Introduction to Software Reliability Estimation

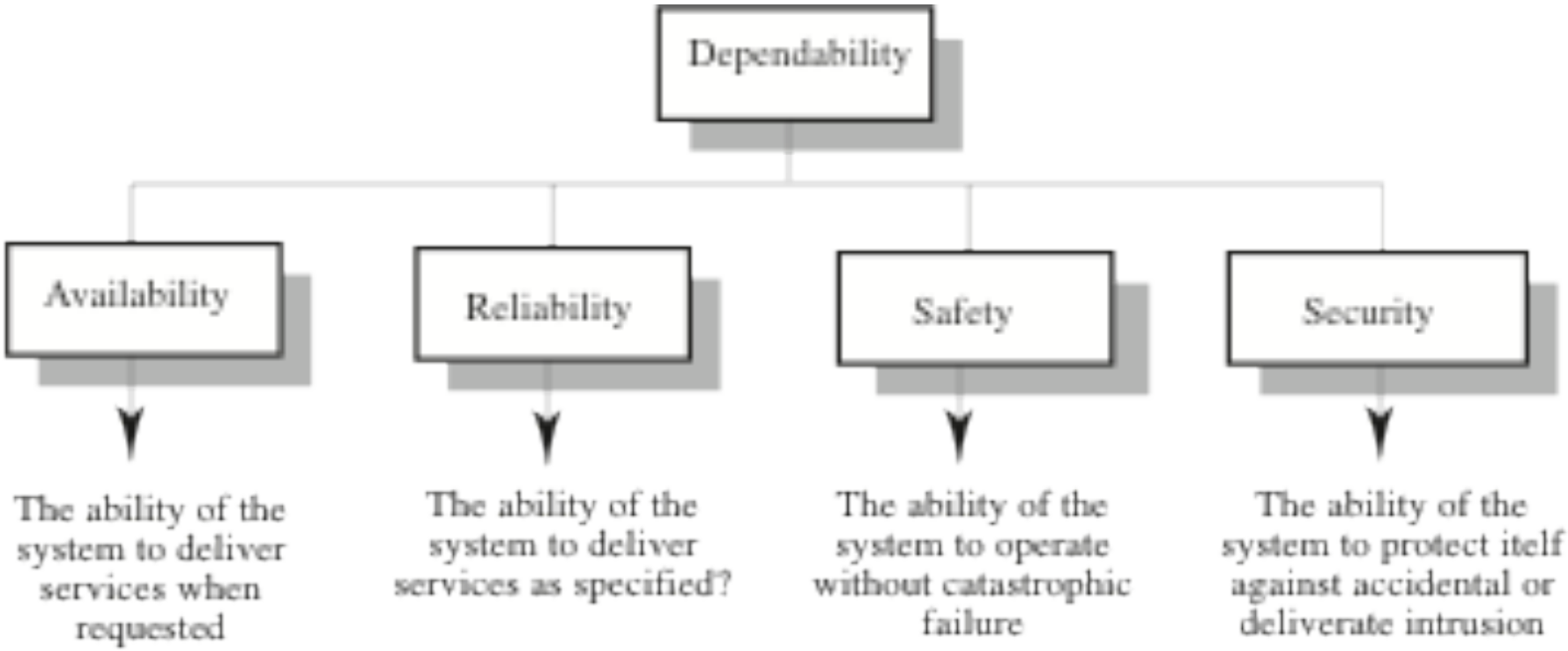
Motivations

- Software is an essential component of many safety-critical systems
- These systems depend on the reliable operation of software components
- How can we decide, whether the software is ready to ship after system test?
- How can we decide, whether an acquired or developed component is acceptable?

What is Reliability?

- IEEE-Std-729-1991: “Software reliability is defined as the probability of failure-free operation for a specified *period of time* in a specified *environment*”
- ISO9126: “Reliability is the capability of the software product to maintain a specified level of performance when used under specified conditions”
- Informal: Reliability is a measure of how well the software provides the services expected by the customer.
- Quantification: Number of failures, severity

Reliability as part of Dependability

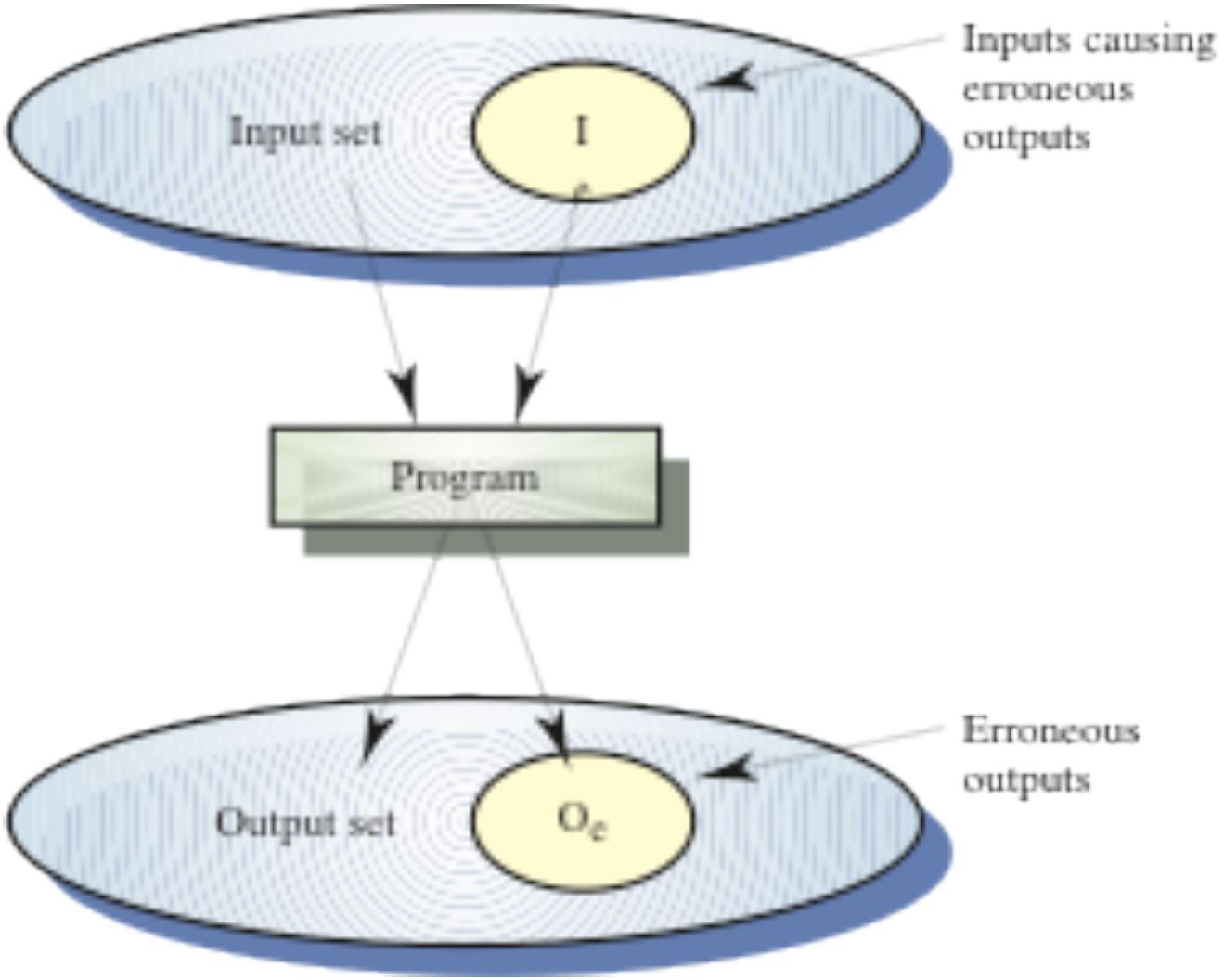


Sommerville

Hardware Reliability

- Well-developed reliability theories
- A hardware component may fail due to design error, poor fabrication quality, momentary overload, deterioration, etc.
- The random nature of system failure enables the estimation of system reliability using probability theory
- Can it be adapted for software? No physical deterioration, most programs always provide the same answer to the same input, the “fabrication” of code is trivial, etc.

