Reliable Software SOW

- ann.neufelder@missionreadysoftware.com
- http://www.missionreadysoftware.com

About the Presenter

- Chairperson of IEEE 1633 Recommended Practices for Software Reliability Working Group (2016 edition)
- 39 years of SW engineering and SW reliability experience
- Authored NASA's Software FMEA training webinar
- Authored NASA's Software FTA training webinar
- Authored NASA's Software Analyses training materials
- Authored Intel's Software Vendor Assessment
- Has taught Software Reliability to more than 5000 people
- Co-authored USAF Rome Laboratory "System and Software Reliability Assurance Notebook", with P. Lakey, Boeing Corp.
- Authored "Ensuring Software Reliability", Marcel-Dekker, 1993.
- Authored "Effective Application of Software Failure Modes Effects Analysis", published for CSIAC, 2014.
- Performed 100+ software reliability analyses on real software systems
- U.S. Patent 5,374,731 for a predictive model

Agenda

Purpose: Provide a cookbook for quantitative software reliability SOW specifications including language, expected deliverables

- SOW language for
  - Software reliability plans, reports, assessments, measurements, analysis
    - Reliability SOW
    - Inclusion of software in Section L
    - CDRLS and 1423
  - Why the task is needed
  - Tailoring guidance for size of program, type of program, Agile development, maturity of software
  - SOW language for specific software reliability thresholds (i.e. MTBF, availability, etc.)
  - Section L and M

Lessons Learned

Reliable Software SOW

- Required system reliability is not meeting specifications because of software failures.
- DoD is finding out far too late in development and test that system requirements are not being met due to the software.
- Software intensive systems often times have too many restarts, resets, and/or reboots which collectively cause the system to be down longer than required.

Goals

Reliable Software SOW

- Provide insight into the software development artifacts and activities so that the Government can independently assess both the software artifacts and the contractor's ability to make the software mission ready.
- Define acceptable system metrics supported by Reliability and Maintainability (R&M) to measure and evaluate (define how software related failures impact current R&M system metrics).
- Implement effective R&M requirements and metrics into software development programs that are employing Development, Security, and Operations (DevSecOps).
- Contract for software reliability and effectively evaluate the risks of contractor's proposal to achieve software reliability.
- Differentiate roles, responsibilities, and interactions of reliability, software, and systems engineering.
- Provide for a contractual means for using lessons learned for reliable design to build software that is more failure resistant and fault tolerant.
- Reduce the occurrence or impact of software failures during operation.

Applicability

Reliable Software SOW

- Weapon and combat systems, and the mission systems that support weapon and combat systems that have software
- This guidance is not intended for or use with enterprise or business systems acquisitions
- Most, if not all, modern weapon and combat systems have software
- The reliability engineer can determine from the software engineering counterpart if the system has software

The SOW language and guidance

Report No. FCDD-AMR-MR-22-08

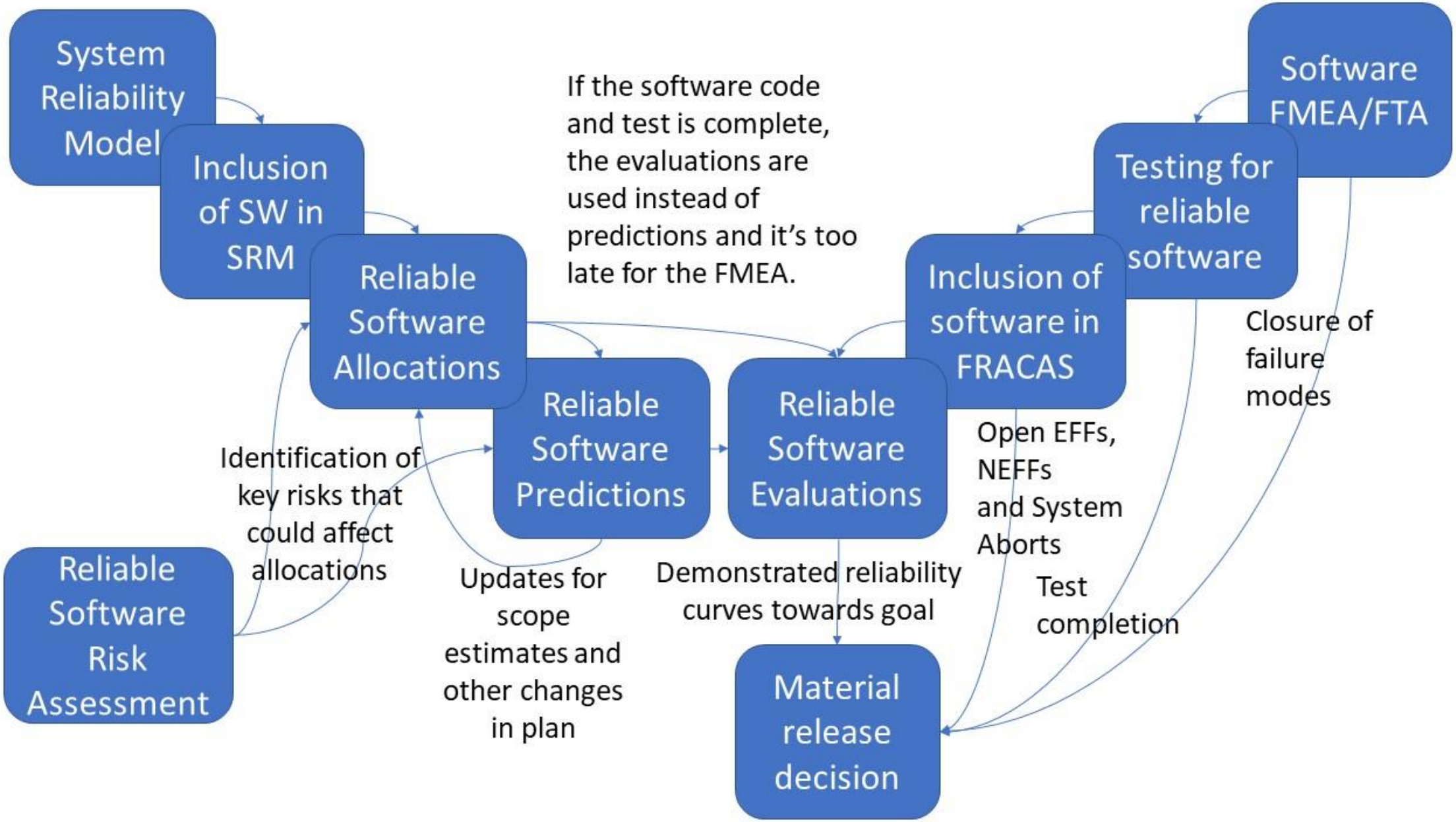
- The Reliable Software SOW language written by Assessment Division, Systems Readiness, Directorate, Combat Capabilities Development Command, Aviation and Missile Center
- The SOW language may/will be incorporated or adapted by other DoD services
- The SOW language covers:
  - The reliable software tasks to be delivered in working groups/CDRLs
  - When the tasks must be completed
  - References to acceptable sources for conducting the tasks
  - DiDs
  - Cross matrix to software development artifacts such as software development plan (SDP), Software Requirements Specifications (SRS), Software Design Document (SDD), Software Test Plan (STP), Software Test Description (STD), Software Test Report (STR
- Guidance for RAM personnel to coordinate with Government Software Engine

