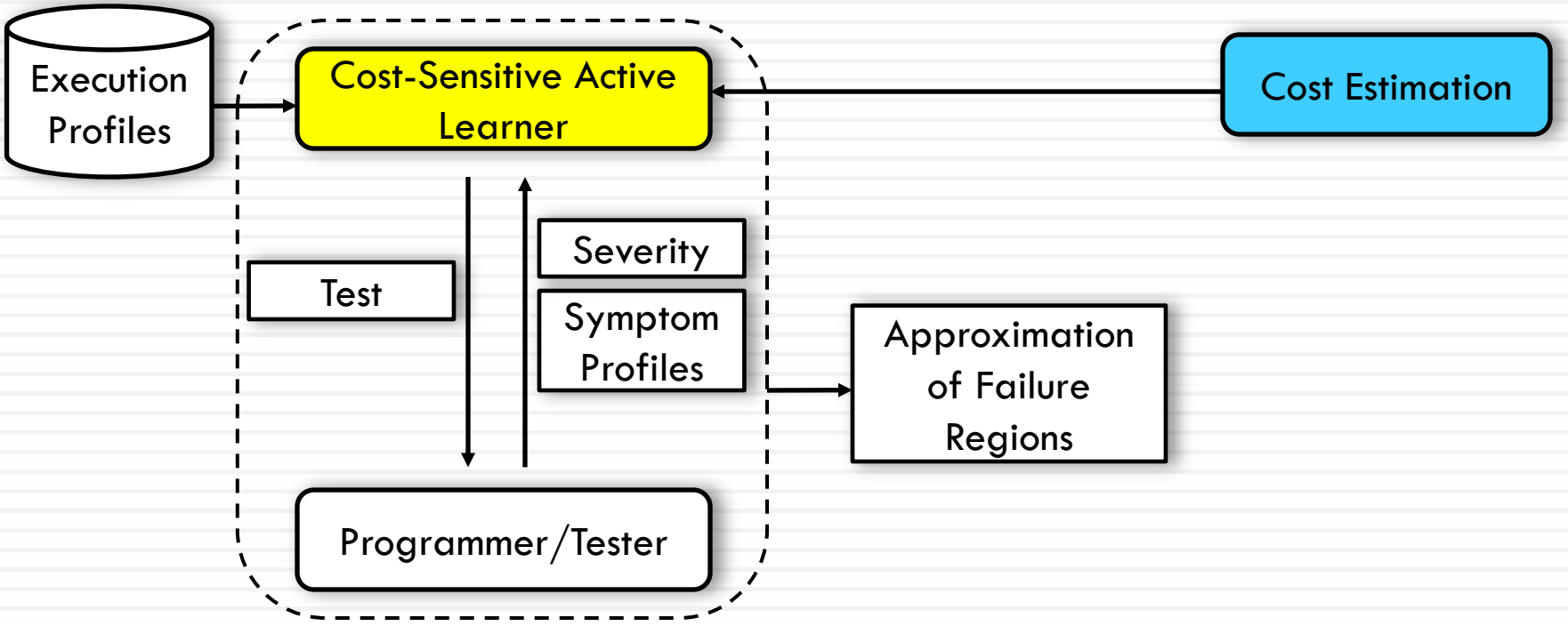
Framework of cost-sensitive active learner



