Software Reliability and Security

Module 5

Winter 2017

Presentation/Lecture Schedule and Report Due Dates

- Presentation 1
  - Related background paper
  - Jan 27, Feb 1, 3
- Presentation 2
  - Project proposal
  - March 1, 3, 8
- Presentation 3
  - Final project report
  - March 24, 29, 31

- Lectures
  Jan 13, 18, 20, 25, 27
  Feb 1, 3, 8, 10, 15, 17
  March 1, 3, 8, 10, 15, 17, 22, 24, 29, 31

- Project Proposal Due
  Tuesday, February 28

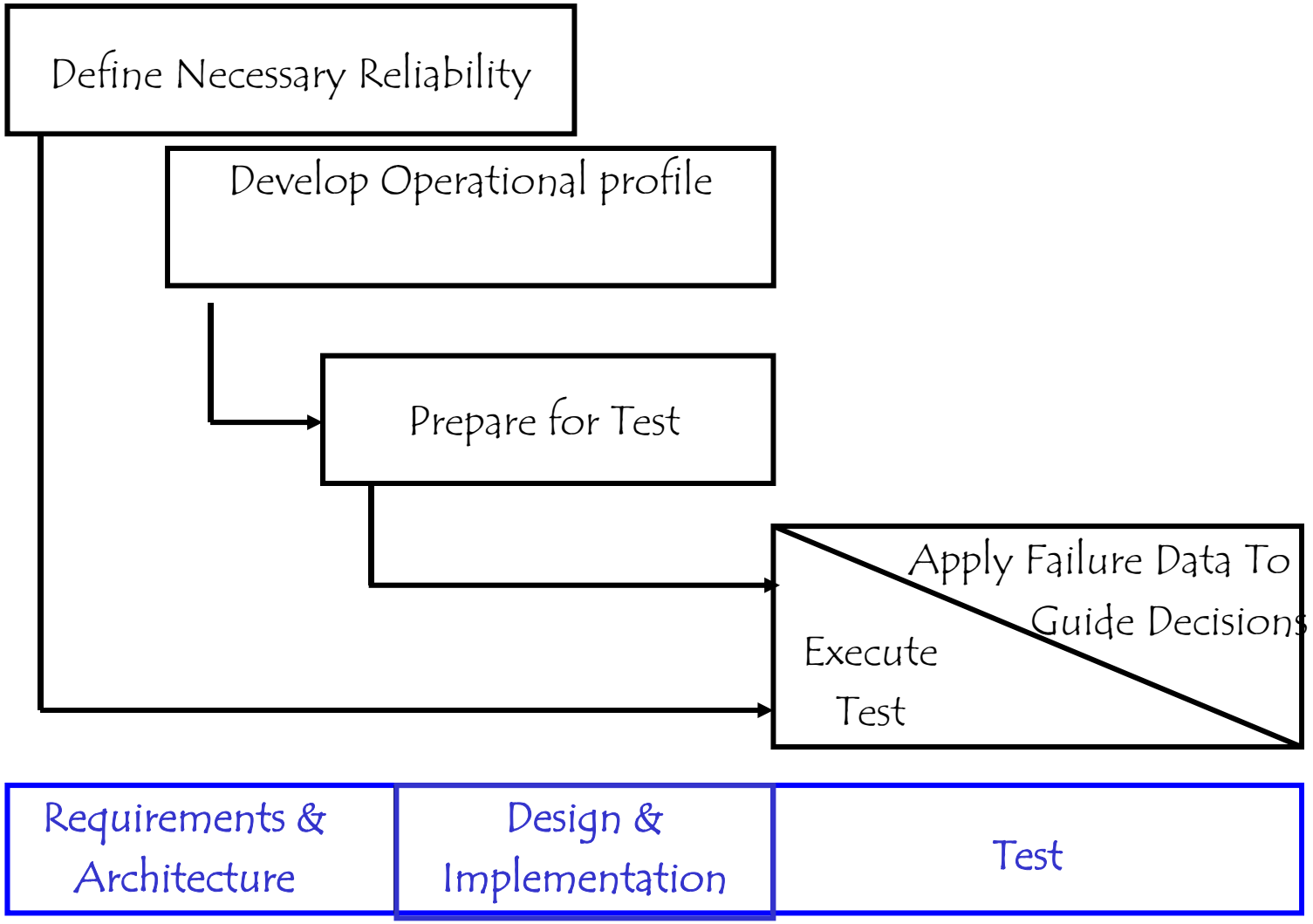
- Final Project Report Due
  Monday, April 10

