Wholistic Engineering for Software Reliability

ISSRE 2000 Tutorial

Yashwant K. Malaiya
Colorado State University

- malaiya@cs.colostate.edu
- http://www.cs.colostate.edu/testing/

Outline

- Why it's time…
- Demarcating, measuring, counting: definitions
- Science & engineering of reliability growth
- Those pesky residual defects
- Components & systems
- The toolbox

Why It's Time: Emergence of SRE

- **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?

Why it's time ..

- We have data on different aspects of reliability to have reasonable hypotheses.
- We know limitations of the hypotheses.
- We will always need more data and better hypotheses …
- We have enough techniques & tools to start engineering.

Why It's Needed Now

- 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.

Learning from Hardware Reliability

- Well known , well established methods
- Now standard practice
- Used by government and industrial org worldwide
- Considered a *hard science* compared with software reliability

Hardware Reliability: The Status (1)

- Earliest tube computers: MTTF comparable to some computation times!
- 1956 RCA TR-1100: component failure rate models
- 1959: MIL-HDBK-217A: common failure rate: 0.4×10^{-6} *for all ICs for all cases*
- Revised about every 7 years

Hardware Reliability: The Status (2)

- 1995 Final update: MIL-HDBK-217F, Notice 2. Still widely used.
- Failure rates predicted often higher by a factor of 2-4, occasionally by an order of magnitude.
- Constant failure-rate, the bathtub curve, the Arrhenius relationship have been questioned.

Hardware Reliability: The Status (3)

- 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

Wholistic Engineering for Software Reliability

Outline

- Why it's time…
- Demarcating, measuring, counting: definitions
- Science & engineering of reliability growth
- Those pesky residual defects
- Components & systems
- The toolbox

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)

Limited use in SRE

- Reliability
  - R(t)=p{no failures in time (0,t)}
- Transaction reliability: probability that a single transaction will be executed correctly.
- Time: may be measures in CPU time or some measure of testing effort.

Wholistic Engineering for Software Reliability

Outline

- Why it's time…
- Demarcating, measuring, counting: definitions
- Science & engineering of reliability growth
- Those pesky residual defects
- Components & systems
- The toolbox

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)
- $D = C.F_{ph}.F_{pt}.F_{m}.F_{s}.F_{rv}$
- *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

Submodel: 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

Submodel: 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

Submodel: Process Maturity Factor F_m

- Based on Jones, Keene, Motorola data.

SEI CMM Level	Multiplier
Level 1	1.5
Level 2	1 (default)
Level 3	0.4
Level 4	0.1
Level 5	0.05

Submodel: Structure Factor F_s

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

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.

Static Model: Example

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=$16\times2.5\times1\times1\times(1+0.4\times0.2)\times1=43.2$/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
  - Combination: *white after black*
- **Test generation**
  - Partitioning
  - Random/Antirandom/Deterministic
- **Input mix**

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

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.

Operational Profile Example

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

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	0.000499
G	Hardware failure recovery	0.000001

A1	Voice call, no pager, answer	0.18
A2	Voice call, no pager, no answer	0.17
A3	Voice call, pager, voice answer	0.17
A4	Voice call, pager, answer on page	0.12
A5	Voice call, pager, no answer on page	0.10

Wholistic Engineering for Software Reliability

Outline

- Why it's time…
- Demarcating, measuring, counting: definitions
- Science & engineering of reliability growth
- Those pesky residual defects
- Components & systems
- The toolbox

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
  - Based on experimental observation
- A model describes a real process approximately
- Ideally should have good predictive capability and a reasonable interpretation

A Basic SRGM

- Testing time t, CPU execution time, man-hours etc.
- *Total expected faults* detected by time t: μ(t)
- **Failure intensity**

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

- **Defects present at time t: N(t)**

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

A Basic SRGM (Cont.)

- **Parameter β_1 is given by:**

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

- S: source instructions,
- Q: number of object instructions/r source instruction,
- r: object instruction execution rate of the computer
- K: *fault-exposure ratio*, range $1x10^{-7}$ to $10x10^{-7}$, (when t is in CPU seconds)

A Basic SRGM (Cont.)

