

# 2003 FAA National Software Conference

## Unreached Code

### **BREAKOUT SESSION**

### **“Unreached Code Brainstorm”**

to discuss

Unreached/Dead/Deactivated/Defective/... Code

*FAA National Software Conference  
Reno, NV  
September 17, 2003*

*Facilitator - Robin Sova  
Phone 816-329-4133, robin.sova@faa.gov  
FAA Small Airplane Directorate, Kansas City, MO*

1

### **Brainstorm – Why & How?**

- Unreached is not defined in RTCA/DO-178B, thus Deactivated is often used in its place.
- Brainstormed ideas will be considered in the development of a \*CAST Position Paper.
- Concentrate on the topic, defining the issues, discussing the terminology, and establishing a baseline of understanding.

*\* Certification Authorities Software Team (international representation)* 2

# 2003 FAA National Software Conference

## Unreached Code

### **Brainstorm Session - Approach**

- Assumptions
- Scope the Issue
- Definitions
- Thoughts/Discussions?
- Understandings/Agreements?

3

### **Assumptions**

- In developing any regulatory or other forms of guidance regarding this topic, we should adhere closely to the DO-178B definitions.
- However, we should have the common sense to recognize any definition ambiguities or shortcomings and accept other Standards' definitions when necessary and appropriate.

4

# 2003 FAA National Software Conference

## Unreached Code

### Scope the Issue

- Unreached/Unreachable Code (a difference?)
- Unexecutable Code
- Defective Code (by requirement, design, code)
- Dead Code
- Deactivated/Disabled Code (a difference?)
- Defensive Programming & Exception Handling Code

5

### Definitions

- Baseline (from RTCA/DO-178B)
- Related to topic (also DO-178B)
- Other Standards
- Proposed

6

# 2003 FAA National Software Conference

## Unreached Code

### Baseline/DO-178B: Software

- (Glossary) “Computer programs and, possibly, associated documentation and data pertaining to the operation of a computer system.”

7

### Baseline/DO-178B: Code

- (Glossary) “The implementation of particular data or a particular computer program in a symbolic form, such as source code, object code or machine code.” *{RS – Definition covers all representations of program logic/data, from high level source thru assembly to machine language. It implies a difference in form between source code (i.e., a high level or assembly language input to a compiler/assembler per next DO-178B definition), object code, and machine code.}*

8

# 2003 FAA National Software Conference

## Unreached Code

### **Baseline/DO-178B: Source Code**

- (Glossary) “Code written in source languages, such as assembly language and/or high level language, in a machine-readable form for input to an assembler or a compiler.”

9

### **Baseline/DO-178B: Object Code**

- (Glossary) “A low-level representation of the computer program not usually in a form directly usable by the target computer but in a form which includes relocation information in addition to the processor instruction information.” *{RS – Assembly language code (mnemonics) or assembled code (binaries)? 1) both are lower-level, 2) neither is directly usable by a target computer until assembled and linked, 3) both contain processor instructions in one form or another, 4) but only assembled code includes relative relocation information; thus, this definition would exclude assembly language code.}*

10

# 2003 FAA National Software Conference

## Unreached Code

### **Baseline/DO-178B: Compiler**

- (Glossary) “Program that translates source code statements of a high level language, such as FORTRAN or Pascal, into object code.” *{RS – This definition leaves open the possibility that assembly language code might be considered object code since it could be the product of a compiler translating from a high level source language. However, assembled (binary) code is the usually expected output of a compiler, as opposed to assembly language code.}*

11

### **Baseline/DO-178B: Executable Object Code**

- (Section 11.12) “The Executable Object Code consists of a form of Source Code that is directly usable by the central processing unit of the target computer and is, therefore, the software that is loaded into the hardware or system.” *{RS – Somewhat ambiguous to state this is a “form of Source Code” (based on the earlier definition implying assembly or higher level language status) but it does correctly imply that EOC is synonymous with compiled or assembled, linked/loaded, machine code since it is “directly usable by the central processing unit...”}*

