MISSION READY SOFTWARE

SOFTREL LLC

THE SOFTWARE COMMON DEFECT ENUMERATION

HTTP://WWW.MISSIONREADYSOFTWARE.COM

ANN.NEUFELDER@MISSIONREADYSOFTWARE.COM

321-514-4659

Problem statement

- The Mitre Common Weakness Enumeration (CWE) (https://cwe.mitre.org/) has provided a means to identify vulnerabilities in an organized fashion so as to provide
  - Examples
  - How to identify the vulnerability
  - How to mitigate the vulnerability
- Software failure modes and root causes that cause mission failures haven't had the same level of organization, aren't updated on a regular basis and aren't as complete
- Software requirements and design reviews are often ineffective because there isn't a CDE to guide the review
- Software FMEAs are often ineffective because they don't cover the range of root causes

Early attempts at enumeration

- These authors have published enumerations for software failure modes
  - Beizer, Boris Software Testing Techniques. Van Nostrand Reinhold, 1984.
  - Kaner, Cem, Jack Falk and Hung Quoc Nguyen (1999). Testing Computer Software (Second Edition). John Wiley & Sons.
  - Binder, Robert V. (2000). Testing Object-Oriented Systems: Models, Patterns, and Tools. Addison-Wesley.
  - Vijayaraghavan, Giri and Cem Kaner. "Bugs in your shopping cart: A Taxonomy." http://www.testingeducation.org/articles/BISC_Final.pdf
  - Whittaker, James A. How to Break Software: A Practical Guide to Testing. Addison Wesley, 2003.
  - Hagar, Jon. Error/Fault Taxonomy Mind Map, 2021.
- These enumerations
  - Largely focus mostly on coding related mistakes
  - Some are specific to certain applications such as OO development or E-commerce
  - Haven't be kept up to date with new technologies and lessons learned

How the CDE originated

- Since 1993 Ann Marie Neufelder has analyzed the root causes of almost 1 million software failure events from public and private sources and categorized them by
  - Development artifact that introduced the fault by commission or omission
  - The type of software fault – functionality, timing, sequencing, state management, error handling, data definition, etc.
- The data shows that defects are often misclassified as "coding" related when in fact they originate in the specifications or design more than 50% of the time
  - Ex: If the software engineer didn't think about an aircraft crossing over the International Date Line and the code doesn't work when the date goes backwards, that's not a "coding" fault. A "coding fault" is when they write code to handle the international date line and it doesn't work correctly.
- A CDE provides for a full range of defect root causes – not just coding related

The Common Defect Enumeration is intended to minimize several of the 17 common mistakes that lead to ineffective software FMEAs

Organizational mistakes

- None of the software FMEA analysts have a background in software
- The analysis is not constructed by a cross functional team
- Conducting the SFMEA too late (most of these failure modes are too expensive to fix once the code is written)
- Conducting the SFMEA without the proper software deliverables such as the SRS, SDD, IRS, etc.
- Failing to track the failure modes and/or make any corrective actions to the requirements, design, code, use case, users manual as a result of the SFMEA
- Failing to tailor the software FMEA to the highest risk areas and most relevant failure modes