- **We get**

$$N(t) = N(0)e^{-\beta_1 t}$$

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

- Where β_o=N(0), total faults that would be eventually detected
- Assumes no new defects are generated during debugging.
- "**Exponential model**": Jelinski-Muranda '71, Shooman '71, Goel-Okumoto '79 and Musa '75-'80.

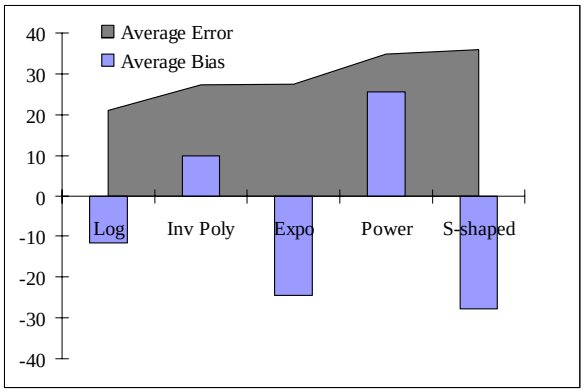
SRGMs (Log Poisson)

- Many SRGMs have been used.
- 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 m(t) $\leq$ N(0). In practice almost always satisfied.
- parameters β_o and β_1 don't have a simple interpretation. A useful interpretation by Malaiya and Denton.

Bias in SRGMs



•Malaiya, Karunanithi, Verma ('90)

SRGM: Preliminary Planning

- 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:
  - Find β_0, β_1
  - Find testing time t_1

SRGM: Preliminary Planning (cont.)

- From exponential model

$$\beta_o = N(0) = 25x10 = 250 \text{ defects},$$

$$\beta_1 = \frac{K}{(S.Q.\frac{1}{r})} = \frac{4.0 \times 10^{-7}}{10{,}000 \times 2.5 \times \frac{1}{70 \times 10^6}}$$

$$= 11.2 \times 10^{-4} \text{ per sec}$$

SRGM: Preliminary Planning (cont.)

- Reliability at release depends on

$$\frac{N(t_1)}{N(O)} = \frac{2.5 \times 10}{25 \times 10} = \exp(-11.2x10^{-4}.t_1)$$

$$t_1 = \frac{-\ln(0.1)}{11.2 \times 10^{-4}} = 2056 \text{ sec.} (CPU\ time)$$

answer

$$\lambda(t_1) = 250 \times 11.2 \times 10^{-4}\, e^{-11.2 \times 10^{-4} t_1}$$

$$= 0.028 \text{ failures/ sec}$$

SRGM: Preliminary Planning (cont.)

- For the same environment, β_1.S is constant.
  - Prior 5 KLOC project β_1 was $2x10^{-3}$ per sec.
  - New 15 KLOC project, β_1 can be estimated as $2x10^{-3}/3 = 0.66x10^{-3}$ per sec.
- Value of fault exposure ratio (K) may depend on initial defect density and testing strategy (Li, Malaiya '93).

SRGM: During Testing

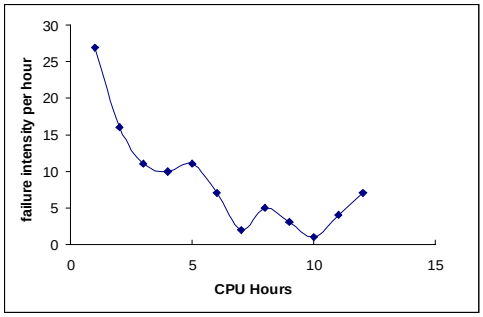
- Collect and pre-process data:
  - To extract the long-term trend, data needs to be smoothed
  - *Grouped* data: test duration intervals, average failure intensity in each interval.
- Select a model and determine parameters:
  - past experience with projects using same process
  - exponential and logarithmic 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.)

Example: SRGM with Test Data (cont.)

