Chap 4. Software Reliability

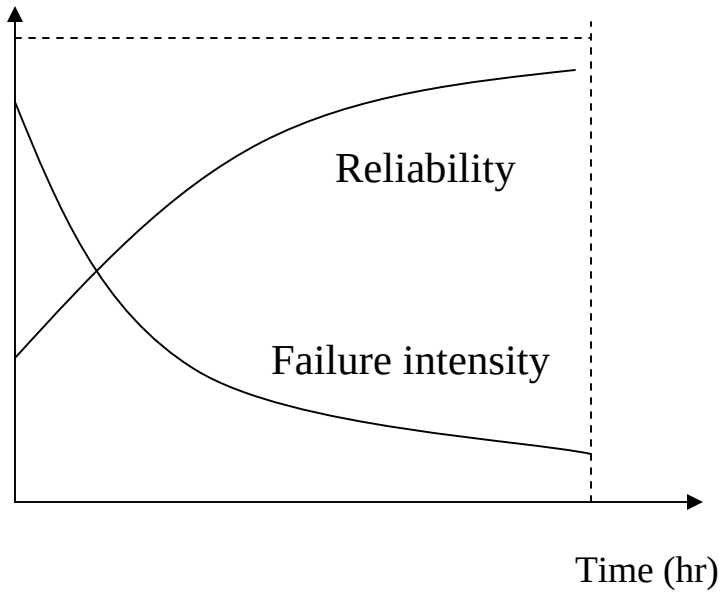
4.2 Reliability Growth

1. Introduction
2. Reliability Growth Models
3. The Basic Execution Model
4. Calendar Time Computation
5. Reliability Demonstration Testing

1. Introduction

-Reliability growth relies on the assumption that faults are removed immediately after being discovered.

- As a result, failure intensity tends to decrease (so it is not constant).



-Many reliability growth models have been proposed in the literature. Examples include the ***basic execution model*** and ***the logarithmic Poisson model***.

-Reliability can be predicted by matching the measured reliability data to known reliability (growth) model.

- This model is then extrapolated to the required level of reliability
- The time that will be required to achieve the reliability target can then be computed from the model; therefore testing and debugging must continue until that time is reached.

-Predicting system reliability from a reliability model allows:

- *Planning of testing*: given the current testing schedule, the time when testing will be completed and required resources can be predicted. This allows conducting proper customer negotiations and deciding when to stop testing.

2. Reliability Growth Models

-The exponential model (see previous chapter) can be regarded as the basic form of software reliability growth models.

- For the past decades, more than a hundred models have been proposed in the research literature.
- Unfortunately few have been tested in practical environments with real data, and even fewer are in use.

-Software reliability growth models can be classified into two major classes depending on the dependent variable.

1. **Time between failures models:**
   - The variable under study is the time between failures
   - It is expected that the failure time will get longer as defects are removed from the software.
   - Assume that the time between, say the (i-1)st and the ith failures follows a distribution whose parameters are related to the number of latent defects remaining in the product after the (i-1)st failure.
   - The distribution used is supposed to reflect the improvement in reliability as defects are detected and removed. Mean time to next failure is the parameter to be estimated for these models.
2. **Fault count models:**
   - The variable is the number of faults or failures (or normalized rate) in a specified time interval.
   - The time can be CPU execution time or calendar time such as hour, week, or month.
   - As defects are detected and removed from the software, it is expected that the observed number of failures per unit time will decrease.
   - The number of remaining defects or failures is the key parameter to be estimated for these models.

Jelinski-Moranda Model

-The Jelinksi-Moranda (JM) model is a time between failures model.

- Assumes N software faults at the start of testing, failures occur purely at random, and all faults contribute equally to cause a failure during testing.
- Assumes also that the fix time is negligible and the fix is perfect. Therefore product's failure rate improves by the same amount at each fix.

-The hazard function (the instantaneous failure rate function) at time t_i between the (i-1)st and the ith failures is given by:

$$Z(t_i) = \Phi \times [N - (i - 1)]$$

-Where N is the number of software defects at the beginning of testing and Φ is a proportionality constant.
-Note that the hazard function is constant between failures but decreases in steps of Φ following the removal of each fault.

Goel-Okumuto Imperfect Debugging Model

-The J-M model assumes perfect debugging; in practice this is not always the case. In the process of fixing a defect, new defects may be injected.

