Five Common Mistakes when Conducting Software Failure Modes Effects Analysis

Ann Marie Neufelder
SoftRel, LLC
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

Five Common Mistakes when Conducting Software Failure Modes Effects Analysis

- The software FMECA is a powerful tool for identifying software failure modes but there are 5 common mistakes that can derail the effectiveness of the analysis.
  - #1 - Software is analyzed as a black box (and shouldn't be).
  - #2 - It's assumed that the software will work as expected
  - #3 - It's conducted far too late in development life cycle
  - #4 - It's conducted at wrong level of abstraction
  - #5 - The most common failure modes aren't considered

Copyright SoftRel, LLC 2019

#1 - Software is analyzed as a black box (and shouldn’t be).

- The single most common mistake is to analyze the software based on what it "is" instead of what it "does".
- The black box approach is common for hardware FMECA.
  - However, it doesn't work well for software.
  - Software doesn't wear out - it fails because the code doesn't perform the required functions.
  - Hence, it must be analyzed from a functionality versus black box standpoint.

Copyright SoftRel, LLC 2019

#1 - Software is analyzed as a black box (and shouldn't be).

Examples of "Black box" SFMECA which should be avoided.

LRU	Failure mode	*Recommendation*
Turret CSCI	CSCI fails to execute	*Doesn't address states, timing, missing functionality, wrong data, faulty error handling, etc.*
Turret CSCI	CSCI fails to perform required function	*CSCI performs far too many features and functions. List each feature and what can go wrong instead.*

Copyright SoftRel, LLC 2019

Example of a use case to move a turret analyzed based on what it does/doesn’t do and not what it is

Use Case	Failure mode	Root causes
Move turret	Faulty timing	• Turret moves too late • Turret moves too early
	Faulty sequencing and state management	• Turret moves inadvertently • Turret fails to move when commanded
	Faulty error handling	• Turret exceeds the maximum range allowed • Failures in turret hardware aren’t detected
	Faulty processing	Turret moves upon startup after an abnormal shutdown
	Faulty data	• Turret moves to the wrong location because of improperly formatted, improperly scaled or null data • Turret comes too close to a hard stop because of overly tight specifications • Turret doesn’t move the entire spectrum of possible radians
	Faulty functionality	Use case doesn’t meet the system requirements

#2 - It's assumed that the software will work as expected

- The "software" FMECA focuses on how the "software" fails.
- Yet many analysts assume that the software will work perfectly.
- There's no point in doing a "software" FMECA if you're going to assume that the software always works.
- One must assume that

  1) Unwritten assumptions will lead to failures

  2) If an important detail isn't in writing it won't get coded or tested

  3) If the requirements don't discuss fault handling the software won't handle faults

  4) even when the requirements are complete, the code may not be written to meet the requirements.

Copyright SoftRel, LLC 2019

Example: Unwritten assumptions in the software requirements leading to a failure

European Space Agency CryoSat-1

Satellite is lost at a cost of $186 million.

↑

Engine continues to operate until fuel is consumed

↑

First stage of launch on 10/8/05 is successful. Second stage stops performing when required command to cut off main engine doesn't occur.

↑

SRS specifications missing requirement for main engine cutoff

Copyright SoftRel, LLC 2019

Example: Important details missing from requirements won't get coded or tested

This is the specification for the logging feature:

1) The software shall log all warnings, failures and successful missions.
2) At least 8 hours of operation shall be captured
3) Logging to an SD card shall be supported in addition to logging to the computer drive

This is what you know about the software organization and software itself

1) Logging function will be called from nearly every use case since nearly every use case checks for warnings, failures and successes
2) Testing will cover the requirements. But no plans to cover stress testing, endurance testing, path testing, fault insertion testing.
3) Software engineers have discretion to test their code as they see fit.
4) There is a coding standard but there is no enforcement of it through automated tools and code reviews only cover a fraction of the code