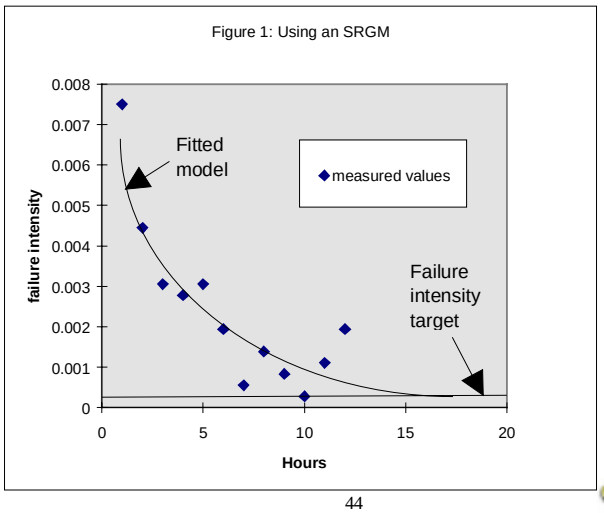
- Fitting we get

  $\beta_o = 101.47$ and $\beta_1 = 5.22 \times 10^{-5}$
- stopping time t_f is then given by:

$$2.78 \times 10^{-4} = 101.47 \times 5.22 \times 10^{-5} e^{-5.22 \times 10^{-5} \times t_f}$$

- yielding t_f = 5 6, 4 73 sec., i.e. 15.69 hours

Example: SRGM with Test Data (cont.)



Example: SRGM with Test Data (cont.)

- Accuracy of projection:
  - Experience with Exponential model suggests
  - estimated β_o tends to be lower than the final value
  - estimated β_1 tends to be higher
  - true value of t_f should be higher. Hence 15.69 hours should be used as a lower estimate.
- Problems:
  - test strategy changed: spike in failure intensity
    - smoothing
  - software under test evolving - continuing additions
    - Drop or adjust early data points

Wholistic Engineering for Software Reliability

Outline

- Why it's time…
- Demarcating, measuring, counting: definitions
- Science & engineering of reliability growth
- Those pesky residual defects
- Components & systems
- The toolbox

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

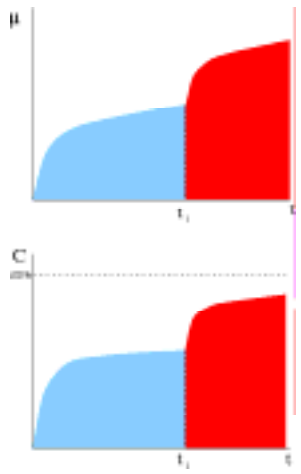
- Defect vs. Test Coverage model, 1994:
  - Malaiya, Li, Bieman, Karcich, Skibbe
- Estimation of number of defects, 1998
  - Li, Malaiya, Denton

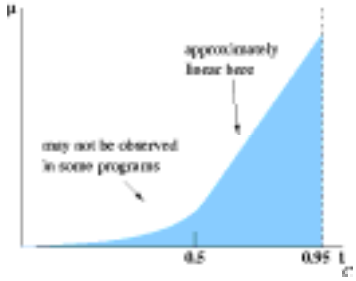
Motivation
Why is Defect Density Important?

- 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

Modeling : Defects, Time, & Coverage

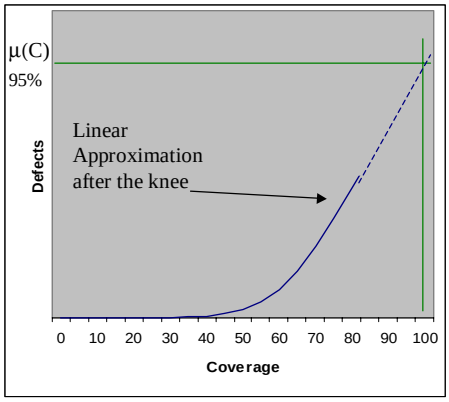




Coverage Based Defect Estimation

- 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

Coverage Model, Estimated Defects



$$\mu(C) = A_0 + A_1 C, \quad C > C_{knee}$$

- Only applicable after the knee
- Assumptions : Logarithmic Poisson Model for defects and coverage elements. Stable Software

