Software Reliability Engineering:
Introduction

Bojan Cukic

Lane Department of Computer Science and Electrical Engineering
West Virginia University
Goals

- Traditional problems with software development
  - (Un)reliability of released products
  - Missed schedules
  - Cost overruns
  - All the attributes of software quality.
- Positive effects on market share and profitability.
- Attention paid on *engineering* aspects of software engineering.
  - Resolution of conflicting quality related demands
  - Quantitative guidance for software development process.
To *engineer* or not to …

- Traditional software testing alone does not make the product reliable!
  - Assurance activities need to be included in all the phases of the development.
- Software reliability engineering practices empower quality engineers (testers) to take leadership in advocating user needs!
  - But it involves almost everyone else, system engineers, architects, designers, managers…
Why SRE?

- Proven, standard best practice.
- Additional advantages (time to market, cost).
- Widely applicable.
- Low cost of application within any project.
  - Observed increase in project cost: less than 1%.
- Minor impact on schedule in terms of adding new activities.
- Can be deployed in stages to maximize adoption benefits.
What is SRE

- The set of best practices that empower testers and developers to
  - Ensure product reliability meets users needs
  - Speed the product to market faster
  - Reduce product cost
  - Improve customer satisfaction (fewer angry users)
  - Increase their productivity

- Applicable to all software based systems
Some definitions

- **Reliability**: The probability that the system or a capability of a system will function without failure for a specified period of time (or natural units) in a specified environment.

- **Natural units**: Amount of processing performed by the software based product (runs, pages of output, transactions, telephone calls, queries, API calls…).
More definitions

- **Failure intensity (FI):** The number of failures per natural or time unit, an alternative way of expressing reliability.

- **Availability:** The average (over time) probability that a system or a capability of a system is functional in a specified environment.
Fundamental ideas (1)

- Increase the effectiveness of available resources.
  - Quantitatively characterize expected use.
  - Focus resources on the most used/critical functions
    - Critical: having great extra value when successful, or great impact if not.
  - Make testing realistically represent field conditions.
    - Balance customer needs.
Fundamental ideas (2)

- Apply resources to maximize customer value.
  - Set quantitative objectives for major quality characteristics (reliability and/or availability, delivery time, price).
  - Track reliability in system test against given objective as one of the release criteria.
Impacts

- AT&T International Definity PBX
  - Reduced customer-reported problems by the factor of 10
  - Reduced system test interval by the factor of 2.
  - Reduced total development time by 30%.
  - No serious service outages in 2 years of deployment.

- Widespread practice: Alcatel, AT&T, Telcordia, Lucent, Ericsson, HP, Lockheed-Martin, Microsoft, Mitre, Tandem, Saab Aircraft…
Wide applicability

- Suitable for any software based product.
- Complete SRE may be impractical for small components (<2 months of staff months effort)
  - Unless used in product lines/families.
  - Could be applied in an abbreviated form.
- Independent of development technology and platform.
- Requires no changes in architecture, design, code, but may suggest beneficial ones.
SRE Process

- Widely used and accepted, especially by the large corporations (Microsoft included!!)

- Predominant SRE workflow:

  - List Associated Systems
  - Define Necessary Reliability
  - Develop Operational Profiles
  - Prepare for Test
  - Execute Tests & Apply Failure Data to Guide Decisions

  Requirements and architecture  Design and Implementation  Test & Validation
SRE Process

- Tasks frequently iterate.
- Post-delivery and maintenance phase (not shown)
- Testers involved throughout the process
  - Allows better understanding of user’s perspective
  - Improvement of system requirements, planning
- Selection of appropriate mix of
  - fault prevention
  - fault removal
  - fault tolerance
Test types

- Types of tests applicable to SRE (based on objectives, rather than phases in the life-cycle)
  - Reliability growth tests (find and remove faults)
    - need a minimum of 10-20 detected faults to achieve statistically meaningful results
  - **Feature** (minimize impact of the environment), **load** (maximize environmental impacts), **regression** tests (following a major change)
  - Certification tests
    - no debugging, accept or reject software under test
    - no. observed failures not important
Defining the “system”

- System is an independently tested unit
- SRE should be applied to subsystems (acquired COTS, OS, for example), systems and supersystems
- Different configuration represents different system
  - Interface stubs may not be correct
- But, more “systems” imply higher cost.
  - Aggregation welcome
  - Product lines help reducing the cost
SRE and SW design & test process

- Use knowledge of operational profile to guide and focus design efforts
- Established failure intensity drives the quality assurance efforts
- Failure intensity goal determines when to stop testing
- Measurement throughout the life-cycle helps identify better methodologies
Is Reliability Important?

- It should be, since it is a measurable property
  - Unlike “software quality”
- Useful, since the software is tested under the conditions of perceived usage.
  - The number of resident faults, for example, is a developer oriented measure. Reliability is a user oriented measure.
  - The number of faults found has NO correlation to reliability. Neither has program complexity.
- Accurate measurements of reliability are feasible.
Isn’t the “best software development process” sufficient?
  
What is “best”?

It is important to measure the results of the process.

Early consideration of target reliability is beneficial, since it impacts cost and schedule.

CMM levels 4 and 5 (and 3, indirectly), recommend reliability measurement.
Common Misconceptions

- Software reliability is primarily concerned with software reliability models.
- It copies hardware reliability theory.
  - No, because reliability of software is more likely to change over time (modifications, upgrades).
- It deals with faults or “bugs”.
- It does not concern itself with requirements based testing.
- Testing “ultrareliable” software is hopeless.
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>
</tbody>
</table>

Mean # failures: 3.04

\[ E[X] = \sum(X_i) \times P(X_i) \]
### 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>

\[
E(X) = 3.04 \quad 7.77
\]
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
Observe failure occurrences in terms of *execution time*.

<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>
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)
Reliability vs. \textit{Failure probability}: 

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

In software reliability hazard rate is equivalent to \textit{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.
Summary

- 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.