WHAT CMM AND OTHER STRUCTURED DEVELOPMENTS DO.

The goal of a CMM-rated development environment is to document procedures for all development processes and to establish a system whereby people understand and carefully follow these procedures. Process owners continually improve the procedures with experience. However, a CMM rating alone does not imply a particular test philosophy or a focus on reliability.

WHAT DEVELOPED STRUCTURES DON'T DO.

Unfortunately, requirements-based testing and robustness testing cannot address all permutations of transactions among software units under test and the outside world, simply because no set of tests can address everything that could happen. Testing cannot address the handling of all unexpected internal values and events due to programming and algorithmic errors, inadequate exception processing, timing or scheduling errors, hardware and I/O failures, bad input data, disturbances on power lines, electromagnetic interference, and many others.

Code walk-through reviews are normally part of DO-178B and other structured processes, and reviewers do find weaknesses in the code – for example, reviewers can find code errors that check to see if an input or expected arguments are within expected ranges. However, discovery of such potential problems is only one part of the walk-through. There is no deliberate focus on what could happen if each variable, considered one by one, suddenly takes on an unexpected value, and what unexpected values -- if any -- could cause serious consequences. In other words, the code is only tested on anticipated requirements and unexpected events that can be tested. Software FMEA is an exhaustive review of the finished software product to make sure the software is as reliable and safe as possible under all circumstances.

THE RELIABILITY EXPERTS.

Unlike most software analysts, Omnicon is a specialist in reliability, maintainability, and safety engineering for mission-critical, safety-critical, and revenue-critical systems. This experience, combined with extensive software development work under stringent development guidelines, means we understand the importance of code robustness and we know how to achieve it.

WHY PERFORM A SOFTWARE FMEA?

Why Perform a Software FMEA? While most developers reasonably assume that “software doesn’t fail,” we all know that things sometimes do go wrong. As a processor executes its code – a memory location can be unintentionally overwritten, algorithmic errors and timing problems can occur, processor or interface circuits fail, and bad data can be received from the outside world. This is why software-related catastrophic failures sometimes make headlines, even though the failed software had been subject to highly stringent safety requirements for development and test.

To help prevent such catastrophes, The Omnicon Group’s analysis looks for worst-case system effects when any one software failure occurs, and in particular to determine whether a single failure can result in a catastrophic event. Since you cannot test for every potential failure, the FMEA takes software quality beyond what qualification testing achieves. It helps avoid the far greater expense of fixing problems after system delivery, potentially ruinous expenses of catastrophic failures and headlines.

WHAT IT DOES FOR YOU

Software FMEA makes critical software systems safer and more reliable. Software FMEA does not predict software failures, but aims to determine whether a single failure can cause specific catastrophic events or other serious effects. At the same time, the analysis can identify possibilities of less serious consequence so that source code can be made more robust in specific places.

WHAT YOU RECEIVE

You receive a complete FMEA report similar to a hardware FMEA report. The report includes:

• A table summarizing all possible system failures and a list of the low-level failures that can cause each of them, including critical failures. A sample table is illustrated on the next page.

• A summary of how the analysis was performed, including ground rules and assumptions.

• A customized FMEA worksheet. A sample work-sheet is illustrated on the next page.

• An optional fault tree analysis showing the inter-relationship of low-level hardware and software failures that together cause undesirable effects. Above all, you receive:

• A totally independent, detailed review of your critical software.

• Identification of potentially serious problems and suggestions for dealing with them.

• Suggestions for making specific parts of the software more robust.
### The Software FMEA Process

Software FMEA is intrinsically tedious and potentially confusing, but a structured approach and specially tailored database tools make the process feasible, highly accurate, and very thorough.

#### What You Provide

- System-level description documents
- Analysis requirements
- System design requirements
- Source code, if analysis is code level
- Design capture documentation
- Design description documents
- Developer feedback in response to analyst questions

The steps at the right summarize the analysis process. You supply Omnicon with available materials to support the analysis at the level you desire – code level or interface level.