Location of the knee

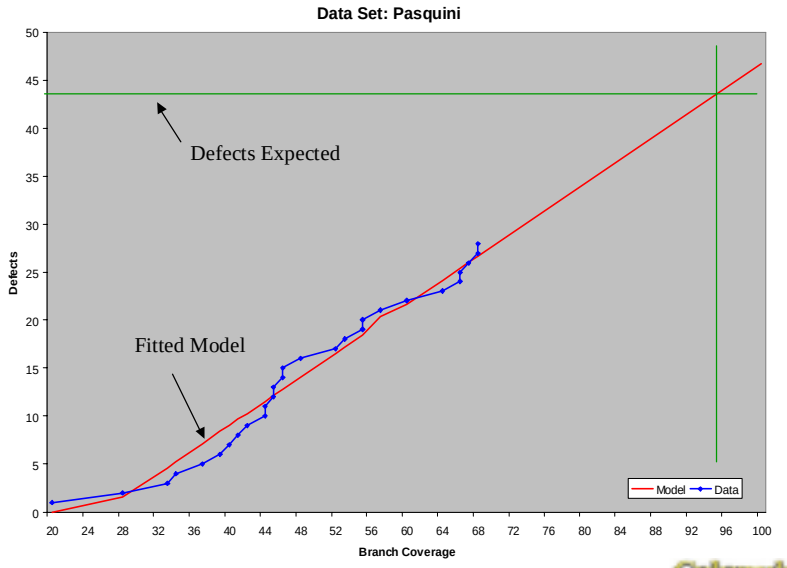
$$C_{knee} < 1 - \left(\frac{E_{\min}}{D_{\min} E_0} \right) D_0$$

- 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)

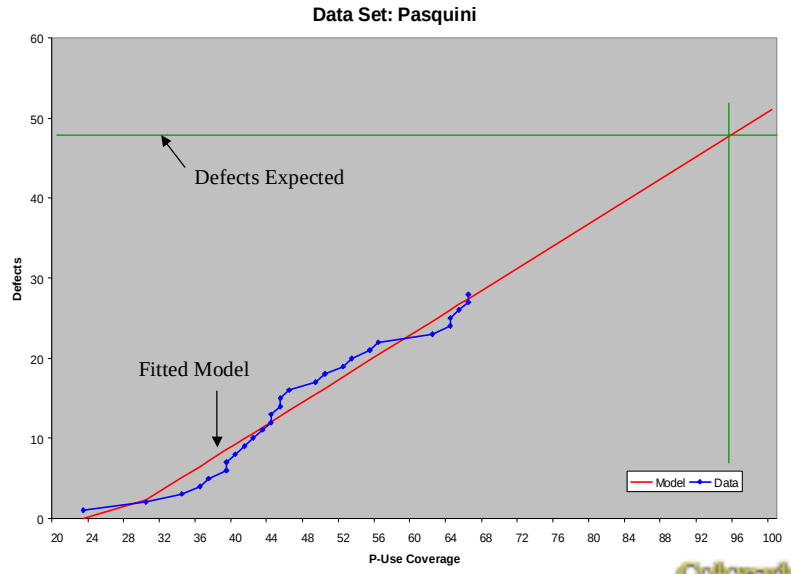
Data Sets Used
Vouk and Pasquini

- 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

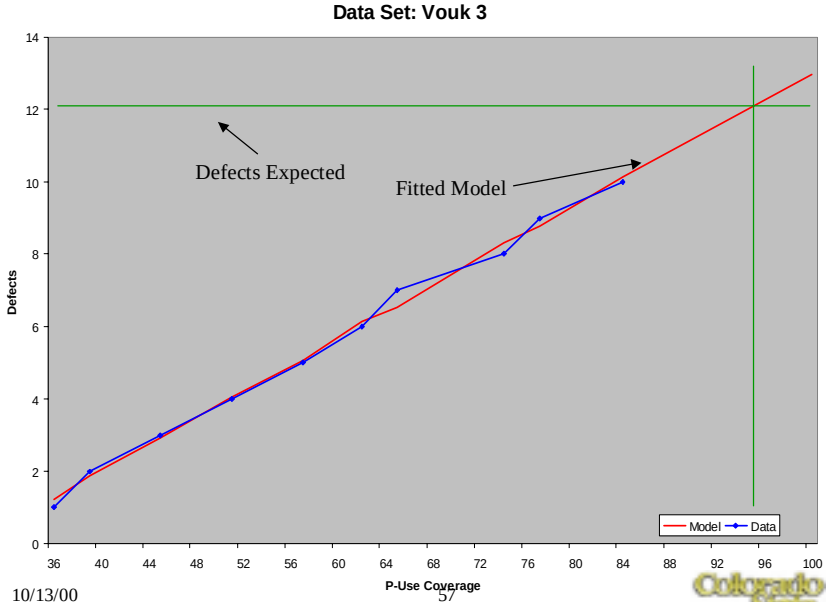


Estimation of Defect Density

- Estimated defects at 95% coverage, for Pasquini data
- 28 faults found, and 33 known to exist

