Defining necessary software reliability
(2)

Bojan Cukic

Department of Computer Science and Electrical Engineering
West Virginia University
So far

**SRE Ideas**

- Focus resources on the most critical/used functions
- Make testing realistically represent field conditions
- Process used throughout the life-cycle
- User oriented, rather than system oriented
- Help engineers/managers to learn how to make more precise decisions
- Providing QUANTITATIVE measurement of software testing
- SRE typically adding under 1% to project development cost
- Includes correctness, safety, operational aspects of usability
- DOES NOT include portability, modifiability, or understandability of documentation
Classifying failures

Failure severity class

- The set of failures that have common per-failure impact on users
- Used for prioritizing failure resolution process

Classification criteria:
- Human life
- Cost (extra operational cost, repair and recovery cost, loss of present or potential business)
- System capability impact (loss of critical data, recoverability, downtime and availability)

Defining failure severity classes requires brainstorming!!!
- Narrow down to the most significant ones
Procedure

- To define necessary reliability, one must
  - Define failure with severity classes for the product
  - Choose a common measure for all associated systems
  - Set a failure intensity objective for each system to be tested

- As a developer, (not a system integrator) you must
  - Find the developed software failure intensity objectives
  - Engineer strategies to meet the developed software failure intensity objective
“Feeling” reliability figures

<table>
<thead>
<tr>
<th>R (for 1h mission time)</th>
<th>Failure intensity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.386</td>
<td>1 failure/h</td>
</tr>
<tr>
<td>0.9</td>
<td>105 failures/1000h</td>
</tr>
<tr>
<td>0.959</td>
<td>1 failure/day</td>
</tr>
<tr>
<td>0.99</td>
<td>1 failure/100 h</td>
</tr>
<tr>
<td>0.994</td>
<td>1 failure/week</td>
</tr>
<tr>
<td>0.9986</td>
<td>1 failure/month</td>
</tr>
<tr>
<td>0.999</td>
<td>1 failure/1000 h</td>
</tr>
<tr>
<td>0.99989</td>
<td>1 failure/year</td>
</tr>
</tbody>
</table>

It helps to involve customers in defining requirements regarding failure rates
Your company is developing an automated garage parking system. The idea is that you leave your vehicle at the garage doors. A device is installed in your car which enables the system to automatically park it. They will charge $15 per service. When defining the necessary reliability, think of the following:

- Time vs. natural units of operation,
- Failure severity classes and damages,
- Public acceptance and attitudes towards failures,
- Profitability.

Typical approach: form a group of 2-3 members, one posing as the potential user.
If COTS will be integrated with custom developed components, how to determine appropriate failure intensities for the components under development?

- Estimate failure intensities of COTS (HW and SW)
  - Operational data
  - Vendor warranty
  - Experience of experts
- Add them
- Subtract this sum from the target system failure rate

This method provides just an estimate, useful for guiding decisions early in the lifecycle

Better methods available (demonstrated later)
Strategies to meet required failure rates

- Developers interested in good engineering practices
- System integrators more interested in reliability certification
- Fault prevention
  - requirements engineering
  - design methodologies
  - reviews, reviews, reviews
  - enforcing process standards
- Difficult to measure “fault content” as a result of process activities
  - Need adequate information from past projects, always difficult to collect
  - Matter of research
Prevention, removal, tolerance

- Fault removal
  - Code reviews and tests
  - Formal methods for code analysis
- Is it possible to measure the effectiveness of fault removal?
  - Possible, not necessarily precise.
  - Methodology originates from a technique called fault based testing
Estimating the number of residual faults (a quick method)

- $M$: the known number of injected faults.
- $k$: the total number of faults detected through testing.
- $m$: the number of injected faults detected by testing
  - $m \leq k$, $m \leq M$
- An estimate of the number of inherent faults $N$ is
  \[ N = \frac{M}{m} \times (k-m) \]
- Assumption: injected and inherent faults are equally likely to be detected
Other measures of fault removal

