SENG 421: Software Metrics

Software Reliability Models & Metrics
(Chapter 9)

Department of Electrical & Computer Engineering, University of Calgary

B.H. Far (far@ucalgary.ca)
http://www.enel.ucalgary.ca/People/far/Lectures/SENG421/09/

Contents

- Reliability concepts and definitions
- Software reliability models and metrics
- Fundamentals of software reliability engineering (SRE)
- Reliability management models
What Affects Software?

- **Time:**
  - Meeting the project deadline.
  - Reaching the market at the right time.
- **Cost:**
  - Meeting the anticipated project costs.
- **Quality:**
  - Working fine for the designated period on the designated system.

What Affects Software Quality?

- **Delivery Date:**
  - Meeting the project deadline.
  - Reaching the market at the right time.
- **Cost:**
  - Meeting the anticipated project costs.
- **Performance:**
  - Working fine for the designated period on the designated system, i.e., reliability, availability, etc.
Failure, Failure Intensity & Availability

- **Failure**: Any departure of system behavior in execution from user needs or expectations.
- **Failure intensity**: the number of failures per natural unit or time unit. Failure intensity is a way of expressing reliability.
- **Availability**: The probability at any given time that a system or a capability of a system functions satisfactorily in a specified environment.

\[
\text{Availability} = \frac{Uptime}{Uptime + Downtime}
\]

Error, Fault and Failure

- An **error** is a human action that results in software containing a fault.
- A **fault** (bug) is a cause for either a failure of the program or an internal error (e.g., an incorrect state, incorrect timing).
- Among the 3 factors only **failure** is observable.
Reliability & Reliability Engg

- **Reliability**: The probability that a system or a capability of a system functions without failure for a specified time in a specified environment.

- **Reliability Engineering**: Engineering of reliability in software products.

- **Reliability Engineering’s goal**: Developing software to reach the market
  - With “minimum” development time
  - With “minimum” development cost
  - With “maximum” reliability

Hardware Reliability Models

- **Uniform model**: Probability of failure is fixed.
  \[ f(t) = \begin{cases} 1 & t \leq T \\ 0 & t > T \end{cases} \]

- **Exponential model**: Probability of failure changes exponentially over time
  \[ f(t) = \lambda_0 e^{-\beta t} \]
Software vs. Hardware Reliability

- Software reliability doesn’t decrease with time, i.e., software doesn’t wear out.
- Hardware faults are mainly physical faults, e.g., fatigue. Software faults are mainly design faults which are harder to measure, model, detect and correct.
- Hardware failure can be “fixed” by replacing a faulty component with an identical one, therefore no growth. Software problems can be “fixed” by changing the code in order to have the failure not happen again, therefore reliability growth is present.
- **Conclusion:** hardware reliability models may not be used identically for software.

Reliability Questions

**Single failure specification:**
- What is the probability of failure of a system (or a component)?

**Multiple failure specification:**
- If a system (or a component) fails at time $t_1$, $t_2$, ..., $t_{i-1}$, what is the probability of its failure at time $t_i$?
Reliability Metrics (Single Failure) /1

- **Probability density function (PDF):** depicting changes of the probability of failure up to a given time $t$.
- A common form of PDF is exponential distribution.
- Usually we want to know how long a component will behave correctly before it fails, i.e., the probability of failure from time $0$ up to a given time $t$.

**Exponential PDF:**

$$ f(t) = \begin{cases} \lambda e^{-\lambda t} & t \geq 0 \\ 0 & t < 0 \end{cases} $$

Reliability Metrics (Single Failure) /2

- **Cumulative density function (CDF):** depicting cumulative failures up to a given time $t$.

$$ F(t) = \int_0^t f(t) $$

- For exponential distribution, CDF is:

$$ F(t) = \begin{cases} 1 - e^{-\lambda t} & t \geq 0 \\ 0 & t < 0 \end{cases} $$
Reliability Metrics (Single Failure) /3

- **Reliability function** *(R)*: depicting a component functioning without failure until time *t*.

