

Statistical Techniques in Software Testing

Alessandro Di Bucchianico

2nd Dutch Workshop on Formal Testing Techniques

September 13, 2007

Outline

- Software Reliability Growth Models
- Certification of Software
- Walking Strategies in Software Models

Acknowledgements

This presentation is based on joint work in progress with:

- Kees van Hee (TU/e)
- Jan-Friso Groote (TU/e)
- Isaac Corro Ramos (TU/e)
- Lusine Hakobyan (TU/e)
- Ed Brandt (Refis)
- Rob Henzen (Refis)

Our research has been supported by grants from

- NWO (STRESS project)
- the Dutch Ministry of Economic Affairs (innovation voucher)

Uses of Statistics in Software Testing

We need to distinguish between:

- statistical methods to *analyze* data from software tests
 - quantify reliability growth (*e.g.*, certification)
 - prediction of testing process
- statistical methods to *generate* software tests (use cases) based on
 - operational profiles
 - formal software models

Software reliability growth models (SRGM)

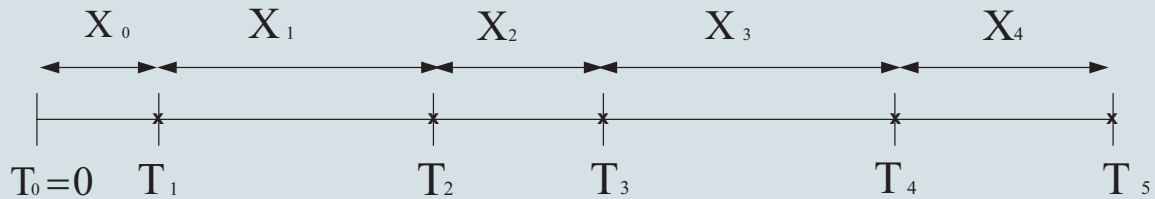
SRGM used to:

- resource planning
- release decisions
- certification
- ...

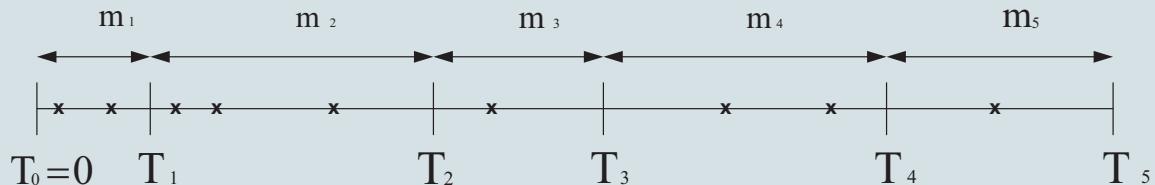
Typical statistical quantities of interest:

- probability of no failure in a given time period
- expected time until next failure
- number of errors left
- ...

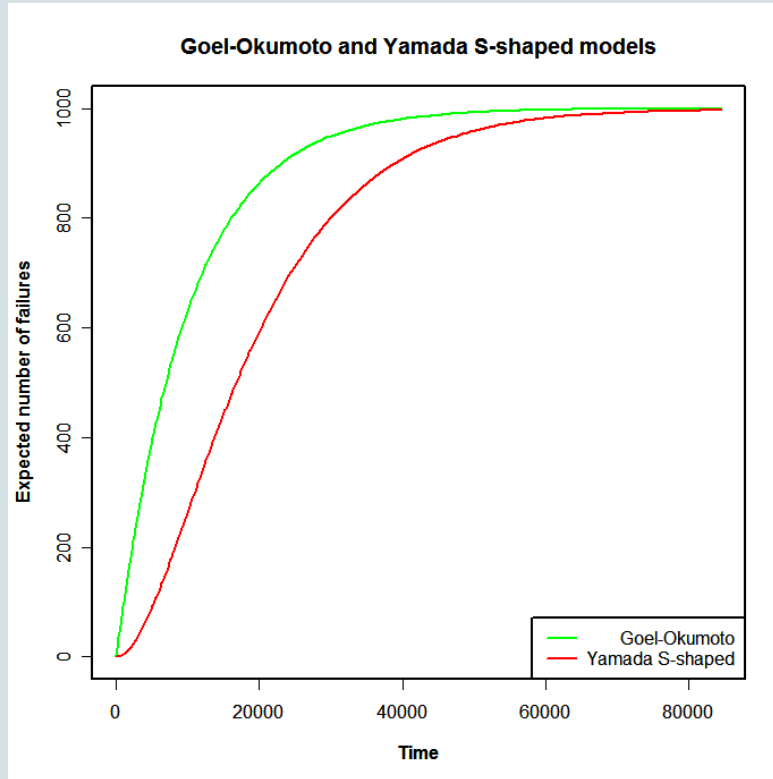
Ungrouped or exact data



Grouped or interval count data



Reliability growth data



Problems with SGRM

- insufficient documentation of the models
- more than 200 models known
- out-dated algorithms