Framework of cost-sensitive active learner



Framework of cost-sensitive active learner



Framework of cost-sensitive active learner

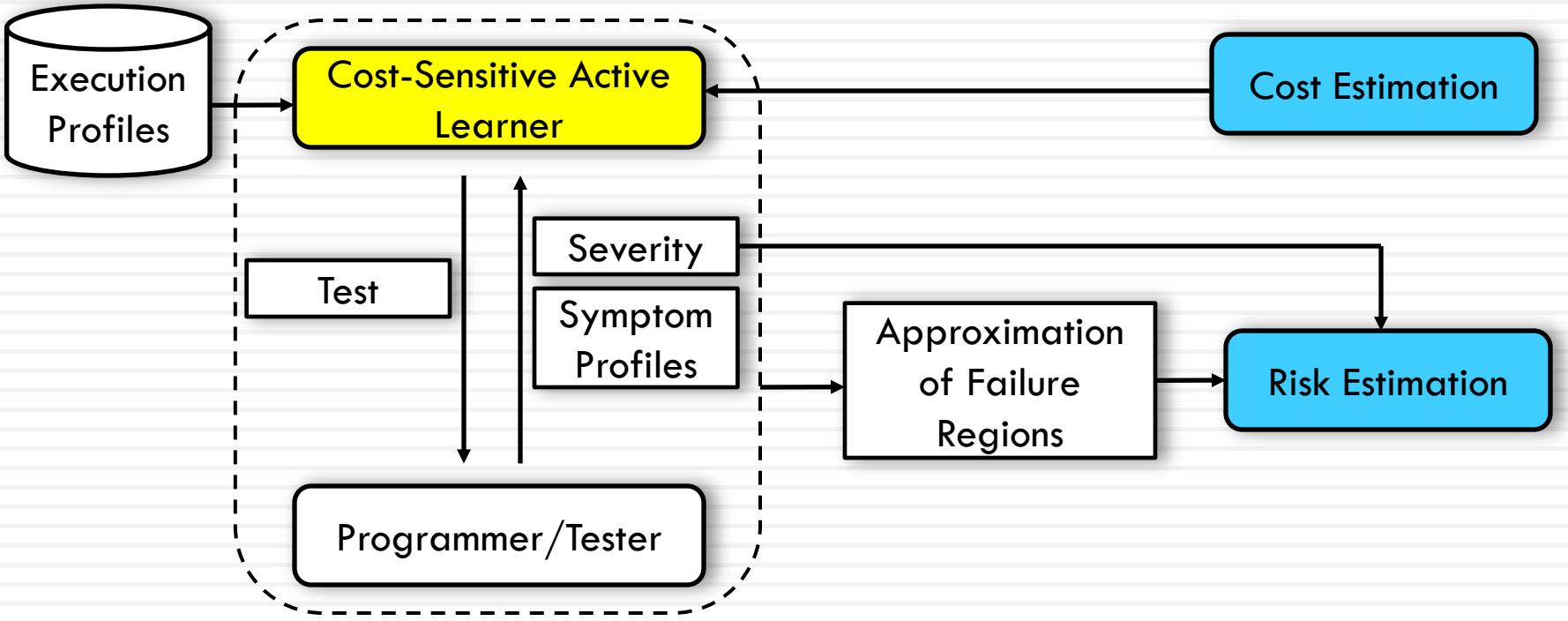
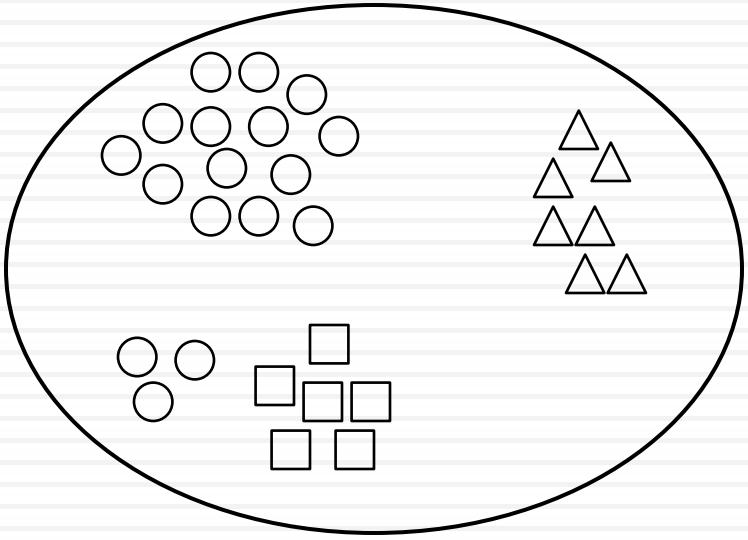