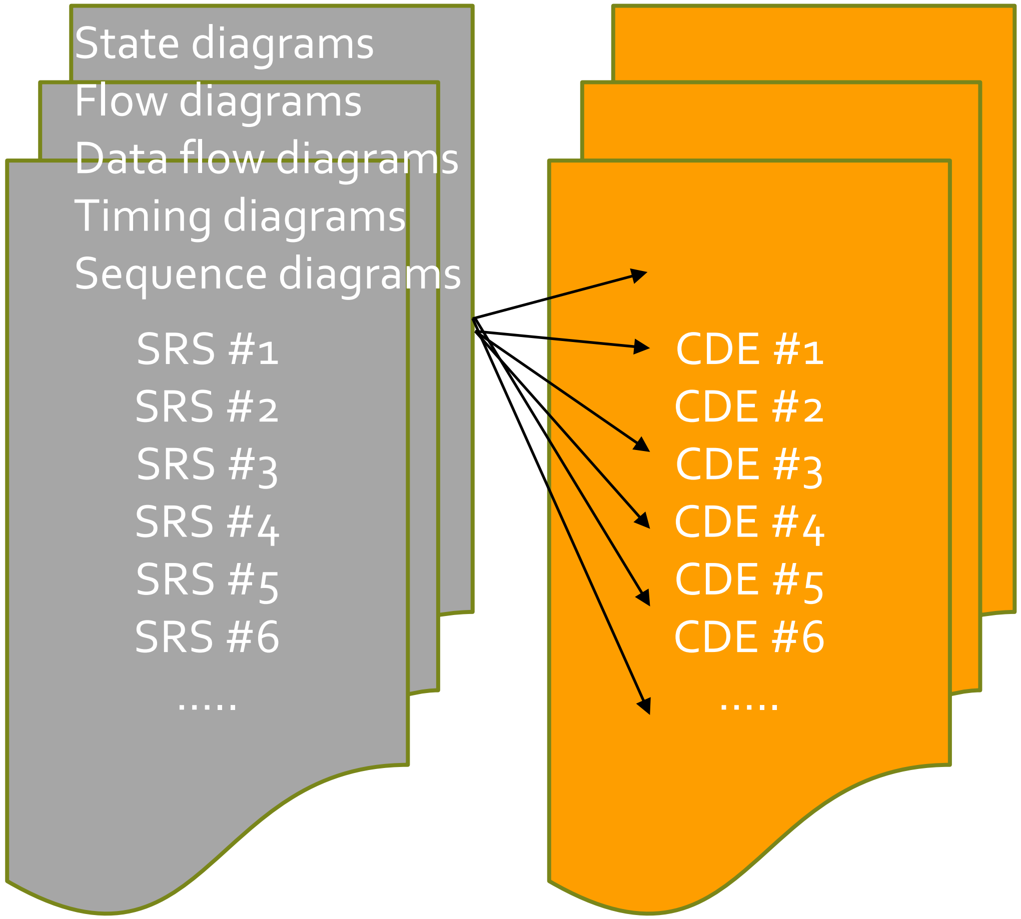
Faulty Assumptions

- **Assumption that all failures originate in a single line of code or specification**
- **Assumption that software works**
- **Assumption that software specifications are correct and complete**
- Assumption that all failure modes will be found and fixed in testing
- Assumption that all failure modes are impossible or negligible in severity

FMEA Execution mistakes

- **Focusing on total failure of the software - failing to consider small things that lead to big things going wrong**
- **Black box versus functional approach – analyze what the software does and not what it is**
- **Ignoring the 6 dimensions that lead to software failures - the system, the users who use the system, the battlefield environment, and the mission**
- **Conducting the SFMEA at too high (system requirements) or too low (lines of code) a level or architecture**
- **Mixing functional failure modes with process failure modes (i.e. fault timing means the software design not the software schedule)**
- Incorrectly assigning a failure rate or likelihood

