An overview of the 2016 IEEE 1633 Recommended Practices for Software Reliability

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

© SoftRel, LLC 2019. This presentation may not be reprinted in whole or part without written permission from amneufelder@softrel.com
• Software reliability timing
• Introduction and motivation to write this document
• Common myths
1962 First recorded system failure due to software

1968 The term “software reliability” is invented.

Martin Trachtenberg notices the “bell curve”

1962 First recorded system failure due to software

Many software reliability growth estimation models developed. Limitation—can’t be used until late in testing.

First predictive model developed by USAF Rome Air Development Center with SAIC and Research Triangle Park—Limitations—model only useful for aircraft and never updated after 1992.

Larry Putnam/QSM quantifies the bell curve used for both scheduling and staffing

SoftRel, LLC models based on RL model. Can be used on any system. Updated every 4 years.

IEEE 1633
Introduction and motivation

• Software reliability engineering
  • Has existed for over 50 years.
  • Fundamental prerequisite for virtually all modern systems
  • Plenty of theory generated over last several decades, but…
  • Practical guidance on how to apply these models has lagged significantly

• Diverse set of stakeholders requires pragmatic guidance and tools to apply software reliability models to assess real software or firmware projects during each stage of the software development lifecycle
  • Reliability engineers may lack software development experience
  • Software engineers may be unfamiliar with methods to predict software reliability
  • Both may have challenges acquiring data needed for the analyses
Myths

• Myth – *The document can’t be used until the software is already developed*

• Fact – The 2008 IEEE 1633 had very little guidance for how to predict software reliability before it’s in a testable state. However, the 2016 edition does cover this.

• 255 pages discuss how to predict software reliability before the code is even written. The other 6 pages discuss how to measure it once it’s testable.

• Reason for myth –
  • The document predicts software reliability BEFORE it is developed but the prediction is *forecasted into operational use*.
  • In other words, you want to predict how the software will behave in operation and you want to do that prediction before the code is written
  • The IEEE 1633 provides multiple means to do that
Myths

• Myth: Our systems have problems with many different vendors, different teams with different skill sets, reworked code, and systems that are continually updated. So, the IEEE 1633 practices won’t work for us.

• Fact: This is a common excuse. Virtually every software intensive system on earth has the same challenges.
  • Some industries (medical, energy) have all of these challenges plus additional challenges involving federal regulations, offshore programmers, etc.
  • The IEEE 1633 models assume that these challenges are at least applicable and are considered in the prediction models themselves
Abstract

- Newly revised IEEE 1633 Recommended Practice for Software Reliability provides actionable step by step procedures for employing software reliability models and analyses
  - During any phase of software or firmware development
  - With any software lifecycle model for any industry or application type.
- Includes
  - Easy to use models for predicting software reliability early in development and during test and operation.
  - Methods to analyze software failure modes and include software in a system fault tree analysis.
  - Ability to assess the reliability of COTS, FOSS, and contractor or subcontractor delivered software.
- This presentation will cover the key features of the IEEE 1633 Recommended Practices for software reliability.
Acknowledgement of IEEE 1633 Working Group members

- Lance Fiondella - UMass
- Peter Lakey - consultant
- Robert Binder – SEI
- Michael Siok – Lockheed Martin
- Ming Li - NRC
- Ying Shi - NASA
- Nematollah Bidokhti - thinkDFR
- Thierry Wandji – US Navy
- Michael Grottke - FAU
- Andy Long - OSD
- George Stark - IBM
- Allen Nikora - NASA
- Bakul Banerjee – retired IEEE
- Debra Greenhalgh Lubas – US Navy
- Mark Sims – US Army
- Rajesh Murthy - consultant
- Willie Fitzpatrick – US Army
- Mark Ofori-kyei – General Dynamics
- Sonya Davis – General Dynamics
- Burdette Joyner – Northrup Grumman
- Marty Shooman –retired NASA
- Andrew Mack
- Loren Garroway – Northrup Grumman
- Kevin Mattos– US Navy
- Kevin Frye - US Navy
- Claire Jones - Boeing
- Robert Raygan - OSD
- Mary Ann DeCicco – General Dynamics
- Shane Smith - OSD
- Franklin Marotta – US Army
- David Bernreuther – US Army
- Martin Wayne – US Army
- Nathan Herbert – US Army
- Richard E Gibbs III - Boeing
- Harry White - Harmonic
- Jacob Axman – US Navy
- Ahlia T. Kitwana - Harris
- Yuan Wei
- Darwin Heiser – General Dynamics
- Brian McQuillan – General Dynamics
- Kishor Trivedi – Duke University

Chair: Ann Marie Neufelder, SoftRel, LLC
Vice Chair: Martha Wetherholt, NASA
Secretary: Debra Haehn, Philips
IEEE Standards Association Chair: Louis Gullo, Raytheon Missile Systems Division
# IEEE 1633 Table of contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2,3</td>
<td>Overview, definitions and acronyms</td>
</tr>
<tr>
<td>4</td>
<td>Tailoring guidance</td>
</tr>
<tr>
<td>5</td>
<td>“Actionable” Procedures with Checklists and Examples</td>
</tr>
<tr>
<td>5.1</td>
<td>Planning for software reliability – preliminary risk assessment</td>
</tr>
<tr>
<td>5.2</td>
<td>Develop a failure modes mode</td>
</tr>
<tr>
<td>5.3</td>
<td>Apply SRE during development</td>
</tr>
<tr>
<td>5.4</td>
<td>Apply SRE during testing</td>
</tr>
<tr>
<td>5.5</td>
<td>Support Release decision</td>
</tr>
<tr>
<td>5.6</td>
<td>Apply SRE in operation</td>
</tr>
<tr>
<td>Annex A</td>
<td>Supporting information on the software FMEA</td>
</tr>
<tr>
<td>Annex B</td>
<td>Detailed procedures on predicting size and supporting information for the predictive models</td>
</tr>
<tr>
<td>Annex C</td>
<td>Supporting information for the software reliability growth models</td>
</tr>
<tr>
<td>Annex D</td>
<td>Estimated cost of SRE</td>
</tr>
<tr>
<td>Annex E</td>
<td>SRE tools</td>
</tr>
<tr>
<td>Annex F</td>
<td>Examples</td>
</tr>
</tbody>
</table>
Clause 4 SRE
Tailoring

• The document is geared towards 4 different roles, any industry and any type of software or firmware.
  • Reliability engineer
  • Software management
  • Software Quality Assurance
  • Acquisitions (people who are acquiring software)

