Software Reliability Engineering: Introduction (Part 2) + Defining Needed Reliability

Bojan Cukic

Lane 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
Few more definitions

- **Software availability**: fraction of time when the system is functioning acceptably
  - uptime/(uptime+downtime)

- **Maintainability**: indicated by the average staff hours needed to resolve a failure.

- **Reliability measurement**
  - *Execution time* (time actually spent by the processor executing instructions of the given program).
    - Models based on ET are more accurate
  - *Wall-clock time*
    - All decisions must be related to WcT to be meaningful
Characterizing failure occurrences

- Time to failure
- Time interval between failures
- Cumulative failures experienced up to a given time
- Failures experienced in a given time interval

<table>
<thead>
<tr>
<th>Failure No</th>
<th>Failure time (s)</th>
<th>Failure Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>19</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>32</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>43</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>58</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>70</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>88</td>
<td>18</td>
</tr>
<tr>
<td>8</td>
<td>103</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>125</td>
<td>22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Cumulative failures</th>
<th>Failures in interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>60</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>90</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>120</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>150</td>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>
Nature of measurements

- Failure occurrences are *random variables*
  - We do not know their value with certainty
  - Random does not mean unpredictable
    - But exact value is unknown
    - Averages and dispersion are known

- Why random
  - Commission of faults by programmers is complex and unpredictable
  - Location of faults unknown
  - Conditions of program executions are unknown
  - Complex interaction between functionality and program paths
Probability distributions

Probability distribution of failures in a time interval

<table>
<thead>
<tr>
<th>Failures in an interval</th>
<th>Probability</th>
<th># failures * Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.18</td>
<td>0.18</td>
</tr>
<tr>
<td>2</td>
<td>0.22</td>
<td>0.44</td>
</tr>
<tr>
<td>3</td>
<td>0.16</td>
<td>0.48</td>
</tr>
<tr>
<td>4</td>
<td>0.11</td>
<td>0.44</td>
</tr>
<tr>
<td>5</td>
<td>0.08</td>
<td>0.4</td>
</tr>
<tr>
<td>6</td>
<td>0.05</td>
<td>0.3</td>
</tr>
<tr>
<td>7</td>
<td>0.04</td>
<td>0.28</td>
</tr>
<tr>
<td>8</td>
<td>0.03</td>
<td>0.24</td>
</tr>
<tr>
<td>9</td>
<td>0.02</td>
<td>0.18</td>
</tr>
<tr>
<td>10</td>
<td>0.01</td>
<td>0.1</td>
</tr>
<tr>
<td>Mean # failures</td>
<td></td>
<td>3.04</td>
</tr>
</tbody>
</table>

\[ E[X] = \text{SUM}(X_i) \times P(X_i) \]
Measurements

- Typical variation of *failure intensity* and *reliability* over testing
- Each expression has its advantages
- Curves not necessarily so smooth
- Alternatives
  - MTTF (larger better), but may be undefined.
  - MTBF=MTTF+MTTR (comes from HW reliability)
Example

<table>
<thead>
<tr>
<th>Failures in period of time</th>
<th>Probability After 1 h</th>
<th>Probability After 5 h</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.1</td>
<td>0.01</td>
</tr>
<tr>
<td>1</td>
<td>0.18</td>
<td>0.02</td>
</tr>
<tr>
<td>2</td>
<td>0.22</td>
<td>0.03</td>
</tr>
<tr>
<td>3</td>
<td>0.16</td>
<td>0.04</td>
</tr>
<tr>
<td>4</td>
<td>0.11</td>
<td>0.05</td>
</tr>
<tr>
<td>5</td>
<td>0.08</td>
<td>0.07</td>
</tr>
<tr>
<td>6</td>
<td>0.05</td>
<td>0.09</td>
</tr>
<tr>
<td>7</td>
<td>0.04</td>
<td>0.12</td>
</tr>
<tr>
<td>8</td>
<td>0.03</td>
<td>0.16</td>
</tr>
<tr>
<td>9</td>
<td>0.02</td>
<td>0.13</td>
</tr>
<tr>
<td>10</td>
<td>0.01</td>
<td>0.1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0.07</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0.05</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0.03</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0.02</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0.01</td>
</tr>
</tbody>
</table>

\[E(X)\] 3.04 7.77
Failure behavior

- Two most important factors guiding failure occurrences
  - The number of faults in software
  - Operational usage of software
- Failure process depends on the system being built, and its use
- Software reliability has originated from hardware reliability
  - Relationships were the matter of research some 25 years ago
  - “Design reliability” does not exist in hardware, and that’s the most important aspect of software reliability
Reliability Theory

Reliability vs. *Failure probability*:

\[ R(t) = 1 - F(t) \]

In software reliability hazard rate is equivalent to *failure intensity*.