- Measure the difference in failure intensity at the beginning and the end of the test (if accompanied by debugging).
- If an inspection was performed in between two series of tests, the difference in failure rates is an indicator of success.
- Other techniques will be discussed later
Fault tolerance

- Achieved by design
- Identify likely deviations and implement countermeasures
  - Software replication
  - Triple modular redundancy with voting
  - n version programming
  - recovery blocks
  - design diversity
The right mix

- Generally unknown
- Low personnel retention rates do not help
- Data? Know of any?
- Every project is different, and the “BEST” strategy (processes, lines of defense) does not exist
- Some general guidelines come from experience (and are subjective)
The right mix

<table>
<thead>
<tr>
<th>Reliability level</th>
<th>Failure intensity range (per 1000 execution hours)</th>
<th>Strategy guidelines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ultrareliable</td>
<td>&lt; 0.1</td>
<td>Fault tolerance, extensive requirements and design reviews essential (IV&amp;V ?)</td>
</tr>
<tr>
<td>High reliability</td>
<td>0.1 - 10</td>
<td>Fairly extensive requirements and design reviews desirable, some fault tolerance</td>
</tr>
<tr>
<td>Commercial</td>
<td>10 to 2000</td>
<td>Guide any requirements or design reviews with operational profile and criticality</td>
</tr>
<tr>
<td>Prototype</td>
<td>&gt; 2000</td>
<td>Testing</td>
</tr>
</tbody>
</table>

- Voodoo, not science
- Prioritize resources based on the operational profile
Special situations

- Failure groupings by
  - Component (physical division)
  - Operation group (functional division)
  - Failure category (safety, user impacts)
  - Job role (data entry, customer service)

- Defining failure intensity similar to the case with multiple severities (adding intensities across all the groups)
Allocation of failure intensities to components

- Simple cases: HW & OS + application software
- Or more complex architectures

Failure intensity assignment:

1) Establish known component values
2) Pick a trial allocation of failure intensities, minimize system development time, risk, cost
3) Add component failure intensities and see whether the requirement is met
4) Modify allocations & goto 2
Component-Based Reliability

- Utilization of off-the-shelf software components for rapid application development.
- Such applications are expected to have high reliability as a result of deploying trusted components.
  - Product-line engineering case studies indicate improved software quality over product families.
- The claims of high reliability need further investigation.
- Most software reliability techniques treat a program as a \textit{monolith}.
  - It might be a \textit{collection of (potentially known) components}. 
Goals

- Probabilistic technique for reliability analysis applicable throughout the development life-cycle.
- The ability to study the sensitivity of the application reliability to reliabilities of components in the system.
  - Allow the system architect to select components with suitable reliability characteristics in cases when alternative reusable assets are available.
- Seamless integration with the information available as the annotations of UML.
SW Reliability and UML

- UML is a notation of choice for software development support.
  - The UML notation provides diagrams that capture software features.
  - No standard software process is required to be used with the UML notation.

- The UML diagrams are syntactically related.
  - Certain types of analyses, such as cross syntax checking, can be performed at the design time.
  - The graphical representation of the UML diagrams makes their comprehension easier.

- UML is open to notational extensions.
  - The potential for developing annotations of UML diagrams.
  - These annotations enrich the software representation and support for software V&V throughout the life-cycle.
Reliability Assessment Process

1. Provide annotations for application’s UML diagram(s).
2. Use annotations as inputs to reliability calculations.
3. Design level analysis (prediction):
   1. The algorithm predicts *expected* system reliability from provided (assumed, hoped for) component reliabilities.
   2. Algorithm supports system-wide cost-benefit analysis (what if I provide a more reliable component here?).
4. Reliability certification analysis (assessment):
   1. Corroborate expected system reliability by random system level testing.
Modeling Assumptions