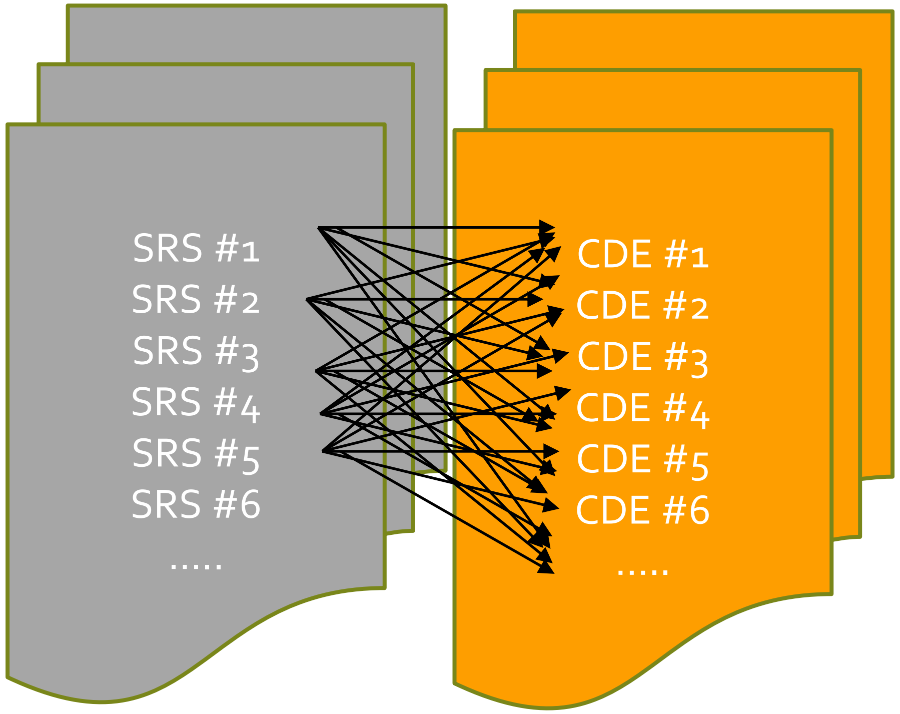
THIS IS EFFECTIVE



Analyze the ***collection*** of software requirements and designs flow against the set of CDEs

1. Prune the CDEs to remove things you don't have in the software (i.e. not all applications have machine learning)
2. Analyze the specifications and design as a whole package against the relevant CDEs

POPULAR BUT INEFFECTIVE

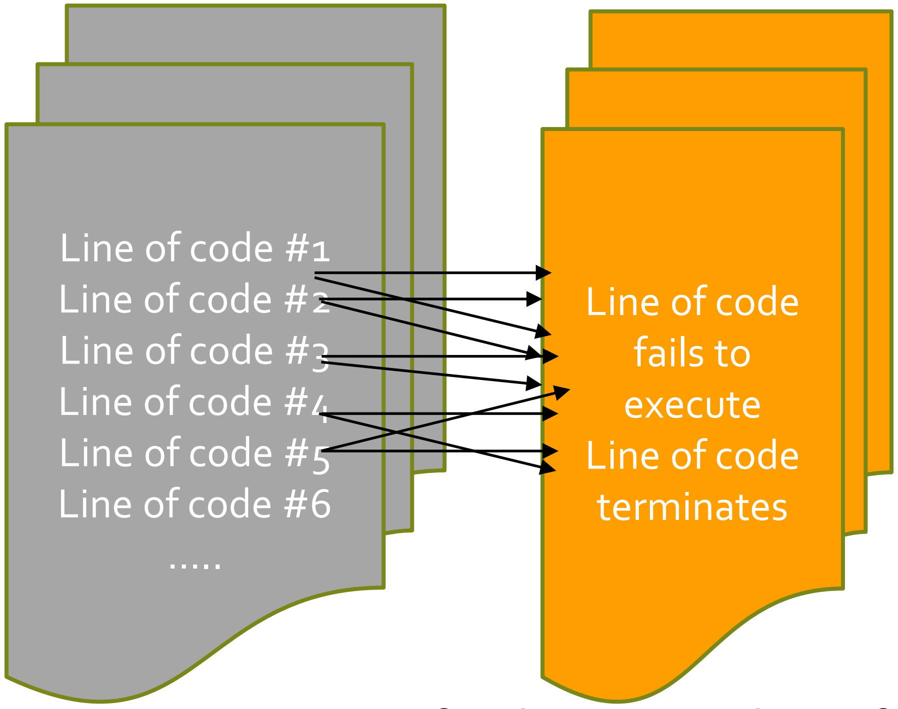


The analysts work through each SRS one at a time and analyze against statement each Common Defect Enumeration one at a time.

This is ineffective because:

1. Only a few of the CDEs pertain to a single statement
2. Majority of operational defects aren't caused by a single faulty statement (because these are individually verified prior to deployment)
3. Only the statements with magic numbers (i.e. timing or accuracy requirements) should be analyzed this way and only against the handful of relevant CDEs
4. It's more effective to use INCOSE requirements analyzers to identify poorly written specifications

POPULAR BUT INEFFECTIVE



Analysts work through each line of code one at a time and analyze against statement each CDE one at a time.
This is ineffective because:

