Quantitative Cyber-Security

Colorado State University

Yashwant K Malaiya

CS559

L 17

CSU Cybersecurity Center
Computer Science Dept

Topics

- Questions (lecture only)
- Testing
- Partitioning, Input mix
- Random testing, Detectability Profile
- Test Coverage and defects

Colorado State University

Quantitative Security

Colorado State University
Yashwant K Malaiya

CS 559

Testing

CSU Cybersecurity Center
Computer Science Dept

Faults

- Faults cause a system to respond in a way different from expected.
- Faults can be associated with bugs in the system/software structure or functionality.
  - Structure: viewed as an interconnection of components like statements, blocks, functions, modules.
  - Functionality: Described by the input/output/state behavior, described externally.
  - Both structure and functionality can be described at a higher level and a lower (finer) level.
- Example: a file > classes > methods etc. > statements

Testing

- Testing and debugging is an essential part of software development and maintenance.
  - Static analysis: code inspection
  - Dynamic: involves execution
- Defects cause functionality/reliability and security problem.
- Vulnerabilities are a subset of the defects (1-5%)
  - If exploited, allow violation of security related assumptions.
  - Vulnerability discovery can involve testing with
    - Random tests (Fuzzing)
    - Generated tests base on security requirements
- The following discussion is general for all defects.

Testing

- We assume that tests are applied at the inputs and the response is observed at the outputs of the unit-under-test.
- A test detects the presence of a fault(s), if the output is different from the expected output.
- Two test approaches:
  - Functional (or Black-box): uses only the functional description of the unit, not its structure to obtain tests. Often random ("fuzzing")
  - Structural testing: uses the structural information to generate tests. Requires more effort, but can be more thorough.
  - Combined

Random Testing

- Termed Black-box, fuzzing when used for vulnerabilities
- Random testing is a form of functional testing. In random testing, each test is chosen such that it does not depend on past tests.
- In actual practice, the "random" tests are generated using Pseudo-random algorithms that approximate randomness.
- As we will discuss later, random testing can be effective for moderate degree of testing, but not for thorough testing.

Test coverage

- A single test typically covers (i.e. tests for related faults) several sub-partitions (elements such as functions, branches, statements)
- The coverage obtained by a test-set can be obtained using coverage tools.
- The test coverage achieved by a test-set is given by ratio:

$$\text{coverage} = \frac{\text{Number of elements covered}}{\text{Total number of elements}}$$

Colorado State University

Coverage Tools

- There are several code coverage tools: Jcov, Gcov etc. for Java, C/C++ etc.
  - Compilation using the tool, instruments the compiled code to collect metrics covered.
  - Coverage metrics:
    - Statements/Blocks
    - Branches/Edges
    - Paths
    - Methods/Functions
    - Data-flow coverage metrics
  - Subsumption hierarchy
    - Complete Path coverage => 100% Branch coverage
    - Complete Branch coverage => 100% Statement coverage

Assumptions:

- A fault is associated with one or more elements.
- Exercising the element may trigger the fault to create an error
- Complete coverage does not guarantee finding all the faults.

Testing objectives

- Ordinary faults:
  - Fault detection: Apply a test input. Is the output what is expected? Triggering fault and propagating error.
  - Fault location: where is the fault?
  - Fixing: what will fix the fault? (debugging)
- Vulnerabilities
  - Apply a slightly unexpected input. Does a program crash or hang?
  - If it does, examine it to see if it leads to a vulnerability.
  - Can the vulnerability be exploited?

Partitioning

- Software can be partitioned to ensure that the software is thoroughly exercised during testing
- It is necessary to partition it to identify tests that would be effective for detecting the defects in different sections of the code.
- For testing purposes, a program may be partitioned either functionally or structurally.
- *Functional partitioning* refers to partitioning the input space of a program.
  - For example, if a program performs five separate operations, its input space can be partitioned into five partitions.
  - Functional partitioning only requires the knowledge of the functional description of the program, the actual implementation of the code is not required.
- *Structural partitioning* requires the knowledge of the structure at the code level.
  - If a software is composed of ten modules (which may be classes, functions or other types of units), it can be thought of as having ten partitions

Sub-Partitioning

- A partition of either type can be subdivided into lower level partitions, which may themselves be further partitionable at a lower level if higher resolution is needed (Elbaum 2001).
- Let us assume that a partition p_i can be subdivided into sub-partitions $\{p_{i1}, p_{i2} \ldots p_{in}\}$.
  - Random testing within the partition p_i will randomly select from $\{p_{i1}, p_{i2} \ldots p_{in}\}$. It is possible that some of them will get selected more often in a non-optimal manner.
  - Code within a sub-partition may be correlated relative to the probability of exercising some faults. Thus the effectiveness of testing may be diluted if the same sub-partition frequently gets chosen.
  - Sub-partitioning has a practical disadvantage when the *operational profile* is constructed, it will require estimating the operational probabilities of the associated sub-partitions.

Input mix: Test Profile

- The inputs to a system can represent different types of operations. The input mix called "Profile" can impact effectiveness of testing.
- Example:
  - elements e_1, e_2, …e_i, …e_n exercised with probabilities p_1, p_2, …p_i, …p_n
  - Profile then is $\{(e_i, p_i)\}$ for all elements
- For example a Search program can be tested for text data, numerical data, data already sorted etc. If most testing is done using numerical data, more bugs related to text data may remain unfound.

Input mix: Test Profile

- **The ideal Profile (input mix) will depend on the objective**
  - A. Find bugs fast? or
  - B. Estimate operational failure rate?

A. Best mix for **functional** bug finding (Li & Malaiya' 94)
  - Quick & limited testing: *Use **operational profile** (next slide)*
  - High reliability: *Probe input space evenly*
    - Operational profile will not execute rare and special cases, the main cause of failures in highly reliable systems.
  - Very high reliability: corner cases and rare combinations

B. For **security bugs**: corner cases and rare combinations
  - Vulnerability finders / exploiters look for these.

N. Li and Y.K. Malaiya, On Input Profile Selection for Software Testing, Proc. Int. Symp. Software Reliability Engineering, Nov. 1994, pp. 196-205.