Selecting the Appropriate Reliable Software Tasks

Sometimes it's too late in the program for some of the software reliability tasks

Reliable software tasks and why you need them

Element	Why you need it
Software reliability program plan	The Software RPP is the document that details the Contractor's planned efforts in order to achieve the software reliability requirements and to ensure the integration of software reliability activities into systems engineering.
Inclusion of software in the system reliability model	Identifies how the software reliability is merged with the hardware reliability to yield a system reliability model
Software reliability Allocation	Ensures that the software is allocated part of the system reliability objective
Software reliability Prediction	The models that the contractor predicts software reliability early in development
Software reliability Evaluation	Ensures that the contractor is demonstrating that the actual software reliability in testing is trending towards the allocation
Software FMEA	Identifies failure modes in the software that are very difficult to identify with traditional nominal case testing. Will often identify hazards due to software that were previously unknown.
Inclusion of SW in FRACAS	Ensures that the contractor has a closed loop system for software failures and corrective actions
Software reliability risk assessment	Ensures that any risk doesn't derail the software reliability
Testing for software reliability	Provides confidence to the DoD that the code has been executed in addition to the requirements and fault injection.


System Reliability Model
Inclusion of SW in SRM
Reliable Software Allocations
If the software code and test is complete, the evaluations are used instead of predictions and it's too late for the FMEA.
Software FMEA/FTA
Testing for reliable software
Inclusion of software in FRACAS
Closure of failure modes
Identification of key risks that could affect allocations
Reliable Software Predictions
Reliable Software Evaluations
Open EFFs, NEFFs and System Aborts
Reliable Software Risk Assessment
Updates for scope estimates and other changes in plan
Demonstrated reliability curves towards goal
Test completion
Material release decision

Guidance for selecting the reliable software tasks

- The software reliability tasks, as a whole, aren't applicable if
  - There aren't any reliability requirements for the program
  - The software reliability has been demonstrated to meet the reliability requirements and there are no more planned upgrades and no more ECPs
- Otherwise tailor each task depending on the program size, type and software maturity
  - All software reliability tasks apply to Agile/Continuous Integration/Continuous Development frameworks and DevSecOps.
    - However, the RAM Government personnel may need to tailor the CDRLS/1423 for incremental development
  - The tasks apply to all software LRUs that are deployed into operational use but does not apply to development and test tools
    - IEEE 1633 2016 clause 5.1.1.1 describes what software is and is not applicable for software reliability

Reliable Software Tasks	MCA programs	Any program with a short contract time (MTA), any small but important program
Reliable Software Program Plan	√	√
Inclusion of Software in System Reliability Model	Model type can be tailored to complexity of SW/HW[1]	Contractor can choose simple model[1]
Reliable Software Allocations	Model selected based on accuracy/ availability of data[1]	Contractor can choose simple model[1]
Reliable Software Predictions	Select models depending on risk[2]	Either remove task or use simplest models[2]
Reliable Software Growth Evaluation	Full or minimal metric set depending on risk[3]	Minimal metric set is an option[3]
Software FMEA	Tailored by risk. [4]	Tailored by risk. [4]
Inclusion of Software in FRACAS	√	√
Software reliability risk assessment	√	√
Software reliability testing	Tailored to apply to the most mission critical software LRUs	

Summary

√- Applicable anytime there is mission critical software intensive system
[1] - Applies if either the software reliability predictions or software reliability growth evaluation is relevant
[2] - Most useful early in the program. Not useful if the coding activities are complete.
[3] - Unless the reliability objective has been demonstrated this task is relevant.
[4] - Most useful before code is complete. Not useful if all testing is complete.

