Cost Effective Applications of High Integrity Software Processes

John H. Robb
Embedded Software Engineer Principal
Lockheed Martin Aeronautics Fort Worth
John.H.Robb@lmco.com
817.935-4648
18 May, 2011

AER201103026

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
MAY 2011		**00-00-2011 to 00-00-2011**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Cost Effective Applications of High Integrity Software Processes	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Lockheed Martin Aeronautics Fort Worth,1 Lockheed Boulevard,Fort Worth,TX,76108**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
Presented at the 23rd Systems and Software Technology Conference (SSTC), 16-19 May 2011, Salt Lake City, UT. Sponsored in part by the USAF. U.S. Government or Federal Rights License

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**26**	

Not All Lessons Learned Are Equal…

"Managing software developers is like herding cats" –
Various (disgruntled managers)

"A man who carries a cat by the tail learns something he can learn in no other way" - Mark Twain

Motivation

- **This presentation looks at the practical lessons learned in applying High Integrity software processes both domestically and internationally**
  - ***High Integrity software is like landing a man on the moon within several feet of the target area***
  - ***Most of the industry would find landing on the moon to be more than sufficient (and cost effective)***

This image is in the Public Domain

- **The goal of this presentation is to discuss practical (cost effective) approaches that provide good (enough) coverage**

Why Should I Care About High Integrity Software?

What is High Integrity software?

- ***Either Safety or Security critical software***
- ***Software that cannot fail because it is doing something so critical the consequence of failure is very high***

What characterizes High Integrity software?

- ***Very high reliability of the software***
- ***Robust processes are used to assure the software achieves its reliability objectives***
- ***Very costly to develop, verify and certify, but very inexpensive to maintain***

How does this contrast with typical industrial software needs?

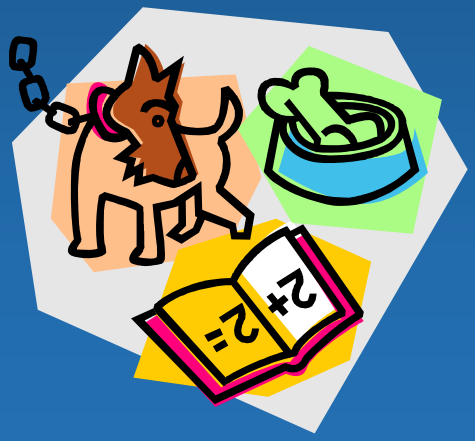
High Integrity Software Practices

- **Requirements**
- **Inspections/peer reviews**
- **Checklists**
- **Programming Languages and Coding Standards**
- **Static Code Analysis**
- **Code complexity**
- **Unit Testing**
- **Automated Testing**
- **Qualification Testing Cycle Time**

Experience Based – Not a Recipe

We’re going to discuss lessons learned from each of these basic tenets and what they may mean to you – this is not intended to be a formula but recommendations

Recipe - “a series of step-by-step instructions for preparing ingredients you forgot to buy, in utensils you don't own, to make a dish the dog won't eat." - Anonymous



This image is in the Public Domain

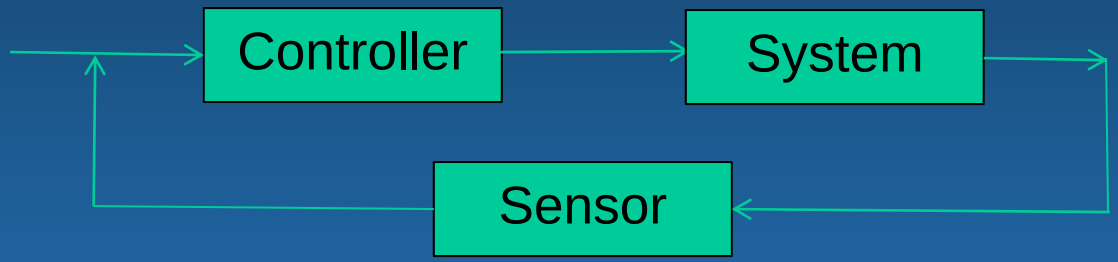
Requirements Development - Some Thoughts

- **Complex large system requirements development is not easy**
  - ***It is an arbitrary slice of abstraction for understanding by the customer, the developers and the testers***
  - ***Each requirement is represented by several design classes and several hundred lines of code***
  - ***Each requirement has its own context and specific knowledge space***
  - ***It is typically specified in English which is not very precise***
- **Despite the numerous requirements specification and elicitation approaches and techniques**
  - ***It still remains one of the most elusive areas of software process improvement***

Requirements Development - Observations

Maturation of requirements is a key concept

- ***The most successful areas seem to use working models in parallel to requirements development***
  - Either through specific models or thru prototypes (e.g., Agile)
- ***Working models tend to faithfully emulate both the controller (software) and the system (e.g., vehicle)***



