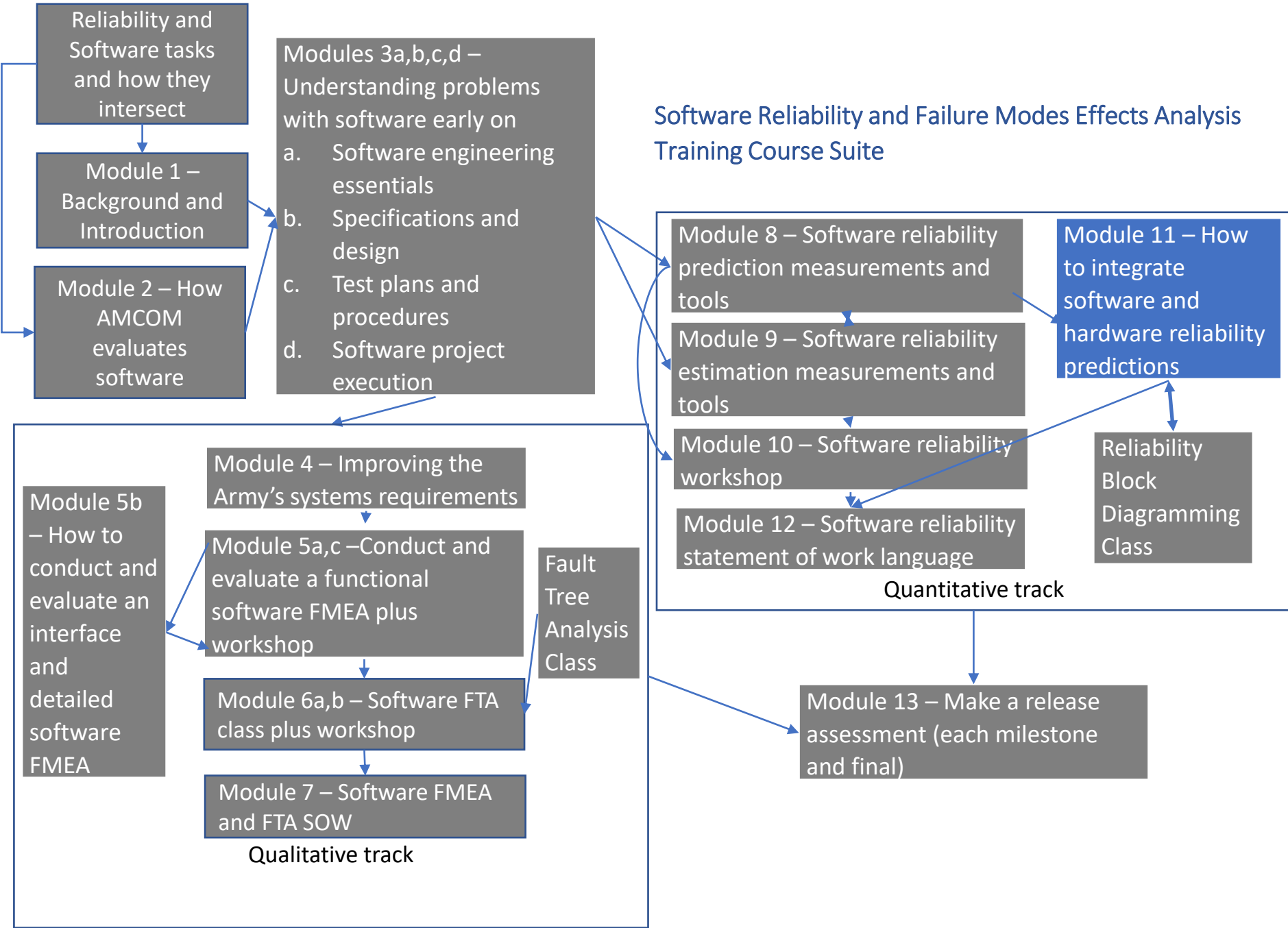


# Module 11 – Integrating SW and HW reliability

Ann Marie Neufelder

# Software Reliability and Failure Modes Effects Analysis Training Course Suite



# Agenda

- The value of merging software and hardware reliability predictions
- Establishing a reasonable top level reliability goal
- How to reconcile reasonable versus targeted
- How to merge the predictions
  - Reliability Block Diagram
  - Mission model
  - Use case model
  - Operational profile model
  - Fault tree
- How to establish allocations between HW and SW
- Tools
- Annex – formulas for computing reliability when there is redundancy



# The value of merging software and hardware reliability predictions

If you don't merge the software predictions with the hardware predictions you won't know if system objective has been met

## Reasons to merge SW and HW predictions

- You won't know if the system objective has been met or will be met
- If the system is entirely software then this step isn't required
  - However, if the duty cycle of each of the software components is different - the models may be necessary to accurately predict the combined reliability of different software components.
- System models such as RBDs, fault trees and Markov models aren't complete if they don't have software measures integrated

# Definitions

---

System objective – The targeted MTBF, reliability, availability, etc. for a system composed of hardware and software

---

Software objective – The targeted MTBF, reliability, availability, etc. for only the software portion of the system

---


System prediction – The forecast of the MTBF, reliability, availability, etc for a system composed of hardware and software. The prediction may or may not meet the objective.

---

Software prediction – The forecast of the MTBF, reliability, availability, etc. for the software components of the system

---

Software allocations – The portion of the system objective allocated to each of the software components as well as the software overall



## Estimate the portion of all system failures that will be due to software

### Why?

- Ensures that the system objective considers that percentage
- Establishes an initial quantitative target for both hardware and software reliability
- Ensures that the system objective is reasonable/feasible for a system that is software intensive.

Estimate the  
portion of  
system failures  
due to  
software

As a customer, you'd like to have a ballpark idea of the contribution of the software so that you can establish a system goal that is reasonable

Early in the program these are ways that a customer can bound the percentage of failures that will be due to software

- \$ budgeted for R&D as per a past similar program or contractor's proposal
  - Number of requirements to be fulfilled by software versus hardware
  - Achievable failure rates
  - Past failure history on other similar systems or a previous version of the system under analysis
- They are listed from lowest to highest accuracy



# R&D \$

## Establishing portion of system failures due to software

Example:

R&D for software is 10M

R&D for hardware is 15M

Portion of system failures predicted to be due to software =  $10/25 = 40\%$

Rationale: Software failures are directly related to software size. Software size is directly related to R&D \$

- A. Identify the R&D budget for software
- B. Identify the R&D budget for hardware
- C. Percentage of failures due to software =  $A / (A+B)$

DESCRIPTION	BENEFITS/ DISADVANTAGES
<b>% of software failures due to software proportional to % of R&amp;D budget for software to total R&amp;D budget</b>	Easy to compute. Minimizes the possibility that software will be allocated zero percent of the system objective.

# Number of requirements to be fulfilled by the software

# Establishing portion of system failures due to software

## Example:

There are 2000 customer requirements to be fulfilled by the software

There are 3000 customer requirements to be fulfilled by the hardware

Portion of system failures predicted to be due to software =  $2000/5000 = 40\%$

Disadvantage – not all requirements are equal in complexity

- A. Identify the total customer requirements tagged to the software
- B. Identify the total customer requirements tagged to the hardware
- C. Percentage of failures due to software =  $A / (A+B)$

DESCRIPTION	BENEFITS/ DISADVANTAGES
<b>% of software failures due to software proportional to % of total requirements</b>	Easy to compute. Assumes that each requirement is relatively equal in complexity.

# Achievable failure rates

## Establishing portion of system failures due to software

Example:

Software failure rate predicted to be .005 failures/ hour

Hardware failure rate predicted to be .003 failures/hour

Portion of system failures predicted to be due to software =  $.005/.008 = 62.5\%$

Rationale: Ball park predictions are more accurate than assuming 0 defects due to software

- A. Use the fast track model discussed in Module 8 establish a failure rate for the software
- B. Using any industry method, establish a failure rate for the hardware
- C. Percentage of failures due to software =  $A/(A+B)$

DESCRIPTION	BENEFITS/ DISADVANTAGES
<b>% of software failures due to software proportional to it's predicted failure rate versus the failure rate of the software</b>	Not as accurate as past history but more accurate than other models if used correctly.

# Recent past history

## Establishing portion of system failures due to software

### Example:

On a similar past system there were 100 failures due to software over the first 2 years of operation.

There were 150 failures due to hardware over the first 2 years of operation.

Historically the portion was 40% but the system was 7 years old.

At 10% increase per year, the new system will have about 2 times as much code or 200 failures.

It's expected that the number of HW components are unchanged.

Portion of system failures predicted to be due to software  
 $= 200/350 = 57\%$

- A. Add up number of system failures caused by software in a recent predecessor or similar program
- B. Increase A by the 10-12% for each year since the software was developed
- C. Add up number of system failures caused by hardware in a recent predecessor or similar program
- D. Adjust C by any reductions or increases in hardware components
- E. Percentage of failures due to software =  $B / (B+D)$

DESCRIPTION	BENEFITS/ DISADVANTAGES
<b>% of software failures due to software proportional to % of past % of system failures adjusted by 10-12 percent per year</b>	Most accurate method and easy to compute but requires failure data from a historical program.

# Review

You learned that the contribution of software failures to the system reliability can be ball park estimated with

- Historical data from a predecessor
- Historical data from an industry software reliability prediction model
- Total R&D \$ estimated
- Number of software versus hardware requirements

Key takeaway –

- Data is the key to any prediction. Whether it comes from a predecessor system or an industry prediction model.
- Data is how hardware reliability predictions are established. Software is no different.



# Identify a realistic system reliability objective

## Why?

- Wishing don't make it so
- Ensures that the objective reflects all components and not just the hardware
- Ensures that the system objective is reasonable for a system that is software intensive.

## Inputs

- Past history for hardware and software failure contributions

## Outputs

- A realistic system objective

## Determine the relevant reliability metric

As discussed in the module 8 the first step is determine which of the reliability figures of merit are relevant

<b>Reliability metric</b>	<b>When it's relevant</b>
Failure rate/MTBF	Most systems
Probability of failure over a mission time - Reliability	Systems with a defined mission time such as missiles, landing gear, aircraft, ground based vehicles, ground based mobile missile launchers
Availability	Systems that operate continually such as security systems, surveillance, radar, satellites

## Process for identifying a system objective

### Reliability objectives and allocations typically evolve

- Customer or stakeholder identifies the targeted system reliability based on its mission objectives
- Engineering organization predicts the reliability for each component in the system
- Merges all predictions into one system prediction
- Allocates the system objective to each configuration item
- If system objective isn't feasible alternatives are investigated
- For software
  - Software COTS considered if possible
  - Reduced or staged SW features
- For hardware
  - Redundancy
  - Different parts
- Initial objective revised as needed



# Deriving an MTBF objective

- When you have historical data from a predecessor
- When there is no predecessor
- When you have \$ or system requirement counts

# Establish a system MTBF objective when there is a predecessor

1. From actual predecessor system MTBF. Determine % of software and hardware failures in operation . Determine actual software MTTF and hardware MTBF from that percentage.

2. Multiply number of actual operational software failures by at least 110% for each year since that predecessor software has been deployed. [B11]

3. Determine how much hardware has changed. Adjust historical hardware failure count by this percentage.

4. Determine the system MTBF objective by combining the new software objective from step 2 with the new hardware objective from step 3.

## Example

Problem: A system is being proposed that will be a successor to an existing system.

The existing system was first deployed 7 years ago. The average MTBF was 300 hours for the entire system.

Software failures were 40% of the total failures for the existing system.

The hardware elements are reduced by 10% since several mechanical functions are now being performed by the software

The goal is to derive a system MTBF objective for the new system.

# Example Solution

Compute the software and hardware MTTF for the existing system by applying the 40% and 60% to the known 300 hour MTBF. This assumes that the software and hardware are independent.

- Software MTTF allocation - 750 hours (300/.4)
- Hardware MTBF allocation -500 hours. (300/.6)

Since the software was developed 7 years ago, first compute its relative size when compared to the existing system.

- If the software size grows 10% a year it will be about 2 times larger after 7 years.
- Size is inversely proportional to MTTF. Hence software MTTF will likely be 375 hours *if the development practices and all other parameters remain the same other than the size.*

Since there is 10% less hardware, its allocation is 550 hours compared to the 500 hours in the predecessor system.

New system prediction =  $((1/550) + (1/375))^{-1} = 223$  hours. Or a failure rate of .004485 failures per hour.