-Goel and Okumoto proposed an imperfect debugging model to overcome the limitation of the assumption.

- In this model, the hazard function is given by:

$$Z(t_i) = \left[N - p(i-1)\right] l$$

-Where N is the number of faults at the start of testing, p is the probability of imperfect debugging, and λ is the failure rate per fault.

Musa-Okumuto Logarithmic Poisson Execution Time Model

-Concerned with modeling the number of failures observed in given testing intervals.

- Consider that the cumulative number of failures observed at time τ, $N(\tau)$, can be modeled as a non-homogeneous Poisson process (NHPP)-as a Poisson process with a time-dependent failure rate.

$$P\{N(t) = y\} = \frac{[m(t)]^y}{y!} e^{-m(t)}, \qquad y = 0, 1, 2, \ldots$$

-Where $m(\tau)$ (mean value function), the expected number of failures observed by time τ :

$$m(t) = \frac{1}{q} \ln(l_0 q t + 1)$$

-Where λ_0 is the initial failure intensity, and θ is the rate of reduction in the normalized failure intensity per failure (also referred to as the failure intensity decay).

- It attempts to consider that later fixes have a smaller effect on software's reliability than earlier ones.
- It is claimed to be superior for highly nonuniform operational user profiles, where some functions are executed much more frequently than others.

Model Selection

-To choose a reliability model, the following steps can be followed:

1. Collect failure data
2. Examine data (density distribution vs. cumulative distribution)
3. Select a model
4. Estimate model parameters
5. Customize model using the estimated parameters
6. Goodness-of-fit test
7. Make reliability predictions

3. The Basic Execution Model

-The basic execution model is the most popular and widely used reliability growth model, mainly because:

- It is practical, simple and easy to understand;
- Its parameters clearly relate to the physical world.
- It can be used for accurate reliability prediction.

-The basic execution model specifies failure behavior initially using execution time. Execution time may later be converted in calendar time.

- The failure behavior is a ***nonhomogeneous Poisson process***, which means the associated probability distribution is a ***Poisson*** process whose characteristics vary in time.
- It is equivalent to the M-O logarithmic Poisson execution time model, with different mean value function.
- The mean value function in this case is based on an exponential distribution.

-Variables involved in the basic execution model:

- Failure intensity (λ): number of failures per time unit.
- Execution time (τ): time since the program is running.
- Mean failures experienced (μ): mean failures experienced in a time interval.

-In the basic execution model, the mean failures experienced m is expressed in terms of the execution time t as

$$m(t) = n_0 \times \left(1 - e^{-\frac{l_0}{n_0}t}\right)$$

Where:

-λ_0 stands for the initial failure intensity at the start of the execution.

-ν_0 stands for the total number of failures occurring over an infinite time period; it corresponds to the expected number of failures to be observed eventually.

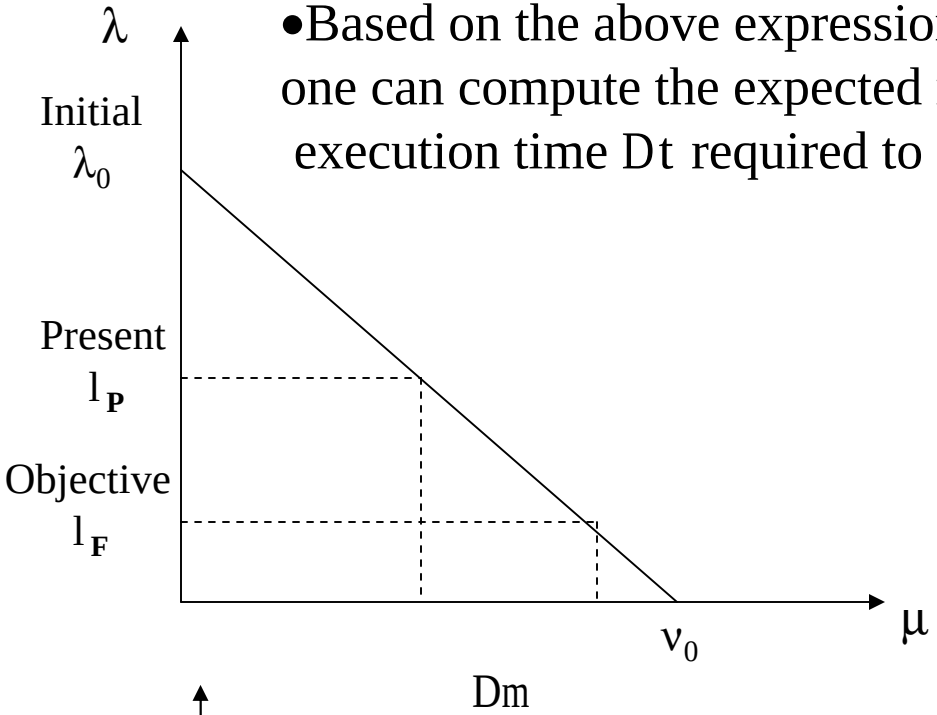
-The failure intensity expressed as a function of the execution time is given by