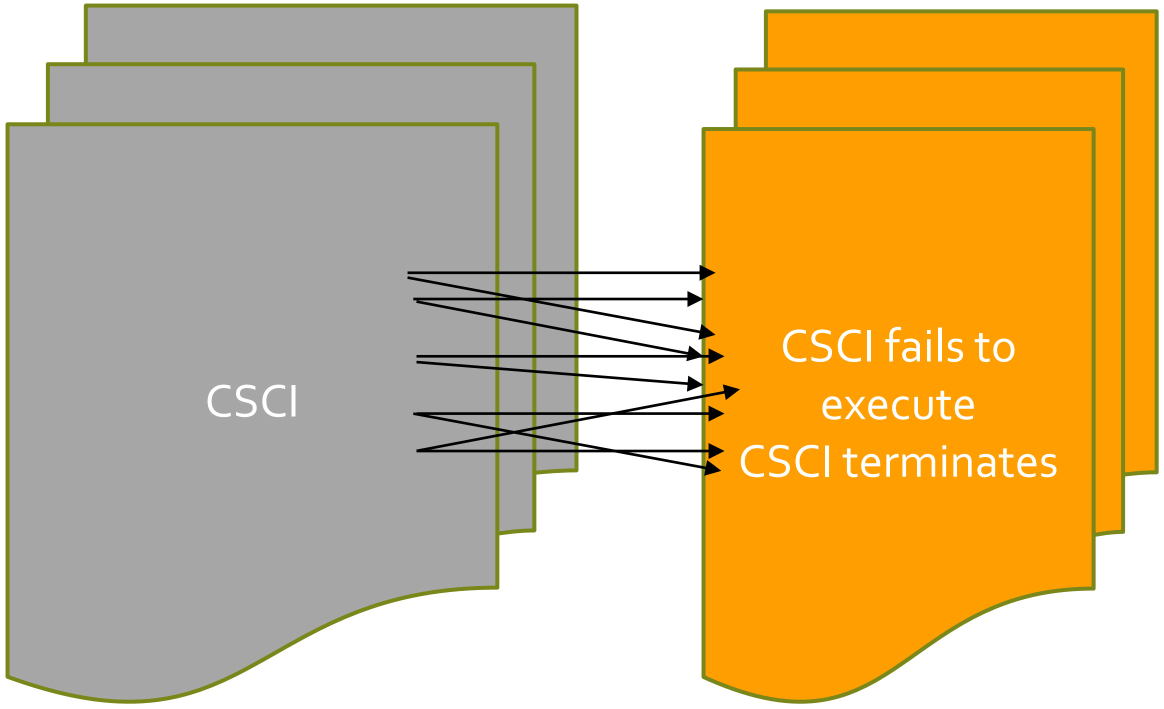
1. Very few failures are due to a **single** line of code
2. When a failure is due to a single line of code it is usually due to mistakes like these
    - Line of code executes the wrong command (i.e. has a compilable typo)
    - Line of code manipulates the wrong data
    - Line of code isn't written properly but still compiles

3

- Lines of code typically don't "fail to execute" unless there is a defect in another line of code
- If a line of code terminates execution it is often because there is missing fault handling or by faulty design

MISSION READY SOFTWARE
SOFTREL LLC

POPULAR BUT INEFFECTIVE



This is a very popular but very ineffective hardware centric approach

Software doesn't fail like hardware. It fails because of its design and specifications.

The above failure modes account for < 1% of all software failures

Goals for the Common Defect Enumeration

- Identify failure modes and root causes that have been tagged as root causes to many of the world's software failure events
- Identify failure modes by level of abstraction
  - Failure modes that effect the entire software system
  - Failure modes that effect a specific capability
  - Failure modes that effect a single software specification statement
  - Failure modes that effect the software interfaces
- Organize the CDE so as to provide
  - Examples
  - How to identify the failure mode
  - How to mitigate the failure mode
- Post the CDE published on a wiki type forum that can be updated with the latest technologies and lessons learned

The CDE format

- This enumeration is for classifying software failure modes and root causes.
- The common software defect enumeration is as follows:
- **<Architectural level> - <Failure Mode> - <Root cause #> - <Artifact> - <Artifact#>**

- Failure mode description: Self explanatory
- Discussion/Example of failure mode: Self Explanatory
- Description: The description is specific to the artifact level. The same root cause can originate in the requirements, design or code.
- Relevance: The types of systems or applications that see this failure mode the most
- Guidance: How highly this CDE is recommended for the software failure mode effects analysis

Architectural Level ID

The CDE enumeration begins with the point of view

- **TL - Top level** failure modes affect the entire software application. The root cause is not directly traceable to one capability or one specification. These are also called mission level failures. This viewpoint provides for the widest coverage of the software but the least level of detail.
- **CL - Capability level** failure modes and root causes affect one feature, use case, or capability. Example - IFF, launch, track, engage, etc.
- **SL - SRS level** failure modes and root causes are related to exactly one software requirements specification that is faulty.
- **IL - Interface level**. These failure modes and root causes originate in the interface design specification. In order to analyze these failure modes, the analysts will need to have an interface requirements specification or an interface design document.