• Hence, section 4 provides guidance for tailoring the document.
  • By role – recommended sections if you are a reliability engineer, software QA, software manager or acquisitions.
  • By life cycle - How to apply the document if you have an incremental or agile life cycle model.
  • By criticality – Some SR tasks are essential, some are typical and some are project specific.
Typical tasks by role

- **Reliability engineering**
  - Predicted reliability for each software LRU
  - Recommended system level modifications to reach objective
  - System requirements
  - System reliability block diagram
  - Determination of whether acceptance criteria has been met for system including software
  - Estimated remaining defects and test hours required to reach objective

- **Software reliability data and models**
  - Recommended development changes to reach objective
  - Sensitivity of each SW LRU
  - Predicted staffing required to support software

- **Software management**
  - List of LRUs
  - Size
  - Schedule
  - Development methods
  - Failure modes
  - Determination as to whether acceptance criteria has been met for software LRUs

- **Acquisitions**
  - System reliability requirements
  - Statement of work
  - Software reliability program plan and results as defined by contract
  - Date and severity of each failure report found during testing
  - Determination of whether acceptance criteria has been met for software LRUs
Tasks by lifecycle

- The below illustrates the SRE tasks for an evolutionary lifecycle.
- The document also covers incremental and waterfall.
Tasks by criticality of project

Each task is identified as essential, typical or project specific. The below is one of the flow diagrams which illustrates the criticality of each task.

Apply Software Reliability during Testing

5.4.3 Increase test effectiveness via fault insertion

5.4.2 Measure test coverage

5.4.1 Develop a Reliability Test Suite

5.4.5 Select and use reliability growth models

5.4.4 Collect failure and defect data

5.4.6 Apply SRE metrics

5.4.7 Determine accuracy of predictions and RG models

5.5 Support Release decision
5.1 Planning for software reliability.
5.2 Develop a failure modes mode
5.3 Apply SRE during development
5.4 Apply SRE during testing
5.5 Support Release decision
5.6 Apply SRE in operation
Clause 5.1
Planning for software reliability

Planning for SRE is essential for a successful and cost effective SRE program

5.1.1 Characterize the software system
5.1.2 Define failures and criticality
5.1.3 Perform a reliability risk assessment
5.1.4 Assess the data collection system
5.1.5 Review available tools needed for software reliability
5.1.6 Develop a software reliability plan
# Tailoring of section 5.1

<table>
<thead>
<tr>
<th>Clause</th>
<th>Topic</th>
<th>Tailoring</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1.1</td>
<td>Characterize the software system – Identify the software and firmware LRUs, Bill of Materials, operational profile and impact of software on the system design.</td>
<td>E</td>
<td>RE L C SQA X SM R</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Define failures and criticality – Every product and system has a unique definition of failure criticality which should be defined in advance so that the software is designed, tested and measured correctly.</td>
<td>E</td>
<td>C C C C L or R</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Perform a reliability risk assessment – safety, security, vulnerability, product maturity, project and schedule risks can have an impact on the required reliability.</td>
<td>T</td>
<td>L X X or L L or R</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Assess the data collection system – The analyses and models will require software defect, product and process related data.</td>
<td>T</td>
<td>X L X R</td>
</tr>
<tr>
<td>5.1.5</td>
<td>Review available tools needed for software reliability</td>
<td>T</td>
<td>L X X R</td>
</tr>
<tr>
<td>5.1.6</td>
<td>Develop a Software Reliability Plan – Identify what software reliability activities are relevant for a particular software release.</td>
<td>E</td>
<td>L X X X R</td>
</tr>
</tbody>
</table>

**Tailoring**
- E – Essential
- T – Typical
- PS – Project Specific

**Role**
- RE – Reliability Engineer or Systems Engineer
- SQA – Software Quality Assurance or Testing
- SM – Software Management
- ACQ – Acquisitions personnel who are purchasing or acquiring software from other organization

**Activity for each role**
- L – This person leads the activity
- X – This person provides inputs to the activity
- C – This person consumes the results of the activity
- R – This person reviews the results of the activity
5.1.1 Characterize the software system

- Today modern systems rarely have only 1 software component. They often have several dozens including:
  - Customer software applications
  - COTS (Commercial Off The Shelf software)
  - FOSS (Free and Open Source Software)
  - Firmware
  - Middleware
  - Glueware
  - Operating System(s)

- The predictions should be performed on each of the software executables, DLLs or applications.
5.1.2 Define failures and criticality

• In order to accurately measure software failures, they have to be defined on a *project specific* basis. A Failure Definition and Scoring Criteria should be prepared prior to developing any specifications.

• Software failures should be defined at the subsystem or system level.

• Good Example: “Missile misses launch window” is defined as serious.
  • Notice that either the software or hardware can cause this event and the description of the event is not subjective

• Bad Example: “Software crashes”.
  • A software crash at the wrong time in the wrong function can be catastrophic while a software crash in a non-critical function during non-peak times could be minor. The failure should pertain to what the “system” does or doesn’t do.
5.1.3 Perform a Risk Assessment

- The risks for software reliability include safety, security, vulnerability, product maturity, schedule, inherent product risks.

- One of the schedule risks is "defect pileup". This is when the technical depth or backlog of defects is increasing from release to release. A project can fail because of defect pileup from previous releases.

![Graph showing total defects predicted from releases 1 to 5 for each month.](image)
5.1.4 Assess the data collection system

• Prior to using any of the software reliability models, one must know what data is available

• Example – Persons in acquisitions don’t always have direct access to defect data in testing while persons in development may not have access to field failure data

• Older SRE guidebooks assumed that the user has perfect data (i.e. number of seconds between software failures)

• This document assumes that the data is imperfect
  • The exact time between failures is often not known
  • While the data will resemble certain distributions, it isn’t likely to fit any distribution perfected
5.1.5 Review available tools needed for SRE

• There isn’t one tool that covers all of the SRE tasks

• Tools may be needed for
  • Collecting, filtering, organizing failure data
  • Predicting and measuring the size of the software
  • Predicting defects and failure rate
  • Forecasting the SWR growth during testing and operation
  • Failure modes analysis

• This section identifies the features needed in the tools

• The annex provides a listing of each
5.1.6 Develop the SR Plan

- The SR plan ensures that SRE tasks are determined at the start of the project and monitored throughout the project.
- The selected SRE tasks are included in the plan.
- The plan can stand alone or be part of the reliability plan or the software development plan.
Clause 5.2
Develop Failure Modes Analysis

Understanding the root causes and effects of software failures is essential for risk reduction