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

- For exponential distribution, *R* is:

\[
R(t) = e^{-\lambda t}
\]

Reliability Metrics (Single Failure) /4

- What is the expected value of failure time *T*?

- It is the mean of the probability density function (PDF), named **mean time to failure** *(MTTF)*

\[
E(T) = \int_{0}^{\infty} t f(t) \, dt
\]

- For exponential distribution, MTTF is:

\[
MTTF = \frac{1}{\lambda}
\]
Reliability Metrics (Single Failure) /5

- **Median time to failure ($t_m$):** a point in time that the probability of failure before and after $t_m$ are equal.
  \[
  \int_0^{t_m} f(t) dt = \frac{1}{2} \quad \text{or} \quad F(t_m) = \frac{1}{2}
  \]

- **Failure (Hazard) Rate $z(t)$:** Probability density function divided by reliability function.
  \[
  z(t) = \frac{f(t)}{R(t)}
  \]
  For exponential distribution, $z(t)$ is: $\lambda$

Reliability Metrics (Single Failure) /6

- **Serial System Reliability:** is multiplication of the reliability of its components.
  \[
  R_{\text{system}}(t) = \prod_{i=1}^{n} R_i(t)
  \]

- For exponential distribution:
  \[
  R_{\text{system}}(t) = e^{-\lambda_1 t} e^{-\lambda_2 t} \cdots e^{-\lambda_n t} = e^{-\left(\lambda_1 + \lambda_2 + \cdots + \lambda_n\right)t}
  \]
  \[
  R_{\text{system}}(t) = e^{-\left(\sum_{i=1}^{n} \lambda_i\right)t}
  \]
Reliability Metrics (Single Failure)

- **System Cumulative Failure Rate:** is the sum of the failure rate of its components.

\[
\lambda_{\text{system}}(t) = \sum_{i=1}^{n} \lambda_i(t)
\]

- For exponential distribution:

\[
\lambda_{\text{system}}(t) = \lambda_1 + \lambda_2 + \cdots + \lambda_n = \sum_{i=1}^{n} \lambda_i
\]

Software Reliability Models: Multiple Failure Specification
Reliability Questions

Single failure specification:
- What is the probability of failure of a system (or a component)?
  (discussed in the previous session)

Multiple failure specification:
- If a system (or a component) fails at time $t_1$, $t_2$, …, $t_{i-1}$, what is the probability of its failure at time $t_i$?

Reliability Metrics (Multiple Failure) /1

- One can assume that the probability of failure (probability density function, PDF) for all failures are the same (e.g., replacing the faulty hardware component with an identical one).

- In software, however, we want to “fix” the problem, i.e., have a lower probability of failure after a repair (or longer $\Delta t_i = t_i - t_{i-1}$). Therefore, we need a model for reliability growth (i.e., reliability change over time).
Reliability Growth Models

- Common software reliability growth models are:
  - **Basic Exponential model**
  - **Logarithmic Poisson model**

- The basic exponential model assumes finite failures ($\nu_0$) in infinite time.
- The logarithmic Poisson model assumes infinite failures.

Validity of the Models

- Software systems are changed (updated) many times during their life cycle.
- The models are good for one revision period rather than the whole life cycle.
Parameters involved in reliability growth models:

1) Failure intensity (failure rate) ($\lambda$): number of failures per natural unit or time unit.
2) Execution time ($\tau$): time since the program is running. Execution time may be different from calendar time.
3) Mean failures experienced ($\mu$): mean failures experienced in a time interval.