$$l(t) = l_0 \times e^{-\frac{l_0}{n_0}t}$$

-Based on the above formula, the failure intensity l is expressed in terms of m as:

$$l(m) = l_0 \times \left(1 - \frac{m}{n_0}\right)$$

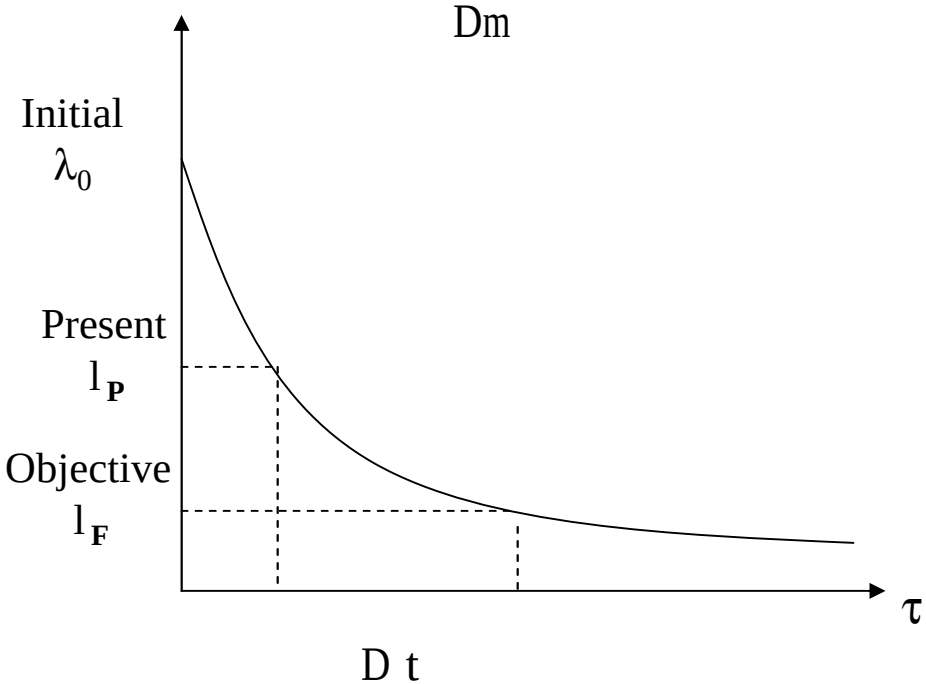
•Based on the above expressions, given some failure intensity objective, one can compute the expected number of failures Dm and the additional execution time Dt required to reach that objective.



$$\Delta m = \frac{n_0}{l_0} \times (l_1 - l_2)$$

Where:

- λ_1 is the current failure intensity
- λ_2 is the failure intensity objective



$$\Delta t = \frac{n_0}{l_0} \times \ln\left(\frac{l_1}{l_2}\right)$$

Example 4.2.3: Assume that a program will experience 100 failures in infinite time. It has now experienced 50 failures. The failure intensity was 10 failures/CPU hr.

1. Calculate the current failure intensity.

2. Calculate the number of failures experienced after 10 and 100 CPU hr of execution.

3. Calculate the failure intensities at 10 and 100 CPU hr of execution.

4. Calculate the expected number of failures that will be experienced and the execution time between a current failure intensity of 3.68 failures/CPU hr and an objective of 0.000454 failure/CPU hr.

4. Calendar Time Computation

-Calendar time is derived from execution time by identifying and considering resource usage constraints and requirements in the project.

