Deciding how long to test software: decision theoretic approach

Simon Wilson    Seán Ó Ríordáin

Trinity College Dublin

With thanks to Kevin McDaid, Dundalk Institute of Technology
Questions addressed in this talk

- Research Question
- Research Method
- Summary of research to date
- Plan for next 12 months
Research Question

First draft: “From a statistical point of view, is it safe to release this software yet? If not how much longer should we test it?”

A key component of this decision process is the uncertainty surrounding the number of bugs and other factors such as severity and the utility parameters that will influence the outcome of any decision. This work aims to model this uncertainty looking the data and the relative importance of expert opinion.
Research Method

Statistical Learning and Normative Decision Theory

The key difference between the **Statistical** Approach and the **Empirical** Approach is that in the former we try to understand and model the underlying process to better understand what is actually happening.
Summary of Research to date
Issues addressed in this talk

- Some exploration of the software testing problem and its solution with decision theory;
- The (simple) Goel-Okumoto model of bug discovery;
- Work done so far, following McDaid and Wilson (2001);
- A simple utility function for testing;
- Different testing plans
- Future work
Typical Software Development Stages

- **Development Phase:**
  - Piece of software created by development group;

- **Testing Phase:**
  - Testing group verifies that software satisfies design specifications;
  - Software is tested in order to find errors (or “bugs”) that cause the software “to fail”;
  - Bugs are repaired.

- **User Phase:**
  - Program is released for general use.

**Software Testing Problem:** when to stop the testing phase and release the software to the user?
The Software Testing Problem

**Decision:** Find optimal time to test before release to user

- If testing stops too early then many bugs remain:
  - large cost of fixing bugs after release;
  - losses due to customer dissatisfaction.
- If testing continues for too long then:
  - large cost of testing;
  - loss of market initiative.
- Releases themselves can cost a lot of money for embedded systems, e.g. car security systems.
- Optimal testing time balances these factors;
- Decision made *before* testing begins, when there is *uncertainty* about actual performance of software.
Utility for testing

<table>
<thead>
<tr>
<th>Cost Matrix</th>
<th>No Bugs</th>
<th>Bug Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release</td>
<td>Profit!</td>
<td>Oh dear!</td>
</tr>
<tr>
<td>Test more</td>
<td>Waste!</td>
<td>Phew!</td>
</tr>
</tbody>
</table>
Utility for testing

- Test for time $T$, discovering $N(T)$ bugs, then release software to user, where $\tilde{N}(T)$ bugs are discovered:

$$U(T, N(T), \tilde{N}(T)) = P - C_1 N(T) - C_2 \tilde{N}(T) - F(T),$$

where:

- $P =$ profit of releasing perfect software immediately;
- $C_1 =$ cost of repairing a bug discovered during testing;
- $C_2 =$ cost of repairing a bug discovered after testing;
- $F(T) =$ opportunity cost; we have $F(T) = fT^g$;
- If $g = 1$ then $U$ is the profit in euro;
- Can also use utility of money e.g. $U(x \text{ euro}) = 1 - e^{-rx}$. 
Elicitation

- Most utility parameters are profits and costs;
- The usual elicitation methods can be used;
- Sensitivity analysis is important;
- Here we use: $P = 500, C_1 = 1, C_2 = 10, f = 0.5, g = 1$;
Goel-Okumoto Model for $N(T)$

- $N(T)$ is a non-homogeneous Poisson process with mean value function $\Lambda(t) = a(1 - e^{-bt})$;
- Motivation:
  $\Lambda(T + \delta T) - \Lambda(T) \approx b(a - \Lambda(T)) \delta T + o(\delta T)$;
- $\mathbb{E}(N(\infty)) = a$;
- $b = \text{rate of discovery}$;
Distribution of $N(T)$

- We have:

\[ P(N(T) = n \mid a, b) = \frac{\Lambda(T)^n}{n!} e^{-\Lambda(T)} \]

\[ = \frac{a^n(1 - e^{-bt})^n}{n!} e^{-a(1-e^{-bt})}. \]

- Distribution of $\bar{N}(T)$ can also be found.
Elicitation

- Given data on bug discover times $t_1, t_t, \ldots$;
- We use Bayesian inference methods to fit $a, b$;
- Allows us to forecast distribution of future numbers of failures;
NTDS Data

- Data on 31 bugs discovered in a system for the US Navy;
- Our values fit the data quite well;
If we know failure times of NTDS data, this is the utility (not expected utility);
Optimal time to test is 116 (utility of 281)
Of course when we decide these data are not known!
Firefox and Agile Processes

Mozilla Firefox

Number of Bugs

version
1.0
1.5.0.x
2.0
3.0
3.5
3.6
4.0
5
6
7
8
9
10
11
12
13
14
15
16
Trunk
unspecified
Firefox and Agile Processes - 3
Firefox and Agile Processes - 5
Firefox Extended Support Release - 2

Firefox Release 10 Extended Support Release (ESR) – pre-release bugs

Days since Release vs Bugs since Release

- enhancement
- trivial
- normal
- minor
- major
- critical

Graph showing the relationship between Days since Release and Bugs since Release for Firefox Release 10 Extended Support Release (ESR) – pre-release bugs.
Firefox Extended Support Release - 3

Firefox Release 10 Extended Support Release (ESR) – all bugs

Days since Release

Bugs since Release

bug_severity
- enhancement
- trivial
- normal
- minor
- major
- critical
- blocker
Plan for next 12 months

- Refine Research Question and prepare Transfer Report
- Gather data from Safety Critical Systems
- Consider alternatives to the Goel-Okumoto model
Future Work

