Some Practical Approaches for Software Reliability Analysis from A Systems Engineering Perspective

Min Xie (谢旻)
Chair Professor, Fellow of IEEE
Associate Dean, College of Science and Engineering
City University of Hong Kong
Min Xie, Chair Prof of Industrial Engineering

- Research in Reliability and Quality Engineering
- **William Mong Visiting Fellow** to Univ of Hong Kong (1996)
- **Invited professor** at UNPG, Grenoble, France (2000)
- Recipient of **Lee Kuan Yew research fellowship** in Singapore (1991)
- **Fellow of IEEE** (2006)
- Supervisor of about 40 PhD students
- Published over 250 SCI Journal papers and 8 books,
- On editorial team in about 20 international journals.
- Organizer, chairman and keynote speaker at numerous conferences.
City University of Hong Kong aspires to become a leading global university, excelling in research and professional education. We have performed strongly and consistently in global ranking exercises.

CityU is ranked #1 in Hong Kong in terms of citations per faculty.

Asian university rankings:
- 9th
- Top 50 under 50
- 5th

World university rankings:
- 124th 2009
- 129th 2010
- 110th 2014
- 57th 2015

According to Quacquarelli Symonds (QS).
INTRODUCTION TO SOFTWARE RELIABILITY

PROBLEMS, MODELS AND ANALYSIS
Software reliability

• In the seventies to eighties, hardware reliability analysis has become a standard.
• What to do with software?
• Traditionally, software is assumed to be perfect,
• But it is not!

*Software is an integral part of modern systems and no scientific theory is available to describe its functions. It is wrong to take it to be deterministic.*
My first problem as a student

- A device is immediately replaced by a new one after failure;
- The lifetimes of each device can be assumed to be independent and identically distributed;
- We want to have an estimate of the expected number of replacement within time $(0,t)$.
- Problem from automobile company (test of new car) and estimation of warranty cost (free replacement)
Renewal Process and Renewal Function

- The expected number of events in a renewal process is called the renewal function as it is a function of time \( t \), \( M(t) = E[N(t)] \).
- \( M(t) \) can be determined by

\[
M(t) = F(t) + \int_0^t M(t - x) f(x) \, dx
\]
A New Problem

\[ M(t) = F(t) + \int_{0}^{t} M(t - x) f(x) \, dx \]

- Slow convergence when numerical algorithms or subroutines are used;
- Possibly because of the singularity when \( f(x) \) is infinity for some \( x \);
- This is common for example when Weibull distribution is used, or when only the empirical distribution is available.
A New (numerical) Method

\[ M(t) = F(t) + \int_0^t M(t-x) dF(x) \]

- The discretization is based on the definition of the Riemann-Stieltjes Integral:

\[ \int_a^b f(x) dg(x) = \sum_{i=1}^n f(x_{i-1/2}) [g(x_i) - g(x_{i-1})] \]
A New (numerical) Method

\[ M(t) = F(t) + \int_0^t M(t-x) f(x) \, dx \]

\[ = F(t) + \int_0^t M(t-x) \, dF(x) \]

- Rewrite the equation;
- Discretize the integral;
- Obtain a set of linear equations;
- Solve the linear equations.
Comparative Studies

• The method is surprisingly
  – Accurate (no need for small step-length)
  – Fast (10-fold time saving)
  – Simple (20-line BASIC programme)

• Reference:
function [X]=c3_mtf(F,g,t)
    
    Find the renewal function given cdf F
    Ref: Xie, M. "On the Solution of Renewal-Type Integral Equations"
    
    [n,m]=size(F); g0=g(1); g(1)=[]; M=F;dno=1-g0;
    M(1)=F(1)/dno;
    for i=2:m
        sum=F(i)-g0*M(i-1);
        for j=1:i-1
            if j==1
                sum=sum+g(i-j)*M(j);
            else
                sum=sum+g(i-j)*(M(j)-M(j-1));
            end
        end
        M(i)=sum/dno;
    end
    
    Output the results
    
    d=20;
    x=[0]; Mt=[0];
    for i=d:d:m
Learning Experience

• Typical application of theoretical models
• Unexpected problems need to be solved (different types of knowledge might be needed)
• Software packages, subroutines, and algorithms are not reliable
• A “pity” that I did not move into software development;
• Got interested in “software reliability”…
Other application areas

• Healthcare systems and medical equipment
• Traffic control systems
• Cloud computing
• …
Difficulties in Software Reliability Analysis

- Software failures can be tracked to individual mistake
- Although in theory we can make it correct, in reality it is impossible
- Testing is costly
- Testing cannot prove the correctness
- There are many testing techniques with varying degree of efficiency
- Difficult to improve reliability
Randomness of Failures