Cost Justification

Task	Relative cost
Software reliability program plan	$
Inclusion of software in the system reliability model	$ The software group has to identify the software CSCIs anyhow so putting them in the model is not costly
Software reliability Allocation	$$ - Predictive models can be used for allocation. Automated tools are available
Software reliability Prediction	
Software reliability Evaluation	$$ - Low cost automated tools are available
Software FMEA	Our recommended tailoring is $$.
Inclusion of SW in FRACAS	$ Sending the software failure reports is not costly
Software reliability risk assessment	$ Consists of a simple checklist
Testing for software reliability	$$ to $$$ Our tailoring minimizes cost and maximizes effectiveness

Applicability for Agile/ DevSecOps

- All tasks apply for software programs developed with Agile infrastructure or DevSecOps
- The timing of the deliverables may be affected by the development framework and is discussed in the guidance for the CDRL/1423.
- Note that IEEE 1633 2016 has guidance in clause 4.4, Tables 16 and 23, clause 5.3.2.4, for the software reliability tasks for agile, incremental and waterfall deliveries.

Reliable Software SOW Language is integrated into the Reliability SOW language

At this time there no “software reliability” DIDs

Existing section of SOW geared towards hardware	Merge in these sections
Reliability program plan	Reliable Software Program Plan Software reliability risk assessment
System Reliability Model	Inclusion of Software Components in System Reliability Model
Reliability Allocations	Software reliability Allocations
Reliability Models, Predictions	Software reliability Prediction
Reliability Growth	Software reliability Evaluation Testing for Software Reliability
FMEA	Software Failure Modes Effects Analysis
FRACAS	Software FRACAS

The software reliability SOW language contained in this document is to be integrated into the appropriate reliability SOW sections.

List of DiDs and referenced documents

	Data Item	Title	Date
1.	DI-SESS-81613A (Sequence A001)	Reliability and Maintainability Program Plan (Reliable Software Program Plan)	15 Jul 14
2.	DI-SESS-81496B (Sequence A002)	Reliability and Maintainability (R&M) Block and Mathematical Models Report	8 Oct 19
3.	DI-SESS-81968 (Sequence A003)	Reliability and Maintainability Allocation Report	10 Jul 14
4.	DI-SESS-81497B (Sequence A004)	Reliability and Maintainability Predictions Report	8 Oct 19
5.	DI-SESS-81628B (Sequence A005)	Reliability Test Report (SW Reliability Growth)	18 Feb 20
6.	DI-SESS-81495A (Sequence A006)	Failure Modes, Effects, and Criticality Analysis Report	16 May 19
7.	DI-SESS-80255B (Sequence A007)	Failure Summary and Analysis Report	15 Oct 19
8.	DI-MGMT-81809 (Sequence A008)	Risk Management Status Report (Reliable Software Risk Assessment)	26 Apr 10
9.	IEEE 1633	IEEE Recommended Practice on Software Reliability	22 Sep 16
10.	MIL-STD-882E	Department of Defense Standard Practice System Safety	11 May 12
11.	SAE ARP-5580	Recommended Failure Modes and Effects Analysis (FMEA) Practices for Non-Automobile Applications	7 Aug 20
12.	INCOSE-TP-2010-006-01	INCOSE Guide for Writing Software Requirements	APR 12
13.	FSC-RELI	System and Software Reliability Assurance Notebook	1997

Summary of the Reliable Software Tasks

SSRM Defined

- A graphical depiction of the system with an underlying analysis, such as the Reliability Block Diagram, Markov Model, Mission Model, and/or Fault Tree Analysis.
- The SSRM is an SRM that includes all software components in the system model.
- The analysis should identify critical weaknesses in the system design which impact software reliability.
- These resources discuss system reliability models
  - IEEE 1633 2016 clause 5.3.4
  - “System and Software Reliability Assurance Notebook FSC-RELI” chapters 4 and 5 provide guidance.
  - See module 3 for examples

Reliable Software Allocation Defined

- This analysis ensures that the portion of the system reliability requirement is allocated appropriately to the software components.
- Software allocations are directly related to size estimates for the software. All size estimates are inherently derived from the amount of effort needed to develop the software.
- More complex software requires more effort. It doesn't matter what unit of measure is chosen for the size estimate. What matters is that size estimates can and will change until coding is complete.
- The software allocations can and will be revised as the size estimates are revised.
- Allocations are an ongoing process which goes hand in hand with the SSRM activity.
- Allocation can be made based on several different techniques with the relative accuracy identified on the next slide.
- IEEE 1633 Recommended Practices for Software Reliability, 2016 clauses 5.3.5 and 5.3.8 discusses several methods for allocation.
- See Module 3 for more information

Allocation methods as per IEEE 1633

Allocation Method	Preference
Historical data which indicates X% of the fielded failures are due to software.	Usually most accurate if the data is recent and the historical data is from a similar system with similar mission. While the accuracy of historical data is typically the best, it's also difficult to collect for DoD systems.
Recent testing data which indicates X% of testing failures are due to the software.	Relatively accurate if the software is being tested in an operational environment (with the target hardware).
Bottom-up allocation – All system configuration items undergo reliability assessment.	The hardware and software configuration items are applied to the SRM. The allocation for software is simply the predicted failure rate over total of all predicted failure rates. Even if the assessment does not meet the system requirement, the allocation is still the relative contribution of the prediction to the system prediction. The accuracy depends on the models used for the bottom-up predictions. More inputs to the model usually means more accuracy if the model is used correctly and inputs are correct.
Allocation by duty cycle. The % allocated to SW depends on the duty cycle of each of the components in the system.	This model is useful when there is varying duty cycle of the system components. Accuracy depends on the accuracy of the prediction model discussed in the software reliability prediction task. If historical data is used, this method is typically accurate.
Allocation by Research and Development cost. The % of R&D engineering $ spent on SW versus % R&D engineering $ spent on HW	Cost is a good indicator of software reliability but only if the cost is accurately predicted. If the cost of developing the software components is similar to the cost of the hardware R&D then the software contribution to failure rate is likely to be similar to the hardware.
Allocation by number of Configuration Items. Count the hardware LRUs and the software LRUs. Allocation is based on relative number of LRUs.	Not as accurate as other methods. There is much variation on how much code comprises an LRU. If there are many small LRUs, this method can over-allocate the software or hardware. If there is one large software LRU, this method can under-allocate the software portion.