H. Hecht, P. Crane, Rare conditions and their effect on software failures, Proc. Annual Reliability and Maintainability Symposium, 1994, pp. 334-337

Modeling Bug Finding Process

- The number of bugs found depend on the effort (measured by testing time) and directedness of testing.
- Directedness: looking for bugs
  - In elements not yet exercised enough
    - These will include corner cases
  - Where bugs of a specific type (specially vulnerabilities) are likely to be present.
    - Experience, expertise, intuition

Nature of faults: Detectability Profile

- All faults are not alike.
- There is no such thing as an *average* fault.
- As testing progresses, the remaining faults are the ones harder to find.

Colorado State University

Detection Probability

- Detection probability of a fault: if there are N distinct possible input vectors, and if a fault is detected by k of them, then its *detection probability* is k/N.
- A fault with detection probability 1/N would be hardest to test, since it is tested by only one specific test and none other.
- A fault which is detected by almost all vectors, would have a detection probability close to 1 and will be found with minimal texting effort. It is a low hanging fruit.

Detectability Profile of a unit under test

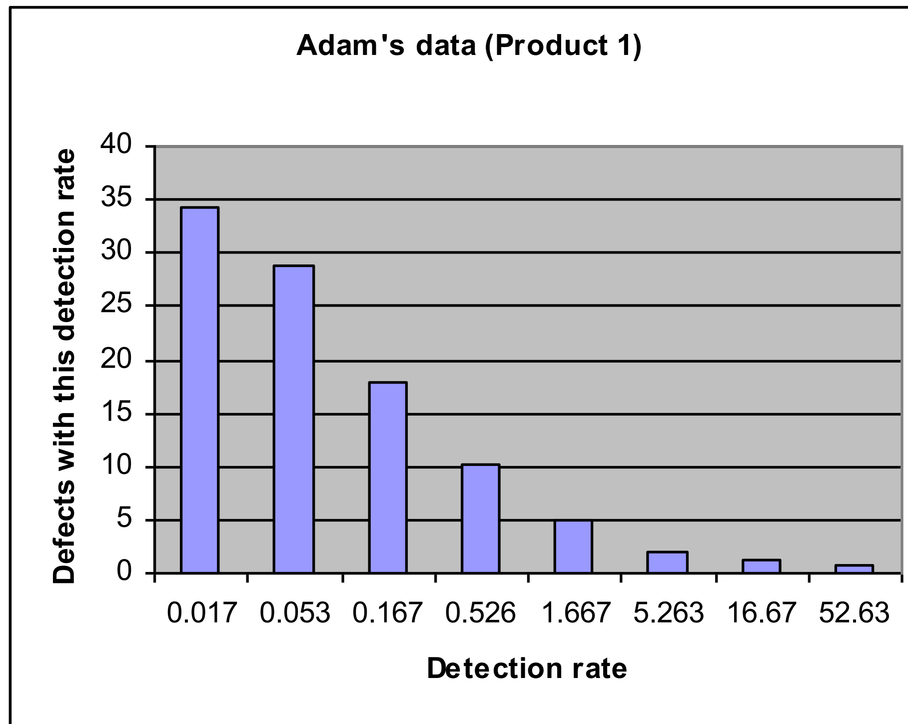
- The Detectability Profile of a unit under test describes how the defects are distributed relative to their detectability.
- Total M faults, total N possible input combinations. The set of faults can be partitioned into these subsets:
- $H = \{h_1, h_2, \ldots h_N\}$
- Where h_k is the number of faults detectable by exactly k inputs. The vector H describes the detectability profile.
  - h_1 is the number of faults that are hardest to find.
  - As testing and debugging continues, harder to find faults will tend to remain. Easy to find faults will get eliminated soon.

Applicable to software and hardware

Y.K. Malaiya and S. Yang, ""The Coverage Problem for Random Testing"
Proc. International Test Conference, October 1984, pp. 237-245

Detectability Profile: software

- Adam's Data for a large IBM software product. Note bugs with high detection rates are mostly gone.



Adams, IBM Journal of Research and Development, Jan. 1984

Detectability Profile: software

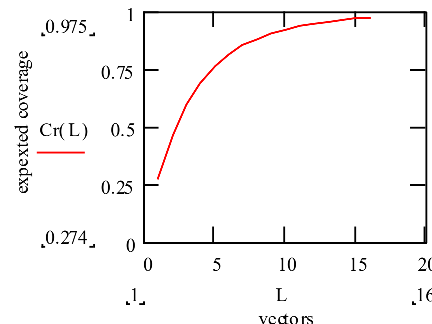
- Regardless of initial profile, after some initial testing, the profile will become asymmetric.
- In the early development phases, inspection and early testing are likely to remove most easy to test bugs, while leaving almost all hardest to test bugs still in.

Coverage Obtained by L Vectors

What fault coverage is achieved by applying L test vector?

- h_k out of M defects detectable by exactly k vectors: detection probability k/N
- P{a defect with dp k/N not detected by a vector} = $(1-\frac{k}{N})$
- P{a defect with dp k/N not detected by L vectors} = $(1-\frac{k}{N})^L$
- Of h_k faults, expected number not covered is $(1-\frac{k}{N})^L h_k$
- Expected test coverage with L vectors

$$C(L) = 1 - \sum_{k=1}^{N} (1-\frac{k}{N})^L \frac{h_k}{M}$$



Colorado State University

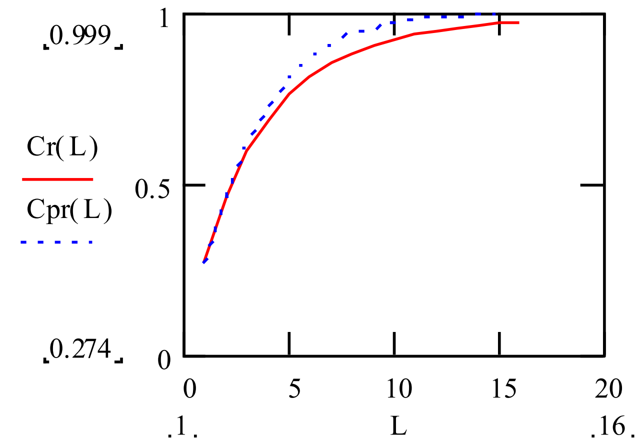
Coverage Obtained by L Vectors

Pseudorandom (PR) testing: a vector cannot repeat, unlike in true Random testing.

