Calculating Producer & Consumer Risk in Systems of Systems

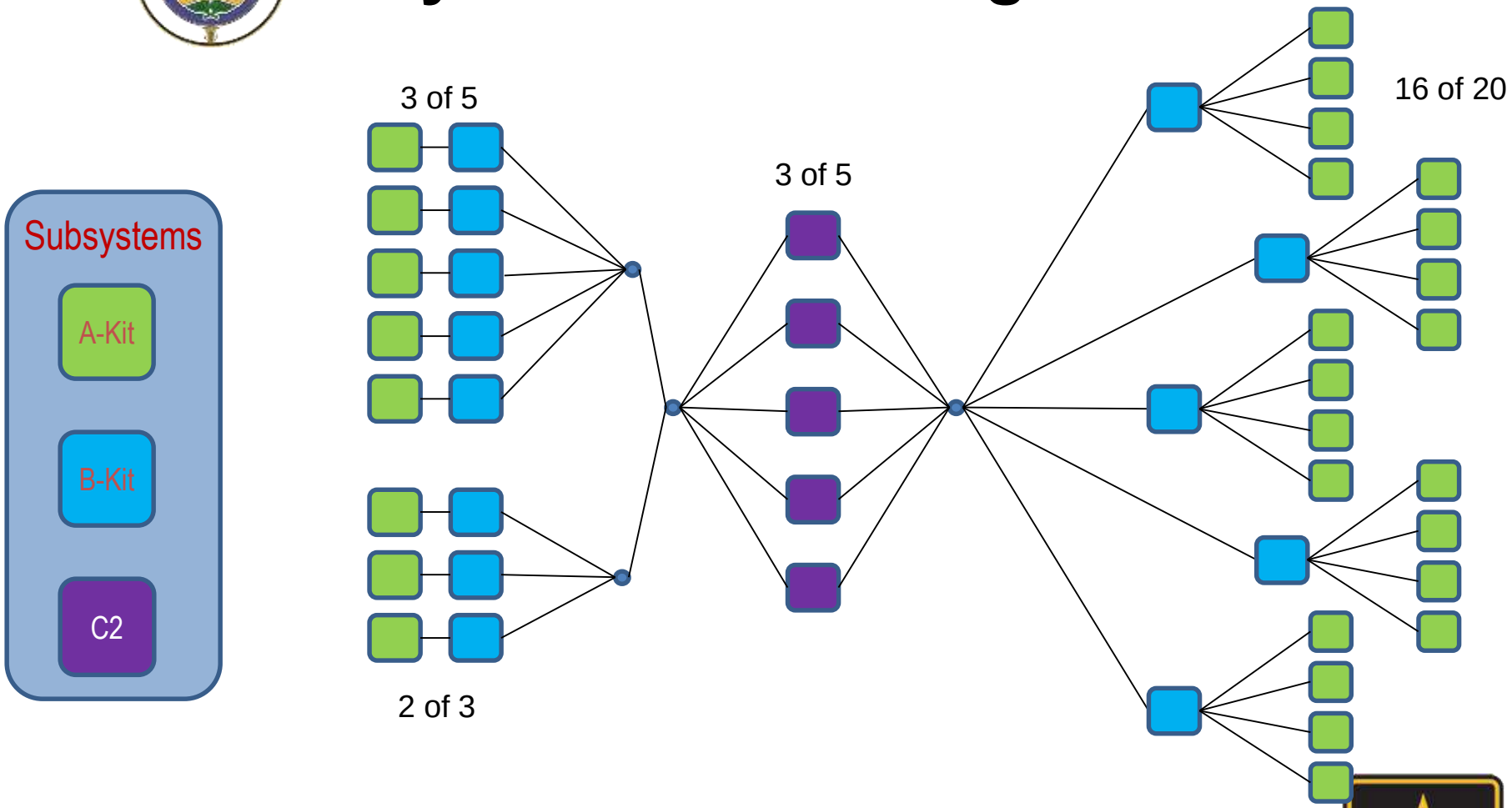
Dr. Daniel Owens
20 August 2015
Army Evaluation Center

U.S. Army Test and Evaluation Command

Problem

- How can we determine the size and duration of a test event to demonstrate the reliability of a System of Systems?
  - How do we calculate System of Systems reliability?
  - How do we calculate consumer risk?
  - How do we calculate producer risk?