# Establish a system objective when there is not a predecessor

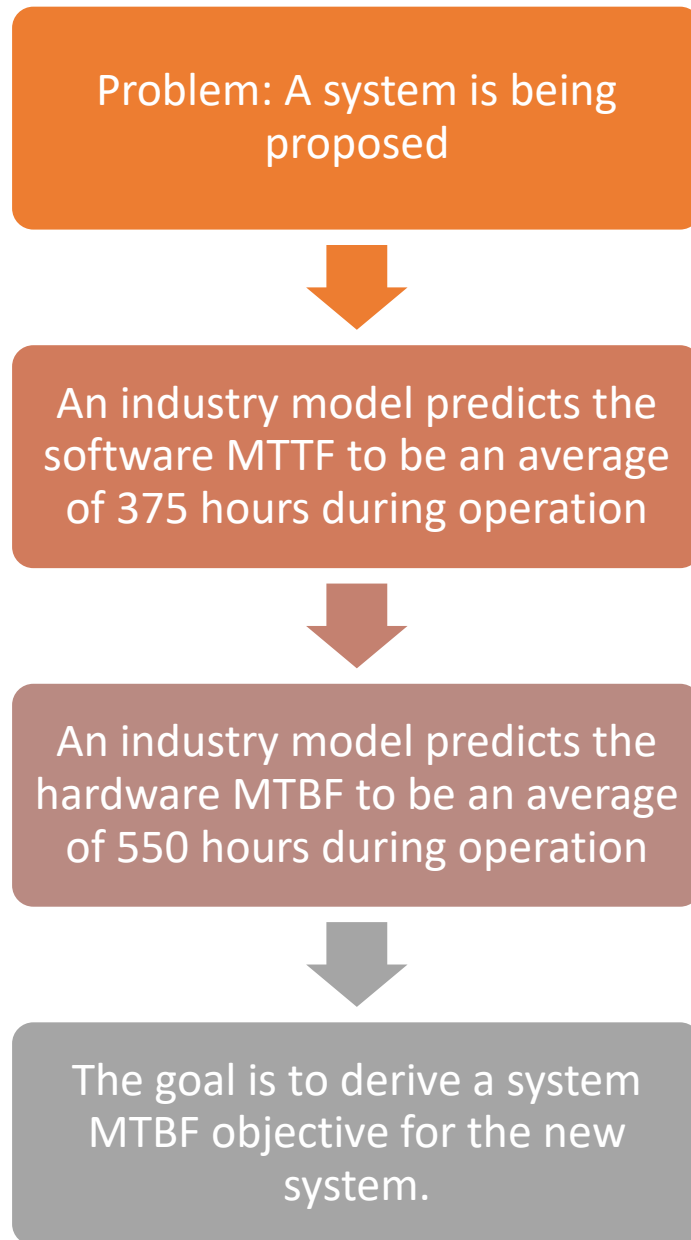
1. Use an industry model to predict the software reliability

2. Use an industry model to predict the hardware reliability

3. Combine the predictions either by assuming that the software and hardware is independent or by using one of the methods in this training class

4. The objective system MTBF is determine by the achievable system failure rate from step 3.

# Example



# Example Solution with no redundancy and independence between SW and HW failures

1. Software MTTF objective is 375 hours
2. Hardware MTBF objective is 550 hours
3. If the software and hardware fail independently and there is no redundancy - then the failure rate objectives can be simply added. The system objective MTBF is therefore the inverse of the system failure rate objective.

System objective MTBF is therefore  
 $((1/550) + (1/375))^{-1} = 223$  hours.

# Other methods

- The methods discussed in the previous section that determine the percentage of failures that will be caused by software versus hardware can also be used to estimate a system MTBF objective
  - R&D \$
  - Requirements
- These might be useful if there's no time for predictions and no predecessor or historical data



## Example using R&D \$

1. The predicted R&D budget for the software is 70 Million

2. The predicted R&D budget for the hardware is 50 Million. The hardware MTBF is predicted to be 550 hours.

3. The hardware is predicted to be 42% of the system so that means the software will have an MTTF of about 398 hours.

4. That means the system objective MTTF is 231 hours.

# Deriving objectives from availability or reliability objectives



Previously, it was illustrated how to derive a software or hardware MTBF from an objective system MTBF



In some cases, the system objective may be in terms of availability or reliability



This section shows how to derive the system availability or system reliability objective from the software MTBF objective

# Establish a system availability objective from a MTTF objective

1. Start from the objective MTBF derivation as shown in previous steps.

2. Predict the Mean Time To Software Restore using the methods shown in IEEE 1633 (Module 8)

3. Predict the MTTR for the hardware

4. Compute a weighted average of the repair time using the portion of failures predicted from HW and SW

5. System Availability objective=  
System MTBF objective / (System MTBF objective+ (Weighted average of MTTR and MTSWR))

# Example

1. Predicted system MTBF objective is 223 hours.

2. Using the methods in the IEEE 1633 the predicted MTSWR = .75 hours

3. The MTTR is predicted to be .5 hours

4. The portion of the objective failures for software is established at 60% while the hardware contribution is 40% so that means that normalized repair time =  $(.6 * .75) + (.4 * .5) = .65$  hours

5. System Availability objective=  
 $223 \text{ hours} / (223 \text{ hours} + (.65 \text{ hours})) = .997094$

# Establish a system reliability objective from a MTTF objective

1. Start from the objective failure rate derivation as shown in previous steps.

2. Identify the mission time for the system

3. Compute the reliability objective =  
 $\exp(-\text{mission time} * \text{system failure rate objective})$

# Example

1. The predicted objective failure rate =  $1/223$  hours =  
.004485 failures / hour

2. Predicted mission time is 5 minutes or .08333 hours

3. Compute the reliability objective =  $\exp(-.08333 * .004485)$   
= .999626

# Review

The relevant reliability metric must be chosen first.

- Availability isn't the best metric for software that has discrete mission times
- Reliability isn't the best metric for software that is continually operating

The process for identifying a system objective is usually iterative

MTBF objectives can be determined from

- Past history from a predecessor
- Achievable failure rates if no predecessor
- The estimated portion of failures from SW/HW (R&D \$, requirements)

Reliability or Availability objectives can be determined from the MTBF objective



# How to reconcile “realistic” versus “desired”

What to do if you have a reasonable top level goal and you don't like it



The methods shown previously establish a “reasonable” objective that doesn’t meet your goals

- The typical reaction is to stick with an unrealistic objective and require the software group to “make their software better”
- This has worked in 0% of all cases attempted since the 1960s for differences of > 5% between target and objective.
- The more software you have the more it will fail. Pretending it’s not so doesn’t make it not so.
- These are some options that can work

# Options

- As discussed in Module 8, software reliability is a function of these things
  - Size
  - Development practices (defect density)
  - Test coverage
  - Reliability growth
- As the customer you have control on only 3 of the above factors
  - Size – you can reduce the scope
  - Test coverage – you can require and pay for more testing such as trajectory, fault injection, boundary, line and branch coverage, zero value, etc.
  - Reliability growth – you can wait longer for the goal to be reached during which time you can't have any new features introduced into the code
- Requiring the contractor to reduce the defect density generally won't work well because
  - It takes a lot of culture change to reduce the defect density
  - If the contractor knew how to do it, they would already be doing it
  - The other 3 factors are significantly more sensitive

# Ways to reduce size (scope)

If the software features are staged over several small releases the code has time to be debugged and tested in an operational environment

- You can't tell the contractor how to develop the code but you can tell them how often you want software releases. Statistically things go better when the major new feature releases are no greater than 9 months.

Any high risk software features should be reconsidered for a future program. Examples of high risk features:

- It's experimental or hasn't been done before on any other system
- It's a very large feature compared to everything else
- The contractor will have difficulty testing the feature in their environment compared to other features

Rerun the top level estimates until no more size adjustments are possible then proceed to the next option

# Example

---

The customer requirements are reviewed and 20% can be offloaded to another time

---

Since the customer doesn't have insight into the size estimates of the software it can only assume that the size has decreased by 20%

---

When size decreases by 20% so does the failure rate

---

This is a conservative estimate because decreasing the size by 20% may have an exponential effect if it means that the project can be on time.

# Ways to increase reliability growth

Rerun the models discussed in module 8 with more reliability growth until the objective is met

However, no new features can be introduced during that reliability growth period, otherwise the MTBF resets as a function of the size of the new feature

- You can't have your cake and eat it too
- You can have reliability growth but not as long as you are adding in new code that hasn't been used in a real world environment.

Continue to the next option if the extended reliability growth without features isn't feasible

# Example

$$\Delta t = ( N_{0e} / \lambda_{0e} * \ln(\lambda_p / \lambda_f)$$

$\Delta t$  = additional test hours

$N_{0e}$  = total predicted defects

$\lambda_{0e}$  = failure rate on first day of testing

$\lambda_p$  is the failure rate objective that doesn't meet the target failure rate objective

$\lambda_f$  is the target failure rate objective

- $\lambda_p$  is the failure rate objective that you don't like
- $\lambda_f$  is the failure rate objective that you want
- In our example  $\lambda_p$  is .0431 failures per hour
- We want  $\lambda_f$  to be .0001 failures per hour
- From Module 9 we learned that we can forecast additional test time with the below formula
- $\Delta t = ( N_{0e} / \lambda_{0e} * \ln(\lambda_p / \lambda_f)$
- From Module 8 we learned that a typical growth rate is 6
- From the Module 9 we also know that the growth rate =  $\ln(1/\text{slope})$  and  $\text{slope} = \lambda_{0e} / N_{0e}$  So:
  - growth rate =  $\ln(N_{0e} / \lambda_{0e})$
  - $\Delta t = \exp(\text{growth rate}) * \ln(\lambda_p / \lambda_f)$
- Therefore  $\Delta t = ( \exp(6) * \ln(.0431 / .0001) ) = 2427$  additional test hours. There are about 168 hours of work per month.
- So assuming the software is tested on only one system during normal work hours - the additional test effort is about 14.4 extra months during which time no new features can be added.

## Example

- Let's assume that 14.4 months without any new features isn't acceptable to the Customer
- Some options for shortening that are
  - Have 2 test harnesses available to the contractor and shorten the time to 7.2 months
  - Have 3 test harnesses available to the contractor and shorten the time to 4.8 months
- Risks associated with shortened growth
  - You still have to pay people to staff those tests
  - The additional \$ of having more test harnesses
  - The system has to be operated as it would be in operation
  - There must be variations in operating profiles to ensure that the code is exercised
  - The contractor has to have people on staff to fix the defects found
  - Many software defects are caused by the software not being fault tolerant to hardware failures.
    - Shortening the reliability growth means that the hardware is less likely to fail during the growth test.

# Ways to increase test coverage

Software organizations typically cover the software requirements and that's all

- Requirements testing typically barely cover 40% of the lines of code, conditions, decisions, or data. The customer finds the defects in the other 60%.

These tests aren't relatively expensive and add value – Go-no go, trajectory, zero value, boundary

- These tests typically require no special instrumentation. .

Fault injection testing is a MUST have for mission critical systems.

Line coverage testing requires automated tools and the development organization will charge a lot for it. Consider a minimum threshold such as 90%. The cost between 90% and 100% is typically more than the cost of the first 90%.

This option will require explicit SOW for which the development organization will be passing the cost on



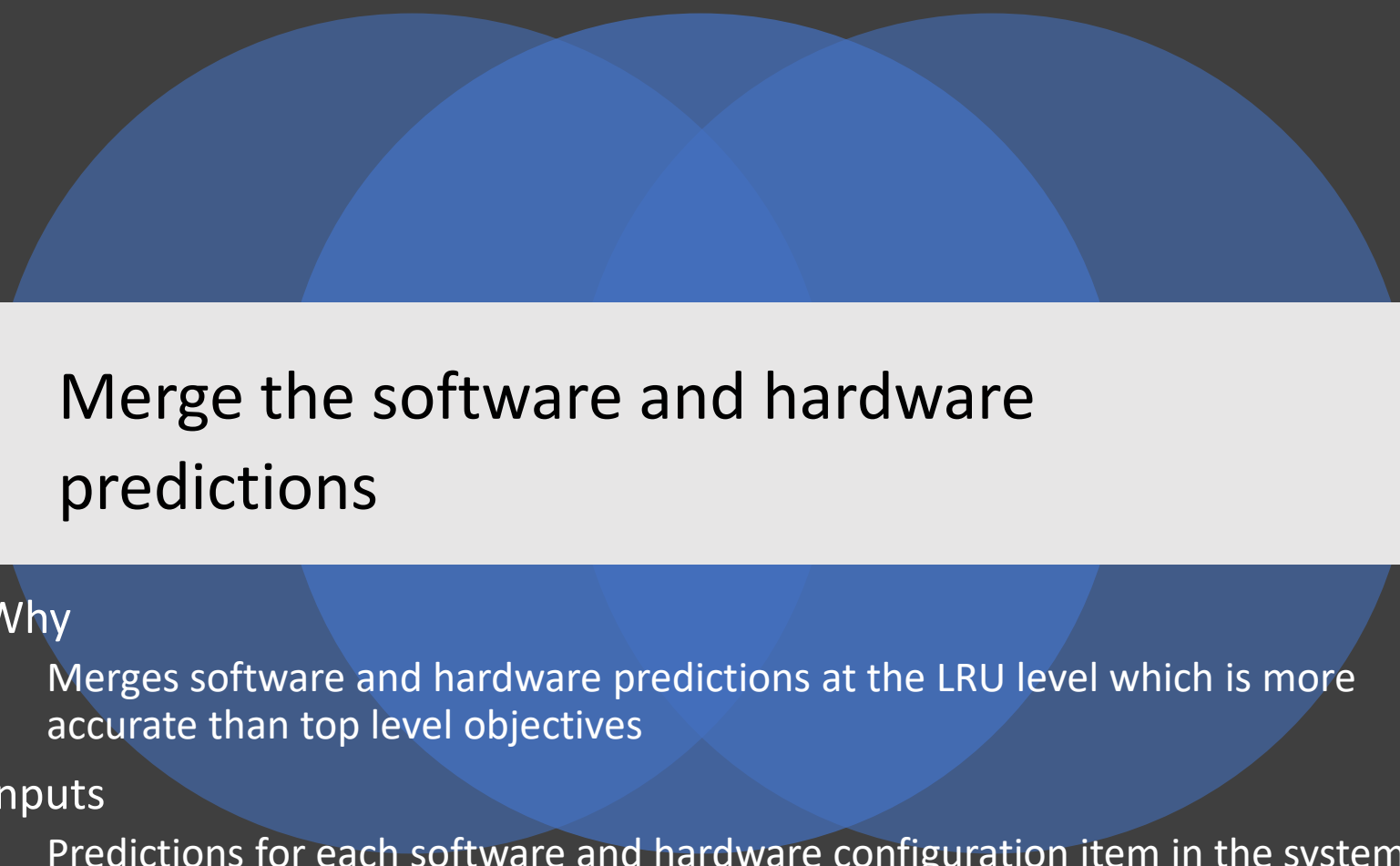
# Example

- Let's assume that the SOW is modified to require fault injection testing, and line coverage testing.
- As per IEEE 1633 clause 5.6 the risk of the software not meeting the objective is now "Very low"
  - That's because 100% of the code is exercised instead of about 40%
  - AND the code is exercised under faulted conditions as opposed to Happy Day scenarios
- This approach is what is required for FAA certified software and it is why commercial aircraft is the most reliable software on earth *even after the 737 Max software failures.*