- For instance, during test phase the amount of testing is constrained by the rates of ***failure identification (I)*** and ***correction (F)*** by the test personnel, and the ***computer time (C)*** available.

-In general, these quantities are established and known at the beginning of the testing stage and remains fairly stable throughout the process.

- For given execution time, some of these resources become **limiting**, affecting as a result the testing process, and as such determining the pace of the entire process.
- Limiting resources determine the rate at which execution times are used across the calendar time. Specifically they allow computing the ratio between execution time and calendar time.
- Hence, knowledge of the ratio and the execution time can be used to compute the calendar time.

-Resource usage c_r can be expressed as linear function of execution time t and mean failures m:

$$c_r = q_r t + m_r m$$

Where:

- θ_r stands for resource usage per CPU hr.
- μ_r represents the resource usage per failure.

-Since the calendar time is computed based on the contribution of each (limiting) resource, we can compute the resource requirement Dc_{rj} , for each resource j $\hat{I}\{I,F,C\}$.

$$\Delta c_{rj} = q_j \times (\Delta t_j) + m_j \times (\Delta m_j)$$

-Assuming that (available) resource quantities and utilizations are constant during the observation period, the change in resource usage per unit of execution time can be obtained by differentiating resource usage:

$$\frac{dc_r}{dt} = q_r + m_r \frac{dm}{dt} = q_r + m_r l$$

- Let **t** be the calendar time. The ratio of calendar time to execution time can be obtained by dividing the resource usage rate of the limiting resource by the constant quantity of resources available:

$$\frac{dt}{dt} = \frac{1}{P_r r_r} \frac{dc_r}{dt} = \frac{q_r + m_r l}{P_r r_r}$$

Where:

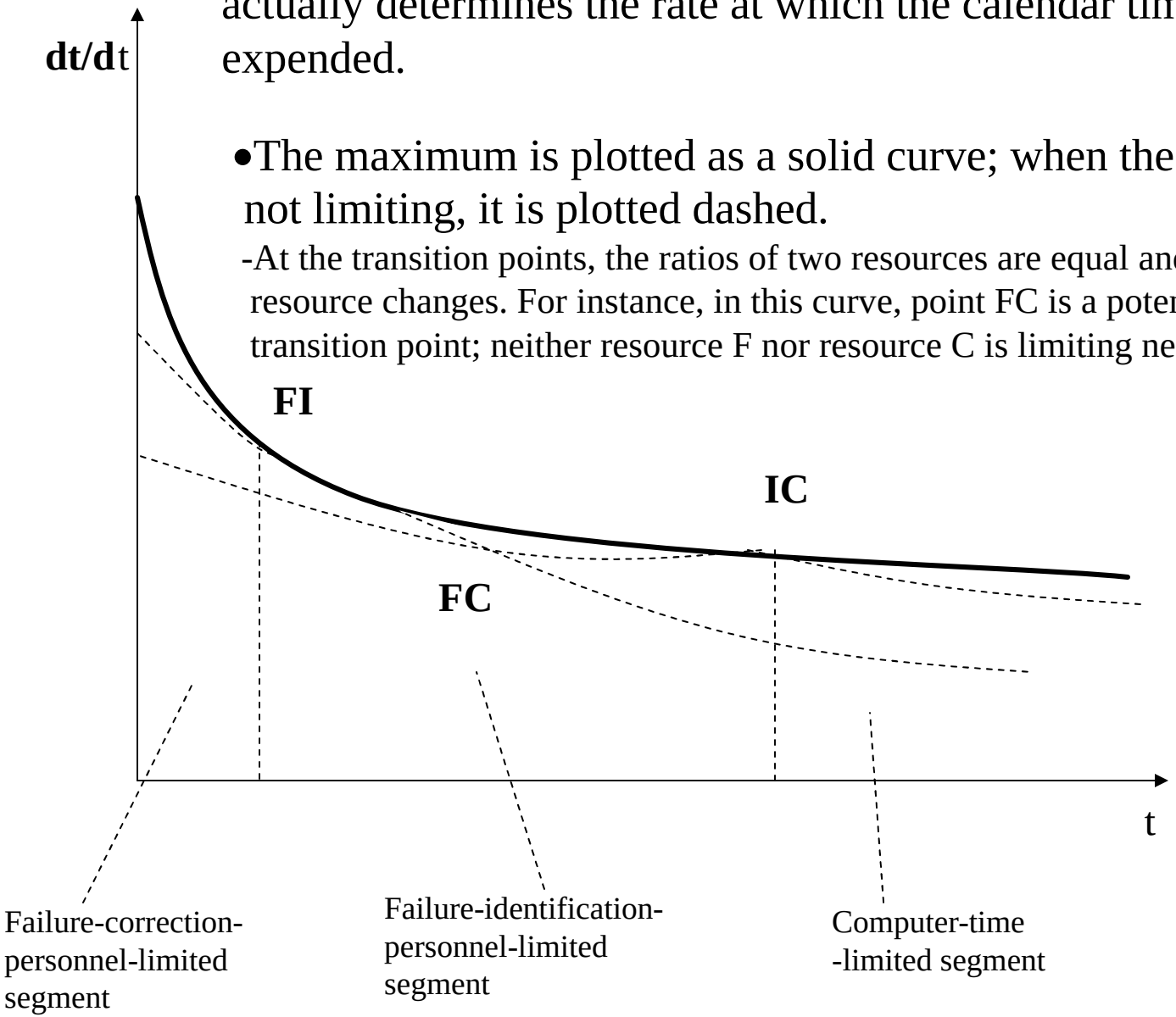
-P_r is the quantity of resources available