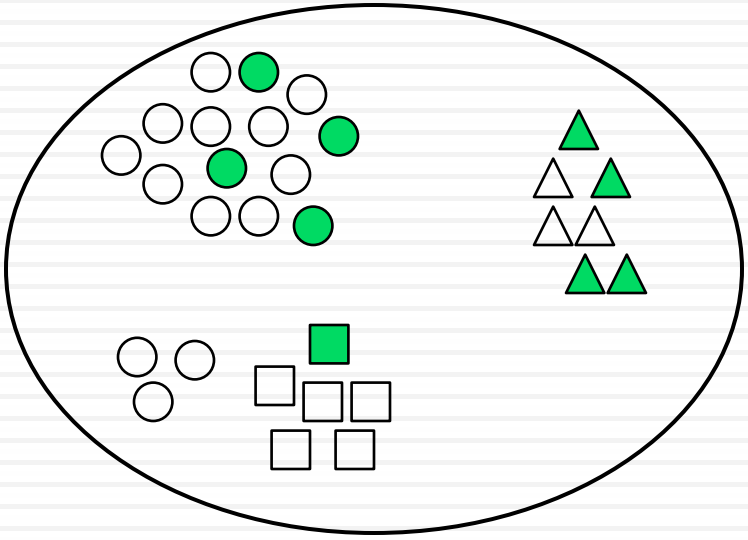
Illustration of the learner



○ Successful tests

□△ Failed tests

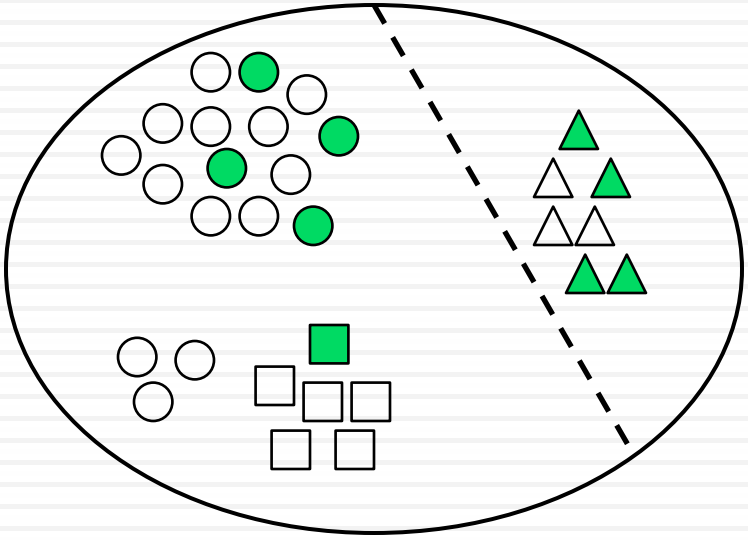
Illustration of the learner



○ Successful tests

□△ Failed tests

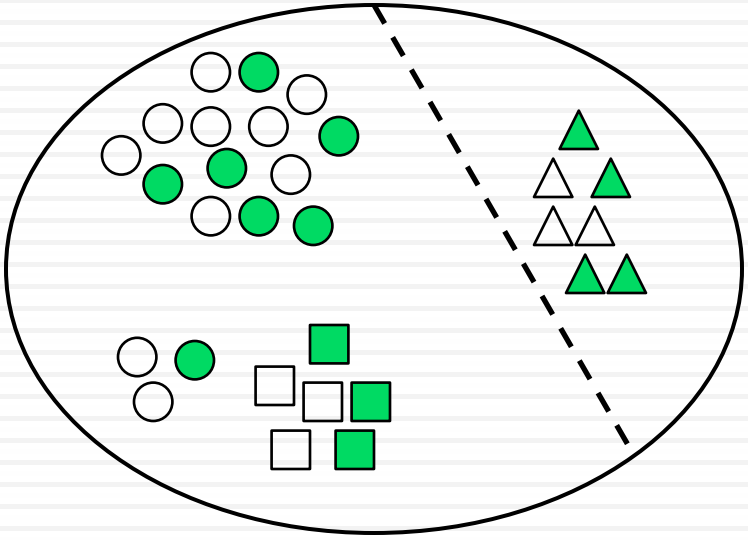
Illustration of the learner



○ Successful tests

□△ Failed tests

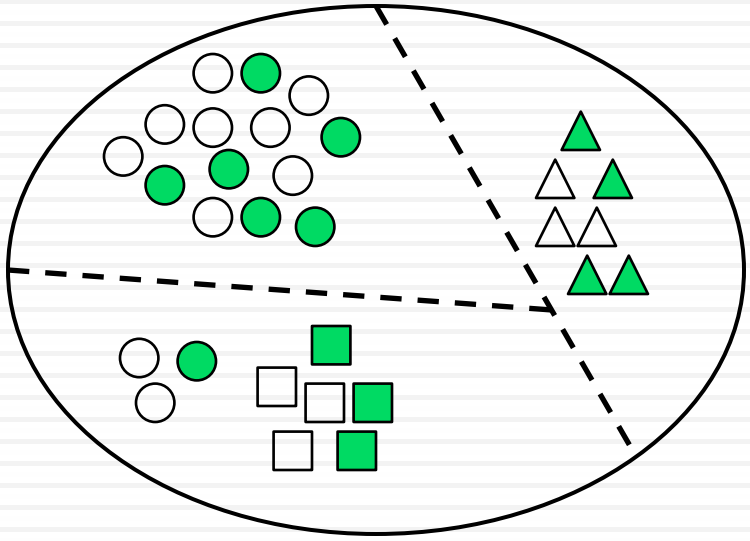
Illustration of the learner



○ Successful tests

□△ Failed tests

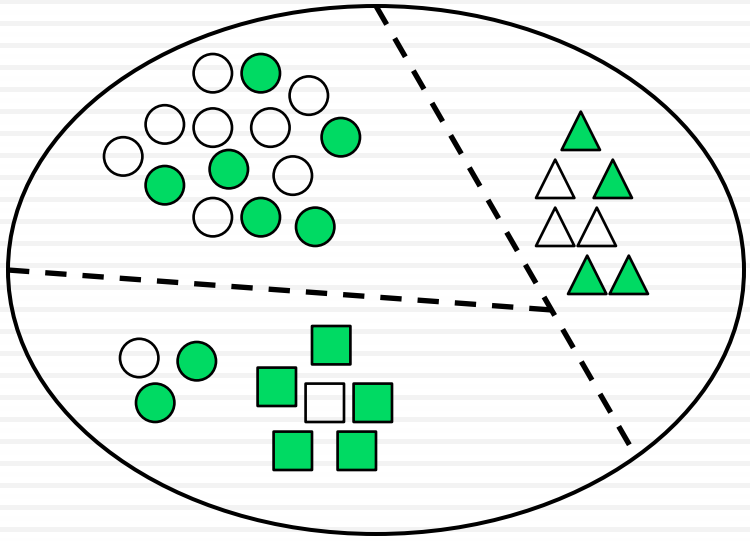
Illustration of the learner



○ Successful tests

□△ Failed tests

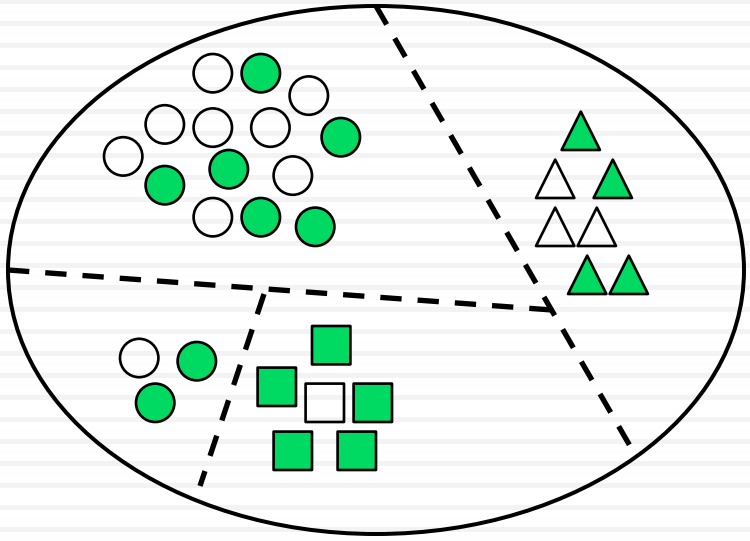