- For PR tests (McClusky 87)

$$C(L) = 1 - \sum_{k=1}^{N-L} \frac{{}^{N-L}C_k}{{}^{N}C_k} \frac{h_k}{M}$$

$$\approx 1 - \sum_{k=1}^{N} \left(1 - \frac{k}{N}\right)^L \frac{h_k}{M} \text{ (for Random)}$$

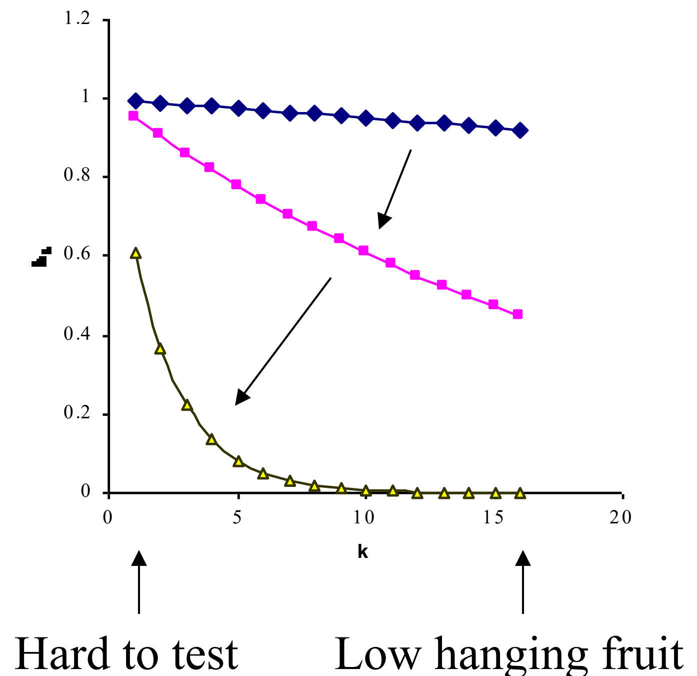


- For large L, terms with only low k (i.e. faults that are hard to test) have an impact. Thus only lower elements of H need to be estimated.
- For CECL Full Adder,

$$C(15) = 1 - [4.2 + 16.4 + 0.9 + 6.3 + 0.84 + 0.03 + 0 + ...] \cdot 10^{-3}$$

Detectability Profile: Software

- Software detectability profile is exponential
- Justification: Early testing will find & remove easy-to-test faults.
  - Inspection, unit testing, integration testing, system testing, ..
- Testing methods need to focus on hard-to-find faults.



As testing time progresses, more of the faults are clustered to the left.

Directed testing

Testing may be directed rather than random because

- Tester may wish to focus on functionality not adequately exercised by random testing (for example recovery code)
- Tester may wish to focus on more critical sections of the code.
- The probability of detecting a fault can be give by p_i, where p_i may be greater or less than k/N.

  P{a defect with dp p_i not detected by L vectors} = $(1 - p_i)^L$
- Where $p_i > \frac{k}{N}$ if the previous tests are not repeated, or the test has a good idea of where to look.
- When the exhaustive set (ES) of inputs are applied, then P{a defect with dp p_i not detected by ES} ≈ 0
  - Unlikely in most real situations.

Some common models

- Several models for ordinary bug finding process. Termed **Software Reliability Growth Models** (SRGMs).
- **Exponential SRGM**: assumes bug finding rate $\lambda(t)$ is proportional to remaining bugs at time N(t).

$$\lambda(t) = -\frac{dN(t)}{dt} = \beta_1 N(t)$$

- Which has the solution

$$\lambda(t) = \beta_0 \beta_1 e^{-\beta_1 t}$$

- Where β_0 and β_1 are parameters to be determined. B_0 represents the initial number of bugs and β_1 a measure of test effectiveness.

Colorado State University

Defect Density

- Exponential defect finding model is

$$\lambda(t) = \beta_0 \beta_1 e^{-\beta_1 t}$$

- β_0 represents the initial number of bugs.
- If the initial *defect density* is D(0), and the software size (measured in 1000 lines of code, i.e. KLOC) is S, then

$$\beta_0 = D(0) \times S$$

- The initial defect density is a function of the software development process and the degree of prior defect removal.
- The defect finding rate gradually declines, it takes infinite time to find them all according to the exponential model.
- The final defect density is sometimes used as a release criterion.

SRGM : "Logarithmic Poisson"

- If testing combines random and directed testing, the Logarithmic Poisson arises.
- **Logarithmic Poisson** model, by Musa-Okumoto, has been found to have a good predictive capability

$$\mu(t) = \beta_o \ln(1 + \beta_1 t) \qquad \lambda(t) = \frac{\beta_o \beta_1}{1 + \beta_1 t}$$

- Applicable as long as $\mu(t) \leq N(0)$. Practically always satisfied.
- Parameters β_o and β_1 don't have a simple interpretation. An interpretation has been given by Malaiya and Denton (What Do the Software Reliability Growth Model Parameters Represent?).

Y.K. Malaiya, A. von Mayrhauser and P. Srimani, "An Examination of Fault Exposure Ratio," IEEE Trans. Software Engineering, Nov. 1993, pp. 1087-1094.

References

- Y. K. Malaiya, S. Yang, “The Coverage Problem for Random Testing,” IEEE International Test Conference 1984, pp. 237-245.
- Y.K. Malaiya, A. von Mayrhauser and P. Srimani, “An Examination of Fault Exposure Ratio,” IEEE Trans. Software Engineering, Nov. 1993, pp. 1087-1094.
- S. C. Seth, V. D. Agrawal, H. Farhat, "A Statistical Theory of Digital Circuit Testability," IEEE Trans. Computers, 1990, pp. 582-586.
- K. Wagnor, C. Chin, and E. McCluskey, “Pseudorandom testing. IEEE Trans. Computer, Mar. 1987, pp. 332—343.
- E. N. Adams, "Optimizing Preventive Service of Software Products," in IBM Journal of Research and Development, vol. 28, no. 1, pp. 2-14, Jan. 1984.
- J R Dunham, "Experiments in software reliability: Life-critical applications," IEEE Tran. SE, January 1986, pp. 110 - 123
- H. Hashempour, F.J. Meyer, F. Lombardi,, "Analysis and measurement of fault coverage in a combined ATE and BIST environment," Instrumentation and Measurement, IEEE Transactions on , vol.53, no.2, pp.300,307, April 2004.