12

# 2003 FAA National Software Conference

## Unreached Code

### Related/DO-178B: Deactivated Code

- (Glossary) “Executable object code (or data) which by design is either (a) not intended to be executed (code) or used (data), for example, a part of a previously developed software component, or (b) is only executed (code) or used (data) in certain configurations of the target computer environment, for example, code that is enabled by a hardware pin selection or software programmed options.” *(RS – Definition addresses only Executable Object Code level, regardless of deactivation at source, assembly, or object levels.)*

13

### Related/DO-178B: Dead Code

- (Glossary) “Executable object code (or data) which, as a result of a design error cannot be executed (code) or used (data) in a operational configuration of the target computer environment and is not traceable to a system or software requirement. An exception is embedded identifiers.” *{RS – Unreachable code, not required or able to be executed, is the only type of code meeting this DO-178B definition. Therefore, code intended to execute (i.e., there is a requirement), yet which doesn't due to a design/coding error, is not dead but is unreachable due to its defective nature.}*

14

# 2003 FAA National Software Conference

## Unreached Code

### Other Standards: Machine Code

- (IEEE) “Computer instructions and definitions expressed in a form [binary code] that can be recognized by the CPU of a computer. All source code, regardless of the language in which it was programmed, is eventually converted to machine code. Syn: object code.” *{RS – This definition implies by synonym that object code is machine code (i.e., assembled code) and therefore, that assembly language code is not object code.}*

15

### Other Standards: Object Code

- (NIST) “A code expressed in machine language [“1”s and “0”s] which is normally an output of a given translation process that is ready to be executed by a computer. Syn: machine code. Contrast with source code. See: object program.” *(RS – This definition also states directly by definition and indirectly by synonym that object code is the product of an assembler/compiler and therefore, assembly language code is not object code.)*

16

# 2003 FAA National Software Conference

## Unreached Code

### Other Standards: Object Program

- (IEEE) “A computer program that is the output of an assembler or compiler.” *(RS – By stating that an object program is the product of an assembler or compiler, it seems to imply that assembly language code cannot be considered object code since it is the input to an assembler.)*

17

### Proposed: Unexecutable Code

- (RS) “Code (of any form) which for any reason will not lead to execution, whether intended or not by requirements, or due to design or coding errors, or due to language, compiler or assembler optimizations.”
- Implied by 6.4.4.3: is due to requirements or test shortcomings, or Dead or Deactivated code.
- Unexecutable can be decomposed at next level to “Dead” (per DO-178B – has no requirement) and “Unreachable” (proposed – a requirement exists).

18

# 2003 FAA National Software Conference

## Unreached Code

### Proposed: Unreachable Code

- (RS) “Code (of any form) that has a requirement to exist but will not lead to execution, either intentionally by requirement or unintentionally due to design or coding errors. At runtime, in Executable Object Code form, it is unreached.”

19

### Proposed: Unreachable (cont'd)

- Unreachable could be decomposed at the next lower level into 4 code categories:
  - Deactivated (EOC per DO-178B definition)
  - Defective (erroneous code, may be or may cause unreachable code)
  - Disabled (defective code that is purposely made unreachable as a temporary fix)
  - Defensive Programming & Exception Handling

20

# 2003 FAA National Software Conference

## Unreached Code

### **Proposed: Defective Code**

- (RS) “Code (of any form) that does not satisfy its requirements and/or contains errors; it may or may not have been verified or had structural coverage achieved.”

21

### **Proposed: Disabled Code**

- (RS) “Defective code (of any form) which has intentionally been made unreachable at some level, thus being rendered unexecutable.”