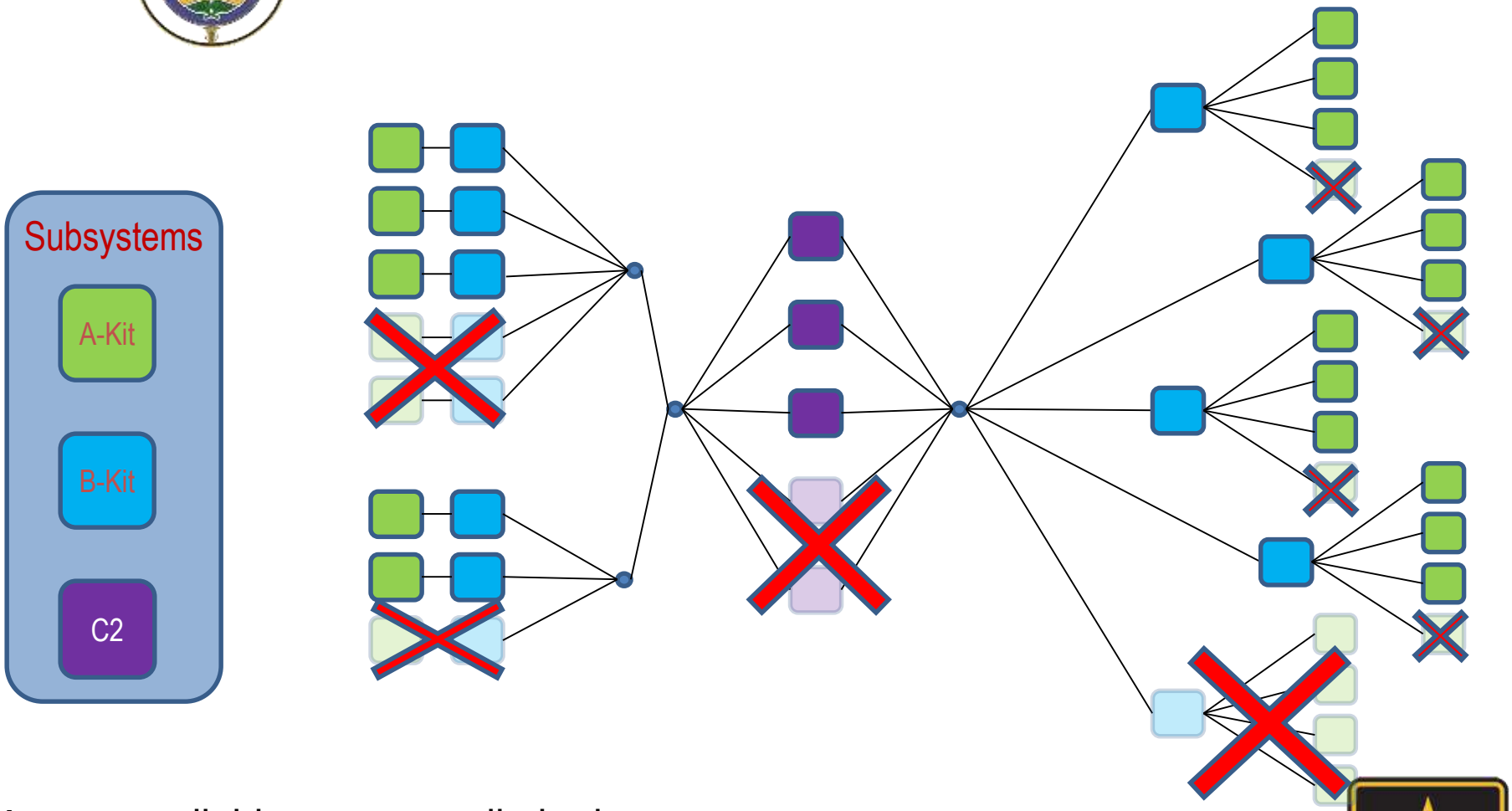
System Block Diagram



Assumptions

- System of Systems requirement is stated in terms of a probability of mission success
  - For example, 90% chance of going 72 hours with no SoS-level failures
- Network is static
- Subsystems are geographically isolated
  - Failures can be assumed to be independent
- Subsystems have exponential failure rates
- Not every subsystem has to be up all the time
  - Built-in redundancy allows for some failures
  - Subsystems can be repaired