Quantitative Security

Colorado State University

Yashwant K Malaiya

CS 559

Coverage based approaches

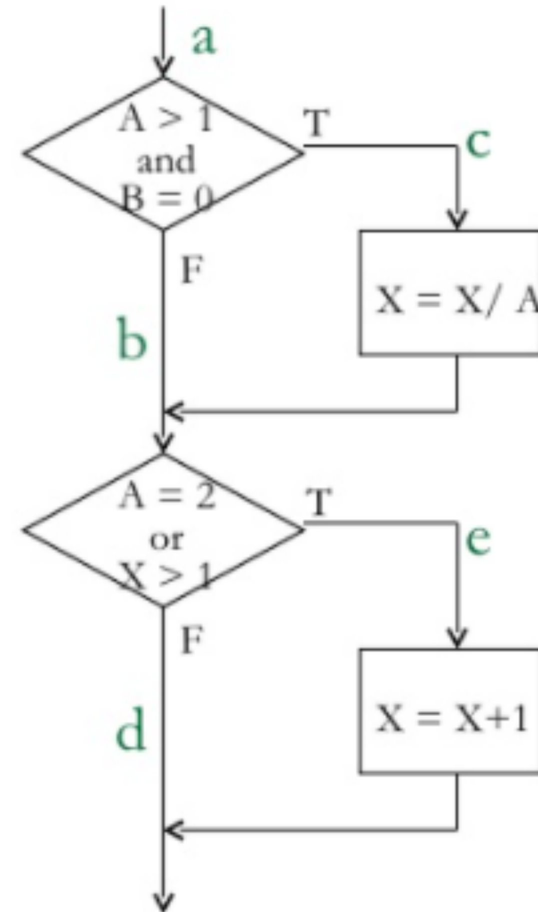
CSU Cybersecurity Center
Computer Science Dept

Test Coverage Measures

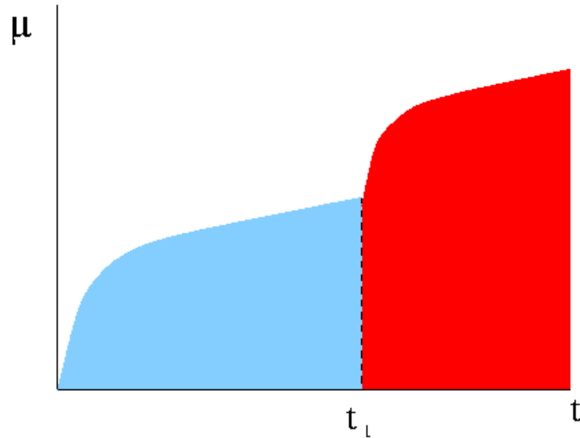
- Structiral:
  - Statement or Block coverage
  - Branch or decision coverage
- Data-flow:
  - P-use coverage: p-use pair: variable defined/modified - use as predicate
  - C-use coverage: similar -use for computation
- Subsumption hierarchy:
  - Covering *all branches* cover *all statements*
  - Covering *all p-uses* cover *all branches*

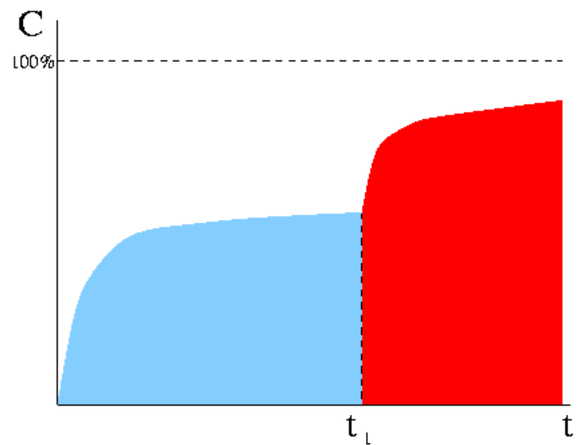
Test Coverage Measures

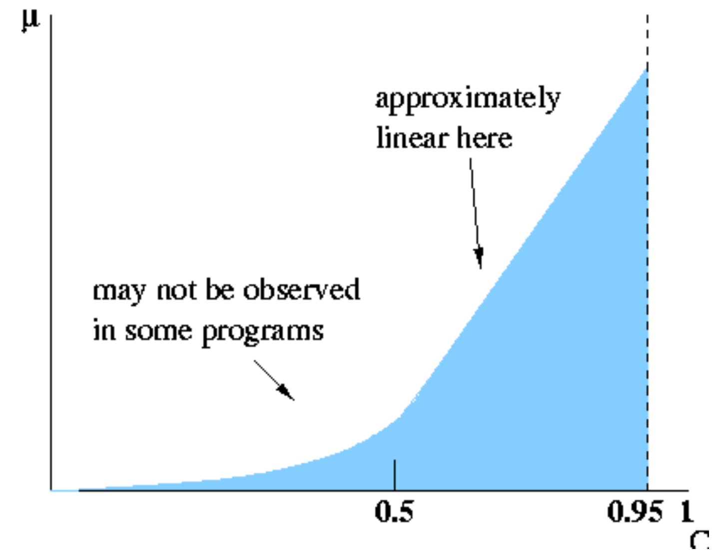
- Test case A =2, B=0, X=4
  - Covers branches a, c, e
  - Covers all the statements
- Test case A =1, B=1, X=1
  - Covers branches a, b, d
- Two test cases for 100% branch coverage.