5.2.1 Perform defect root cause analysis
5.2.2 Perform software failures modes effects analysis
5.2.3. Include software in a system fault tree analysis
## Tailoring of 5.2

<table>
<thead>
<tr>
<th>Clause</th>
<th>Topic</th>
<th>Tailoring</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.1</td>
<td>Perform Software Defect Root Cause Analysis – Identify the most common type of software defect so that practices can be put in place to identify and remove those categories of defects prior to deployment.</td>
<td>PS</td>
<td>RE: C</td>
</tr>
<tr>
<td></td>
<td>5.2.2</td>
<td>Perform Software Failure Modes Effects Analysis (SFMEA) – This bottom up analysis is similar to a hardware (HW) FMEA except for the software failure modes and root causes and the software viewpoints.</td>
<td>PS</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Include Software in the System Fault Tree Analysis – The FTA is useful for identifying single and multiple point failures from a top down viewpoint.</td>
<td>PS</td>
<td>RE: L</td>
</tr>
</tbody>
</table>

**Tailoring**
- E – Essential
- T – Typical
- PS – Project Specific

**Role**
- RE – Reliability Engineer or Systems Engineer
- SQA – Software Quality Assurance or Testing
- SM – Software Management
- ACQ – Acquisitions personnel who are purchasing or acquiring software from other organization

**Activity for each role**
- L – This person leads the activity
- X – This person provides inputs to the activity
- C – This person consumes the results of the activity
- R – This person reviews the results of the activity
5.2.1 Software defect root cause analysis

• The RCA ensures that any SRE improvement efforts address the right types of defects.
  • *Example, if most of the defects are introduced in the design phase, you don’t want to put all of your SRE effort into improving coding practices.*

• Software reliability assessment identified certain gaps and strengths which can lead to a certain “volume” of defects

• A root cause analysis can confirm the “types” of defects
  • Faulty requirements?
  • Faulty design?
  • Faulty implementation?
  • Faulty interfaces?
  • Faulty changes or corrective actions?
  • Faulty source and version control?
  • Faulty installation package?
  • Faulty usability?

• These can and will be unique for each configuration item *even if they have the same development processes*
Example of a root cause analysis

Defects are introduced because of either bad requirements, bad design, bad coding practices or bad change control.

- **Requirements defect** – The “whats” are incorrect, ambiguous or incomplete.
- **Design defect** – The “whats” are correct but the “hows” are not. Logic, state, timing, exception handling are all design related.
- **Coding defect** – The “whats” and “hows” are correct but the software engineer did not implement one or more lines of code properly.
5.2.2 Perform SFMEA

- Software FMEAs are conducted similarly to hardware FMEAs

<table>
<thead>
<tr>
<th>Identify applicability</th>
<th>Identify riskiest software</th>
<th>Select viewpoints</th>
<th>Define scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify resources</td>
<td>Decide selection scheme</td>
<td>Set ground rules</td>
<td>Tailor the SFMEA</td>
</tr>
<tr>
<td>Identify local/ subsystem/ system failure effects</td>
<td>Identify compensating provisions</td>
<td>Identify corrective actions</td>
<td>Select template</td>
</tr>
<tr>
<td>Identify preventive measures</td>
<td>Identify severity and likelihood</td>
<td>Revise RPN</td>
<td>Mitigate</td>
</tr>
<tr>
<td>Identify consequences</td>
<td></td>
<td>Generate a Critical Items List (CIL)</td>
<td></td>
</tr>
</tbody>
</table>

- Analyze failure modes and root causes
5.2.2 Perform SFMEA

• There are 8 software viewpoints including
  • Requirements
  • Design
  • Interface design
  • Corrective actions/changes
  • Usability
  • Serviceability
  • Vulnerability
  • Production

• There are over 400 failure mode/root causes that are at least relevant to all software systems. Key failure modes are:

<table>
<thead>
<tr>
<th>Software failure mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faulty sequencing</td>
<td>Operations happen in the wrong order or state transitions are incorrect</td>
</tr>
<tr>
<td>Faulty timing</td>
<td>Operations start too early or too late or take too long. The right event happens in the right order but at the wrong time</td>
</tr>
<tr>
<td>Faulty data</td>
<td>The output of the system or software is incorrect. Outputs can also be behavioral.</td>
</tr>
<tr>
<td>Faulty functionality</td>
<td>The system does the wrong thing or does the right thing incorrectly</td>
</tr>
<tr>
<td>Faulty error detection</td>
<td>The software fails to detect an error in itself, hardware or environment</td>
</tr>
<tr>
<td>False alarms</td>
<td>The software is overly sensitive and detects an error which does not exist</td>
</tr>
</tbody>
</table>
5.2.3 Include software in a system fault tree

• A fault tree is a top down analysis

• Hence a software fault tree should be part of an overall system fault tree

• There may be several software fault trees on a system fault tree

• Examples of insufficient fault tree details on the leaves of the tree
  • “software failed”.
    • This is not specific enough to be useful. One can’t assess likelihood or severity or corrective action.

• Example of sufficient fault tree details on the leaves of the tree
  • “software provided incorrect results”
  • “software failed to complete task in the required time”
    • These are specific. One can determine likelihood, severity, corrective actions.
Difference between SFMEA and FTA

Failure Events

Visible to end users

SFMEA works up

Software Failure Modes
Requirements, interfaces, design, code

SFTA works down

Visible to software engineers
Clause 5.3 Apply sre during development

It’s important to predict the reliability .... before the software is developed and tested while there is time to perform tradeoffs

5.3.1. Determine/obtain system reliability objectives in terms of reliability, availability, MTBF
5.3.2. Perform software reliability assessment and prediction
5.3.3. Sanity check the early prediction
5.3.4. Merge the predictions into the over system prediction
5.3.5 Determine the total software reliability needed to reach the objective
5.3.6. Plan the reliability growth needed to reach the system objective
5.3.7. Perform a sensitivity analysis
5.3.8. Allocate the required objective to each software LRU
5.3.9. Employ software reliability metrics
Early prediction models versus reliability growth models

- Early prediction models are different from the reliability growth models discussed in 5.4. See the differences in the below table.

