Regression Testing

- Developed first version of software
- Adequately tested the first version
- Modified the software; version 2 now needs to be tested
- How to test version 2?
- Approaches
  - Retest entire software from scratch
  - Only test the changed parts, ignoring

Regression Testing

- **“Software maintenance task performed on a modified program to instill confidence that changes are correct and have not adversely affected unchanged portions of the program.”**

Regression Testing vs. Development Testing

- During regression testing, an established test set may be available for reuse

- Approaches
  - Retest all
  - Selective retest (selective regression testing) ← Main focus of research

Formal Definition

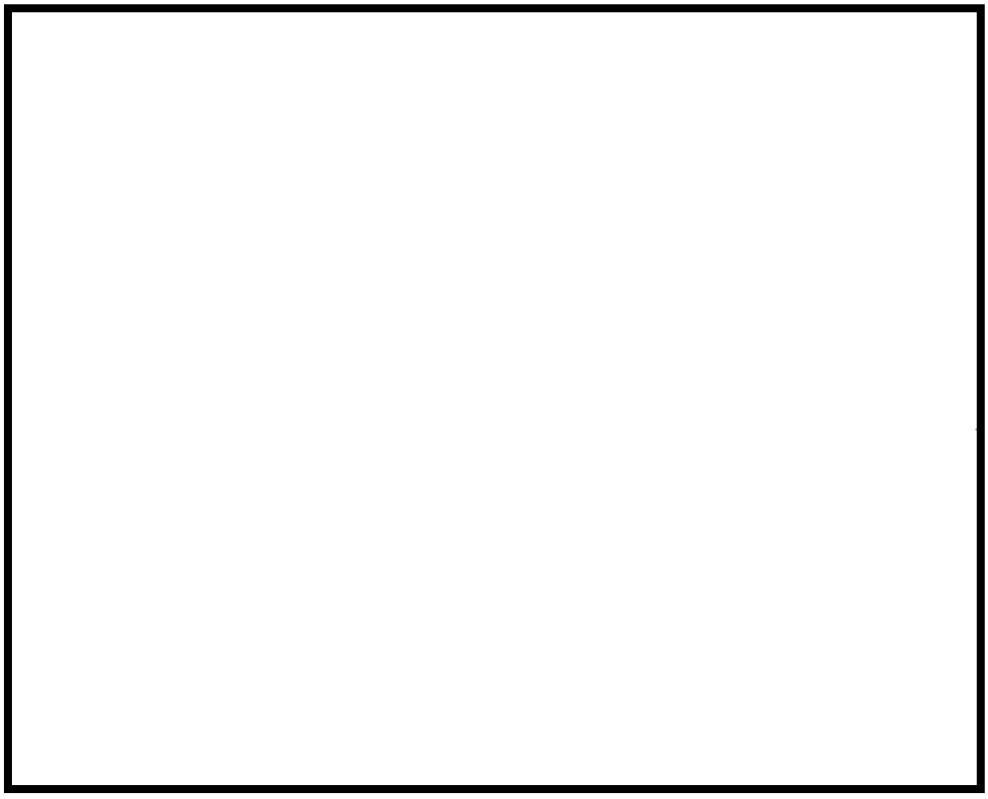
- Given a program P,
- its modified version P', and
- a test set T
  - used previously to test P
- find a way, making use of T to gain sufficient confidence in the correctness of P'

