Software Reliability & Security Engineering

Yashwant K Malaiya
BMAC Sept 11, 07

- Demarcating, measuring, counting: definitions
- Science & engineering of reliability growth
- Those pesky residual defects
- Components & systems
- Vulnerability: special defects

Time to go Wholistic

- We have data on different aspects of reliability to have reasonable hypotheses.
- We know limitations of the hypotheses.
- We have enough techniques & tools to start engineering.
- Accuracy **comparable or better** than established hardware reliability methods.

Why It's Needed Now

- **Craft**: incremental intuitive refinement
- **Science**: ***why*** it is so
  - Observe, hypothesize, assess accuracy
- **Engineering**: ***how*** to get what we want
  - Approximate, integrate, evaluate
- Are we ready to engineer software reliability?
  - Reliability expectations growing fast
  - Large projects, little time
  - Quick changes in developing environments
  - Reliance on a single technique not enough
  - Pioneering work has already been done.

Hardware Reliability: The Status

- Well known , well established methods
  - Now standard practice
  - Used by government and industrial org worldwide
- Earliest tube computers: MTTF comparable to some computation times!
- 1959-95: MIL-HDBK-217A-F, Still widely used.
  - Failure rates predicted higher by a factor of 2-4.
  - Constant failure-rate, the bathtub curve, the Arrhenius relationship have been questioned.
- Several commercial tools available.

Hardware Reliability: The Status (2)

- Why use hardware reliability prediction?
  - Feasibility Study: initial design
  - Compare Design Alternatives: Reliability along with performance and cost
  - Find Likely Problem Spots- high contributors to the product failure rate
  - Track Reliability Improvements

Hardware vs Software Reliability

	Models	Parameters
Hardware	Past experience with similar units	Past experience with similar units
Software	Past experience* with similar units	**Early:** past experience with similar units **Later:** from the same unit

* Also suggested: from the same unit

Basic Definitions

- Defect: requires a corrective action
- Defect density: defects per 1000 non-comment source lines.
- Failure intensity: rate at which failures are encountered during execution.
- MTTF (mean time to failure): inverse of failure intensity

Basic Definitions (2)

- Reliability
  - R(t)=p{no failures in time (0,t)}

Limited use in SRE

- Transaction reliability: probability that a single transaction will be executed correctly.
- Time: may be measures in CPU time or some measure of testing effort.

Why is Defect Density Important?

- Important measurement of reliability
- Often used as release criteria

Beginning Of Unit Testing	Release		
	Frequently Cited	Highly Tested	NASA
16	2.0	0.33	0.1

Static and Dynamic Modeling

- Reliability at release depends on
  - Initial number of defects (parameter)
  - Effectiveness of defect removal process (parameter)
  - Operating environment
- **Static modeling:** estimate parameters before testing begins
  - Use static data like software size etc.
- **Dynamic modeling:** estimate parameters during testing
  - Record when defects are found etc.
  - *Time* or *coverage* based

What factors control defect density?

- **Need to know for**
  - *static estimation of initial defect density*
  - *Find room for process improvement*
- **Static defect density models**:
  - *Additive (ex: Takahashi-Kamayachi)*
    $D=a_1f_1+a_2f_2+a_3f_3\ldots$
  - *Multiplicative (ex. MIL-HDBK-217, COCOMO, RADC)*
    $D=C.F_1(f1).F_2(f_2).F_3(f_3)\ldots$

A Static Defect Density Model

- Li, Malaiya, Denton (93, 97)
- $\boxed{D=C.F_{ph}.F_{pt}.F_m.F_s.F_{rv}}$
- *C is constant of proportionality, based on prior data.*
- *Default value of each function (submodel) is 1.*
- *Calibration based on past, similar projects*

Phase
Programming Team
Process Maturity
Structure
Requirement Volatility

Submodels: F_{ph}, F_{pt}

- Phase Factor F_{ph} Based on Musa, Gaffney, Piwowarski et al.

At beginning of phase	Multiplier
Unit testing	4
Subsystem testing	2.5
System testing	1 (default)
Operation	0.35

- Programming Team Factor F_{pt} : Based on Takahashi, Kamayachi. Decline by 14% per year up to seven years.