During the FMEA process, Omnicon analysts will advise your developers of any suspected software flaws or weaknesses as we discover them.

---

### Omnicon’s FMEA Process, Step by Step

1. **Become Familiar with System and Software**
   - Use tools and established guidelines
   - Identify where more information is needed
   - Obtain needed info, make necessary assumptions

2. **Capture Data in Database Tool**
   - Customize database for analysis level and analysis requirements
   - Develop shorthand annotation where applicable

3. **Develop Rules and Assumptions**
   - Build upon experience
   - Involve entire analysis team
   - Involve entire system

4. **Develop Descriptive Failure Modes**
   - Determine ways that elements can fail
   - Build a table to avoid duplicate descriptions

5. **Determine How Individual Failures Affect System**
   - Examine elements subject to failure, one by one
   - Select appropriate system failure modes from menus using known hardware failure modes where possible

6. **Generate the Report**
   - Generate tables (real samples are shown here)
   - Summarize FMEA ground rules and assumptions
   - Write additional report material
   - Assemble the report, save on electronic media

---

### Software Failures Causing System Failure

<table>
<thead>
<tr>
<th>ID</th>
<th>Input Variable</th>
<th>Type</th>
<th>HW</th>
<th>Function</th>
<th>Failure Mode</th>
<th>Failure Cause</th>
<th>Local Failure Effect</th>
<th>System Failure Effect</th>
<th>Module</th>
<th>Line</th>
<th>Notes / Suggestions</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.13</td>
<td>DeltaTimeScaleFactor</td>
<td>float</td>
<td>None</td>
<td>Used as a constant to scale time differences. 0.1/76.8.</td>
<td>Incorrect Value</td>
<td>Hardware / Software Failure</td>
<td>The delta between LastFinTim and last fin time will be incorrect.</td>
<td>Definite incorrect time or sequence data returned to IGU.</td>
<td>CommandProc</td>
<td>270</td>
<td>None.</td>
</tr>
<tr>
<td>12.14</td>
<td>InbufCmdNumber</td>
<td>int</td>
<td>None</td>
<td>Command sent by IGU. (Position of bit set in Inbuf command byte).</td>
<td>Value equal to WHACTIVATECMDNO</td>
<td>Hardware / Software Failure</td>
<td>WHACounters will not be zeroed when they should be.</td>
<td>Possible WH inadvertently activated.</td>
<td>CommandProc</td>
<td>284</td>
<td>None.</td>
</tr>
<tr>
<td>12.15</td>
<td>InbufCmdNumber</td>
<td>int</td>
<td>None</td>
<td>Command sent by IGU. (Position of bit set in Inbuf command byte).</td>
<td>Value not equal to WHACTIVATECMDNO</td>
<td>Hardware / Software Failure</td>
<td>Counters will be zeroed.</td>
<td>Possible loss of ability to activate WH.</td>
<td>CommandProc</td>
<td>284</td>
<td>None.</td>
</tr>
<tr>
<td>12.16</td>
<td>CommandProcSubsTblAdr</td>
<td>addr</td>
<td>None</td>
<td>Table for indexed call (Used as Constant).</td>
<td>Incorrect Address</td>
<td>Hardware / Software Failure</td>
<td>Program will jump to a random location in memory. May cause major problems.</td>
<td>Unpredictable.</td>
<td>CommandProc</td>
<td>297</td>
<td>This should be checked. Affect on variables cannot be predicted.</td>
</tr>
<tr>
<td>12.17</td>
<td>InbufCmdNumber</td>
<td>int</td>
<td>None</td>
<td>Command sent by IGU. (Position of bit set in Inbuf command byte).</td>
<td>Incorrect Value</td>
<td>Hardware / Software Failure</td>
<td>If not the command it should be then the wrong handling subroutine will be called.</td>
<td>Unpredictable.</td>
<td>CommandProc</td>
<td>297</td>
<td>May end up in a random place in code. This value should be checked. Used as offset into CommandProcSubsTbl.</td>
</tr>
</tbody>
</table>