Mean failures experienced ($\mu$) for a given time period (e.g., 1 hour execution time) is calculated as:

$$\mu = \sum_{i=1}^{n} i \times p_i$$

$p_i$: probability of occurrence

$i$: number of failures

<table>
<thead>
<tr>
<th>No. of failures in interval ($n$)</th>
<th>Probability ($p$)</th>
<th>$n \times p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.10</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>0.40</td>
</tr>
<tr>
<td>6</td>
<td>0.05</td>
<td>0.30</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>0.10</td>
</tr>
</tbody>
</table>

Mean failure ($\mu$) 3.04
Reliability Growth Models /3b

<table>
<thead>
<tr>
<th>Failure(s) in time period</th>
<th>Probability Elapsed time (1 hour)</th>
<th>Probability Elapsed time (5 hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.10</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>0.10</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>
<tr>
<td>Mean</td>
<td>3.04</td>
<td>7.77</td>
</tr>
</tbody>
</table>

Failure intensity ($\lambda$) versus execution time ($\tau$)

(B) $\lambda(\tau) = \lambda_0 e^{(-\frac{\lambda_0}{\nu_0})\tau}$

(P) $\lambda(\tau) = \frac{\lambda_0}{\lambda_0 \theta \tau + 1}$

- $\lambda_0$ Initial failure intensity
- $\nu_0$ Total failures
- $\theta$ Decay parameter
Reliability Growth Models /5

- Failure intensity ($\lambda$) versus mean failures experienced ($\mu$)

\[
\begin{align*}
\text{(B)} & \quad \lambda(\mu) = \lambda_0 \left(1 - \frac{\mu}{\nu_0}\right) \\
\text{(P)} & \quad \lambda(\mu) = \lambda_0 e^{-\theta \mu}
\end{align*}
\]

Reliability Growth Models /6

- Mean failures experienced ($\mu$) versus execution time ($\tau$)

\[
\begin{align*}
\text{(B)} & \quad \mu(\tau) = \nu_0 \left[1 - e^{-\frac{\lambda_0 \tau}{\nu_0}}\right] \\
\text{(P)} & \quad \mu(\tau) = \left(\frac{1}{\theta}\right) \ln(\lambda_0 \theta \tau - 1)
\end{align*}
\]
Failure Specification /1

1) Time of failure
2) Time interval between failures
3) Cumulative failure up to a given time
4) Failures experienced in a time interval

<table>
<thead>
<tr>
<th>Failure no.</th>
<th>Failure times (hours)</th>
<th>Failure interval (hours)</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>
<tr>
<td>10</td>
<td>150</td>
<td>25</td>
</tr>
<tr>
<td>11</td>
<td>169</td>
<td>19</td>
</tr>
<tr>
<td>12</td>
<td>199</td>
<td>30</td>
</tr>
<tr>
<td>13</td>
<td>231</td>
<td>32</td>
</tr>
<tr>
<td>14</td>
<td>256</td>
<td>25</td>
</tr>
<tr>
<td>15</td>
<td>296</td>
<td>40</td>
</tr>
</tbody>
</table>

Time based failure specification

Failure Specification /2

1) Time of failure
2) Time interval between failures
3) Cumulative failure up to a given time
4) Failures experienced in a time interval

<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>10</td>
<td>2</td>
</tr>
<tr>
<td>180</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>210</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>240</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>270</td>
<td>14</td>
<td>1</td>
</tr>
</tbody>
</table>
How to Use the Models

- **Release criteria**: time required to test the system to reach a target failure intensity:

  \[
  \Delta \tau = \frac{V_0}{\lambda_0} \ln \frac{\lambda_p}{\lambda_F}
  \]

  \[
  \Delta \tau = \frac{1}{\theta}\left(\frac{1}{\lambda_F} - \frac{1}{\lambda_p}\right)
  \]

  \(\lambda_p\): Present failure intensity

  \(\lambda_F\): Target failure intensity

Reliability Metrics

- **Mean time to failure (MTTF)**: Usually calculated by dividing the total operating time of the units tested by the total number of failures encountered (assuming that the failure rate is constant).

  Example:
  - MTTF for Windows 2000 Professional is 2893 hours or 72 forty-hour workweeks.
  - MTTF for Windows NT Workstation is 919 hours or 23 workweeks.
  - MTTF for Windows 98 is 216 hours or 5 workweeks.

- **Mean time to repair (MTTR)**: mean time to repair a (software) component.