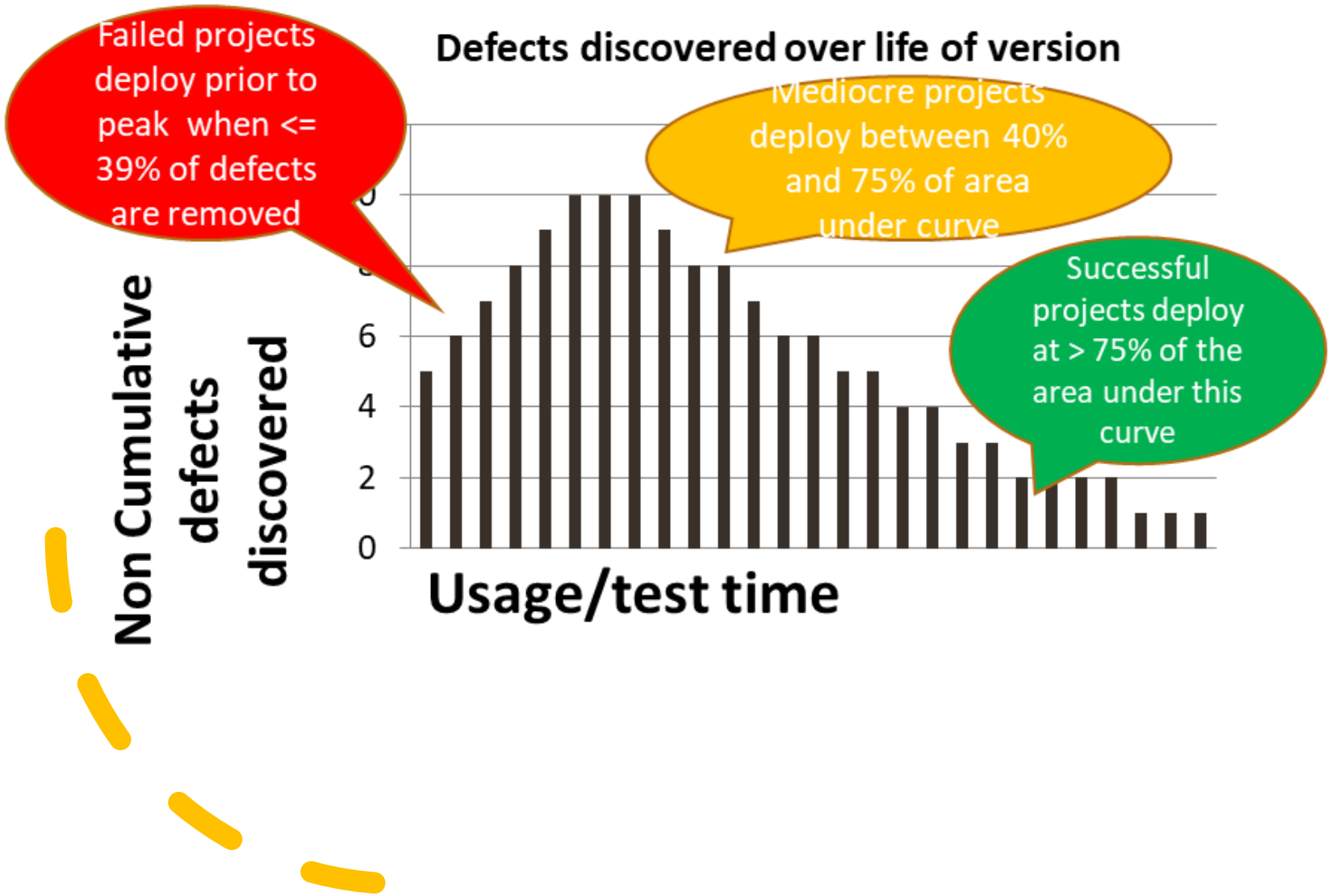
Overview of software reliability prediction methods

Method	Pros	Cons
Historical data from similar systems	Usually the most accurate when calibrated for any differences in mission or development practices	Many organizations either don't have any or don't have processes to collect it
Detailed assessment surveys	Next to historical data, these are most accurate as they are based on historical data. Accuracy depends on 1) number of questions 2) ability for organization to answer all questions accurately and 3) the age of the model. (Models > 20 years old are generally not accurate).	Requires contractor's software and reliability people to be coordinated. Time to complete assessment depends on how many questions there are and whether the RAM engineer can get the answers from software engineering.
Rayleigh model	When based on historical data such as QSM's SLIM, these models are relatively accurate.	Requires contractor's software and reliability people coordinate~~d~~ activities.
Simple look up tables based on application type or CMMi	Quick and easy	Are the least accurate of the other methods shown above but significantly more accurate than subject matter guessing

Reliable Software Evaluation Defined

- Reliability growth is the positive improvement in reliability metric over a period of time due to the implementation of corrective actions. For software, reliability growth is a function of:
- Amount of testing hours and test hours during which there are no new features added to the software system
- The stability of the reused and off the shelf software components
- The number of installed sites during reliability growth - more installed sites and end users means faster growth while fewer installed sites usually means less rapid growth
- The Reliable Software Evaluation should measure the:
  - Defect discovery of failures from software (increasing, peaking, or decreasing or some combination)
  - Actual software reliability tracked against Software reliability goals
  - Capability drops and expected effect on reliability
  - Degradation due to test environment, scalability, etc.
  - Confidence bounds on predicted and extrapolated reliability metrics

Software reliability Evaluation



- This illustrates typical defect discovery profile over the life of a software version (Only unique defect discoveries graphed).
- If the contractor deploys the software before the peak, the software is immature and not suitable for the customer.
- If contractor deploys the software between the peak and when the software stabile (defects flatten out), the software may be usable but not meet the reliability goals.
- If the software deploys once the defect discovery rate flattens either the reliability objectives of the program have been met or are on the path to meeting those objectives stabilize.