- Component failure rates available.
- Failure Independence.
  - A component’s failure probability does not depend on the failure probabilities of the other components.
- Regularity
  - A component’s failure probability is the same across all the busy periods of a component.
- Pessimism, appropriate for early life-cycle.
  - Component failure results in a system failure.
  - Can be corrected in the corroboration phase.
UML annotations

Annotated Use Case Diagram

\[ P(x) = \sum_{i=1}^{m} q_i \times P_{ix} \]

Annotated Sequence Diagram

Annotated Deployment Diagram
Component failure probability within scenario $j$

$$\theta_{ij} = 1 - (1 - \theta_i)^{bp_{ij}}$$

Connector failure probability within scenario $j$

$$\psi_{lij} = 1 - (1 - \psi_k)^{|Interact(l,i,j)|}$$

System failure probability

$$\Theta_S = 1 - \sum_{j=1}^{K} p_j \left( \prod_{i=1}^{N} (1 - \theta_i)^{bp_{ij}} \cdot \prod_{k \in (l,i,j)} (1 - \psi_k)^{|Interact(l,i,j)|} \right)$$
Reliability prediction

- $\theta_i$ are random variables with *Beta* distributions. PDF is
  \[
g_i = \frac{1}{B(a_i, b_i)} \theta_i^{a_i-1} (1 - \theta_i)^{b_i-1}, 0 \leq \theta_i \leq 1.
\]

- System reliability:
  \[
  \Theta_S = 1 - \sum_{j=1}^{K} p_j \prod_{i=1}^{N} (1 - \Theta_i)^{b_{pij}}.
  \]

- Probability density function of $\Theta_S$ intractable.
  - Observe the values of $\Theta_S$ in simulations.
  \[
  \int_0^1 \frac{1}{B(a_i, b_j)} \nu^{a_i-1} (1 - \nu)^{b_i-1} d\nu = U_i
  \]

- Fit Beta distribution of $\Theta_S$ as a 95% confidence interval of 10,000 random observations.
Black-box operational testing.

$n$ tests yield $r$ observed failures, all test cases equally likely, the posterior is:

$$\hat{\Theta}_S(r/n) = \frac{a + r}{a + b + n}.$$

Scenario based system testing yields

$$\hat{\Theta}_{SS} = 1 - \sum_{j=1}^{K} p_j E[\prod_{i=1}^{N} (1 - \Theta_i)^{b p_{ij}} | n_j \text{ tests of the } j^{th} \text{ scenario resulted in } r_j \text{ failures }].$$
Design Level Analysis: An Example

A WEB-based transaction processing system (WBTPS)

Use cases:

Local Operations : 1 scenario.
Remote Read : 2 scenarios.
Remote Write : 1 scenario.
Annotated SD for “remote write” use case
Deployment Diagrams