Failure Mode and Root Cause ID

The CDE enumeration begins with the type of failure mode

- **Failure Mode ID**
  - **SM - State management** - The software is unable to maintain state, executes incorrect transitions, dead states, etc.
  - **EH - Error handling** - The software is unable to identify, and handle known system faults
  - **T - Timing** - The software executes the right thing too early or too late
  - **SE - Sequencing** - The software executes the right thing in the wrong order
  - **DD - Data definition** - The software has wrong or incompatible definitions of size, type, format, unit of measure, scale, etc.
  - **PR** - Processing - The software is unable to handle peak loading, extended duration, file I/O etc.
  - **F - Functionality** - The software does the wrong thing perfectly. The software does not meet the basic reason for the software.
  - **A - Algorithm** – The simplest algorithm is a division of two numbers. The most common algorithm fault is when the software engineer fails to write code to handle a denominator that is near zero.
  - **U – Usability** – Usability faults can and have led to mission faults.
  - **ML - Machine learning**
- **Root Cause ID**
  - This is a unique sequential identifier for multiple root causes related to the failure mode.

Artifact ID and

Artifact

Regardless of whether the viewpoint is top level, capability level, SRS level, or interface level the root cause can originate in the following activities:

- **S** - The root cause originates in the software specification due to omission or commission.
- **D** - The root cause originates in the software design due to omission or commission.
- **C** - The root cause originates in the code. The specification and design are clearly correct.

Artifact

This is a unique identifier for multiple root causes originating from the same artifact. This identifier is not always used.

Common Defect Enumerations that Effect the Entire Software Application

- These failure modes don't correspond to a single line of code or single specification statements
- But rather they effect the entire software application
- They are typically not identifiable in code reviews
- They are related to common oversights in engineering that aren't detectable until relatively late in the development

Failure mode	Top Level CDEs that apply to a capability	# Root causes
State management	TL-SM-1 through TL-SM-12	24 originating in specifications and 12 originating in code
Error handling	TL-EH-1 through TL-EH-30	30 originating in specifications and 30 originating in code
Functionality	TL-FC-1 through TL-FC-7	9 originating in specifications and 10 originating in code
Processing	TL-PR-1 through TL-PR-8	13 originating in specifications and 14 originating in code
Timing	TL-T-1 through TL-T-7	7 originating in specifications and 6 originating in code
User	TL-U-1 through TL-U-10	10 originating in specifications and 10 originating in code
Data definition	TL-DD-1 through TL-DD10	10 originating in specifications and 10 originating in code
Algorithm	TL-A-1 through TL-A-9	18 originating in specifications and 9 originating in code
Machine Learning	TL-M-1 through TL-M-3	5 originating in sampling errors, 4 originating in ML process and 9 in modeling