22

# 2003 FAA National Software Conference

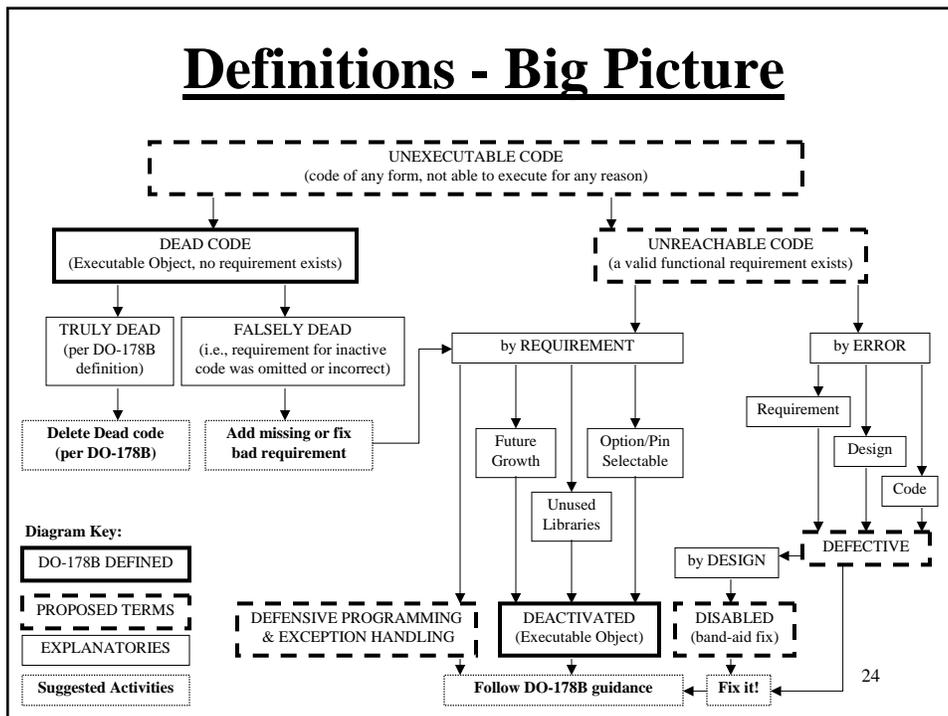
## Unreached Code

### Proposed: Defensive Programming & Exception Handling Code

- Is there a problem here which needs addressing?

23

### Definitions - Big Picture



# 2003 FAA National Software Conference

## Unreached Code

**Thoughts/Discussion?**

25

**Understandings/Agreements?**

26

# **2003 FAA National Software Conference Unreached Code**

**Thank You for  
Your Attendance and  
Participation!**

27

**Following slides are for backup  
and discussions only; not to be  
printed or distributed as  
handouts or included with  
meeting minutes except when  
used during the session.**

28

# 2003 FAA National Software Conference

## Unreached Code

### Example Scenario-Unreachable

**Requirements: 1) Count & Print from 0-1000, inclusive, by 2's  
2) Do remaining task**

```
** Initialize counter to its first value of "zero" and then print
   counter = 0
   print counter
*** Increment counter by 2's up to 1000, printing each value
   do until counter = 1001
       counter = counter + 2
       print counter
   end do
*** Unreachable code follows because the counter attains values of 1000
*** & 1002 but never 1001, so stuck in infinitely increasing counter loop
   do remaining task
   end do
```

*(Requirements OK; design/coding error in meeting requirement #1 causes infinite counting loop and makes requirement #2 code Unreachable.)*

29

### Example Scenario-Dead

**Requirement: 1) Count & Print from 0-1000, inclusive, by 2's**

```
** Initialize counter to its first value of "zero" and then print
   counter = 0
   print counter
*** Increment counter by 2's up to 1000, printing each value
   do until counter = 1000
       counter = counter + 2
       print counter
   end do
*** Dead code follows because of goto & no existing requirement
   go to end
       do remaining task
       end do
   end
```

*(Requirement #1 is OK and is met; go to causes skipping of code block which has no corresponding requirement anyway, therefore it is Dead code.)*

30

# 2003 FAA National Software Conference

## Unreached Code

### Example Scenario-Deactivated

**Requirements: 1) Count & Print from 0-1000, inclusive, by 2's  
2) Build in future option for remaining task**

```
** Initialize counter to its first value of "zero" and then print
   counter = 0
   print counter
*** Increment counter by 2's up to 1000, printing each value
   do until counter = 1000
       counter = counter + 2
       print counter
   end do
*** Deactivated code follows because of future growth option requirement
   go to end
       do remaining task
       end do
   end
```

*(Requirements are OK, code meets both; goto prevents execution of future option code which is required for growth, therefore it is Deactivated code.)*