<table>
<thead>
<tr>
<th>Comparison attribute</th>
<th>Software reliability prediction models</th>
<th>Software reliability growth models</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Used during this phase of development</strong></td>
<td>Any phase as long as the scope of the software is defined</td>
<td>After software system testing commences</td>
</tr>
<tr>
<td><strong>Inputs</strong></td>
<td>• Indicators such as development practices, personnel, process, inherent risks</td>
<td>• Defects observed per time or defects observed during some internal.</td>
</tr>
<tr>
<td></td>
<td>• Size</td>
<td>• Testing hours per interval.</td>
</tr>
<tr>
<td></td>
<td>• Duty cycle</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Expected reliability growth</td>
<td></td>
</tr>
<tr>
<td><strong>Outputs</strong></td>
<td>Predicted defects, failure rate, reliability, availability</td>
<td></td>
</tr>
<tr>
<td><strong>Benefits</strong></td>
<td>• Allows for a sensitivity analysis before the software is developed</td>
<td>• Identifies when to stop testing</td>
</tr>
<tr>
<td></td>
<td>• Allows for determination of an allocation before the software is developed</td>
<td>• Identifies how many people are needed to reach an objective</td>
</tr>
<tr>
<td></td>
<td>• Identifies risks early</td>
<td>• Validates the prediction models</td>
</tr>
<tr>
<td></td>
<td>• Useful for assessing vendors</td>
<td></td>
</tr>
<tr>
<td><strong>Model framework</strong></td>
<td>Uses empirical data from historical projects in which the development practices and the delivered defects are known</td>
<td>Uses various statistical models to forecast based on the current trend</td>
</tr>
</tbody>
</table>
## Tailoring 5.3

<table>
<thead>
<tr>
<th>Clause</th>
<th>Topic</th>
<th>Tailoring</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3.1</td>
<td><strong>Identify/Obtain the initial system reliability objective</strong> – The required or desired MTBF, failure rate, availability, reliability for the entire system.</td>
<td>E</td>
<td>RE</td>
</tr>
<tr>
<td>5.3.2</td>
<td><strong>Perform a Software Reliability Assessment and Prediction</strong> – Predict the MTBF, failure rate, availability, reliability and confidence bounds for each software LRU.</td>
<td>E</td>
<td>RE</td>
</tr>
<tr>
<td>5.3.3</td>
<td><strong>Sanity Check the Prediction</strong> – Compare the prediction to established ranges for similar software LRUs</td>
<td>T</td>
<td>PS</td>
</tr>
<tr>
<td>5.3.4</td>
<td><strong>Merge the predictions into the overall system predictions</strong> – Various methods exist to combine the software and hardware predictions into one system prediction.</td>
<td>E</td>
<td>PS</td>
</tr>
</tbody>
</table>

### Tailoring
- E – Essential
- T – Typical
- PS – Project Specific

### Role
- **RE** – Reliability Engineer or Systems Engineer
- **SQA** – Software Quality Assurance or Testing
- **SM** – Software Management
- **ACQ** – Acquisitions personnel who are purchasing or acquiring software from other organization

### Activity for each role
- L – This person leads the activity
- X – This person provides inputs to the activity
- C – This person consumes the results of the activity
- R – This person reviews the results of the activity
## Tailoring 5.3 continued

<table>
<thead>
<tr>
<th>Clause</th>
<th>Topic</th>
<th>Tailoring</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3.5</td>
<td><strong>Determine an appropriate overall Software Reliability Requirement</strong> – Now that the system prediction is complete, revisit the initial reliability objective and modify as needed.</td>
<td>E</td>
<td>RE L SQA X SM X AC Q</td>
</tr>
<tr>
<td>5.3.6</td>
<td><strong>Plan the reliability growth</strong> – Compute the software predictions during and after a specific level of reliability growth testing.</td>
<td>E</td>
<td>RE L SQA X SM C AC R</td>
</tr>
<tr>
<td>5.3.7</td>
<td><strong>Perform a Sensitivity Analysis</strong> – Identify the practices, processes, techniques, risks that are most sensitive to the predictions. Perform tradeoffs.</td>
<td>PS</td>
<td>RE X SQA L SM C AC R</td>
</tr>
<tr>
<td>5.3.8</td>
<td><strong>Allocate the Required Reliability to the Software LRUs</strong> – The software and hardware components are allocated a portion of the system objective based on the predicted values for each LRU.</td>
<td>T</td>
<td>RE L SQA X SM C AC R</td>
</tr>
<tr>
<td>5.3.9</td>
<td><strong>Employ software reliability metrics for transition to testing</strong> – These metrics determine if the software is stable enough to be tested.</td>
<td>T</td>
<td>RE X SQA L SM C AC R</td>
</tr>
</tbody>
</table>

**Tailoring**
- E – Essential
- T – Typical
- PS – Project Specific

**Role**
- RE – Reliability Engineer or Systems Engineer
- SQA – Software Quality Assurance or Testing
- SM – Software Management
- ACQ – Acquisitions personnel who are purchasing or acquiring software from other organization

**Activity for each role**
- L – This person leads the activity
- X – This person provides inputs to the activity
- C – This person consumes the results of the activity
- R – This person reviews the results of the activity
5.3.1 Determine/obtain system reliability objectives

A system reliability objective is important in order to determine the software reliability objective.

This document provides 3 ways to determine a system reliability objective

1. When you have a system that has a precedent
2. When you have a system that has no precedent
3. When you have a mass produced system

The figure of merit may be MTBF, reliability, availability, etc depending on the type of system

<table>
<thead>
<tr>
<th>Figure of Merit</th>
<th>When it is most applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTBF/MTBCF/MTBSA/MTBEFF/failure rate</td>
<td>Most applications</td>
</tr>
<tr>
<td>Reliability (as a fraction between 0 and 1)</td>
<td>Mission type systems that have a defined mission time such as vehicles or aircraft.</td>
</tr>
<tr>
<td>Availability (as a fraction between 0 and 1)</td>
<td>Systems that run continually (satellites, networks, etc.)</td>
</tr>
</tbody>
</table>
Section 5.3.2 Perform software reliability assessment and prediction

- Since the 1970s most of the software reliability models are usable only during test or operation when it’s too late to do planning, tradeoffs, improvements.
- The models presented in this section can be used for the code is even written. The predictions are then merged with the hardware reliability predictions and compared to the system reliability objective.
If you can predict this fault profile you can predict all of the other reliability figures of merit.