- **Mean time between failures (MTBF)**:

  \[\text{MTBF} = \text{MTTF} + \text{MTTR}\]
Reliability Metrics: Availability

- Software System Availability \((A)\):
  \[
  A(t) = \frac{1}{1 + t_m \lambda(t)} \quad \text{or} \quad \lambda(t) = \frac{1 - A(t)}{t_m A(t)}
  \]
  - \(\lambda\) is failure intensity
  - \(t_m\) is downtime per failure

- Another definition of availability:
  \[
  A = \frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF}
  \]

- Example: If a product must be available 99% of time and downtime is 6 min, then \(\lambda\) is about 0.1 failure per hour (1 failure per 10 hours) and MTTF=594 min.

Reliability Metrics: Reliability

- Software System Reliability \((R)\):
  \[
  \lambda(t) = -\frac{\ln R(t)}{t} \quad \text{or} \quad \lambda \approx \frac{1 - R(t)}{t} \quad \text{for} \quad R(t) \geq 0.95
  \]
  - \(\lambda\) is failure intensity
  - \(R\) is reliability
  - \(t\) is natural unit (time, etc.)

- Example: for \(\lambda=0.001\) or 1 failure for 1000 hours, reliability \((R)\) is around 0.992 for 8 hours of operation.
Using Reliability Models

- Suppose that we have developed and tested our software and documented the failures for a certain period of time.
- How do we know that the reliability software is getting better after bug fixes?
- How to measure the increase/decrease of reliability?

SRE Tools: CASRE

- Computer-Aided Software Reliability Estimation (CASRE) Tool.
- CASRE is a PC-based tool that was developed in 1993 by the Jet Propulsion Laboratories.
- CASRE requires the WINDOWS operating environment.
- Once the data is entered, CASRE automatically provides the analyst with a raw data plot.
- CASRE provides the analyst with the ability to convert from time-domain data to interval-domain data and vice versa.
SRE Tools: CASRE /2

- CASRE provides operations to transform or smooth the failure data; the user can select and/or define multiple models for application to the data and make reliability predictions based on the best model.

Using CASRE /1

1. Prepare input data

- Input data can be either failure count or failure per interval data

Sample failure count data

<table>
<thead>
<tr>
<th>&lt;failure number&gt;</th>
<th>&lt;number of natural or time units since previous failure&gt;</th>
<th>&lt;severity class&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>55</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>60</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>90</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>150</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>120</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>215</td>
<td>1</td>
</tr>
</tbody>
</table>
Using CASRE /2

2. Read input file
3. Normalize data by multiplying by failure intensity objective ($\lambda_F$)
4. Select data range
5. Select and run model(s)
6. View and interpret results

Case Study

- Project X is a web based application for accessing a database using a browser.
- This version of the software is a minor release with changes to the GUI display and data access engine.
- Two programmers were assigned to the project. One programmer worked on the GUI, and the other on the data access engine.
- The project took approximately 4 weeks to complete.
Case Study (contd.)

- A single tester was assigned to the project.
- The test phase was completed in approximately 25 hours (3 working days or 90,000 seconds).
- 136 failures were discovered during the testing.
- Using the dates and times recorded for the failures discovered during testing, a “time between failures” input file was generated for CASRE.
- The severity of all the failures was set to
  - 1 - Low Severity

Time Between Failures Plot
Project Results

In order to determine which models would provide the best fit for the project data, the following models were run:

- Geometric
- Jelinski - Moranda
- Littlewood - Verrall
- Musa Basic
- Musa - Okumoto