- Number of failures per unit time is random
- Time to next failure is random
- This is because
  - the location of faults in the programme is unknown
  - the usage of programme is not predictable

Correctness of the results

UNKNOWN

INPUT
Reliability of Combined System

- Assuming both are needed for the system to work
- Failure of one should not affect the other
- The failure causes should be able to be isolated
- Software may not be more reliable than hardware
- Important to consider serious failures
Markov Process Models

- Jelinski-Moranda
- Earliest model
- Equal contribution of all faults
- Finite number of possible failures
- Debugging assumed to be perfect
Related references


m.xie@cityu.edu.hk
Reliability Growth Pattern

- Failure cause is identified after a failure
- Action is taken to remove the cause
- Same type of failure will not occur
- Time to next failure is likely to be longer
The Goel-Okumoto Model

• Probably the most well-known SRM
• Derived assuming the same detection rate of remaining faults
• Simple model for finite number of faults

\( m(t) = a(1 - e^{-bt}), \quad a > 0, \quad b > 0; \)

\( \lambda(t) = \frac{dm(t)}{dt} = abe^{-bt}. \)
Extensions

- Imperfect debugging
- Learning effect
- Incorporating metrics information
- Failure correlation
- System modeling and analysis
- Cost analysis and decision making
Release Time Determination - cost minimization

- Time to minimize total cost
  - need a cost model

$$c(T) = c_1 m(T) + c_2 [m(\infty) - m(T)] + c_3 T.$$  

- $c_1 = \text{expected cost of removing a fault in testing}$
- $c_2 = \text{expected cost of removing a fault in field}$
- $c_3 = \text{expected cost per unit time of software testing including the cost of testing, the cost due to a delay in releasing the software, etc.}$
Some related research issues

• Relate reliability parameters to system and process metrics for better prediction
• Use of information from earlier version for earlier prediction
• Considering fault-detection and fault-correction process
• Study of open source software
Numerical Example

<table>
<thead>
<tr>
<th>Week $i$</th>
<th>$d_i$</th>
<th>$c_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>23</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>43</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>32</td>
</tr>
<tr>
<td>5</td>
<td>84</td>
<td>53</td>
</tr>
<tr>
<td>6</td>
<td>97</td>
<td>78</td>
</tr>
<tr>
<td>7</td>
<td>109</td>
<td>89</td>
</tr>
<tr>
<td>8</td>
<td>111</td>
<td>98</td>
</tr>
<tr>
<td>9</td>
<td>112</td>
<td>107</td>
</tr>
<tr>
<td>10</td>
<td>114</td>
<td>109</td>
</tr>
<tr>
<td>11</td>
<td>116</td>
<td>113</td>
</tr>
<tr>
<td>12</td>
<td>123</td>
<td>120</td>
</tr>
<tr>
<td>13</td>
<td>126</td>
<td>125</td>
</tr>
<tr>
<td>14</td>
<td>128</td>
<td>127</td>
</tr>
<tr>
<td>15</td>
<td>132</td>
<td>127</td>
</tr>
<tr>
<td>16</td>
<td>141</td>
<td>135</td>
</tr>
<tr>
<td>17</td>
<td>144</td>
<td>143</td>
</tr>
</tbody>
</table>

- Best fit: GO-model with constant $\Delta$
Fault Correction Process
Reliability of Combined System

- Assuming both are needed for the system to work
- Failure of one should not affect the other
- The failure causes should be able to be isolated
- Software may not be more reliable than hardware
- Important to consider serious failures

\[ R_{\text{system}} = R_{\text{hardware}} \cdot R_{\text{software}} \]
Reliability Growth Pattern

- Failure cause is identified after a failure
- Action is taken to remove the cause
- Same type of failure will not occur
- Time to next failure is likely to be longer
Resource allocation based on system failure numbers

<table>
<thead>
<tr>
<th></th>
<th>System A</th>
<th>System B</th>
<th>System C</th>
</tr>
</thead>
<tbody>
<tr>
<td>#failures</td>
<td>8</td>
<td>15</td>
<td>35</td>
</tr>
<tr>
<td>This year</td>
<td>5</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>1 yr ago</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>2 yrs ago</td>
<td>1</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3 yrs ago</td>
<td>0</td>
<td>4</td>
<td>20</td>
</tr>
</tbody>
</table>

How do we allocate the resources?

How do we allocate the resources now?
Questions? Contact me:
minxie@cityu.edu.hk

M Xie, Fellow of IEEE
Chair Professor of Industrial Engineering
Dept of Systems Engineering and Engineering Management
City University of Hong Kong