Failure Mode and Effect Analysis

Lecture 4-3

Software FMEA

"Failure Mode and Effects Analysis in Software Software Development," Pries, SAE Technical Paper 982816

Software Development is Different

FMEA

- Process variation can never be eliminated or even reduced below a moderate level
- No two modules are alike so process performance always includes an intrinsic degree of variability
- There are very large differences in skills and experience from one developer to another
- Specifications are not based around tolerances
- Systems don't fail because they are assembled from many loosely toleranced components. A single well-placed defect in a low level component can be catastrophic

2

Measurements

Software development processes can be fully characterized by three simple measurements

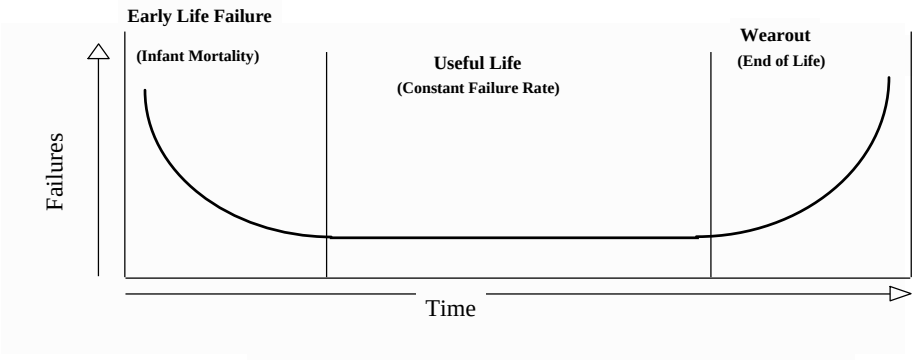
- **Time**: the time required to perform a task
- **Size**: the size of the work product produced
- **Defects**: the number and type of defects, removal time, point of injection and point of removal

3

Hardware vs Software

Behavior over the system life cycle

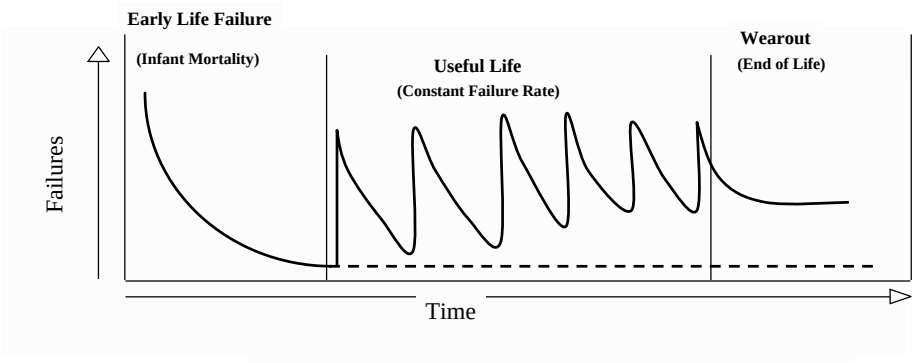
Hardware Model



4

Software Life Cycle Trend

Software Model



5

FMEA

Software Life Cycle Trend

- Software does not wear out in the physical sense, it deteriorates
- Deterioration is not a function of time
- It is a function of the side effects of changes made in the maintenance phase to:
  - correct latent defects
  - adjust software to changing requirements
  - improve performance

6

Software Life Cycle Trend

- Hardware deteriorates due to lack of maintenance but software deteriorates because of maintenance
- Software doesn't wear out. It ceases to be useful and / or cost effective in a changing technical environment.
- Software failure is a function of usage, not of time

7

Software FMEA

Applications:

- Anticipating software problems
- Prelude to test design
- Safety and hazard analysis, causes and potential solutions
- Documentation

8

SWFMEA

- SWFMEA is not to be a replacement for traditional software reliability methods, rather it is intended to be a systematic thinking tool, a means of anticipating issues and improving validation

9

Begin with Outputs

- Outputs are observable behavior
- Observable behavior leads to failure modes that may be detected by a driver, customer test group, or a servide organization
- One or more inputs will be related to each output
- At the end of the process, all outputs must capture all inputs

10

Form

FMEA

DESIGN FMEA

PRODUCT:		FMEA NO
PART / SYSTEM:		PAGE
DRAWING REFERENCE:	DATE:	BY:

SEVERITY | OCCURRENCE | EFFECTIVENES | RPN = S x O x E

PART OR SYSTEM NAME / FUNCTION(S)	POTENTIAL FAILURE MODE	EFFECTS OF FAILURE -LOCAL, NEXT LEVEL, END USER	SEV	POTENTIAL CAUSES OF FAILURE	OCC	DESIGN VERIFICATION ACTIVITES	EFF	RPN	*ACTION PRIORITY*	RECOMMEN. CORRECTIVE ACTIONS	RESPONSIB &DUE DATE	ACTIONS TAKEN	RESULT SEV	RESULT OCC

FUNCTION OUTPUT	POTENTIAL FAILURE MODE	EFFECTS OF FAILURE	SEV	POTENTIAL CAUSES	OCC	ENGINEERING VERIFICATION	EFF	RPN
Speedometer / correct vehicle speed	**Gauge dither aperiodic**	**Driver cannot assess speed resulting in safety hazard**	**10**	**Poor filter algorithm and oscillating data from sensor output**	**5**	**Test 001 - randomly oscillating signal**	**3**	**150**
			10	**Input values not properly scaled**	**7**	**Test 003 - scaled input**	**2**	**140**
	Guage dither periodic	**Driver cannot assess speed resulting in safety hazard**	**10**	**Poor filter algorithm and oscillating data from sensor output**	**3**	**Test 001 - randomly oscillating signal**	**3**	**90**

