Predict Software Reliability
Before the Code is Written

Ann Marie Neufelder
SoftRel, LLC
amneufelder@softrel.com
http://www.softrel.com

© SoftRel, LLC 2013. This presentation may not be reprinted in whole or part without written permission from amneufelder@softrel.com
1962 First recorded system failure

1968 The term “software reliability” is invented.

1962 Many software reliability estimation models developed. 
**Main obstacle** – can’t be used until late in life cycle.

1968 First publicly available model to predict software reliability early in lifecycle developed by USAF Rome Air Development Center with SAIC and Research Triangle Park –
**Main obstacles** – model only useful for aircraft and model never updated after 1992.

2000’s A few proprietary models developed

SoftRel, LLC develops models based on RL model but usable on all defense/space
Software reliability modeling

- Software reliability can be predicted before the code is written, estimated during testing and calculated once the software is fielded.

- This presentation will discuss the prediction/assessment models.

<table>
<thead>
<tr>
<th>Prediction/Assessment</th>
<th>Reliability Growth Estimations</th>
<th>Field reliability calculations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used before code is written</td>
<td>Used during system level testing or operation</td>
<td>Used once software is operational</td>
</tr>
<tr>
<td>• Predictions can be incorporated into the system RBD</td>
<td>• Determines when to stop testing</td>
<td>Actual failure rate, MTBF, etc is measured directly from field data</td>
</tr>
<tr>
<td>• Supports planning</td>
<td>• Validates prediction</td>
<td></td>
</tr>
<tr>
<td>• Supports sensitivity analysis</td>
<td>• Less useful than prediction for planning and avoiding problematic releases</td>
<td></td>
</tr>
<tr>
<td>• A few models have been available since 1987</td>
<td>• Many models have been developed since 1970s such as the Musa Model.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The exponential model most commonly used.</td>
<td></td>
</tr>
</tbody>
</table>
Software reliability assessment goals and outputs

- Predict any of these reliability related metrics
  - Defect density (test and operation)
  - Defects (test and operation)
  - Mean Time To Failure (MTTF), reliability, availability at any point in testing or operation
  - Reliability ty growth in any of the above metrics over time
  - Mean Time To Software Restore (MTSWR)
  - Maintenance and testing staffing levels to reach an objective

- Use prediction to
  - Analyze sensitivity to make a specific growth in one or more metrics
  - Analyze sensitivity between software and hardware
  - Benchmark defect density to others in industry
  - Identify practices that aren’t effective for reducing defects
If you can predict this defect profile you can predict failure rate

- For decades the defect profile has been the basis for nearly all software reliability models[2]
  - During development you can predict the entire profile or parts of it
  - During testing you can extrapolate the remainder of the profile
Industry approach to early software reliability predictions

1. Predict effective size
2. Predict testing or fielded defect density
3. Predict testing or fielded defects
4. Identify defect profile over time
5. Predict failure rate/MTTF during test or operation
6. MTSWR and availability
7. Predict mission duration and reliability

Sensitivity Analysis
1. Predict size

If everything else is equal, more code means more defects

- For in house software
  - Predict effective size of new, modified and reused code using best available industry method

- For COTS software (assuming vendor can’t provide effective size estimates)
  - Determine installed application size in KB (only EXEs and DLLs)
  - Convert application size to KSLOC using industry conversion
  - Assess reuse effectiveness by using default multiplier of 1%
    - Accounts for fact that COTS has been fielded to multiple sites
Available Methods for predicting defect density

Ideally defect density prediction model optimizes simplicity, and accuracy and is updated on a regular basis.

<table>
<thead>
<tr>
<th>Method</th>
<th>Simplicity</th>
<th>Last updated on..</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predict defect density from historical data</td>
<td>Medium</td>
<td>N/A</td>
<td>Usually most accurate IF historical data is simple and recent</td>
</tr>
<tr>
<td>Predict defect density using an industry lookup chart or from SEI CMMi lookup chart</td>
<td>Easy</td>
<td>Varies</td>
<td>Usually the least accurate. Most useful for COTS software.</td>
</tr>
<tr>
<td>Predict defect density via surveys such as Shortcut, Full-scale, Rome Laboratory</td>
<td>Easy to Detailed</td>
<td>Softrel models are updated every 2 years Rome Labs model was last updated in 1992</td>
<td>If the survey is answered properly these are usually most accurate. RL model is geared only towards aircraft.</td>
</tr>
</tbody>
</table>
### Survey Based Defect Density Models

<table>
<thead>
<tr>
<th>Survey based model</th>
<th>Number of questions</th>
<th>Comments</th>
</tr>
</thead>
</table>
| Shortcut model     | 22                  | • More accurate than lookup charts  
|                    |                     | • Questions can be answered by almost anyone familiar with the project |
| Rome Laboratory    | 45-212              | • Some questions are outdated |
| Full-scale model A | 98                  | • More accurate than the shortcut model  
|                    |                     | • Questions require input from software leads, software testing, software designers |
| Full-scale model B | 200                 | • More accurate than the Full-scale model A  
|                    |                     | • Questions require input from software leads, software testing, software designers |
| Full-scale model C | 300                 | • More accurate than the Full-scale model B  
|                    |                     | • Questions require input from software leads, software testing, software designers  
|                    |                     | • 100 questions require expert review of development artifacts |
How the Shortcut or Full-scale Survey Models Works