Goodness of Fit Test

On Graphic display window select: Display → Goodness of fit

<table>
<thead>
<tr>
<th>Model Name</th>
<th>K3 Distance</th>
<th>95% Fit?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometric</td>
<td>8.81101e-002</td>
<td>Yes</td>
</tr>
<tr>
<td>Jelinski-Moranda</td>
<td>9.443151e-002</td>
<td>Yes</td>
</tr>
<tr>
<td>Littlewood-Verrall</td>
<td>7.504933e-002</td>
<td>Yes</td>
</tr>
<tr>
<td>Musa Basic</td>
<td>9.203077e-002</td>
<td>Yes</td>
</tr>
<tr>
<td>Musa-Okumoto</td>
<td>8.791571e-002</td>
<td>Yes</td>
</tr>
</tbody>
</table>
CASRE Model Ranking

On Graphic display window select:
Display → Model rankings → Rank summary or Rank details

Display Results

On Graphic display window select:
Results → Select model results

Only 3 graphs can be displayed at a time
Interpreting Results /1

- Accuracy of estimation of the $\lambda/\lambda_F$ ratio depends on the number of failures experienced (i.e., the sample size).
- Good results in estimating failure intensity are generally experienced for programs with 5,000 or more developed source lines and at least 20 failure data points.

Interpreting Results /2

- When the $\lambda/\lambda_F$ ratio is very large and the trend indicates little chance of achieving the $\lambda_F$ by the scheduled release date, consider taking one or more of the following actions:
  - Adding additional test and debugging resources
  - Adjusting the balance among the objectives for failure intensity, development time, and development cost
  - Deferring features
SRE: Process

- There are 5 steps in SRE process (for each system to test):
  1) Define necessary reliability
  2) Develop operational profiles
  3) Prepare for test
  4) Execute test
  5) Apply failure data to guide decisions

1. Failure Intensity Objective (FIO)

- Failure intensity ($\lambda$) is defined as failure per natural units (or time), e.g.
  - 3 alarms per 100 hours of operation.
  - 5 failures per 1000 print jobs, etc.

- Failure intensity of a system is the sum of failure intensities for all of the components of the system.

- For exponential distribution:

\[
Z_{\text{system}}(t) = \lambda_1 + \lambda_2 + \cdots + \lambda_n = \sum_{i=1}^{n} \lambda_i
\]
How to Set FIO?

- Setting FIO in terms of system reliability \((R)\) or availability \((A)\):

\[
\lambda = \frac{-\ln R}{t} \quad \text{or} \quad \hat{\lambda} \approx \frac{(1-R)}{t} \quad \text{for} \quad R \geq 0.95
\]

\[
\lambda = \frac{1-A}{t_m A}
\]

\(\lambda\) is failure intensity
\(R\) is reliability
\(t\) is natural unit (time, etc.)
\(t_m\) is downtime per failure

2. Operational Mode / Profile

- **Operational mode** is a distinct pattern of system use and/or set of environmental conditions that may need separate testing due to likelihood of stimulating different failures.

- Example:
  - Time (time of year, day of week, time of day)
  - Different user types (customer or user)
  - Users experiences (novice or expert)

- The same operation may appear in different operational mode with different probabilities.

- **System operational profile** must be developed for all of its important operational modes.
Operational Profile

- **Operational Profile** is the set of operations (operation names and frequencies) and their probabilities of occurrences.
- There are four principal steps in developing an operational profile:
  - Identify the operation initiators (i.e., user types, external systems, and the system itself)
  - List the operations invoked by each initiator
  - Determine the occurrence rates
  - Determine the occurrence probabilities by dividing the occurrence rates by the total occurrence rate

SRE: 3. Prepare for Test

- The Prepare for Test activity uses the operational profiles to prepare test cases and test procedures.
- Test cases are allocated in accordance with the operational profile.
- A **Test Case** is a partial specification of a run through the naming of its direct input variables and their values.
- Test case is specified with its direct input variables.
Types of Test

- **Certification Test**: Accept or reject (binary decision) an acquired component for a given target failure intensity.
- **Feature Test**: A single execution of an operation with interaction between operations minimized. Sometimes called *Unit test*.
- **Load Test**: Testing with field use data and accounting for interactions.
- **Regression Test**: Feature tests after every build involving significant change, i.e., check whether a bug fix worked.