Copyright SoftRel, LLC 2019

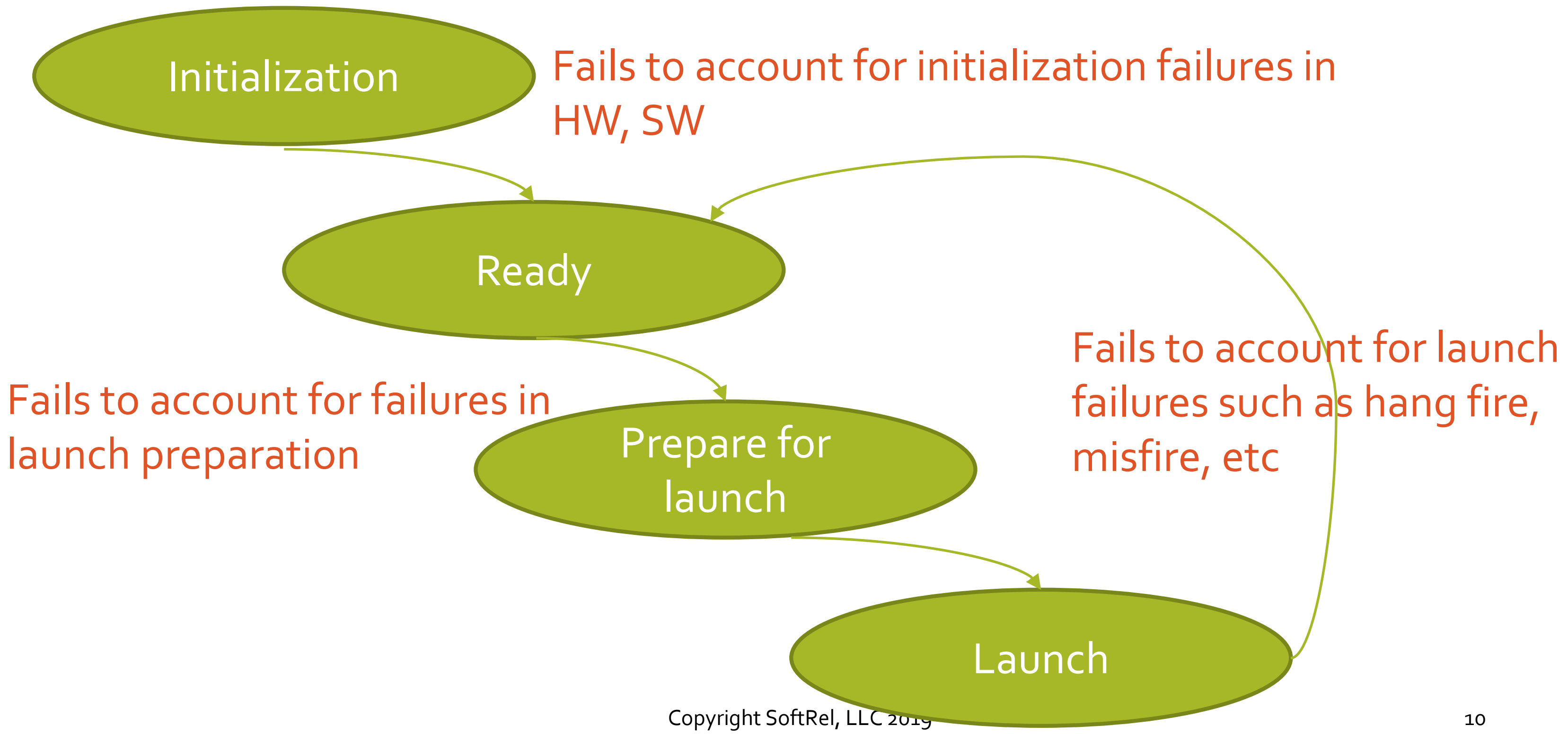
Example: Important details missing from requirements won't get coded or tested

- **These are the faults that can/will fall through the cracks**
  - No checking of read/write errors, file open, file exist errors which are common
  - No rollover of log files once drive is full (may be beyond 8 hours)
  - No checking of SD card (not present, not working)
  - Logging when heavy usage versus light or normal usage (might take less than 8 hours to fill drive if heavy usage)
- **This is why these faults aren't found prior to operation**
  - No one is required to explicitly test these faults
  - No one is required to review the code for this fault checking
  - No one is required to test beyond 8 hours of operation
- **This is the effect if any of these faults happens**
  - Entire system is down because it crashes on nearly every function once drive is full, SD card removed, file is open or read/write errors
- With the SFMEA you cannot assume that best practices will be followed unless there is a means to ***guarantee that.*** *Even when that's the case the root cause should be tracked.*

