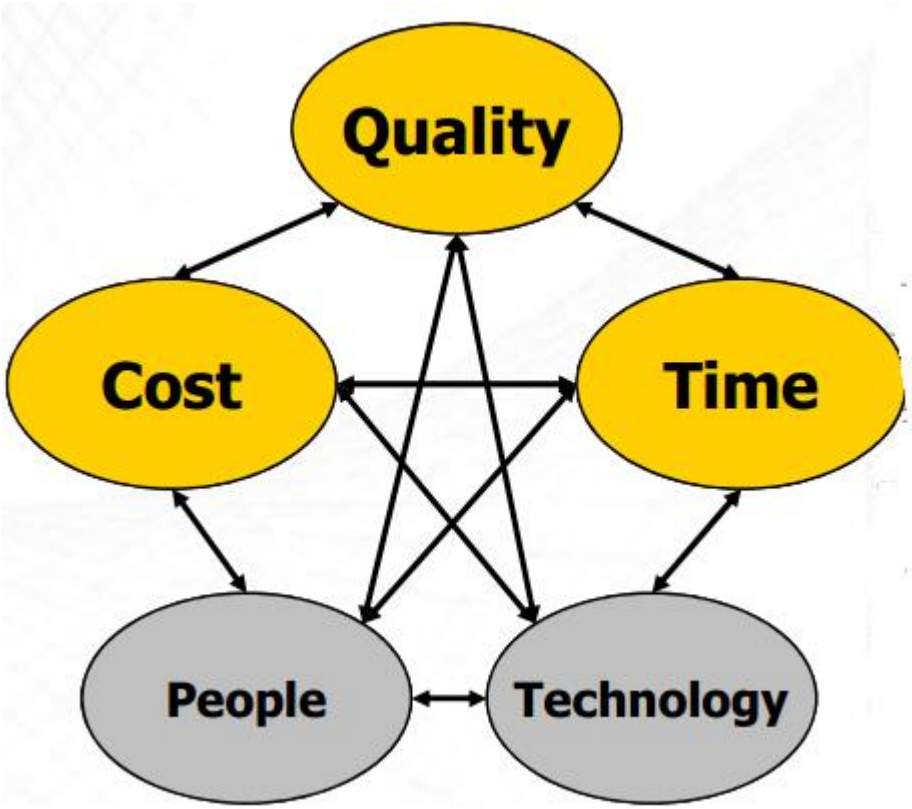
Software Reliability Models & Metrics (Chapter 9)

- Contents
- Reliability concepts and definitions
- Software reliability models and metrics
- Fundamentals of software reliability engineering (SRE)
- Reliability management models

What Affects Software?

- **Time:**
  - Meeting the project deadline.
  - Reaching the market at the right time.
  - Cost:
  - Meeting the anticipated project costs.
  - Quality:



  - Working fine for the designated period on the designated system

What Affects Software Quality?

- Delivery Date:
  - Meeting the project deadline.
  - Reaching the market at the right time.
- Cost:
  - Meeting the anticipated project costs.
- Performance:
  - Working fine for the designated period on the designated system, i.e., reliability, availability, etc.

Failure, Failure Intensity & Availability

- Failure: Any departure of system behavior in execution from user needs or expectations.
- Failure intensity: the number of failures per natural unit or time unit. Failure intensity is way of expressing reliability.
- Availability: The probability at any given time that a system or a capability of a system functions satisfactorily in a specified environment.

$$Availability = \frac{Uptime}{Uptime + Downtine}$$

Error, Fault and Failure

- An error is a human action that results in software containing a fault.
- A fault (bug) is a cause for either a failure of the program or an internal error (e.g., an incorrect state, incorrect timing).
- Among the 3 factors only failure is observable.