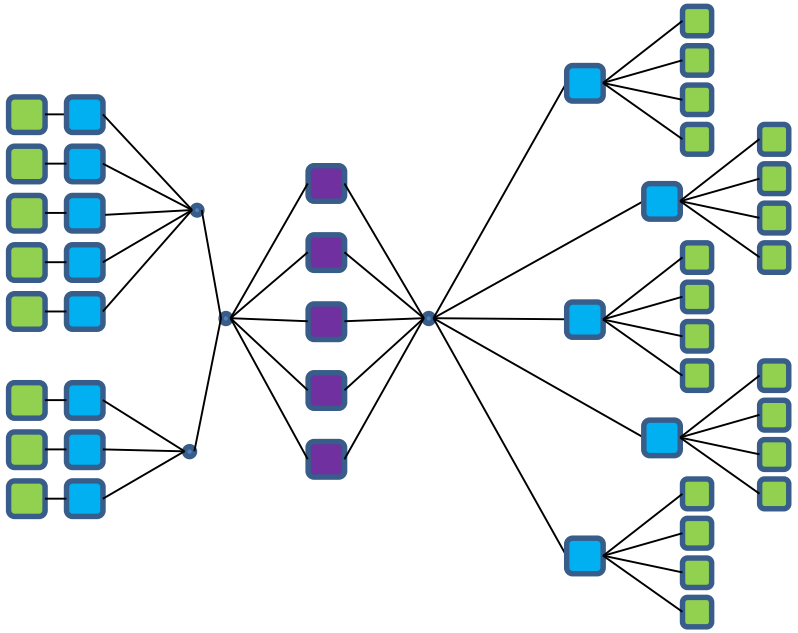
Available Test Assets



Assets available to test are limited to a fraction of the full System of Systems

Strategy

Evaluate reliability of each subsystem, then use reliability block diagrams (RBDs) to calculate the system of systems reliability



Calculating System of Systems Reliability

Can we calculate system of systems reliability if:

(1) We assume subsystems are never repaired and we know the exact subsystem reliability?

(2) Subsystems can be repaired?

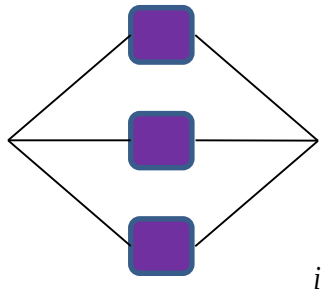
(3) Subsystems are repaired, and we must estimate subsystem reliability from test data?

System of Systems Reliability

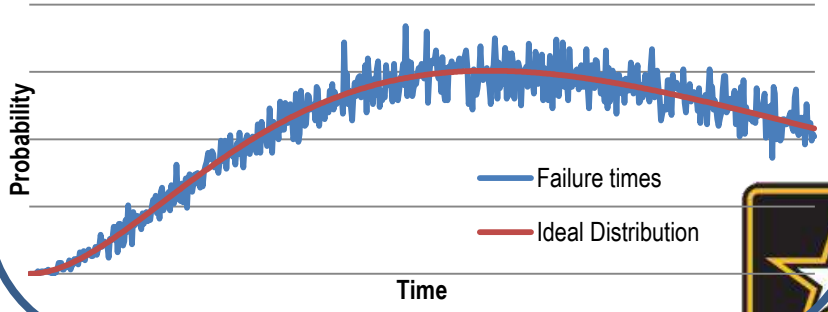
Assuming no repair, and exact subsystem reliability

- System of systems reliability can be calculated analytically
- Simulation algorithm (***no repair***):
  - Begin with all subsystems up at time t=0
  - Randomly generate failure time for each subsystem
  - SoS failure time is when too many subsystems have failed
  - Repeat

k-of-n

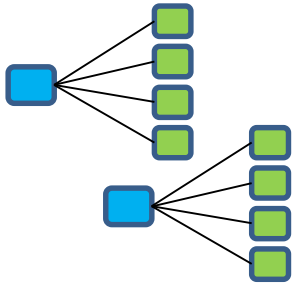


$$R(t) = \sum_{i=0}^{n-k} \frac{n!}{(i)!(n-i)!} \left(e^{-(n-i)\lambda t}\right)\left(e^{1-\lambda t}\right)^{i}$$