Software reliability Evaluation

- The most important metric is the defect discovery trend.
- If the trend is not decreasing, then most of the other metrics are largely irrelevant.
- The second most important metric is the fix rate which ensures that the contractor is fixing the defects fast enough to address the failures that effect reliability or availability.
- Also important are the defects not piling up from release to release or Sprint to Sprint.
- Figure on previous page is an example of defect pileup.
  - The discovered defects are plotted in increments of 10 usage hours.
  - When Sprint 2 was merged in at 190 hours, the most recent defect discovery rate was at 1 defect per 10 hours. However, at 350 hours, the most recent rate is at 5 per 10 hours (4 from Sprint 2 and 1 from Sprint 1).
  - Sprint 3 is about to be merged in despite the increase in the rate and the fact that Sprint 2 received 30 hours less of testing than Sprint 1.
  - If Sprints 3 and beyond continue in this pattern, eventually the software will be released with an increasing defect rate. The FDSC should provide specific examples of software failures that effect reliability and availability and should not be limited to only those failures that result in a shutdown.

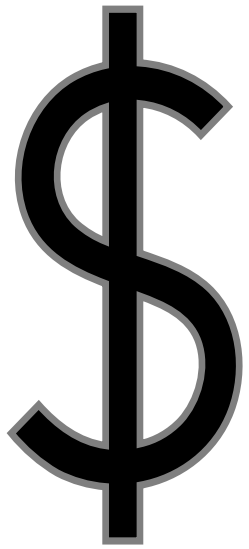
Software Failure Modes Effects Analysis

Below are just a few examples of the failure modes that lead to serious software failures:

- Faulty error handling – Quantas flight 72 un-commanded downward pitch (incorrect fault recovery), Mars Polar Lander (software failed to detect spurious data) , Denver Airport (software assumed the luggage would not get jammed) , NASA Spirit Rover (too many files on drive not detected)
- Faulty data definition – ESA Ariane 5 explosion (16/64-bit mismatch) , Mars Climate Orbiter (Metric/English mismatch) , TITANIV (wrong constant defined)
- Faulty logic/sequence - Solar Heliospheric Observatory spacecraft mishap , AT&T Mid Atlantic outage in 1991 , Operator's choice of weapon release overridden by software control
- Faulty state management - Incorrect missile firing from invalid setup sequence
- Faulty algorithm – Flight controls fail at supersonic transition , Mariner 1 mishap
- Faulty timing –2003 Northeast blackout , Therac 25 race condition , Missile launch timing error , Apollo 11 lunar landing
- Faulty endurance – Patriot system failure
- Peak load conditions - IOWA caucus failure
- Faulty usability
- Software makes it too easy for humans to make irreversible mistakes –PANAMA city over-radiation
- Insufficient positive feedback of safety and mission critical events – 2007 GE over-radiation

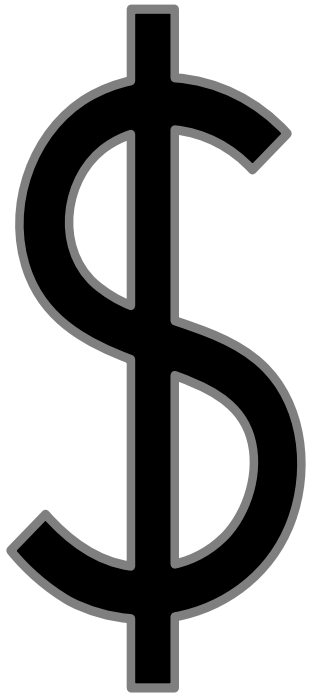
Software FMEA approaches

Method	Description
Functional	Focus on the design and specifications and in particular what they are NOT stating. Applied at 3 levels: TL - Top level – things that has effected other systems (peak loading, etc. FL - Feature level – things that go wrong at the use case, capability level SL- Specification level – things that go wrong with a single specification statement **Common Defect Enumeration is posted on DoD R&M CoP Website. See my other presentation on this.**
Interface	Focus on the common interface faults such as metric/English conflicts, conflicts with data type/size/format/scale/resolution **Common Defect Enumeration is posted on DoD R&M CoP Website. See my other presentation on this.**
Detailed	Focus on the code. This is most labor intensive and does not identify faults due to "missing code". THIS approach is expense and NOT recommended.