-ρ_r is the resource utilization

- The maximum of the ratios for the three limiting resources actually determines the rate at which the calendar time is expended.

- The maximum is plotted as a solid curve; when the resource is not limiting, it is plotted dashed.

-At the transition points, the ratios of two resources are equal and the limiting resource changes. For instance, in this curve, point FC is a potential but not true transition point; neither resource F nor resource C is limiting near this point.



-The **potential** transition points between two resource-limited periods *i* and *j* can be computed by:

$$l_{ij} = \frac{P_j r_j q_i - P_i r_i q_j}{P_i r_i m_j - P_j r_j m_i}$$

Where (i,j) ∈ {(F,C), (F,I), (I,C)}.

-Actual transition points, so actual resource-limited periods, are determined by examining the boundaries and computing the maximum calendar time to execution time ratio for each period.

- The maximum calendar time to execution time ratio is computed as:

$$\left.\frac{dt}{dt}\right|_{max} = \max\left(\frac{q_r + m_r l}{P_r r_r}\right)$$

- The quantity λ is any failure intensity in the range $[\lambda_{j1}, \lambda_{j2}]$, where λ_{j1} and λ_{j2} are the failure intensities at the boundaries of the limiting resource period.

-Calendar time is computed by summing durations of all the resource-limited periods:

$$t = \sum_{j \in \{I, C, F\}} \frac{\Delta c_{rj}}{P_j r_j}$$

Example 4.2.4: Suppose the test team runs test cases for 8 CPU hr and identifies 20 failures. The effort required per hr of execution time is 6 person hr. Each failure requires 2 hr on average to verify and determine its nature. The test team consists of 2 members.

1. Calculate the total failure identification effort required.
2. Assuming that both test team members are fully utilized, (and that the failure identification personnel are the limiting resource) compute the elapsed calendar time.

Example 4.2.5: The resource usage parameters for a software project to build an inventory control system are the following:

Resource	**Resource Usage**	
	Per failure	**Per CPU hr**
Failure Identification personnel (person hr)	3	5
Failure correction personnel (person hr)	8	0
Computer time (CPU hr)	1	1

- The test team has 4 people and there are 12 debuggers. Every one works a normal 8-hr day. The computer operates 16 hr/day, 5 days/week; work hours are organized so that this time can be used; so P_C =2. Computer utilization is 0.75. Failure correction personnel utilization is 0.333.
- Failure identification personnel utilization is 1. The initial and current failure intensity is10 failures/CPU hr. The failure intensity objective is 0.1 failure/CPU hr. The total number of failures is 100.

Compute the calendar time required to reach the objective.

Example 4.2.6: Calculate the resource requirements and cost of system test for the inventory control software project. Assume that:

- The personnel cost (including loading or overhead) is $75/hr
- The computer cost is $500/CPU hr.