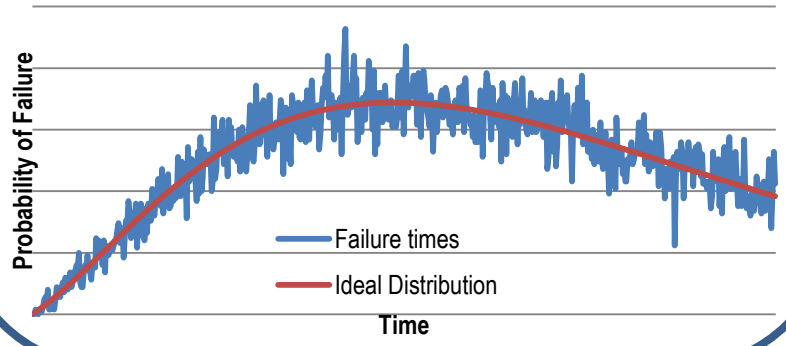
System of Systems Reliability

k-of-n trees

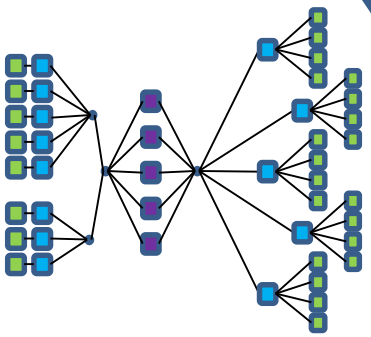


$$R(t) = e^{-n_2\lambda_2 t}\sum_{i=0}^{n_1-k_1}\frac{n_1!}{(i)!(n_1-i)!}\left(e^{-(n_1-i)\lambda t}\right)\left(e^{1-\lambda t}\right)^i$$

$$+\,n_2 e^{-(n_2-1)\lambda_2 t}\left(1-e^{-\lambda_2 t}\right)^{n_1-n_1/n_2-k_1}\sum_{i=0}\frac{(n_1-n_1/n_2)!}{(i)!(n_1-n_1/n_2-i)!}\left(e^{-(n_1-n_1/n_2-i)\lambda t}\right)\left(e^{1-\lambda t}\right)^i$$



Full system

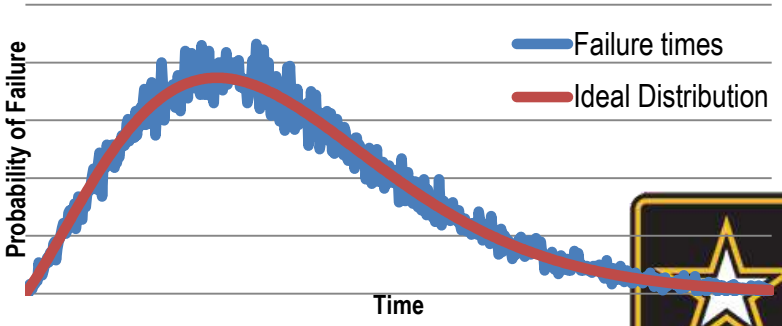


$$R(t) = \left[\sum_{i=0}^{n_1-k_1}\frac{n_1!}{(i)!(n_1-i)!}\left(e^{-(n_1-i)\lambda_1 t}\right)\left(e^{1-\lambda_1 t}\right)^i\right]$$

$$\times\left[\sum_{i=0}^{n_2-k_2}\frac{n_2!}{(i)!(n_2-i)!}\left(e^{-(n_2-i)(\lambda_2+\lambda_3)t}\right)\left(e^{1-(\lambda_2+\lambda_3)t}\right)^i\right]$$

$$\times\left[\sum_{i=0}^{n_3-k_3}\frac{n_3!}{(i)!(n_3-i)!}\left(e^{-(n_3-i)(\lambda_2+\lambda_3)t}\right)\left(e^{1-(\lambda_2+\lambda_3)t}\right)^i\right]$$

$$\times\left[\begin{array}{l} e^{-n_5\lambda_3 t}\sum_{i=0}^{n_4-k_4}\frac{n_4!}{(i)!(n_4-i)!}\left(e^{-(n_4-i)\lambda_2 t}\right)\left(e^{1-\lambda_2 t}\right)^i \\ +\,n_5 e^{-(n_5-1)\lambda_3 t}\left(1-e^{-\lambda_3 t}\right)^{n_4-n_4/n_5-k_1}\sum_{i=0}\frac{(n_4-n_4/n_5)!}{(i)!(n_4-n_4/n_5-i)!}\left(e^{-(n_4-n_4/n_5-i)\lambda_2 t}\right)\left(e^{1-\lambda_2 t}\right)^i \end{array}\right]$$