Copyright SoftRel, LLC 2019

Example: If the requirements don't discuss fault handling the software won't handle faults

- This state diagram based on the written software requirements, doesn't have a faulty state or transitions to/from a faulty state
- Hence, these faults are unlikely to be handled in design, code or test plan
- The SFMECA should not assume otherwise


Initialization
Fails to account for initialization failures in HW, SW
Ready
Fails to account for launch failures such as hang fire, misfire, etc
Fails to account for failures in launch preparation
Prepare for launch
Launch

Copyright SoftRel, LLC 2019

Example: Even when the requirements are complete, the code may not be written to meet the requirements

Cost = $18.5 million of 1962 dollars.

↑

Rocket destroyed 293 seconds after liftoff.

↑

Faulty corrections sent the rocket off course.

↑

Without the smoothing function the software treated normal variations in velocity as if they were serious.

↑

The requirements document clearly indicated an overbar which was supposed to be an averaging function of velocity. However, the programmer ignored the superscript when transcribing the formula into code.

Mariner 1 rocket failure in 1962.
[Mariner]

Copyright SoftRel, LLC 2019

#3 It's conducted far too late in the development life cycle

- The perfect time to conduct a software FMECA is immediately after the first pass of the software requirements/use cases and before the code is written to those requirements.
- Typically the first pass of the SRS and use cases is when the "shalls" are defined.
- In the second pass is when the "shall nots" or alternative flows should be defined.
- The SFMECA can be used to strengthen the requirements and can even be used as a requirements review tool.
- If SFMECA is conducted after code is written
  - Less effective but still time to effect test procedures
- If SFMECA is conducted after testing is finished
  - Significantly less effective – can only effect user training or next release of software