- ***What seems to distinguish the quality of a model is not the quality of the controller model but of the system model***
  - When the system model is complicated a modeling process (e.g. MBD) seems to work better than a prototype process
- ***A good functional model is still very much needed even in the UML paradigm***

Requirements Development - Recommendations

- **When the system model is complex**
  - ***Use a model based approach to mature the requirements***
  - ***This doesn't necessarily mean that the model must generate executable target code***
- **When the system model is not complex**
  - ***Utilize a prototype approach (Agile or Spiral)***
  - ***The customer/user in essence becomes the system model***
- **In either case, the model or prototype doesn't necessarily have to be the deliverable product**
  - ***The main consideration is reduction of rework, not in auto-generation or reuse of code (which is relatively cheap)***
  - ***Typically, it is better to have software engineers develop the production target for long term maintenance reasons***

The Eye of The Beholder…

Many things are a matter of perspective

Ponton: I consider her the most beautiful woman in the world… What about yourself?

Inspector Clouseau: No, I don't consider myself a beautiful woman

We can become too familiar with the beauty of our own product

One very effective way to achieve this is to have someone else observe our work and critique it from their own perspective

Inspection/Peer Reviews

- **Reduce costly rework**
  - ***Focus on defect removal – rework can cost up to 20x the cost of correcting the same problem during an inspection/review***
  - ***Good software development teams use inspections (peer reviews) to remove up to 80 percent of their defects***

- **It doesn't have to be hard**
  - ***Reviews can be of many different types (very formal inspections all the way to peer reviews)***
  - ***The key is to have adequate preparation, expert participation, and good leadership ("moderator")***
  - ***If you can't review all the products start at the beginning of the life cycle first (i.e., requirements are the most important)***
  - ***Reviews are so important that they should be built into the schedule/budget and should drive the initial schedule***
  - ***Do not sacrifice reviews without first presenting a business case to program management***

Checklists and Using Them

- **Checklists**
  - ***Pilots use them for pre-flight and pre-landing***
  - ***Used for surgery and other complicated medical procedures***
  - ***They are consistently cited as a best practice for software development (e.g. peer reviews)***
  - ***Yet they are still not widely used***

- **Lack of use prevents several effective things from happening:**
  1. ***Defect prevention (checklist updated from experience)***
  2. ***Training of new engineers***
  3. ***Audit records and proof of compliance***
  4. ***General process improvement through sharing of checklists***

Checklists and Using Them (cont.)

- **Best thing to do is to create checklists and require them as entry criteria for peer reviews (inspections)**

- **If you don't have checklists then create some**
  - ***Borrow from industry best practice checklists***
  - ***Survey your company to determine if specific projects already have them***
  - ***If you are fortunate enough to have a common software resource group see if they have them***

What About Languages?

“Why can a man never starve in the Great Desert?
Because he can eat the sand which is there.

But what brought the sandwiches there?

Why, Noah sent Ham and his descendants mustered and bred”

- Richard Whately, Archbishop of Dublin

Then you should say what you mean," the March Hare went on.

"I do," Alice hastily replied; "at least--at least I mean what I say--that's the same thing, you know."

"Not the same thing a bit!" said the Hatter. "You might just as well say that 'I see what I eat' is the same thing as 'I eat what I see'!“

- Lewis Carroll, Alice in Wonderland

Programming Languages and Coding Standards

- **Just like verbal languages programming languages also have idiosyncrasies**
  - ***Features that reduce or eliminate determinism***
    - Exception handling
    - Object lifetime (constructors, destructors)
    - Tasking
    - Memory allocation/deallocation
    - Dead and/or deactivated code
    - Type conversions and numerical representations (e.g. NaN)
  - ***Non-determinism in software development is the same as a surprise (and these rarely uncover buried treasure)***
    - Spend a little time considering some of these features and develop a simple coding standards to address these
    - The coding standards document should not be a dissertation
    - Try to automate as many checks as possible

An Extra Set of Eyes for Code Reviews

- **Static Code Analysis (SCA) - the static analysis of computer programs – typically performed by a tool**

- **SCA tools are used to detect the following**
  - ***Dead/unreachable/unfeasible code***
  - ***Uninitialized variables/null-pointers/Divide by zeroes***
  - ***Buffer underflow/overflow***
  - ***Language syntax errors/known vulnerabilities***
  - ***Expert best practices***

- **SCA tools have several advantages**
  - ***Streamline code reviews (to focus code reviews on semantics)***
  - ***Train new programmers (enforce language standards)***
  - ***View them as a advanced compiler***

Smart Use of SCA Tools

- **Good things to consider**
  - ***Develop programming standards***
  - ***Use SCA tools as entry criteria for peer reviews of initial code baseline (training)***
  - ***Best to have a common resource group own the SCA tool and standards (share start-up costs) and to develop training***