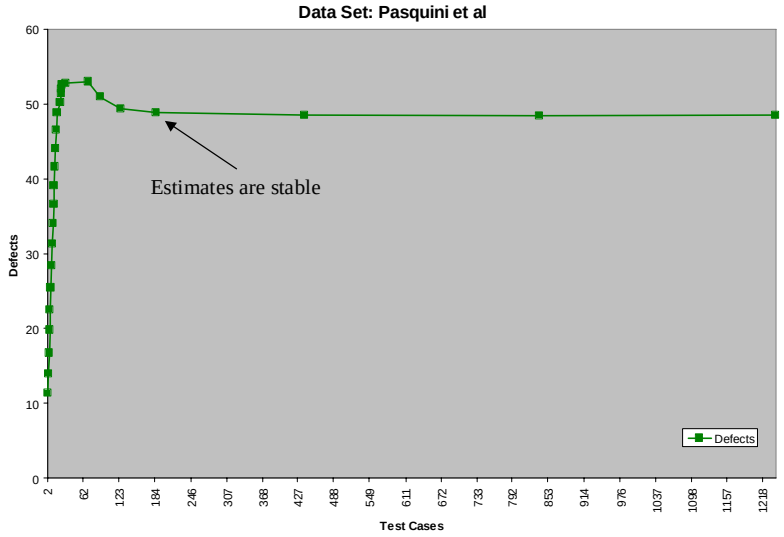
Measure	Coverage Achieved	Expected Defects
Block	82%	36
Branch	70%	44
P-uses	67%	48

Defects vs. P-Use Coverage



10/13/00 57 ISSRE'00 ©Y.K. Malaiya

Coverage Based Estimation

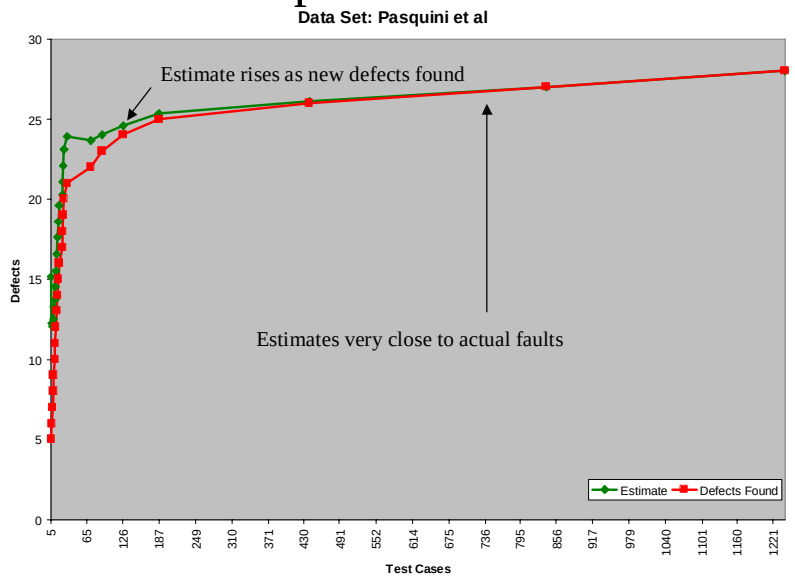


10/13/00 58 ISSRE'00 ©Y.K. Malaiya

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



Recent Conformation of Model

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

Observations and Conclusions

- Estimates with new method are very stable
  - Visual confirmation of earlier projections
- Which coverage measure to use?
  - Stricter measure will yield closer estimate
- Some code may be dead or unreachable
  - Found with compile or link time tools
  - May need to be taken into account

Wholistic Engineering for Software Reliability

Outline

- Why it's time…
- Demarcating, measuring, counting: definitions
- Science & engineering of reliability growth
- Those pesky residual defects
- Components & systems
- The toolbox