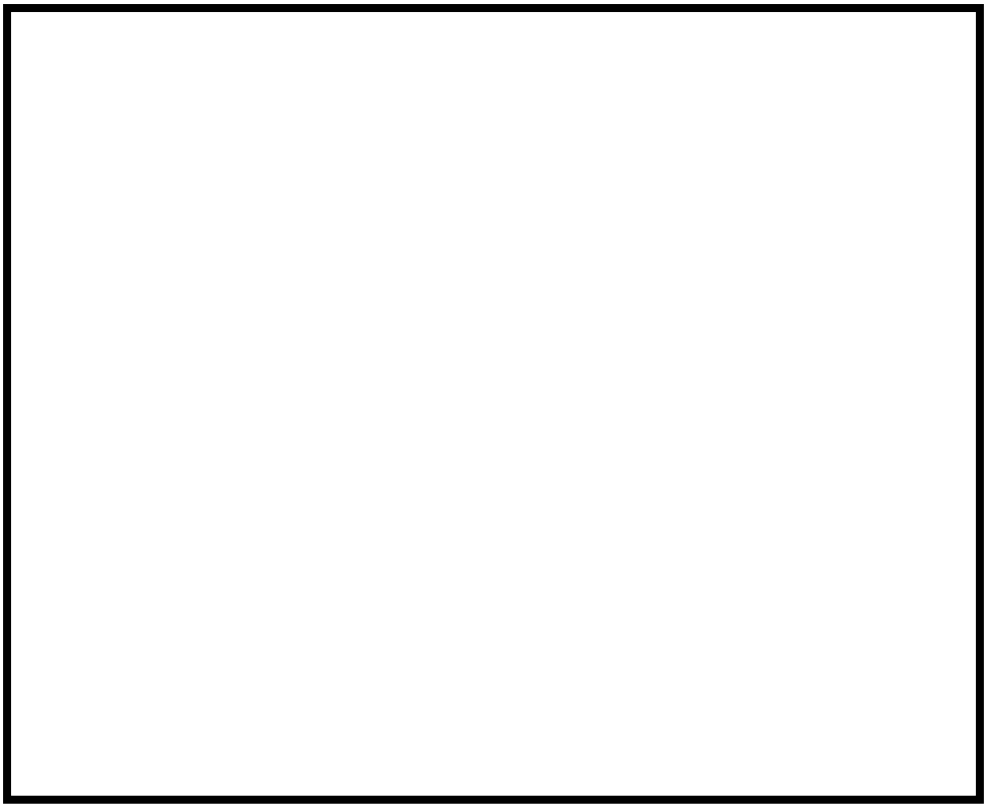
Regression Testing Steps

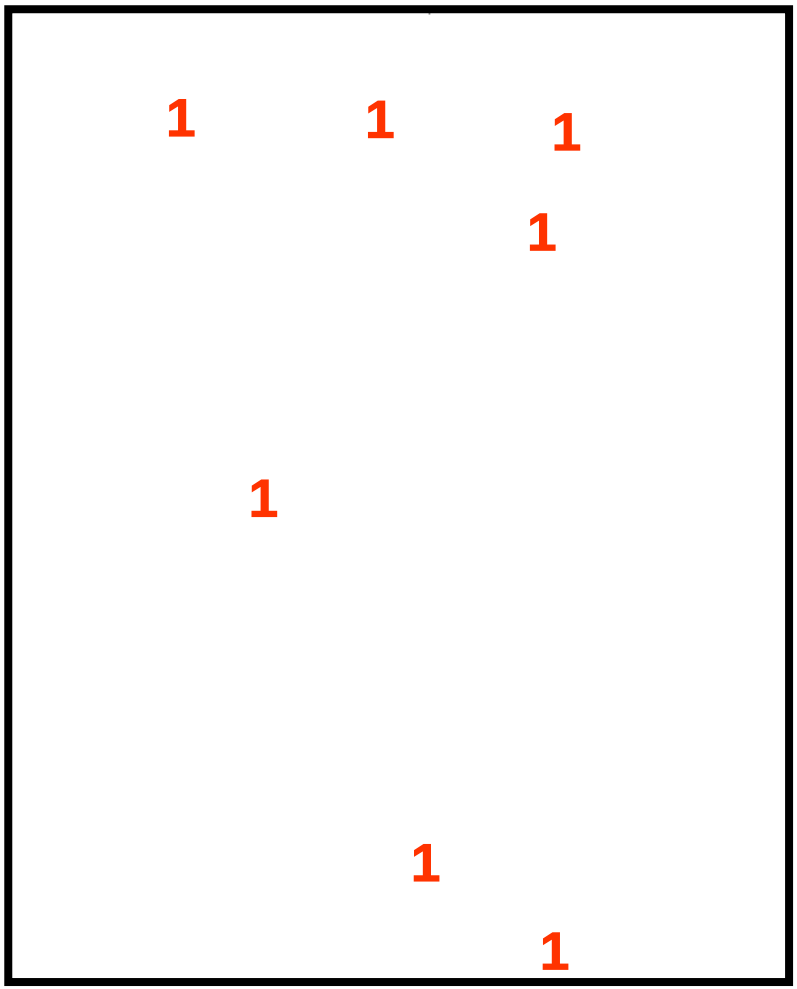
1. Identify the modifications that were made to P
   - Either assume availability of a list of modifications, or
   - Mapping of code segments of P to their corresponding segments in P'
2. Select T' $\subseteq$ T, the set of tests to re-execute on P'
   - May need results of step 1 above
   - May need test history information, i.e., the input, output