Example 4.2.7: Consider a securities portfolio tracking and evaluation program that is installed at 1000 locations. It is considered to have a useful life of 2 years. It operates 250 days/year. On the average, we can expect 2 CPU hr of execution time per day.

- Determine the operational cost of failure for the system, assuming that the failure intensity objective is 2 failures/1000 hr, and that the cost of a failure averages $1000.

5. Reliability Demonstration Testing

-Reliability demonstration testing occurs in situations where it is desirable or necessary to prove that a given level of reliability has been achieved.

-The purpose of a reliability demonstration is to determine if the failure intensity objective (FIO) is met with high confidence or not.

-In conducting a demonstration, the following assumptions are made:

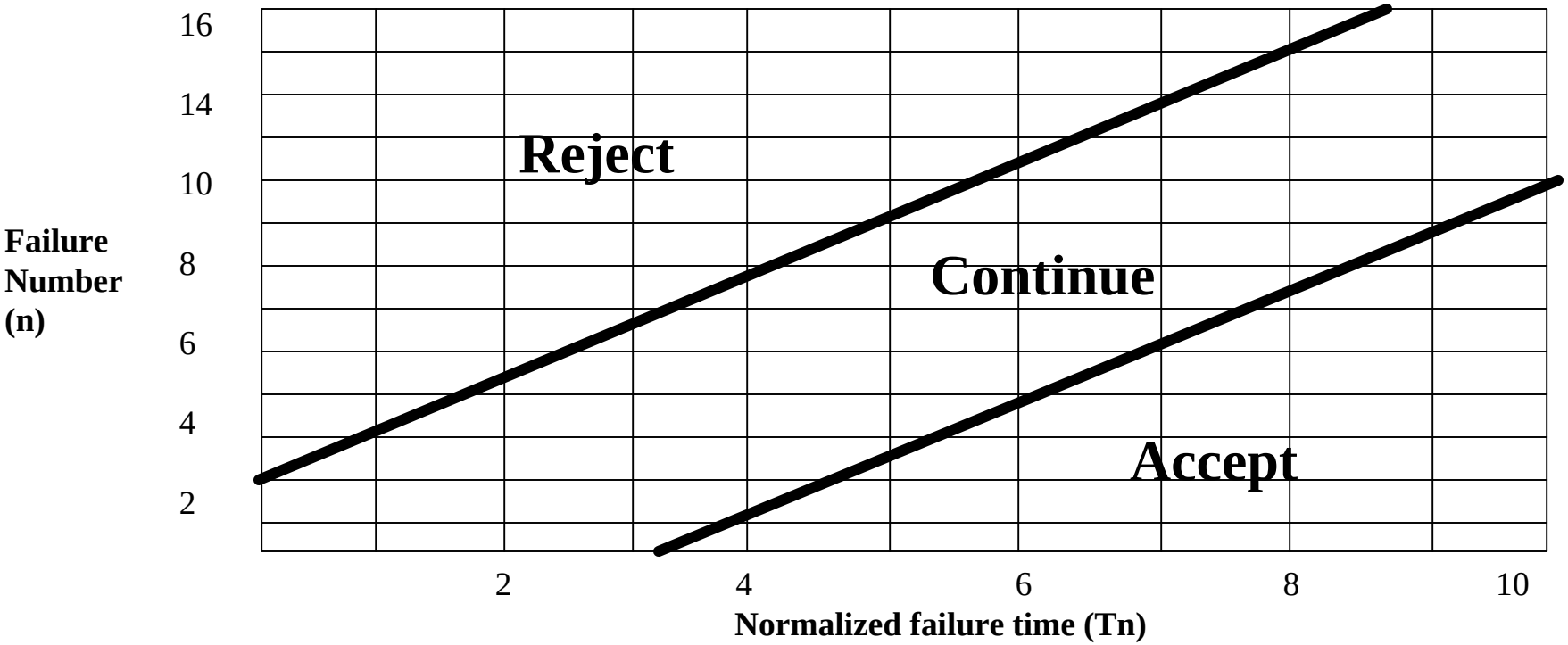
- Proper operational profile has been executed; it must represent actual operation.
- The software is being tested as it has been or would be delivered; no repair will take place.

-The demonstration approach is based on sequential sampling theory.