Reliability of Multi-component Systems

- Software system: number of modules.
- Individual modules developed and tested differently: different defect densities and failure rates.
  - **Sequential execution**
  - **Concurrent execution**
  - **N-version systems**

Sequential execution

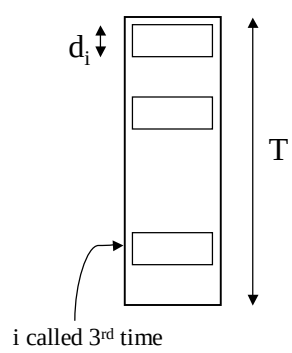
- Assume one module executed at a time.
- f_i: fraction of time module i under execution; λ_i its failure rate
- Mean system failure rate:

$$\lambda_{sys} = \sum_{i=1}^{n} f_i \lambda_i$$

Sequential Execution (cont.)

- T: mean duration of a single transaction
- module i is called e_i times during T, each time executed for duration d_i

$$f_i = \frac{e_i \bullet d_i}{T}$$



Sequential Execution (cont.)

- System reliability $R_{sys} = \exp(-\lambda_{sys} T)$

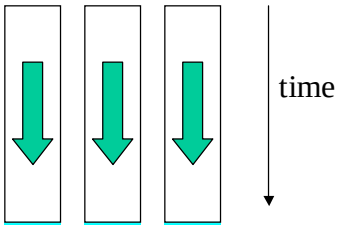
$$R_{sys} = \exp\left(-\sum_{i=1}^{n} e_i \, d_i \, \lambda_i\right)$$

- Since $\exp(-d_i\lambda_i)$ is R_i,

$$R_{sys} = \prod_{i=1}^{n} (R_i)^{e_i}$$

Concurrent execution

- Concurrently executing modules: all run without failures for system to run
- j concurrently executing modules

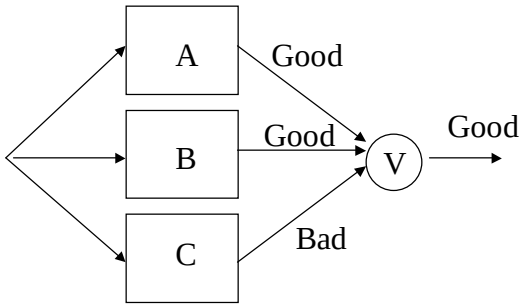


$$\lambda_{sys} = \sum_{j=1}^{m} \lambda_j$$

N-version systems

- Critical applications, like defense or avionics
- Each version is implemented and tested independently
- Common implementation uses triplication and voting on the result

N-version Systems (Cont.)



$R_{sys}=1-(1-R)^3-3R(1-R)^2$

$R=0.9 \Rightarrow Rsys=.972$

N-version systems: Correlation

- Correlation significantly degrades fault tolerance
- Significant correlation common in N-version (Knight-Leveson)
- Is it cost effective?

N-version systems: Correlation

- 3-version system
- q_3: probability of all three versions failing for the same input.
- q_2: probability that any two versions will fail together.
- Probability P_{sys} of the system failing

$$P_{sys} = q_3 + 3q_2$$

N-version systems: Correlation

- Example: *data collected by Knight-Leveson; computations by Hatton*
- *3-version system, probability of a version failing for a transaction 0.0004*
- *in the absence of any correlated failures*