Modeling : Defects, Time, & Coverage







Malaiya, Li, Bieman, Karcich, Skibbe, 1994
Li, Malaiya, Denton, 1998

Coverage Based Defect Estimation

- Coverage is an objective measure of testing
  - Directly related to test effectiveness
  - Independent of processor speed and testing efficiency
- Lower defect density requires higher coverage to find more faults
- Once we start finding faults, expect coverage vs. defect growth to be linear

Logarithmic-Exponential Coverage Model

- Hypothesis 1: defect coverage growth follows logarithmic model

$$C^0(t) = \frac{\beta_0^0}{N^0}\ln(1+\beta_1^0 t), \quad C^0(t) \le 1$$

- Hypothesis 2: test coverage growth follows logarithmic model

$$C^i(t) = \frac{\beta_0^i}{N^i}\ln(1+\beta_1^i t), \quad C^i(t) \le 1$$

Log-Expo Coverage Model (2)

- Eliminating t and rearranging,

$$C^0 = a_0^i \ln[1 + a_1^i(\exp(a_2^i C^i) - 1)], \quad C^0 \le 1$$

where C^0 : defect coverage, C^i : test coverage

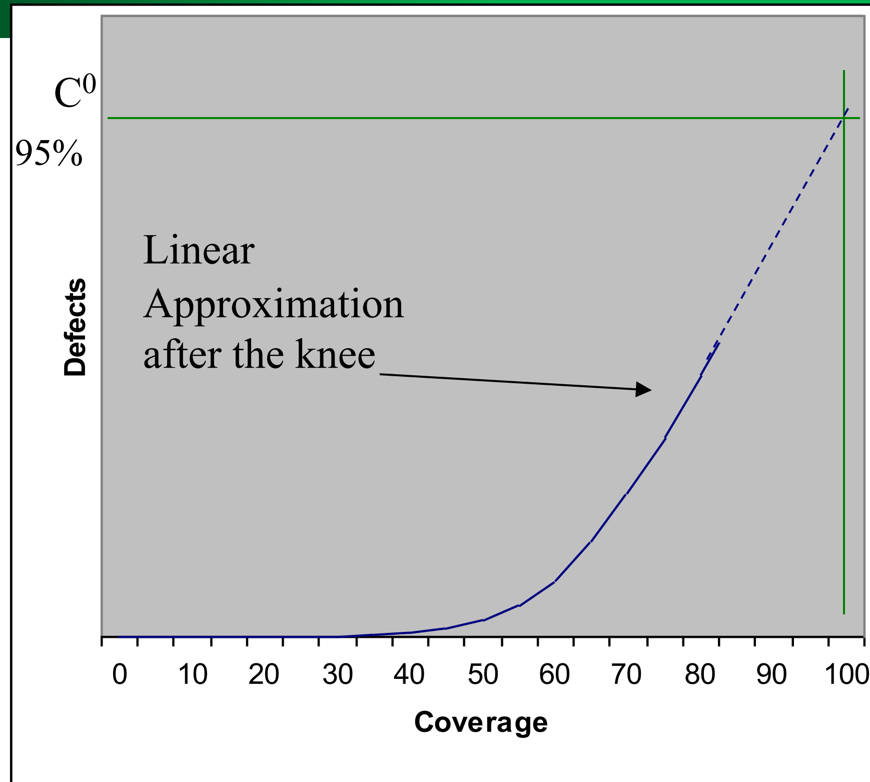
a_0^i, a_1^i, a_2^i : parameters; i : branch cov, p-use cov etc.

- For "large" Ci, we can approximate

$$C^0 = -A^i + B^i C^i$$

Colorado State University

Coverage Model, Estimated Defects



$$C^0 = -A^i + B^i C^i, \quad C^i > C^i_{knee}$$

- Only applicable after the knee
- Assumptions : Stable Software

Location of the knee

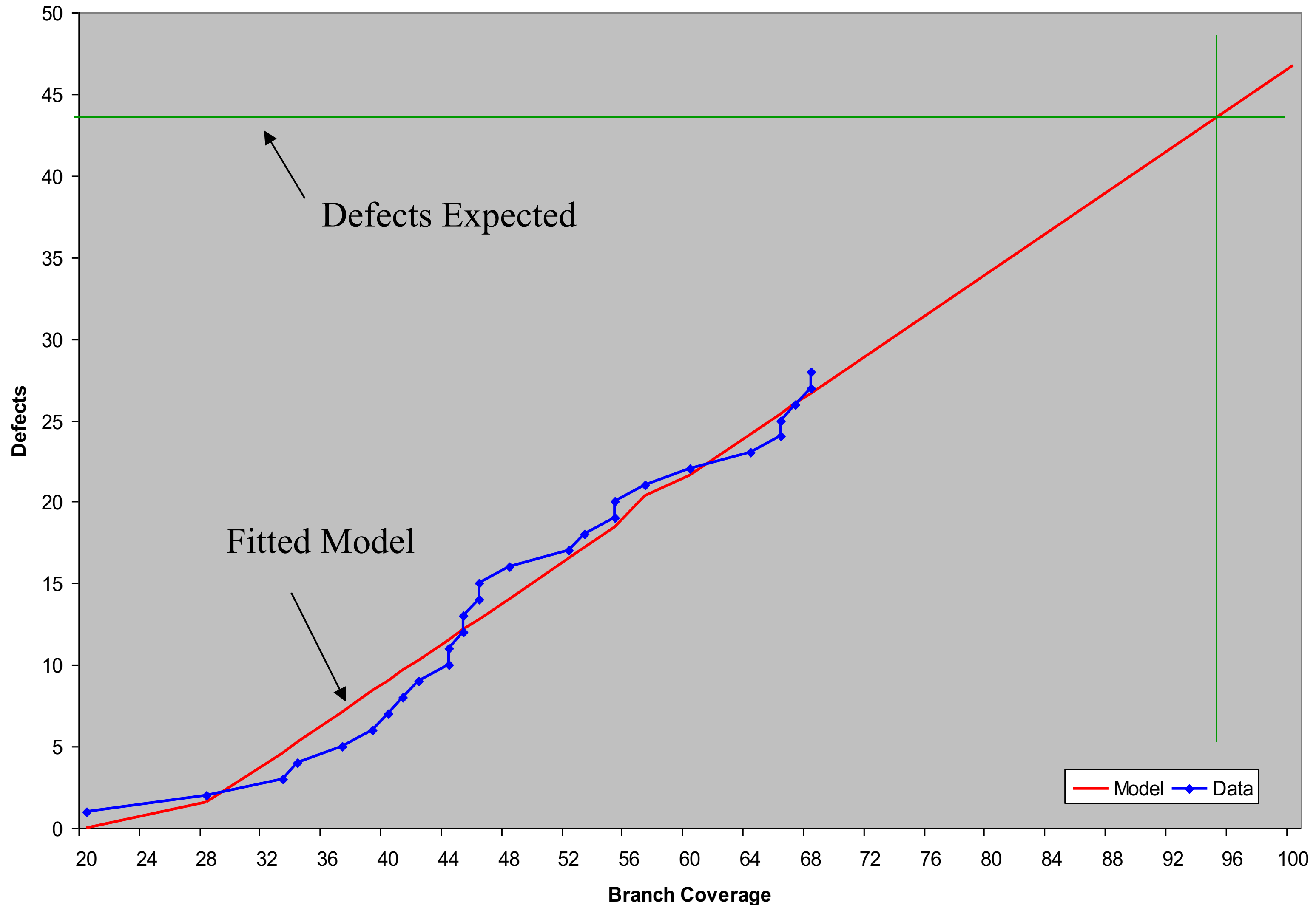
$$C_{knee} = 1 - \left(\frac{E_{min}}{D_{min} E_0}\right) D_0$$