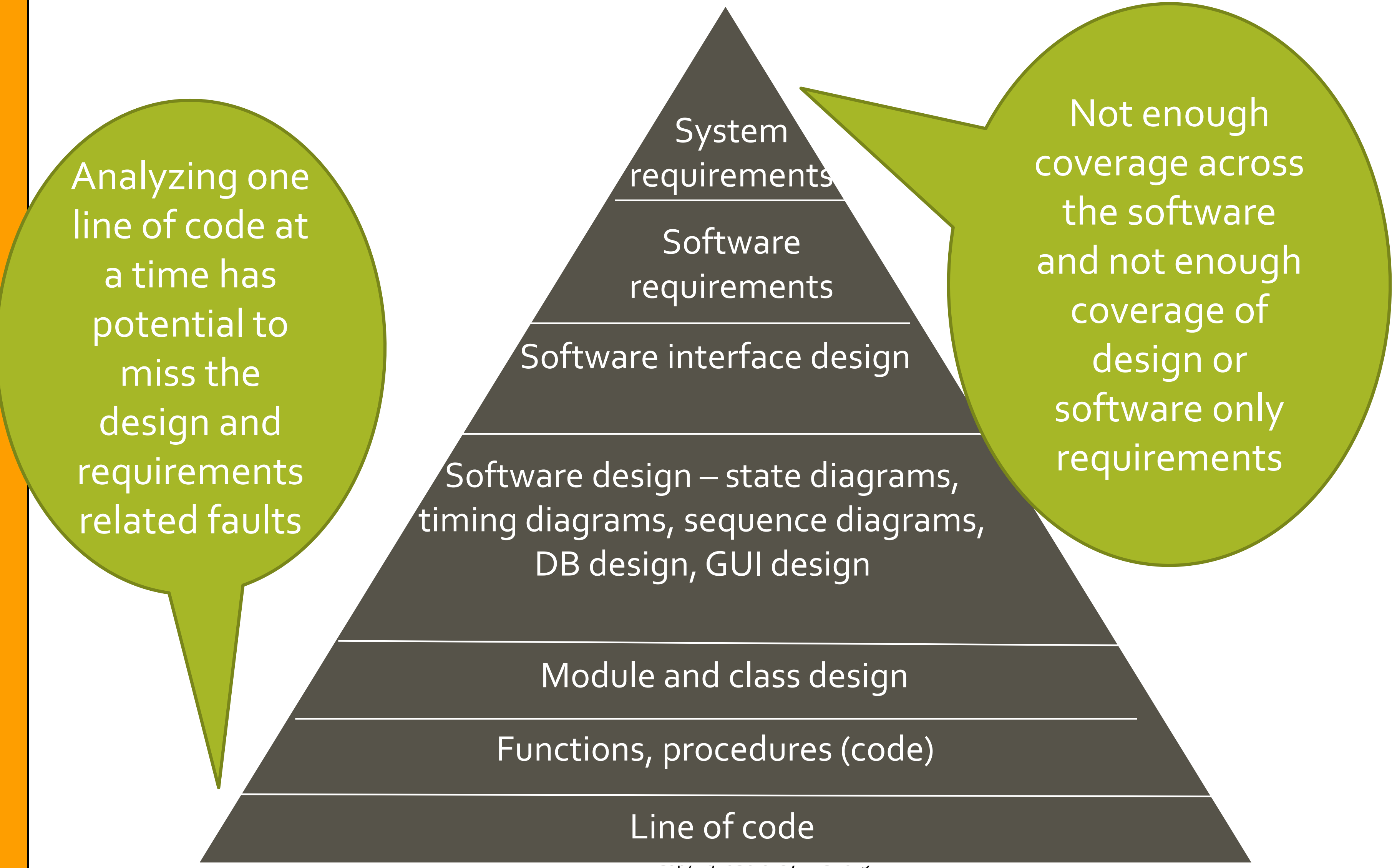
Copyright SoftRel, LLC 2019

#4 It's conducted at the wrong level of abstraction

- Some analysts work through the code one line at a time and analyze how that single line of code could fail.
- For software functions that are associated with particularly high hazards that may be appropriate but not necessarily sufficient.
- When analyzing one line of code at a time the analyst misses the failure modes due to
  - 1) required code is missing altogether
  - 2) defects that are caused by more than one line of code.
- Effective software FMECAs focus on the requirements, use cases, interfaces, detailed design and usability.

Copyright SoftRel, LLC 2019

Focusing at too high or too level a level of abstraction


System requirements
Software requirements
Software interface design
Software design – state diagrams, timing diagrams, sequence diagrams, DB design, GUI design
Module and class design
Functions, procedures (code)
Line of code
Analyzing one line of code at a time has potential to miss the design and requirements related faults
Not enough coverage across the software and not enough coverage of design or software only requirements