# Review

You learned that if there is a difference between the realistic objective and the target objective some options are:

- Recommend that software organization makes more frequent smaller releases so that the software gets more reliability growth
- Recommend that reliability growth be extended by having more shifts or more test harnesses and that
  - There be no new features introduced during that reliability growth
  - The software organization either fix the defects found or document a viable workaround
  - The software be exercised as it would in operation
  - The scenarios be varied to ensure greater coverage of conditions, decision and data
- The test coverage be increased with
  - Go-No Go, Trajectory and boundary testing which is not relatively expensive
  - Fault injection testing is mandatory for all mission critical systems
  - Line/branch coverage thresholds
- Arbitrarily requiring the software organization to make up the difference doesn't work if the difference is more than 5%



# Merge the software and hardware predictions

## Why

Merges software and hardware predictions at the LRU level which is more accurate than top level objectives

## Inputs

Predictions for each software and hardware configuration item in the system

## Outputs

A system reliability prediction (differs from an objective which is a goal)

# Merge software reliability predictions

---

Previous sections covered how to establish a software allocation at the top software level

---

In this section, it is shown how to combine predictions at the LRU level

---

This type of merging can also be used to establish allocations for the software and hardware

---

These techniques require more knowledge concerning the system components and LRUs

---

Hence these techniques may be used once the system design is complete

# Merge methods

---

System reliability block diagram

---

Mission model

---

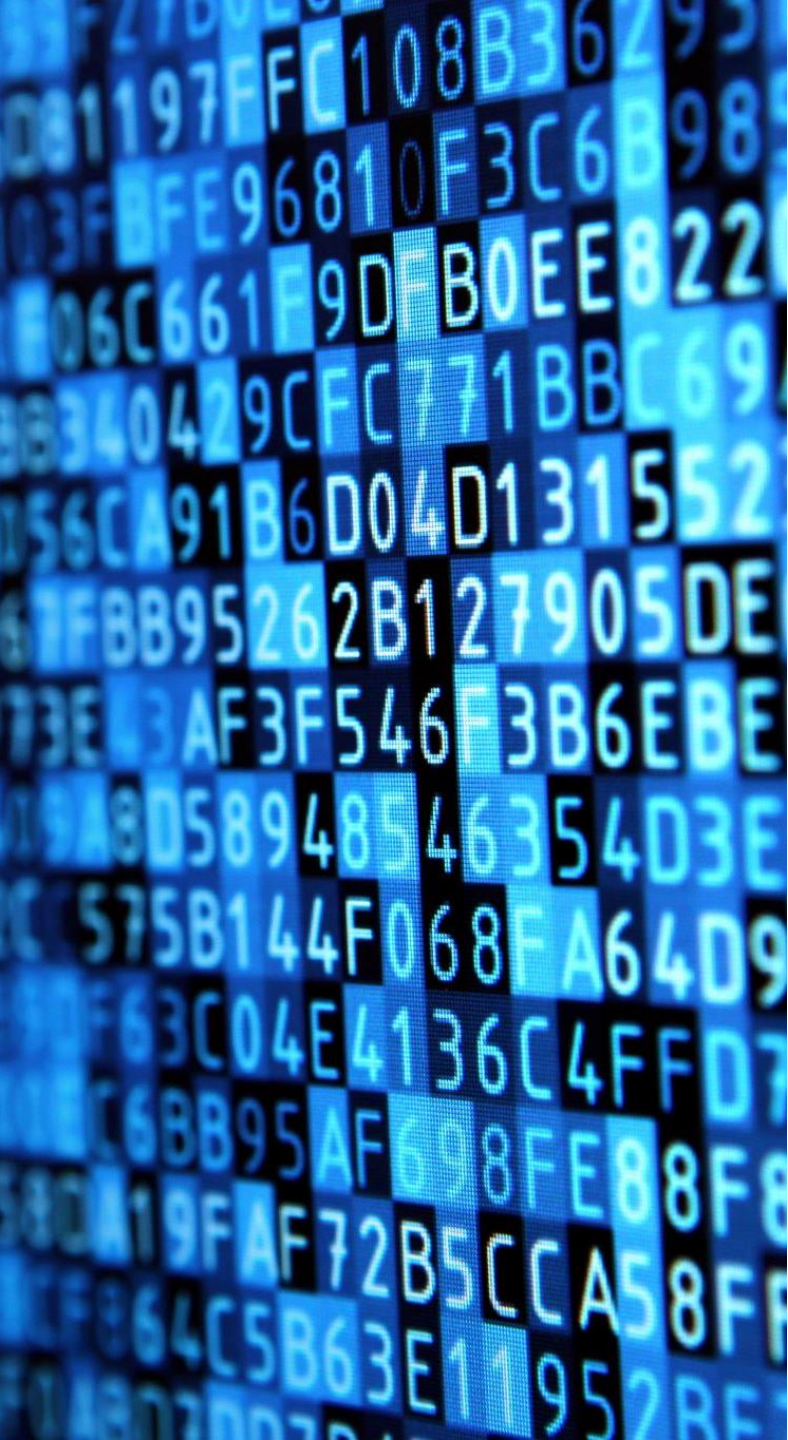
Use case model

---

Operational profile model

---

Fault tree model



# System reliability block diagram

Options for merging predictions

# RBD Inputs/Outputs

## Inputs

- The system reliability block diagram should already have a listing of each hardware item
- Predictions for each software configuration item
- Identification of physical software LRUs for each software configuration item
  - CSCI is software configuration item. However, there's not always 1:1 relationship to physical software LRU.
  - One CSCI may have multiple physical LRUs.
  - While it's not common, one CSCI may have <1 physical LRU if multiple companies are developing code for one LRU.

## Outputs

- System reliability block diagram with software and hardware

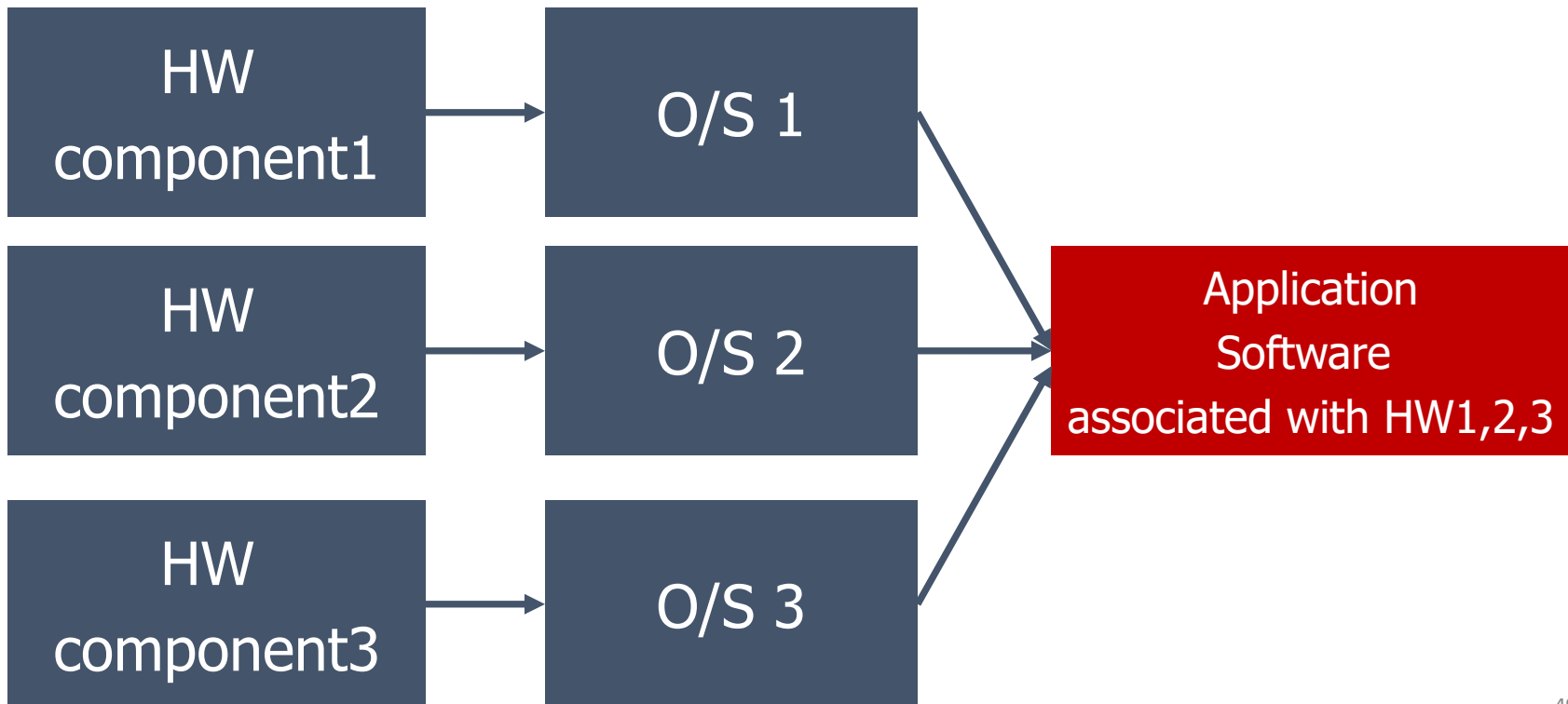
## Steps for adding software to an RBD

1. First assess the number of physical software LRUs. These are executables, applications, Dynamically Linked Libraries (DLLs).
2. If there is only one physical software LRU for all software CSCIs then there is one big block on the RBD which is in series with all hardware.
3. Assess relationship between each software LRU and each hardware configuration item.
4. If one software LRU supports multiple hardware subsystems, there could be an unnecessary risk due to poor cohesion.
5. Otherwise...For each hardware configuration item
  - Add the predictions for each of the software LRU to the reliability block diagram so that each software LRU is in series with the hardware that it supports
  - When hardware is redundant, the software is in series with the entire redundant configuration because the software is not redundant



# RBD when there is one physical software LRU

- When software is developed in “big blobs” the RBD can/will illustrate that software is a single point failure regardless of redundancy
- This is an often overlooked and terrible engineering practice



# RBD when one physical software LRU supports multiple hardware subsystems



Identify all physical software LRUs and hardware subsystems



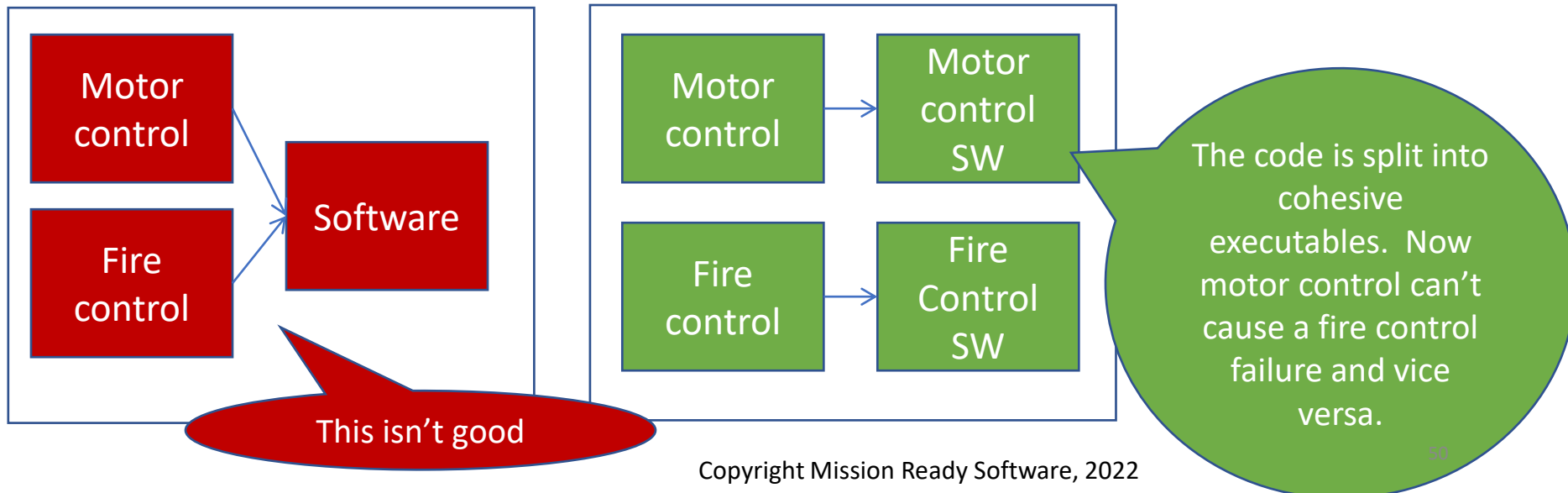
Identify all software components that are associated with > 1 hardware subsystem