4. Execute Test

- Allocate test time among the associated systems and types of test (feature, load, regression, etc.).
- Invoke the test cases at random times, choosing operations randomly in accordance with the operational profile.
- Identify failures, along with when they occur.
- Use this information to decide whether release the software or not.
Release Criteria

Consider releasing the product when:

1. All acquired components pass certification test
2. Test terminated satisfactorily for all the product variations and components with the $\lambda/\lambda_F$ ratios for these variations don’t appreciably exceed 0.5 (Confidence factor)

More Info

- Download tools:
  IEEE Software Reliability Engineering Working Group (SREWG)
  http://www.srewg.org/Tools/

- Info on other tools:
  Center for Reliability Engineering at the University of Maryland
  http://www.enre.umd.edu/tool.htm
The Goal

- It is important to be able to assess the status of quality of a software product, predict the number of failures or estimate the mean time to the next failure during the development.
- Such tasks are the target of the quality and reliability management models.
**Trend Analysis: Questions ...**

- Is the system reliability increasing, decreasing or stable?
- Which reliability growth model fits best the gathered failure data?
- Can the same reliability model be used in all the cases of reliability growth, decrease and stable?

**And The Answers ...**

- Reliability trends can be analyzed by “trend tests”.
- Trend tests can be used to help determine whether the system undergoes reliability growth, decrease or stable reliability.
- Trend analysis also helps select appropriate reliability model for each phase.
Failure Data for Trend Analysis

- The trend tests work with two forms of failure data.
  - Inter-failure times (failure in intervals)
  - Number of failures per unit of time (failure intensity)
- The trend can be analyzed using inter-failure times data or failure intensity data.

Types of Trend Tests

- Two types of trend tests:
  - **Graphical tests** and **Analytical tests**
  - **Graphical tests** consist of plotting some observed failure data such as the inter-failure times or the number of failures per unit of time versus time in order to visually obtain the trend displayed by the data.
  - **Analytical tests** are based on statistical analysis of the raw failure data to derive trend factors.
- Two trend tests are commonly carried:
  - Arithmetical mean test
  - Laplace tests
1. Inter-failure Times Data

- The arithmetical mean of the inter-failure times (failure in intervals) consists of calculating arithmetical mean $\tau(i)$ of the observed inter-failure times $\theta_j$.

$$\tau(i) = \frac{1}{i} \sum_{j=1}^{i} \theta_j$$

- An increasing series of $\tau(i)$ indicates reliability growth and a decreasing series suggests decrease of reliability.

1. Inter-failure Times Data

- For $N(T)$ as the cumulative number of failures over the time period $[0, T]$, the Laplace factor $u(T)$ is derived:

$$u(i) = \frac{1}{i-1} \sum_{n=1}^{i-1} \sum_{j=1}^{n} \theta_j - \frac{\sum_{j=1}^{i} \theta_j}{2} \sqrt{\frac{1}{12(i-1)}}$$

For the case that $T$ is equal to the time of occurrence of failure $i$. 
1. Inter-failure Times Data

- Negative values of the Laplace factor $u(i)$ indicate a decreasing failure intensity, i.e., reliability growth.
- Positive values of the Laplace factor $u(i)$ indicate an increasing failure intensity, i.e., reliability decrease.
- Values between $-2$ and $+2$ indicate stable reliability.

Midpoint of the observation interval: $T/2$

Statistical center of data:

$$1 \over N(T) \sum_{n=1}^{N(T)} \sum_{j=1}^{n} \theta_j$$

- For the failure intensity decrease (i.e., reliability growth), the interfailure times $\theta_j$ tend to occur before the midpoint; hence the statistical center tends to be smaller than the mid-interval.
- For the failure intensity increase, the statistical center tends to be larger than the mid-interval.
2. Failure Intensity Data

For the time period \([0, T]\), divided into \(k\) units of equal length and for \(n(i)\) be the number of failures observed during the time interval \(i\), the Laplace factor \(u(k)\) is derived by:

\[
    u(k) = \frac{\sum_{i=1}^{k} (i-1)n(i) - \left(\frac{k-1}{2}\right)\sum_{i=1}^{k} n(i)\sqrt{\frac{k^2 - 1}{12}\sum_{i=1}^{k} n(i)}}{\sqrt{\frac{k^2 - 1}{12}\sum_{i=1}^{k} n(i)}}
\]

Negative values of the Laplace factor \(u(k)\) indicate a decreasing failure intensity, i.e., reliability growth.