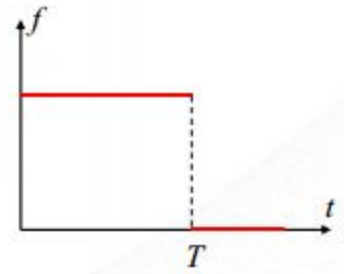
Reliability & Reliability Engineering

- Reliability: The probability that a system or a capability of a system functions without failure for a specified time in a specified environment.
- Reliability Engineering: Engineering of reliability in software products.
- Reliability Engineering's goal: Developing software to reach the market
  - With "minimum" development time
  - With "minimum" development cost
  - With "maximum" reliability

Hardware Reliability Models

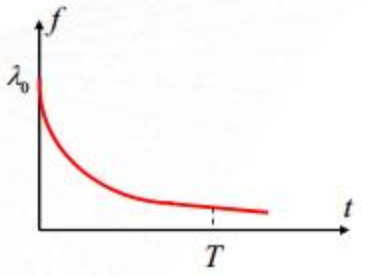
- **Uniform model:**
  - Probability of failure is fixed.

$$f(t) = \begin{cases} 1 & t \le T \\ 0 & t > T \end{cases}$$



- **Exponential model:**
  - Probability of failure changes exponentially over time

$$f(t) = \lambda_0 e^{-\beta t}$$



Software vs. Hardware Reliability

- Software reliability doesn’t decrease with time, i.e., software doesn’t wear out.
- Hardware faults are mainly physical faults, e.g., fatigue. Software faults are mainly design faults which are harder to measure, model, detect and correct.
- Hardware failure can be “fixed” by replacing a faulty component with an identical one, therefore no growth. Software problems can be “fixed” by changing the code in
- order to have the failure not happen again, therefore reliability growth is present.
- Conclusion: hardware reliability models may not be used identically for software

Reliability Questions

Single failure specification:

- What is the probability of failure of a system (or a component)?

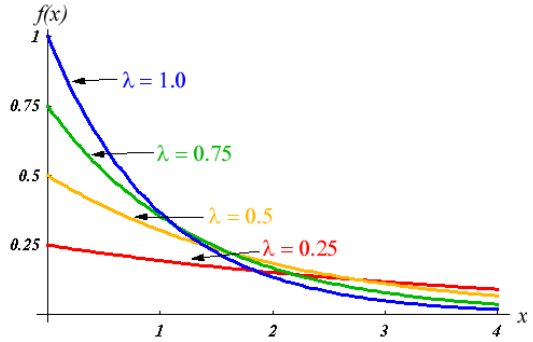
Multiple failure specification:

- If a system (or a component) fails at time t_1, t_2, …, t_{i-1}, what is the probability of its failure at time t_i?

Reliability Metrics (Single Failure)

Reliability Metrics (Single Failure)

- **Probability density function (PDF):** depicting changes of the probability of failure up to a given time *t*.
- A common form of PDF is exponential distribution
- Usually we want to know how



Exponential PDF:

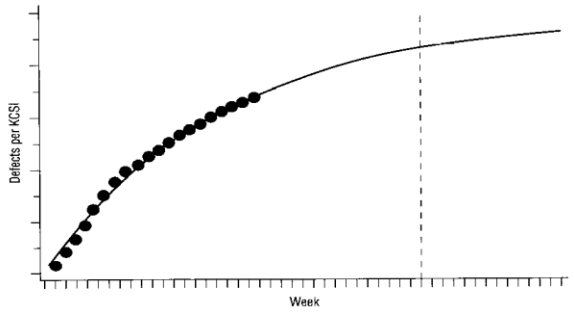
- long a component will behave correctly before it fails, i.e., the probability of failure from time *0* up to a given time *t*.

$$f(t)=\begin{cases}\lambda e^{-\lambda t} & t\geq 0\\ 0 & t<0\end{cases}$$

Reliability Metrics (Single Failure) /2

- Cumulative density function (CDF): depicting cumulative failures up to a given time t.

$$F(t) = \int_0^t f(t)$$



- For exponential distribution, CDF is:

$$F(t) = \begin{cases} 1 - e^{-\lambda t} & t \geq 0 \\ 0 & t < 0 \end{cases}$$

- Reliability function (R): depicting a component functioning without failure until time t.