What is needed:

- systematic description of SRGM (general methodology + algorithms)
- support pre-selection of models
- apply specific state-of-the-art algorithms for the models (convergence issues)
- adaptable tool to perform analyses

Systematic description of software reliability growth models

1. Probability model

- (a) Joint distribution of $\{N(t)\}_{t>0}$ and/or T_1, T_2, \dots, T_n and/or X_1, X_2, \dots, X_n (including (in)dependence structure)
- (b) Model assumptions
- (c) Interpretation of model parameters

2. Trend analysis

- (a) ...

Systematic description of software reliability growth models (Cont.)

3. Parameter estimation

- (a) Point estimation procedures for parameters (Maximum Likelihood and/or Least Squares)
 - i. Data requirements
 - ii. Existence results for parameter estimates
 - iii. Performance of parameter estimators (bias, efficiency)
 - iv. Algorithms to compute parameter estimates
- (b) Confidence interval procedures for parameters (Maximum Likelihood and/or Least Squares)
 - i. Distributional description of underlying estimators
 - ii. Algorithms to compute confidence intervals

SRGM description

- **GOS** models: T_1, T_2, \dots, T_n order statistics of a sample Z_1, Z_2, \dots, Z_N with c.d.f. F_θ .
 - Jelinski-Moranda or EOS: $X_i \sim \text{Exp}(\lambda(N - i + 1))$ i.i.d.
- **NHPP** models: $N(t_i) - N(t_{i-1}) \sim \text{Poisson}(\lambda(t))$ independent with mean

$$\mathbb{P}[N(t_i) - N(t_{i-1}) = k] = e^{-(\Lambda(t_i) - \Lambda(t_{i-1}))} \frac{(\Lambda(t_i) - \Lambda(t_{i-1}))^k}{k!}$$

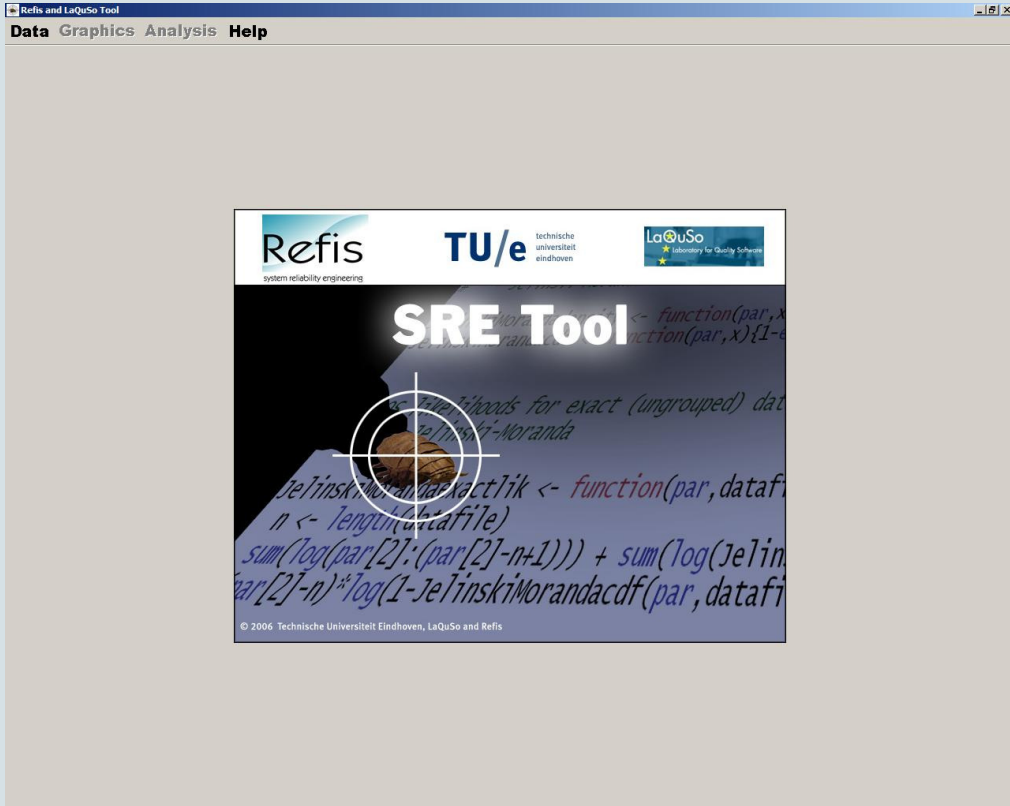
- Goel-Okumoto: $\Lambda(t) = a(1 - e^{-bt})$
- Yamada S-shaped: $\Lambda(t) = a(1 - (1 + bt)e^{-bt})$
- Duane: $\Lambda(t) = \left(\frac{t}{\alpha}\right)^\beta$