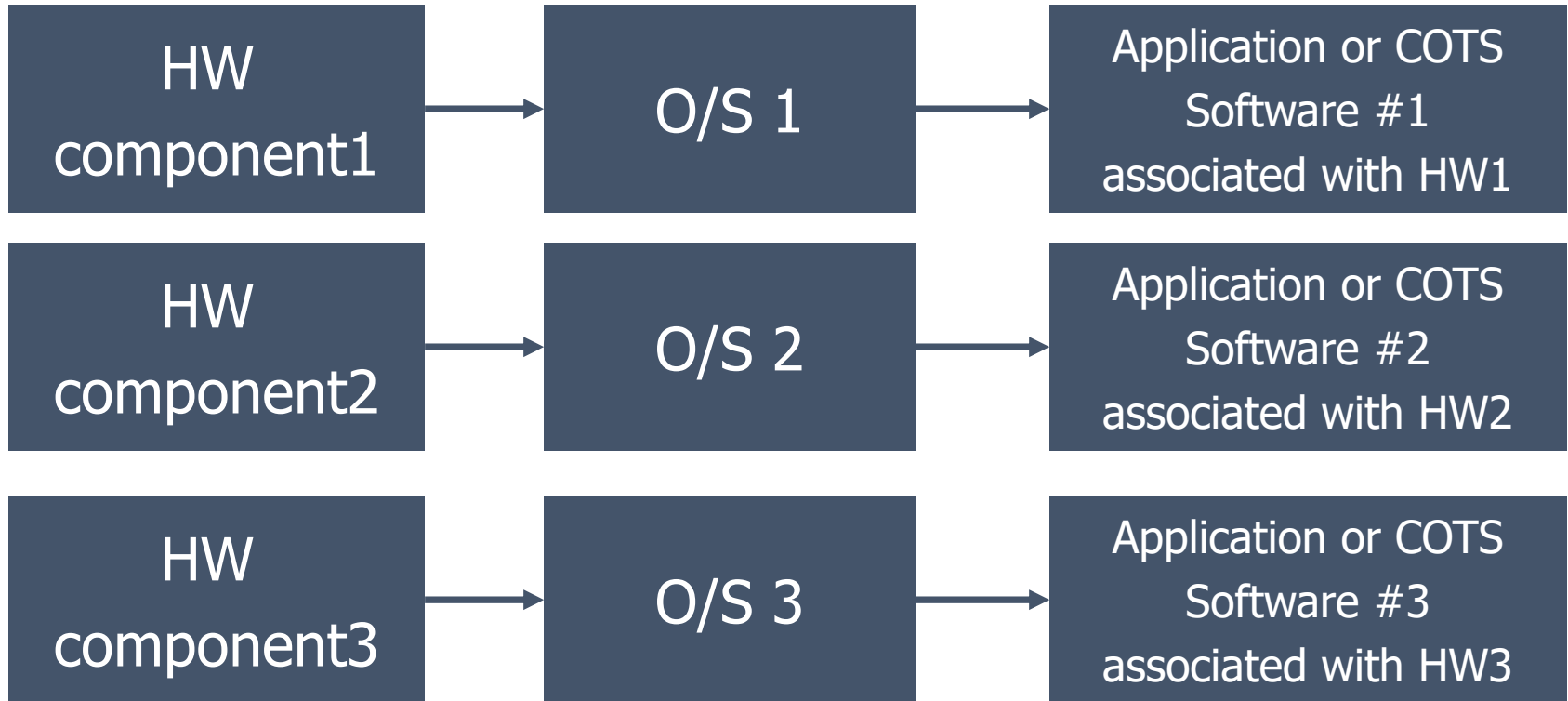
These software items may have unrelated functions in the same executable which can have negative impact on prediction



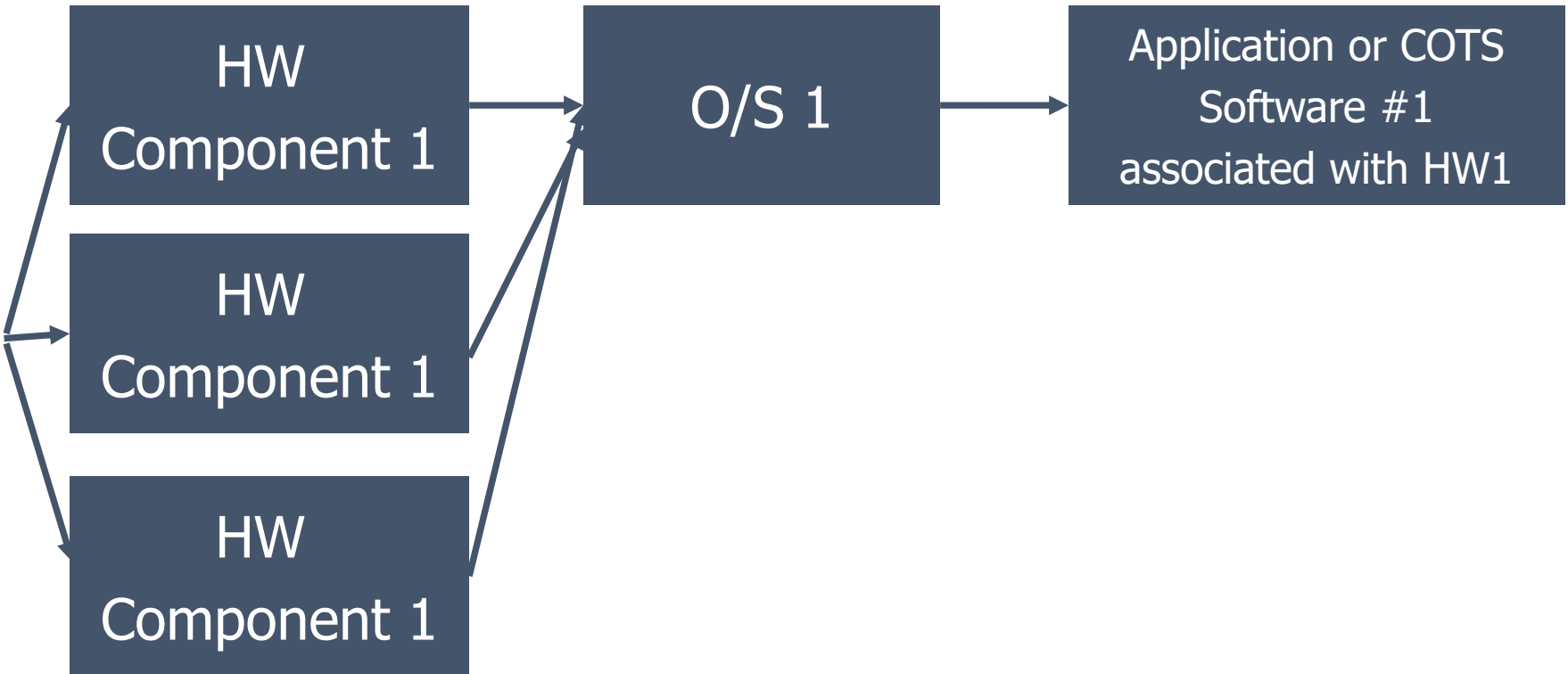
Partitioning unrelated software programs can reduce system failure rate



# Reliability Block Diagram when software and associated hardware are not redundant



# Reliability Block Diagram when software and associated hardware is redundant



# Notes about Software redundancy

It's unlikely that any of the software components are "redundant"

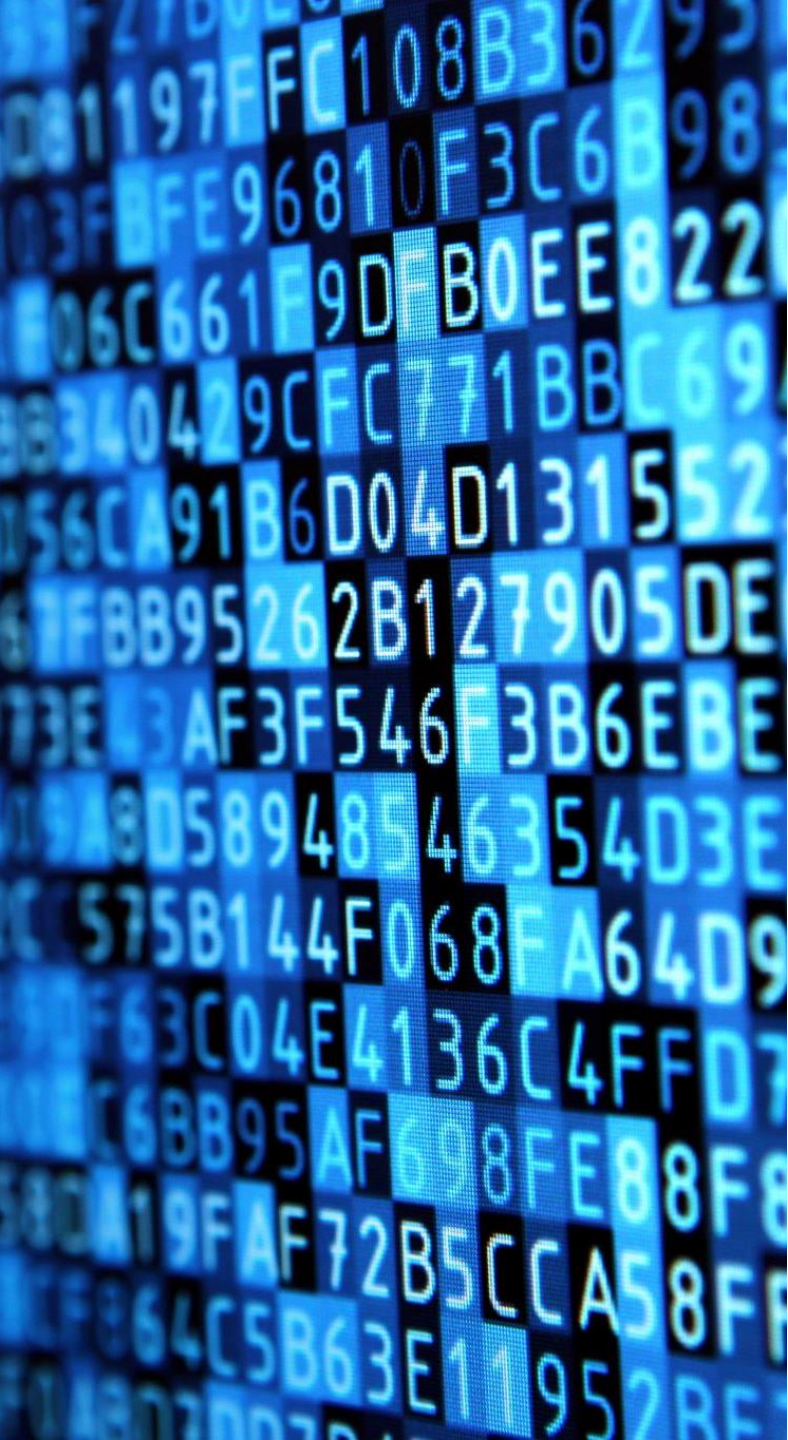
- Big difference between having multiple copies of the same software installed and "redundancy"

Software is redundant if and only if there is N version programming

- Different versions are implemented by different software organizations to do the same functions
- Generally very expensive

Otherwise, redundancy applies to hardware and not to software

It should not be "assumed" that hardware redundancy will address all software failures