$$R(t) = 1 - F(t)$$

Reliability Metrics (Single Failure) /3

- For exponential distribution, R is:

$$R(t) = e^{-\lambda t}$$

- What is the expected value of failure time T?
- It is the mean of the probability density function (PDF), named mean time to failure (MTTF)

$$E(T) = \int_0^{\infty} t\, f(t)\, dt$$

- For exponential distribution, MTTF is:

$$MTTF = \frac{1}{\lambda}$$

Reliability Metrics (Single Failure) /4

- Median time to failure (t_m): a point in time that the probability of failure before and after tm are equal.

$$\int_0^{t_m} t\, f(t)\, dt = \frac{1}{2} \quad or \quad F(t_m) = \frac{1}{2}$$

- Failure (Hazard) Rate z(t): Probability density function divided by reliability function.

$$z(t) = \frac{f(t)}{R(t)}$$

- For exponential distribution, z(t) is: λ

- Serial System Reliability: is multiplication of the reliability of its components.

$$R_{system}(t) = \prod_{i=1}^{n} R_i(t)$$

Reliability Metrics (Single Failure) /5

- For exponential distribution:

$$R_{system}(t) = e^{-\lambda_1 t}\, e^{-\lambda_2 t} \cdots e^{-\lambda_n t} = e^{-(\lambda_1+\lambda_2+\cdots+\lambda_n)t}$$

$$R_{system}(t) = e^{-\left(\sum_{i=1}^{n}\lambda_i\right)t}$$

- System Cumulative Failure Rate: is the sum of the failure rate of its components.

$$z_{system}(t) = \sum_{i=1}^{n} z_i(t)$$

- For exponential distribution:

$$z_{system}(t) = \lambda_1 + \lambda_2 + \cdots + \lambda_n = \sum_{i=1}^{n} \lambda_i$$

Software Reliability

Models: Multiple Failure

Specification

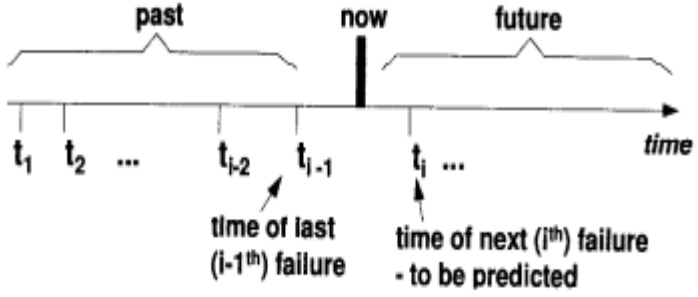
Reliability Questions

Single failure specification:

- What is the probability of failure of a system (or a component)?

Multiple failure specification:

- If a system (or a component) fails at time t1,t2, …, ti-1, what is the probability of its failure at time ti?
- One can assume that the probability of failure (probability density function, PDF) for all failures are the same (e.g., replacing the faulty hardware component with an identical one).
- In software, however, we want to "fix" the problem, i.e., have a lower probability of failure after a repair (or longer Δti = ti-ti-1). Therefore, we have reliability growth (i.e., reliability change over time).

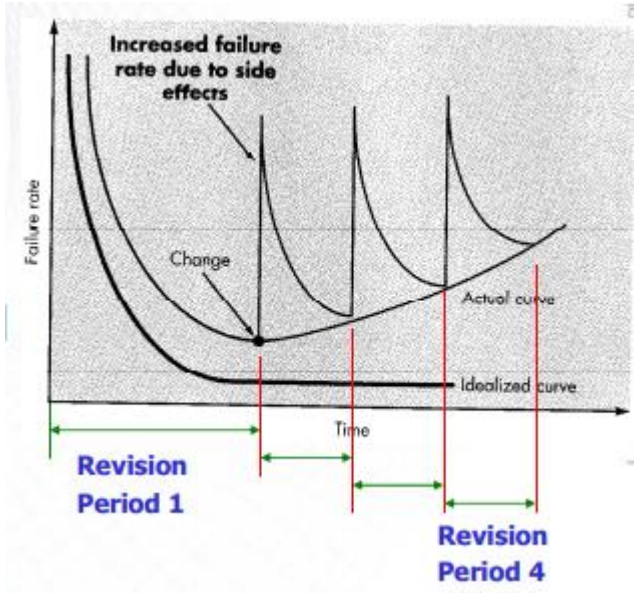


Reliability Growth Models /1

- Common software reliability growth models are:
  - Basic Exponential model
  - Logarithmic Poisson model
- The basic exponential model assumes finite failures ($\nu 0$) in infinite time.
- The logarithmic Poisson model assumes infinite failures.