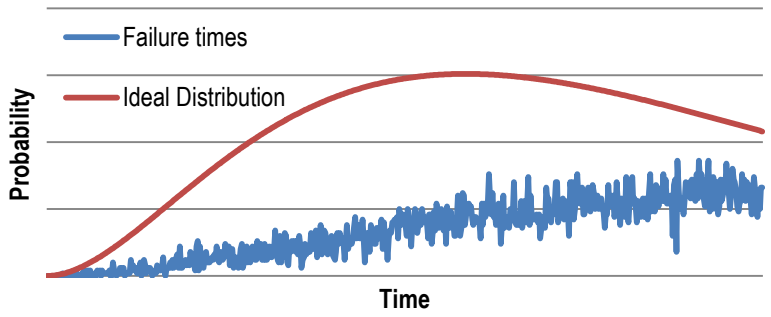


System of Systems Reliability

Assuming repair

- Analytical solution no longer holds
- New algorithm (***repair***):
  - Begin with all subsystems up at time t=0
  - Randomly generate failure time for each subsystem
  - When a subsystem fails, randomly generate repair time
  - SoS failure time is when too many subsystems are down simultaneously
  - Repeat

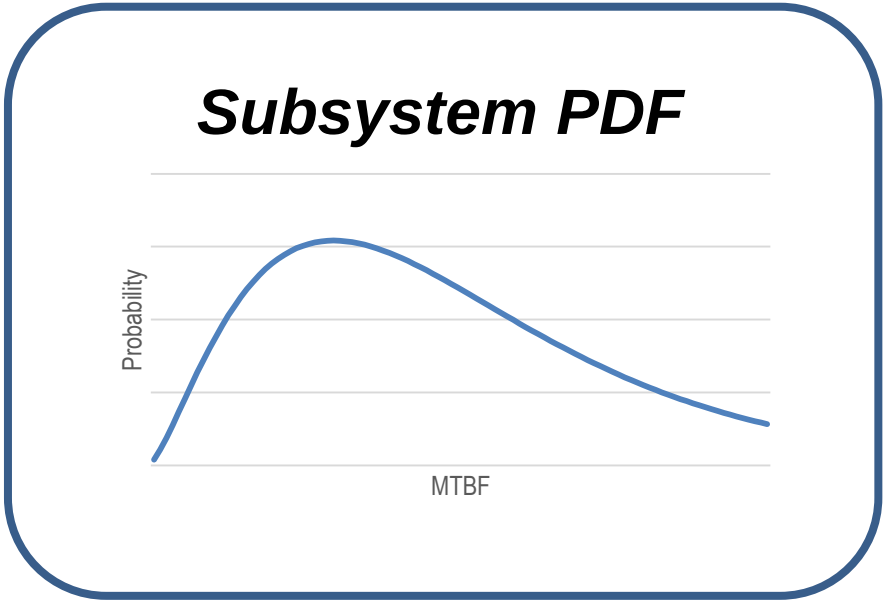
3-of-5 good, with repair



System of Systems Reliability

Assuming repair, and subsystem failure rate estimated from test

•Subsystem reliability probability density function can be described using a chi-squared distribution