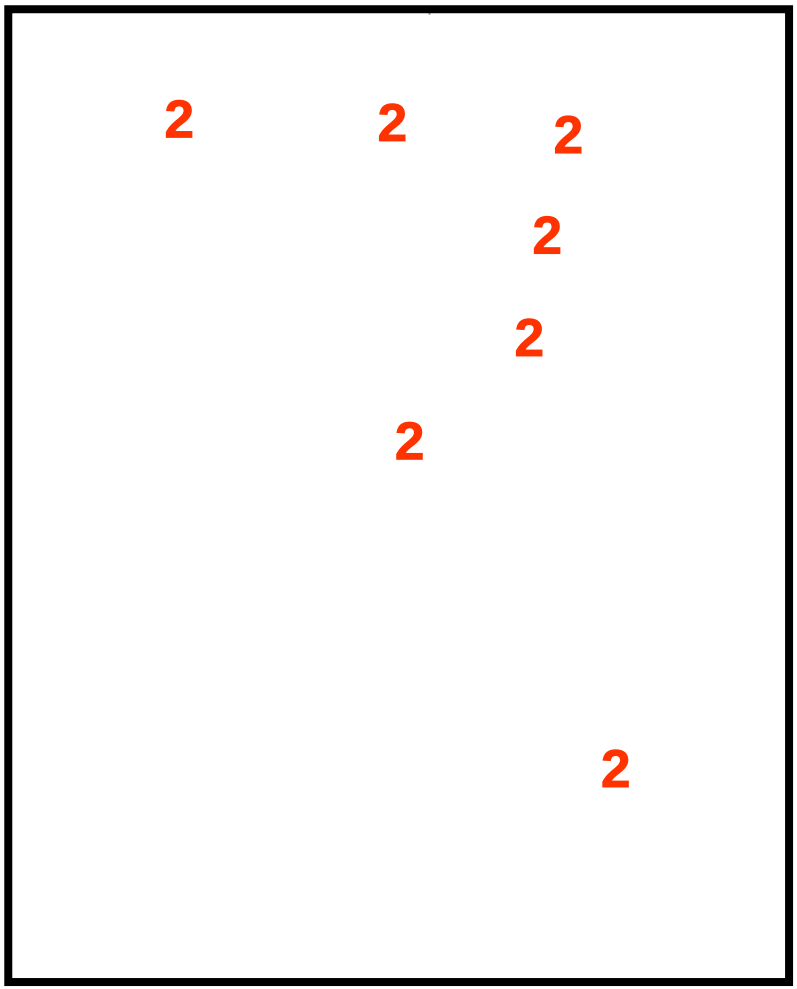
Selective Retesting

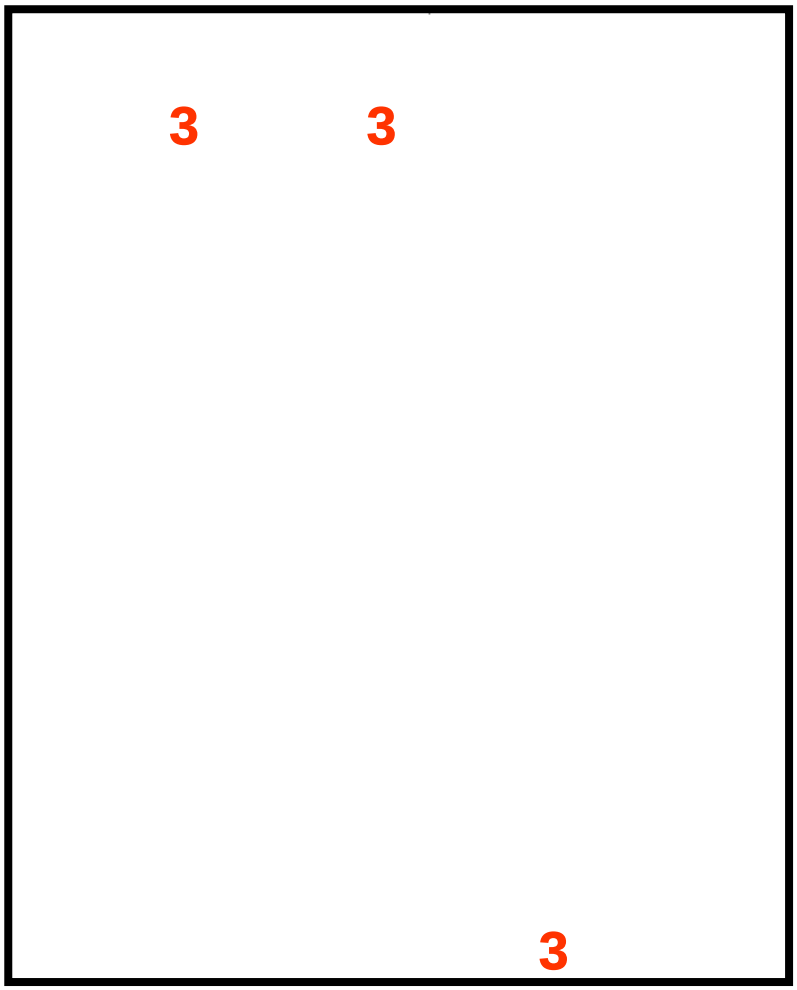
- Tests to rerun
  - Select those tests that will produce different output