Validity of the Models

- Software systems are changed (updated) many times during their life cycle.
- The models are good for one revision period rather than the whole life cycle.



Reliability Growth Models /2

- Parameters involved in reliability growth models:

  1) Failure intensity (failure rate) (λ): number of failures per natural unit or time unit.

  2) Execution time (τ): time since the program is running. Execution time may be different from calendar time.

  3) Mean failures experienced (μ): mean failures experienced in a time interval.

Mean failures experienced (μ) for a g ..., 1 hour execution time) is calculated as:

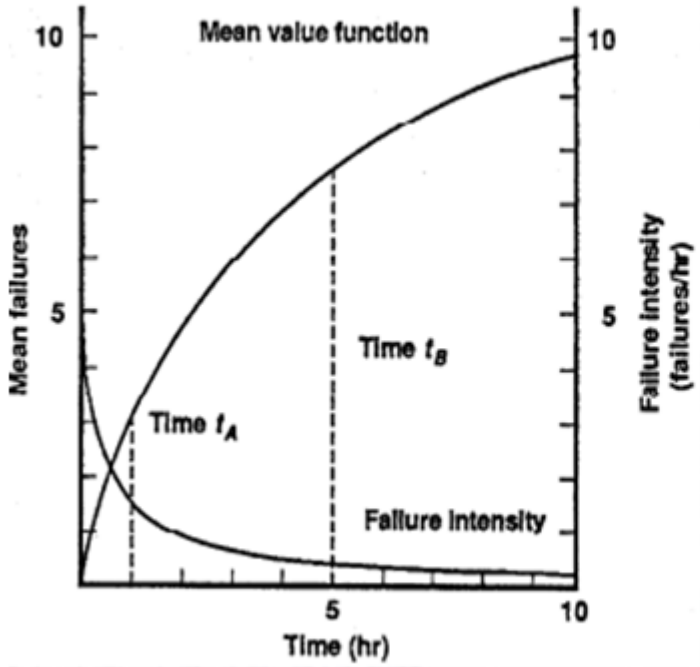
$$\mu = \sum_{i=1}^{n} i \times p_i$$

p_i: probability of occurrence

i : number of failures

No. of failures in interval (n)	Probability (p)	n × p
0	0.10	0
1	0.18	0.18
2	0.22	0.44
3	0.16	0.48
4	0.11	0.44
5	0.08	0.40
6	0.05	0.30
7	0.04	0.28
8	0.03	0.24
9	0.02	0.18
10	0.01	0.10
Mean failure (μ)		3.04

Reliability Growth Models /3



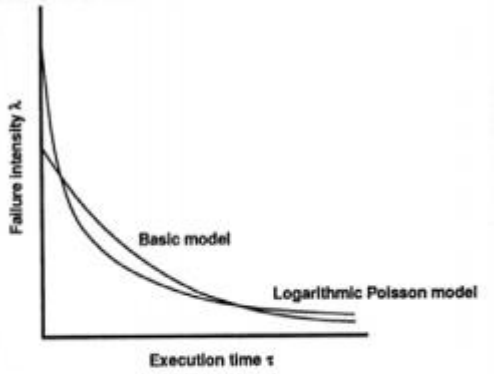
Failure(s) in time period	Probability	
	Elapsed time (1 hour)	Elapsed time (5 hours)
0	0.10	0.01
1	0.18	0.02
2	0.22	0.03
3	0.16	0.04
4	0.11	0.05
5	0.08	0.07
6	0.05	0.09
7	0.04	0.12
8	0.03	0.16
9	0.02	0.13
10	0.01	0.10
11	0	0.07
12	0	0.05
13	0	0.03
14	0	0.02
15	0	0.01
Mean	3.04	7.77

Reliability Growth Models /4

- Failure intensity (λ) versus execution time (τ)