$$P_{sys} = (0.0004)^2 + 3(1 - 0.0004)(0.0004)^2$$
$$= 4.8 \times 10^{-7}$$

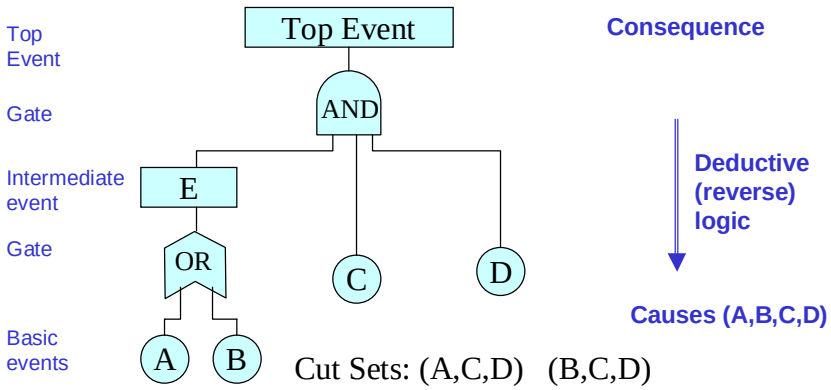
N-version systems: Correlation

- Uncorrelated improvement factor of $0.0004/4.8 \times 10^{-7} = 833.3$
- Correlated: $q_3 = 2.5\times10^{-7}$ and $q_2 = 2.5\times10^{-6}$
- $P_{sys} = 2.5\times10^{-7} + 3.2.5\times10^{-6} = 7.75\times10^{-6}$
- improvement factor: $0.0004/7.75\times10^{-6}$= **51.6**
- state-of-the-art techniques can reduce defect density by a factor of **10**

Safety

- Analyze system to identify possible conditions leading to unsafe behavior.
- Eliminate or reduce the probability of occurrence of such events.
- Safety involves only a part of the system functionality.

Fault Tree Analysis



Cut Sets: (A,C,D) (B,C,D)

$P_{TE}=P_A P_C P_D + P_B P_C P_D - P_A P_B P_C P_D$

Using Fault Trees

- Deterministic analysis: prove that occurrence of unsafe events implies a logical contradiction. Feasible for small programs.
- Probabilistic analysis: compute probability of occurrence of an unsafe event. Software, hardware and human factors.

Hazard Criticality Index Matrix

	Frequent	Probable	Occasional	Remote	Improbable	Impossible
Catastrophic	1	2	3	4	9	12
Critical	3	4	6	7	12	12
Marginal	5	6	8	10	12	12
Negligible	8	11	12	12	12	12

Risk = frequency (*events/unit time*)× severity (*detriment/event*)

Example from Navy 1986

Hazard Probability

Frequent	MTBH<<UL
Probable	MTBH<UL
Occasional	MTBH≈UL
Remote	MTBH>UL
Improbable	MTBH>>UL
Impossible	Probability 0

MTBH: Mean time to hazard, UL: Unit life
Compare with MIL-STD-882D App.A

Wholistic Engineering for Software Reliability

Outline

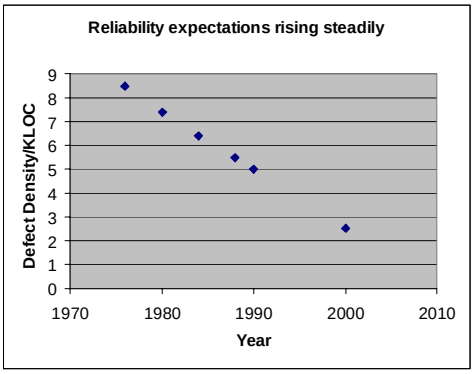
- Why it's time…
- Demarcating, measuring, counting: definitions
- Science & engineering of reliability growth
- Those pesky residual defects
- Components & systems
- The toolbox

Tools

For Automating Software Reliability Engineering

- Can we eliminate debugging?
- Bugs would occur even with formal methods like VDM and Z [McGibbon]
- hardware design and test: tools is now regarded to be mandatory
- Software: increasing dependence

Why Tools Will be Mandatory



Source: Poston & Sexton

Software Testing Tools: History

- 70s: LINT: picks out all the fuzz
- 74: code instrumentor JAVS for coverage
- 80s: capture-replay etc.
- 92: Memory leak defectors
- Late 90s: Y2K tools

Manual vs. automated testing (QAI)

Test step	Manual testing	Automated testing	Percent Improvement
Test plan development	32	40	-25%
Test case development	262	117	55%
Test execution	466	23	95%
Test result analyses	117	58	50%
Defect tracking	117	23	80%
Report creation	96	16	83%
Total hours	1090	277	75%