- Final Exam
  Wednesday, April 12, 10:00am

Outline

- Software Reliability vs. Hardware Reliability
- Software Reliability Terminology
- Software Reliability Engineering Process
- Software Reliability Modeling

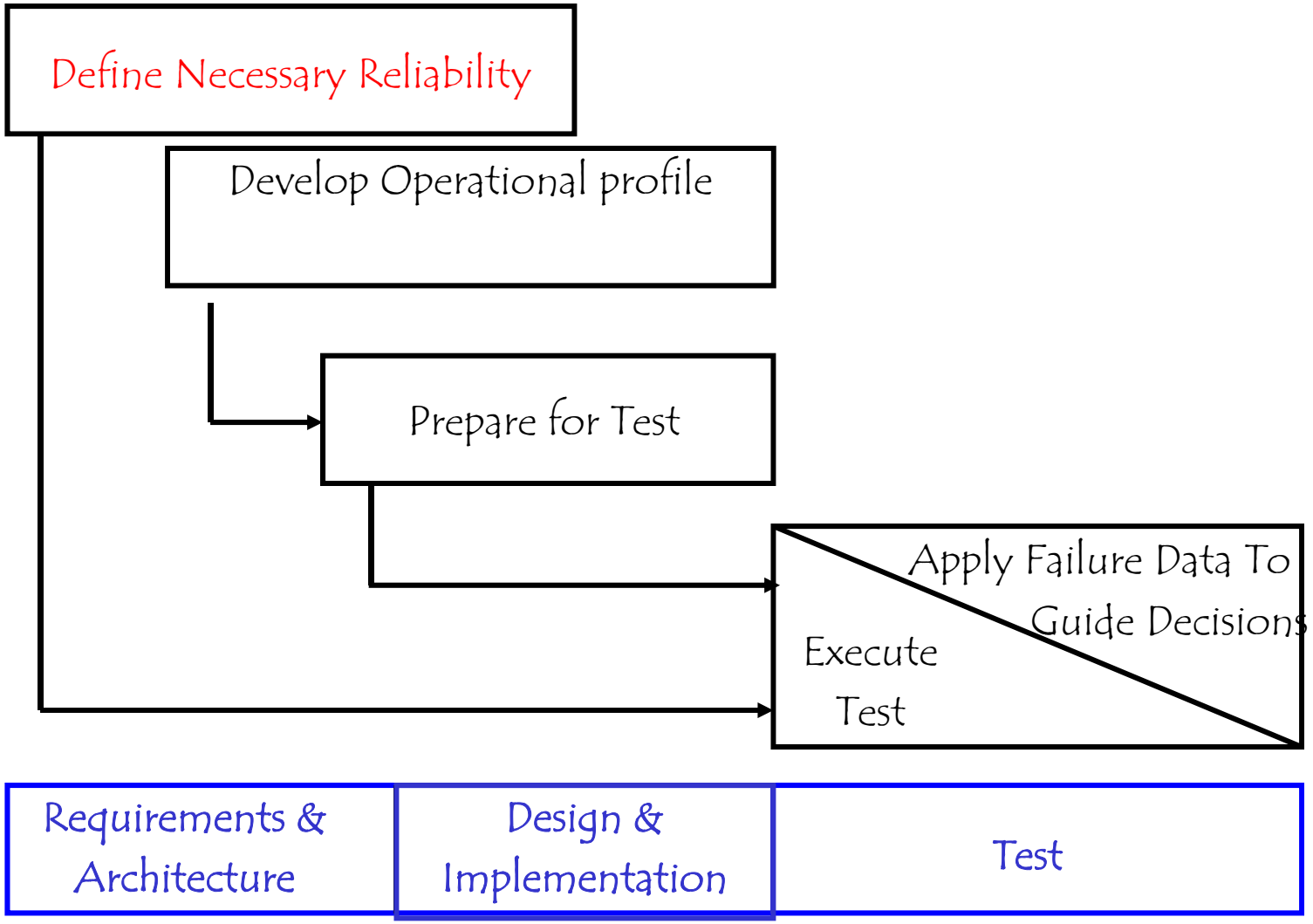
Software Reliability Engineering Process