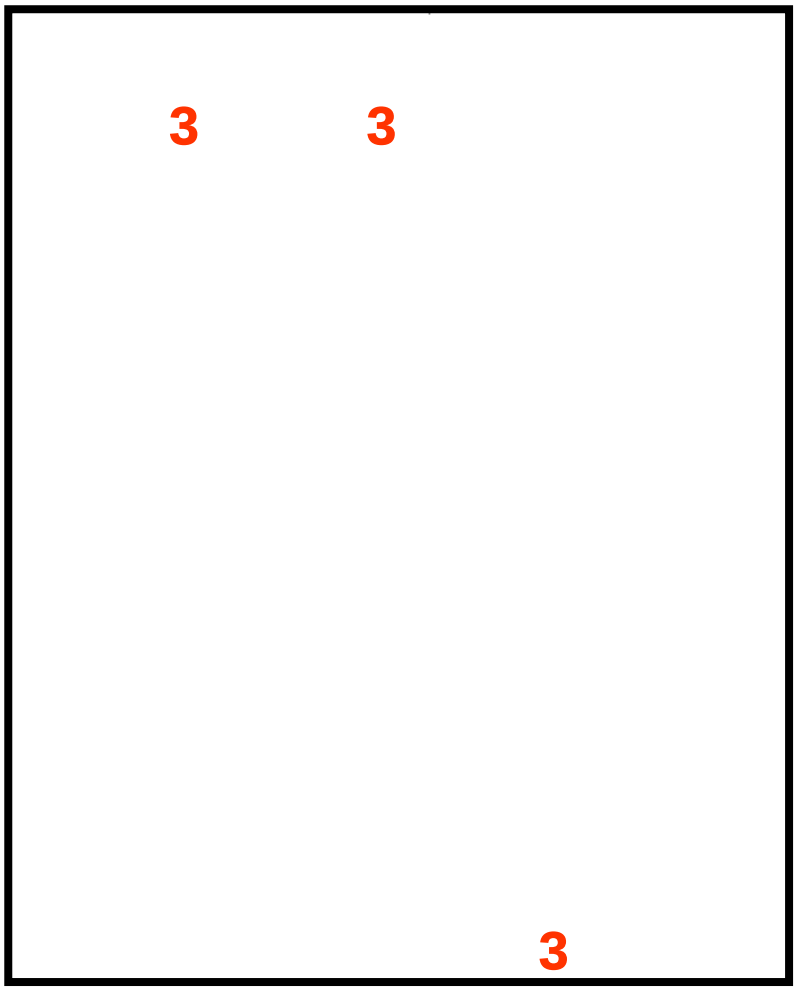




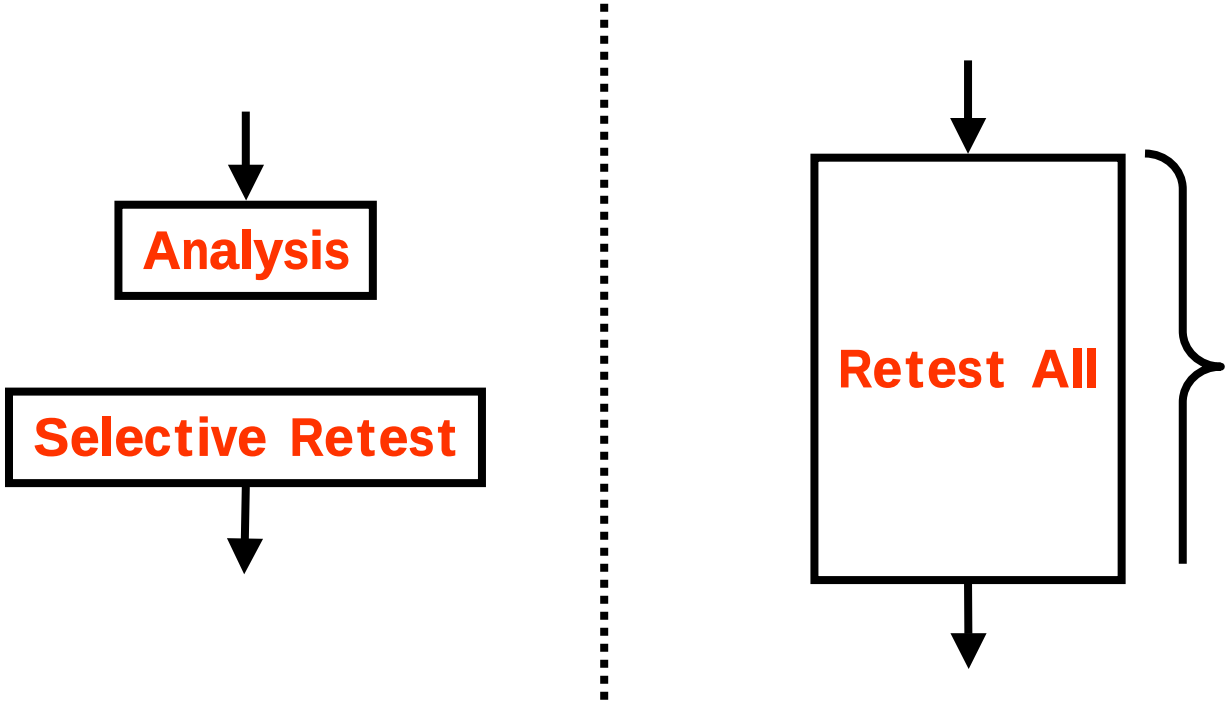

1
1
1
1
1
1
1


2
2
2
2
2
2
2


3
3
3


3
3
3

Cost of Regression Testing



Selective-retest Approaches

- **Coverage-based approaches**
  - **Rerun tests that could produce different output than the original program. Use some coverage criterion as a guide**
- **Minimization approaches**
  - **Minimal set of tests that must be run to meet some structural coverage criterion**
    - E.g., every program statement added to or modified for P' be executed (if possible) by at least one test in T

Selective-retest Approaches

- **Safe approaches**
  - **Select every test that may cause the modified program to produce different output than the original program**
    - E.g., every test that when executed on P, executed at least one statement that has been deleted from P, at least one statement that is new in or modified for P'
- **Data-flow coverage-based approaches**
  - **Select tests that exercise data interactions that have been affected by modifications**
    - E.g., select every test in T, that when executed on P, executed at least one def-use pair that has been deleted from P', or at least one def-use pair that has been modified for P'

Selective-retest Approaches

- Ad-hoc/random approaches
  - Time constraints
  -

Factors to consider

Modeling Cost

- Did not have implementations of all techniques
  - Had to simulate them
- Experi0 Twras run on severall 185, -10lt

Modeling Fault-detection

- Per-test basis
  - Given a program P and
  - Its modified version P'
  - Identify those tests that are in T and reveal a fault in P', but that are not in T'
  - Normalize above quantity by the number of fault-revealing tests in T
- Problem
  -