(B) $\lambda(\tau) = \lambda_0 e^{\left(-\frac{\lambda_0}{\nu_0}\right)\tau}$

(P) $\lambda(\tau) = \frac{\lambda_0}{\lambda_0 \theta \tau + 1}$



λ_0 Initial failure intensity

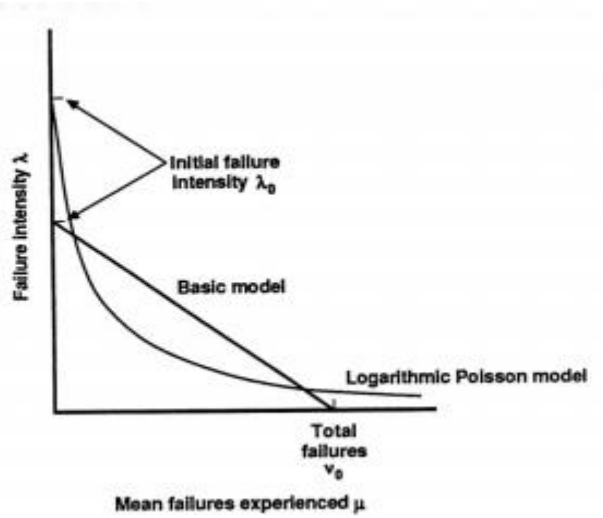
ν_0 Total failures

θ Decay parameter

- Failure intensity (λ) versus mean failures experienced (μ)

(B) $\lambda(\mu) = \lambda_0\left(1 - \frac{\mu}{\nu_0}\right)$

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

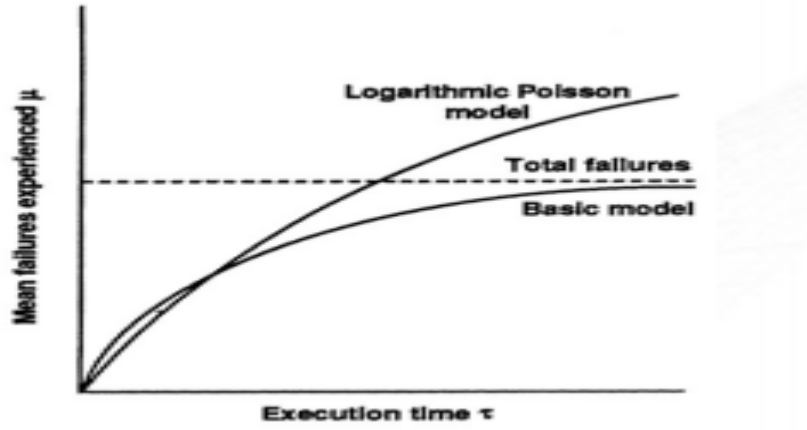


Reliability Growth Models /5

- Mean failures experienced (λ) versus execution time (τ)

(B) $$\mu(\tau) = \nu_0 \left[1 - e^{-\left(\frac{\lambda_0}{\nu_0}\tau\right)} \right]$$

(P) $$\mu(\tau) = \left(\frac{1}{\theta}\right) \ln(\lambda_0 \theta \tau - 1)$$



Failure Specification

- Failure Specification

  1) Time of failure

  2) Time interval between failures

  3) Cumulative failure up to a given time

  4) Failures experienced in a time interval

Time based failure specification

Failure no.	Failure times (hours)	Failure interval (hours)
1	10	10
2	19	9
3	32	13
4	43	11
5	58	15
6	70	12
7	88	18
8	103	15
9	125	22
10	150	25
11	169	19
12	199	30
13	231	32
14	256	25
15	296	40

Failure based failure specification

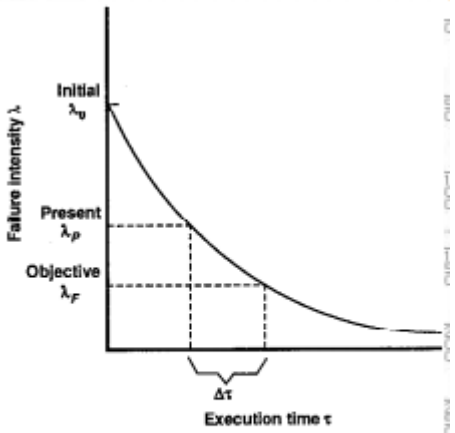
Time(s)	Cumulative Failures	Failures in interval
30	2	2
60	5	3
90	7	2
120	8	1
150	10	2
180	11	1
210	12	1
240	13	1
270	14	1