When it's most cost effective

- The software FMEA ***is not*** generally cost effective if:
  - The software is completely finished developed and tested and there haven't been any serious unexpected software failures from this product recently
  - OR the software is in a very mature state and there's no major upgrades or ECPs planned
- The SFMEA identifies failure modes that effect design and even specifications
- If the analysis is completed after code is done it will be expensive for software engineering to make fixes
- It is essential that the SFMEA be conducted in line with software development
- SFMEA has ZERO value if completed after software testing is done



Alternative for reducing cost

- A top level FMEA is the cheapest and covers the most functionality if conducted properly
- See the Top-Level Common Defect Enumeration tables.
  - Endurance – system degrades during life of mission – CDE TL-PR-1 and TL-PR-2
  - Peak loading – system cannot handle multiple threats at same time or different threats CDE TL-PR-5 through TL-PR-8
  - Processing – Videos, data logs, files build up over time and eventually cause mission computers to shut down – CDE TL-3
  - Inability to detect or handle hardware faults, power faults, communication faults, computations faults or user faults- CDE TL-EH-1 through CDE TL-EH-30
  - Changes in mission such as duration – CDE TL-FC-4
  - Prohibited state transition allowed by code – CDE TL-SM-1 and CDE TL-SM-2
  - Software is unable to recover after an abort or unexpected shut down or loss of power – TL-SM-5

Inclusion of software failures in FRACAS

- Contractors quite often store software problem reports separately from the hardware reports and rarely call the system a “FRACAS”
  - They often use “JIRA” or “Clearquest” to store their reports
- The contractor is required to have a closed loop process for software failure reports. This task is simply making those reports available to the Government and tagging the failures that effect reliability.

Software reliability risk assessment

- The software FMEA and software fault tree identify specific functional failure modes that directly lead to a specific system failure. Software risks are organizational decisions that can lead to many software failures.
- Historically these risks were known from the start of the program but no one paid attention to them or understood their effect on the program.
- These are some of the risks that can single handedly derail a program:
  - The contractor plans to reuse code that's not really reusable
  - The contractor is not planning to reuse code when they should
  - Grossly underestimated the work required to modify the code for a new mission duration or mission type or new weapon hardware
  - The contractor has a team of software engineers that does not understand the mission, weapon, customer or industry
  - The contractor has sudden high turnover of software engineers working the program
  - The contractor has software people who aren't near the target hardware or hardware engineers
  - The contractor is attempting to do handle too many learning curves in a single customer release
  - Learning curves include but aren't limited to:
    - Technology that is new to the software team (i.e. the first time they have developed a cloud application for example)
    - Hardware interfaces that are undefined and evolving
    - A sudden change in staff or company leadership.

Reliable Tests

Test type	Software functional level example	System functional level example
Go-no go	Pressing a button is “Go”. Not pressing a button is “No Go”	The car does not brake when not commanded, does not accelerate when not commanded, the convertible top is not put down when not commanded
Timing	The BIT test starts no later than 100ms after startup and finishes no later than 2 seconds	The car can brake or change lanes within the time required
Boundary	The algorithm accepts values between 50 and 90. The boundary tests are 49, 50, 90, and 91.	The vehicle is accelerated from stop, and from maximum speed limit
Trajectory	Using the same algorithm- it can range from 50 to 90. Trajectory tests might include starting at 50 and transitioning to 90 and vice versa. Starting at 75 and transitioning to 50. Starting at 75 and transitioning to 90. Many others.	The vehicle is accelerated and deaccelerated from each of these velocities to every other velocity – 1) Very low speeds (school bus scenario), 2) low speed (side streets), 3) medium speed (major roads), and 4) high speed (highways)
Power test	Cutting the power while running any software intensive function.	Run out of gas and verify that the vehicle does not accelerate or shift into reverse immediately after refueling. (i.e., should not remember what it was doing before running out of gas).
State tests	Testing the lower-level state transitions for all software functions. Showing ~~that~~ all low-level prohibited state transitions are not allowed by the software.	The vehicle does not transition to park mode while driving or transition to drive mode while parking; or the convertible top is not allowed to go up or down while moving (whether commanded or not).

Reliable Tests

Test type	Software functional level example	System functional level example
Data value tests	Using the algorithm example for boundary testing – testing large jumps in values, small jumps in values, fractional changes in values.	Small velocity changes (going a few MPH faster or slower), velocity changes in whole numbers, velocity changes in fractional numbers, big velocity changes (i.e., from 40 to 70mph or 70 to 40)
Fault injection tests	Injecting bad data such as NaN (not a number)	Fault injection with faulty vehicle hardware or consumables (brakes, oil, fluids, tires, etc.)
Zero value test	Setting values in computations to zero or near zero.	Verify transitioning to stop (zero velocity) from all four (4) velocity ranges and transitioning from stop to all four (4) velocity ranges
Peak loading tests	Testing each function with the maximum volume of concurrent inputs	Rapid succession of stop and go (traffic lights or school bus)
Endurance test	Test each software function for the maximum mission time for that function	Get on a major highway and drive until nearly out of fuel
TLYO	Drive like real people drive (teenagers, adults, working people, retired people, professional drivers, etc.)	

Guidance on selecting tests

Not all tests apply to all systems.

Test type	Applicability	Justification
TLYO	Applicable to all systems.	TLYO is the closest test to end user operation. The trajectory tests are an important ingredient of TLYO.
Trajectory	Applicable to all systems.	
Go-no go	Applicable to all systems.	Can be easily combined with requirements testing.
Fault injection tests	Applicable to all systems.	Can be covered at same time as a hardware fault injection test if the required behavior of the software under various faulted conditions is documented. Power testing is a subset of fault injection testing.
Power test	Applicable to all systems.	
State tests	Applicable to all systems.	This test ensures no inadvertent irreversible weapon events. It is not expensive to test.
Timing	Applicable to all systems.	Timing is critical for weapons. If the software specifications cover timing budgets this will be implicitly tested in the requirements testing. However, tests for scheduling analysis are typically not covered in the contractor's requirements testing.
Endurance test	Applicable to all systems. Most relevant for systems that are on for an extended duration (more than a few hours).	This consists of one test for the duration of the mission time without reboot. If the mission time is particularly long benchmarking of timing and accuracy can establish whether the software degrades for the entire mission.
Peak loading tests	Applicable to systems that have multiple users, multiple simultaneous threats, multiple workstations, etc.	This test is not expensive to run. Most of the work is in the setup of the workstations, users, etc.
Boundary	Applicable to all systems.	Zero values and boundary tests are conducted at the same time. These tests cost effectively verify ranges of value so as to test the values that are most likely to be problematic. These tests are not expensive, particularly when conducted at an LRU level.
Zero value test	Applicable to all systems.	
Data value tests	Applicable to all systems.	Data value tests are combined with other tests such as boundary, zero value and trajectory tests to minimize the total test cases. The goal is simply to test with varying data types and sizes.