Input/Output Mapping

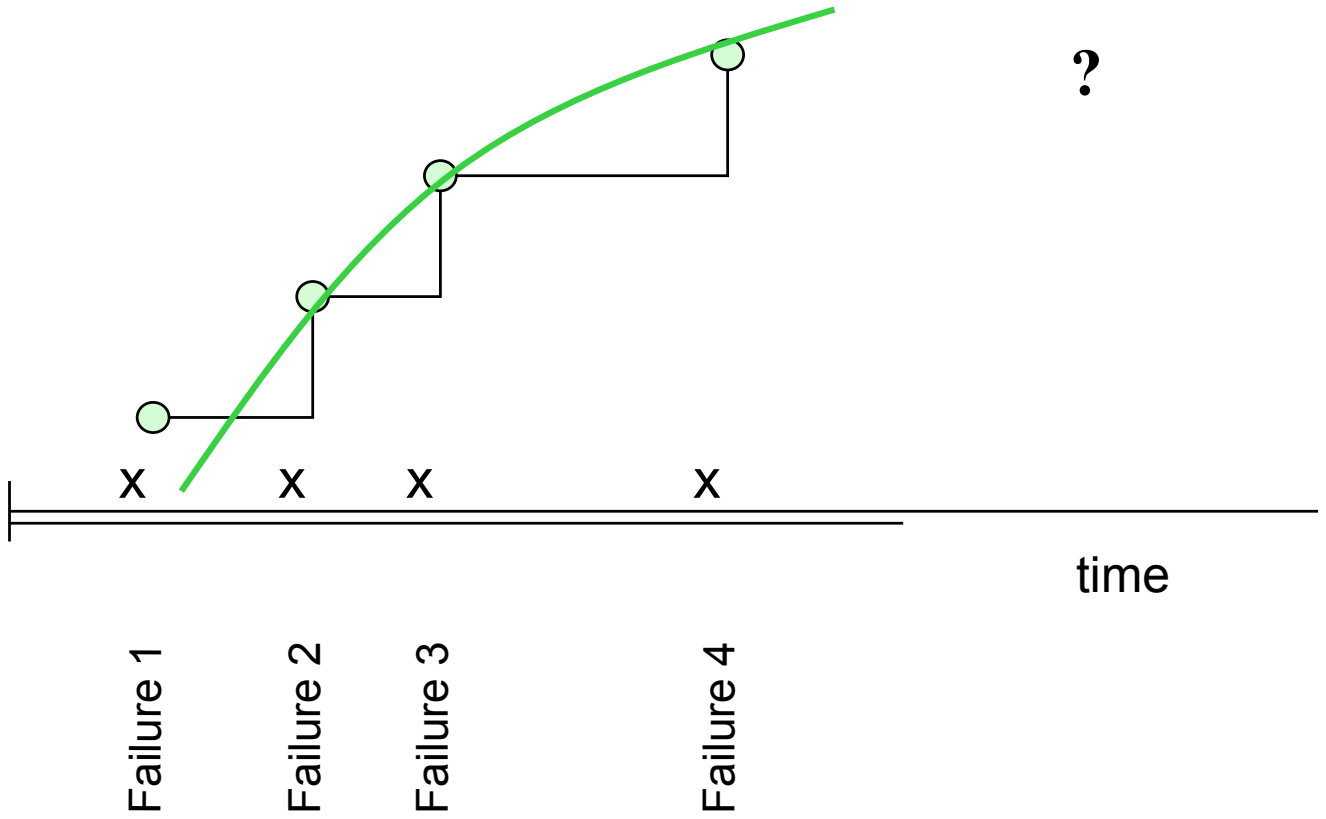


Sommerville

How Can we Measure and Model Reliability

- The cumulative number of failures by time t
- *Failure Intensity*: The number of failures per time unit at time t
- *Mean-Time-To-Failure MTTF*: Average value of inter-failure time interval
- We do not know for sure when the software is going to fail next => model failure behavior as a random process

Modeling Cumulative # Failures



Software Reliability Engineering

- No method of development can guarantee totally reliable software => important field in practice!
- A set of statistical modeling techniques
- Enables the achieved reliability to be *assessed* or *predicted*, quantitatively and objectively
- Based on observation of system failures during system testing and operational use
- Uses general reliability theory, but is much more than that

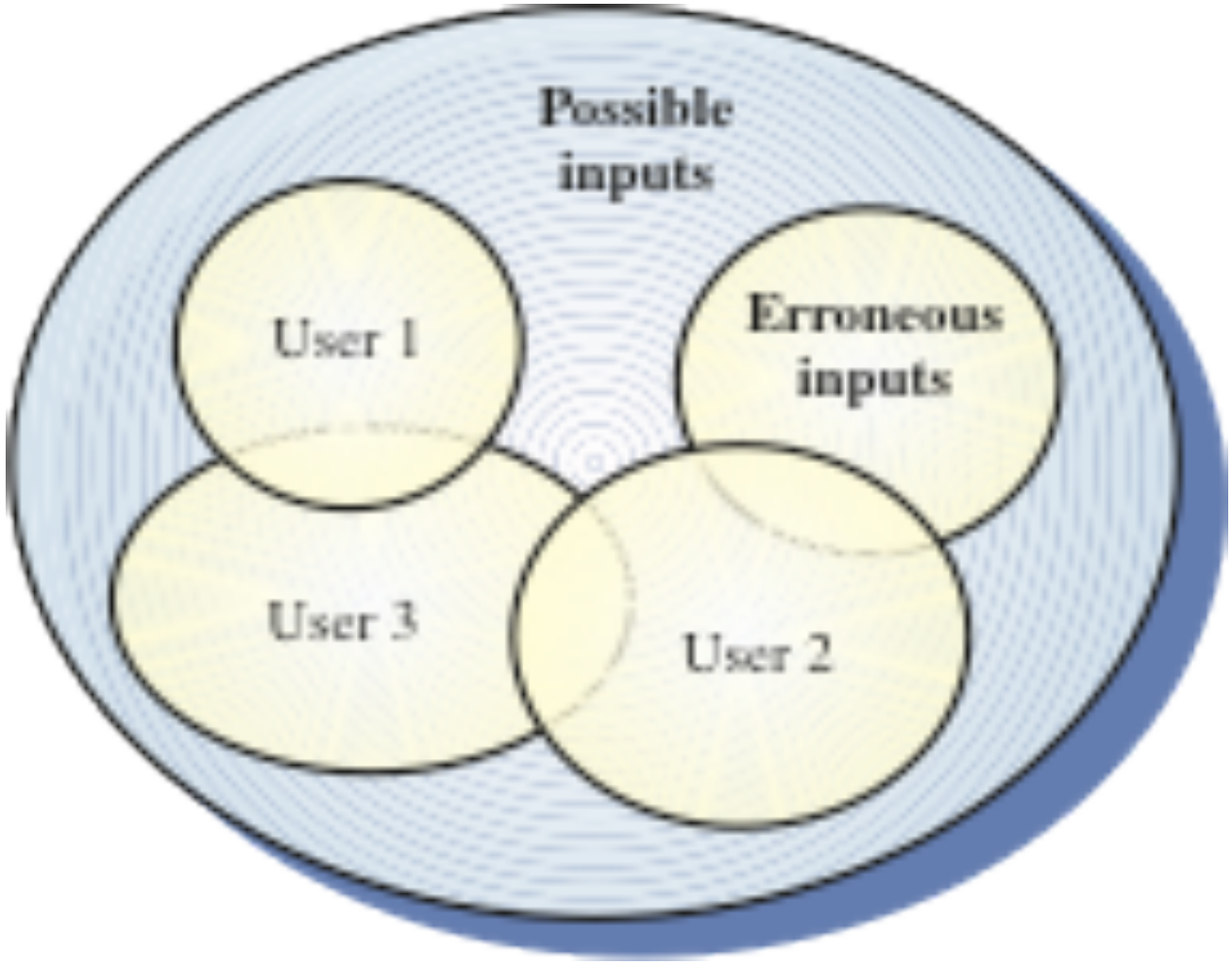
Applications

- How reliable is the program/component now?
- Based on the current reliability of a purchased/reused component: can we accept it or should we reject it?
- Based on the current reliability of the system: can we stop testing and start shipping?
- How reliable will the system be, if we continue testing for some time?
- When will the reliability objective be achieved?
- How many failures will occur in the field (and when) – to help plan resources for customer support?

Problems with Quantification?

- US standard for equipment on civil aircraft:
  - [A critical failure must be] so unlikely that [it] need not be considered ever to occur, unless engineering judgment would require [its] consideration.
- *Typical requirement for critical flight systems*: 10^{-9} probability of failure per flying hour based on a flight mean duration of 10 hours.
- *Problem 1*: such a level of reliability cannot be demonstrated – would require too much testing time
- *Problem 2*: Reliability is relative to the usage of the system

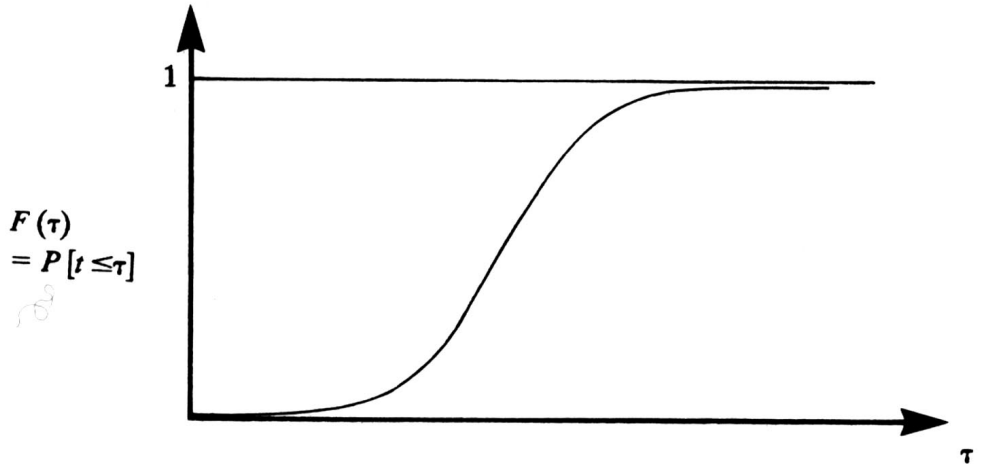
Usage Patterns



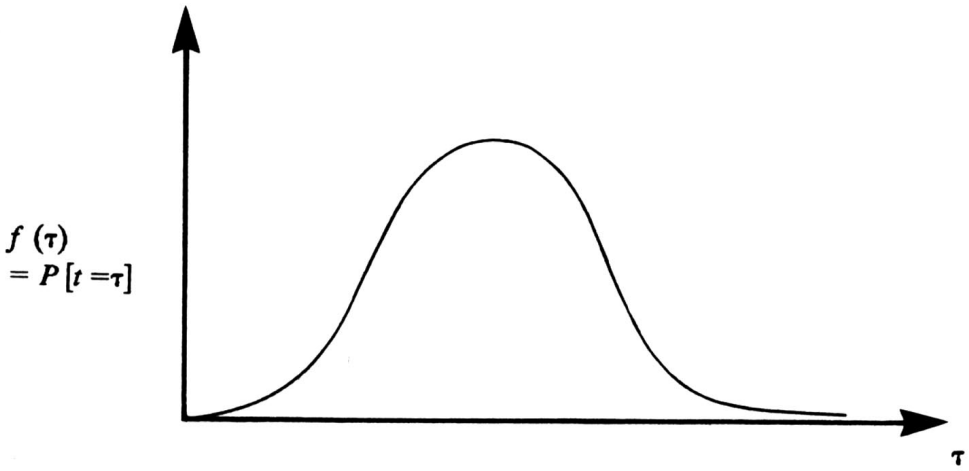
Sommerville

Basics of Reliability Theory

- Random variable to be modeled: time to failure *t*
- P: Probability that the time to failure is within some time interval
- F(t) value of the cumulative probability distribution function at point t
- f(t) value of the probability density function at point t
- Question of interest: $P(\tau \le t \le \tau + \Delta\tau)$ where $\Delta\tau$ may represent the time of a mission for a spacecraft
- $P(\tau \le t \le \tau + \Delta\tau) = f(\tau)\Delta\tau = F(\tau+\Delta\tau) - F(\tau)$
- If we let $\Delta\tau \to 0$, $f(\tau) = \dfrac{dF(\tau)}{d\tau}$



(b) Defect interval distribution function.