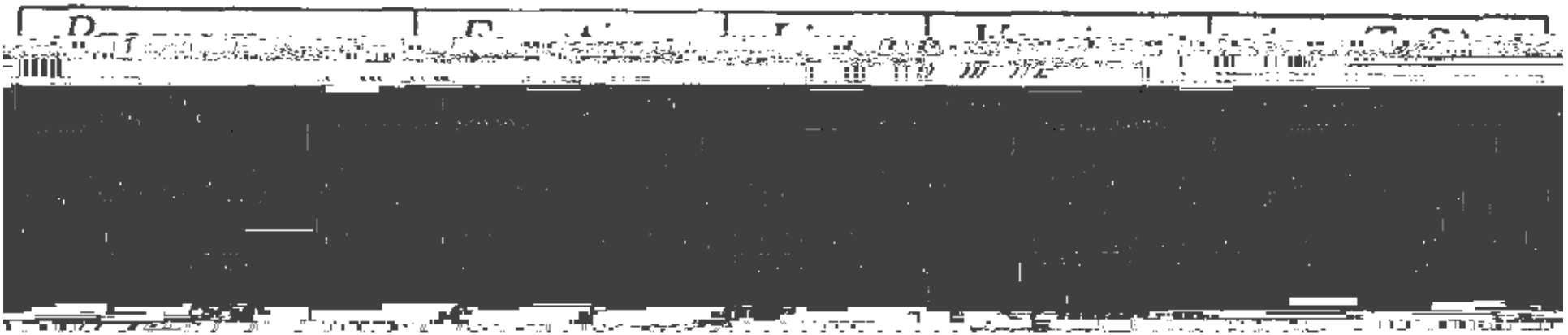
Test Suites and Versions

- **Given a test pool for each program**
  - **Black-box test cases**
    - Category-partition method
  - **Additional white-box test cases**
    - Created by hand
    - Each (executable) statement, edge, and def-use pair in the base program was exercised by at least 30 test cases
- **Nature of modifications**
  - **Most cases single modification**
  - **Some cases, 2-5 modifications**

Versions and Test Suites

- Two sets of test suites fodach ()Tf01.0815 0 T

Another look at the subjects



Variables

- The subject program
  - 6 programs, each with a variety of modifications
- The test selection technique
  - Safe, data-flow, minimization, random(25), random(50), random(75), retest-all
- Test suite composition
  - Edge-coverage adequate
  - random

Dependent variables

- Average reduction in test suite size
-

Number of runs

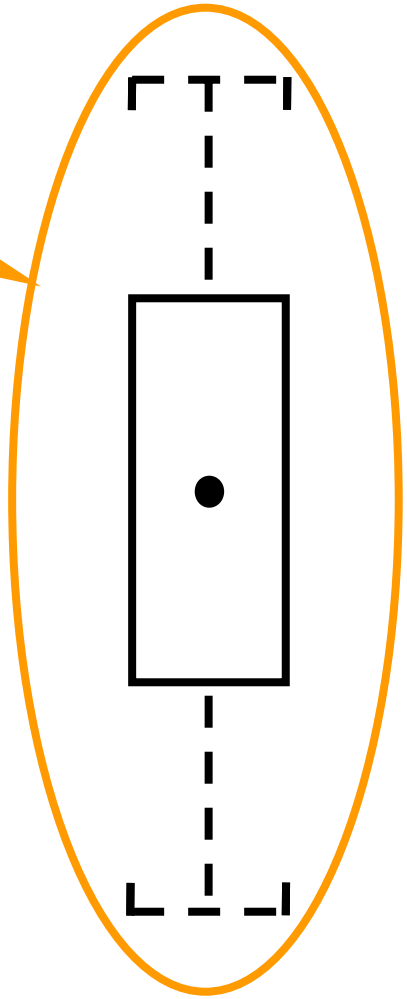
- For each subject programw9Oo1Ae ()Tf01 (

Fault-detection Effectiveness

Percentage of test suites in which T' does not reveal a fault in P'

How to read the graphs

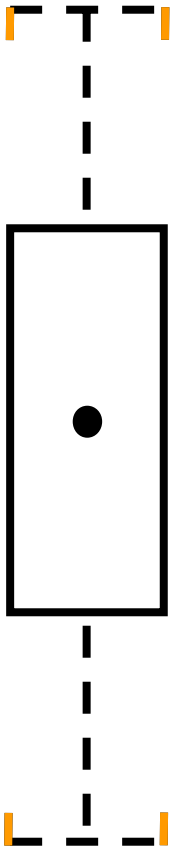
Entire structure represents a data distribution