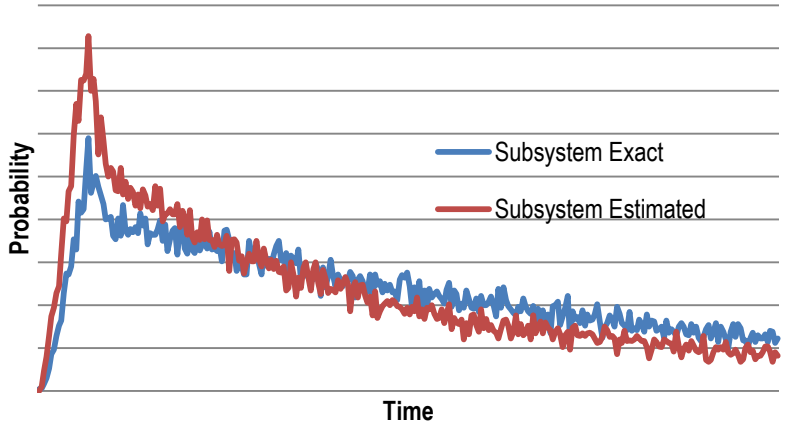
System of Systems Reliability

Assuming repair, and subsystem failure rate estimated from test

- New algorithm (***subsystem estimate***):

- Randomly sample subsystem reliability times from subsystem PDFs
  - Perform ***repair*** algorithm for this scenario
  - Repeat
- Aggregate results from all scenarios

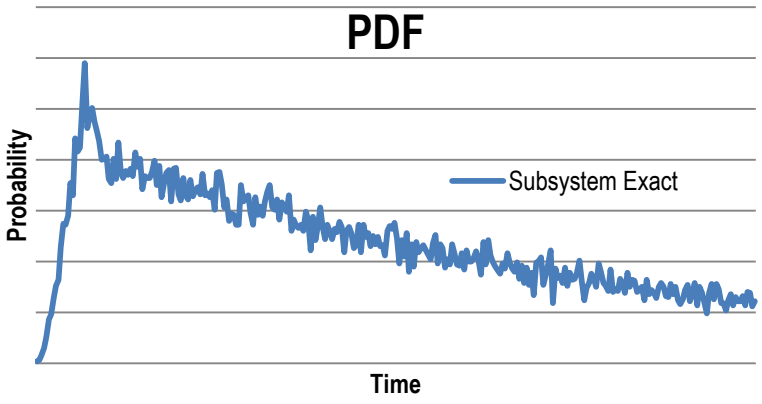
3-of-5 good, exact vs. estimated subsystem reliability

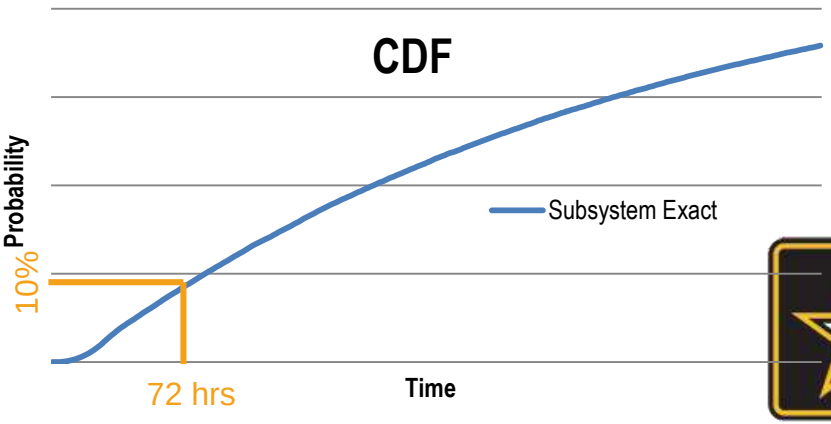


Consumer Risk

- Reliability typically must be demonstrated with 80% statistical confidence at IOT
- IOT requirement would be:
  - Demonstrate with 80% confidence that the system of systems has a 90% probability of going 72 hours without a system abort