Software Reliability Engineering Process (SREP)

- Feedbacks to earlier process (similar to spiral, no waterfall)
- "Execute Test" and "Apply Failure Data to Guide Decisions" steps are followed in parallel
- Two steps not shown
  - System Engineering
    - Identify the systems that require separate test
  - Post delivery and maintenance phase
    - Determine the achieved reliability and actual operational profile

Software Reliability Engineering Process



Define Necessary Reliability

- For each system
  - 1a. Define failure with severity classes for the product
  - 1b. Choose a common measure for all associated systems
  - 1c. Set a failure intensity objective for each system to be tested
- For any software of the product
  - 2a. Determine the software failure intensity objective
  - 2b. Engineer the reliability strategies to meet the failure intensity objective

Define Necessary Reliability – contd.

- 1a. Define the failure with severity classes
  - Severity class - a set of failures that have the same per-failure impact
  - Some examples of severity classes

Severity Class	Definition (Based on Cost)	Definition (Based on System Capability Impact)
1	>100,000	Unavailability to users of one or more key operations
2	10,000-100,000	Unavailability to users of one or more important operations
3	1000-10,000	Unavailability to users of one or more operations but workarounds available
4	<1000	Minor deficiencies in one or more key operations

Define Necessary Reliability – contd.

- 1b. Choose a common measure for all associated systems
  - Choose the natural or time unit you will use
  - A natural unit is related to the output of a software-based product
    - Ex. One failure per 1kh of Internet service (failure intensity)
  - The time may be execution time or wall clock time

Define Necessary Reliability – contd.

- 1c. Set the failure intensity objective for each system
  - Factors – development time and cost, technological evolution
    - Example: lower intensity, greater cost

Failure impact	Typical failure intensity objective (failure/hour)	Time between failure
Hundreds of deaths, more than 10^9 cost	10^{-9}	114,000 years
One or two deaths, around 10^6 cost	10^{-6}	114 years
Around $1000 cost	10^{-3}	6 weeks

Define Necessary Reliability – contd.

- Failure intensity and reliability/availability
  - Failure Intensity (Ý) and Reliability (R): Ý= -ln R/t and R= exp(-Ýt)
    - t = number of natural or time units
  - Failure Intensity (Ý) and Availability (A): $A = t_u/(t_u+t_d)$ & $t_d = t_m$ Ýt_u,
    - t_u is the uptime, t_d is the downtime, and t_m is the downtime per failure
  - An Example: "The overall availability requirements are of the order of 99.999 percent or higher which would imply that the network cannot be down for more than 6 min/yr on average" [Ramaswami 2000, IEEE]

Define Necessary Reliability – contd.

- More Examples

Reliability for 1-h mission time	Failure Intensity
0.368	1 failure/h
0.9	105 failures/kh
0.99	10 failures/kh
0.999	1 failure/kh
0.99989	1 failure/year

Define Necessary Reliability – contd.

- 2a. Determine software failure intensity objective
  - Find the expected failure intensity for each system (both HW & SW)
  - The estimate should consider operational data, vendor warranty, expert experience of experts, etc. and their priorities
  - To get the system failure intensity - add all of the expected component failure intensities
    - component failure: a behavior that would cause system failure