31

### Example Scenario-Defective

**Requirement: 1) Count & Print from 0-1000, inclusive, by 2's**

```
** Initialize counter to its first value of "zero" and then print
   counter = 0
   print counter
*** Increment counter by 2's up to 1000, printing each value
   do until counter = 1001
       counter = counter + 2
       print counter
   end do
*** Defective code in the counter above causes Unreached code to follow
*** due to seeing values of 1000 & 1002 but never 1001, stuck in infinite loop
       do remaining task
       end do
```

*(Requirement is OK; design error in meeting requirement causes infinite counting loop preventing reaching the unrequired code, therefore it is Defective code.)*

32

# 2003 FAA National Software Conference

## Unreached Code

### Example Scenario-Disabled

**Requirement: 1) Count & Print from 0-1000, inclusive, by 2's**

```
** Initialize counter to its first value of "zero" and then print
   counter = 0
   print counter
*** Increment counter by 2's up to 1000, printing each value
   go to end
   do until counter = 1001
       counter = counter + 2
       print counter
   end do
   end
```

*(Requirement is OK; design/coding error (i.e., defective code) in meeting requirement causes infinite counting loop, which is avoided by inserting goto/end thus creating unrequired deactivated defective code block, or as now proposed to be called, Disabled code. Obviously, the original requirement is not met either.)*

33

### Proposed: Activated Code

- (WS) “Airborne code that has been fully verified and is active in the aircraft system configuration.” *(RS – I would recommend changing to “Airborne Executable Object Code which by design is intended to be executed or used, has been fully verified, and is active in the aircraft system configuration.”)*

34

# 2003 FAA National Software Conference

## Unreached Code

### **Proposed: Defective Code**

- (WS) “Code which it is known, does not satisfy its requirements and/or contains errors, may or may not have been verified and structural coverage achieved, but can be “disabled” for a specific aircraft system configuration.” *(RS – suggest deleting last portion since not all defective code is disabled; included in disabled definition.)*

35

### **Proposed: Unreachable Code**

- (WS) “Airborne code inserted by the computer that has not been fully R-B tested nor structural coverage achieved but has been analyzed and determined to meet its requirements and be harmless to leave in the airborne build.”

36

# 2003 FAA National Software Conference

## Unreached Code

### My Thoughts – Dead Code

- Based on DO-178B definition stating that dead code “is not traceable to a system or software requirement...a result of a design error” and section 6.4.4.3.c statement that “Dead...code should be removed and an analysis performed to assess the effect and need for reverification” - are there any good reasons or justification for keeping truly dead code?
  - One bad reason for leaving dead code in is performance related; the developer is not really 100% confident that the code is dead (e.g., due to spaghetti code nature) and its removal may lead to future problems.
  - Another bad reason is cost/schedule based; the developer may be 100% confident of “code deadness” but does not want to incur the added expense/time penalties for analysis and reverification activities required as a result of removing what they truly believe is benign.

37

### My Thoughts – Pt./Counter Pt.

One Proposed Approach - if activation of dead code is determined to have no detrimental safety effects, yet would be difficult to remove, then it may be allowed to remain.

Point - I'm not sure this position can be logically justified; i.e., it is illogical to accept the “safe outcome determination” of an analysis based on the “unintentional activation of dead code” when that activation itself implies failure of other code within the program, thus making the results of the analysis questionable. Therefore, if code is truly dead, whether or not there are safety/convenience reasons to keep it, there is no “logical” way to justify keeping it. For example, if analysis says its execution is unsafe - it must be removed; however, if analysis says execution would be safe, yet depends on accepting the predictability of the result based on a “presumably dead code” process executing – then removal would be the only guarantee against potentially detrimental safety effects.

Counterpoint – since we're assuming correct performance of the “live” code, without the presence of the dead code, then we cannot penalize the results of the safety analysis by now assuming the live code has failed in the presence of the dead code.

Conclusion – if analysis shows truly dead code would be safe even if executed, then it's permissible to keep the dead code.