Illustration of the learner



○ Successful tests

□△ Failed tests

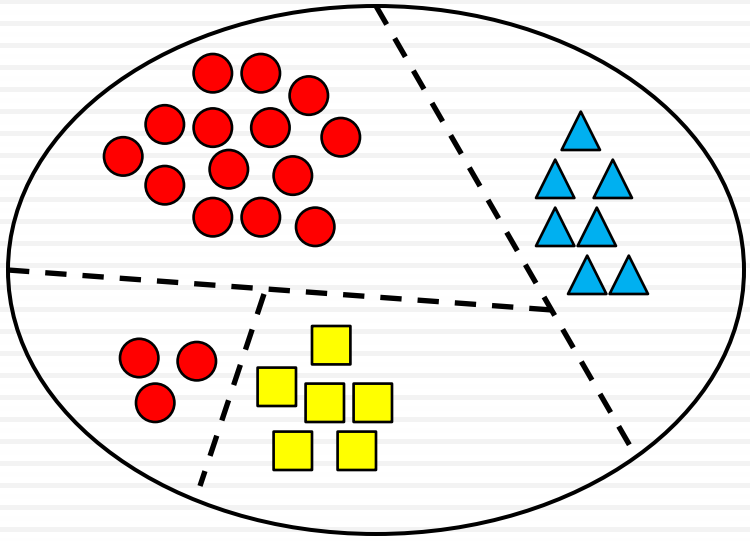
Illustration of the learner



○ Successful tests

□△ Failed tests

Illustration of the learner



○ Successful tests

□△ Failed tests

HSAL: Hierarchical Sampling with Active Learning

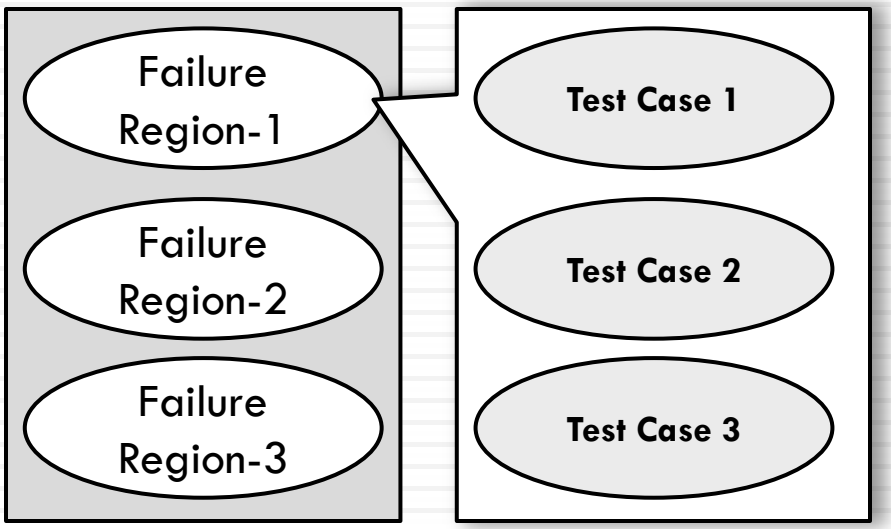
- **General idea**
  - Work with a hierarchical clustering, and quickly find a *pruning P* of the cluster tree, such that its constituent clusters are pure.
  - **Iteration**
    - SelectCluster*(P)*→*v*
      - Select an *impure* cluster
    - SelectNode(*v*)→*n*
      - Randomly select a node

CostHSAL: Reduce Cost in Test Output Checking

- Cost-sensitive analysis is applied to the two selection procedures
- SelectCluster'*(P)*→*v*
  - Select a cluster, such that
    - It is impure
    - A node being selected is likely to have low cost
- SelectNode'(*v*)→*n*
  - Nodes with low cost are tend to be selected

Tests Selection

- **Two-stage test case selection**
  - **Stage-1:** Select a failure region F_i with *high risk*
  - **Stage-2:** Select a test case t_i with *low cost* from F_i