The predictive models predict the fault profile first and then failure rate, MTBF, reliability and availability is predicted from that.
## Prediction models in order of simplicity

<table>
<thead>
<tr>
<th>Model</th>
<th>Number of inputs</th>
<th>Predicted output</th>
<th>Industry supported</th>
<th>Effort required to use the model</th>
<th>Relative accuracy</th>
<th>Year developed/ Last updated</th>
<th>Sub-Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>Industry tables [B12] [B76]</td>
<td>1</td>
<td>Defect density</td>
<td>Several</td>
<td>Quick</td>
<td>Varies</td>
<td>1992, 2015</td>
<td>6.2.1.2</td>
</tr>
<tr>
<td>CMMI® tables [B12]</td>
<td>1</td>
<td>Defect density</td>
<td>Any</td>
<td>Quick</td>
<td>Increases with CMMI® level</td>
<td>1997, 2012</td>
<td>6.2.1.3</td>
</tr>
<tr>
<td>Shortcut model [B56]</td>
<td>23</td>
<td>Defect density</td>
<td>Any</td>
<td>Moderate</td>
<td>Medium</td>
<td>1993, 2012</td>
<td>6.2.1.1</td>
</tr>
<tr>
<td>Metric based models [B28]</td>
<td>Varies</td>
<td>Defects</td>
<td>Any</td>
<td>Varies</td>
<td>Varies</td>
<td>NA</td>
<td>Annex B.2.2</td>
</tr>
<tr>
<td>Historical data</td>
<td>A minimum of 2</td>
<td>Defect density</td>
<td>Any</td>
<td>Moderate</td>
<td>High</td>
<td>NA</td>
<td>Annex B.2.3</td>
</tr>
<tr>
<td>Rayleigh model [B29]</td>
<td>3</td>
<td>Defects</td>
<td>Any</td>
<td>Moderate</td>
<td>Medium</td>
<td>NA</td>
<td>Annex B.2.4</td>
</tr>
<tr>
<td>Neufelder model [B68]</td>
<td></td>
<td>Defect density</td>
<td>Any</td>
<td>Detailed</td>
<td>Medium to high</td>
<td>2015</td>
<td>Annex B.2.6</td>
</tr>
</tbody>
</table>


Section 5.3.3 Sanity check the predictions

• Why - One of the biggest struggles with software reliability is that the reliability engineers rarely know whether their prediction is in the correct ballpark
• This document provides a table of actual software reliability values based on the size of the software and the number of installed sites
• Sanity checking the prediction can provide confidence in the overall prediction and inputs

<table>
<thead>
<tr>
<th>Size range in software people years (include only those writing code)</th>
<th>Worst case MTBF at initial deployment</th>
<th>Average MTBF at initial deployment</th>
<th>Best case MTBF at initial deployment</th>
<th>Worst case MTBF after 1 year of 24/7 operation with no new feature drops</th>
<th>Average MTBF after 1 year of 24/7 operation with no new feature drops</th>
<th>Best case MTBF after 1 year of 24/7 operation with no new feature drops</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 person project</td>
<td>Typical MTBF values for various sizes of software at initial deployment and after 1 year of reliability growth are shown in the document</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Very small. 2-9 software people years.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Small to medium. 10-49 software people years.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Medium. 50-99 software people years.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Large. 100-149 people years.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Very large. 200 or more people years.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Section 5.3.4 Merge the predictions into the overall predictions

• Once the software reliability is predicted for each Line Replaceable Unit, the predictions are merged with the hardware predictions taking into consideration...
  • Redundancy of hardware
  • Voting algorithms for software
• In general the software predictions will be in series with the hardware that the software support
• For most modern systems you would not expect to see one prediction for all software components in series with the rest of the system
  • i.e. when the media system in your car stops working, it shouldn’t affect your brake or transmission system
• The example below shows multiple software components in series with the hardware they support
5.3.5. Determine the total software reliability needed to reach the objective

- In section 5.3.1 the document discusses how to set an overall system reliability objective
- This section discusses how to allocate the objective to the software system

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocation by equal apportionment</td>
<td>Assumes for a series of “n” subsystems each are allocated the same reliability requirement goal.</td>
</tr>
<tr>
<td>Allocation by $[B32]$</td>
<td>Each component’s allocation is based on how much R&amp;D funding is required to develop that component</td>
</tr>
<tr>
<td>Past failure contribution</td>
<td>Assumes that the failure rate of a component is relatively equal to what it was on a past similar system</td>
</tr>
<tr>
<td>Allocation by achievable failure rate</td>
<td>Uses the predicted failure rate of each component to determine its allocation</td>
</tr>
<tr>
<td>ARINC Apportionment Techniques $[B33]$</td>
<td>Assumes series subsystems with constant failure rates, such that any subsystem failure causes system failure and that subsystem mission time is equal to system mission time.</td>
</tr>
<tr>
<td>Feasibility of Objectives Technique $[B34]$</td>
<td>Subsystem allocation factors are computed as a function of Delphi numerical ratings of system intricacy, state of the art, performance time, and environmental conditions.</td>
</tr>
<tr>
<td>Minimization of Effort Algorithm $[B17]$</td>
<td>Assumes a system comprised of n subsystems in series. The reliability of each subsystem is measured at the present stage of development, and apportions reliability such that greater reliability improvement is demanded of the lower reliability subsystems.</td>
</tr>
</tbody>
</table>
5.3.6 Plan the reliability growth needed to reach the system objective

• Reliability growth is often overestimated because
  • Software schedule delays mean less reliability growth
  • There will not be reliability growth of software as long as new features are added to it
  • The equipment needed for the growth is often unavailable
• Tradeoffs are often needed between reliability growth and new feature releases
• This section discusses how to
  • Forecast the reliability growth
  • Avoid unrealistic assumptions about reliability growth
5.3.7 Perform sensitivity analysis

- Once the predictions are computed, sanity checked, merged with the hardware and reliability growth is predicted then...
- It may become necessary to perform a sensitivity analysis if the predictions aren’t sufficient for reaching a required reliability objective
- The predictions are sensitive to:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Effective size (code that is new or modified)</td>
<td>Inverse Linear – Cutting effective size in half doubles MTBF.</td>
</tr>
<tr>
<td>Reliability growth</td>
<td>Exponential – Adding feature drops when the reliability is scheduled to grow can mean missing prediction by more than linear amount</td>
</tr>
<tr>
<td>Inherent product risks</td>
<td>Linear</td>
</tr>
<tr>
<td>Defect density – development practices, personnel, process, domain experience, etc.</td>
<td>Inverse Linear – Reducing defect density by half doubles MTBF</td>
</tr>
</tbody>
</table>
5.3.8. Allocate the required objective to each software LRU

• For management purposes the overall software allocations may need to be allocated to each software LRU. Some methods include:
  • Top down allocation
    • System allocation flows to subsystem which flows to software LRU
    • System allocation flows to hardware, software system which then flow to hardware, software LRUs
  • Bottom up – each LRU in the system gets an allocation proportional to it’s predicted failure rate

![Diagram showing top-down and bottom-up allocation methods]

Component 1
Reliability Requirement 0.9999
Component 1
Reliability Prediction 0.9900
Component 2
Reliability Requirement 0.9567
Component 2
Reliability Prediction 0.9500
Component n
Reliability Requirement 0.9789
Component n
Reliability Prediction 0.9999
System Reliability Requirement 0.9950
System Reliability Prediction 0.9800

Top-Down Allocation
Bottom-Up Derivation
5.3.9 Employ software reliability metrics

- In addition to predicting failure rate, MTBF, reliability, availability, etc. other metrics may be needed to determine the confidence in the predictions.
- It’s important that these metrics get put into place before the software and system testing begins.
- These metrics are required to put the reliability growth models shown next into perspective.

