Some Stochastic Models of Software Evolution

Dr. Richard J. Botting
Computer Science
CSUSB
San Bernardino, CA 92407, USA
http://www.csci.csusb.edu/dick

The support of the CSUSB and the National Science Foundation under award 9810708 is gratefully acknowledged

Outline of Paper

- ★Previous stochastic models
- ★General Form
- ★Special Cases
- ★Conclusions

CMM Level 1 Processes

CMM levels: 1:Chaotic, 2:Repeatable, 3:Defined,
4:Managed, 5:Optimizing

- ★ Level 1 processes are random.
  - ★ We can use *stochastic processes* to model them.
- ★ One stochastic process fits the "Hacker Ethic".
  - ★ It is a well known stochastic process.
  - ★ We can predict the consequences.
  - ★ They are not good.
- ★ Other processes ameliorate the problems.

[*CMU SEI*]

Earlier Stochastic Models of Software Development

(Reliability Theory)

- Poisson processes:
  - Like ants in my kitchen, bugs pop up and disappear at random.
  - No model of software structure.
  - No model of diagnosing the defect from the bug.

[Sophisticated example: Carol Smidts 1999: A Stochastic Model of Human Errors in Software Development: Impact of Repair Times. *IEEE Proceedings of the 10th International Symposium on Software Reliability Engineering*]

Recent Stochastic Model

Jayant Rajgopal & Mainak Mazumbar

- Given modules, transition probabilities between modules, and the probabilities of a module failing they calculate reliability (prob. of failure).
- They model internal behavior of software.
- I will be modeling software evolution.

[Modular Operational Test Plans for Inferences on Software Reliability Based on a Markov Model *IEEE Trans Software Engineering* V28n4(Apr 2002)pp358-363]

Debugging as a Process

Definition of Debugging

- ★The programmer works alone.
- ★No documentation.
- ★Many cycles of:
  - ★Run a test on the whole program.
  - ★Change one piece of the code.
- ★Different stopping rules.

An Immature Modular Software Process

- ★ Software has many parts.
  - ★ Some parts are defective, some are good.
  - ★ It is not easy to tell good from defective.
- ★ Programmers are human and so error prone.
  - ★ Immaturity is modeled by randomness
- ★ A string of lights will light only if you can find and replace the broken one.
  - ★ It may be easier to replace the whole string!

Parameters

- Software
  - n is the number of pieces or parts.
  - R is the number of defective pieces out of n
    - R is a random variable with values 0,1,2,..., n.
    - When $R=0$, the software will pass its test.
  - $D = R/n$, the *defect density*.
- Process
  - α : [0..1] is the probability that a piece of code is defect free after a change.
  - $\beta = 1- \alpha$, is the probability that a piece of code has a defect after a change.
  - t: 0,1,2,..., is the number of change+test cycles.

Markov Chain

$p_r(t)$ = Pr[$R = r$ after t cycles of debugging]

- $p(t)$: row vector of $n+1$ probabilities
  - $p(t) = (p_0(t), p_1(t), p_2(t), ..., p_n(t))$
- $p(t+1) = p(t)\,P$
  - where P is a particular $(n+1) \times (n+1)$ matrix
- $p(t) = p(0)\,P^t$

Finite state, time invariant Markov Chain [Bhat Chapter 3]