1. Complete survey and calculate score

   Score
   - 1%: World class
   - 10%: Very good
   - 25%: Good
   - 50%: Average
   - 75%: Fair
   - 90%: Poor
   - 99%: Distressed

2. Find defect density and Probability (late) from associated row

   Predicted Percentile Group
   - 1%: World class
   - 10%: Very good
   - 25%: Good
   - 50%: Average
   - 75%: Fair
   - 90%: Poor
   - 99%: Distressed

3. When improving to next percentile
   - Average defect reduction = 55%
   - Average probability (late) reduction = 25%
Seven clusters used to predict defect density and ultimately software reliability

Percentile group predictions...
- Predicted directly from answering a survey and scoring it
- Pertain to a particular product version
- Can only change if or when risks or strengths change
- Some risks/strengths are temporary; others can’t be changed at all
- Can transition in the wrong direction on same product if
  - New risks/obstacles added
  - Opportunities are abandoned
- World class does not mean defect free. It simply means better than the defect density ranges in database.

More fielded defects:
- 99% Distressed
- 90% Poor
- 75% Fair
- 50% Average

More risks than strengths

Strengths and risks offset each other

More strengths than risks

Fewer fielded defects:
- 25% Good
- 10% Very good
- 1% World Class

0% Distressed

90% Poor

75% Fair

50% Average

25% Good

10% Very good

1% World Class

More fielded defects

Fewer fielded defects
3. Predict testing or fielded defects

Defects can be predicted as follows

- Testing defect density * Effective size = Defects predicted to be found during testing (Entire yellow area)
- Fielded defect density * Effective size = Defects predicted to be found in operation (Entire red area)
4. Identify shape of defect profile

An exponential formula is solved as an array to yield this area:

\[ \text{Defects (month } i) = N \left( \exp\left(-\frac{Q(i-1)/TF}{TF}\right) - \exp\left(-\frac{Q*i}{TF}\right) \right) \]

This width is growth period (time until no more residual defects occur) = TF = usually 3* average time between releases. Default = 48.

Growth rate (Q) derived from slope. Default = 4.5. Ranges from 3 to 10.

Calendar time

Default = 4.5. Ranges from 3 to 10.
Rate at which defects result in observed failures (growth rate)

Faster growth rate and shorter growth period – Example: Software is shipped to millions of end users at the same time and each of them uses the software differently.

By default, the growth rate will be in this range

Slower growth rate and longer growth period – Example: Software deliveries are staged such that the possible inputs/operational profile is constrained and predictable.
5. Use defect profile to predict failure rate/MTTF

- Dividing defect profile by duty cycle profile yields a prediction of failure rate as shown next.

- \( T_i \) = duty cycle for month \( i \) - how much the software is operated during some period of calendar time. Ex:
  - If software is operating 24/7 -> duty cycle is 730 hours per month
  - If software operates during normal working hours -> duty cycle is 176 hours per month

- \( MTTF_i = \frac{T_i}{Defectprofile_i} \)

- \( MTTCF_i \) = \( \frac{T_i}{\%severe * Defectprofile_i} \)

- \% severe = \% of all fielded defects that are predicted to impact availability
6. Predict MTSWR (Mean Time To Software Restore) and Availability

- Needed to predict availability
- For hardware, MTTR is used. For software, MTSWR is used.
- MTSWR = weighted average of time for applicable restore actions by the expected number of defects that are associated with each restore action
- Availability profile over growth period = Availability_i = \[
\frac{MTTCF_i}{MTTCF_i + MTSWR}
\]
- In the below example, MTSWR is a weighted average of the two rows

<table>
<thead>
<tr>
<th>Operational restore action</th>
<th>Average restore time</th>
<th>Percentage weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct the software</td>
<td>40 hours</td>
<td>.01</td>
</tr>
<tr>
<td>Restart or reboot</td>
<td>15 minutes</td>
<td>.99</td>
</tr>
</tbody>
</table>
7. Predict mission time and reliability

- Reliability profile over growth period =
  \[ R_i = \exp\left( -\frac{\text{mission time}}{\text{MTTCE}_i} \right) \]

- Mission time = how long the software will take to perform a specific operation or mission
  - Not to be confused with duty cycle or testing time
  - Example: A typical dishwasher cycle is 45 minutes. The software is not executing outside of this time, so reliability is computed for the 45 minute cycle.
Software prediction confidence bounds are a function of

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Contribution to prediction error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size prediction error due to scope change</td>
<td>Until code is complete, this will usually have the largest relative error</td>
</tr>
<tr>
<td>Size prediction error due to error in sizing estimate (scope unchanged)</td>
<td>Minimized with use of tools, historical data</td>
</tr>
<tr>
<td>Defect density prediction error</td>
<td>Minimized by validating model inputs</td>
</tr>
<tr>
<td>Growth rate error</td>
<td>Not usually a large source of error</td>
</tr>
</tbody>
</table>

Confidence Bounds and prediction error

MTTF

Months after delivery

© SoftRel, LLC 2013. This presentation may not be reprinted in whole or part without written permission from amneufelder@softrel.com
Predictions can be used for scheduling and maintenance

- Predictions can be used to determine how far apart releases should be to optimize warranty costs and response time
- This is an example from industry. The defects were predicted to pile up up after the third release.