3-of-5 good, exact subsystem reliability



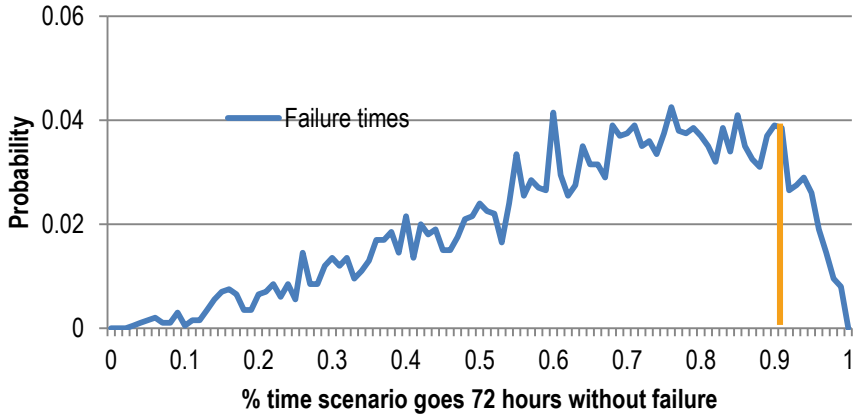


Consumer Risk

Assuming repair, and subsystem failure rate estimated from test

- New algorithm (***consumer risk***):
- Randomly sample subsystem reliability times from subsystem PDFs
  - Perform ***repair*** algorithm for this scenario
  - Was this scenario successful? (Did more than 90% of runs go 72 hours without repair?)
  - Repeat
- Were more than 80% of scenarios successful?

3-of-5 good, exact vs. estimated subsystem reliability



Test Planning

- Two questions to ask in test planning:
  - Given a test of a certain length, how many failures can we allow? **–or–** Given a certain number of allowed failures, how long must the test be? (Corresponds to consumer risk)
  - Given a test of a certain length, what is the probability that, even if my system is good enough, it fails the test anyway? (Corresponds to producer risk)

With non-redundant systems, these can be calculated analytically

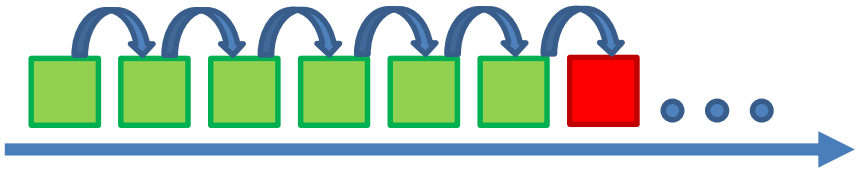
With redundant systems, we must again resort to stochastic processes

Test Planning – Consumer Risk

- Consumer Risk – Given a test of a certain length, how many failures can we allow? **–or–** Given a certain number of allowed failures, how long must the test be?

- Our ***consumer risk*** algorithm lets us answer the question: given a test of a certain length, and an observed number of failures, does the system pass?

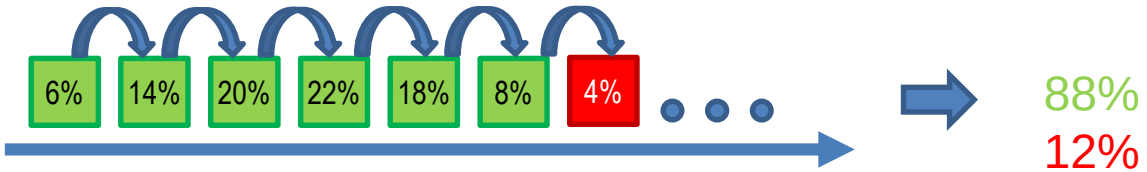
- In order to use this algorithm for test planning, we must iterate over the number of subsystem failures until we find the cutoff where the system no longer passes