Empirical Evaluation

Empirical Evaluation

- **Research Questions**
  - RQ-1: Accuracy of Risk Estimation
  - RQ-2: Efficiency of the test selection scheme in risk reduction

Dataset

- **Subject Programs**
  - JavaPDG: Java program dependence graph generator
  - ROME: RSS/Atom parser
  - Xerces2: XML parser
- **Summary**

	#Test Cases	%failures	#Defects	Output	Cost Estimation
JavaPDG	1295	3.9%	9	PDG	#Edges
ROME	5425	5.2%	6	XML	Size of Output
Xerces2	4773	5.4%	8	XML	Size of Output

Methodology

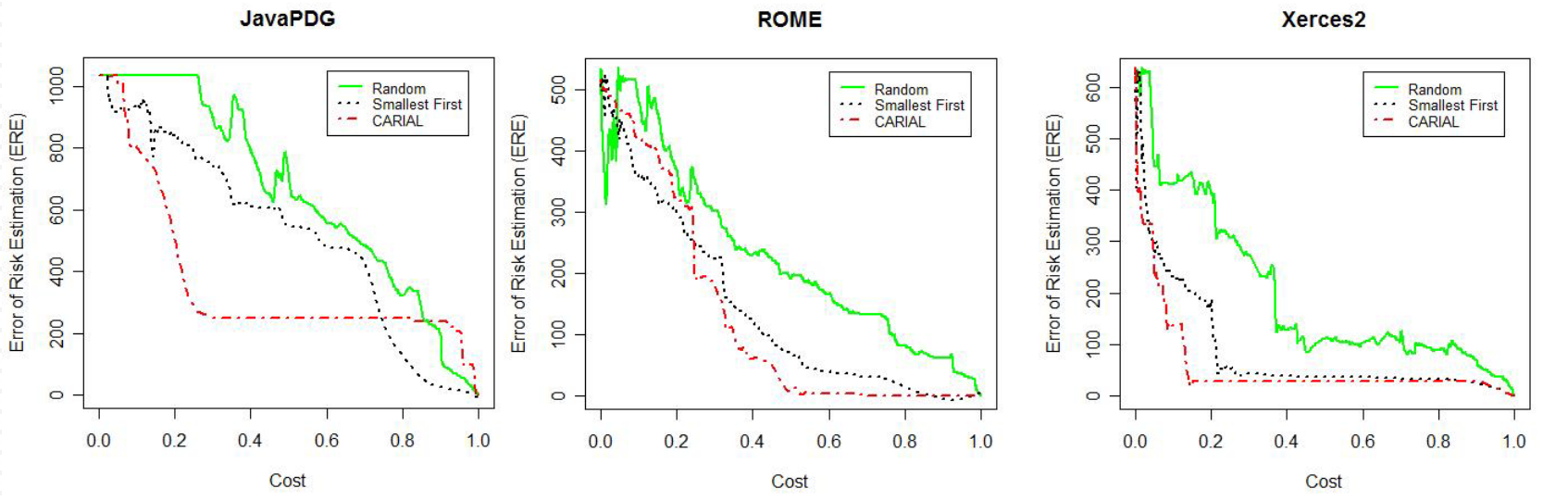
- **Baseline approaches**
  - Random Sampling
  - Smallest-First Sampling
- **Error of Risk Estimation**
  - Loss function