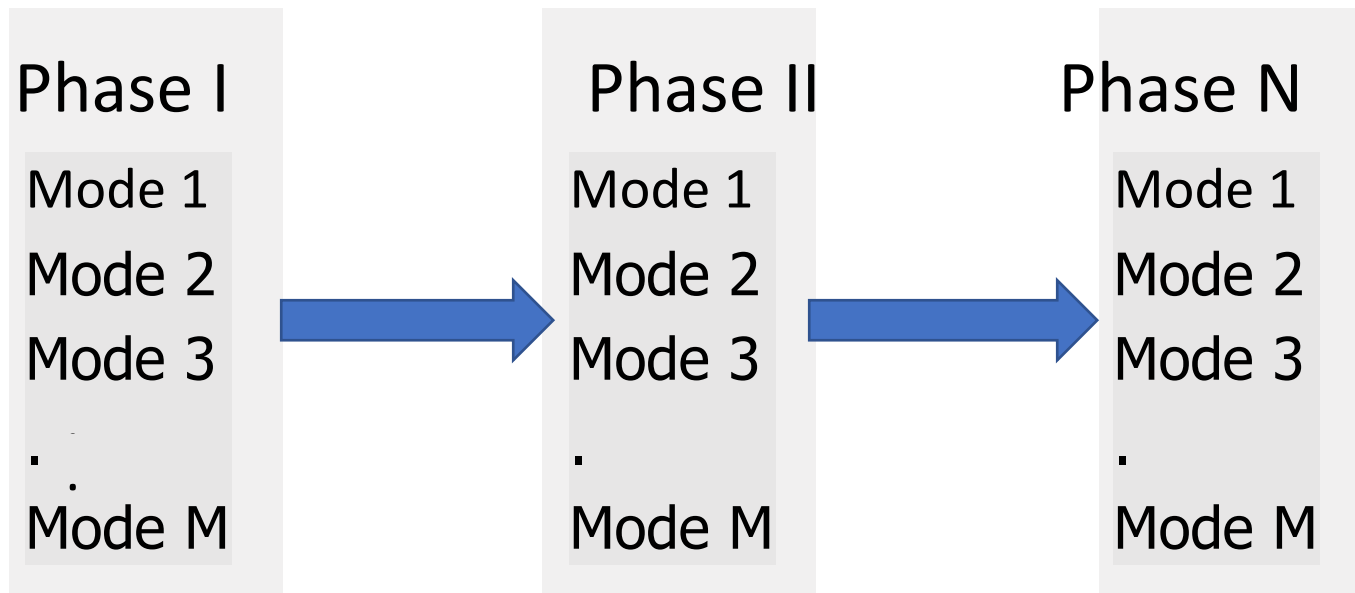


# Mission model

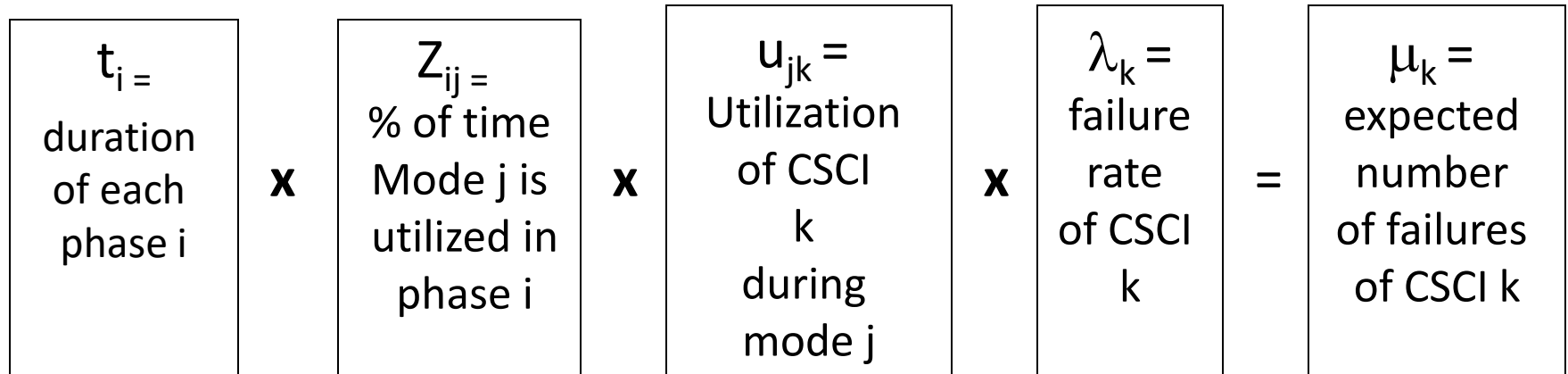
Options for merging predictions

# Mission Reliability Model

- Used when there is a defined start and end
  - Landing gear
  - Aircraft
  - Space shuttle
  - Dishwashers



# Mission Model





# Example of mission model – Step 1

Phase	Duration	Mode utilizations			
		Idle	Scan	Track	Maintenance
Startup	0.1	1	0	0	0
Taxi	0.1	1	0	0	0
Climb	0.2	0.5	0.5	0	0
Loiter	1	0	0.8	0.2	0
Attack	0.3	0	0.333333	0.666666667	0
Return	0.2	0.5	0.5	0	0
Land	0.1	1	0	0	0
Shutdown	0.2	0	0	0	1
Total duration	2.2				

Identify the phases

Identify the duration of each phase -  $t_i$

Identify the mode utilizations of each mode  $z_i$  - The sum of mode utilizations must equal 1 across each phase

## Example of mission model – Step 2

Phase	Duration	Mode utilizations				Maintenance
		Idle	Scan	Track		
Startup	0.1	0.1	0	0	0	
Taxi	0.1	0.1	0	0	0	
Climb	0.2	0.1	0.1	0	0	
Loiter	1	0	0.8	0.2	0	
Attack	0.3	0	0.1	0.2	0	
Return	0.2	0.1	0.1	0	0	
Land	0.1	0.1	0	0	0	
Shutdown	0.2	0	0	0	0.2	
<b>Total duty cycle of each mode in hours</b>	<b>2.2</b>	<b>0.5</b>	<b>1.1</b>	<b>0.4</b>	<b>0.2</b>	

1. Determine the duty cycle of each mode in each phase by multiplying the phase times  $t_i$  by the mode utilizations  $z_i$
2. Sum up the duty cycles of each mode across all phases

## Example of mission model – Step 3

	Idle	Scan	Track	Maintenance
Duty cycle of each mode in hours	0.5	1.1	0.4	0.2
Configuration item	Active in this mode?			
Executive	1	1	1	1
Test	0	0	0	1
Scan	0	1	0	0
Track	0	0	1	0
Calibration	1	0	0	1

1. Identify which software LRUs are active in each mode.
2. “1” means active and “0” means not active

## Example of mission model – Step 4

		Idle	Scan	Track	Maintenance
Configuration item	Failure rate (hours)	Failure rate x activation			
Executive	0.0001	1	1	1	1
Test	0.00005	0	0	0	1
Scan	0.00001	0	1	0	0
Track	0.00002	0	0	1	0
Calibration	0.00003	1	0	0	1

1. Identify the predicted failure rate of each of the software LRUs using the methods in module 8 (early in development) or module 9 (during testing)

# Example of mission model – Step 5

Configuration item	Failure rate (hours)	Failure rate x activation			
		Idle	Scan	Track	Maintenance
Executive	0.0001	0.0001	0.0001	0.0001	0.0001
Test	0.00005	0	0	0	0.00005
Scan	0.00001	0	0.00001	0	0
Track	0.00002	0	0	0.00002	0
Calibration	0.00003	0.00003	0	0	0.00003
Sum of failure rates		0.00013	0.00011	0.00012	0.00018
Duty cycle of each mode		0.5	1.1	0.4	0.2
Failure rate x duty cycle		0.000065	0.000121	0.000048	0.000036

1. Multiply the LRU failure rates by the activation matrix for each mode
2. Sum the failure rates of each mode
3. Multiply the failure rate of each mode by the duty cycle of each mode

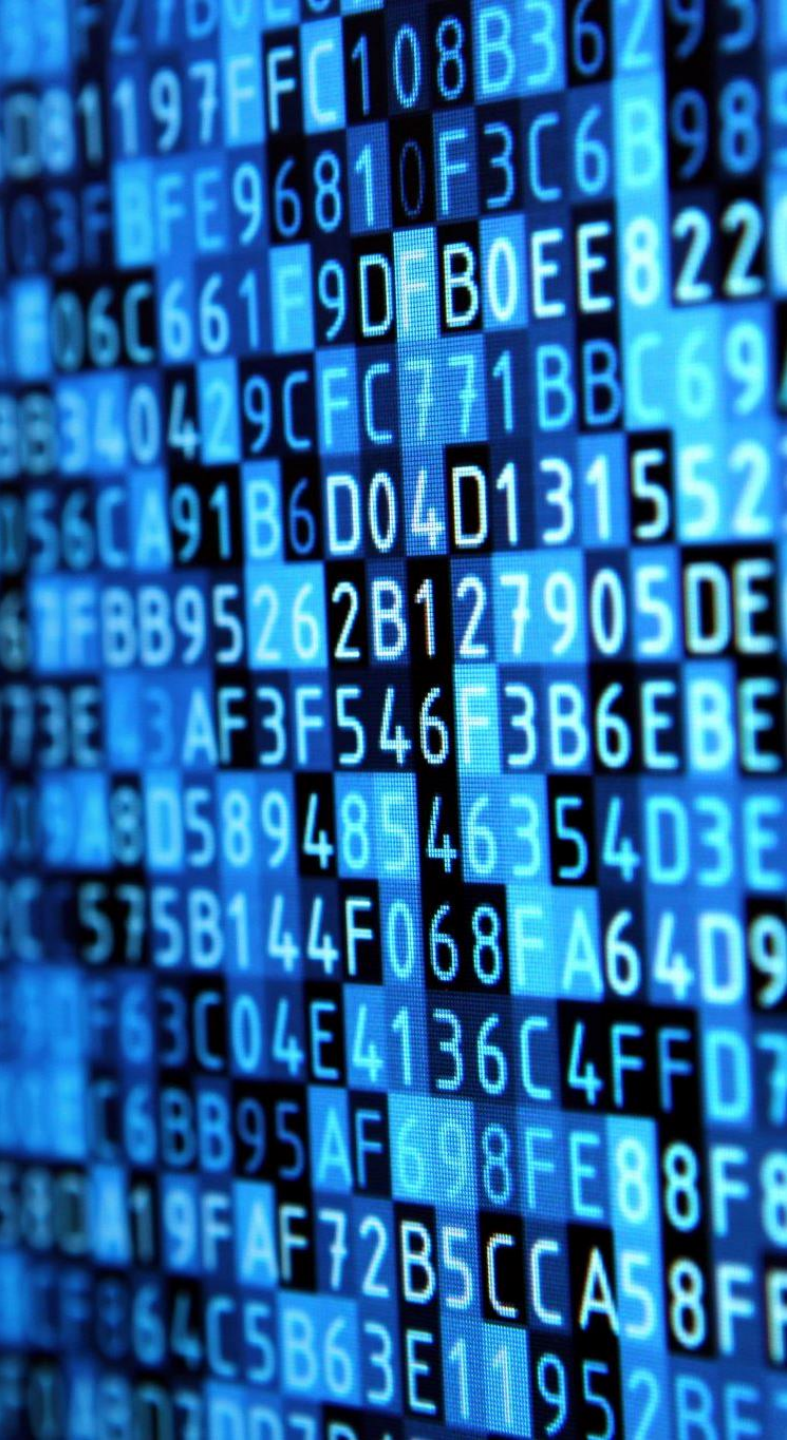
# Example of mission model – Step 6

Configuration item	Failure rate (hours)	Failure rate x activation			
		Failure rate x duty cycle	0.000065	0.000121	0.000048

1. Sum up the failure rate x duty cycle cells for each of the modes. This is the predicted failures during the mission. In this example it is .027 failures per hour.
2. Divide the predicted failures per mission by the total mission time. In this example the total mission time is 2.2 hours. So the predicted failure rate of the mission for the software is .00012273 failures per hour
3. The reliability of the mission is  $\exp(-.00012273 * 2.2) = .99973004$

# Notes

- This example did not merge in the hardware failure rates.
- This can be done in step 5



# Use case model

Options for merging predictions



## This model combines failure rates by use case

### Use case model

- Identify all software use cases
- Predict the size of the code executed for each use case. Use the models in IEEE 1633 Recommended Practices for Software Reliability, predict the reliability metrics for each use case.
- Predict the mission time or up time of each use case
- Combine the software use case reliability with the hardware reliability predictions for hardware that is active for each use case to establish reliability for each use case.
- Model resembles the mission model except that instead of predicting failure rate of each CSCI, predict failure rate of each use case

# Example

<b>Use case</b>	<b>New or modified 1000 SLOC</b>	<b>Predicted failure rate in hours as per IEEE 1633</b>	<b>Mission time in hours</b>
Use case A	100	0.00001	0.2
Use case B	200	0.00002	0.3
Use case C	150	0.000015	0.1
Use case D	50	0.000005	0.9
Use case E	25	0.0000025	0.4
Total			1.9

There are 5 use cases. The size of each use case is predicted based on the code that's active for that use case. Using the size and the models discussed in IEEE 1633 the failure rate in hours of each use case is predicted. The mission time of each use case is also estimated based on the mission profile

# Example

Use case	Predicted failure rate in hours as per IEEE 1633	Failure rates of hardware configuration items active in this use case execution in failures per hour						
		HW A	HW B	HW C	HW D	HW E	HW F	HW G
Use case A	0.00001	0.0001				0.000011		0.000001
Use case B	0.00002		0.00002			0.000011		0.000001
Use case C	0.000015				0.000001			0.000001
Use case D	0.000005			0.000015			0.000002	0.000001
Use case E	0.0000025	0.0001			0.000001		0.000002	0.000001

Next identify which hardware configuration items are active in each use case. Put the HW failure rate on the matrix if it's active in that use case.