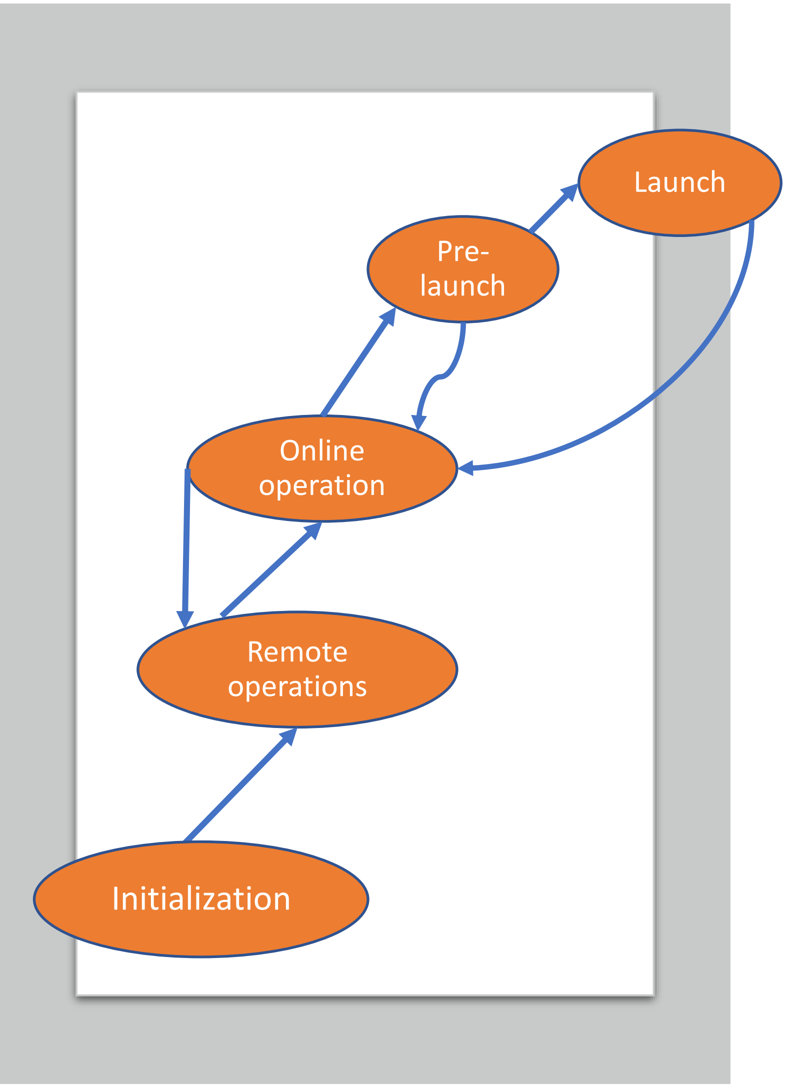
MISSION READY SOFTWARE
SOFTREL LLC

Example Top level Enumeration

Failure Mode ID	Root cause	Tailoring/ Relevance	Example	CDE	Description of root cause
TL-SM-1	Prohibited state transitions are executed	Highly recommended for all mission critical software. This failure mode often has severe consequences, and it is relatively easy to identify.	Prohibited transitions are what lead to irrecoverable events such as inadvertent missile launches, inadvertent radiation, etc.	TL-SM-1-S-1	The specifications fail to identify allowed or disallowed state transitions
				TL-SM-1-S-2	The specifications identify allowed state transitions but fail to state that the disallowed state transitions are prohibited
				TL-SM-1-C-1	The specification for prohibited transitions is clear but the software does not meet it.

MISSION READY SOFTWARE
SOFTREL LLC

Example of prohibited state transitions



- As with nearly all state diagrams only the “allowed” transitions are shown.
- The software failure modes lie in the “prohibited’ transitions.
  - Initializing to online, prelaunch, launch
  - Remote to prelaunch, launch, initializing
  - Prelaunch to remote, initializing
  - Launch to remote, launch to initializing
  - Faulted to launch, prelaunch, online, remote
- The code may allow the prohibited transitions due to a coding error or poor specifications.
- Prohibited transitions are rarely tested because the test engineers only test the valid transitions.

Example SFMEA using CDE

Failure Mode ID	Failure mode	Specific root cause	CDE	Origin	Effect	Severity	Likelihood		Detectability	RPN
							Manifestation	Controls		
TL-SM-1	Prohibited state transitions are executed	Transition from initializing to launch is allowed	TL-SM-1-S-2	The specifications identify allowed state transitions but fail to state that this transition is not allowed at all	Inadvertent launch	10	10	10	10	1000

Manifestation – This is a single point failure so manifestation likelihood is 10 out of 10
No controls for this transition exist so controls likelihood is also 10 out of 10.
Likelihood is average of manifestation and control likelihood which averages to 10.
There is no specification or test case for this transition so it won't be detected in testing.
If this prohibited transition is controlled/mitigated and tested the controls and detectability risk levels are reduced to 1 so residual RPN is 50.