How to Use the Models

- Release criteria: time required to test the system to reach a target failure intensity:

$$(B)\quad \Delta\tau = \frac{\nu_0}{\lambda_0}\ln\frac{\lambda_P}{\lambda_F}$$

$$(P)\quad \Delta\tau = \frac{1}{\theta}\left(\frac{1}{\lambda_F}-\frac{1}{\lambda_P}\right)$$



λ_P : Present failure intensity

λ_F : Target failure intensity

Reliability Metrics

- Mean time to failure (MTTF): Usually calculated by dividing the total operating time of the units tested by the total number of failures encountered (assuming that the failure rate is constant).
- Example:
  - MTTF for Windows 2000 Professional is 2893 hours or 72 fortyhour workweeks.
  - MTTF for Windows NT Workstation is 919 hours or 23 workweeks.
  - MTTF for Windows 98 is 216 hours or 5 workweeks.
- Mean time to repair (MTTR): mean time to repair a (software) component.
- Mean time between failures (MTBF):

$$MTBF = MTTF + MTTR$$

Reliability Metrics: Availability

- Software System Availability (A):

$$A(t) = \frac{1}{1 + t_m \lambda(t)} \quad or \quad \lambda(t) = \frac{1 - A(t)}{t_m A(t)}$$

λ is failure intensity

t_m is downtime per failure

- Another definition of availability:

$$A = \frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF}$$

- Example: If a product must be available 99% of time and downtime is 6 min, then λ is about 0.1 failure per hour (1 failure per 10 hours) and MTTF=594 min.

Reliability Metrics: Reliability

- Software System Reliability (R):

$$\lambda(t) = \frac{-\ln R(t)}{t} \quad or \quad \lambda \approx \frac{1 - R(t)}{t} \quad for \quad R(t) \geq 0.95$$

λ is failure intensity

R is reliability

t is natural unit (time, etc.)

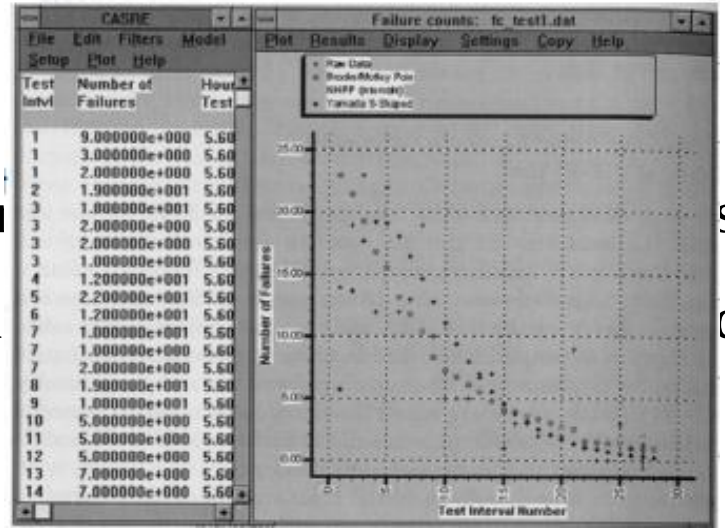
- Example: for λ =0.001 or 1 failure for 1000 hours, reliability (R) is around 0.992 for 8 hours of operation.

Using Reliability Models

- Suppose that we have developed and tested our software and documented the failures for a certain period of time.
- How do we know that the reliability software is getting better after bug fixes?
- How to measure the increase/decrease of reliability?

SRE Tools: CASRE /1

- Computer-Aided Software Reliability Estimation (CASRE) Tool.
- CASRE is a PC-based tool that was developed in 1993 by the Jet Propulsion Laboratories.
- CASRE requires the WINDOWS operating environment.
- Once the data is entered, CASRE automatically provides the analyst with a raw data plot.
- CASRE provides the analyst with the ability to convert from time-domain data to interval-domain data and vice versa.
- CASRE provides operations to transform o ser can select and/or define multiple models for application to th dictions based on the best model.