- **Things to avoid**
  - ***Underestimating the number of false positives – you will need a dedicated resource to develop a good "true positives" report***
  - ***Running a "quickie" SCA tool check on an existing baseline – analysis of false positives will take a while (maybe weeks)***
  - ***Running a SCA tool check without having programming standards***
  - ***Using an SCA for any other event than a peer review entry gate***

On Simplicity…

“’Think simple’ as my old master used to say - meaning reduce the whole of its parts into the simplest terms, getting back to first principles.” – Frank Lloyd Wright

“Newton was a genius, but not because of the superior computational power of his brain. Newton's genius was … so that it became, in some measure, tractable to the brains of perfectly ordinary men.” - Gerald M. Weinberg

Code Complexity - Cyclomatic and Essential Complexity

- **Overly complex modules are**
  1. ***More prone to error***
  2. ***More difficult to understand***
  3. ***More difficult to test***
  4. ***More difficult to modify***

This image is in the Public Domain

- **Cyclomatic and essential complexity measures provide a way of developing modules that avoid these issues**
  - ***Cyclomatic complexity measures the amount of decision logic in a single software module***
  - ***Essential complexity quantifies the extent to which software is unstructured***

- **Tools (including SCA tools) can provide both of these measures – use them prior to submitting code for review**
  - ***You will need to establish a threshold for complexity first***

Don't Build Upon a House of Cards

- **The best way to avoid building upon a house of cards is**

This image is in the Public Domain

- **To build upon a strong foundation - unit testing is that first foundation**

Unit Testing – Building a Strong Test Foundation

- **Unit Testing – testing of the smallest part of an application to make it fit for later test**
  - ***Required for High Integrity software***
  - ***Cornerstone of the Agile (Extreme Programming) process***
  - ***Will require either a unit test harness or an unit test framework (automated test tool)***

- **Unit Testing completion criteria**
  - ***For High Integrity software this is typically either full source or full object (MC/DC) – can be very expensive***
  - ***For Agile the objective is to "test everything that can possibly break"***

- **For High Integrity software most of the cost is incurred in**
  - ***Achieving initial coverage requirements***
  - ***Updating unit test cases and re-testing with each code change***

Unit Testing - Testing From the Bottom-Up (cont.)

- **Practical recommendations**
  - ***Test each function both nominal and stress cases – do not try to achieve coverage required for High Integrity software***
  - ***Require test to be performed prior to code peer review***
  - ***Perform test on each code baseline and perform peer review on code changes***
  - ***May want to look at cyclomatic complexity and size when deciding what to unit test***
    - Complexities of less than 3 to 4 may not require unit testing (depending upon the size) and may only require a peer review
  - ***Object oriented design/coding also affects unit testing strategies***
    - Higher complexities/larger size tend to go toward the “controller” classes which warrant unit testing
    - Test complexity is shifted more toward integration (of controllers and methods and with other controllers) than unit testing

Automation of Test

- **Excellent and poor reasons to automate**
  - ***Because it sounds impressive***
  - ***Because management directed it***
  - ***To reduce the ever growing backlog***
  - ***There are many repetitive tests and/or data driven tests***
  - ***To reduce time spent performing regression testing and you expect to perform regression testing quite a bit***

This image is in the Public Domain

- **Test tasks to automate**
  - ***Any test that is highly repetitive with little expected change***
  - ***To produce a consistent test cycle time***
  - ***To reuse tests***

- **Typically these lead to automation of Functional, Regression, Stress and Performance tests**

Automation of Test (cont.)

- **Some tests may not be good candidates for automation**
  - ***User Interface (when very volatile and automation doesn't offset the cost of change)***
  - ***One of a kind tests (e.g., destructive tests)***
  - ***Tests against a specific baseline that are not intended to be repeated***
  - ***Tests that have a near-term deadline where automation is too expensive to create and/or tools don't exist***
- **Automation decisions need to be driven by engineering economics**

Qualification Testing Cycle Time

- **Eventually after initial product delivery you will make an amazing discovery**
  - ***Quick turn updates are driven by the time it takes to implement/test the update and to ensure that no side affects have occurred***
  - ***Typically the latter (qualification test) takes the longest and drives the entire quick turn cycle***

- **It is prudent to establish early in the development a reasonable qualification test cycle**
  - ***Identify what functions will be part of the regression test suite***
  - ***Identify how long the test cycle should take (and estimate how long it might take)***
  - ***Automate what drives the qualification cycle (if possible)***
  - ***Look at trading parallelism between sequential activities for slightly increased risk (e.g., starting test with in intermediate product)***

This image is in the Public Domain

Summary

We've discussed some of the tenets of High Integrity software and how they could be applied to your domain

- ***Requirements***
- ***Inspections/peer reviews***
- ***Checklists***
- ***Programming Languages and Coding Standards***
- ***Static Code Analysis***
- ***Cyclomatic complexity***
- ***Unit Testing***
- ***Automated Testing***
- ***Qualification Testing Cycle Time***

The rest is up to you

This image is in the Public Domain