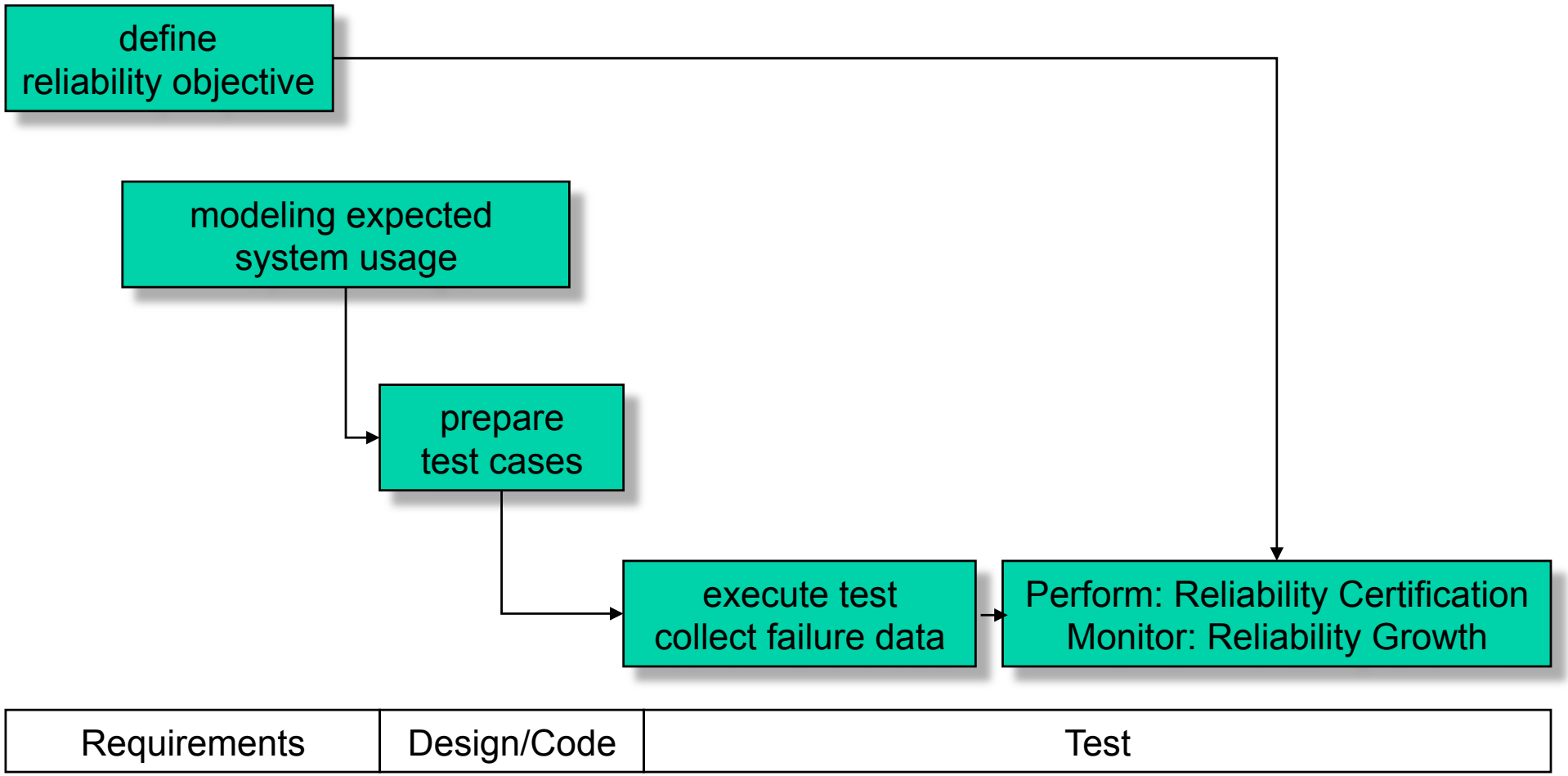


(c) The probability density function.

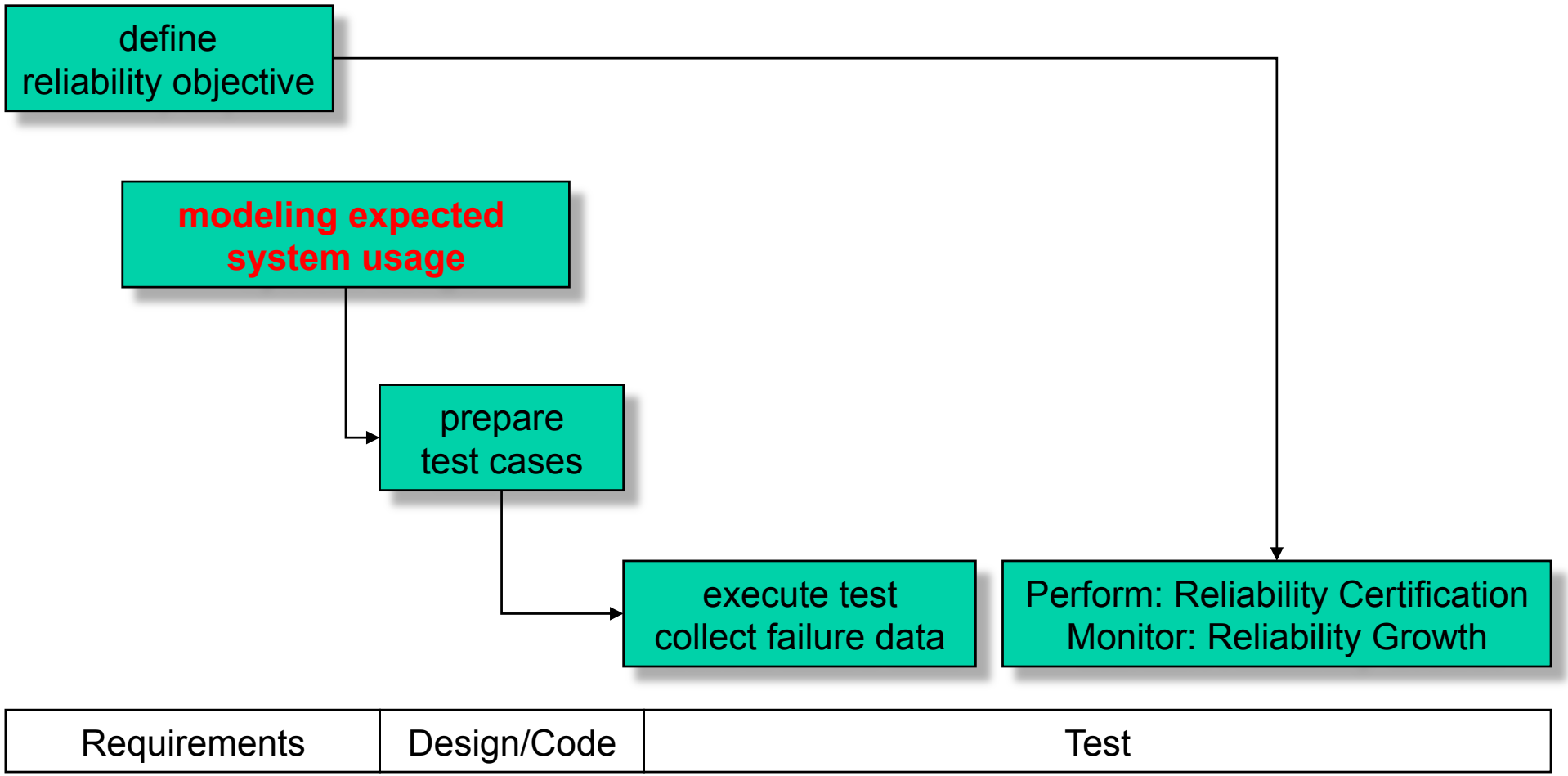
Basics of Reliability Theory II

- $P_f(\tau) = P(0 \le t \le \tau) = F(\tau) - F(0)$
- *Obviously F(0) is 0*
- $P_f(\tau) = F(\tau) = \int_0^{\tau} f(x)dx$
- *Reliability:* $R(\tau) = 1 - P_f(\tau) = \int_{\tau}^{\infty} f(x)dx$
- $MTTF = E(t) = \int_0^{\infty} \tau\, f(\tau)d\tau = \int_0^{\infty} R(\tau)d\tau$

Software Reliability Engineering Process



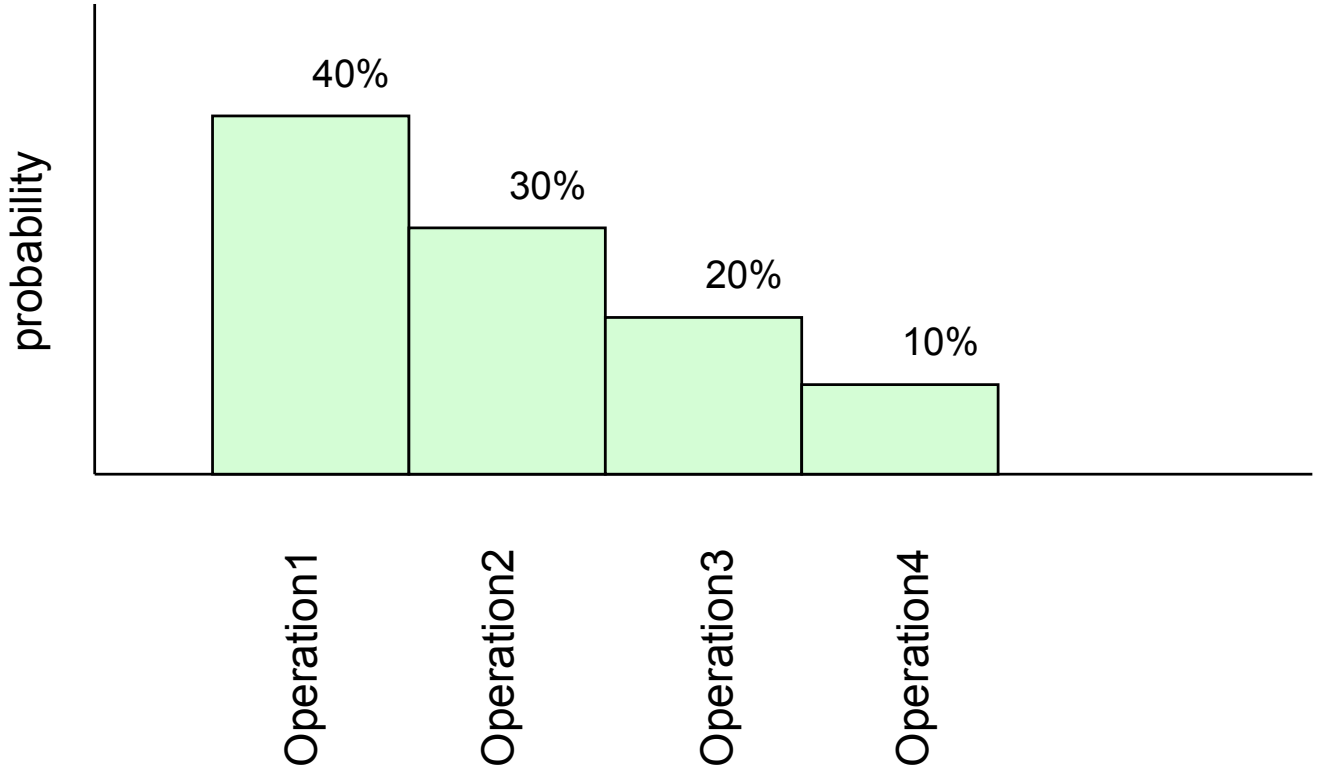
Software Reliability Engineering Process