<table>
<thead>
<tr>
<th>Software Metrics</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Traceability</td>
<td>Degree to which the requirements have been met by the architecture, code and test cases</td>
</tr>
<tr>
<td>Structural coverage</td>
<td>Degree to which the lines of code, paths, and data have been tested</td>
</tr>
</tbody>
</table>
Clause 5.4 Apply SRE during testing

Once the software is integrated, the reliability growth is measured from actual failure data

5.4.1. Develop a reliability test suite
5.4.2. Increase test effectiveness via fault insertion
5.4.3. Measure test coverage
5.4.4. Collect failure and defect data
5.4.5 Select and use reliability growth models
5.4.6. Apply SRE metrics
5.4.7. Determine accuracy of the models
5.4.8. Revisit the defect root cause analysis
Section 5.4 Apply SRE during testing

- Software reliability growth models have existed since the 1970s
  - Many of them provide nearly identical results
  - SWRG models have been difficult to implement and understand due to poor guidance from academic community
  - Several models assume data which is not feasible to collect on non-academic large software systems
  - Every model has a different symbols for the same thing

This document provides
- Models that are feasible for real software systems
- Consolidation of models that provide similar results
- Consolidation of symbols across models
- Step by step instructions for how to select the best model(s) based on
  - The observed defect discovery trend (see next slide)
  - Inherent Defect Content
  - Effort required to use the model(s)
  - Availability of data required for the model(s)
  - How to apply them when you have an incremental life cycle
  - Test coverage methods which affect the accuracy of all SWRG models
## Tailoring 5.4

<table>
<thead>
<tr>
<th>Clause</th>
<th>Topic</th>
<th>Tailoring</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4.1</td>
<td><strong>Develop a Reliability Test Suite</strong></td>
<td>T</td>
<td>L</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>5.4.2</td>
<td><strong>Increase Test Effectiveness via Software fault insertion</strong> – This is when identified failure modes are instrumented such that the software identifies and recovers from them as required.</td>
<td>PS</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>5.4.3</td>
<td><strong>Measure test coverage</strong> – Measures the software coverage from both a structural and functional standpoint.</td>
<td>T</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>5.4.4</td>
<td><strong>Collect Fault and Failure Data</strong> – This data is collected during testing and is required for using the software reliability metrics and models. This Sub-Clause includes 2 very simple metrics which help determine the best reliability growth model.</td>
<td>E</td>
<td>L</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R</td>
</tr>
</tbody>
</table>

### Activity for each role

- **L** – This person leads the activity
- **X** – This person provides inputs to the activity
- **C** – This person consumes the results of the activity
- **R** – This person reviews the results of the activity

### Tailoring

- **E** – Essential
- **T** – Typical
- **PS** – Project Specific

### Role

- **RE** – Reliability Engineer or Systems Engineer
- **SQA** – Software Quality Assurance or Testing
- **SM** – Software Management
- **ACQ** – Acquisitions personnel who are purchasing or acquiring software from other organization
## Tailoring 5.4 continued

<table>
<thead>
<tr>
<th>Clause</th>
<th>Topic</th>
<th>Tailoring</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4.5</td>
<td><strong>Select Reliability Growth Models</strong> – The best models are those that fit the observed fault trends.</td>
<td>E</td>
<td>RE, SQA, SM, ACQ</td>
</tr>
<tr>
<td>5.4.6</td>
<td><strong>Apply software reliability metrics</strong> – These metrics are used in conjunction with the software reliability models.</td>
<td>T</td>
<td>RE, SQA, SM, ACQ</td>
</tr>
<tr>
<td>5.4.7</td>
<td><strong>Determine the accuracy of the prediction and reliability growth models</strong> – The prediction models in 5.3.2 and the SWRG models in 5.4.5 can be validated against the actual observed MTBF and failure rates to determine which model(s) are the most accurate for this particular software release.</td>
<td>E</td>
<td>RE, SQA, SM, ACQ</td>
</tr>
<tr>
<td>5.4.8</td>
<td><strong>Revisit the defect root cause analysis</strong> – The defect root cause analysis can be updated once testing is complete and used to improve the defect removal of the next software release.</td>
<td>PS</td>
<td>RE, SQA, SM, ACQ</td>
</tr>
</tbody>
</table>

### Tailoring
- **E** – Essential
- **T** – Typical
- **PS** – Project Specific

### Role
- **RE** – Reliability Engineer or Systems Engineer
- **SQA** – Software Quality Assurance or Testing
- **SM** – Software Management
- **ACQ** – Acquisitions personnel who are purchasing or acquiring software from other organization

### Activity for each role
- **L** – This person leads the activity
- **X** – This person provides inputs to the activity
- **C** – This person consumes the results of the activity
- **R** – This person reviews the results of the activity
## Section 5.4.1. Develop a reliability test suite

<table>
<thead>
<tr>
<th>Type of test</th>
<th>Inputs</th>
<th>Purpose/Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black box testing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operational profile testing</td>
<td>The Operational Profile (OP) from 5.1.1.3</td>
<td>Exercises the SUT according to how it will be used operationally.</td>
</tr>
<tr>
<td>Requirements based testing</td>
<td>The software requirements</td>
<td>Exercises the SUT to provide assurance that it satisfies its requirements as specified.</td>
</tr>
<tr>
<td>Model based testing</td>
<td>Test models may be derived from requirements and design documentation.</td>
<td>Exercises the SUT to evaluate compliance with a test model that represents critical aspects, including permitted/excluded operation sequences as well as input/output domains and combinations. This may reveal many kinds of defects, including dead or unreachable states.</td>
</tr>
<tr>
<td>Stress case testing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timing and performance</td>
<td>• Timing and scheduling diagrams which are usually found in the design documentation • Performance requirements • The effects of software on the system design from 5.1.1.4</td>
<td>Exercises the SUT to evaluate compliance with requirements for real-time deadlines, resource utilization.</td>
</tr>
<tr>
<td>Failure modes</td>
<td>• The identification of failure modes from 5.2.2</td>
<td>Exercises the conditions that are associated with the identified failure modes. This is the only test that verifies that software works properly in a degraded environment.</td>
</tr>
</tbody>
</table>

Contrary to popular belief testing only the requirements is rarely sufficient. Covering the above tests increases the confidence of the models shown in this section.
Section 5.4.2 Increase test coverage via fault insertion

- Software Fault injection (FI) is the process of systematically perturbing the program's execution according to a fault model, and observing the effects.