38

# 2003 FAA National Software Conference

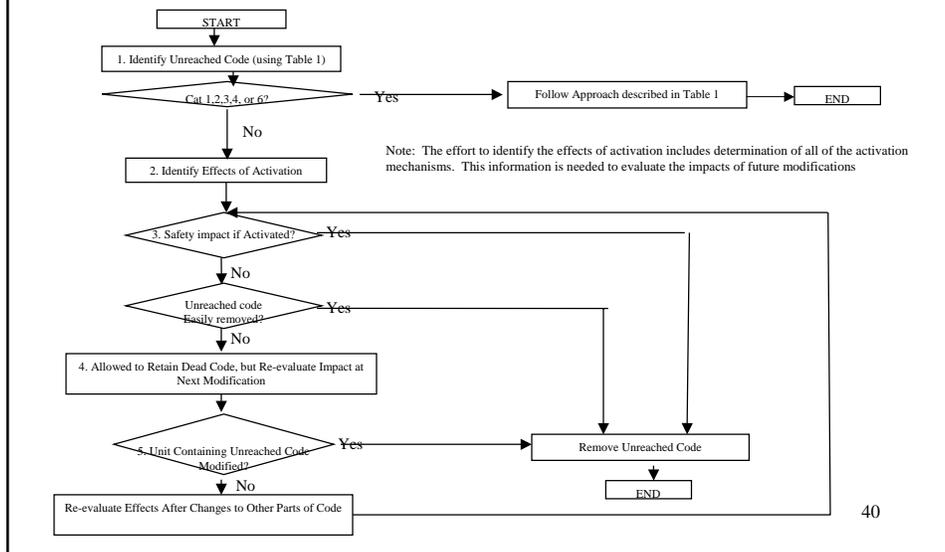
## Unreached Code

### My Thoughts – Disabled Code

- With respect to allowing disabled defective code to exist in an operational application: I do not think “broken” is a good case for claiming “deactivation” since the acknowledged state of brokenness implies a “known unknown” aspect! It’s risky enough in software development that we must accept the existence of unknown unknowns in code via the “warm fuzzy of process assurance,” but intentionally accepting a “bad known unknown” seems to be pushing our luck with a potential for dangerous consequences.

39

### Solutions? (suggested)



# 2003 FAA National Software Conference

## Unreached Code

### Solutions? (suggested-Table 1 )

Category of unreached code	Approach	Notes
#1. Code can be tested by requirements based tests at a lower level of integration, such as module tests	Test at the lower level of integration	It is important that all tests should be requirements based.
#2. Shortcomings in requirements based test cases or test procedures	<ul style="list-style-type: none"> <li>Follow the guidance in DO-178B 6.4.4.3.a.</li> <li>Investigate need for adapting the development process to prevent this in the future</li> </ul>	Basically, correct/enhance test cases/procedures
#3. Inadequacies in software requirements	<ul style="list-style-type: none"> <li>Follow the guidance in DO-178B 6.4.4.3.b.</li> <li>Investigate need for adapting the development process to prevent this in the future</li> </ul>	Basically, correct requirements
#4. Caused by incorrect development process	<ul style="list-style-type: none"> <li>Correct development process</li> <li>Check whether this was a problem in previous products that used this process</li> <li>Re-enter corrected development process, which should remove the unreached code</li> </ul>	In the DO-178B definition of dead code, it is not clear whether "design error" refers to a flaw in the development process, or an incorrect application of the development process. This is the difference between # 4 and # 5 in this table.
#5. Caused by incorrect application of the development process	<ul style="list-style-type: none"> <li>Apply proper development process, which should remove the unreached code</li> <li>Check whether this was a problem in previous products</li> <li>Investigate need for adapting the development process to prevent this in the future</li> </ul>	
#6. De-activated code	<ul style="list-style-type: none"> <li>Follow the guidance in DO-178B 6.4.4.3.d.</li> </ul>	Basically, show that it can not be inadvertently executed; or test the code using the requirements for the configuration in which the code is activated. May be there to address future expansion. Software included for future expansion is typically handled as deactivated code.
#7. Other causes	??	?? Examples ?? Defensive programming? Although I use the example myself, I would expect that at some level of integration (or lack of integration) the code can be tested. Inability to implement certain hardware failures.

41