11

Example - Speedometer

FMEA

FUNCTION OUTPUT	POTENTIAL FAILURE MODE	EFFECTS OF FAILURE	SEV	POTENTIAL CAUSES	OCC	ENGINEERING VERIFICATION	EFF	RPN
Speedometer / correct vehicle speed	**Gauge dither aperiodic**	**Driver cannot assess speed resulting in safety hazard**	**10**	**Poor filter algorithm and oscillating data from sensor output**	**5**	**Test 001 - randomly oscillating signal**	**3**	**150**
			10	**Input values not properly scaled**	**7**	**Test 003 - scaled input**	**2**	**140**
	Guage dither periodic	**Driver cannot assess speed resulting in safety hazard**	**10**	**Poor filter algorithm and oscillating data from sensor output**	**3**	**Test 001 - randomly oscillating signal**	**3**	**90**

12

Severity

SEVERITY of Effect		
None	No effect	1
Very Minor	Observable software-driven behavior does not conform to spec. Discriminating customer notices	2
Minor	Observable software-driven behavior does not conform to spec. Average customer notices	3
Very low	Observable software-driven behavior does not conform to spec. Customer notices.	4
Low	Software makes item operable but comfort/convenience items have reuced performance. Customer is uncomfortable	5
Moderate	Software makes item operable but comfort/convenience items are inoperable. Customer is uncomfortable	6
High	Software makes item operable but at reduced levels of performance. Customer dissatisfied	7
Very high	Software makes item inoperable with loss of primary function	8
Hazardous - warning	Software failure affects safe operation of item and / or non-compliance with government regulation	9
Hazardous - no warning	Software failure affects safe operation of item and / or non-compliance with government regulation	10

13

Interface Analyses

Software

Software units and components interface with each other through sub-routine arguments, through messages, or through a global pool. When possible, the software engineer should generate a complete analysis of interface variables using a software tool. Otherwise, the engineer must examine all assignment and quasi-assignment statements as candidates for software interface variables.

14

Interface Analyses

Hardware

The engineer frequently will have a tendency to allow the software DFMEA to “drift” into becoming a hardware DFMEA. To handle this issue, the engineer can define a hardware input failure, for example, and then analyze the failure on the basis of how the software reacts to the hardware failure. A typical case is that of an open circuit.

- Does the software meet any requirements for handling this error condition?
- Does the software fulfill design intent with respect to handling errors in the hardware?
- Does the software provide some kind of visual indication that a negative event has occurred?

15

SWFMEA and Software Quality

Walkthroughs

the software DFMEA can be used to enhance the quality of requirements, design, and code walkthroughs by highlighting potential failure modes. As with inspections, the DFMEA can be used to keep the walk-through “honest” and ensure that product flaws are at least considered and heeded during the process.

16

SWFMEA and Software Quality

Inspections

Generally, software inspections are a quasi-formal event with a checklist of known software problems supplied to the "inspectors." The software DFMEA can be used to enhance the typical failures enumerated on the checklist or the checklist can be used to enhance the failure modes enumerated in the DFMEA. The difference lies in the emphasis of the two methods:

- inspection looks for typical software problems,
- DFMEA is a tool for anticipating failure mode effects and eliminating them when possible.

17

SWFMEA and Software Quality

Testing

The software DFMEA is a useful first-pass and multi-pass tool for developing test cases. The test designer can take the software DFMEA failure modes, define them as test cases, and develop procedures for stimulating the conditions that can lead to the failure. All versions of the software DFMEA should be retained through the life of the project, preferably under automated configuration control; that way, failure modes that have presumably been eliminated will not disappear from the software DFMEA document.

18

SWFMEA and Static Code Analysis

The DFMEA technique allows the software engineer to anticipate software problems based on defined outputs and inputs. The DFMEA method has no code requirement; that is, the engineer can apply this method early in the life cycle of project development—a capability generally unavailable when using other techniques of requirements, design, and code analysis.

19

Key Challenges

- To find effective ways of classifying software failures into appropriate failure modes
- Estimating the likelihood of each failure mode

Significant difference from traditional FMEA is that ***the software design is not assumed to be perfect and will contain systematic faults inadvertently introduced by software developers***

20

Systems SWFMEA

- SWFMEA may be used very early (before design) to indicate whether the technical plan for SW development is sufficiently robust, to prevent, detect and remove, likely kinds of design faults
- SWFMEA may be helpful in the design stage to identify aspects of the design which may need modification (i.e. design techniques)

Only a subset of final system failure modes will be detectable at the design stage - other failure modes will be introduced during implementation

21

Systems SWFMEA

- SWFMEA may be used before implementation to help guide the selection of appropriate languages, tools and testing techniques
- SWFMEA may be used before operational cut-over to help gauge whether the implemented system falls within specified risk parameters

22

Areas of Research

Need for major research

- into causes and effects of failures in software-based systems
- establish appropriate measures and metrics
- address the causal chain in system development
- establish an effective classification scheme for failures
- costs and benefits in the development process