- The intent is to determine the error resiliency of the software program under faulty conditions so as to increase test coverage.

- Fault insertion Fault insertion infrastructures enable the users to
  - Inject faulty conditions into a wide range of applications
  - Control at runtime, the kinds of faulty conditions that should be injected and where to inject them in the program.
  - Trace the propagation of a faulty condition, after injecting it, and examining the state of selected program elements in relation to their fault-free state.
Section 5.4.3. Measure test coverage

More coverage means in the below areas means more confidence in the reliability growth estimates.

5.3.9 shows how to put the metrics in place. In this section, the metrics are used while testing the software.

Test coverage is not a one time measurement. It is measured regularly throughout testing and up until deployment.

Operational profile testing
Requirements based testing
Model based testing
Timing and performance
Failure modes
## Section 5.4.4 Collect failure data

<table>
<thead>
<tr>
<th>Day</th>
<th>f (faults per day)</th>
<th>x (usage hours per day)</th>
<th>n (cumulative faults)</th>
<th>t (cumulative usage hours)</th>
<th>n/t</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-28</td>
<td>1</td>
<td>8</td>
<td>1</td>
<td>8</td>
<td>0.125</td>
</tr>
<tr>
<td>5-3</td>
<td>1</td>
<td>16</td>
<td>2</td>
<td>24</td>
<td>0.083</td>
</tr>
<tr>
<td>5-5</td>
<td>3</td>
<td>16</td>
<td>5</td>
<td>40</td>
<td>0.125</td>
</tr>
<tr>
<td>5-6</td>
<td>2</td>
<td>8</td>
<td>7</td>
<td>48</td>
<td>0.146</td>
</tr>
<tr>
<td>5-10</td>
<td>2</td>
<td>16</td>
<td>9</td>
<td>64</td>
<td>0.141</td>
</tr>
<tr>
<td>5-12</td>
<td>2</td>
<td>16</td>
<td>11</td>
<td>80</td>
<td>0.138</td>
</tr>
<tr>
<td>5-13</td>
<td>1</td>
<td>8</td>
<td>12</td>
<td>88</td>
<td>0.136</td>
</tr>
<tr>
<td>5-14</td>
<td>2</td>
<td>64</td>
<td>14</td>
<td>152</td>
<td>0.092</td>
</tr>
<tr>
<td>5-26</td>
<td>9</td>
<td>8</td>
<td>23</td>
<td>160</td>
<td>0.144</td>
</tr>
<tr>
<td>5-27</td>
<td>1</td>
<td>8</td>
<td>24</td>
<td>168</td>
<td>0.143</td>
</tr>
<tr>
<td>6-1</td>
<td>1</td>
<td>24</td>
<td>25</td>
<td>192</td>
<td>0.13</td>
</tr>
<tr>
<td>6-2</td>
<td>1</td>
<td>8</td>
<td>30</td>
<td>200</td>
<td>0.15</td>
</tr>
<tr>
<td>6-3</td>
<td>2</td>
<td>8</td>
<td>32</td>
<td>208</td>
<td>0.154</td>
</tr>
<tr>
<td>6-4</td>
<td>2</td>
<td>8</td>
<td>34</td>
<td>216</td>
<td>0.157</td>
</tr>
<tr>
<td>6-5</td>
<td>2</td>
<td>8</td>
<td>36</td>
<td>224</td>
<td>0.161</td>
</tr>
<tr>
<td>6-6</td>
<td>4</td>
<td>8</td>
<td>40</td>
<td>232</td>
<td>0.172</td>
</tr>
<tr>
<td>6-7</td>
<td>1</td>
<td>8</td>
<td>41</td>
<td>240</td>
<td>0.171</td>
</tr>
<tr>
<td>6-17</td>
<td>2</td>
<td>64</td>
<td>43</td>
<td>304</td>
<td>0.141</td>
</tr>
<tr>
<td>6-23</td>
<td>1</td>
<td>40</td>
<td>44</td>
<td>344</td>
<td>0.128</td>
</tr>
<tr>
<td>6-25</td>
<td>1</td>
<td>16</td>
<td>45</td>
<td>360</td>
<td>0.125</td>
</tr>
<tr>
<td>7-6</td>
<td>1</td>
<td>56</td>
<td>46</td>
<td>436</td>
<td>0.111</td>
</tr>
<tr>
<td>7-23</td>
<td>2</td>
<td>40</td>
<td>48</td>
<td>456</td>
<td>0.105</td>
</tr>
<tr>
<td>7-26</td>
<td>1</td>
<td>8</td>
<td>49</td>
<td>464</td>
<td>0.106</td>
</tr>
<tr>
<td>7-30</td>
<td>1</td>
<td>32</td>
<td>50</td>
<td>496</td>
<td>0.101</td>
</tr>
<tr>
<td>8-3</td>
<td>1</td>
<td>16</td>
<td>51</td>
<td>512</td>
<td>0.1</td>
</tr>
<tr>
<td>8-5</td>
<td>2</td>
<td>8</td>
<td>53</td>
<td>520</td>
<td>0.102</td>
</tr>
<tr>
<td>8-11</td>
<td>3</td>
<td>32</td>
<td>56</td>
<td>552</td>
<td>0.101</td>
</tr>
<tr>
<td>8-24</td>
<td>2</td>
<td>64</td>
<td>58</td>
<td>616</td>
<td>0.094</td>
</tr>
<tr>
<td>8-30</td>
<td>1</td>
<td>24</td>
<td>59</td>
<td>640</td>
<td>0.092</td>
</tr>
</tbody>
</table>

Failure data like this is collected on a regular basis such as weekly because the reliability growth models require it.

The first three columns are collected from testing failure reports.

The last three columns are computed directly from the data collected.
• The fourth and sixth columns of the data on previous page is plotted as shown to the right.

• In the top example the fault rate is generally decreasing as new faults are observed.

• In the example in the middle, the early trend is increasing fault rate while later trend is decreasing.

• In the example on the bottom, the fault rate is steadily increasing.