Client

C1  C2
C4  C3

Web Server

C5  C6
C7

Remote Server

C10  C11
C12

Remote Application Server

C8  C9
## From Annotations to Reliability

### C²R²: Component - Connector Reliability Record

**Table 1: Information record of each component and connector.**

<table>
<thead>
<tr>
<th>component/connector</th>
<th>name</th>
<th>failure probability</th>
<th>confidence interval</th>
<th>number of busy periods</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$b_{p_{i1}}$</td>
</tr>
<tr>
<td>$C_1$</td>
<td>Main Interface</td>
<td>0.009</td>
<td>(0.006,0.012)</td>
<td>6</td>
</tr>
<tr>
<td>$C_2$</td>
<td>Client Application</td>
<td>0.005</td>
<td>(0.003,0.007)</td>
<td>3</td>
</tr>
<tr>
<td>$C_3$</td>
<td>Local DB</td>
<td>0.005</td>
<td>(0.003,0.007)</td>
<td>1</td>
</tr>
<tr>
<td>$C_4$</td>
<td>Browser</td>
<td>0.010</td>
<td>(0.007,0.013)</td>
<td>0</td>
</tr>
<tr>
<td>$C_5$</td>
<td>Web Interface</td>
<td>0.009</td>
<td>(0.006,0.012)</td>
<td>0</td>
</tr>
<tr>
<td>$C_6$</td>
<td>Web Application</td>
<td>0.005</td>
<td>(0.003,0.007)</td>
<td>0</td>
</tr>
<tr>
<td>$C_7$</td>
<td>DB</td>
<td>0.003</td>
<td>(0.001,0.005)</td>
<td>0</td>
</tr>
<tr>
<td>$C_8$</td>
<td>Remote Interface</td>
<td>0.009</td>
<td>(0.006,0.012)</td>
<td>0</td>
</tr>
<tr>
<td>$C_9$</td>
<td>Remote Application (RA)</td>
<td>0.005</td>
<td>(0.003,0.007)</td>
<td>0</td>
</tr>
<tr>
<td>$C_{10}$</td>
<td>RS Interface</td>
<td>0.039</td>
<td>(0.025,0.054)</td>
<td>0</td>
</tr>
<tr>
<td>$C_{11}$</td>
<td>RS Application</td>
<td>0.005</td>
<td>(0.003,0.007)</td>
<td>0</td>
</tr>
<tr>
<td>$C_{12}$</td>
<td>Remote DB (RS DB)</td>
<td>0.007</td>
<td>(0.003,0.010)</td>
<td>0</td>
</tr>
<tr>
<td>$l_1$</td>
<td>(Client, WEB Server)</td>
<td>0.009</td>
<td>(0.006,0.012)</td>
<td>-</td>
</tr>
<tr>
<td>$l_2$</td>
<td>(Client, RA Server(RAS))</td>
<td>0.009</td>
<td>(0.006,0.012)</td>
<td>-</td>
</tr>
<tr>
<td>$l_3$</td>
<td>(RAS, Remote Server)</td>
<td>0.003</td>
<td>(0.001,0.005)</td>
<td>-</td>
</tr>
</tbody>
</table>
Table 2: Number of interactions per pair of components overall the scenarios.

<table>
<thead>
<tr>
<th>pair (l,i)</th>
<th>scenario 1</th>
<th>scenario 2</th>
<th>scenario 3</th>
<th>scenario 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C₁, C₂)</td>
<td>2</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>(C₂, C₁)</td>
<td>2</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>(C₂, C₃)</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(C₃, C₂)</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(C₁, C₄)</td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>(C₄, C₁)</td>
<td>1</td>
<td>2</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>(C₄, C₅)</td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>(C₅, C₄)</td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>(C₅, C₆)</td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>(C₆, C₅)</td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>(C₆, C₇)</td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>(C₇, C₆)</td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>(C₄, C₈)</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(C₈, C₄)</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(C₈, C₉)</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(C₉, C₈)</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(C₉, C₁₀)</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(C₁₀, C₉)</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(C₁₀, C₁₁)</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(C₁₁, C₁₀)</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(C₁₁, C₁₂)</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(C₁₂, C₁₁)</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Component Reliabilities

Component Failure Probabilities as PDFs (Beta Distributions)
Reliability Prediction

95% confidence interval of system failure probability is $(0.114, 0.163)$.

Reliability range $(0.837, 0.886)$

Plot of Prior Probability Density Function of the System Failure Probability $\Theta_S$ fitted to the normalized histogram from simulation observations.
Sensitivity Analysis

- Change reliabilities of individual components (and/or connectors) and observe the impact on overall system reliability.

Improve Web servers
C₅: 0.009 -> 0.002
C₆: 0.005 -> 0.001
θₛ: 0.13 -> θₛ': 0.11

Worse remote servers
C₁₁: 0.005 -> 0.02
C₁₂: 0.007 -> 0.025
θₛ'': about 2% worse
Component reliability increases with its cost. At least, it should.

Optimization problem:

Find out component cost allocations so as to maximize system reliability subject to the fixed total budget.