Model Pre-selection

	Relative Importance	Geometric	Jelinski-Moranda	Littlewood-Verrall	Musa basic	Musa-Okumoto	Goel-Okumoto	Shick-Wolverton	Schneidewind	Yamada S-shaped	Duane
Data Requirements and Assumptions											
Data may be exact failure times (ungrouped data)	2	x	x	x	x	x	x	x	x	x	x
Data may be grouped failure times (interval count data)	2	x	x	x	x	x	x	x	x	x	x
Testing intervals may be of different length	3	x	x	x	x	x	x			x	x
Failures need not occur equally likely	2	x		x		x	x		x	x	x
Detection of faults may be dependent of each other	2			x	x	x	x		x	x	x
Failures need not be of the same severity	1			x	x	x	x		x	x	x
Detection rate depends on time (testing effort)	3			x	x	x	x		x	x	x
Detection rate depends on number of remaining defects	3	x	x					x			
Failures need not be repaired instantaneously	3		x		x		x	x	x	x	x
Imperfect repair of defects allowed	2										
Infinite number of errors allowed	2					x					x

Systematic implementations: software reliability tool

- Existing packages do not make full use of state-of-the art statistical methodology
- Interface in Java (platform independent)
- Statistical computations in R (open-source free software)
- Communication: JRI and JavaGD libraries (Computational Statistics group, University of Augsburg)
- Financially supported by a grant of the Dutch Innovation Platform



Certification of software

Traditional approaches to release procedures:

- estimation of number of remaining errors (usually unstable procedure)
- optimization of cost function:
 - costs of further testing
 - costs due to remaining errors in software (quantification of costs is difficult)

We wish to: *certify software by using a statistical test procedure that*

- *guarantees the software is error free with a certain probability*
- *makes no assumptions on the initial number of errors*

Basic set-up of model

- n errors (unknown, fixed)
- discrete time: number of test runs (test run is sequence of actions with predefined begin and end (use case))
- θ detection probability of one error during one test run, $\varphi = 1 - \theta$
- detection of errors independent during test runs
- test runs are independent
- perfect repair of errors
- one error only will be repaired after detection of errors
- only number of tests until detection of failures is recorded

Distribution of number of test runs

- n number of initial errors
- θ detection probability of one error during one test run, $\varphi = 1 - \theta$
- detection of errors independent during test runs

Probability of no error detection during test run after detection of $(i - 1)$ th error:

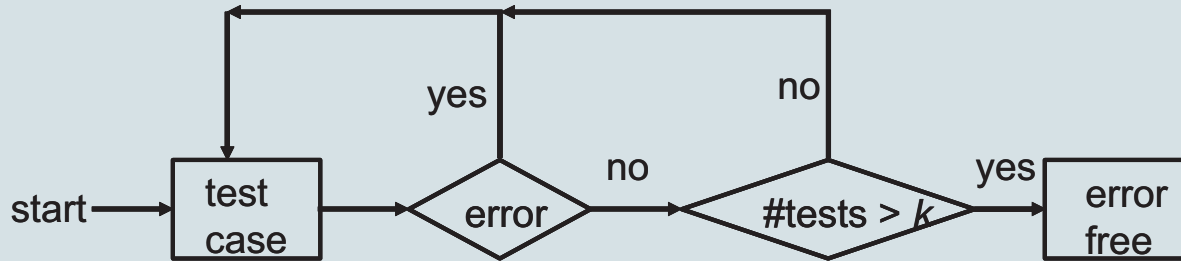
$$(1 - \theta)^{n-i+1} = \varphi^{n-i+1}$$

Distribution of number of test runs after detection of $(i - 1)$ th error and until detection of i th error (X_i):

$$P(X_i = m) = \varphi^{(m-1)(n-i+1)} (1 - \varphi^{n-i+1})$$

X_i has geometric distribution with success rate $1 - \varphi^{n-i+1}$.

Test procedure



Statistical setup :

- sequential testing of null hypothesis: there are undetected errors
- type I error = stop testing when there are undetected errors (with probability α)
- **no** type II error: continue testing when there are no undetected errors

Observations $T_i = \min(X_i, k + 1)$; Stopping time: $I = \min\{i \mid T_i > k\}$

Must choose k such that $P_{\theta,n}(I \leq n) \leq \alpha$ for given α and *all* n .

Scenarios for test procedure

- θ known
- prior distribution for θ (learning from previous tests), no update during testing
- one-stage testing with prior distribution (update of detection probability at end)
- adaptive testing (continuous updating of detection probability)

Main Result: strategies for one-stage testing work reasonably well for adaptive testing.