RAM/Software Engineering Coordination

- Integrating the reliability activities with the software design and the Systems Engineering effort ensures that all parties are aware of software design changes and how software design changes impact reliability of the software and system level reliability.
- The software architecture can have a direct impact on the reliability block diagram.
  - For example, if all of the software code is developed so that it is in one software configuration item, that item can be a single point failure in the RBD.
  - Additionally, the reliability engineers will need to have size estimates and other data in order to perform software reliability predictions and assessments.
  - The Contractor should have methods in place to ensure that there is a two way communication channel between the reliability and software design teams and that these teams are working together towards a common goal.

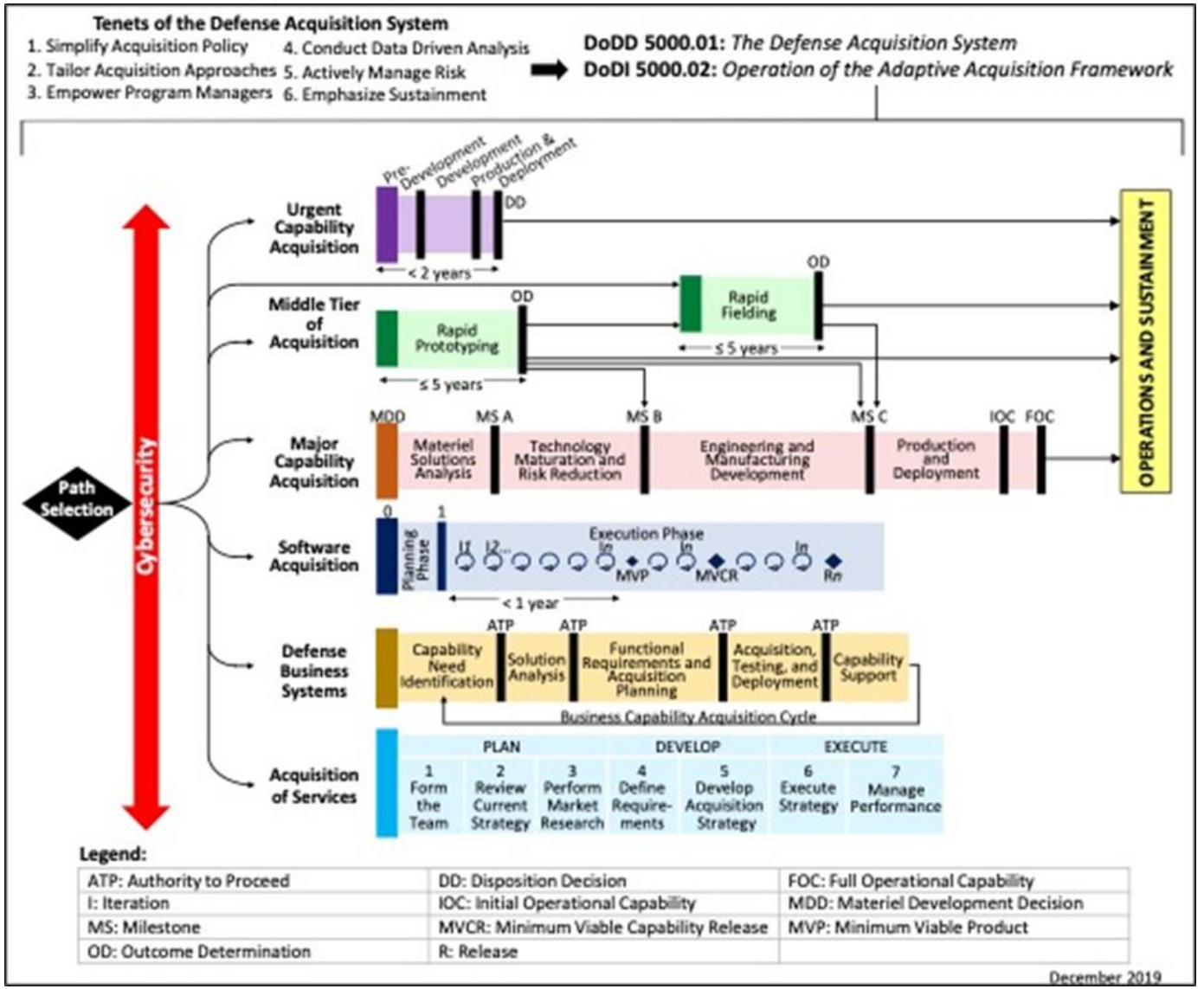
Lessons Learned

- Very few organizations have true “software reliability” personnel who understand both software and reliability engineering
- Systems design ensures that both software AND hardware design is optimized for *system* reliability. Many organizations design vacuums.
- Software engineering and reliability engineering personnel rarely interact during proposal phase, development or engineering
- Reliability engineering often proposes the software reliability tasks without coordination from software engineering
- Software engineering is often blindsided by commitments and assessments conducted by reliability engineering