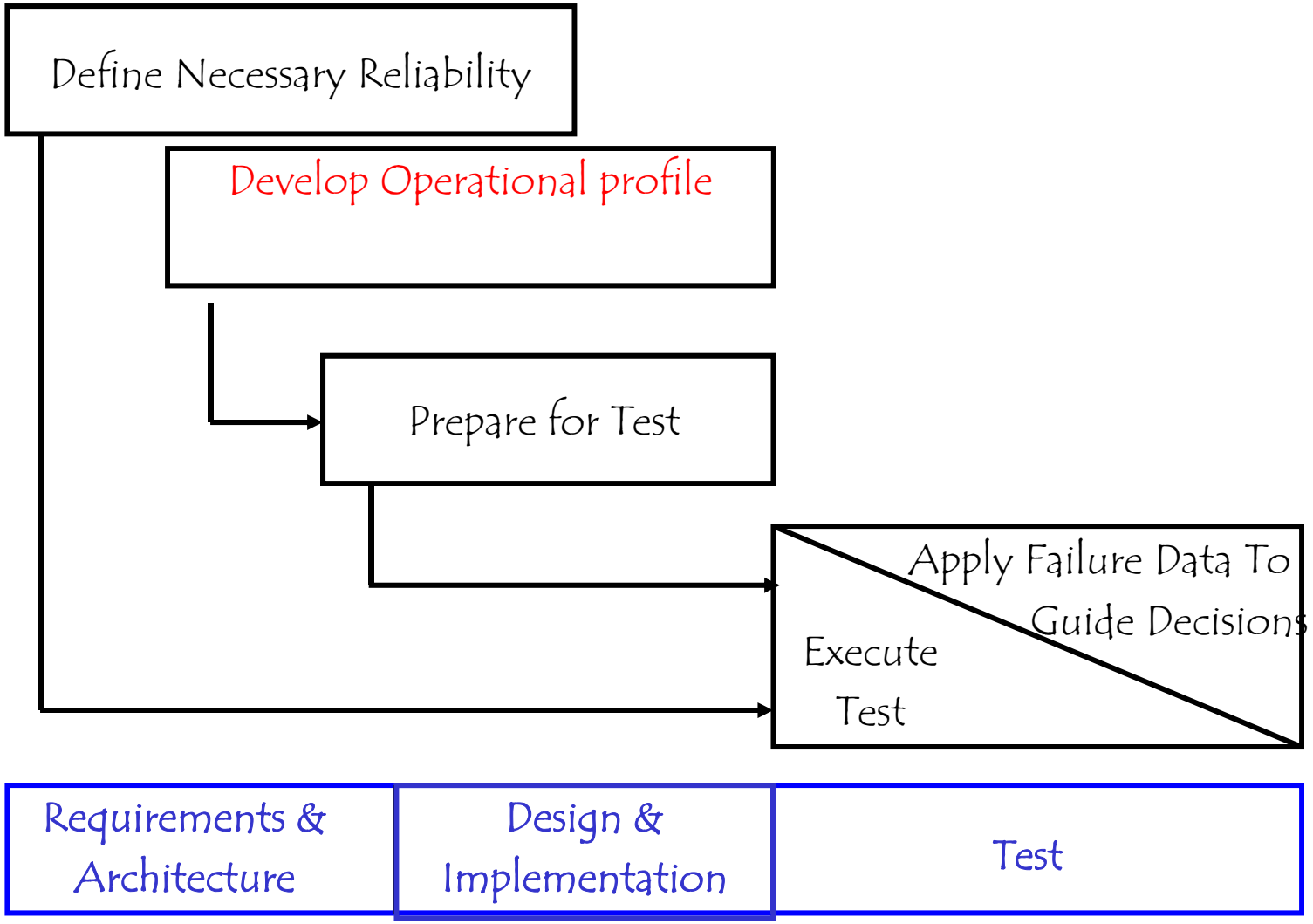
Define Necessary Reliability – contd.

- 2b. Engineer the reliability strategies to meet the failure intensity objective
  - Principal strategies: fault prevention, fault removal, and fault tolerance

Some General Guidelines

Reliability Level	Failure Intensity Range (Failures/1000 Execution Hours)	Strategy /Guidelines
Ultra	<0.1	Fault tolerance, extensive requirements and design reviews
High	0.1-10	Fairly extensive requirements and design reviews, some fault tolerance
Commercial	10-2000	Guide any requirement or design reviews with operational profile and criticality
Prototype	>2000	Testing usually more practical than fault tolerance or requirements or design reviews