Model Expected System Usage

- Definition of reliability assumes a *specified environment*
  - ⇨ To make statements on reliability in field during system test, we must test in conditions that are "similar to field conditions"
- *Model how users will employ the software*: environment, type of installation, distribution of inputs over input space
- According to the usage model, test cases are selected *randomly*
- One example of usage model
  - *Operational Profile* (Musa): Set of system operations and their probabilities of occurrence

Operational Profile



Operations

- Major system logical task of short duration, which returns control to the system when complete and whose processing is substantially different from other operations
  - major: related to functional requirement
  - logical: not bound to software, hardware, users, machines
  - short: 100s-1000s operations per hour under normal load conditions
  - different: likely to contain different faults
- In OO systems, an “operation” is ~ use case

Examples

- Command executed by a user
- Response to an input from an external system or device, e.g., processing a transaction, processing an event (alarm)
- Routine housekeeping, e.g., file backup, database cleanup

Develop Operational Profiles

- *Identify who/what can initiate operations*
  - Users (of different types), external systems and devices, system itself
- *Create a list of operations for each operation initiator and consolidate results*
  - Source: requirements, draft user manuals, prototypes, previous program versions, discuss with expected users
  - 20 to several hundred operations are typical
- *Determine occurrence rates (per hour) of the individual operations*
  - existing field data, record field operations, simulation, estimates
- *Derive occurrence probabilities*

Example: Fone Follower (FF)

- Requirements
  - Forward incoming phone calls (voice, fax) anywhere
  - Subscriber calls FF, enters phone numbers for where he plans to be as a function of time
  - FF forwards incoming calls from network (voice, fax) to subscriber as per program. If no response to voice call, subscriber is paged (if subscriber has pager). If no response or no pager, voice calls are forwarded to voice mail
  - Subscribers view service as standard telephone service combined with FF
  - FF uses vendor-supplied operating system of unknown reliability

Initiators of Operations

- Event driven systems often have many external systems that can initiate operations in them
- Typically, the system under study may initiate itself administrative and maintenance operations
- FF:
  - User types: subscribers, system administrators
  - External system: Telephone network
  - FF (audits, backups)

FF Operations List

- **Subscriber**
  - Phone number entry
- **System Administrator**
  - Add subscriber
  - Delete subscriber
- **Network**
  - Proc. voice call, no pager, answer
  - Proc. voice call, no pager, no answer
  - Proc. voice call, pager, answer
  - Proc. voice call, pager, ans. on page
  - Proc. voice call, pager, no ans. on page
  - Proc. fax call
- **FF**
  - Audit section of phone number database
  - Recover from hardware failure

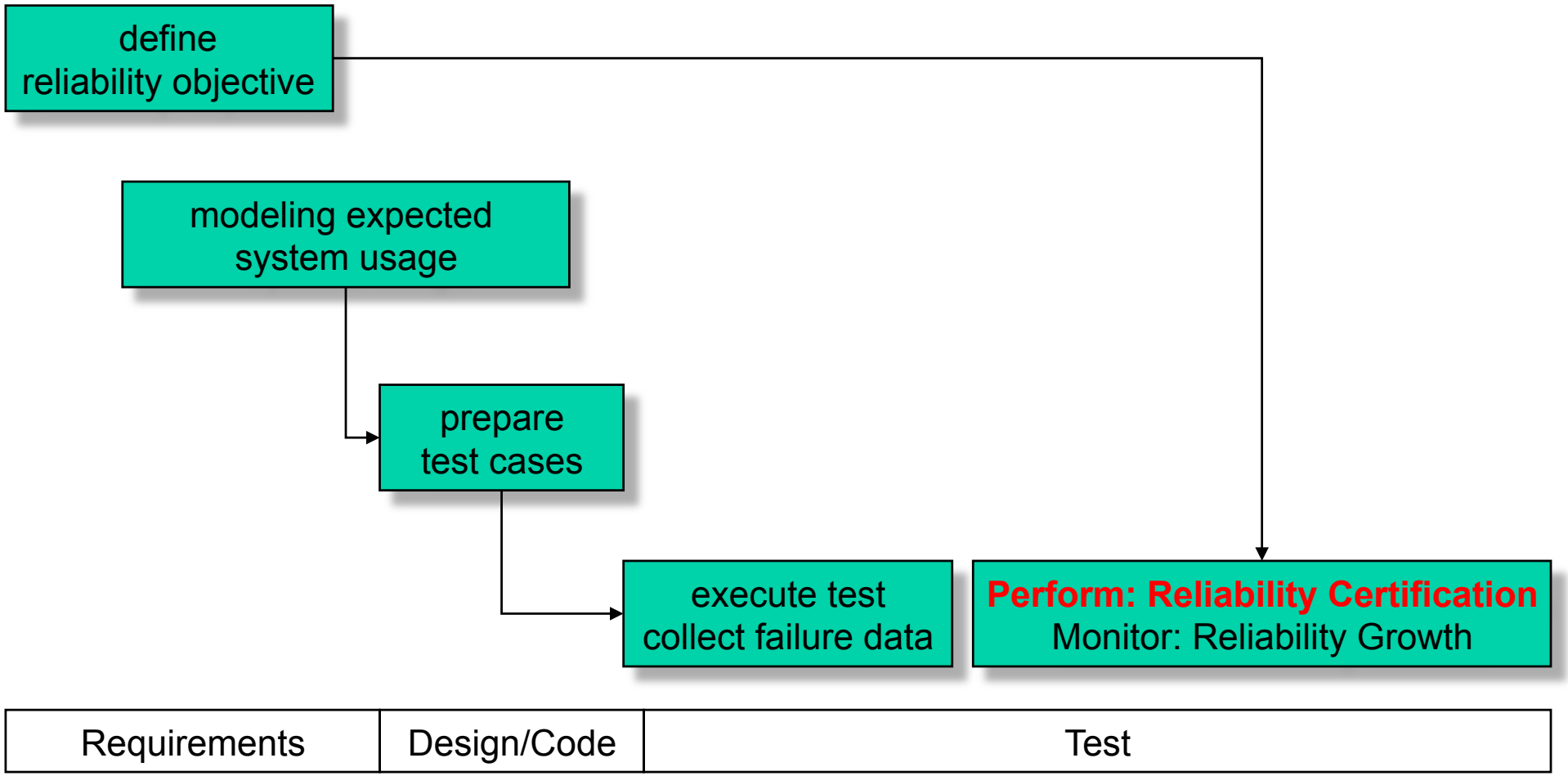
FF Operational Profile

Operation	Occ.Rate (per hr)	Occ.Prob.
Phone number entry	10.000	0,10
Add subscriber	50	0,0005
Delete subscriber	50	0,0005
Proc. voice call, no pager, answer	18.000	0,18
Proc. voice call, no pager, no answer	17.000	0,17
Proc. voice call, pager, answer	17.000	0,17
Proc. voice call, pager, ans. on page	12.000	0,12
Proc. voice call, pager, no ans. on page	10.000	0,10
Proc. fax call	15.000	0,15
Audit section of phone number database	900	0,009
Recover from hardware failure	0,1	0,000001

Statistical Testing

- Testing based on operational profiles is referred to as *statistical testing*
- This form of testing has the advantage of testing more intensively the system functions that will be used the most
- Good for reliability estimation, but not very effective in terms of finding defects
- Hence we differentiate testing that aims at finding defects (verification) and testing whose purpose is reliability assessment (validation).
- There exists research on techniques that combine white-box testing and statistical testing …

