IV&V Coverage of NASA Software Guidelines

Jacob Cox

2013 IV&V Workshop

Introduction

- This presentation presents the results of evaluating NASA IV&V static code analysis processes with respect to the NASA Software Standards checklist items.
- Attempts were made to identify:
  - What NASA IV&V does cover? *Quite a bit*
  - What NASA IV&V can cover? *Almost all*
  - What NASA IV&V should cover? *Most*
  - How can NASA IV&V cover the item?

What Was Looked At

- **NASA Software Safety Guidebook; NASA-GB-8719.13; 3/21/2004**
  - **Appendix H, Checklists**
    - **H.5, *Checklist of generic (language independent) programming practices***
    - ***H.8, Checklist of C programming practices for safety***
    - ***H.9, Checklist of C++ programming practices for safety***

The Standard and the Guidebook

- NASA-STD-8719.13 is mandatory for many projects IV&V supports, and 8719.13 is a non-mandatory expansion of the content of 8719.13
- From 7150.2
  - When a project is determined to have safety-critical software, the project shall ensure that the safety requirements of NASA-STD-8719.13, Software Safety Standard, are implemented by the project. [SWE-023]

Statements from NASA-GB-8719.13 GB

- a guidebook for assessing software systems for software's contribution to safety and techniques for analyzing and applying appropriate safety techniques and methods to software…
- The document: (provides)
  - good software engineering practices which contribute to software system safety.
  - means to scope and tailor the software safety and software engineering activities to obtain the most cost effective, best quality, and safest products.
  - analyses, methods and guidance which can be applied during each phase of the software life cycle.
  - development approaches, safety analyses, and testing methodologies that lead to improved safety in the software product.

General Thoughts

- Most checklist items are the same as those in other standards
- Most are obvious in the sense that they are taught in programming classes
- It is good to have an explicit list of good programming practices
- It is good to have rationale to refer to when writing issues

Methodology

Filled out an MS Excel spreadsheet with the following information:

Column Name	Intent
NASA Software Standards checklist item	The text of the checklist item from 8719.13
General Severity	An estimate of the general ORBIT issue severity for any IV&V TIM that may be generated based on the item
IV&V Covers with Code Analysis	IV&V currently writes issues with respect to the item
IV&V Can Cover	IV&V can write issues or risks based on the item
IV&V Should Cover	IV&V should write issues or risks based on the item
Rationale	Reason that IV&V should address the checklist item indicted
Can write an issue?	Sufficient data is explicitly provided to allow independent evaluation of the checklist item
Static Analysis	Can be found with Static Analysis tools
Simple Search Available	Text searches with regular expressions can find
Code Review	Requires the analyst to review the code manually
Other Analysis	Could be found with analysis other than either syntactic or semantic code analysis
Project Start	Is a systematic software development and verification/validation issue and the indicated checklist item is not specifically related to a given Software release or the resultant code

Sample From the Worksheet

Checklist	Severity	IV&V Covers with Code Analysis	IV&V Can Cover	IV&V Should Cover	Rationale	Atomic (can write an issue)	Static Analysis	Simple Search Available	Code Review	Other Analysis	Project Start
Minimize use of dynamic memory. Using dynamic memory can lead to memory leaks. To mitigate the problem, release allocated memory as soon as possible. Also track the allocations and deallocations closely.	risk	Yes	Yes	Yes	NA	No	No	Yes	No	No	No
Minimize memory paging and swapping. In a real-time system, this can cause significant delays in response time.	any	No	Yes	Maybe	Architecture Analysis	No	No	No	No	Yes	No
Avoid goto's. Goto's make execution time behavior difficult to fully predict as well as introducing uncertainty into the control flow. When used, clearly document the control flow, the justification for using goto's, and thoroughly test them.	4	Yes	Yes	Yes	NA	Yes	No	Yes	No	No	No
Minimize control flow complexity. Excessive complexity makes it difficult to predict the program flow and impedes review and maintenance. Project guidelines or coding standards should set specific limits on nesting levels.	4	Yes	Yes	Maybe	Project Interest	No	Yes	No	No	No	No

Observations

- Would other analysts always agree with the entry choices? **No**
- Is it possible that there were ORBIT issues that I was unable to find showing that NASA IV&V covered a particular item? **Yes**
- Were the checklist items always properly interpreted? **Probably Not**

ORBIT is NASA IV&Vs issue tracking system

Checklist Items

- 120 checklist items (*Item text is italicized*)
- Some can be found with multiple types of analysis
- There is duplication and partial overlap amongst some items
  - *Explicitly define class operators (assignment, etc.). Declare them private if they are not to be used*
  - *For all classes, define the following: Default constructor, copy constructor, destructor, operator=*

Checklist Items Continued

- Most items are not absolute e.g. “avoid” “minimize”
- Some are “do” statements
  - *Minimize control flow complexity*
- Some are “do not” statements
  - *Avoid goto’s*
- Most of the C++ items are written as implied “be aware statements”
  - *Error associated with Functions - Unwanted side effects in a function*

?? Items

- There were 7 checklist items that were phrased in a way that it would be possible to interpret them in multiple ways and they were left out of the analysis.
- Examples:
  - *Error associated with Functions - Improper use of the same function for assignment and evaluation*
  - *Error associated with Functions - Improper use of built in functions and/or compiled libraries*