Positive values of the Laplace factor \(u(k)\) indicate an increasing failure intensity, i.e., decrease of reliability.
Typical Plots /1

- Typical graphs for failure intensity \( n(k) \) and cumulative failure intensity \( N(k) \)

Typical Plots /2

- Typical plot for the Laplace factor \( u(k) \)
Typical Plots /3

- Typical plot for Laplace factor during various project phases

![Graph showing reliability growth and decrease of reliability phases](Figure from SRE Handbook)

Decisions Based on Testing

- What will be the outcome of certification testing?
  - Accept/reject an acquired component

- What will be the outcome of feature, regression and load testing?
  - Guiding software development process
  - Releasing the product
Certification Test Using Reliability Demonstration Chart

- A way of checking whether the Failure Intensity Objective ($\lambda_F$) is met or not.
- It is based on collecting failure data at time points.
- **Vertical axis:** failure number
- **Horizontal axis:** normalized failure data, i.e., 
  \[ \text{failure time} \times \lambda_F \]

How to Make a Reliability Demo Chart?

- **Boundary between reject and continue region:**
  \[ \ln \left( \frac{\beta}{1-\alpha} \right) - n \ln \gamma \]
  \[ \frac{1-\gamma}{1-\gamma} \]

- **Boundary between accept and continue region:**
  \[ \ln \left( \frac{1-\beta}{\alpha} \right) - n \ln \gamma \]
  \[ \frac{1-\gamma}{1-\gamma} \]
Parameters Involved /1

- **Discrimination ratio (γ):** Acceptable error in estimating failure intensity.
- **Consumer risk (β):** Probability that the developer is willing to accept of falsely saying the failure intensity objective is met when it is not.
- **Supplier risk (α):** Probability that the developer is willing to accept of falsely saying the failure intensity objective is not met when it is.

Parameters Involved /2

For $\alpha = 10\%$ and $\beta = 10\%$ and $\gamma = 2$

- There is 10% risk of wrongly accepting the software when its failure intensity objective is actually equal or greater than twice the failure intensity objective.
- There is 10% risk of wrongly rejecting the software when its failure intensity objective is actually equal or less than half the failure intensity objective.
**Example 1**

<table>
<thead>
<tr>
<th>Failure no.</th>
<th>Measure (million transactions)</th>
<th>Normalized Measure (MTTF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1875</td>
<td>0.75</td>
</tr>
<tr>
<td>2</td>
<td>0.3125</td>
<td>1.25</td>
</tr>
<tr>
<td>3</td>
<td>1.25</td>
<td>5</td>
</tr>
</tbody>
</table>

\[ \lambda_F = 4 \text{ failures/million transactions} \]
\[ \alpha = 10\% \]
\[ \beta = 10\% \]
\[ \gamma = 2 \]

Entering Accept region, therefore Certification test passed

**Example 2**

- We are going to buy a new laser printer. We want to have only one failure for 10,000 pages of output \( (\lambda_F = 1/10000 \text{ pages}) \)
- We observe that failures occur at 4,000 pages, 6,000 pages, 10,000 pages, 11,000 pages, 12,000 pages and 15,000 pages of output.
- What can we conclude about this printer?