Software Reliability Engineering Process

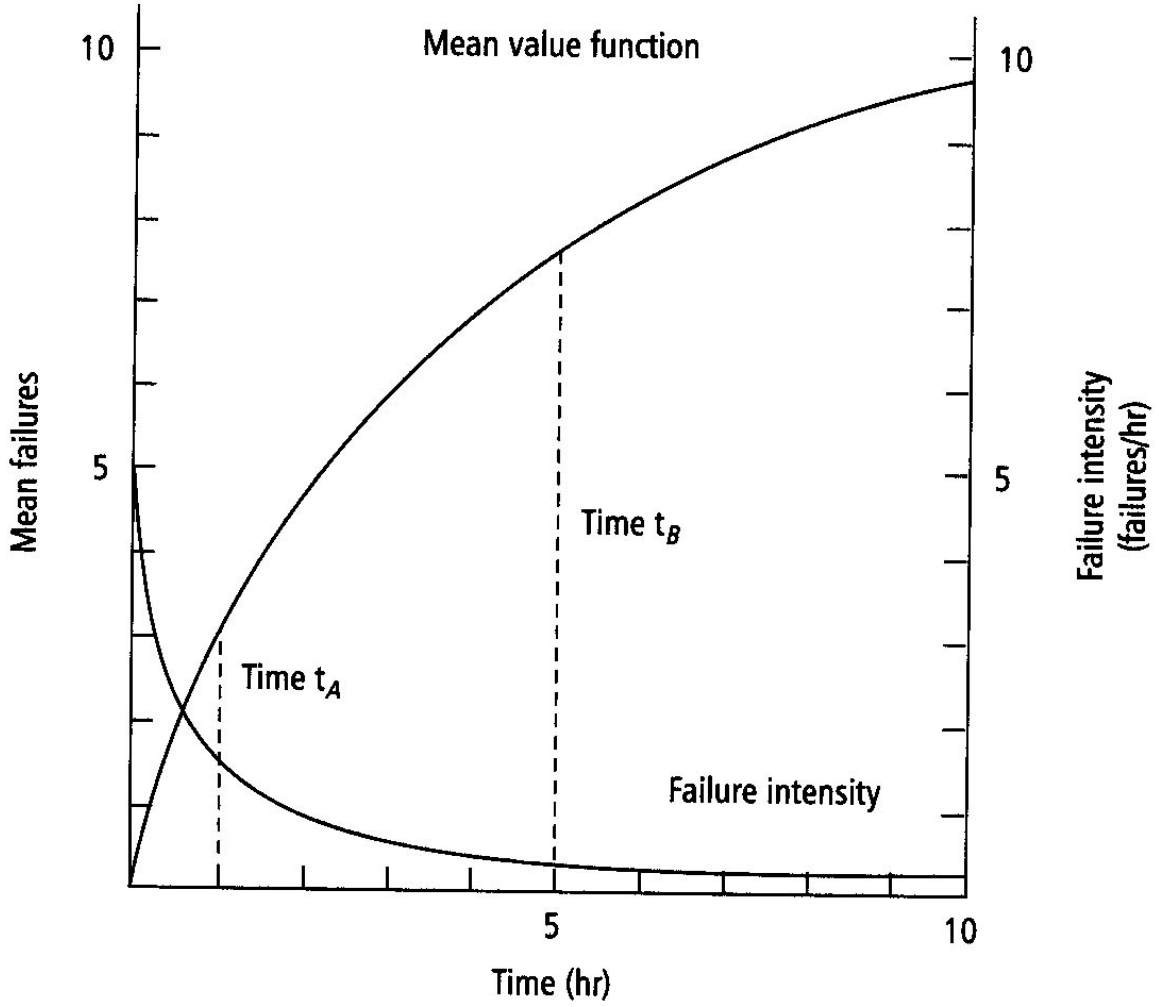


Failure Intensity

- $\mu(t) = E[M(t)]$, with $M(t)$ be the random process denoting the cumulative number of failures by time t, we denote $\mu(t)$ as its mean value function
- $\lambda(t)$: Failure intensity, average number of failures by unit of time at time t, instantaneous rate of change of the *expected number of failures* ($\mu(t)$)
- $\lambda(t)$ often a useful way to look at reliability, as an alternative to mean time to failures (MTTF), e.g., Telecom example next
- MTTF is probably more suitable for safety-critical systems, such as spacecrafts, sent on a mission of determined time
- Reliability growth implies: $d\lambda(t)/dt < 0$

$$\lambda(t) = \frac{d\mu(t)}{dt}$$

Graphical Representation



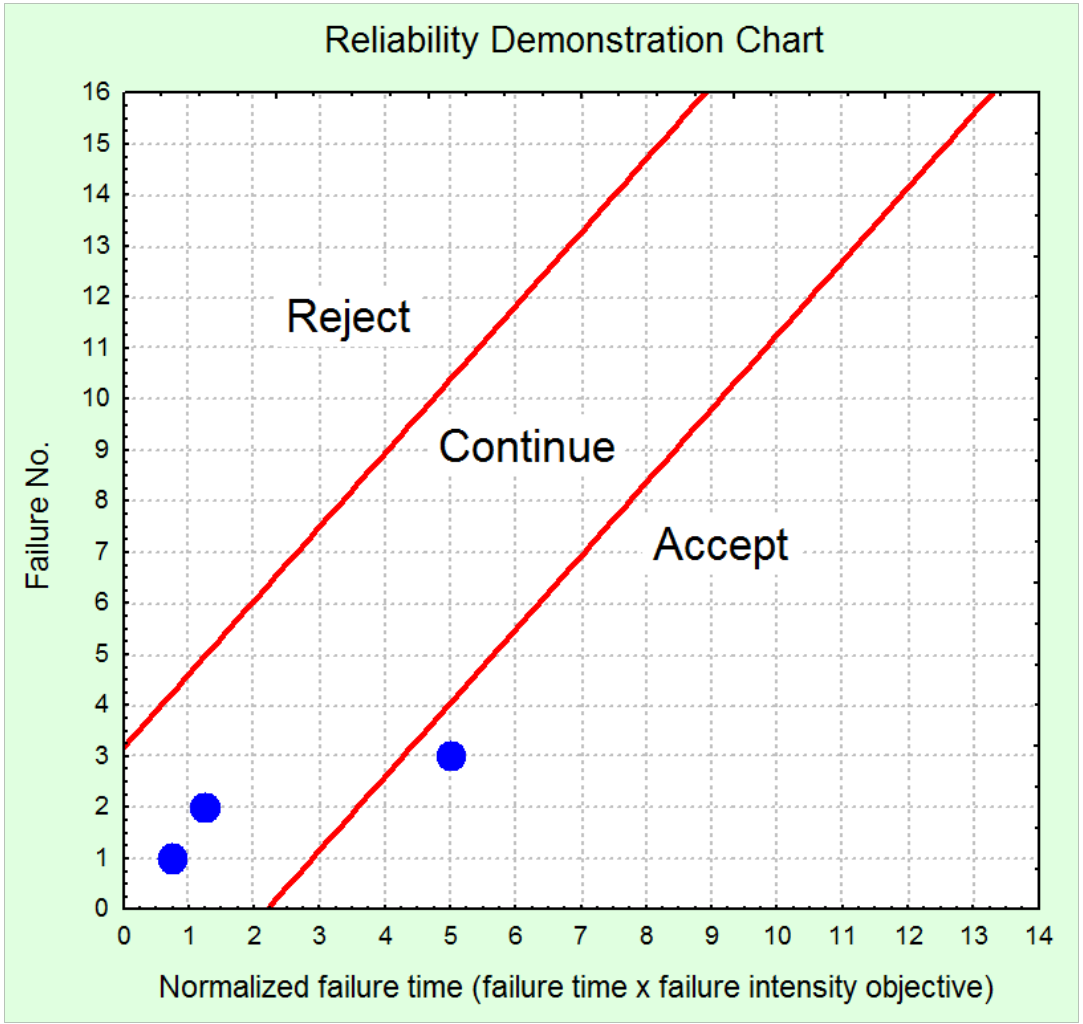
Can We Accept a Component?

- *Certification Testing*: Show that a (acquired or developed) component satisfies a given reliability objective (e.g., failure intensity)
- Generate test data randomly according to usage model (e.g., operational profile)
- Record all failures (also multiple ones) but *do not correct*

Procedure

- Use a *hypothesis testing control chart* to show that the reliability objective is/is not satisfied
  - *Reliability Demonstration Chart*
    - Collect times at which failures occurred
    - Normalize data by multiplying with failure intensity objective (using same units!)
    - Plot each failure in chart
    - Based on region in which failure falls, accept or reject component

Reliability Demonstration Chart



Fail.No	Mcalls at failure	Normalized measure
1	0.1875	0.75
2	0.3125	1.25
3	1.25	5.0

=> accept

Failure Intensity Objective:
4 failures/Mcalls

John Musa, Software Reliability, 1998

Creating Demonstration Charts

- Select *discrimination ratio* γ (acceptable factor of error in estimating failure intensity)
- Select *consumer risk* β (probability of accepting a system that does not satisfy failure intensity objective)
- Select *supplier risk* α (probability of rejecting a system that does satisfy failure intensity objective)
- Recommended defaults (γ=2, β=0.1, α=0.1)
  - 10% risk of wrongly accepting component when failure intensity is actually ≥ 2 * failure intensity objective
  - 10% risk of wrongly rejecting component when failure intensity is actually ≤ 1/2 * failure intensity objective

Boundary Lines

The next step is to construct boundary lines (between reject and Continue and Continue and Accept) according to the following formulae

$$T_N = \frac{A - n\ln\gamma}{1-\gamma} \qquad T_N = \frac{B - n\ln\gamma}{1-\gamma} \qquad A = \ln\frac{\beta}{1-\alpha} \qquad B = \ln\frac{1-\beta}{\alpha}$$