- Based on interpretation through logarithmic model
- Location of knee based on initial defect density
- Lower defect densities cause knee to occur at higher coverage
- Parameter estimation : Malaiya and Denton (HASE ‘98)

Data Sets Used: Vouk and Pasquini

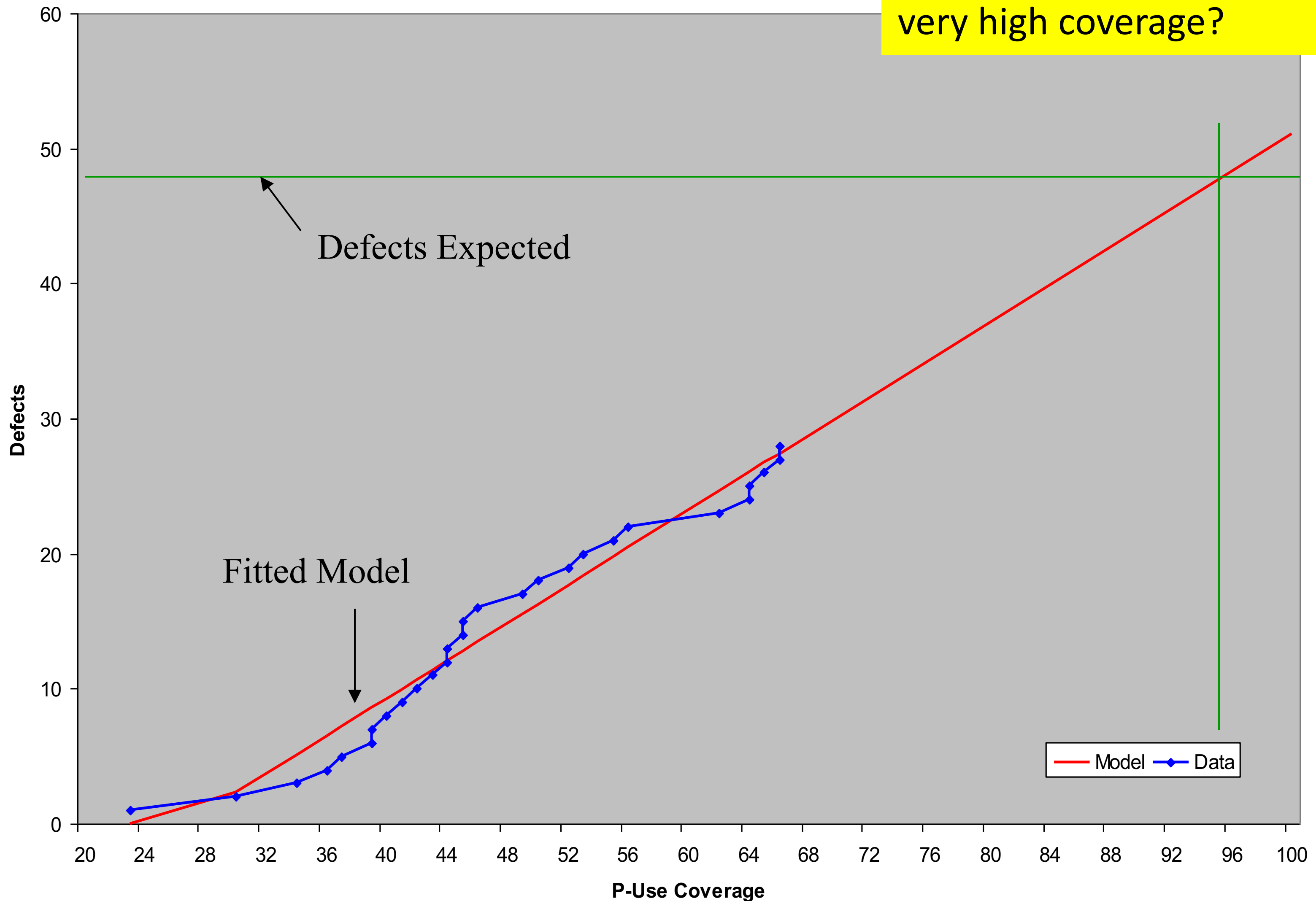
- Vouk data
  - from N version programming project to create a flight controller
  - Three data sets, 6 to 9 errors each
- Pasquini data
  - Data from European Space Agency
  - C Program with 100,000 source lines
  - 29 of 33 known faults uncovered

Defects vs. Branch Coverage



Defects vs. P-Use Coverage

Q: Will linear relation hold at very high coverage?