Entering Reject region. Because of failing the certification test we will reject the printer.
Waterfall Development Characteristics

Waterfall Process

- Delays confirmation of critical risk resolution
- Measures progress by assessing work-products that are poor predictors of time-to-completion
- Delays and aggregates integration and testing
- Precludes early deployment
- Frequently results in major unplanned iterations

Key problem: Waterfall system development pushes risk forward in time

Conventional Software Process

Sequential activities

Requirements → Design → Code → Integration → Test

- Integration Begins
- Late Design Breakage
- Original Target Date
**Prevention Methods & Models**

- The most important principle in software engineering is **“do it right the first time.”**
- This may be done via:
  - Preventing errors from being injected into the development process.
  - Improving the front end of the development process to remove as many failures as early as possible; e.g., for waterfall development process, rigorous design reviews and code inspections are needed.
  - In the integration phase, unit test or pre-integration test (the development phase prior to system integration) is the last chance to catch the bugs before they being integrated into the system.

**Bidirectional Quality Strategy**

- The Rayleigh model is an overall model for quality management in successive development versions of software.
- If the error injection rate is reduced, the area under Rayleigh curve becomes smaller.
- The earlier the peak comes, the smaller the area under the curve.
Bidirectional Quality Strategy

- Interpretation of “do it right the first time”: shift the peak of the Rayleigh model to the left while lowering its peak.

Defect Rate vs. Inspection Effort

- Is failure count (defect rate) a good metrics for software quality?
- The more bugs found and fixed doesn’t necessarily imply better quality because the fault injection rate may be different.
- From the defect removal data how do we assess the error injection rate or defect removal?
- Using inspection effort vs. detect rate chart
Defect Rate vs. Inspection Effort

- **Best case scenario** high effort/low defect rate: an indication that the design/code was cleaner before inspections, and yet the team spent enough effort in design review/code inspection that better quality was ensured.

- **Good/not bad scenario** high effort/high defect rate: error injection may be high, but higher effort spent is a positive sign and that may be why more defects were removed. If effort is significantly higher than the target, this situation may be a good scenario.

- **Unsure scenario** low effort/low defect rate: not sure whether the design and code were better. Less time in inspection was needed or inspections were hastily done, hence finding fewer defects.

- **Worst case scenario** low effort/high defect rate: an indication of high error injection but inspections were not rigorous enough. Chances are more defects remained in the design or code at the exit of the inspection process.

<table>
<thead>
<tr>
<th>Inspection Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Higher</td>
</tr>
<tr>
<td>Good/Not Bad</td>
</tr>
<tr>
<td>Lower</td>
</tr>
</tbody>
</table>

Reliability Models: Variations

- **Evolving programs**
  - Dealing with small increments of program size and/or stepwise evolution

- **Unreported failures**
  - How unreported failures affect failure intensity estimation?
Evolving Programs /1

- Reliability models used in estimating failure intensity ($\lambda$) assume that the executed program is stable (not changing, except for those changes that result from failure correction).

- Programs can evolve due to:
  - Requirements changes
  - Integration of new parts
  - Necessity to adapt to changing hardware and software environment
  - Necessity for system performance improvement
  - Evolution as part of the development process

Evolving Programs /2

Possible scenarios:

- Ignoring changes for programs evolving slowly and in small size increments
- Ignoring old data on the old phase and base estimates on the new phase
- Stepwise evolution
  - Applying changes by components
The effects of evolution can be ignored for programs evolving slowly and in small size increments (increments of less than 5% of total code per week).

**Advantages:** No extra testing and data collection is required.

**Disadvantages:** Estimates of model parameters and $\lambda/\lambda_F$ ratios will lag the true situation and will have error, but the range of errors may be acceptable.

In component by component approach, add the component failure intensities separately to obtain the system failure intensity at each stage.

In operation group approach, add the weighted failure intensities of operation groups to obtain the system failure intensity at each stage.

Stepwise evolution approaches generally work best when having a small number of large changes, each resulting from the addition of an independent element.
Evolving Programs /5

- Advantages and disadvantages of stepwise evolution approaches:

- **Advantages:** System operational profile can be applied directly.

- **Disadvantages:**
  - The extra data collection required because of the multiple elements
  - Greater estimation error (or later achievement of a specified degree of accuracy) due to smaller sample sizes