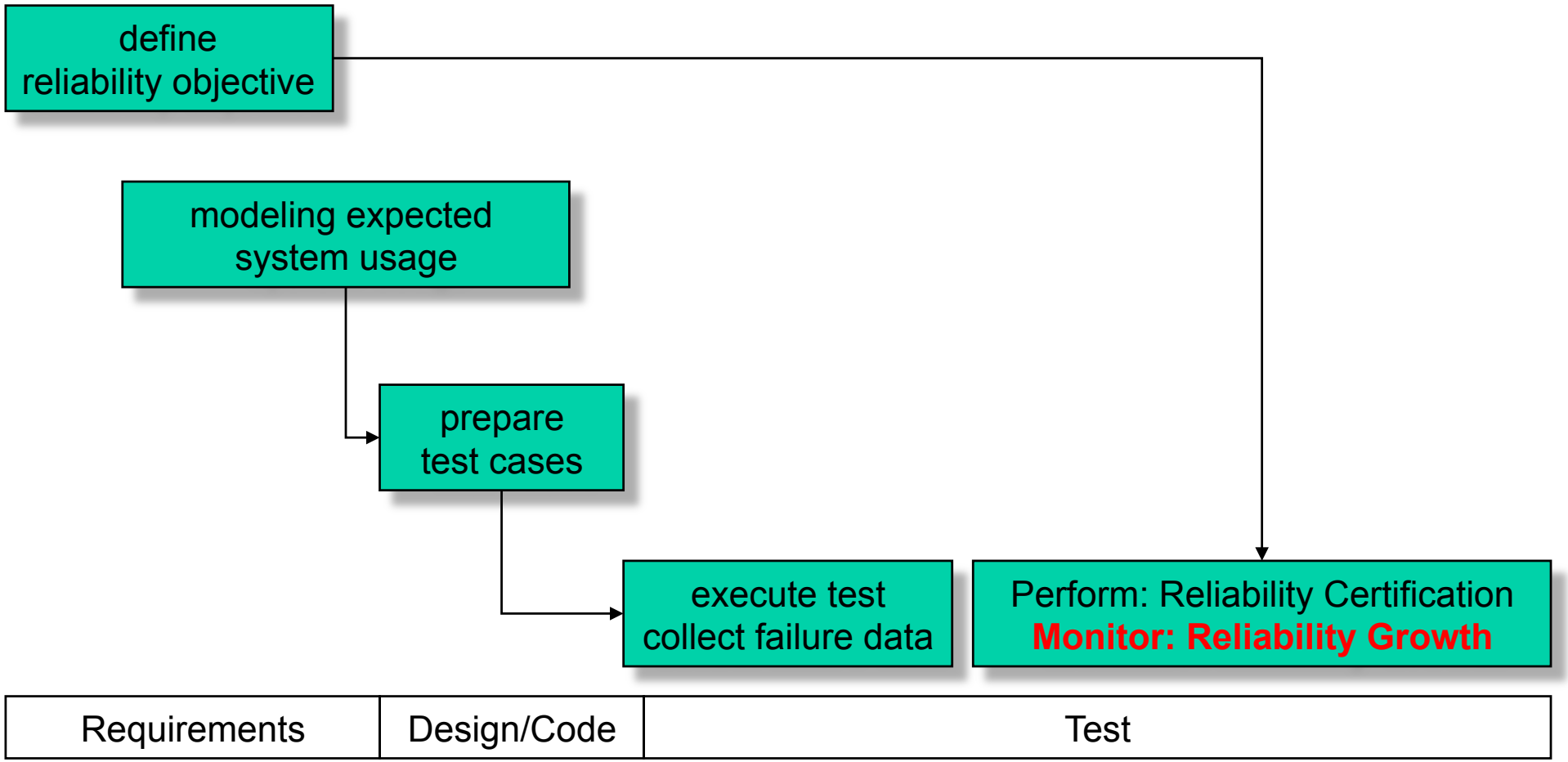
(Lower) (Higher)

Where

T_N = boundaries for normalized failure time (x axis)

n = failure number (y axis)

Software Reliability Engineering Process



Failure Data: Interfailure times

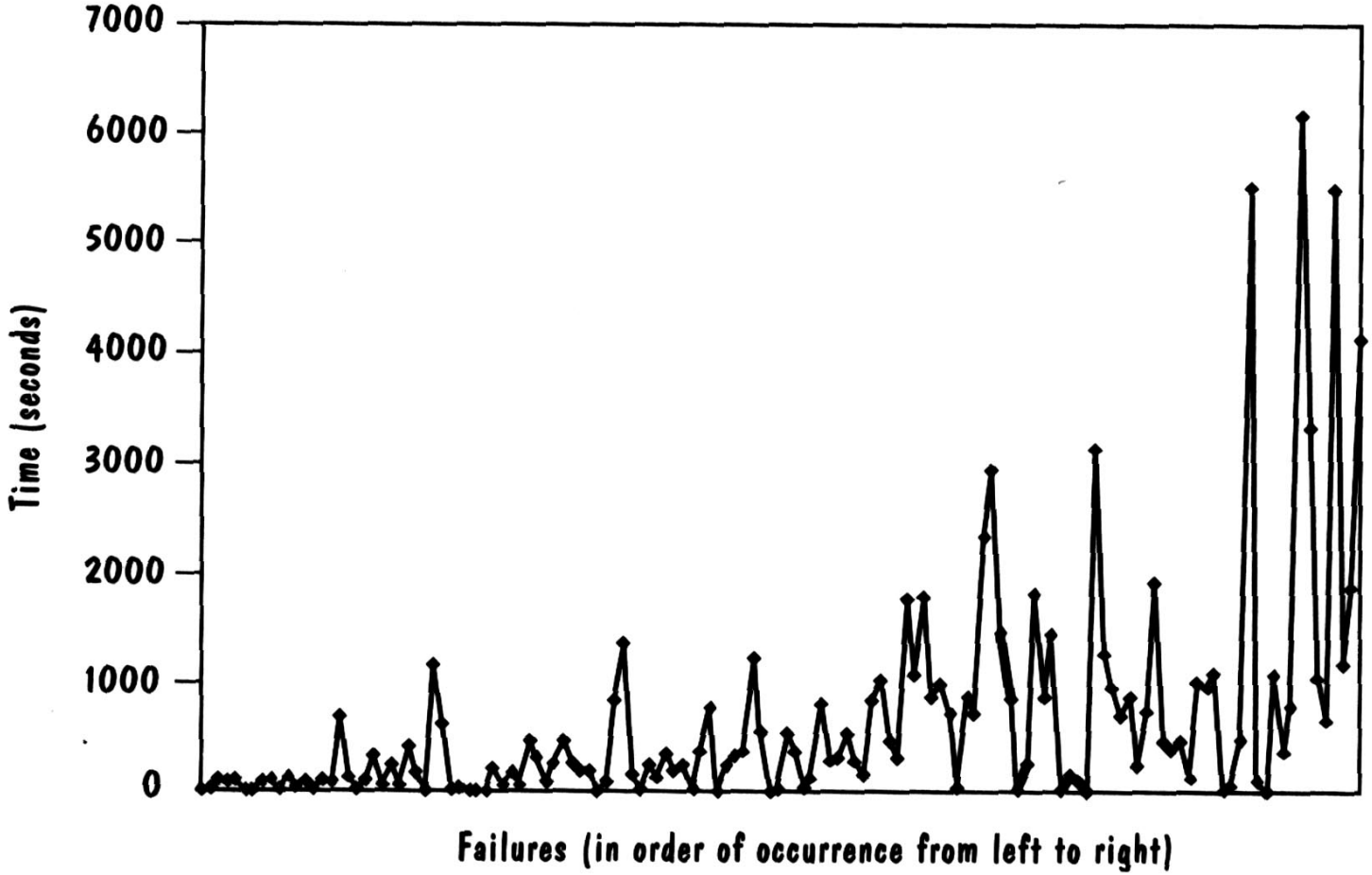
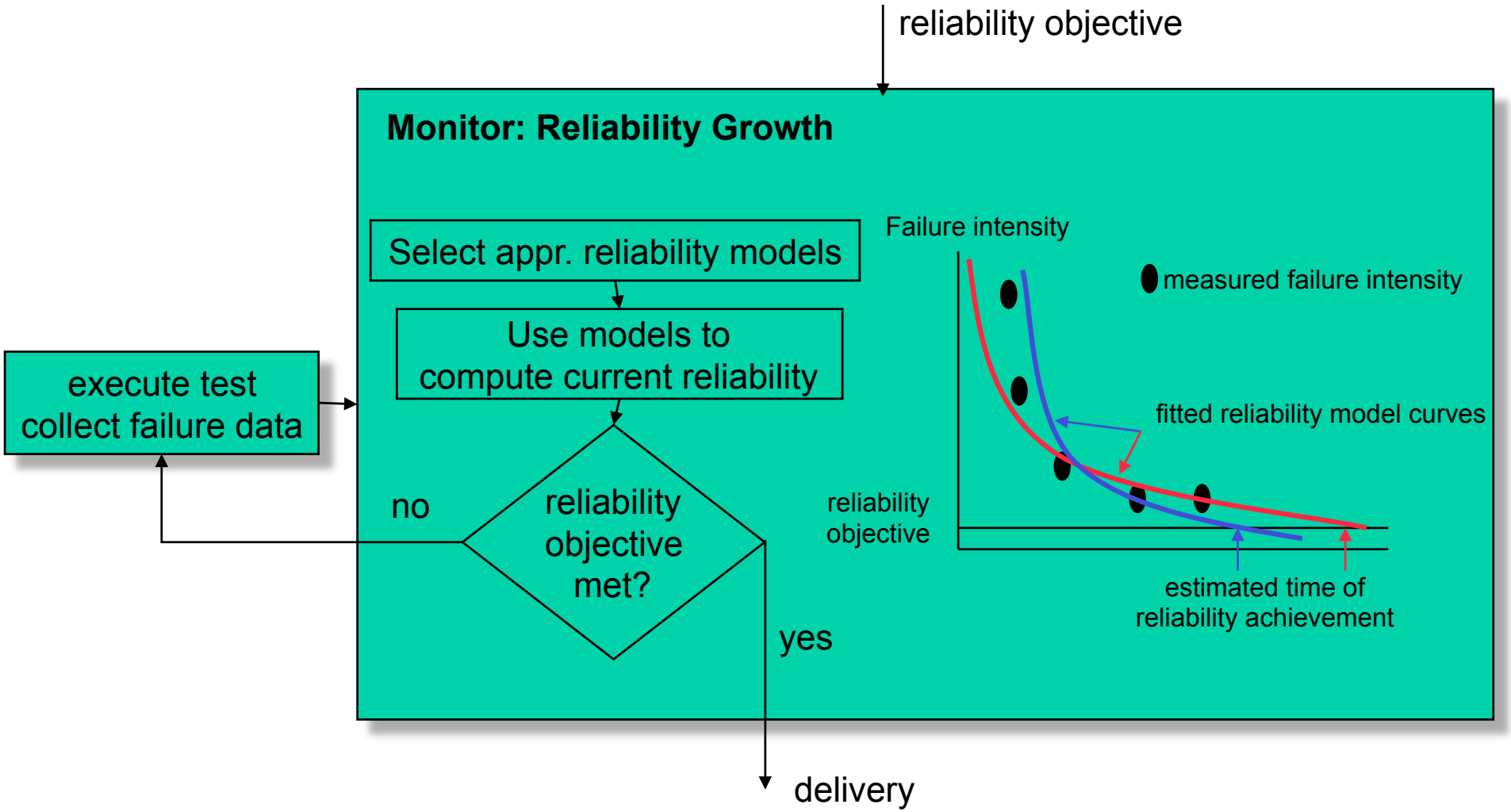


FIGURE 8.7 Graph of failure data from Table 8.3.