Tools for all Phases

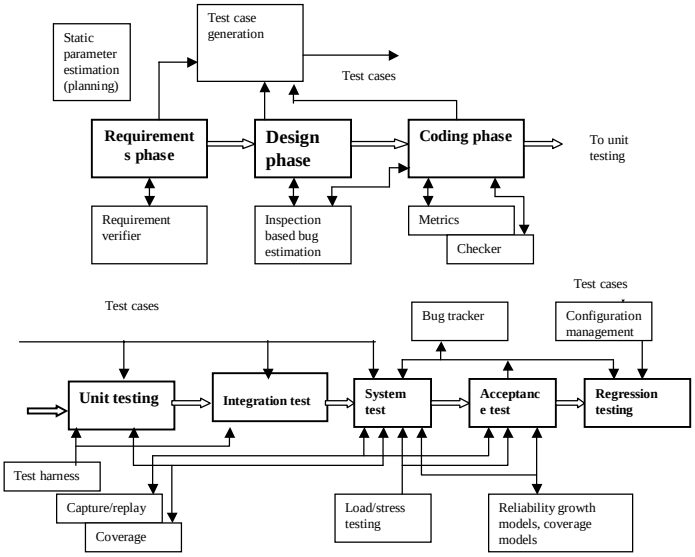
- Requirements phase Tools
  - Requirement Recorder/Verifier
  - Test Case Generation
- Programming Phase Tools (Static tools)
  - Metrics Evaluators
  - Code Checkers:
  - Inspection Based Error Estimation

Tools for all Phases (cont.)

- **Testing Phase Tools**
  - Capture-Playback Tool
  - Memory Leak Detectors
  - Test Harness:
  - Coverage Analyzers
  - Load/performance tester
  - Bug-tracker

Tools for all Phases (cont.)

- **Testing Phase Tools** (cont.)
  - Defect density estimation
  - Reliability Growth Modeling tools
  - Coverage based Reliability Tools
  - Fault tree analysis
  - Markov reliability Evaluation



Tool Costs

- Tool identification
- Tool acquisition
- Tool installation/maintenance
- Study of underlying principles
- Familiarity with operation
- Risk of non-use
- Contacting user groups/support

References

- J. D. Musa, A. Ianino and K. Okumoto, *Software Reliability-Measurement, Prediction, Applications*, McGraw-Hill, 1987.
- Y. K. Malaiya and P. Srimani, Ed., *Software Reliability Models*, IEEE Computer Society Press, 1990.
- A. D. Carleton, R. E. Park and W. A. Florac, *Practical Software Measurement*, Tech. Report, SRI, CMU/SEI-97-HB-003.
- P. Piwowarski, M. Ohba and J. Caruso, ``Coverage Measurement Experience during Function Test,'' Proc. Int. Conference on Software Engineering, 1993, pp. 287-301.
- Y. K. Malaiya, N. Li, J. Bieman, R. Karcich and B. Skibbe ``The Relation between Test Coverage and Reliability ,'' Proc. IEEE-CS Int. Symposium on Software Reliability Engineering, Nov. 1994, pp. 186-195.

References

- Y.K. Malaiya and J. Denton, ``What do the Software Reliability Growth Model Parameters Represent,'' Proc. IEEE-CS Int. Symposium on Software Reliability Engineering ISSRE, Nov. 1997, pp. 124-135.
- M. Takahashi and Y. Kamayachi, ``An Emprical study of a Model for Program Error Prediction,'' Proc. Int. Conference on Software Engineering, Aug. 1995, pp. 330-336.
- J. Musa, Software Reliability Engineering'', McGraw-Hill 1999.
- N. Li and Y. K. Malaiya, ``Fault Exposure Ratio: Estimation and Applications,'' Proc. IEEE-CS Int. Symposium on Software Reliability Engineering, Nov. 1993, pp. 372-381.
- N. Li and Y. K. Malaiya, ``Enhancing accuracy of Software Reliability Prediction,'' Proc. IEEE-CS Int. Symposium on Software Reliability Engineering, Nov. 1993, pp. 71-79.

References

- P.B. Lakey and A. M. Neufelder, *System and Software Reliability Assurance Notebook*, Rome Lab, FSC-RELI, 1997.
- L. Hatton, ``N-version Design Versus One Good Design'' IEEE Software, Nov./Dec. 1997, pp. 71-76.
- Tom McGibbon, "An Analysis of Two Formal methods VDM and Z, http//www.dacs.dtic.mil, Aug. 13, 1997.
- Robert Poston, "A Guided Tour of Software Testing Tools," Aonix, March 30, 1998.
- M.R. Lyu Ed., Software Reliability Engineering, McGraw-Hill, 1996.