Common Defect Enumerations that Effect a Specific Capability

- These failure modes don't correspond to a single lines of code or single specification statements
- But rather they effect a specific capability or feature within the software
- Most of the top level failure modes apply to a specific capability
- However, at the capability level there can be problems due sequencing, consistency with other capabilities, timing within the capabilities that comprise the application

Failure mode	Capability Level CDE	Top Level CDEs that apply to a capability
State management		TL-SM-1 through TL-SM-12
Error handling		TL-EH-1 through TL-EH-30
Functionality	CL-FC-1 and CL-FC-2 2 root causes originating in specifications and 2 in coding	TL-FC-1 through TL-FC-7
Processing	CL-PR-1 1 root cause originating in specifications and 1 in coding	TL-PR-3, TL-PR7 and TL-PR-8
Sequencing	CL-SE-1 through CL-SE-5 5 root causes originating in design and 5 in coding	
Timing	CL-T-1 through CL-T-6 6 root causes originating in design and 6 in coding	
User		TL-U-1 through TL-U-10
Data definition	CL-DD-1 through CL-DD-5 5 root causes originating in design and 5 in coding	TL-DD-1 through TL-DD10
Algorithm	CL-A-1 and CL-A-2	TL-A-1 through TL-A-7

Example Capability level Enumeration

Failure Mode ID	Failure mode description	Tailoring/ Relevance	Discussion/Example of failure mode	Common Defect Enumeration	Description
CL-PR-12	Capability is interrupted while executing	Any mission or safety critical software capability	The software specifications fail to state what is required to happen when a capability is prematurely aborted or there is a loss of power while the capability is executing	CL-PR-12-S-1	It's a common oversight to neglect to consider what the system is required to do when there is a loss of power or abort while a capability is executing
			The software specifications for interruption of a capability are clear but the code does not meet the specification.	CL-PR-12-C-1	

Analyze what the software does AFTER the interruption – not the interruption itself

Example SFMEA using CDE

Failure Mode ID	Failure mode	Specific root cause	CDE	Origin	Effect	Severity	Likelihood		Detectability	RPN
							Manifestation	Controls		
CL-PR-12	Capability is interrupted while executing	Loss of power while turret is unstowed and in motion	CL-PR-12-S-1	The specifications fail to state what the software does after a loss of power while turret is moving	Upon restoration of power the turret may be unstowed and might move	10	10	10	10	1000

Manifestation – This is a single point failure so manifestation likelihood is 10 out of 10
No controls for stowing the turret upon startup.
Likelihood is average of manifestation and control likelihood which averages to 10.
There is no specification or test case for stowing turret after power loss so this failure mode won't be detected in test.
If the software stows the turret upon startup and the power loss is tested the RPN reduces to 50

Common Defect Enumerations that Effect a Specific Specification Statement

Failure mode	Specification Level CDE
State management	SL-SM-1
Functionality	SL-FC-1 to SL-FC-6
Timing	SL-T-1 through SL-T-5
Data definition	SL-DD-1 and SL-DD2

- These failure modes are caused by a single specification that is faulty
- Specification statements can cause failure modes when they are not accurate, complete or verifiable
- This includes all top level and capability level specifications that arise in the specification artifact

Examples of Specification level Enumeration

Common defect enumeration	Failure mode description	Discussion/Example of failure mode	Tailoring/ relevance
SL-T-3	The timing range has a lower bound but no upper bound	Ex: The software shall wait at least 100ms after verifying that voltages are up to transition to the next state. What if the voltages never come up? Or take several minutes to come up?	All requirements with timing specifications
SL-T-4	The timing range has an upper bound but no lower bound	Ex: The software shall take no longer than x ms to transition to the next state. What if the transition occurs immediately? Can the rest of the system handle that?	

There can be 10,000+ software requirements for some systems. No time to analyze them all. However, the ones with "magic" numbers are prone to faults because everyone assumes the number is correct.