- Sequential sampling theory provides an efficient approach to testing a hypothesis by taking just enough data and no more.
- The (failure) data points are added one by one, and data collection stops as soon as a decision can be reached.
- The collected data points are used to build a ***reliability demonstration chart***, which serves for decision making.

-A ***reliability demonstration chart*** consists of a graph in which:

- *Vertical axis:* failure number (***n***)
- *Horizontal axis:* normalized failure data (***Tn***), i.e., failure time $\times \lambda_F$



-The steps for conducting a reliability demonstration include:

1. Establish the FIO being tested.
2. Run the test and record the time of the next failure that is experienced.
3. Normalize the failure time by multiplying it by the FIO.
4. Locate the normalized failure time on the reliability demonstration chart.

   - If it falls in the "accept" region, then the FIO is considered with high confidence.
   -If it falls in the "reject" region, there is little possibility of meeting the objective; it is not worth testing further.
   -If it falls in the "continue" region, the result is uncertain,; testing should continue.

-The boundaries between the different regions of the chart (i.e., reject-continue, continue-accept) can be determined by specifying the following parameters:

- ***Discrimination ratio (g):*** Acceptable error in estimating failure intensity.

- ***Customer risk (b) :*** Probability that the developer is willing to accept of falsely saying the failure intensity objective is met (i.e., ***acceptance***) when it is not.

- ***Developer risk (a) :*** Probability that the developer is willing to accept of falsely saying the failure intensity objective is **not** met (i.e., ***rejection***) when it is.

-Example: For a =10% and b= 10% and g =2

- There is 10% risk (b) of wrongly accepting the software when its failure intensity objective is actually equal or greater than twice (g =2) the failure intensity objective.

- There is 10% risk (a) of wrongly rejecting the software when its failure intensity objective is actually equal or less than half (g =2) the failure intensity objective.

-Based on α, β, and γ, the following parameters are computed and used to determine the different regions of the chart:

$$A = \ln \frac{b}{1-a}$$

$$B = \ln \frac{1-b}{a}$$

- ***A*** changes rapidly with *customer risk* but very slightly with *developer risk* and it determines the intercept of ***accept*** boundary with the horizontal line *n=0*
- ***B*** changes rapidly with *developer risk* but very slightly with *customer risk* and it determines the intercept of ***reject*** boundary with the vertical line *Tn=0*

-Boundary between ***accept*** and ***continue*** regions:

$$T_n = \frac{A}{1-g} - \frac{\ln g}{1-g} n$$

-Boundary between ***reject*** and ***continue*** regions:

$$T_n = \frac{B}{1-g} - \frac{\ln g}{1-g} n$$

- When risk levels (α ✔ ● ___ β) decrease, the system will require more test before reaching the accept or reject regions, *i.e.*, the continue region becomes wider.
- When discrimination ratio (γ) decreases, the system will require more test before reaching the accept or reject regions, *i.e.*, the continue region becomes wider.

Example 4.2.8: A personal computer manufacturer wishes to know if the software Company furnishing the operating system for a new machine has met the failure intensity objective of 0.1 failure/CPU hr. A series of tests are applied and the following failure data at different CPU hr are recorded (see table).

Decide whether the outcome of testing based on the following parameters:
α= 10%, β=10%, γ=2

Failure Number	**Failure Time (CPU hr)**
1	8
2	19
3	60

Example 4.2.9: We have developed a program for a Web server with the failure intensity of 1 failure/100,000 transactions. The program runs for 50 hours, handling 10,000 transactions per hour on average with no failures occurring. How confident are we that the program has met its objective? Can we release the software now?

Example 4.2.10: Suppose that a new component is added to the program serially to make a new package. The failure intensity of the new component is 0.5 failures/100,000 transactions. The new package fails after 10 hrs. Can we release the new package now?

Example 4.2.11: A company is planning to purchase several new color laser printers. Before finalizing the purchase, they acquire a similar printer for the test run and conduct certification test on it.

Vendor's data shows that the toner should be changed every 10,000 pages. The goal of the company is to have the system running without any failure between the two consecutive toner changes and in the worst case having only one failure during the same period.

a) What shall be the failure intensity objective for the system?

b) During the test run, it is observed that failures occur at 4,000 pages, 6,000 pages, 10,000 pages, 11,000 pages, 12,000 pages and 15,000 pages of output. Using the reliability demonstration chart, what can we conclude about this printer?