Can We Stop Testing?



The Musa-Okumoto Model

General Assumptions

- The system test reflects the intended usage of the system
- The failures are independent
- No new faults introduced during corrections

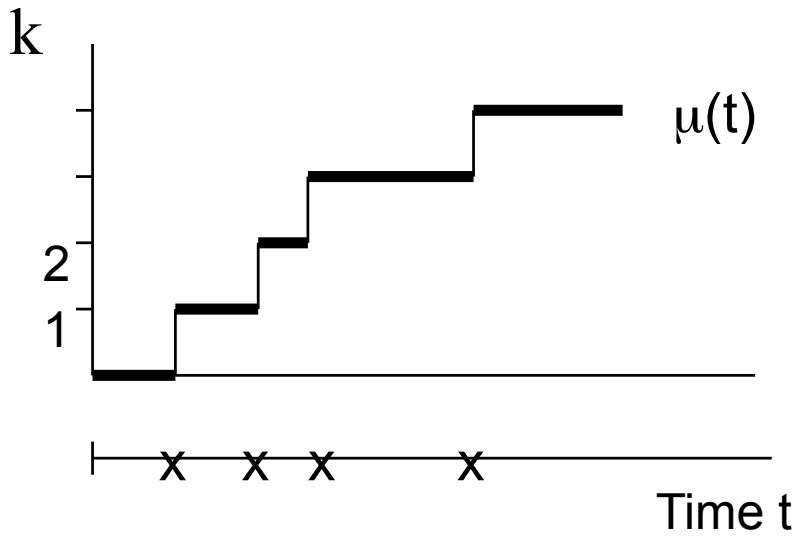
Model-specific Assumptions

- The cumulative number of failures by time t follows a Poisson Process (NHPP)
- Failure intensity decreases exponentially with the expected number of failures experienced μ:
  $\lambda(\mu)=\lambda_0 \exp(-\theta\mu)$
  $\theta>0$ is failure intensity decay parameter, λ_0 is initial failure intensity.

Data Requirements

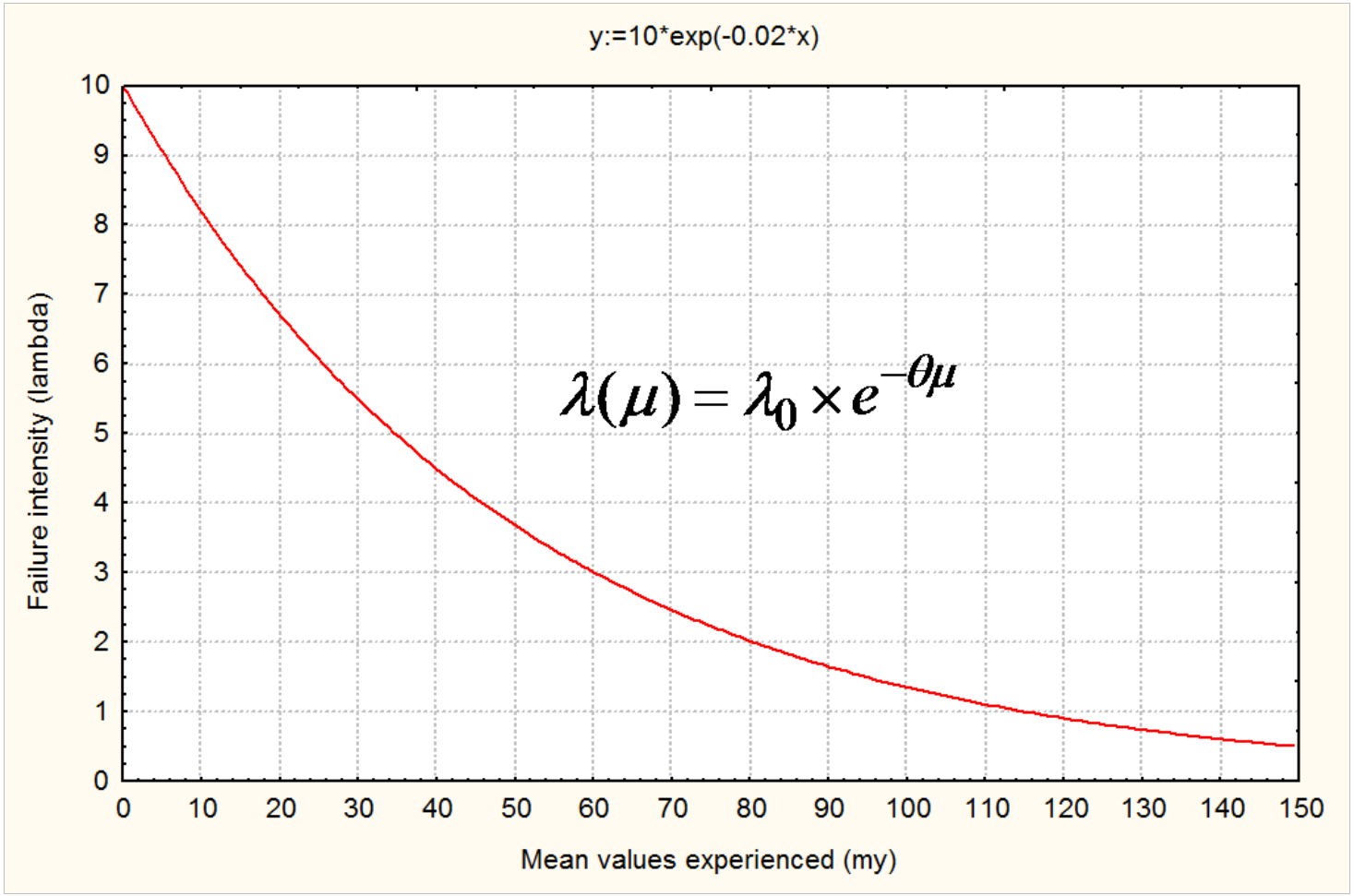
- Actual times when failures occurred or failure time intervals (execution time)

Poisson Processes



$$P_k(t) = e^{-\alpha t} \frac{(\alpha t)^k}{k!}$$

The Musa-Okumoto Model (2)



Failure intensity versus failures experienced

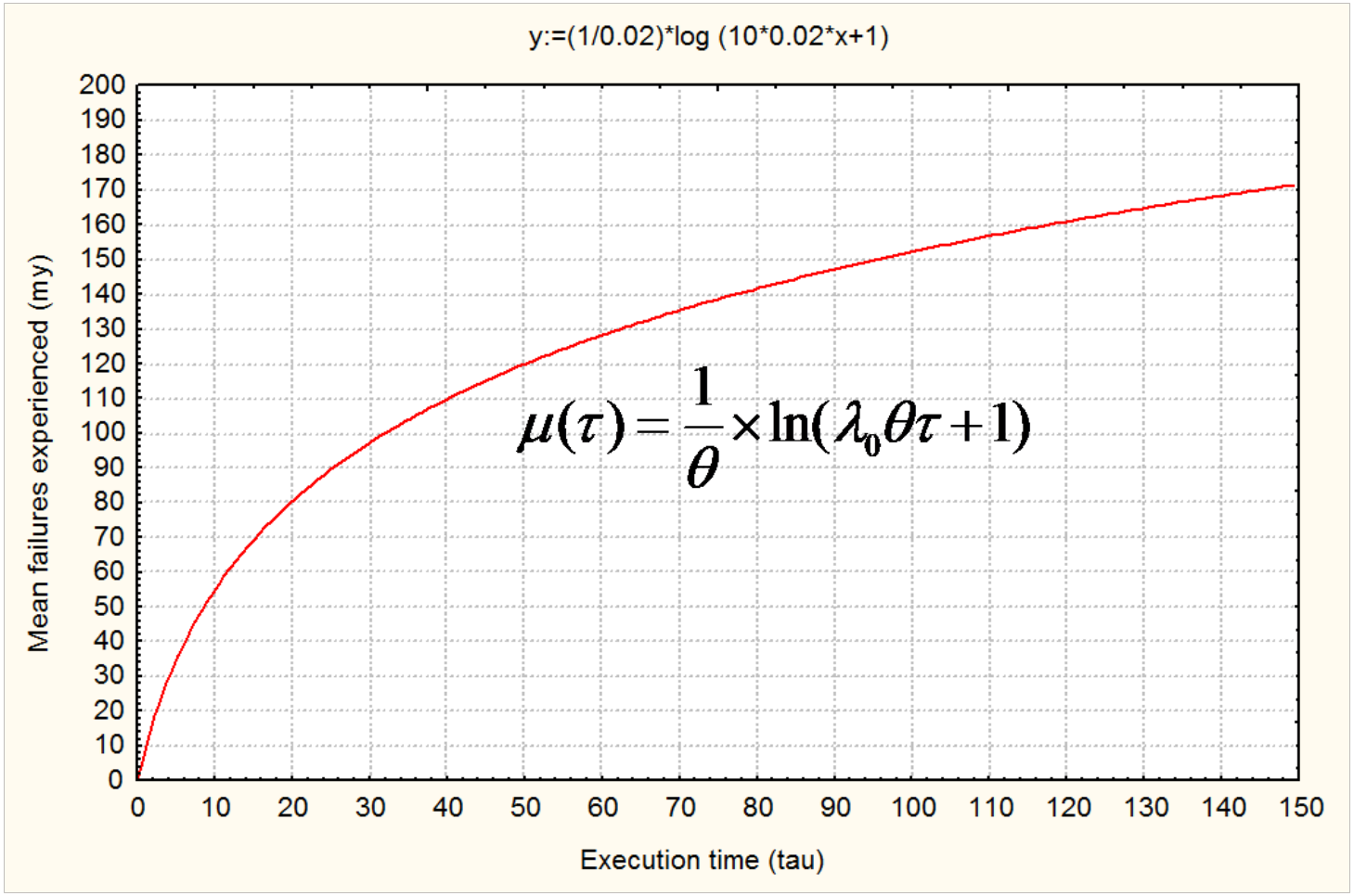
Deriving μ(t) and λ(t)

$$\lambda(\mu) = \lambda_0 \times e^{-\theta\mu}$$

$$\lambda(\tau) = \frac{d\mu(\tau)}{d\tau} = \lambda_0 \times e^{-\theta\mu(\tau)} = \frac{\lambda_0}{\lambda_0\theta\tau + 1}$$