The methodology has been developed and applied to the running example.

Outstanding issues: expressing component reliability as a function of cost, discrete cost functions.
Experimentation

- **Personnel Access Control System (PACS)**
- **NSA certified specification**
- **Actors:** regular users and security officers.
## Analytical Results

<table>
<thead>
<tr>
<th>Component</th>
<th>Name</th>
<th>Failure Probability</th>
<th>Confidence Interval</th>
<th>Busy Periods</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>bpi1</td>
</tr>
<tr>
<td>C1</td>
<td>Communication Driver</td>
<td>0.002</td>
<td>(0.001, 0.003)</td>
<td>24</td>
</tr>
<tr>
<td>C2</td>
<td>Pacs</td>
<td>0.005</td>
<td>(0.003, 0.007)</td>
<td>4</td>
</tr>
<tr>
<td>C3</td>
<td>Validator</td>
<td>0.002</td>
<td>(0.001, 0.003)</td>
<td>1</td>
</tr>
<tr>
<td>C4</td>
<td>User LCD</td>
<td>0.006</td>
<td>(0.003, 0.009)</td>
<td>3</td>
</tr>
<tr>
<td>C5</td>
<td>Officer LCD</td>
<td>0.006</td>
<td>(0.003, 0.009)</td>
<td>0</td>
</tr>
</tbody>
</table>

### Prediction

<table>
<thead>
<tr>
<th>No error propagation probability included.</th>
<th>0.082</th>
<th>(0.062, 0.103)</th>
<th>System failure probability worse then required.</th>
<th>More reliable components needed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error propagation probability included.</td>
<td>0.221</td>
<td>(0.167, 0.278)</td>
<td>System failure probability much worse than above.</td>
<td>The effect of error propagation!</td>
</tr>
</tbody>
</table>
## Analysis

### Error propagation matrix:

<table>
<thead>
<tr>
<th></th>
<th>Driver</th>
<th>Pacs</th>
<th>Validator</th>
<th>UserLCD</th>
<th>OfficerLCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver</td>
<td>0</td>
<td>0.425</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pacs</td>
<td>0.804</td>
<td>0</td>
<td>0.024</td>
<td>0.083</td>
<td>0.073</td>
</tr>
<tr>
<td>Validator</td>
<td>0</td>
<td>0.687</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UserLCD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OfficerLCD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- So, improving the reliability of the communication driver should help!
… we improve the Communication Driver, which is highly used and leaks errors?

<table>
<thead>
<tr>
<th>Prediction</th>
<th>Failure Probability</th>
<th>Confidence Interval</th>
<th>Discussion</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>No error propagation probability included.</td>
<td>0.060</td>
<td>(0.047, 0.074)</td>
<td>Mean failure probability reduced 25% (0.082 to 0.060)</td>
<td>Driver component failure probability reduced by 50%.</td>
</tr>
<tr>
<td>Error propagation probability included.</td>
<td>0.156</td>
<td>(0.122, 0.192)</td>
<td>Mean failure probability reduced 30% (0.22 to 0.156)</td>
<td>Driver component failure probability reduced by 50%. More significant improvement than above.</td>
</tr>
</tbody>
</table>
Summary

- A Bayesian framework for reliability prediction in component based systems.
  - Fully integrated with UML.
  - The first reliability algorithm that integrates reliability prediction (design level analysis) with reliability certification (system integration and analysis task).
  - Tool support available.

- Limitations:
  - Still rather simplistic (assumes failure independence, the same as all the other similar algorithms do).
Further Work

- Remove failure independence assumption.
  - Error propagation to be presented later today may be a good candidate for extension.
- Extend the level of detail in UML annotations.
- Include reliability optimization into the tool support.
- Integrate with risk analysis (through fault trees).
- Apply to realistic applications.
- Planning a proposal for consideration of this methodology in the standardization effort by OMG.