Using CASRE

1. Prepare input data
   - Input data can be either failure count or failure per interval data
   - Sample failure count data

<failure number> <number of natural or time units since previous failure> <severity class>

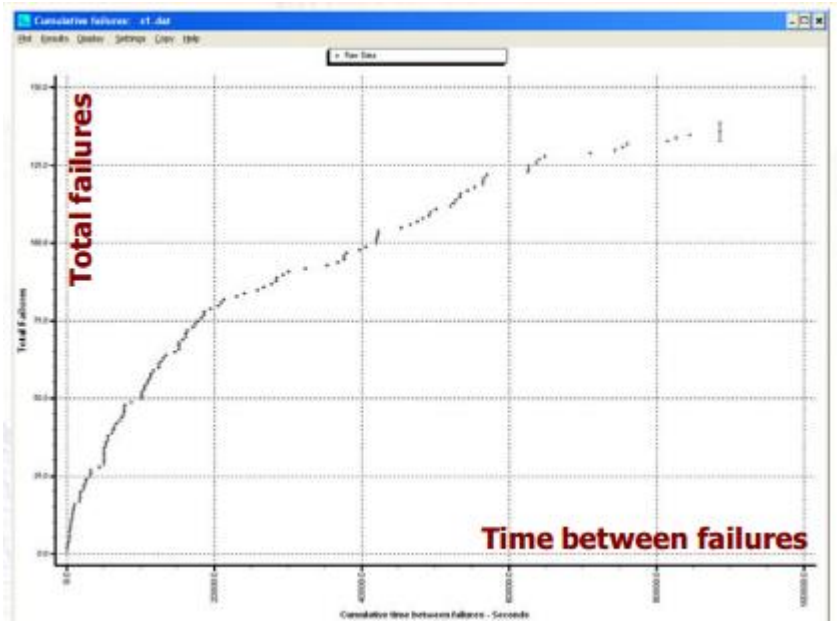
1	30	1
2	55	1
3	70	1
4	60	1
5	90	1
6	110	1
7	100	1
8	150	1
9	120	1
10	215	1

2. Read input file
3. Normalize data by multiplying by failure intensity objective (λ_F)
4. Select data range
5. Select and run model(s)
6. View and interpret results

Case Study

- Project X is a web based application for accessing a database using a browser.
- This version of the software is a minor release with changes to the GUI display and data access engine.
- Two programmers were assigned to the project. One programmer worked on the GUI, and the other on the data access engine.
- The project took approximately 4 weeks to complete.
- A single tester was assigned to the project.
- The test phase was completed in approximately 25 hours (3 working days or 90,000 seconds).
- 136 failures were discovered during the testing.
- Using the dates and times recorded for the failures discovered during testing, a “time between failures” input file was generated for CASRE.
- The severity of all the failures was set to
  - 1 - Low Severity

Time Between Failures Plot



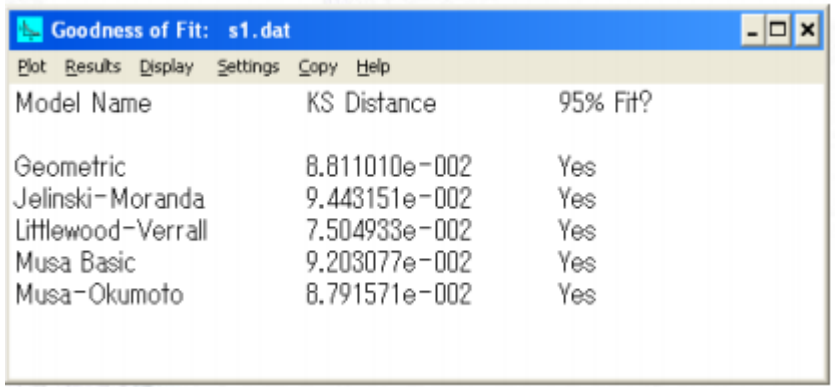
Project Results

- In order to determine which models would provide the best fit for the project data, the following models were run
  - Geometric
  - Jelinski – Moranda
  - Littlewood – Verrall
  - Musa Basic
  - Musa - Okumoto