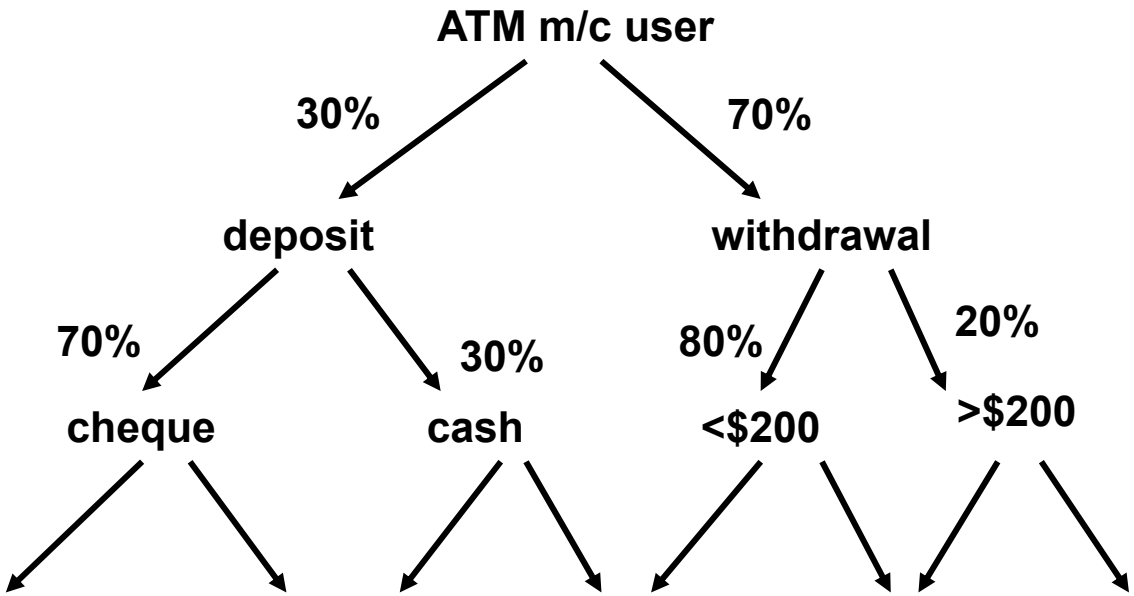
Software Reliability Engineering Process



Operational Profile

- Most software systems cannot be tested exhaustively
- One option: Test more used parts of a system more thoroughly (manage test distribution)
  - Use operational profile
- May have different operational profile for each component
- An operation
  - A logical task of short duration
  - Returns control to the system when complete
  - An operation's steps should be substantially different from other operations