Team's average skill level	Multiplier
High	0.4
Average	1 (default)
Low	2.5

Process Maturity Factor F_m

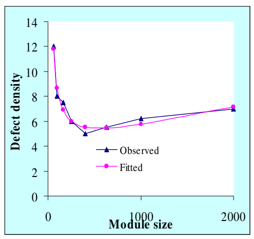
SRI Capability Maturity Model

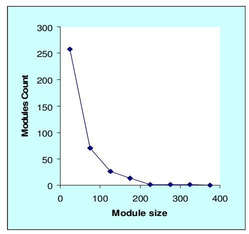
- Statistical quality control (Deming's TQM, Juran, Crosby),
- Process Maturity Factor F_m : data by Jones, Keene, Motorola

Level	Feature	How many	Multiplier F_m
1.Initial	ad hoc	80%	1.5
2. Repeatable	basic management	15%	1
3. Defined	standardized	5%	0.4
4. Managed	quantitative control	4%	0.1
5. Optimizing	continuous improvement	Handful	0.05

Submodel: Structure Factor F_s

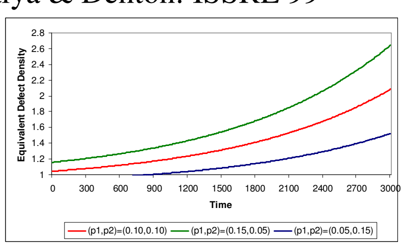
- Assembly code fraction: assuming assembly has 40% more defects
  - Factor=1+0.4×fraction in assembly
- Complexity: Complex modules are more fault prone, but there may be compensating factors.
- Module size: research reported at ISSRE 2000!

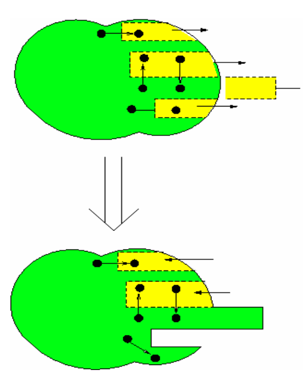




Submodel: Requirement volatility Factor F_{ph}

- Degree of changes and when they occur
- Most impact when changes occur near the end of testing
- Malaiya & Denton: ISSRE 99





Using the Defect Density Model

- Calibrate submodels before use using data from a project as similar as possible.
- Constant C can range between 6-20 (Musa).
- Static models are very valuable, but high accuracy is not expected.
- Useful when dynamic test data is not yet significant.

For an organization, C is between 12 and 16. Average team and SEI maturity level is II. About 20% of code in assembly. Other factors are average (or *same as past projects*).

Estimate defect density at beginning of subsystem test phase.

•Upper estimate=16×2.5×1×1×(1+0.4 ×0.2)×1=43.2/KSLOC

•Lower estimate= 12x2.5x1x1x(1+0.4x0.2)x1=32.4/KLOC

Test methodologies

- **Static** (review, inspection) vs. **dynamic** (execution)
- **Test views**
  - Black-box (functional): input/output description
  - White box (structural): implementation used
- **Test generation**
  - Partitioning
  - Random/Antirandom/Deterministic
- **Input mix**

Operational Profile

- **Profile:** set of disjoint actions, operations that a program may perform, and their probabilities of occurrence.
- **Operational profile**: probabilities that occur in actual operation
  - Begin-to-end operations & their probabilities
  - Markov: states & transition probabilities
- There may be multiple operational profiles.
- Accurate operational profile determination may not be needed.

A	Voice call	0.74
B	FAX call	0.15
C	New number entry	0.10
D	Data base audit	0.009
E	Add subscriber	0.0005
F	Delete subscriber	0.00049 9
G	Hardware failure recovery	0.00000 1

- *"Phone follower" call types (Musa)*

Input Mix: Operational Profile?