Number of Failures

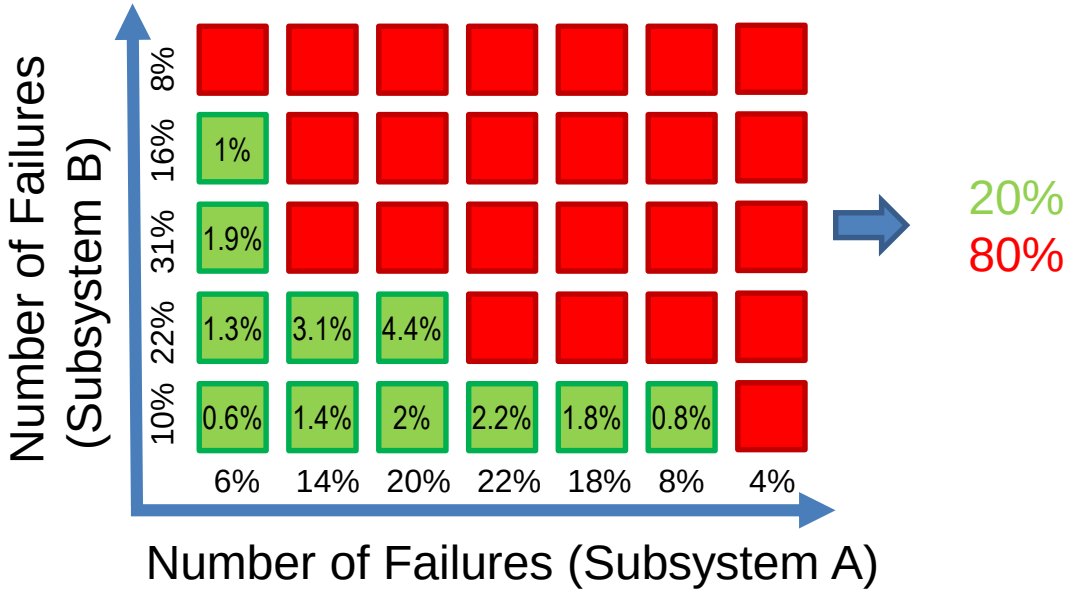
Test Planning – Producer Risk

- Producer Risk – Given a test of a certain length, what is the probability that, even if my system is good enough, it fails the test anyway?
- To calculate this we need to put a number on how reliable the system truly is. Typically we would use the reliability design goal for this value
- Based on the true reliability, we can analytically calculate how likely each of the test outcomes are



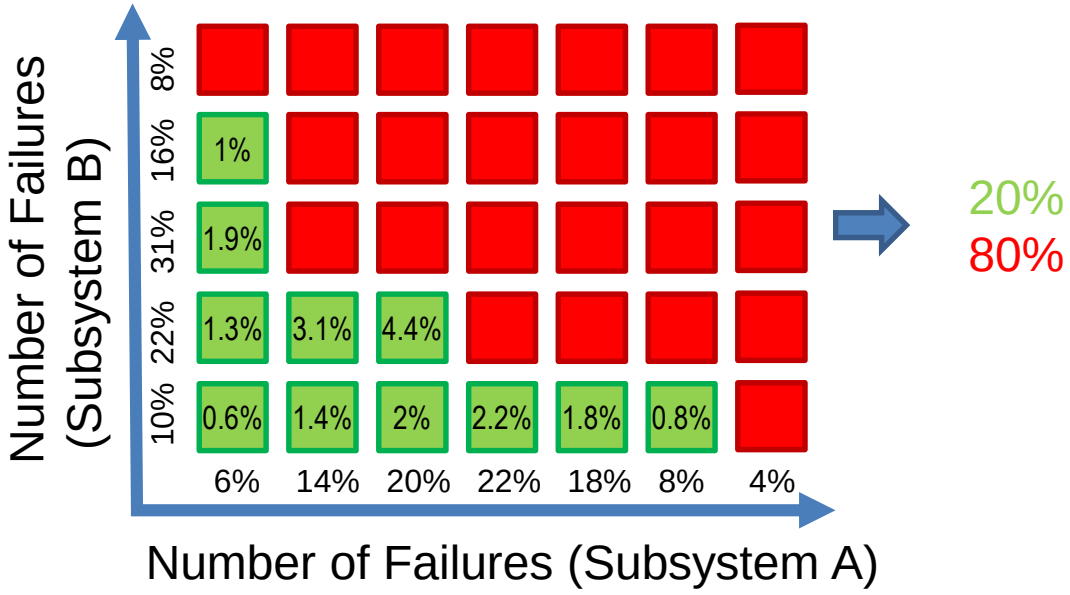
Producer Risk – multiple subsystems

- The previous example has only one type of subsystem. What if there are more?
- It is necessary to expand the algorithm to iterate over the number of failures for each subsystem
- The likelihood of each outcome is simply the product of the likelihood of the number of failures for each subsystem



Producer Risk – multiple subsystems

- This layout illustrates a couple of points worth noting:
  - There is trade space among the subsystem reliabilities—if one does poorly but another does very well, they can to some degree cancel each other out
  - The overall likelihood of passing the test is *NOT* just the product of the likelihoods from the subsystems



Test planning

- Big question: How many test hours do we need on each subsystem in order to meet goals for both consumer risk and producer risk?

- This is a trial and error process using the above algorithms. Pick a number of hours, see what the risk is, adjust the hours, re-evaluate, etc…
  - In the process, it may become apparent that particular subsystems have greater impact on risk than others. This opens the possibility to optimize test resources to focus on those that give the greatest return on investment

Questions?