The relationship between MTTF and reliability:

\[ MTTF = \int_{0}^{\infty} R(t) \, dt \]

Integration performed over the operating time of the system.
SW vs. HW reliability

- Research topic in the late 70’s.
  - The division is somewhat artificial.
- Both depend on environment.
  - Major source of failures in software are design faults.
  - In HW, the major source of failures is physical deterioration.
    - Design reliability in HW has not been heavily studied!
- Software, in general, logically more complex.
  - SW reliability volatile in design and test (rework).
  - HW reliability volatile at burn-in and burn-out.
To Summarize

- Definition of software reliability.
- Software reliability engineering is the process that leads to high reliability software.
  - Based on statistical evaluation of quality factors throughout the development lifecycle.
- Reliability can be assessed using different approaches.
- Simple activities can significantly reduce software failure rates.
Defining necessary reliability

- One of the key steps in the overall SRE process
  - Balance between needed reliability, delivery date, cost
- Definitions
  - *Failure*: departure from the specified system behavior (user oriented concept)
  - *Fault*: The defect that causes errors and failures (developer oriented concept)
- Hardware, software, human, documentation F/F
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
Failure severity classifications

- Severity classes based on cost
  - Class | Cost
  - 1    | > 100,000
  - 2    | 10,000 - 100,000
  - 3    | 1,000 - 10,000
  - 4    | < 1,000

- Severity classes based on system capability impact
  - Class | Description
  - 1    | Unavailability of one or more key operations
  - 2    | Unavailability of one or more important operations
  - 3    | Unavailability as above, but workarounds available
  - 4    | Minor deficiencies in one or more operations
Simplicity and intuitive appeal

- HW: based on wall clock time
- SW: based on execution time
  - 1 failure per 1,000 printed [pages/telephone calls]
  - Applicable to distributed systems due to the execution time origin

System failure intensity is the sum of component failure intensities (not so for reliabilities)
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
Setting system failure intensity

- Start with subsystems, aggregate to the system level
- Determine user’s satisfaction with current systems
- At a given release, the product of the following factors tends to be constant
  - *Failure intensity*  
    - Down  
    - Up
  - *Development time*  
    - Up  
    - Up
  - *Development cost*  
    - Up  
    - Down
- BUT, operational cost may decrease with lower failure intensity
- Exact relationships may not be known: need for data collection from different projects
Typical failure intensity objectives

<table>
<thead>
<tr>
<th>IMPACT</th>
<th>Objective</th>
<th>TBF</th>
</tr>
</thead>
<tbody>
<tr>
<td>100’s deaths, &gt;$10^9$ cost</td>
<td>$10^{-9}$</td>
<td>114,000y</td>
</tr>
<tr>
<td>1-2 deaths, around $10^6$ cost</td>
<td>$10^{-6}$</td>
<td>114 y</td>
</tr>
<tr>
<td>$1,000$ cost</td>
<td>$10^{-3}$</td>
<td>6 weeks</td>
</tr>
<tr>
<td>$100$ cost</td>
<td>$10^{-2}$</td>
<td>100 h</td>
</tr>
<tr>
<td>$10$ cost</td>
<td>$10^{-1}$</td>
<td>10 h</td>
</tr>
<tr>
<td>$1$ cost</td>
<td>1</td>
<td>1 h</td>
</tr>
</tbody>
</table>
Defining failure rates

- **Product specific**

- **What to do when system has severity levels?**
  - Tricky. If one failure class, there is no problem
  - Otherwise, set failure intensities for each severity class, using the ratios between target intensities between classes
    - Side effect is the reduction in the sample size for measurements

- Optionally, define different failure intensities for different system operations
Reliability vs. failure intensity

- **Simplified conversion rules**
  - \( f_i = -\ln(R)/t \)
  - Assumes constant failure rate
  - \( t \) expressed in appropriate units (execution time)
  - If \( R > 0.95 \), then \( f_i \sim (1-R)/t \)
  - \( R = e^{-f_i t} \), or
  - \( R \sim 1 - f_i t \)

- If reliability is 0.992 over 8 hours, what is the corresponding failure intensity?
Reliability vs. failure intensity

 Availability: The fraction of up-time

\[ A = \frac{t_u}{t_u + t_d} \]

\[ t_d = t_m \times f_i \times t_u \] where \( t_m \) represents the downtime per failure, i.e., time to recover (not time to debug)

\[ f_i = \frac{1-A}{A \times t_m} \]

Define required failure intensity if the product must be available 99% of the time, and downtime is 6 minutes?
"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
Workshop

- 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
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>

 disclosed.

- 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:**

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