- **Need to do**
  - find bugs fast?
  - estimate operational failure intensity?
- Best mix for efficient bug finding (Li & Malaiya)
  - Quick & limited testing: *Use operational profile*
  - High reliability: *Probe input space evenly*
    - Operational profile will not execute rare and special cases
  - In general: Use combination
- For acceptance testing: Need Operational profile

Modeling Reliability Growth

- Testing cost can be 60% or more
- Careful planning to release by target date
- Decision making using a *software reliability growth model* (SRGM). Obtained using
  - Analytically using assumptions, or and
  - Based on experimental observation
- A model describes a real process approximately
- Ideally should have good predictive capability and a reasonable interpretation

Exponential Reliability Growth Model

- We need a model to describe
  - **Undetected defects present at time t:** N(t)
  - **Faults** detected by time t: $\mu(t) = N_0(t) - N(t)$
  - **Failure intensity: fault detection rate** $\lambda(t)$
- **Note that**

$$\lambda(t) = \frac{d}{dt}\mu(t)$$

Exponential SRGM (cont)

- Assumption: $$-\frac{dN(t)}{dt} = \beta_1 N(t)$$
- Solution: $$N(t) = N(0)\, e^{-\beta_1 t}$$

$$\mu(t) = \beta_o (1 - e^{-\beta_1 t}) \qquad \lambda(t) = \beta_o \beta_1 e^{-\beta_1 t}$$

- For $t \rightarrow \infty$, total $\beta_o = N(0)$ faults would be eventually detected.
  - A *"finite-faults-model"*.
  - Jelinski-Muranda '71, Shooman '71, Goel-Okumoto '79 and Musa '75-'80. also called Basic

A Basic SRGM (Cont.)

- **Parameter β_1 is given by:**

$$\beta_1 = \frac{K}{T_L} = \frac{K}{(S.Q.\frac{1}{r})}$$

- S: source instructions,
- Q: number of object instructions per source instruction,
- r: object instruction execution rate of the computer
- K: *fault-exposure ratio*, range $1\text{x}10^{-7}$ to $10\text{x}10^{-7}$, (t is in CPU seconds). Assumed constant here.

SRGM : “Logarithmic Poisson”

- Many SRGMs have been proposed.
- **Logarithmic** model, by Musa-Okumoto, 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. Term *infinite-faults-model* misleading.
- Parameters β_o and β_1 don’t have a simple interpretation. A useful interpretation by Malaiya and Denton.

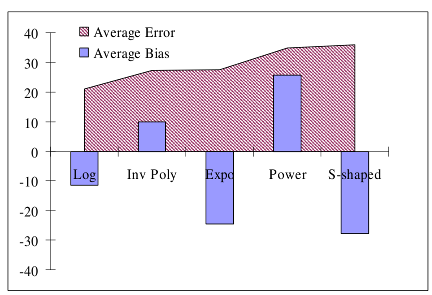
Comparing Models

- Goodness of fit: may be misleading
- Predictive capability
  - Data points: (λ_i, t_i), i=1 to n
  - Total defects found: D, estimated at i: D_i

$$\text{Average error : AE} = \frac{1}{n}\sum_{i=1}^{n-1} \left| \frac{D_i - D}{D} \right|$$

$$\text{Average bias : AB} = \frac{1}{n}\sum_{i=1}^{n-1} \frac{D_i - D}{D}$$

- We used a many datasets from diverse projects for comparing different models.



SRGM: Preliminary Planning: Example

- Example:
  - initial defect density estimated 25 defects/KLOC
  - 10,000 lines of C code
  - computer 70 million object instructions per second
  - *fault exposure ratio* K estimated to be 4×10^{-7}
  - Estimate the testing time for defect density 2.5/KLOC
- Procedure:
  - $\beta_0 = 250$ defects, $\beta_1 = 11.2 \times 10^4$ per sec
  - Thus Testing time $t_1 = 2056$ sec.

Value of fault exposure ratio (K) may depend on initial defect density and testing strategy (Li, Malaiya '93).

SRGM: During Testing

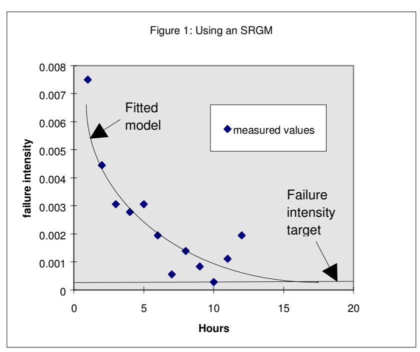
- Collect and pre-process data:
  - To extract the long-term trend, data needs to be smoothed
  - *Grouped* data: test duration intervals, average failure intensity in each interval.
- Select a model and determine parameters:
  - past experience with projects using same process
  - exponential and logarithmic models often good choices
  - model that fits early data well, may not have best predictive capability
  - parameters estimated using *least square* or *maximum likelihood*
  - parameter values used when *stable* and *reasonable*

SRGM: During Testing (cont.)