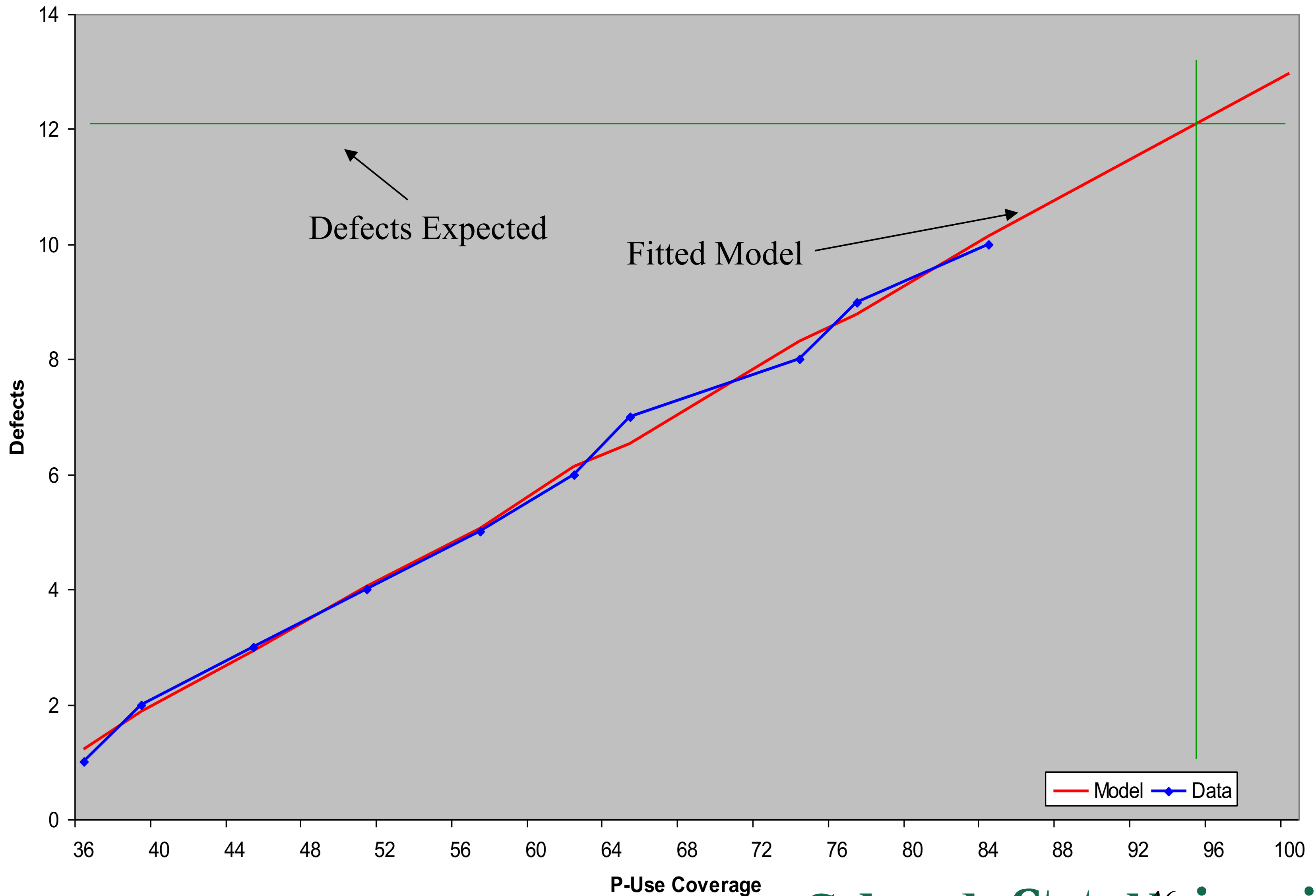


Estimation of Defect Density

- Estimated defects at 95% coverage, for Pasquini data (assume 5% *dead code*)
- 28 faults found, and 33 known to exist

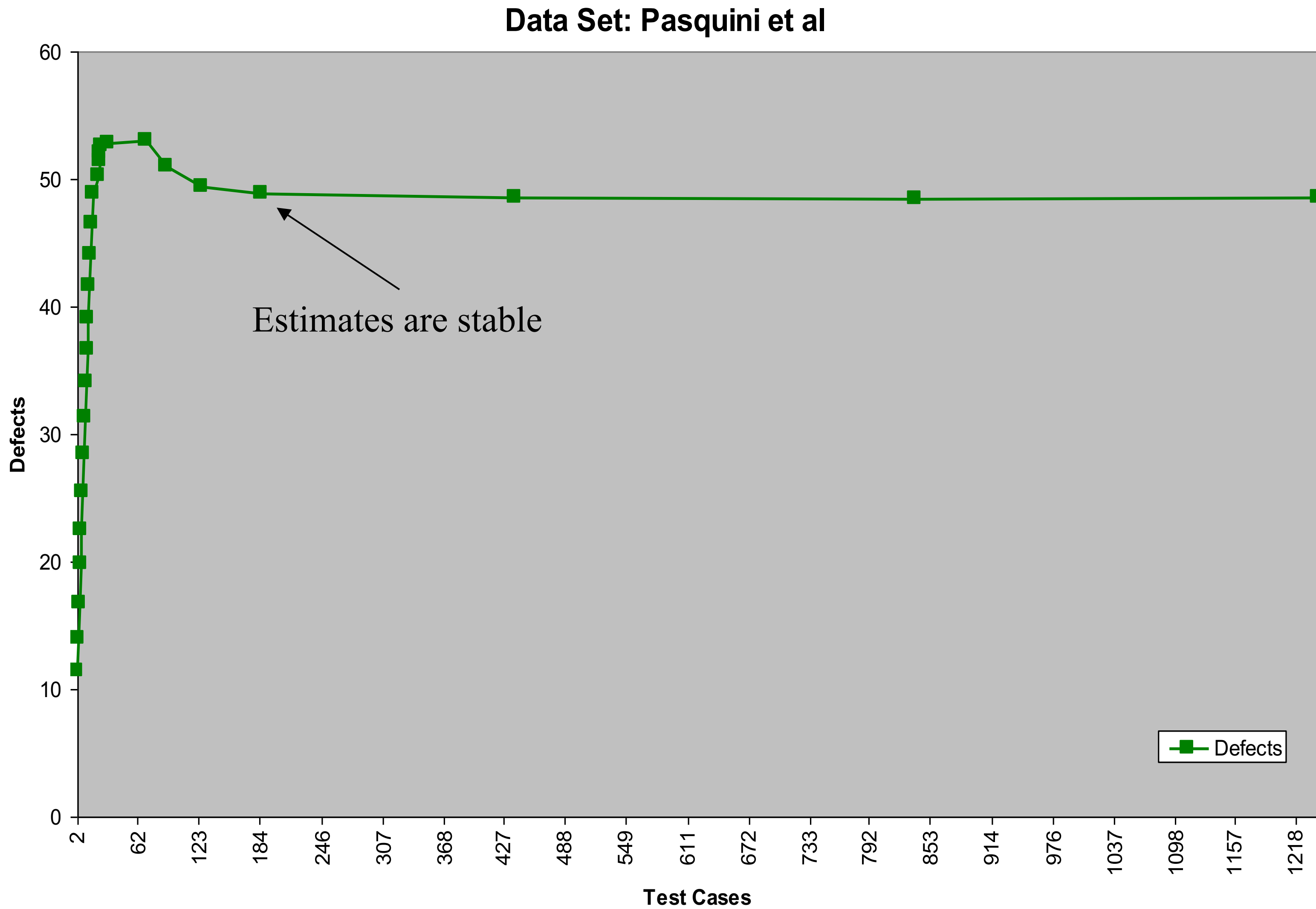
Measure	Coverage Achieved	Expected Defects
Block	82%	36
Branch	70%	44
P-uses	67%	48