An Operational Profile Example



- End of a month
- Beginning of a month
- Location of the machine

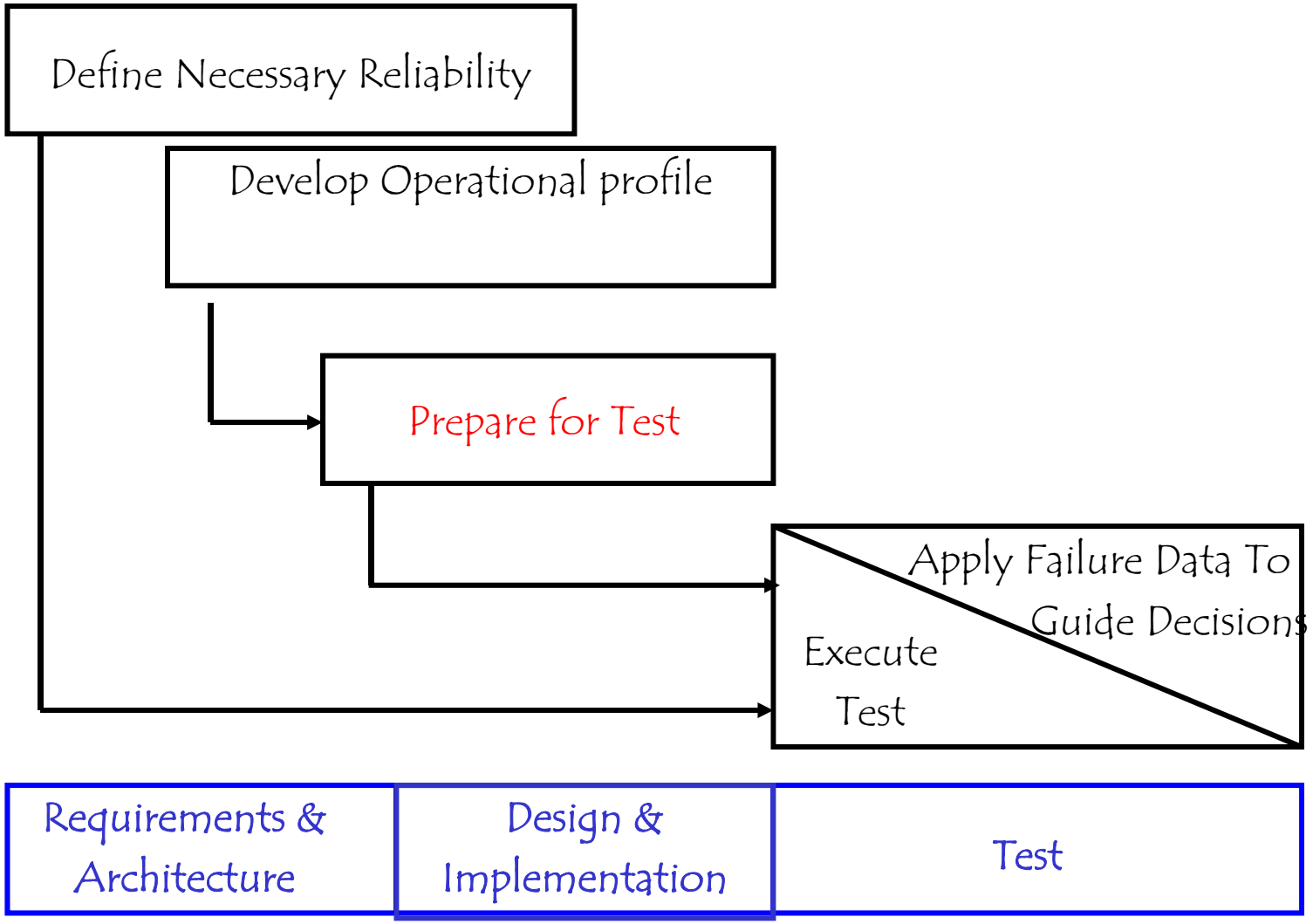
Develop Operational Profile – Steps

- Determine the operational modes of each system
  - An operational mode is a distinct pattern of system use, e.g., the operational mode of an ISP: peak hour, off hour
  - We need an operational profile for each operational mode
- Identify operation initiators
  - User types, external, internal, etc.
- Tabular or graphical representation
  - Should be the same for all operational modes

Develop Operational Profile - Steps

- Create an operation list for each operation initiator
  - Based on requirements specifications, system engineers, user manuals, prototypes, previous system, potential users feedback
- Determine occurrence rates and probabilities (individual operation occurrence rates / total occurrence rates) of the individual operations
  - The same operation may occur in different operational mode but with different occurrence of probabilities

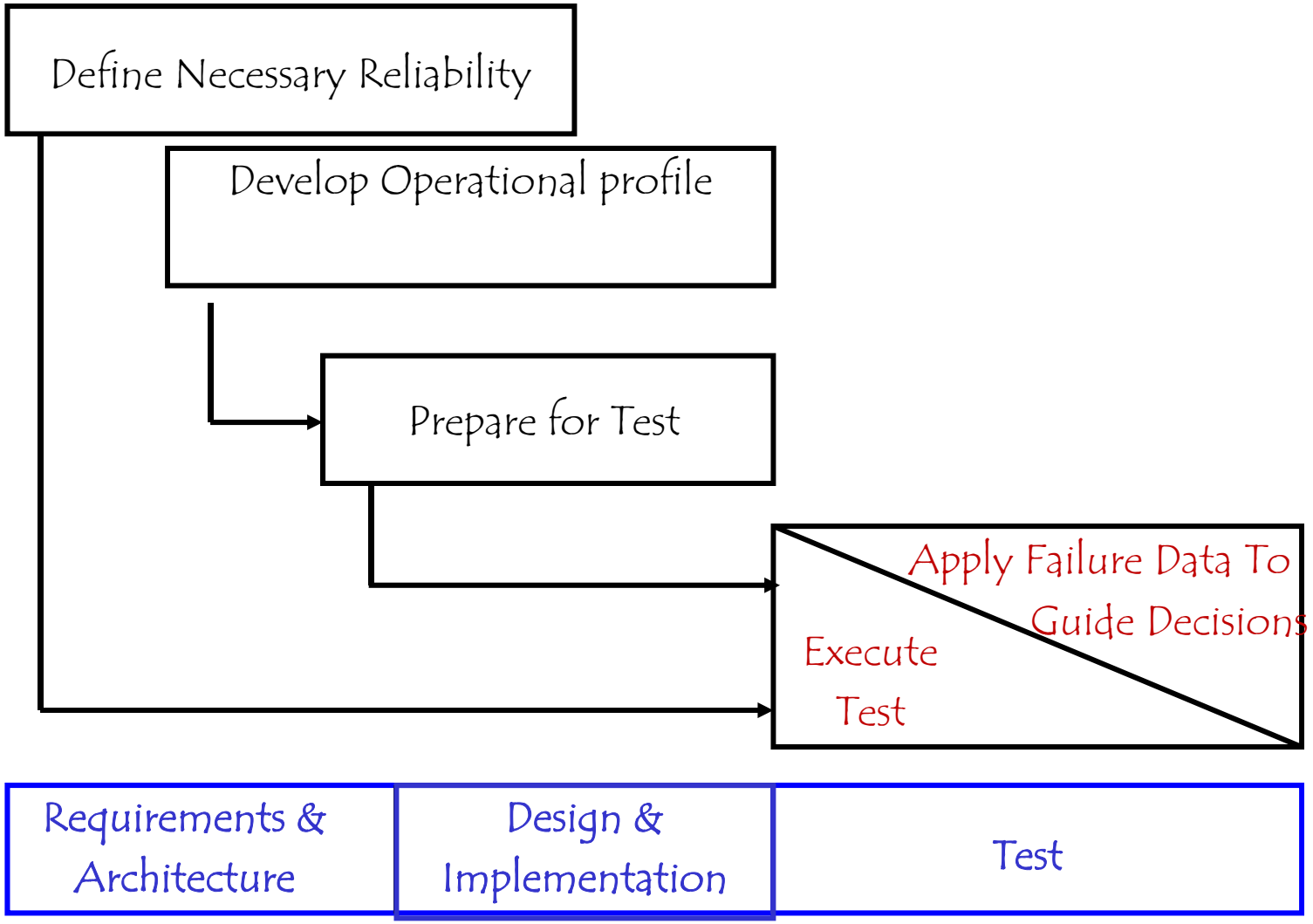
Software Reliability Engineering Process