MISSION READY SOFTWARE
SOFTREL LLC

Examples of Specification level Enumeration

Common defect enumeration	Failure mode description	Discussion/Example of failure mode	Tailoring/ relevance
SL-DD-1	Accuracy requirements are too loose	Accuracy requirements are developed based on subject matter expertise. Unfortunately, because are they are defined by systems experts few software people question their origin or validity. Example: Faulty requirement: The comparison of the velocity input from GPS receiver to software-based estimates was specified to have accuracy of ± 2 m/s when it should have been 1 m/s.	All requirements with accuracy specifications
SL-DD-2	Accuracy requirements are too tight	The above example could have also been too tight and that the actual accuracy requirement could have been > 2 m/s	

Any time there is an accuracy requirement there are exactly two failure modes. The requirement is too loose or too tight. Until simulations are conducted to show that the number is the right number, this failure mode is possible.

MISSION READY SOFTWARE
SOFTREL LLC

Common Defect Enumerations that Effect a Software Interface

- These failure modes are caused by a faulty software interface
- All interface failures are related to faulty data definition
  - Data is the wrong type, wrong unit of measure, wrong resolution, wrong scale, missing default values, missing minimum and maximum values, etc.

Failure mode	Interface level CDE
Data definition	IL-DD-1 and IL-DD-18

Interface failures are the most likely when two different software organizations are writing the code.

If the two organizations are in different countries the risk is even greater.

Examples of Interface Level Enumerations

Failure mode ID	Failure mode description	Common defect enumeration	Description	Example
IL-DD-4	The interface data is the wrong scale (i.e. ms vs sec)	IL-DD-4-S-1	The specification does not have the correct scale or has no scale at all	One software component is expecting seconds as an input but the other is outputting ms. The results will be off by 100.
		IL-DD-4-C-1	The specification is correct but the code is not to spec	
IL-DD-5	The interface data is the wrong unit of measure (meter vs. feet)	IL-DD-5-S-1	The specification does not have the correct unit of measure or has no unit of measure at all	The NASA Mars Climate Orbiter crash
		IL-DD-5-C-1	The specification is correct but the code is not to spec	
IL-DD-7	The interface data has no default value	IL-DD-7-S-1	The specification does not have a default value	A “default” value is the value of a data item on startup and whenever there is a failure condition. If this isn’t defined the data can become corrupted.
		IL-DD-7-C-1	The specification is correct but the code is not to spec	

These apply to all numerical interface parameters

MISSION READY SOFTWARE SOFTREL LLC

Summary

- The Common Defect Enumeration provides the following value to software engineering and reliability engineering
  - An organized and enumerated list of failure modes
  - The enumerations never change however new failure modes and root causes can be added based on lessons learned
  - The CDEs aren’t limited to only those failure modes introduced during the coding activities
  - The CDE is posted in a Wiki format
  - The CDE can be used for:
    - Improving requirements, design and code reviews
    - Improving the coverage of a software FMEA

MISSION READY SOFTWARE
SOFTREL LLC

About the Author

- Authored the industry guidance on software FMEA - "Effective Application of Software Failure Modes Effects Analysis", published for CSIAC, 2014.
- Chairperson of IEEE 1633 Recommended Practices for Software Reliability Working Group (2016 edition) –See video for more information: https://www.youtube.com/watch?v=vmW2EM5KkMo&t=18s
- 39 years of software engineering and software reliability experience
- Authored the DoD SOW Language for Software Reliability
- Authored NASA's Software FMEA and software FTA training webinar
- Authored Intel's process for evaluating vendors with regards to software
- Co-authored USAF Rome Laboratory "System and Software Reliability Assurance Notebook", with Boeing Corp.
- Authored "Ensuring Software Reliability", Marcel-Dekker, 1993.
- Benchmarked 200+ software intensive systems for reliability, on time delivery and customer satisfaction. See video for more information. https://www.youtube.com/watch?v=HApDHxtG_Mk&t=1s
- Has analyzed almost 1 million failures due to software and categorized by failure mode and root cause. See video for more information. https://www.youtube.com/watch?v=XdrzT8b8qXs&t=20s
- IEEE Lifetime Achievement Award, 2017, Reliability Society.
- Managed small and large software development and test teams throughout career and has applied virtually every development practice for almost 40 years
- U.S. Patent 5,374,731 for predictive modeling
- 1983 Graduate of Georgia Tech

MISSION READY SOFTWARE
SOFTREL LLC

MISSION READY SOFTWARE

HTTP://WWW.MISSIONREADYSOFTWARE.COM

SALES@MISSIONREADYSOFTWARE.COM

321-514-4659