- Compute how much more testing is needed:
  - fitted model to project additional testing needed
    - desired failure intensity
    - estimated defect density
  - recalibrating a model can improve projection accuracy
  - Interval estimates can be obtained using statistical methods.

Example: SRGM with Test Data

CPU Hours	Failures
1	27
2	16
3	11
4	10
5	11
6	7
7	2
8	5
9	3
10	1
11	4
12	7



- Target failure intensity 1/hour ($2.78.10^{-4}$ per sec.)
- β_0 = 101.47 and β_1 = 5.22 x 10-5
- Stopping time t_f = 56,473 sec

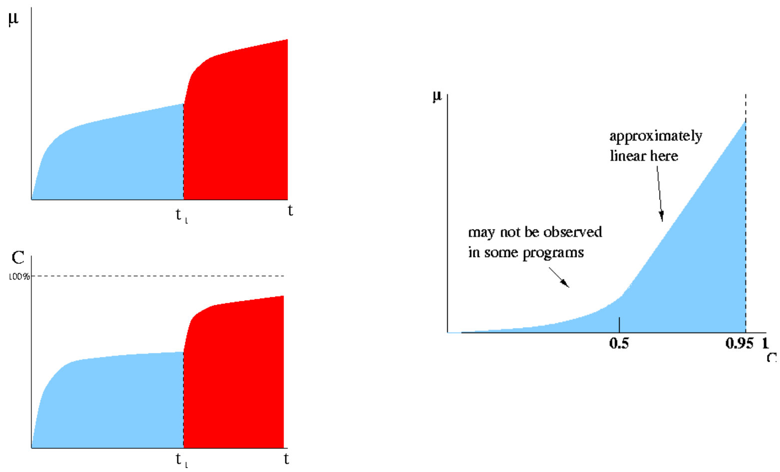
Test Coverage & Defect Density: Yes, they are related.

- Defect vs. Test Coverage model, 1994:
  - Malaiya, Li, Bieman, Karcich, Skibbe
- Estimation of number of defects, 1998
  - Li, Malaiya, Denton

Test Coverage Measures

- Statement or Block coverage
- Branch or decision coverage
- P-use coverage: p-use pair: variable defined/modified - use as predicate
- Subsumption hierarchy:
  - Covering *all branches* cover *all statements*
  - Covering *all p-uses* cover *all branches*

Modeling : Defects, Time, & Coverage



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,

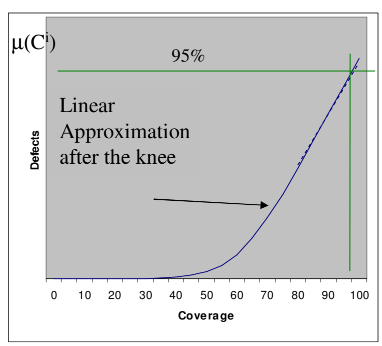
$$C^0 = a_0^i \ln[1 + a_1^i(\exp(a_2^i C^i) - 1], \quad C^0 \le 1$$