Prepare for Test

- Prepare Test Cases
  - Estimate the number of new test cases for the current version
  - Allocate new test cases among the systems to be tested
  - Specify new test cases
    - Compute distribution of test cases
    - Translate test cases into system under test inputs
  - Add the new test cases to the test cases from previous releases
- Prepare Test Procedures
  - Need one test procedure for each operational mode

Software Reliability Engineering Process



Execute Test

- Allocate test time
- Invoke tests
- Identify system failures
  - Analyze the test output for deviations
  - Determine which deviations are failures
  - Establish when the failures occurred
  - Assign failure severity classes for failure priority resolution

Failure Data to Guide Decisions

- Accept or reject acquired components
  - Decide using certification test
- Guide software development process
  - Guide process changes
  - Prioritize failures for process resolutions
- Accept or reject a super system
  - Decide using certification test
- Release a product
  - Based on reliability growth modeling (discussed next)

Software Reliability Modeling

- Predict future failures (times and rates) based on historically observed failure data and a mathematical model
- Observed failure data ----> **Software Reliability Model** ----> Predicted failure data
- Models are independent of implementation method, architecture, or language – black box
- A mathematical model is selected based on how well it matches the observed failure data
  - Provides close estimate of future failure data
  - Measure desired aspects
  - Based on reasonable assumptions
  - Simple to apply

Software Reliability Modeling

- Two main purposes w.r.t. testing efforts
  - To predict software testing time to achieve a specific level of reliability
  - To predict the expected operational reliability after testing

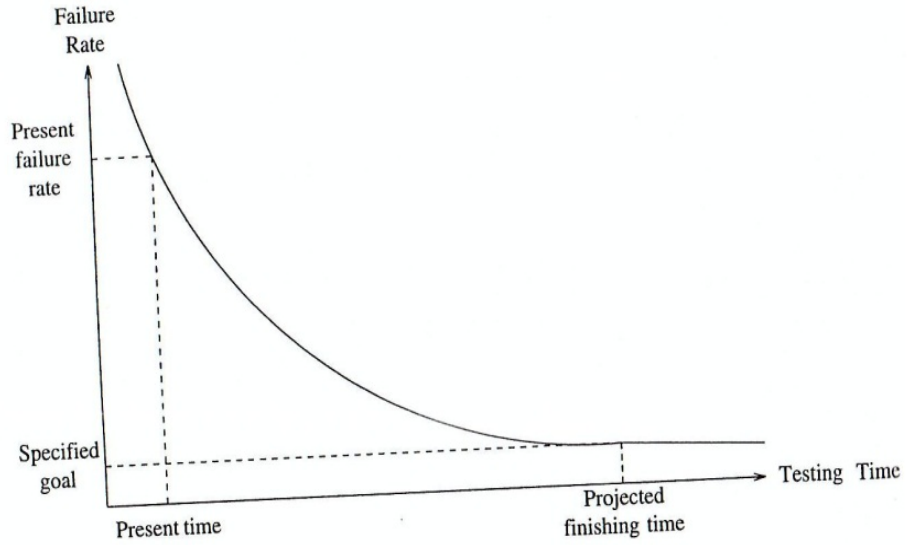


Figure 1.4 Basic ideas on software reliability modeling.