![Graph showing total defects predicted from releases 1 to 5 predicted for each month]

© SoftRel, LLC 2013. This presentation may not be reprinted in whole or part without written permission from amneufelder@softrel.com
Sensitivity analysis and defect reduction

- SoftRel survey models and the Rome Laboratory model were developed for the purpose of supporting defect reduction scenario analysis.
- Use the models to find the gaps and determine sensitivity of each gap.
- Develop strategies for reducing the defects and rework the predictions based on a few key improvements.
Know which software characteristics/practices have biggest impact on software reliability

To date 600+ characteristics related to the 3 P’s have been mathematically correlated to software reliability by SoftRel, LLC[1]

- Product/industry/application type
- People
- Practices/process

Of these, 120 are so strongly related that they are used collectively to predict before the code is even written

[1]See the entire research and complete list of practices at “The Cold Hard Truth About Reliable Software”, A. Neufelder, SoftRel, LLC, 2014
Research results revealed some surprises

- Some practices, tools, metrics don’t always result in better software when...
  - Required prerequisites may not in place
  - Required training may not in place
  - Practices, tools or metrics used incorrectly
  - Software group not mature enough to implement practice, tool or metric
  - Metric provides results that aren’t useful

Examples

<table>
<thead>
<tr>
<th>Practice that’s not always related to lower defect density</th>
<th>Why</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expensive automated design and testing tools</td>
<td>Requires training and maturity</td>
</tr>
<tr>
<td>Peer code reviews</td>
<td>Agenda is often adhoc or superficial</td>
</tr>
<tr>
<td>Complexity and depth of nesting metrics</td>
<td>Correlated at extremely high or low values but not in between</td>
</tr>
<tr>
<td>Advanced software life cycle models</td>
<td>Model not executed properly or it’s not the right model for this software product</td>
</tr>
</tbody>
</table>
These are the 10 factors mostly strongly related to software reliability

1. Software engineers have product/industry domain expertise
2. Do formal white/clear box unit testing
3. Start writing test plans before any code is written
4. Outsource features that aren’t in your organization’s line of business
5. Avoid outsourcing features that are your organization’s line of business
6. Don’t skip requirements, design, unit test or system testing even for small releases
7. Plan ahead – even for small releases. Most projects are late because of unscheduled defect fixes from the previous release (and didn’t plan on it)
8. Reduce “Big Blobs” - big teams, long milestones - especially when you have a large project
9. Don’t use automated tools until group has expertise in whatever the tool is automating
10. Define in writing what the software should NOT do
Conclusions

- Software reliability can be predicted before the code is written
  - It can be applied to COTS software as well as custom software
  - A variety of metrics can be predicted
  - The predictions can be used for sensitivity analysis and defect reduction
  - A variety of methods exist depending on how much data is available
Can I predict the software reliability when there is an agile or incremental software development lifecycle?

- Yes, your options are
  - You can use the models for each internal increment and then combine the results of each internal increment to yield a prediction for each field release
  - You can add up the code size predicted for each increment and do a prediction for the field release based on sum of all increment sizes

How often are the predictions updated during development?

- Whenever the size estimates have a major change or whenever there is a major review
- The surveys are not updated once complete unless it is known that something on the survey has changed
  - i.e. there is a major change in staffing, tools or other resource during development, etc.
Frequently Asked Questions

▶ Which defect density prediction models are preferred?

▶ The ones that you can complete accurately and the ones that reflect your application type

▶ If you can’t answer most of the questions in a particular mode survey then you shouldn’t use that model

▶ If the application lookup charts don’t have your application type you shouldn’t use them

▶ How can I get the defect density prediction models?

▶ [Software Reliability Toolkit Training Class](#)

▶ [Software Reliability Toolkit](#)

▶ Frestimate [Software](#)
References


[2] Four references are


c) Section 8 of MIL-HDBK-338B, 1 October 1998

Related Terms

- **Error**
  - Related to human mistakes made while developing the software
  - Ex: Human forgets that b may approach 0 in algorithm \( c = a/b \)

- **Fault or defect**
  - Related to the design or code
  - Ex: This code is implemented without exception handling “\( c = a/b; \)”
  - Defect rate is from developer’s perspective
  - Defects measured/predicted during testing or operation
  - Defect density = defects/normalized size

- **Failure**
  - An event
  - Ex: During execution the conditions are so that the value of b approaches 0 and the software crashes or hangs
  - Failure rate is from system or end user’s perspective

- **KSLOC**
  - 1000 source lines of code – common measure of software size