$$\mu(\tau) = \frac{1}{\theta} \times \ln(\lambda_0\theta\tau + 1)$$

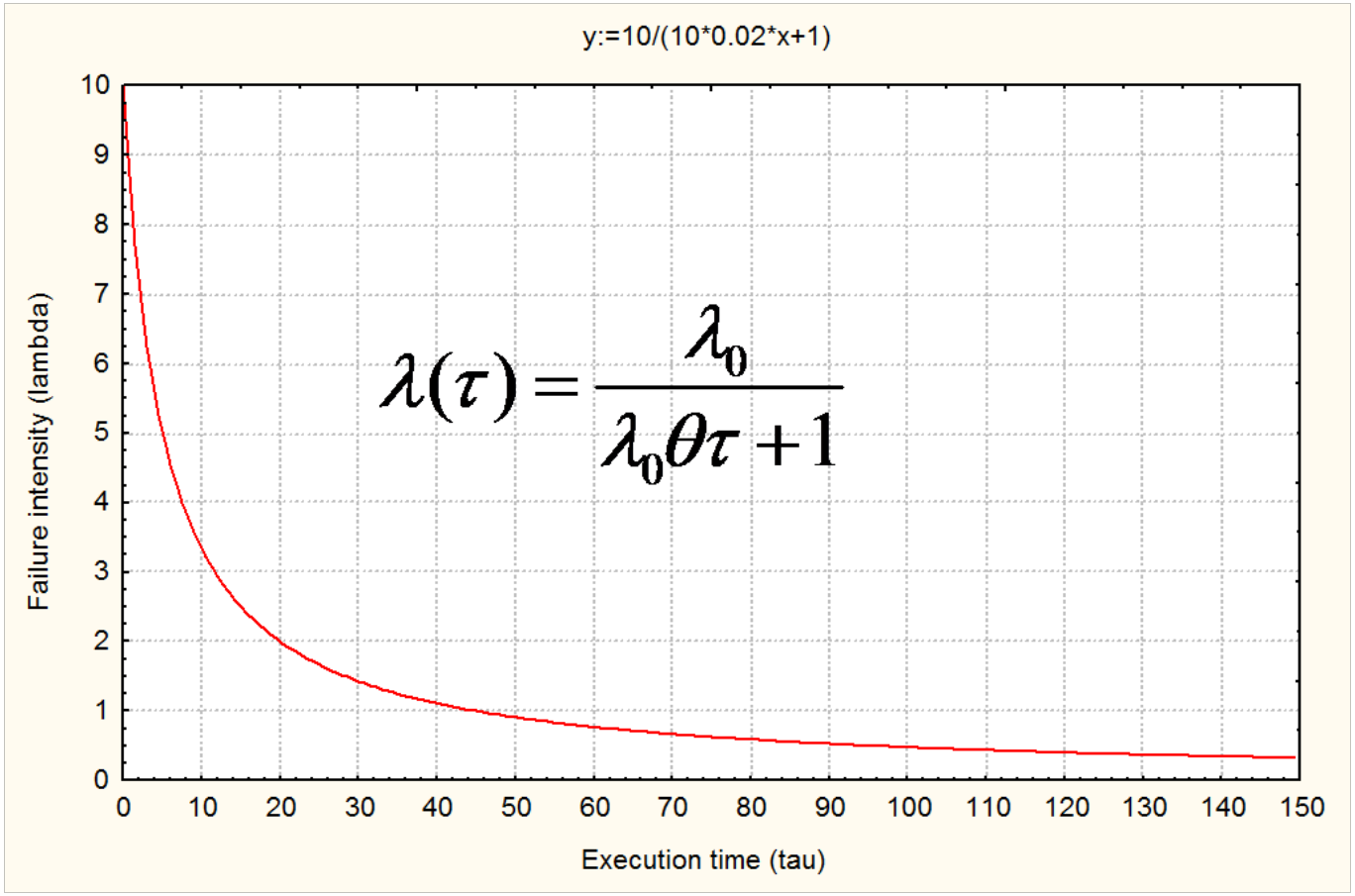
The Musa-Okumoto Model (3)



Failures experienced versus execution time τ

The Musa-Okumoto Model (4)

Failure intensity versus execution time τ



- **Collect failure data: time when failure occurred**
- **Use tools to determine model parameters λ_0 and θ**
- **Determine whether reliability objective is met or how long it might take to reach**

The Musa-Okumoto Model (5)

- Once the parameters λ_0 and θ are estimated, it can be used to predict failure intensity in the future, not only to estimate its current value. From this, we can plan how much additional testing is likely to be needed
- The model allows for the realistic situation where fault correction is not perfect (infinite number of failures at infinite time)
- When faults stop being corrected, the model reduces to a homogeneous Poisson process with failure intensity λ as the parameter – the number of failures expected in a time interval then follows a Poisson distribution

Reliability Estimation

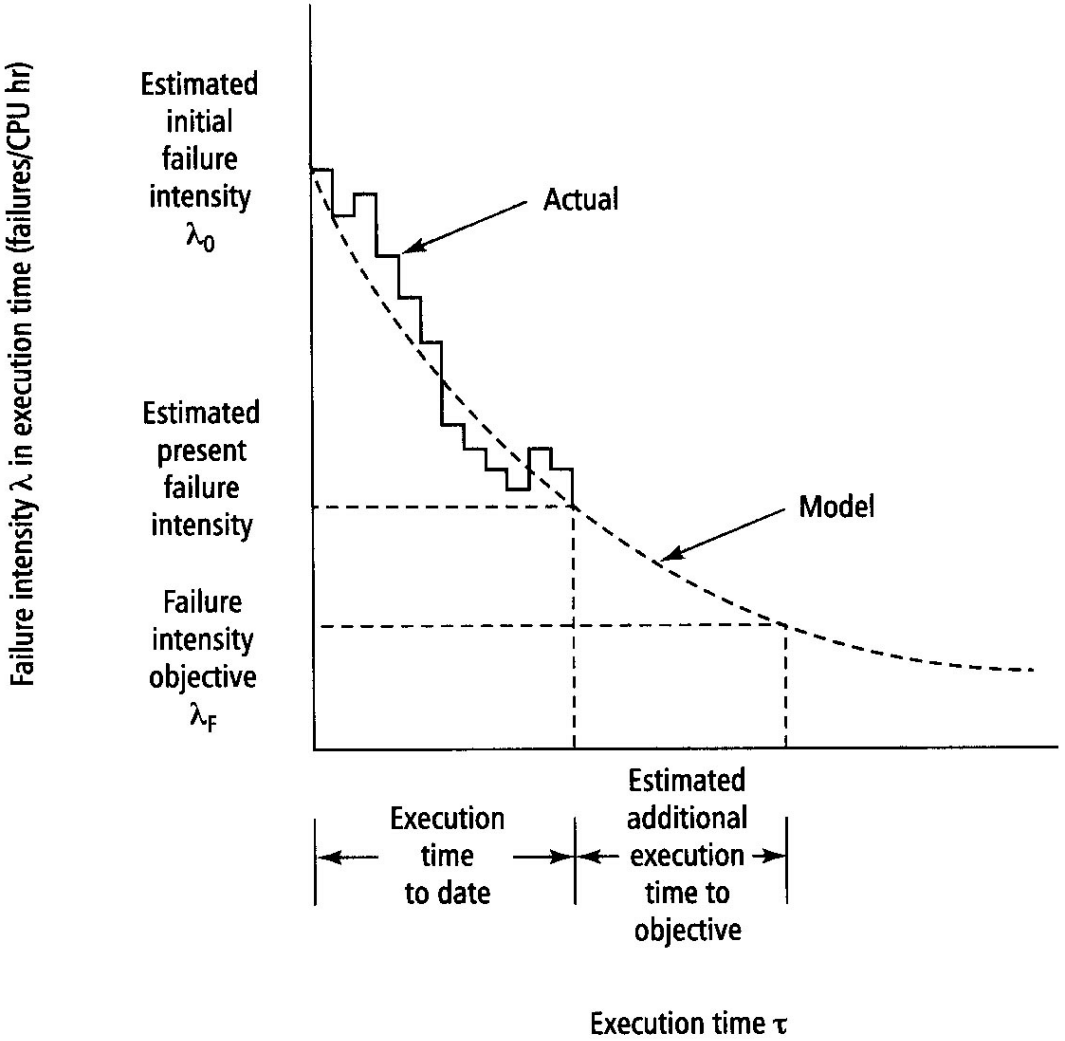
- Probability of 0 failures in a time frame of length τ (reliability):

$$P_0(\tau) = e^{-\lambda\tau}$$

- Additional time/failures to reach, from present failure intensity λ_P , the required failure intensity λ_F: $\Delta\tau$

$$\Delta\tau = \frac{1}{\theta} \times \left(1/\lambda_F - 1/\lambda_P \right) \qquad \Delta\mu = \frac{1}{\theta} = \lambda_P/\lambda_F$$

Conceptual View of Prediction



Time Measurement

- Execution (CPU) time: may be difficult to collect but accurate
- Calendar time: Easy to collect but makes numerous assumptions (equivalent testing intensity with time periods)
- Testing effort: Same as calendar time, but easier to verify the assumptions
- Specific time measurement may often be devised: #Calls, Lines of code compiled, #transactions

Selection of Models

- Several Models exist (ca. 40 published) => Which one to select? Not one model is consistently the best.
- Assumptions of Models:
  - Definition of testing process
  - Finite or infinite number of failures?
  - No faults introduced during debugging?
  - Distribution of data (Poisson, Binomial)?
  - Data requirements? (inter-failure data, failure count data)
- Assessing the goodness-of-fit
  - Kolmogorow-Smirnov, Chi-Square
- Trends in Data (prior to model application)
  - Usually, Reliability Models assume reliability growth. Laplace test can be used to test whether we actually experience growth

Pro's and Con's

Pro's

- Reliability can be specified
- Objective and direct assessment of reliability
- Prediction of time to delivery

Con's

- Usage model may be difficult to devise
- Selection of reliability growth model difficult
- Measurement of time crucial but may be difficult in some contexts
- Not easily applicable to safety-critical systems as very high reliability requirements would take years of testing to verify.