Selecting a Software Reliability Model

- Collect the observed failure data
  - Failure-triggered count
  - Time-triggered count
- Select an SRE model
- Estimate the SRE model parameters
  - Using historically observed failure data and statistical approaches (e.g., least square, maximum likelihood)
- Select a model that matches the observed failure pattern most
- Forecast future failure data
  - Failures per time period
  - Time between failures

Software Reliability Models

- Many software reliability models exist
  - Examples: Musa Basic, Musa exponential, Schniderwind, S-shaped, etc.
- Some Assumptions for the Models
  - Failures are not related to each other
  - With and without fault removal?
  - There exists no undetected or unobserved failure
  - Each model has its own assumptions – a model may be selected based on how well they match the model specific assumptions

Musa Basic SRG Model

- Model assumptions
  - Failure intensity function decreases uniformly (derivative with respect to the number of expected failures is constant)

$\acute{Y}(\mu) = \acute{Y}_0(1 - \mu/v_0)$

$\acute{Y}(\mu)$: failure intensity (no. of failures/hour)

$\acute{Y}_0$: initial failure intensity at the start of execution

μ : average total number of failures, at a given point in time

v_0 : total number of failures over infinite time

Logarithmic Model

- A.K.A Musa/Okomoto Model
- Useful when software is used in a non-uniform manner
  - $\acute{Y}(\mu) = \acute{Y}_0 \exp(-\text{theta} \times \mu)$
  - theta: failure intensity decay parameter (unit/failure)
- Differences in assumptions
  - Basic model - zero failure intensity
  - Logarithmic model - convergence to zero failure intensity
  - Basic model - finite number of failures may occur
  - Logarithmic model - infinite number of failures may occur

Summary

- Software reliability vs. hardware reliability
  - Achieving software reliability is more difficult
- Software Reliability Terminology
  - Error, fault, and failure
  - Failure functions, failure data collection
  - Operational profile and software reliability models
- Software Reliability Engineering Process
  - Define necessary reliability
  - Develop operational profile
  - Prepare for test
  - Execute test
  - Apply failure data to guide decisions
- Software Reliability Growth Modeling
  - Reliability modeling process
  - Some popular models

Lecture Sources

- John Musa, Software Reliability Engineering, McGraw-Hill, 1999.
- Paul Rook (editor), Software Reliability Handbook, Kluwer Academic Publishers, 2002.
- Hoang Pham, Software Reliability, Springer, 2000.
- Michael R. Lyu (Editor), Handbook of Software Reliability Engineering, McGraw Hill Text, 1996.
- Pressman, Roger S., Software Engineering: A Practitioner's Approach, 4th ed., McGraw-Hill, 1997.

Presentation /Report 1 - Project Proposal

- Tell what you want to do for your project – set your own requirements
- Motivation and background
- Proposed work and research challenges
- Milestones for the project
- Presentation Length: 20 minutes (+10 minutes Q/A)
- Report Length: 4 pages in IEEE CS Proceedings paper format (a template may be available on the web)

Presentation/Report

- Important for Report – Copy Versus Paraphrase
  - If you use words/sentences,/paragraphs directly from another paper/book/website, put the imported words/sentences/paragraphs in quotation marks
  - It is better to paraphrase
  - In any case, you must reference the source

Report/Presentation - Very Useful Links

- How (and How Not) to Write a Good Systems Paper
  - http://www.usenix.org/events/samples/submit/advice.html
- A Simple (Incomplete) Check-list for Writing Papers
  - http://www.ece.umn.edu/users/sachin/misc/writing.html
- Advice on Research and Writing
  - http://www-2.cs.cmu.edu/afs/cs.cmu.edu/user/mleone/web/how-to.html