Annex | DoD Pathways

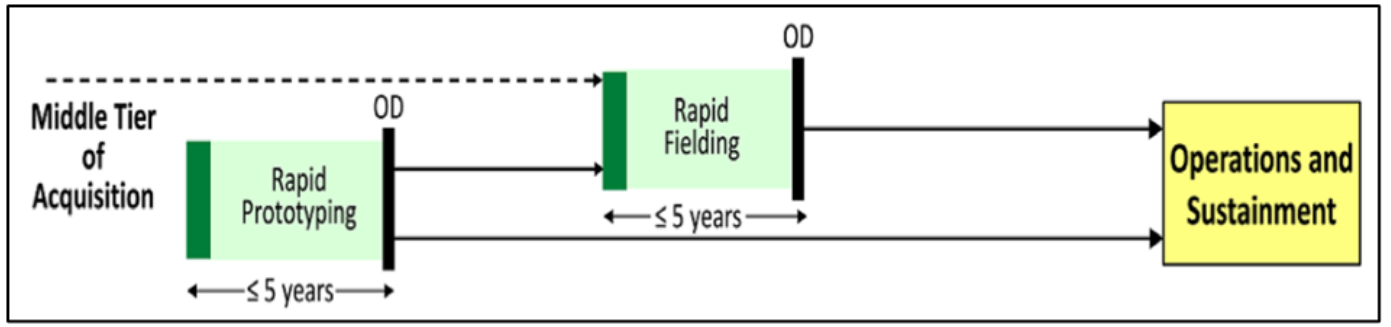
This figure the six DoD acquisition pathways. This SOW guidance is only for MCA, MTA, and Software Acquisition.
DoDI 5000.02, "Operation of the Adaptive Acquisition Framework," January 23, 2020)

DoD Pathways



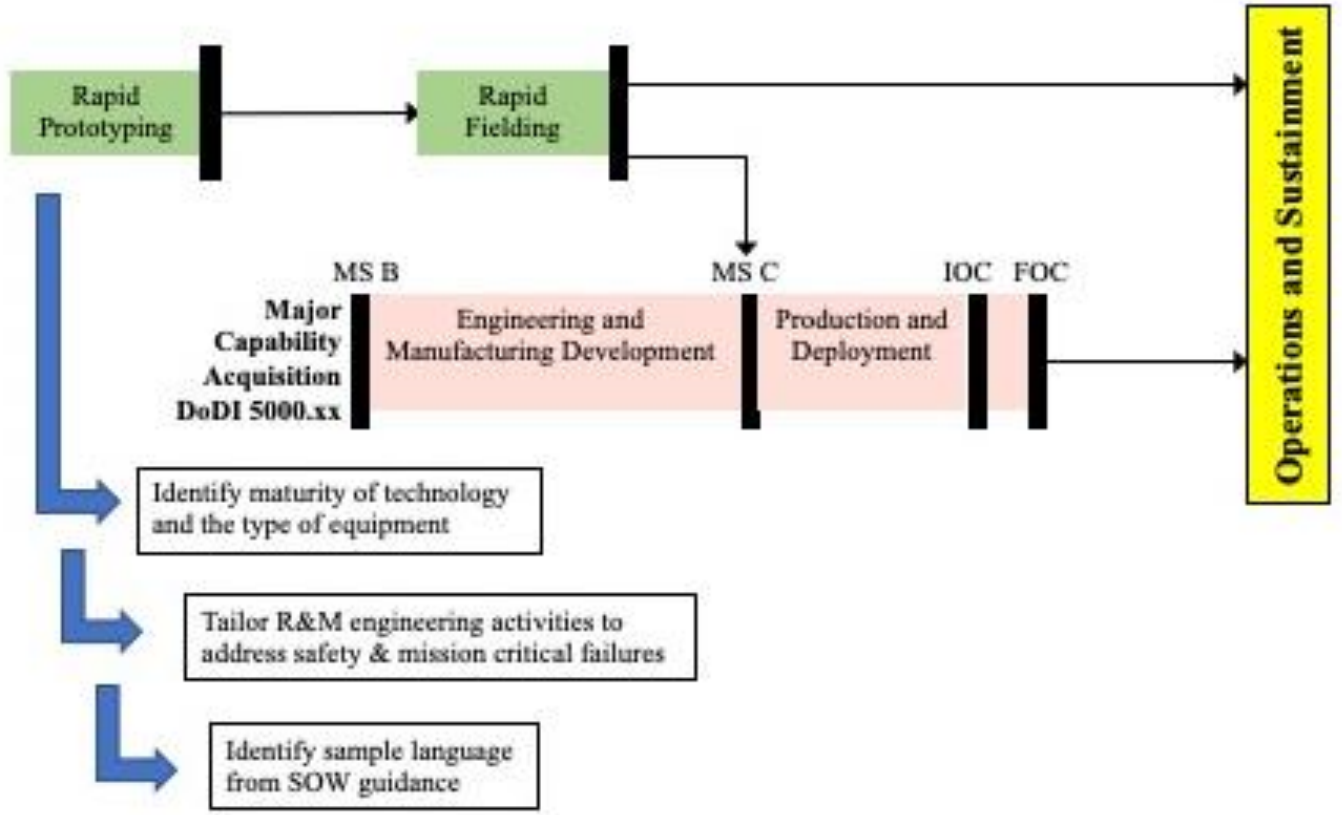
RP to RF to field

MTA Path



MTA with RPRP Path with a transition to RF with a transition to MCA

MTA Path ways



MTA with RP transition to MCA

MTA Path ways