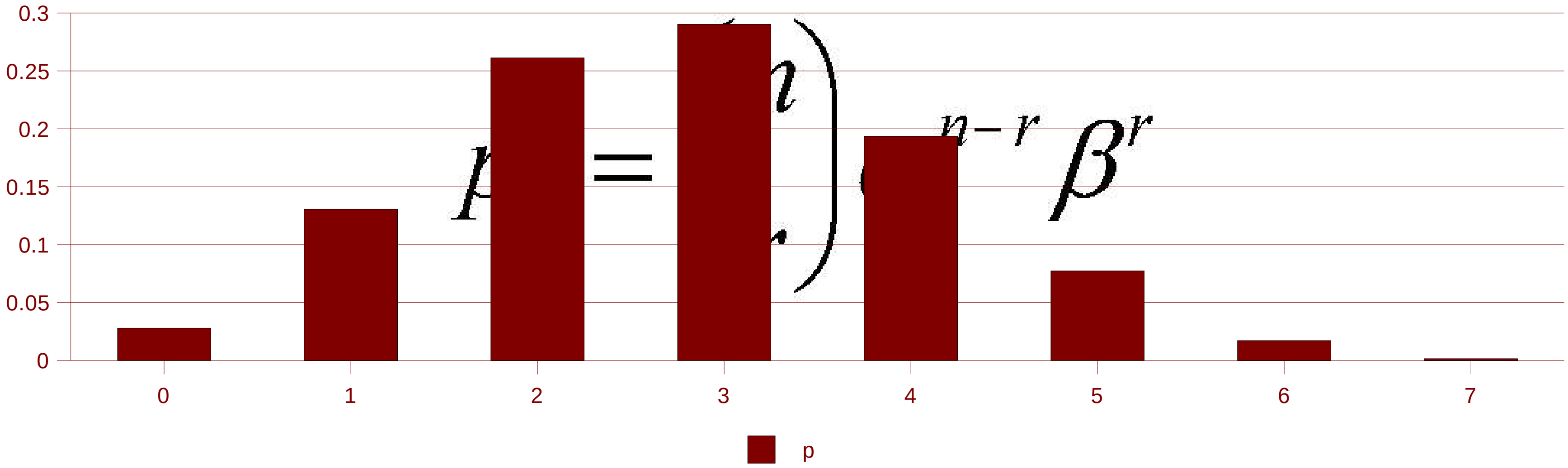
The Matrix P

(Hacking)

$$P = \begin{pmatrix} n\alpha & n\beta & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \alpha & (n-1)\alpha+\beta & (n-1)\beta & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 2\alpha & (n-2)\alpha+2\beta & (n-2)\beta & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 3\alpha & (n-3)\alpha+3\beta & (n-3)\beta & 0 & \cdots & 0 & 0 \\ 0 & & 0 & 4\alpha & (n-4)\alpha+4\beta & (n-4)\beta & \cdots & 0 & 0 \\ 0 & & 0 & 0 & 5\alpha & (n-5)\alpha+5\beta & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & & 0 & 0 & 0 & 0 & \cdots & (\alpha+(n-1)\beta) & \beta \\ 0 & & 0 & 0 & 0 & 0 & \cdots & n\alpha & n\beta \end{pmatrix} / n$$

The Limit $t \to \infty$

Text book result: tends to the Binomial Distribution



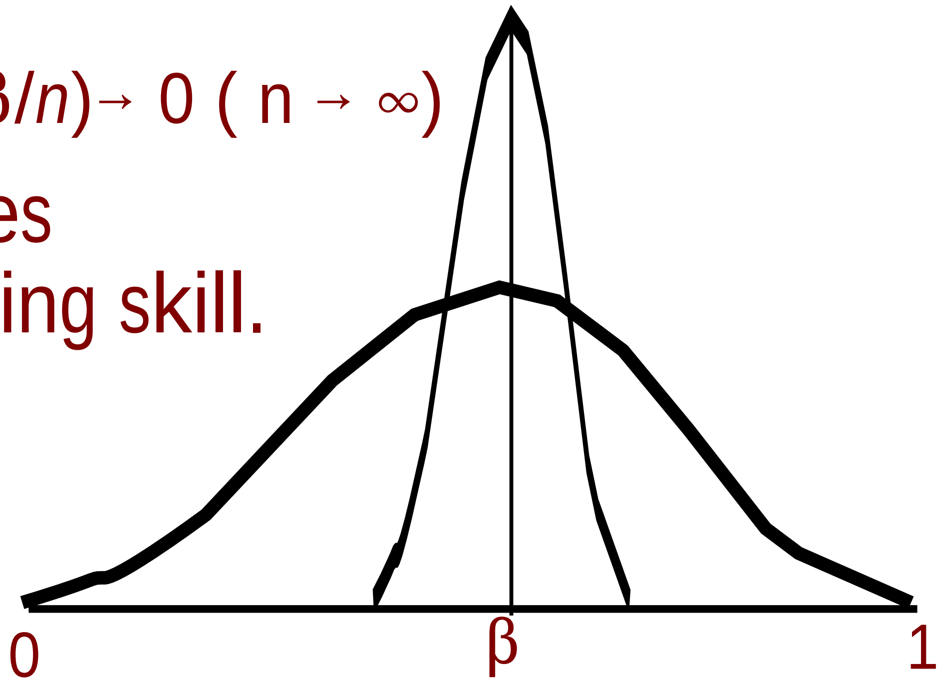
[*Bhatt 84*, *Ross 96*]

Consequences

- Number defective parts R
  - Mean $= n\ \beta$, variance $= n\ \alpha\beta$

- Defect Density D
  - Mean $= n\ \beta / n = \beta$.
  - Standard dev'n $= \sqrt{(\alpha\beta/n)} \rightarrow 0\ (n \rightarrow \infty)$

- Defect density comes to depend only on fixing skill.