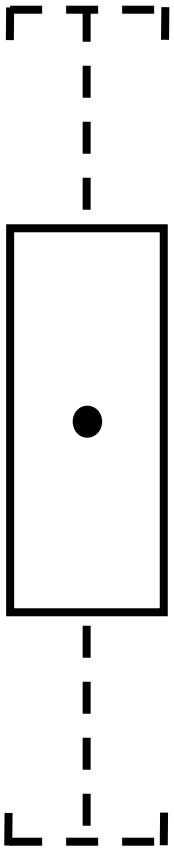
How to read the graphs

How to read the graphs

Entire structure represents a data distribution

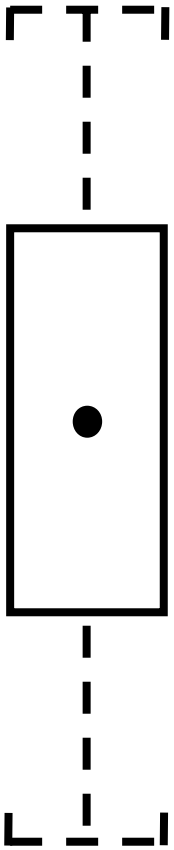


How to read the graphs

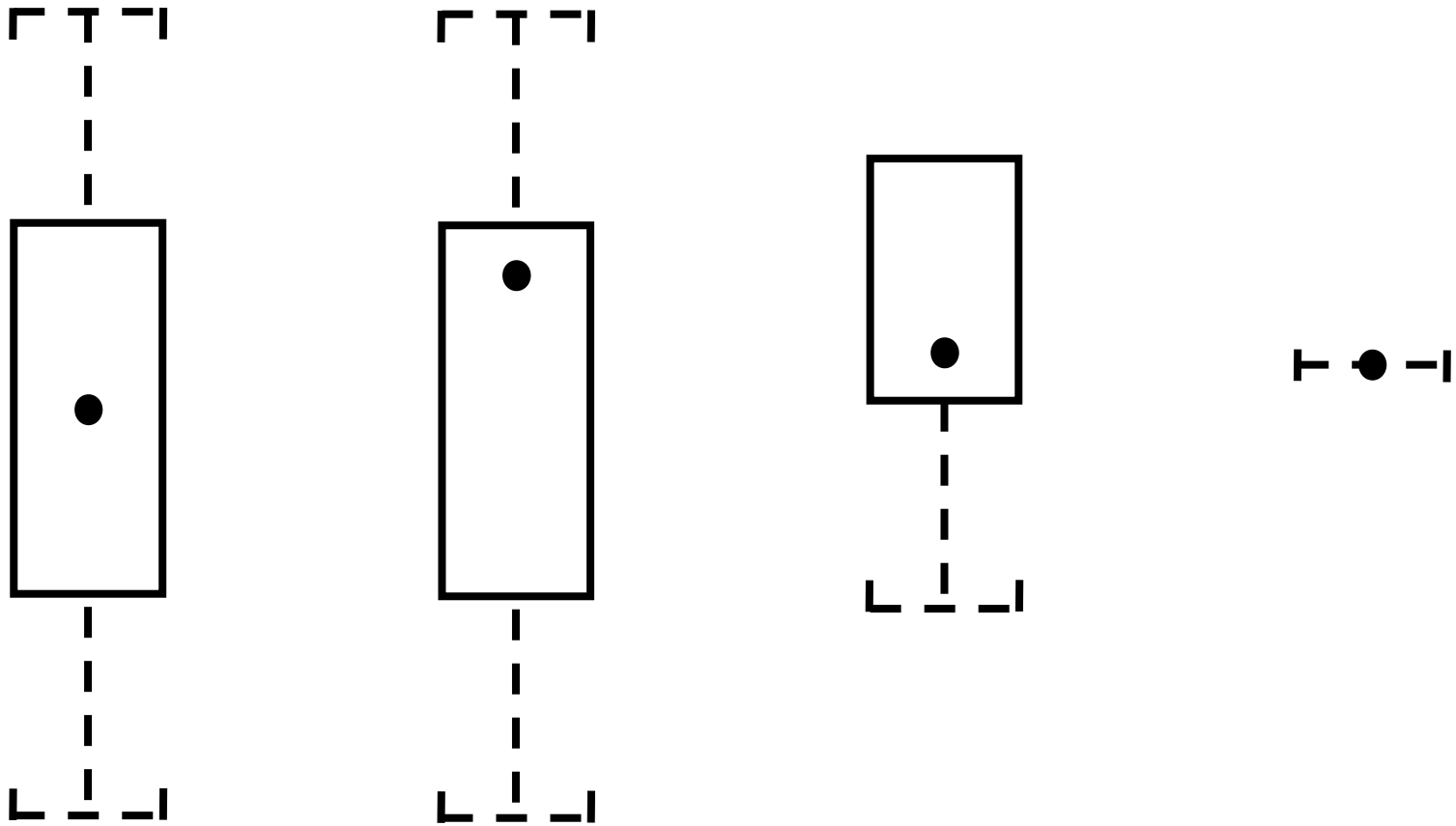


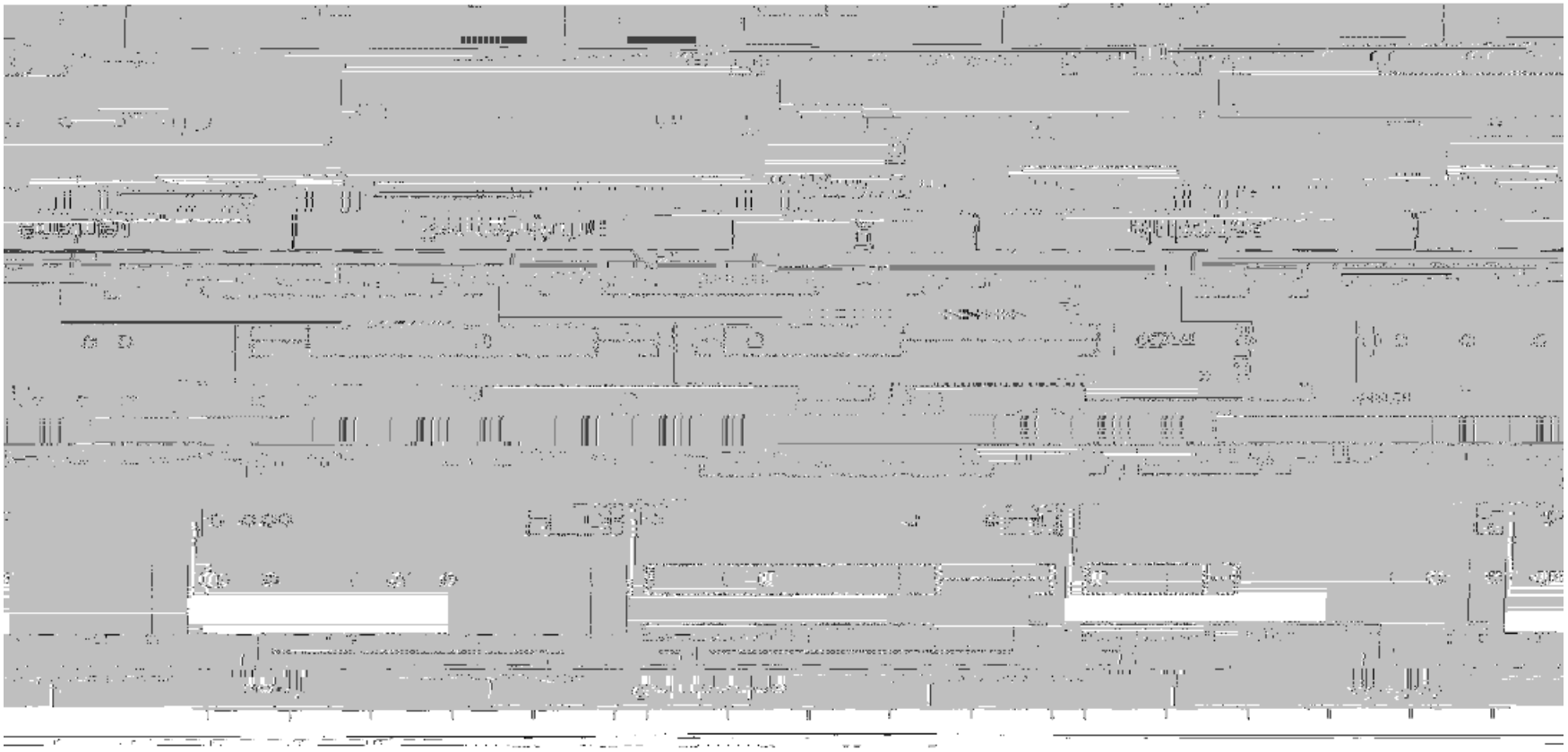
How to read the graphs

Entire structure represents a



How to read the graphs





Conclusions

- Minimization produces the smallest and the least effective test suites
- Random selection of slightly larger test suites yielded equally good test suites as far as fault-detection is concerned
- Safe and data-flow nearly equivalent average behavior and analysis costs
  - Data-flow may be useful for other aspects of regression testing
-