White Box Testing

- Defining a test procedure that uses knowledge of the structure of the software system and the prior test history
 - By having an accurate description of how the system behaves we achieve exhaustive testing
- Statistical release procedure based on the test history
 - Defining stopping criterion based on the probability of certain number of remaining errors
- It is not conformance or model based testing
- It is not statistical testing (generate test cases using probability distributions)

Assumptions

- Software systems running one sequential process only
- Modeled as a workflow transition system (WTS)
- Errors are only in the transitions
 - Symbolic markings
 - Found when error marked transitions are executed
- Run: path from the initial state to either the final state or an error marked transition

Workflow transition system

$W = (S, T, R)$, is a labelled transition system with special states i and f called initial and final states, respectively

- S non empty finite set of states
- T non empty finite set of transitions
- $R \subseteq S \times T \times S$

Error is a function $M : T \rightarrow \{0, 1\}$

$$M(t) = \begin{cases} 1, & \text{if } t \text{ is error marked} \\ 0, & \text{if } t \text{ is error free} \end{cases}$$

Run is a path $(i_W, t_1, \dots, s_n, t_n, s_{n+1})$. If $s_{n+1} = f_W$, then the run is said to be *successful*. If $s_{n+1} \neq f_W$, then the run is said to be *failure*.

Modelling framework for testing

Walking strategy and update: Special test procedure with probability distribution on branching points. Update of the walking strategy is assigning probability zero to some already executed the transitions

- **Purpose:** After each successful run increase the probability of visiting new transition.
- **How:** By discarding already some visited parts.

Walking Strategy (case of SMWf nets)

1. **First approach: random walk.** Being in place p_i , equal probability of visiting its neighbours.
2. **Second approach: optimal.** The probabilities to visit each transition, starting from the initial place i , are as "equal as possible".
3. **Third approach: adaptive.** The probabilities to visit each transition are adapted after each run (like sampling without replacement: we discard previously visited parts).

Test Model

- $G = (P, T, F)$
 - where P = set of places, T = set of transitions.
 - $F \subseteq P \times T \times P$
 - $P[G = g]$ is known.
- $D \subseteq T$, the *defect* transitions where $P[D = d \mid G = g]$ is known in our test environment .
- X_1, X_2, X_3, \dots outcomes of the test-runs
 - can be just Success of Failure,
 - can be the sequence of visited transitions,
 - or anything in between.

- X_1, X_2, X_3, \dots outcomes of the test-runs, $X_0 = \emptyset$.
- $G_n = (P, T, F, w_n)$
 - $w_n : T \rightarrow [0, 1]$ probabilities for transitions, w_0 determined by algorithm.
 - $w_{n+1} = W(G, w_n, X_n)$
 - W is determined by the walking strategy.
 - G_n is a Markov chain.
 - $P[X_n = x \mid G_n = g]$ is known
- D_{n+1} is equal to
 - D_n , if X_n is a successful run, or
 - $D_n \setminus \{T_n\}$, where T_n is the last transition of X_n if it is a failure run.

- X_1, X_2, X_3, \dots outcomes of the test-runs.
- G_1, G_2, G_3, \dots Markov chains for the test-runs.
- S is a stopping rule for X_1, X_2, X_3, \dots
 - may be dependent of the structure of G .
- Quality measures:
 - $E[(D - D_S) / D]$ expected percentage of errors left.
 - $P[D_S \neq D]$ error of the first kind.

Statistical release procedure

Error marking process: select without replacement a transition and labeling it as an error with unknown probability θ .

1. $W = (S, T, R)$

2. $T = \{t_1, \dots, t_N\}$

3. $\varepsilon_i \sim \text{Ber}(\theta), i = 1, \dots, N$, i.i.d.

4. $M(t_i) = e_i$ for all $t_i \in T$, where M is the error marking function and e_i the realizations of ε_i

5. t_i is error marked with probability θ

6. $D = \sum_{i=1}^N \varepsilon_i \sim \text{Bin}(N, \theta)$ unknown number of error marked transitions

Bayesian release procedure

1. The error probability is a random variable Θ with prior distribution function $F_{\Theta}(\theta)$
2. Probability of having at most k remaining errors when we decide to stop testing is greater than or equal to $1 - \alpha$ (α small, normally 0.05 or 0.01)
3. $s = \sum_{j=1}^n x_j$ the number of error marked transitions in the sample of visited transitions
4. Calculate the minimal value of n such that

$$\mathbb{P}[s \leq D \leq s + k | \mathbf{X} = \mathbf{x}] = \sum_{d=s}^{s+k} \mathbb{P}[D = d | \mathbf{X} = \mathbf{x}] \geq 1 - \alpha \quad (\text{I})$$