$$\hat{L}(severity(d_i)) = 2^{severity(d_i)} * 1000\$$$

  - Error of risk estimation

$$ERE = \left| \sum_i \hat{R}(\widehat{F}_i) - \sum_i R(d_i) \right|$$

Cost-Against-ERE curve



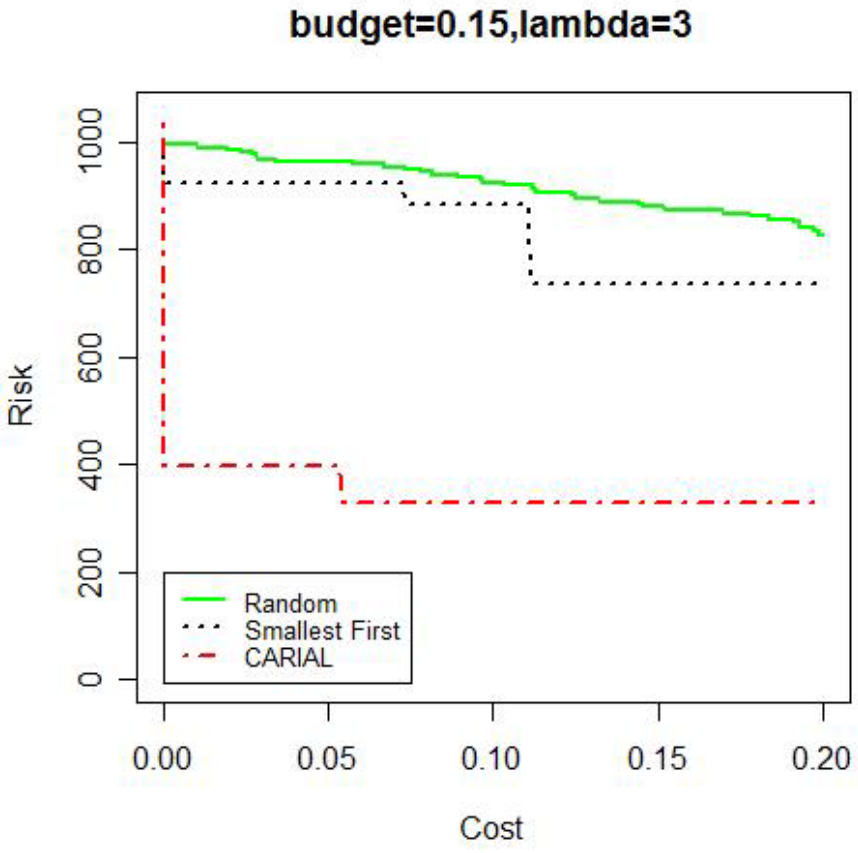
CARIAL

Random Sampling

Smallest First Sampling

RQ-2: Risk Reduction

- **Cost vs Risk using the test case selection scheme**



CARIAL

Random Sampling

Smallest First Sampling

Conclusions

- CARIAL has a fast learning curve for risk estimation.
- CARIAL is helpful in risk reduction with less cost.
- Areas of improvement:
  - Assumption on profiles: test cases sharing the same profiles may trigger different defects and vise versa.
  - Cost estimation: debugging and fault localization cost may be considered in future work.

Thanks for your attention!

Backup Slides

HSAL

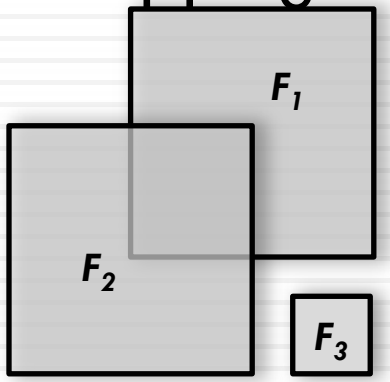
- SelectCluster(*P*)→*v*
  - Select v_i with probability proportional to $w_i * (1 - p_i)$
  - p_i is the proportion of the fraction of the majority label among labeled examples in v_i.
  - $w_i = |v_i| / \left| \sum_i v_i \right|$
- SelectNode(*v*)→*n*
  - Starting from this tree root, a child *ch* of the current cluster v_i is selected with probability proportional to the size of unlabeled data in *ch*. The iteration continues until a leaf node is reached and returned.

CostHSAL

- **MeanSampCost(*v*)**
  - Call SampleNode (*v*) *k* times and return average cost
- **SelectCluster*(P)*→*v***
  - Rank v_i by $w_i * (1 - p_i)$
  - Walk the list until the *MeanSampCost* starts to increase
  - Return the cluster with the lowest cost observed so far
- **SelectNode(*v*)→*n***
  - For current *v*, select a child *ch* with probability proportional to TotalCost(*v*)/MeanSampCost(*ch*)

Overlapping of Failure Regions

- Overlapping Failure Regions



- Non-overlapping Failure Regions