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

- For "large" $C_i >> C_{knee}$, we can approximate

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

- Assumes stable software

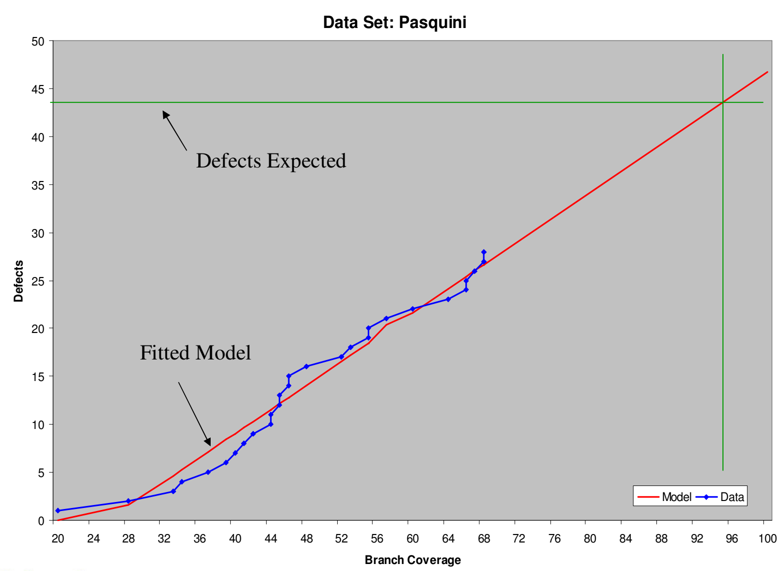


Location of knee based on initial defect density
Lower defect densities cause knee to occur at higher coverage
Malaiya and Denton (HASE '98)

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. Coverage

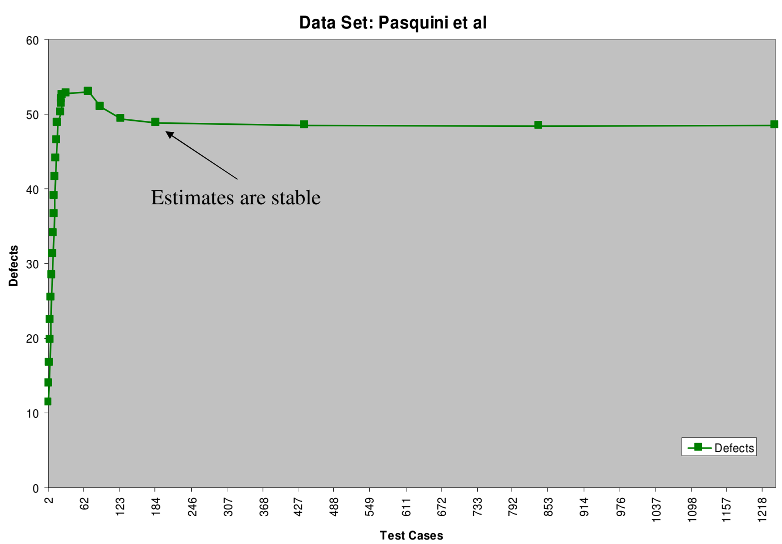


Estimation of Defect Density

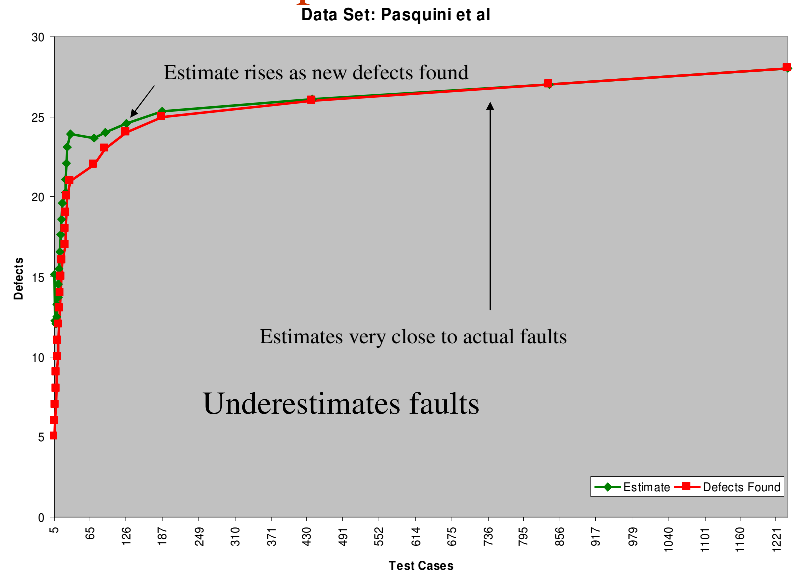
- Estimated defects at 95% coverage, for Pasquini data (assume 5% *dead code*)
- 28 faults found, and 33 known to exist

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

Comparison: Coverage Based Estimation



The Exponential Model



9/27/2007 FTC- YKM 41

Recent Conformation of Model

- Frankl & Iakouneno, Proc. SIGSOFT '98
  - 8 versons of European Space Agency program, 10K LOC
  - Single fault reinsertion
- Tom Williams, 1999
  - analysis from first principles

9/27/2007 FTC- YKM 42