Severity Summary

4	non-runtime issue	44
<=3	run time fault of some kind	48
any	could be minor or severe	8
NA	a compiler error	1
risk	a program risk	12

Severity Examples

- <=3
  - *Error associated with Control Flow - Incorrect precedence assumptions*
- 4
  - *Use single purpose functions and procedures. This facilitates review and maintenance of the code.*
- any
  - *Error associated with Variables - Incorrect use of global variables*
- risk
  - *Use version control tools (configuration management)*

IV&V Coverage with Code Analysis

Partial		4
Inconsistent		2
No	No SA issues found in ORBIT	74
Yes	Static Analysis issues found	33

- Partial
  - *Error associated with Control Flow - Interface errors (e.g., inaccurately ordering or reversing the order of parameters passed to a function)*
- Inconsistent
  - *Use #define instead of numeric literals. This allows the reader or maintainer to know what the number actually represents (RADIUS_OF_EARTH_IN_KM, instead of 6356.91). It also allows the number to be changed in one place, if a change is necessitated later.*
- No
  - *Be careful when using operator overloading. While it can help achieve uniformity across different data types (which is good), it can also confuse the reader (and programmers) if used in a non-intuitive way.*

IV&V is Capable of Covering

No	IV&V cannot cover	2
Yes	IV&V can cover	111

- IV&V Can’t Cover
  - *Error associated with Control Flow - Parameters of incompatible type with the function prototype*
  - *Limit the number and size of parameters passed to routines. Too many parameters affect readability and testability of the routine. Large structures or arrays, if passed by value, can overflow the stack, causing unpredictable results. Always pass large elements via pointers.*

IV&V Should Cover

Difficult	Large amount of manual effort	35
Maybe	Project dependent	4
No	IV&V should not cover	4
Yes	Should be covered by IV&V	70

- Difficult
  - *Error associated with Variables - Reuse of variables without reinitialization*
- Maybe
  - *Minimize control flow complexity*
- No
  - *use single entry and exit points in subprograms*

TIMS can be Written

No	Analysts cannot write issues	16
Yes	TIMs can be written	97

- No
  - *Minimize dynamic binding. Dynamic binding is a necessary part of polymorphism. When used, it should be justified. Keep in mind that it causes unpredictability in name/class association and reduces run-time predictability.*
  - *Create coding standards for naming, indentation, commenting, subprogram size, etc. These factors affect the readability of the source code, and influence how well reviews and inspections can find errors.*

Can be Found with Static Analysis

No	Cannot be found with SA	88
Sometimes	Elements can be found with SA	2
Yes	Can be found with SA	23

- No
  - *Use single purpose functions and procedures. This facilitates review and maintenance of the code.*
- Sometimes
  - *Check input data validity. Checking reduces the probability of incorrect results, which could lead to further errors or even system crashes. If the input can be "trusted", then checking is not necessary.*
- Yes
  - *Declare the destructor virtual. This is necessary to avoid problems if the class is inherited.*

Can be Found with Text Searches

No	Text searching cannot find	94
Yes	Can be found with text searches	19

- No
  - *Use const variables and functions whenever possible. When something should not change, or a function should not change anything outside of itself, use const.*
- Yes
  - *When using switch…case, always explicitly define default.*
  - find rootDirectory/ -name '*.c*' | xargs ggrep -E 'default|switch' {} | less

Can be Found with a Code Review

No	Code review cannot find	63
Yes	Can be found with a code review	50

- No
  - *Provide adequate precision and accuracy in calculations, especially within safety-critical components.* **(Design Analysis)**
- Yes
  - *Error associated with Control Flow - Interface errors (e.g., inaccurately ordering or reversing the order of parameters passed to a function)*

Non-Implementation Analysis

No	Best found in implementation	84
Yes	Found with other IV&V analysis	29

- No
  - *Error associated with Variables - Over estimation of predefined type's size.*
  - *Do not use ++ or – operators on parameters being passed to subroutines or macros. These can create unexpected side effects.*
- Yes
  - *Error associated with Memory - System running on unreliable data* (**Dynamic Testing or Data Validation**)
  - *Explicitly define class operators (assignment, etc.). Declare them private if they are not to be used.* (**Design Analysis with OO Diagrams**)

Found at Project Start

No	Found during project execution	109
Yes	Can be found at start of project	4

- Yes
  - *Use version control tools (configuration management)*
  - *Enable and read compiler warnings. If an option, have warnings issued as errors.*
  - *Utilize a bug tracking tool or database*
  - *Use data typing. If the language does not enforce it, include it in the coding standards and look for it during formal inspections*

Next Steps

- Klocwork and Flexelint warnings for each applicable checklist item
- Regular expressions for searchable items

Conclusion

- IV&V can cover almost 93% of the checklist items; 111 of 120 (including the 7 ?? Items)
- Static Code analysis and text searching cover 45 % of checklist items for which TIMs could be written; 44 of 97

Next Conclusion

- Of the 70 Checklist items IV&V "should" cover:
  - 36 can be covered with static code analysis
  - 16 can be covered with text searches
  - 14 can be covered with code reviews
  - 36 can be covered with other types of analysis
  - 3 can be addressed during project startup

Note that there is significant overlap

- Of the 70, IV&V already covers 33 of the checklist items

Final Conclusion

- NASA IV&V should develop process assets and modify methods in the NASA IV&V Catalog of Methods to insure that all the checklist items that should be covered are covered.