What to focus on and when

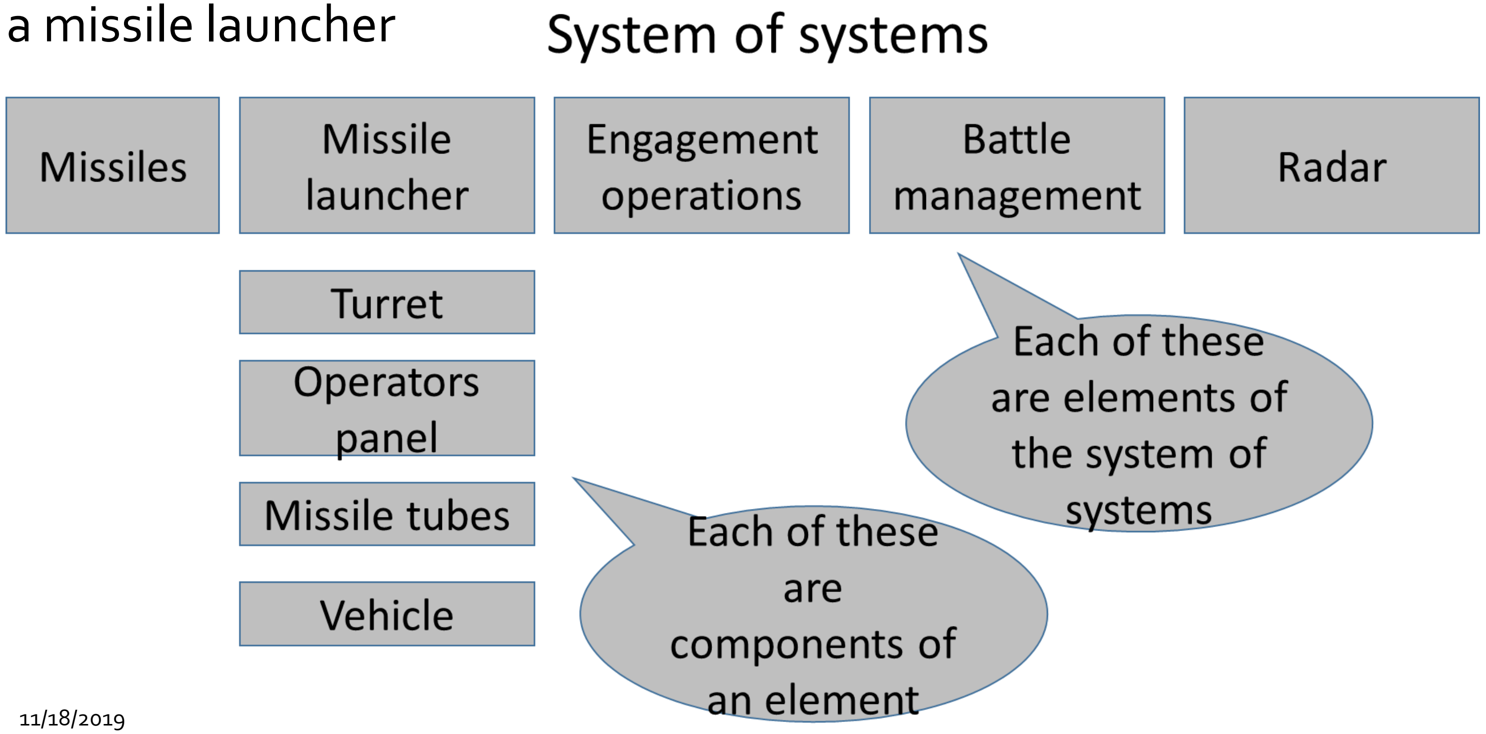
FMEA Viewpoints		
Level of architecture applicable for viewpoint	**Failure Modes**	**When focusing on this is most effective**
The use cases, system and software requirements	The system does not do it's required function or does the wrong function	New requirements or new system. Major changes to an existing system.
The interface design	The system components aren't synchronized or compatible	Many components developed by more than one organization.
The detailed design or code	The design and/or code isn't implemented to the requirements or design	When there is detailed logic or algorithms that are mission critical – i.e. launch calculator.
The ability for the software to be consistent and user friendly	The end user causes a system failure because of the software interface	When the user can cause a failure or when lack of usability can cause a mission failure

Use cases are highly recommended

- Use cases have been proven to reduce software defects because they “visualize” the software requirements in terms of sequence, timing, and data
  - Software engineers can visualize how the software works better with use cases then with only a list of text software requirements
- Use cases also increase software FMEA effectiveness
  - Software FMEA analysts can visualize what can go wrong faster with use cases then with only a list of text software requirements
  - Failure modes that span across the requirements easier to identify
  - Failure modes related to missing level of detail easier to identify
  - Failure modes related to faulty error handling easier to identify
  - Failure modes related to sequence easier to identify
  - Failure modes related to flow of data easier to identify

SFMEAs are most effective when boundary determined in advance of analysis

- Example – a System of System is comprised of several elements
- System of system level SFMEA would focus on all of the above elements interfacing with each other
- Element level SFMEA would focus on just one of these elements
- Component level would focus on a part of one element such as the turret in a missile launcher



#5 The most common failure modes aren't considered

- The most common failure modes that apply to all software intensive systems are:
  - Faulty functionality - missing required functionality, function doesn't work as required
  - Faulty processing - can't perform after an interruption of service or extended usage
  - Faulty error handling - doesn't handle hardware, interfaces, software or user faults
  - Faulty state management - executes when it shouldn't, encounters dead states, faulty state transitions, etc.
  - Faulty timing - race conditions, a function executes too early, too late, accumulates timing errors when left on too long, etc.
  - Faulty data isn’t handled - missing, corrupt, improperly sized, improperly formatted, improperly scaled data isn't handled