- Need to better understand the data!
- Need data, especially from safety critical systems! PEL have a close working relationship with the testing group in IBM and that you expect/hope to get some data from them.
- Incorporate other software metrics; e.g. Bug Severity; Bug Reporter, Lines of Code, Number of Developers, etc.
- Study how to better incorporate expert knowledge.
- Move away from Goel-Okumoto model
- Modeling of multiple version releases of one system in Agile / short fixed release cycles.
- Comparison of systems with short fixed releases versus more flexible release cycles, e.g. Firefox (fixed 46 days) versus Chromium (flexible release dates)
- How should the results of the model be reported?
Acknowledgements

This work was supported, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero - the Irish Software Engineering Research Centre (www.lero.ie), under the Robust Test Environments work-package led by Professor John Murphy of UCD.

http://lero.ie/project/rte
One stage testing

- Simplest test;
- Test for a time $T^*$ and then release software;
- Only prior information used to decide $T^*$;
- Expected utility for testing to time $T$ is:

$$\mathbb{E}(U(T, N(T), \tilde{N}(T)))$$

$$= \sum_{N(T)=0}^{\infty} \sum_{\tilde{N}(T)=0}^{\infty} U(T, N(T), \tilde{N}(T)) P(N(T), \tilde{N}(T))$$

$$= P - fT^g - C_1 \mathbb{E}(N(T)) - C_2 \mathbb{E}(\tilde{N}(T))$$

$$= P - fT^g - \frac{\beta_a}{\alpha_a} \left[ C_1 + (C_2 - C_1) \left( \frac{\alpha_b}{\alpha_b + T} \right)^{\beta_b} \right];$$
One stage testing

- So it's easy to find $T^* = \arg \max_T \mathbb{E}(U(T, N(T), \tilde{N}(T)))$.
- When $g = 1$ we have:

$$T^* = \max \left( 0, \alpha_b \left[ \left( \frac{\beta_a \beta_b (C_2 - C_1)}{\alpha_a \alpha_b f} \right)^{1/(\beta_b+1)} - 1 \right] \right)$$

- Note: purely based on prior (expert) opinion - no data available at time when decision is made;
Decision Tree for One Stage Testing

Decide Test Time $T$  Release

$N(T)$ Bugs are Discovered in Testing  $\tilde{N}(T)$ Bugs are Discovered after Release

$U(T, N(T), \tilde{N}(T))$
Expected Utility for Elicited Parameters

\((T^* = 180)\)
Two Stage Testing

- Test for a time $T_1$;
- If $N(T_1) > N$ then test an additional time $T_2$;
- To decide: optimal $T_1$, $N$ and $T_2$;
- All decided at $T = 0$ (again, only using prior information).
Decision Tree for Two Stage Testing

\[ D \xrightarrow{T_1} i \text{ bugs found} \]

- Release if \( i \leq N \)
- Test until \( T_2 \) if \( i > N \)

\[ j \text{ bugs found} \rightarrow U(T_1, i, j) \]

\[ k \text{ bugs found} \rightarrow U(T_2, i + k, j) \]

- Release if \( j \) bugs found
- Release if \( k \) bugs found
Two Stage Testing

- For the same prior and utility parameters as before, the optimal test is: $T_1 = 116.5, N = 8, T_2 = 189$.
- For NTDS, 21 discovered by $t = 116.5$, hence test until 189 (another 2 discovered);
- Compare with testing for 180 with single stage (21 bugs discovered).
Sequential Testing

- Better idea: decide to test for another stage at the end of the previous stage;
- This decision can be made using data from testing to the end of the last stage;
- In theory, no limit to the number of stages!

\[
\text{Decide } T_1 \quad \rightarrow \quad \text{Test to } T_2 \quad \rightarrow \quad \text{Test to } T_3 \quad \rightarrow \quad \text{Release Utility}
\]

\[
N(T_1) \quad \rightarrow \quad N(T_2) \quad \rightarrow \quad \text{Release Utility}
\]
Sequential Testing

- Difficult to solve the tree;
- We can solve various approximations to the sequential test;
- We can also prove some results about the relationship between the approximations.
One Stage Look Ahead –
Markov Decision Process

• Consider that each stage is a separate one stage problem;
• After each stage, solve one stage problem again but using the *posterior distribution* of $N(T)$ and $\bar{N}(T)$ given data to the end of last stage;
One Stage Look Ahead

- For NTDS with our prior and utility parameters:
  - First stage is just the one stage test to 180 (21 bugs discovered);
  - Repeat one stage test using posterior for $N(T)$ and $\bar{N}(T)$ with data to 180 and optimal test time is now 218;
  - So test another 38 time units;
  - Repeat one stage test using posterior for $N(T)$ and $\bar{N}(T)$ with data to 218 and optimal test time is now < 218;
  - So release!!
One Fixed Time Look Ahead

- A priori, select a sequence of times at which you will decide to continue or stop testing e.g. each week;
- After $i$th stage, decide whether to test to end of stage $i + 1$ or release (a binary decision);
- For NTDS with our prior and utility parameters, and stopping every 50 time units:
  - Expected utility of testing to 50 is greater than to 0 so test to 50;
  - Given data to 50, expected utility of testing to 100 greater than testing to 50 so test to 100;
  - Given data to 100, expected utility of testing to 150 greater than testing to 100 so test to 150;
  - Given data to 150, expected utility of testing to 200 greater than testing to 150 so test to 200;
  - Given data to 200, expected utility of testing to 250 less than testing to 200 so release!!
One Bug Look Ahead

- Test until you find and repair a bug;
- Decide to release or test until you find another;
- Could test forever!
- For NTDS data and our prior and utility parameters, we test until the 24th bug discovered (at 247);

All 3 “one step ahead” plans are easy to implement for our model and utility.
Firefox and Agile Processes - 3

Defects found per day (5% trimmed top and bottom) vs. version