• Notice that none of the data sets is “perfect”. The trends observed in the actual data are what determine the best reliability growth model. The 2008 version of the IEEE 1633 document assumed the fault rate was increasing and that the data is perfectly trending which is unrealistic.
5.4.5 Selecting the SWRG model

- Most important criteria is the current defect discovery trend
- A few models can be used when the discovery rate is increasing or peaking. Most can be used when decreasing or stabilizing.
- If the defect discovery is plotted first, the user will know which models can be used
The software reliability growth models

• Over the years, many overcomplicated models have been published by academia

• The models that can be used with only a spreadsheet or calculator are shown in the normative part (6.3) of the document with a real world (imperfect data) example

• All other models are shown in the annex
## Reliability growth models and when they can be used

<table>
<thead>
<tr>
<th>Model name</th>
<th>Inherent defect count</th>
<th>Effort required (1 low, 3 high)</th>
<th>Can be used when exact time of failure unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Increasing fault rate</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weibull [B46]</td>
<td>Finite/not fixed</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>Peak</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shooman Constant Defect Removal Rate Model[B61]</td>
<td>Finite/fixed</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Decreasing fault rate</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shooman Constant Defect Removal Rate Model[B61]</td>
<td>Finite/fixed</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Linearly Decreasing</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>General exponential models including:</td>
<td>Finite/fixed</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>• Goel-Okumoto [B47]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Musa Basic Model [B45]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Jelinski-Moranda [B48]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shooman Linearly Decreasing Model[B61]</td>
<td>Finite/fixed</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>Duane [B49]</td>
<td>Infinite</td>
<td>2</td>
<td>No</td>
</tr>
<tr>
<td><strong>Non-Linearily Decreasing</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Musa-Okumoto (logarithmic) [B50]</td>
<td>Infinite</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>Shooman Exponentially Decreasing Model[B62]</td>
<td>Finite/fixed</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>Log-logistic [B51]</td>
<td>Finite/fixed</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>Geometric [B52]</td>
<td>Infinite</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td><strong>Increasing and then decreasing</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yamada (Delayed)</td>
<td>Infinite</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>S-shaped [B53]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weibull [B46]</td>
<td>Finite/not fixed</td>
<td>3</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Clause 5.5 Support the Release Decision

The release decision should never be based on one metric alone. All of the SRE tasks shown in the document can/should be used to support a release decision.

5.5.1 Determine release stability
5.5.2 Forecast additional test duration
5.5.3 Forecast remaining defects and effort required to correct them
5.5.4 Perform a reliability demonstration test
5.5.5 Support release decision
Section 5.5 Support Release Decision

Once the development and testing is complete the SRE analyses, models and metrics can be used to determine the risk a making a release decision

- Decision is based on
  - Requirements and Operational Profile coverage
  - Stress test coverage
  - Code coverage
  - Adequate defect removal
  - Confidence in reliability estimates

- SRE Tasks performed prior to acceptance
  - Determine Release Stability – do the reliability estimates meet the objective?
  - Forecast additional test duration – If the objective hasn’t been met how many more test hours are required?
  - Forecast remaining defects and effort required to correct them – Will the forecasted defects pile up? Impact the next release?
  - Perform a Reliability Demonstration Test – Determine statistically whether the software meets the objective
### Tailoring 5.5

<table>
<thead>
<tr>
<th>Clause</th>
<th>Topic</th>
<th>Tailoring</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.5.2</td>
<td>Forecast additional test duration – If the required reliability is not met, determine how many more testing hours are required to meet it.</td>
<td>T</td>
<td>C</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Forecast remaining defects and effort required to correct them – If the required reliability is not met, determine how many more defects and staffing is required to meet it.</td>
<td>E</td>
<td>C</td>
</tr>
<tr>
<td>5.5.4</td>
<td>Perform a Reliability Demonstration Test – Statistically determine whether a specific reliability objective is met.</td>
<td>PS</td>
<td>L</td>
</tr>
</tbody>
</table>

**Tailoring**
- E – Essential
- T – Typical
- PS – Project Specific

**Role**
- RE – Reliability Engineer or Systems Engineer
- SQA – Software Quality Assurance or Testing
- SM – Software Management
- ACQ – Acquisitions personnel who are purchasing or acquiring software from other organization

**Activity for each role**
- L – This person leads the activity
- X – This person provides inputs to the activity
- C – This person consumes the results of the activity
- R – This person reviews the results of the activity
5.6.1. Employ SRE metrics to monitor software reliability
5.6.2. Compare operational and predicted reliability
5.6.3. Assess changes to previous characterizations or analyses
5.6.4. Archive operational data
## Tailoring 5.6

<table>
<thead>
<tr>
<th>Clause</th>
<th>Topic</th>
<th>Tailoring</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.6.1</td>
<td><strong>Employ software reliability metrics to monitor operational software reliability</strong> – It’s important to monitor operational reliability to know how the reliability in the field compares to the estimated and predicted.</td>
<td>T</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td>X</td>
</tr>
<tr>
<td>5.6.2</td>
<td><strong>Compare operational reliability to predicted reliability</strong></td>
<td>T</td>
<td>L</td>
</tr>
<tr>
<td>5.6.3</td>
<td><strong>Assess changes to previous characterizations or analyses</strong> – Adjust the reliability models inputs and assumptions accordingly to improve the next prediction</td>
<td>T</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td>X</td>
</tr>
<tr>
<td>5.6.4</td>
<td><strong>Archive operational data</strong> – Organize the data for use for future predictions</td>
<td>T</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Tailoring**
- E – Essential
- T – Typical
- PS – Project Specific

**Role**
- RE – Reliability Engineer or Systems Engineer
- SQA – Software Quality Assurance or Testing
- SM – Software Management
- ACQ – Acquisitions personnel who are purchasing or acquiring software from other organization

**Activity for each role**
- L – This person leads the activity
- X – This person provides inputs to the activity
- C – This person consumes the results of the activity
- R – This person reviews the results of the activity
Section 5.6 Apply SRE in Operations

- All of the SRE tasks are more accurate when operational data is collected and analyzed
  - Root causes
  - SFMEA
  - Fault trees
  - Predictions during development
  - Estimations during testing
  - Test coverage approach
  - Reliability objectives
Summary

- IEEE 1633 2016 puts forth recommended practices to apply qualitative software failure modes analyses and qualitative models
  - Improve product and ensure software or firmware delivered with required reliability

- IEEE 1633 2016 includes improved guidance
  - Offers increased value more accessible to a broader audience
    - Reliability engineers
    - Software quality engineers
    - Software managers
    - Acquisitions