# Example

Use case	Total failure rate of use case	Predicted failure rate in hours as per IEEE 1633	Failure rates of hardware configuration items active in this use case execution in failures per hour						
			HW A	HW B	HW C	HW D	HW E	HW F	HW G
Use case A	0.000122	0.00001	0.0001				0.000011		0.000001
Use case B	0.000052	0.00002		0.00002			0.000011		0.000001
Use case C	0.000017	0.000015				0.000001			0.000001
Use case D	0.000023	0.000005			0.000015			0.000002	0.000001
Use case E	0.0001065	0.0000025	0.0001			0.000001		0.000002	0.000001

Sum up all failure rates in each use case

# Example

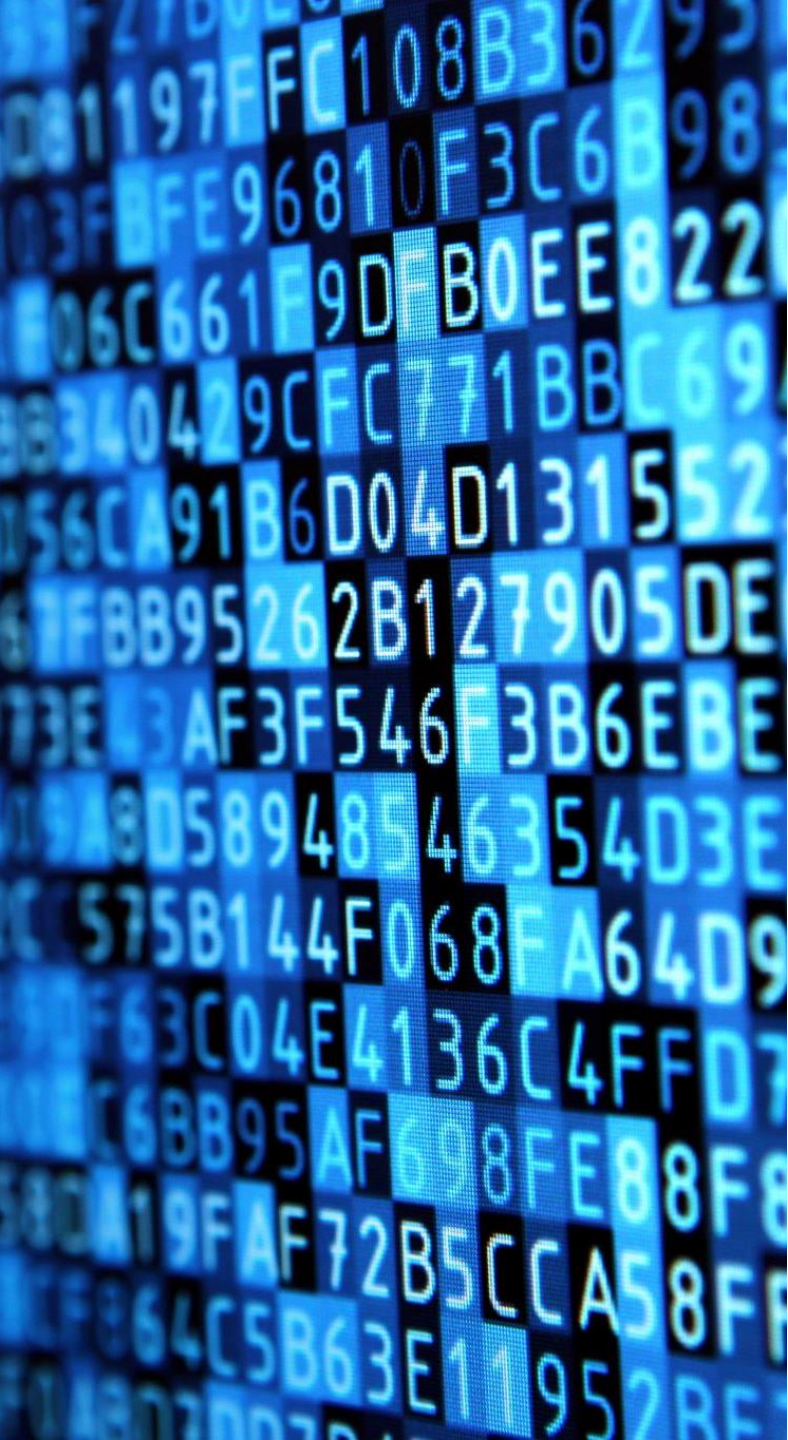
Use case	Sum of HW and SW failure rates for each use case	Mission time of use case	Predicted failures over mission
Use case A	0.000122	0.2	0.0000244
Use case B	0.000052	0.3	0.0000156
Use case C	0.000017	0.1	0.0000017
Use case D	0.000023	0.9	0.0000207
Use case E	0.0001065	0.4	0.0000426
	Total	1.9	0.000105
	Failure rate = failures/mission time		0.000055
	Reliability of mission		99.9895006%

Predicted failures of each use case = use case failure rate \* mission time of use case

Total predicted failures = .000105

The divide by the total mission time of 1.9 hours = .0000555 FPH

Reliability of mission =  $\exp(-1.9 \text{ hours} * .0000555 \text{ FPH}) = 99.9895\%$



# Operational profile model

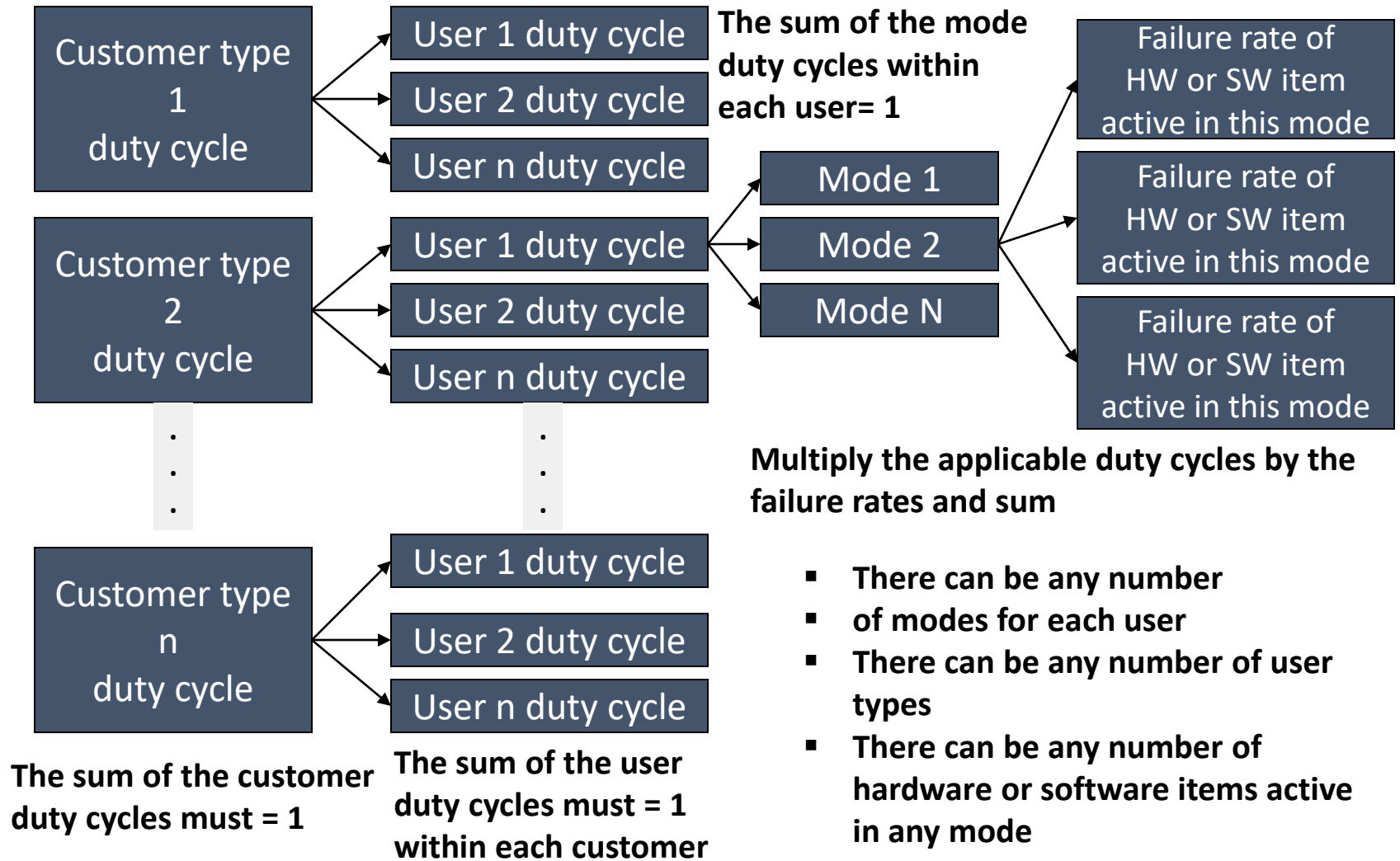
Options for merging predictions

# Operational Profile Model

- Used when
  - > 1 end user type and/or more than one customer type
  - How the software is used varies by end user/customer

Profile	Description
Customer	Used to describe multiple customers
User profile	Used to describe multiple user types
System mode profile	Used to describe behavior of execution
Functional profile	Used to describe relative usage of different software components

# Operational Profile Model





# Example of a commercial desk top printer

Customer profile	Small businesses											Copy shops														
	70%											30%														
User profile	Professionals (lawyers, accountants, etc.)						High tech professionals (engineers, computer					Walk in customer			Copy shop employee											
	40%						60%					25%			75%											
System mode profile	Printing		Scanning		Faxing		Printing		Scanning		Faxing	Printing		Scanning		Faxing	Printing		Scanning		Faxing					
	50%		25%		25%		60%		40%		0%	95%		5%		0%	10%		30%		60%					
Functional profile	Legal size	11x17	8.5x11	Auto feed	Manual	Auto dial	Manual dial	Legal size	11x17	8.5x11	Auto feed	Manual		Legal size	11x17	8.5x11	Auto feed	Manual	Auto dial	Manual dial						
	80%	0%	20%	50%	50%	20%	80%	5%	30%	65%	60%	40%		15%	15%	70%	0%	100%		15%	15%	70%	50%	50%	0%	100%
Operational profile	11.20%	0.00%	2.80%	3.50%	3.50%	1.40%	5.60%	1.26%	7.56%	16.38%	10.08%	6.72%		1.07%	1.07%	4.99%	0.00%	0.38%		0.34%	0.34%	1.58%	3.38%	3.38%	0.00%	13.50%

In this example the software with the biggest duty cycle is printing high tech documents (i.e. diagrams) on 8.5x11, faxing in manual dial mode and printing legal documents on legal sized paper . Yet the software test team didn't think to test either of these cases. The software stalled when printing the very large high tech documents and the very long legal documents.

# Example of a commercial desk top printer

Customer profile	Small business		Copy shop	
	Professional	High tech	Walk in	Store employee
Customer profile duty cycle	0.7	0.7	0.3	0.3
User duty cycle	0.4	0.6	0.25	0.75
Print duty cycle per user	0.5	0.6	0.95	0.1
Legal sized paper duty cycle	0.8	0.05	0.15	0.15
11x17 paper duty cycle	0	0.3	0.15	0.15
8.5x11 paper duty cycle	0.2	0.65	0.7	0.7
Scan duty cycle per user	0.25	0.4	0.05	0.3
Auto scan	0.5	0.6	0	0.5
Manual scan	0.5	0.4	1	0.5
Fax duty cycle per user	0.25	0	0	0.6
Auto fax	0.2	0	0	0
Manual fax	0.8	0	0	1

The profiles are rearranged so that duty cycle of each function is on a different row

Multiply the duty cycle for each function by its user duty cycle and customer profile duty cycle as shown on next page

# Example of a commercial desk top printer

	Professional	High tech	Walk in	Employee	Total duty cycle
Total	0.28	0.42	0.075	0.225	1
Legal sized paper duty cycle	0.112	0.0126	0.010688	0.003375	0.1386625
11x17 paper duty cycle	0	0.0756	0.010688	0.003375	0.0896625
8.5x11 paper duty cycle	0.028	0.1638	0.049875	0.01575	0.257425
Auto scan	0.035	0.1008	0	0.03375	0.16955
Manual scan	0.035	0.0672	0.00375	0.03375	0.1397
Auto fax	0.014	0	0	0	0.014
Manual fax	0.056	0	0	0.135	0.191
All	0.28	0.42	0.075	0.225	1

- The duty cycle is now summed up for each user and each function
- Next identify the hardware and software elements
  - Sheet handler
  - Scan bed
  - Paper tray
  - Fax keyboard
  - Telephone port
  - Print software
  - Scan software
  - Fax software

# Example of a commercial desk top printer

	Total duty cycle	Sheet handler	Scan bed	Paper tray	Fax keyboard	Telephone port	Print CSCI	Scan CSCI	Fax CSCI
<b>Predicted Failure rate</b>		.0005	.0009	.000008	.000007	.000001	.0002	.0015	.00002
<b>Legal sized paper duty cycle</b>	0.1386625	x	x	x			x		
<b>11x17 paper duty cycle</b>	0.0896625	x	x	x			x		
<b>8.5x11 paper duty cycle</b>	0.257425	x	x	x			x		
<b>Auto scan</b>	0.16955	x	x					x	
<b>Manual scan</b>	0.1397		x					x	
<b>Auto fax</b>	0.014	x	x		x	x			x
<b>Manual fax</b>	0.191	x	x		x	x			x

- Determine the predicted failure rates for both software and hardware
- Identify which configuration items are active as shown above

# Example of a commercial desk top printer

	Duty cycle per function mode	Sheet handler	Scan bed	Paper tray	Fax keyboard	Telephone port	Print CSCI	Scan CSCI	Fax CSCI
<b>Predicted Failure rate</b>		.0005	.0009	.000008	.000007	.000001	.0002	.0015	.00002
<b>Legal sized paper duty cycle</b>	0.1386625	1	1	1	0	0	1	0	0
<b>11x17 paper duty cycle</b>	0.0896625	1	1	1	0	0	1	0	0
<b>8.5x11 paper duty cycle</b>	0.257425	1	1	1	0	0	1	0	0
<b>Auto scan</b>	0.16955	1	1	0	0	0	0	1	0
<b>Manual scan</b>	0.1397	0	1	0	0	0	0	1	0
<b>Auto fax</b>	0.014	1	1	0	1	1	0	0	1
<b>Manual fax</b>	0.191	1	1	0	1	1	0	0	1
<b>Duty cycle per CI</b>		<b>0.8603</b>	<b>1</b>	<b>0.48575</b>	<b>0.205</b>	<b>0.205</b>	<b>0.48575</b>	<b>0.30925</b>	<b>0.205</b>