Weak development area	Common failure modes/root causes
Design is conducted after code is written or is too high level. No logic diagrams when needed.	Faulty logic
Requirements/Design/Use cases doesn't describe detailed state transitions, faulty states, prohibited states	Faulty state management
Requirements/Design/Use cases doesn't cover error handling, alternative flows	Faulty error handling
Requirements/Design/Use cases don't cover data definitions or interface data	Faulty data not handled
Requirements are too high level	Faulty functionality
No timing diagrams on timing sensitive software	Faulty timing

Tip: The most common failure modes/root causes are related to weakest link of development

Element level events	Software Related Failure mode
Missile misfires	• Faulty processing - Software aborts during specific points of missile release
Missile hang fires	
Missile misses target trajectory	• Faulty timing - Missile launches too early or too late • Faulty data - Launch calculator can't handle faulty data • Faulty algorithm in launch calculator
Missile fails to launch when commanded	• Faulty state transitions with missile launching software
Missile launches when not commanded	
Turret moves when not commanded	• Faulty state transitions with turret movement

Tip: Identify software related failure modes by working backwards

Failure mode	Root causes
Faulty processing - Software aborts during specific points of missile release	Software crashes, computer is shut down or loses power, end user aborts mission
Missile launches too early Missile launches too late	Response parameters are too short Response time parameters are too long Software processing is sluggish Software built up time inaccuracy
Launch calculator can't handle faulty data	Calculator has incorrect specification for algorithm
Launch calculator has faulty algorithm	Calculator has correct specification for algorithm but incorrect implementation

Tip: Identify software related root causes by working backwards

Failure mode	Root causes
Faulty state transitions with missile launching software	• Launch software is missing code for specified state transition • Launch software is missing a required state transition in specifications
Faulty state transitions with missile launching software	Launch software doesn't check for required launch conditions prior to launch
Faulty state transitions with turret movement	Software doesn't stow when commanded or doesn't stow when it should

Tip: Identify software related root causes by working backwards

Failure Event	Associated failure mode
Several patients suffered radiation overdose from the Therac 25 equipment in the mid-1980s. [THERAC]	Faulty timing - A race condition combined with ambiguous error messages and missing hardware overrides.
AT&T long distance service was down for 9 hours in January 1991. [AT&T]	Faulty sequencing - An improperly placed "break" statement was introduced into the code while making another change.
Ariane 5 Explosion in 1996. [ARIAN5]	Faulty data - An unhandled mismatch between 64 bit and 16 bit format. Faulty error handling – One size fits all reboot
NASA Mars Climate Orbiter crash in 1999.[MARS]	Faulty data - Metric/English unit mismatch. Mars Climate Orbiter was written to take thrust instructions using the metric unit Newton (N), while the software on the ground that generated those instructions used the Imperial measure pound-force (lbf).
On October 8th, 2005, The European Space Agency's CryoSat-1 satellite was lost shortly after launching. [CRYOSAT]	Faulty functionality - Flight Control System code was missing a required command from the on-board flight control system to the main engine.
A rail car fire in a major underground metro system in April 2007. [RAILCAR]	Faulty error handling - Missing error detection and recovery by the software.

Copyright SoftRel, LLC 2019

Just a few examples of failure modes that causes major failure events

Number 6-10 on common causes for ineffective SFMEA

6. Not following up with the root-causes and mitigations identified
7. Assigning the analysis to a person who doesn't have experience with software development
8. Too much time spent on analyzing the probability/frequency when analyzing controls is what's important
9. Assigning the analysis to exactly one person
10. Trying to apply the SFMEA to everything or picking an arbitrary starting point

Copyright SoftRel, LLC 2019

Additional references

- Effective Application of Software Failure Modes Effects Analysis
  - This book provides practical guidance and examples for conducting effective software FMECAs.
- If you want to learn more- attend the Software Reliability Bootcamp in Huntsville, AL January 28th-30th