The Perfect Hacker

A Special Case: $\beta = 0$

- If there are *R* defective pieces initially,
- it takes $n\ H_R$ cycles on average to fix all *R* parts.
  - H_R is the Harmonic number $1/1 + 1/2 + 1/3 + \dots 1/R$
- Cycles be fore fixing = $O(\ n \log(n)\ )$

[*Knuth 69*]

Debug & Release

Fixed point R = 0

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \alpha & (n-1)\alpha+\beta & (n-1)\beta & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 2\alpha & (n-2)\alpha+2\beta & (n-2)\beta & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 3\alpha & (n-3)\alpha+3\beta & (n-3)\beta & 0 & \cdots & 0 & 0 \\ 0 & & 0 & 4\alpha & (n-4)\alpha+4\beta & (n-4)\beta & \cdots & 0 & 0 \\ 0 & & 0 & 0 & 5\alpha & (n-5)\alpha+5\beta & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & & 0 & 0 & 0 & 0 & \cdots & (\alpha+(n-1)\beta) & \beta \\ 0 & & 0 & 0 & 0 & 0 & \cdots & n\alpha & n\beta \end{pmatrix} / n$$

Evolving Software & Requirements Creep

When R = 0, then R' = 1

$$
P=\begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\
\alpha & (n-1)\alpha+\beta & (n-1)\beta & 0 & 0 & 0 & \cdots & 0 & 0 \\
0 & 2\alpha & (n-2)\alpha+2\beta & (n-2)\beta & 0 & 0 & \cdots & 0 & 0 \\
0 & 0 & 3\alpha & (n-3)\alpha+3\beta & (n-3)\beta & 0 & \cdots & 0 & 0 \\
0 & & 0 & 4\alpha & (n-4)\alpha+4\beta & (n-4)\beta & \cdots & 0 & 0 \\
0 & & 0 & 0 & 5\alpha & (n-5)\alpha+5\beta & \cdots & 0 & 0 \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
0 & & 0 & 0 & 0 & 0 & \cdots & (\alpha+(n-1)\beta) & \beta \\
0 & & 0 & 0 & 0 & 0 & \cdots & n\alpha & n\beta
\end{pmatrix} / n
$$

Time to Re-Release

μ = mean time to rerelease

$n = 10$

- $\beta = 0.0$, $\mu = 10$
- $\beta = 0.01$, $\mu = 11$
- $\beta = 0.1$, $\mu = 19$
-
-
- $\beta = 0.7$, $\mu = 241{,}928$

As β increases the time before the software is released rapidly becomes very large.

Five Other Processes

- ★ Classic Software Engineering
- ★ Clean Room
- ★ Extreme Programming
- ★ Open Source
- ★ Biological Evolution

Classic Software Engineering

- ★All Requirements known at the start.
- ★Requirements mapped into design.
- ★Design mapped into code
- ★Documentation! $n \rightarrow 1$, Birth-Death models
- ★Co$t of documentation

Clean Room

No debugging

- ★ Inspections remove defects before coding.
- ★Tests estimate mean time to failure in use.
- ★Testing debugs the process, not the code.
- ★The Process evolves slowly.
- ★The Code has few if any bugs.

[Linger & Trammell in "*Industrial-Strength Formal Methods in Practice*", Hinchey & Bowen, Springer 1999]

Extreme Programming (XP)

Not Level 1 !

- ★ XP reduces *n* by using unit tests and merciless refactoring.
- ★ XP improves β by pair programming.
- ★ XP is limited to projects where face to face meetings are possible.

[*IEEE Software Mag Dec 2001 Paulk 01*]

Open Source

The Bazaar

- ★Parallel debugging.
- ★No bug is deep if enough eyeballs look for it.
- ★Closer to biological evolution.
- ★Users must also be programmers

[*Raymond01*]

Biological Evolution

- ★Genes effect many offspring.
- ★All offspring tested in parallel.
- ★Death terminates testing.
- ★Mixes best-of-breed genes.
- ★Performs much better than one blind hacker.

- ★But
  - ★Genetic Engineering fits the hacker model.

Conclusions

- ★One Size Does Not Fit All.
- ★Keep your *n* small and your α high.
- ★A million monkeys will beat a blind watch maker!