- Sum multiply the failure rates for each component to establish the duty cycle per configuration item

# Example of a commercial desk top printer

	Sheet handler	Scan bed	Paper tray	Fax keyboard	Telephone port	Print CSCI	Scan CSCI	Fax CSCI
Failure rate	0.0005	0.0009	0.000008	0.000007	0.000001	0.0002	0.0015	0.00002
Duty cycle	0.8603	1	0.48575	0.205	0.205	0.48575	0.30925	0.205
Adjusted failure rate	4.3015 E-4	0.0009	3.886 E-6	1.435 E-6	2.05 E-7	9.72E-05	4.64 E-4	4.1E-6
Total	0.001900801 failures per hour							

- Compute the adjusted failure rate for each configuration item and then add all for a total failures pe hour



# Fault tree model

Options for merging predictions


Refer to the Software Fault Tree Analysis class for more information

# Review

You learned several techniques for merging software and hardware predictions

- Reliability Block Diagram – The software LRU is in series with the hardware it support
- Mission model, Use case Model, Operational Profile model – These merge hardware and software predictions based on their corresponding duty cycle
- Fault tree – Merge the software failure rates onto a system fault tree
- We did not cover the Markov model which is in the System Software Reliability Assurance Notebook, 1997, USAF Rome Labs.
- Pros and Cons
  - Use case may be more specific and therefore more accurate
  - Depends on the software group being able to estimate size by use case
  - Mission model might be needed if software group isn't estimating by use case
  - OP model useful if multiple user roles or when the user roles don't have same duty cycle
  - Fault tree model is useful if that's how they are predicting the hardware reliability
  - Markov model is useful for a continuously operating system ( security, surveillance, etc.). Only if the contractor has a Markov model tool.





## Allocate the top software objective to the software LRUs and hardware LRUs

### Why

It allows for different software/hardware groups to work towards a goal

### Inputs

Predictions for each software configuration item in the system

### Outputs

A portion of the overall system objective that must be met by a particular CSCI or HWCI

# Methods for allocating down to each software LRU

Allocations to the CSCI or HWCI level are done for the purposes of establishing goals for each engineering department to work towards.

It does not matter to the customer how they do the allocation, only that they have identified a way for each engineering organization to work towards their portion of the goal.

## Bottom up

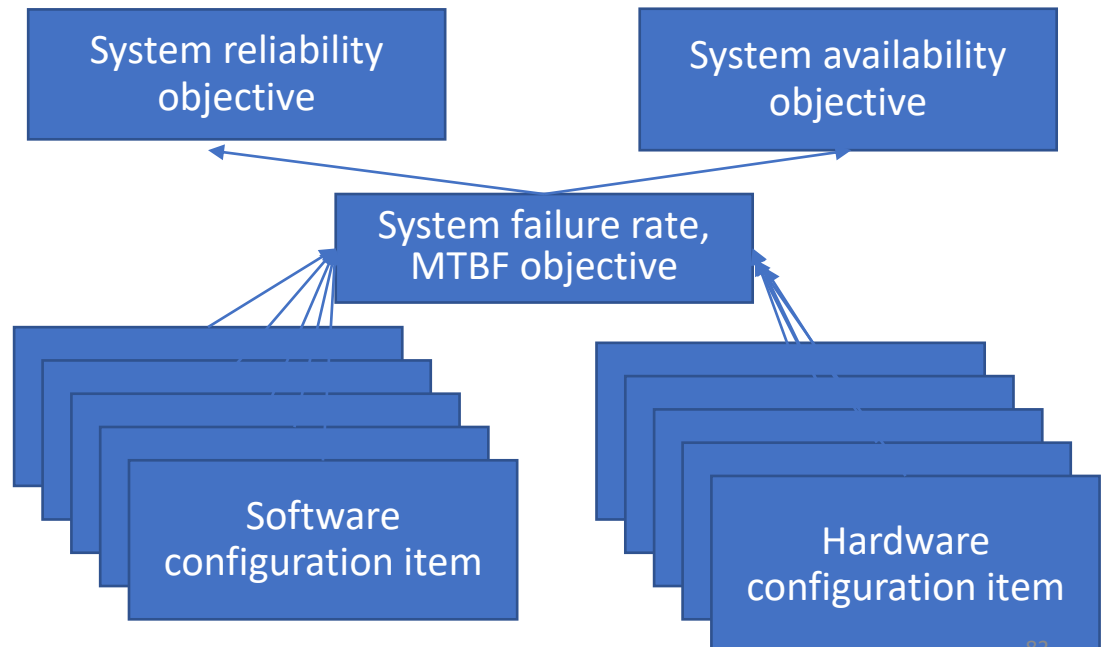
- Start with predictions of each LRU
- Allocations are based on relative percentage of each LRU to system objective

## Top down methods

- Each LRU is allocated portion of its subsystem allocation
- Each LRU is allocated portion of either HW or SW allocation

- Bottom up allocations start with the predictions of every system component including software
- Every prediction is then allocated based on its relative prediction
- Advantages of bottom up allocation
  - Each software or hardware LRU allocation depends only on its portion of the total system
  - So, if the SW portion was computed incorrectly, each software LRU is still allocated based on its contribution to the system objective

Bottom up allocations



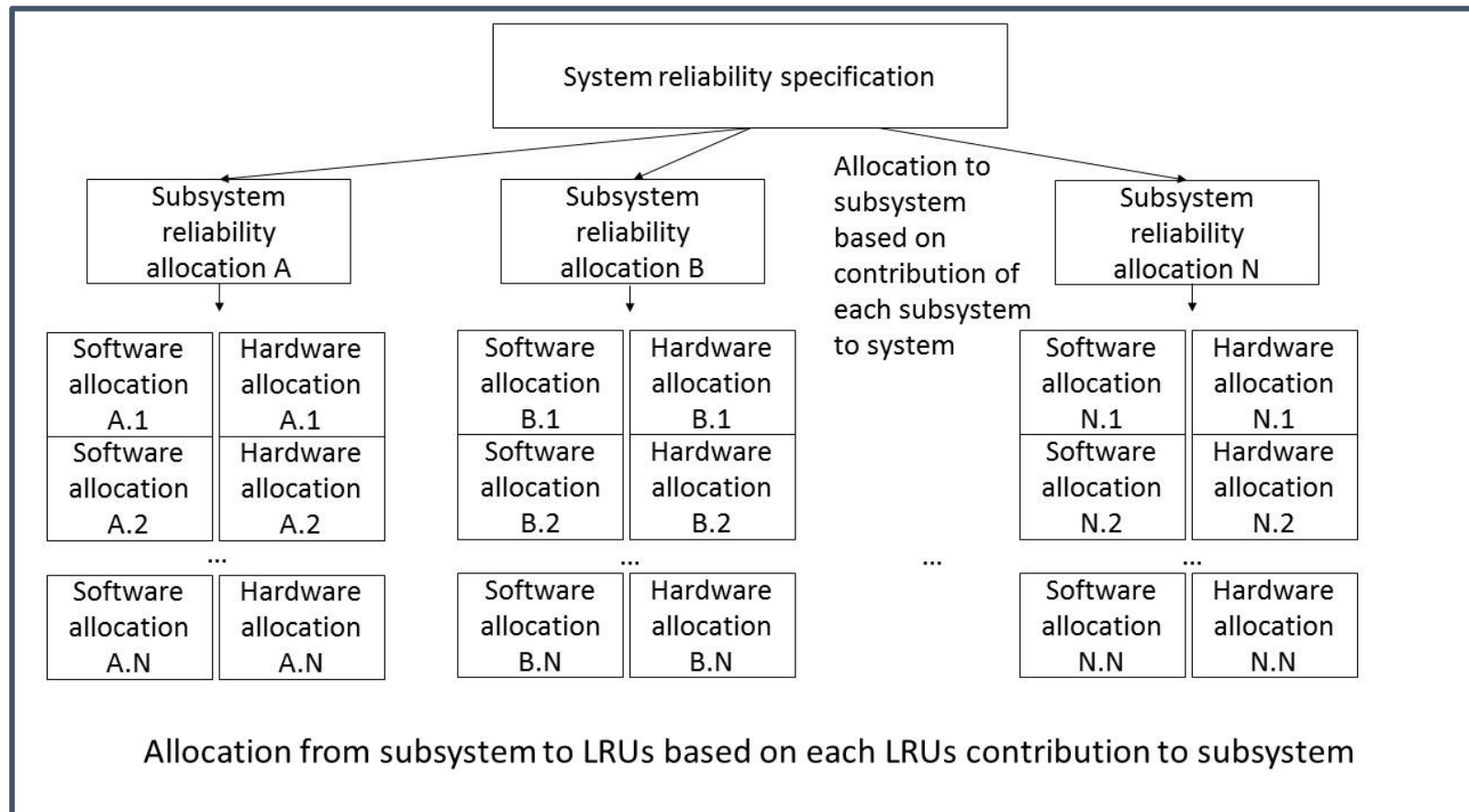
## Bottom up allocation example

- These are the predicted failure rates for each component in the desktop printer
- The objective is .001 failures per hour for the desktop printer
- The allocations are established both as unweighted and weighted by the operational profile
- The unweighted allocation doesn't take into consideration how much the configuration item is used
- The weighted allocation is preferred

	Sheet handler	Scan bed	Paper tray	Fax keyboard	Telephone port	Print CSCI	Scan CSCI	Fax CSCI	total
Failure rate per hour not weighted by duty cycle	0.0005	0.0009	8E-06	7E-07	1E-06	0.0002	0.0015	0.00002	0.003136
Allocation unweighted	1.59E-04	2.87E-04	2.55E-06	2.23E-06	3.19E-07	6.38E-05	4.78E-04	6.38E-06	0.001
Failure rate per hour weighted by duty cycle	4.3015 E-4	0.0009	3.89E-06	1.435 E-6	2.05 E-07	9.715 E-05	4.63875E-04	4.1E-06	1.900801 E-03
<b>Allocation weighted by duty cycle</b>	<b>2.26E-04</b>	<b>4.73E-04</b>	<b>2.04E-06</b>	<b>7.55E-07</b>	<b>1.08E-07</b>	<b>5.11E-05</b>	<b>2.44E-04</b>	<b>2.16E-06</b>	<b>0.001</b>

# Top down allocation method #1

- Method #1 - Each LRU allocation is based on its portion of the subsystem comprising the system
- Advantage of option #1
  - If the subsystems are being developed by multiple contractors or sites, this approach allows for each subsystem versus HW/SW allocation



Example of top down allocation method #1

Subsystem	Failures from past history	Relative portion of objective	Allocation
A	100	50%	.00005
B	75	37.5%	.0000375
C	25	12.5%	.0000125

- There are 3 subsystems in a system
- Based on past history Sub-system A will have half the failures, B will have 37.5% and C will have 12.5%
- The goal is a failure rate of .0001 failures per hour for the entire system
- Each subsystem receives a proportionate amount of the system goal
- Next the subsystem goal is allocated down to the sub-system components using past historical data