Defects vs. P-Use Coverage



Colorado State University

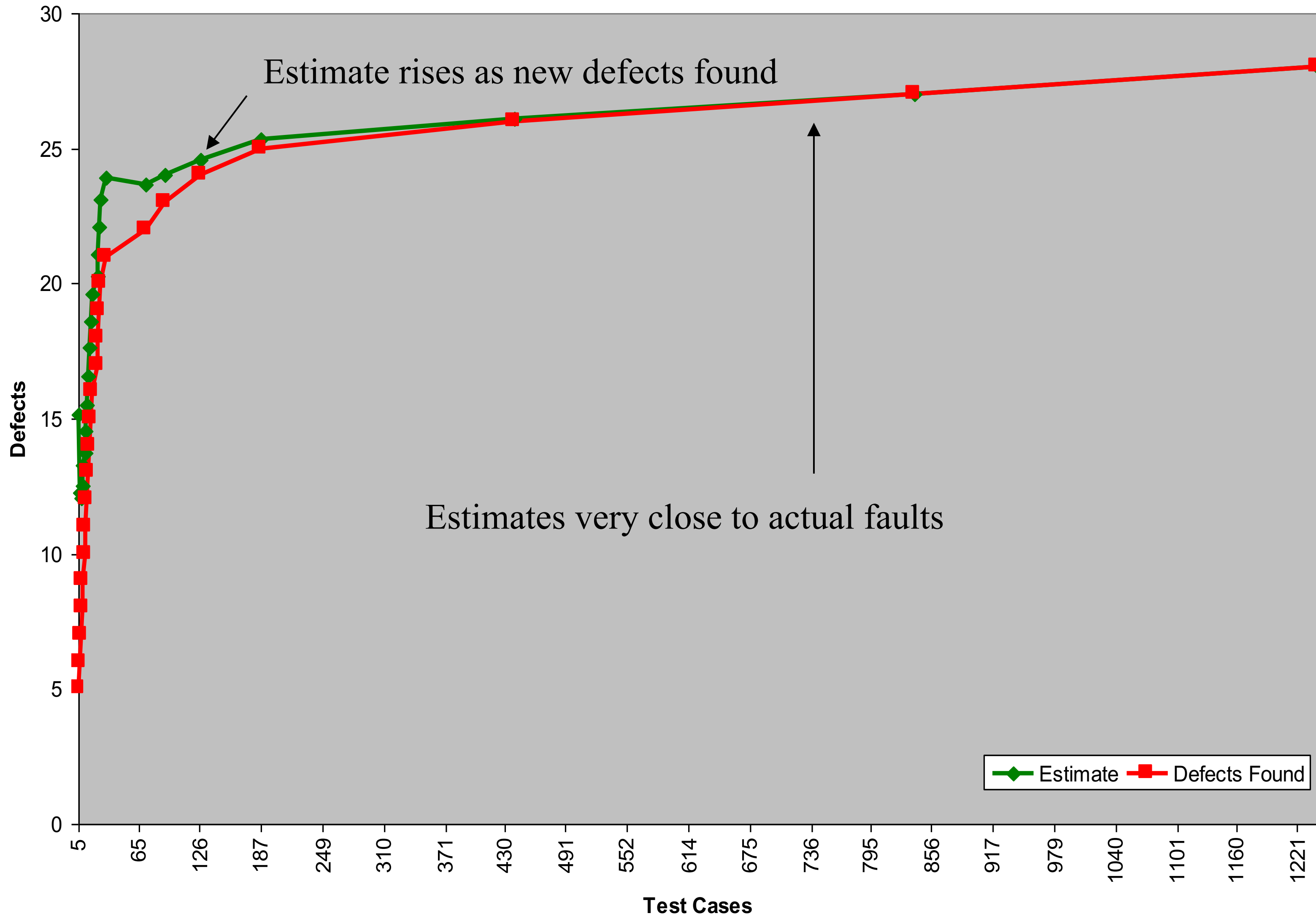
Coverage Based Estimation



Current Methods

- Development process based models allow for *a priori* estimates
  - Not as accurate as methods based on test data
- Sampling methods often assume faults found as easy to find as faults not found
  - Underestimates faults
- Exponential model
  - Assume applicability of exponential model
  - We present results of a comparison

The Exponential Model



Colorado State University

Fuzzing and Coverage

- Directed Fuzzing is used for guiding vulnerability discovery.
- Fuzzing is directed using test coverage.

Related articles

- Frankl & Iakouneno, Proc. SIGSOFT '98
  - 8 versons of European Space Agency program, 10K LOC, Single fault reinsertion
- Williams, Mercer, Mucha, Kapur, 2001
  - "Code coverage, what does it mean in terms of quality?,"
  - analysis from first principles
- Peter G Bishop, SAFECOMP 2002
  - A related model, unreachable code
- Mockus, A.; Nagappan, N.; Dinh-Trong, T.T., "Test coverage and post-verification defects: A multiple case study," ESEM 2009.
  - Avaya lab data
  - "The test effort increases exponentially with test coverage, but the reduction in field problems increases linearly with test coverage."

Related articles

- Mockus, A.; Nagappan, N.; Dinh-Trong, T.T., "Test coverage and post-verification defects: A multiple case study," Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on , vol., no., pp.291,301, 15-16 Oct. 2009
- Avaya lab data
- *"The test effort increases exponentially with test coverage, but the reduction in field problems increases linearly with test coverage."*