Example of top down allocation method #1

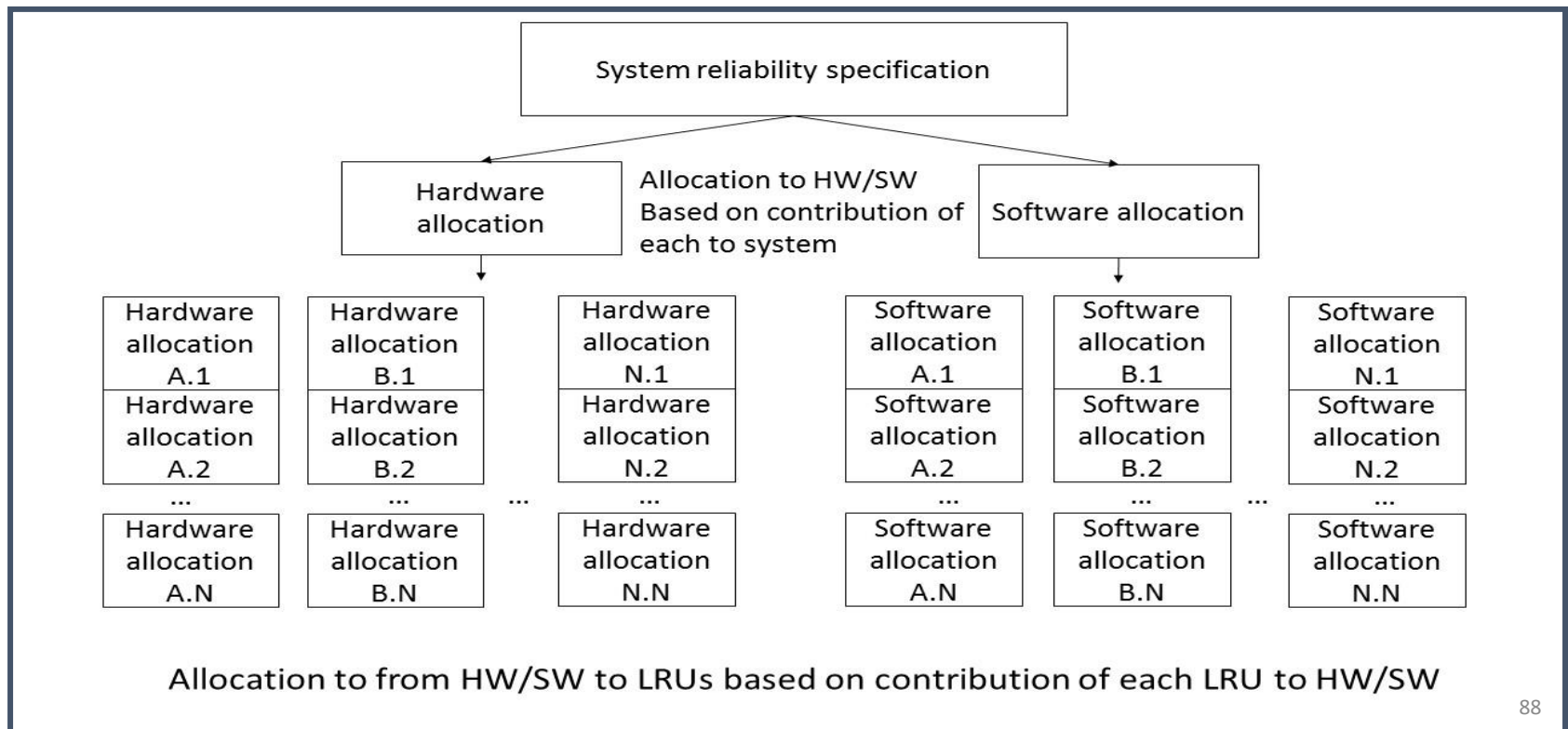
Subsystem	Failures from past history	Relative portion of objective	Allocation
<b>A</b>	<b>100</b>	<b>50.00%</b>	<b>0.00005</b>
HWA1	20	20.00%	0.00001
HWA2	25	25.00%	0.0000125
HWA3	5	5.00%	0.0000025
SWA1	50	50.00%	0.000025
<b>B</b>	<b>75</b>	<b>37.50%</b>	<b>0.0000375</b>
HWB1	40	53.33%	0.00002
HWB2	10	13.33%	0.000005
SWB1	10	13.33%	0.000005
SWB2	15	20.00%	0.0000075
<b>C</b>	<b>25</b>	<b>12.50%</b>	<b>0.0000125</b>
HWC1	10	40.00%	0.000005
HWC2	5	20.00%	0.0000025
SWC1	10	40.00%	0.000005

- Sub-system A,B and C are further allocated to their HW and SW components based on past field data

For each of the subsystems, use the past failure data to determine a percentage allocation and then multiply by the failure rate allocation for the subsystem

# Top down allocation method #2

- Method #2 Each LRU allocation is based on its portion of the HW or SW allocation
- Advantages - If all of the software LRUs are being developed by one organization this allocation method could be easier to manage
- Disadvantages
  - Total software allocation is established by the “leftover” method which means it gets whatever is leftover from the hardware. This method is not recommended unless there is sufficient evidence that the software portion of the allocation is feasible





Example of top down allocation method #2

LRUs	Demonstrated failures on past system	Relative portion of the system objective	Allocated failure rate in hours
HW	115	58%	0.0000575
SW	85	43%	0.0000425

- Using the same example, in the recent past the hardware accounted for 60% of the total failures while the software accounted for 40%
- So
  - All hardware LRUs combined must meet  $.6 * .0001$  failures per hour
  - All software LRUS combined must meet  $.4 * .0001$  failures per hour
- The decomposition to each software LRU is shown next

## Example of top down allocation method #2

SW LRU	Demonstrated failures on past system	Relative portion of the system objective	Allocated failure rate from past history	Predicted failure rate in hours	Relative portion of the system objective	Allocated failure rate from prediction in hours
SWA1	40	47%	0.00002	0.000023	46%	0.00001955
SWB1	22	26%	0.000011	0.000017	34%	0.00001445
SWB2	17	20%	0.0000085	0.000006	12%	0.0000051
SWC1	6	7%	0.000003	0.000004	8%	0.0000034
Total	85	1	0.0000425	0.00005	1	0.0000425

- The software received an allocation of the system of .0000425 failures per hour
- Each software configuration item receives its portion of the SW objective based on either its past history or it's predicted failure rate
- Since the predictions take into account the most recent size estimations, the predictions and are almost 18% higher than the historical data
- Using the relative portions in the second to last column corrects the relative difference between predicted and historical but does not correct the absolute difference. The software components are likely to be under predicted by a combined total of 17.6%.

Determine whether software or hardware configuration items can meet objective

If the allocation is more than 10% from past history or the predictions it's likely infeasible

Once the objective has been driven down to each software component determine

- Whether or not the objective is feasible with given time and budget
- Whether or not the objective is feasible with any level of time and budget

Since software grows at about 10-12% per year, the objective may not be feasible

- Module 8 and Module 9 discuss tradeoffs that can be made when the allocation isn't feasible

Perform tradeoffs when objective cannot be met

## Inputs

- Allocated versus predicted reliability figures of merit
- Known development practices as per the survey models

## Outputs

- “What if” scenarios
  - Adjustments to system architecture
    - Compartmentalizing software based on hardware it supports
    - Decomposition of any “big blob” software LRUs
- Adjustments to software LRUs shown previously
  - Adjustments for effective size
  - Adjustments for defect density (inherent risks, personnel, techniques, process)
  - Adjustments for reliability growth

# Review

You learned a few methods for allocating a system reliability objective down to the software and hardware LRUs:

Bottom up – Conduct predictions for every LRU. The allocation for every LRU (hardware or software) is directly proportional to the predicted for the LRU divided by the sum of all of the predictions for every LRU. If the methods for prediction are modern, use historical data and are conducted properly this can be the most accurate relatively speaking.

Top Down –

#1 – Top down objective allocated to each subsystem which is then allocated to software and hardware LRUs using any method. For multiple organizations this is easiest to manage.

#2 – Top down objective allocated to HW and SW. Each LRU allocation is then allocated from either the HW or SW allocation. This is least preferred recent data is used to establish the allocation to all software.



# Using Excel

See worksheets that come with this class



# Options for trending data in Microsoft Excel

---

See the integrating software and hardware  
reliability spreadsheet



# Conclusions



# In this class you learned



The value of integrating software and hardware reliability predictions



How to merge the software and hardware predictions



How to evaluate the results and identify alternatives if the feasible goal isn't the target goal



How to establish a feasible system reliability objective



The tools that you can use



# References

---

- [1] Neufelder, A, "The Cold Hard Truth About Software Reliability", Edition 6.e, SoftRel, LLC, 2014.
- [2] System Software Reliability Assurance Guidebook, Sections 4 and 5, P. Lakey, A. Neufelder, 1995, produced for Rome Laboratories.
- [3] Rome Laboratory RL-TR-92-15 Reliability Techniques for Combined Hardware/Software Systems, 1992, Rome NY.



# Annex

Reliability block diagramming formulas

## Computing reliability [3]

- The reliability of a group of redundant components is:

$$\mathbf{R(t)} = \sum_{\mathbf{i=1}}^{\mathbf{k}} \mathbf{e^{-\lambda_i t}} \prod_{\substack{\mathbf{j=1} \\ \mathbf{j \neq i}}} (\lambda_j / (\lambda_j - \lambda_i))$$

- If all components have the same failure rates then:

$$\mathbf{R(t)} = \mathbf{e^{-\lambda t}} \sum_{\mathbf{i=0}}^{\mathbf{k-1}} (\lambda t^i / i!)$$

# Computing failure rate[3]

- When the individual component failure rates are different, the failure rate of the group is a constant based on time to first failure:

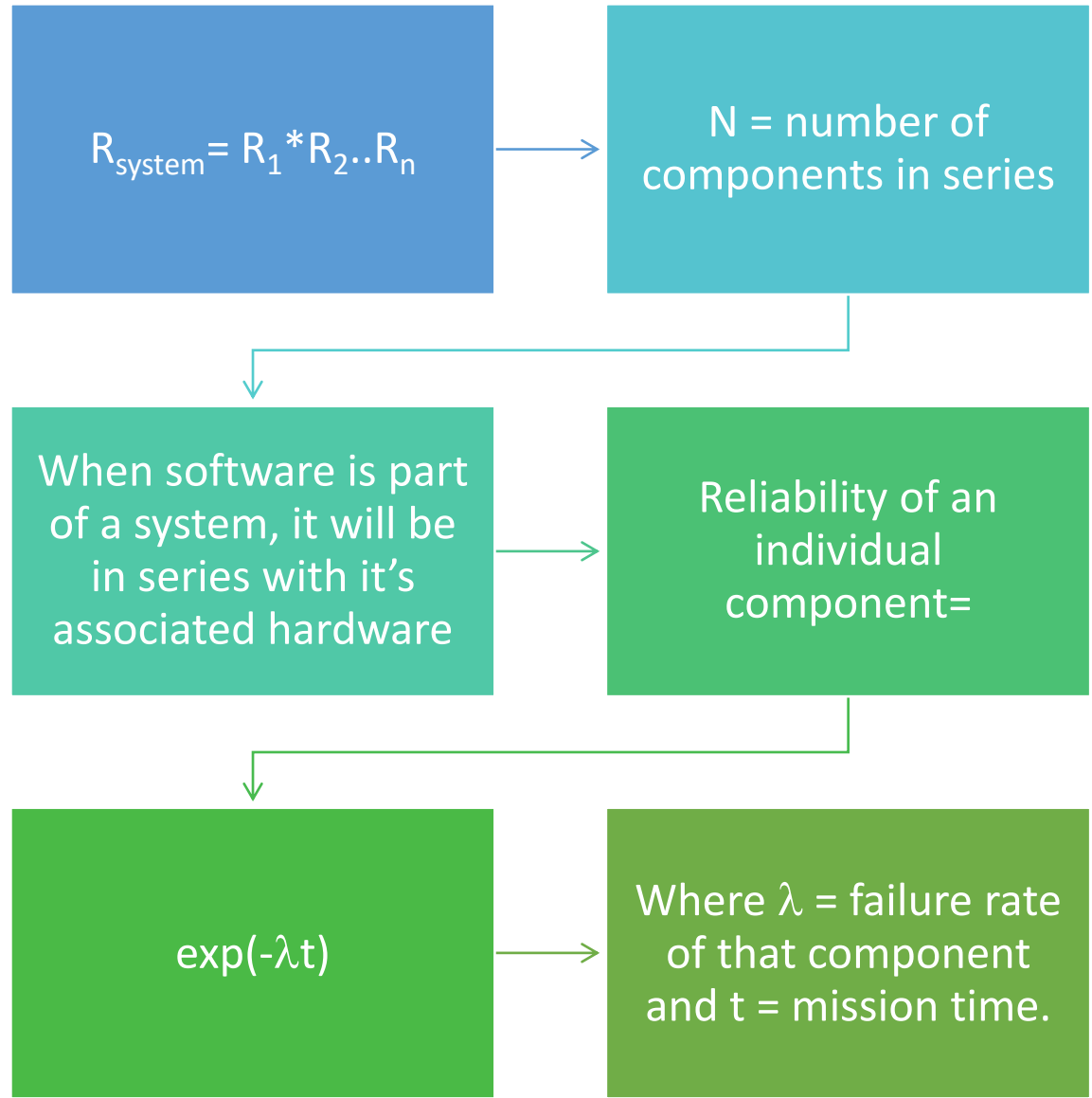
$$\lambda_{\text{sys}} = \mathbf{1} / \left( \sum_{\mathbf{i}=1}^{\mathbf{k}} (\mathbf{1} / \lambda_{\mathbf{i}}) \right)$$

- When the individual component failure rates are different, the instantaneous failure rate of the group is:

$$\lambda_{\text{sys}}(\mathbf{t}) = \lambda / \left( (\mathbf{k}-1)! \sum_{\mathbf{i}=0}^{\mathbf{k}-1} (\lambda \mathbf{t})^{\mathbf{i}+1-\mathbf{k}} / \mathbf{i}! \right)$$

- The effective failure rate is a constant equal to  $\lambda/k$ .
- Weight all terms except for  $i=1$  with probability of the switch from one component to another failing.
- Also consider that there are other errors which can result from the acceptance test:
  - 1 - Acceptance test rejects result when it was acceptable
  - 2 - Acceptance test accepts and unacceptable result

# Blocks in series



# Reliability Blocks in Parallel

Two components in parallel

$$R_{\text{system}} = R_1R_2 + R_1Q_2 + Q_1R_2 + Q_1Q_2$$

Where Q = probability of component failure

The sum of these must add to 1

If only one component success is required for system success then

$$R_{\text{system}} = 1 - Q_1Q_2 = R_1R_2 + R_1Q_2 + Q_1R_2$$

## Function of

Reliability of  
primary  
component

Reliability of  
secondary  
component

Reliability of  
switching  
mechanism



Standby  
redundancy

Primary and secondary  
components are in parallel  
with each other and in series  
with switching mechanism



## Standby Redundancy

If switching is perfect the reliability of primary and backup are the same then

$$R_{\text{system}} = \sum_{i=0}^{n-1} (\lambda t)^i / i! e(-\lambda t)$$

Where  $i$  = number of redundant components

# M out n redundancy

$$\begin{aligned} \blacksquare R_{\text{system}} &= \\ & \sum_{k=m}^N \binom{N}{k} R^k Q^{N-k} \end{aligned}$$