Goodness of Fit Test

- On Graphic display window select:

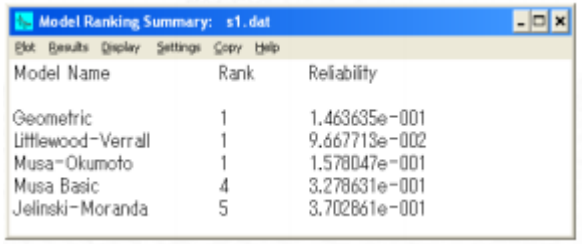
  Display → Goodness of fit

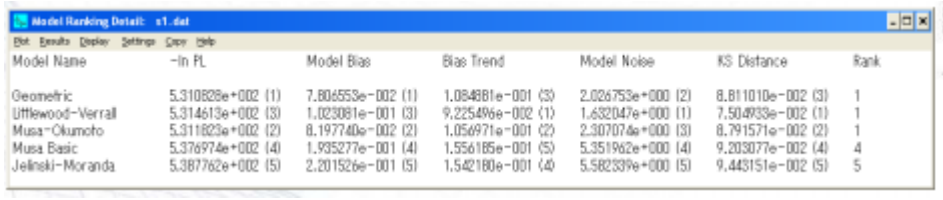


CASRE Model Ranking

On Graphic display window select:

Display → Model rankings → Rank summary or Rank details

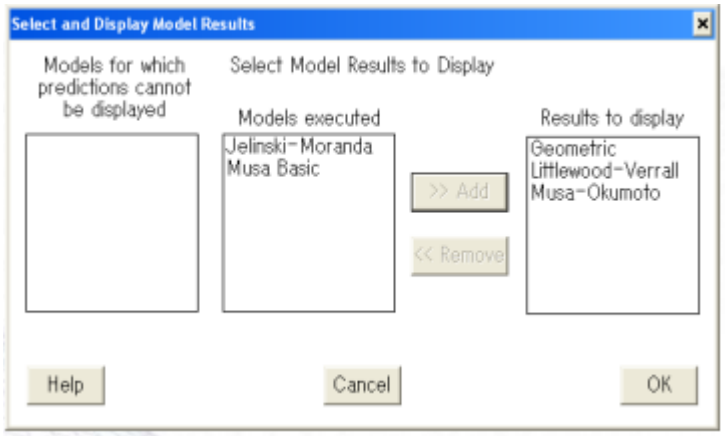




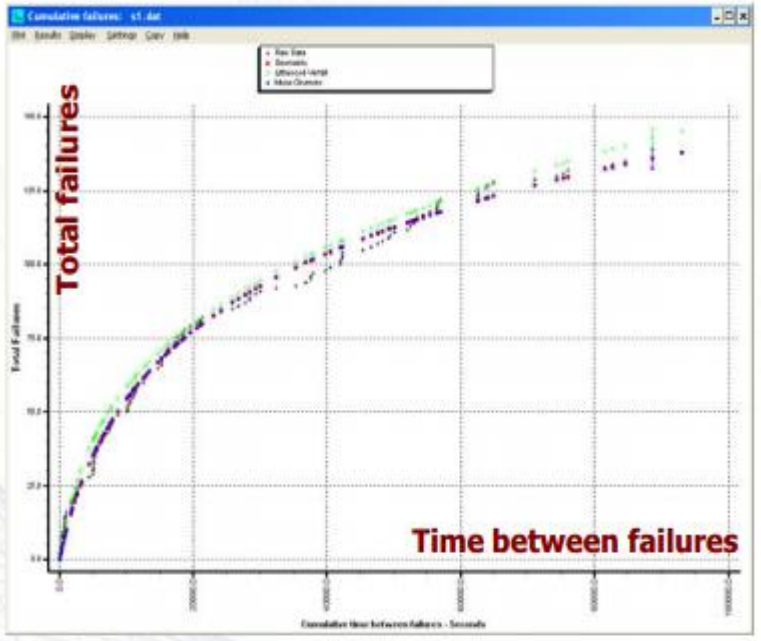
Display Results

On Graphic display window select:

Results → Select model results Only 3 graphs can be displayed at a time

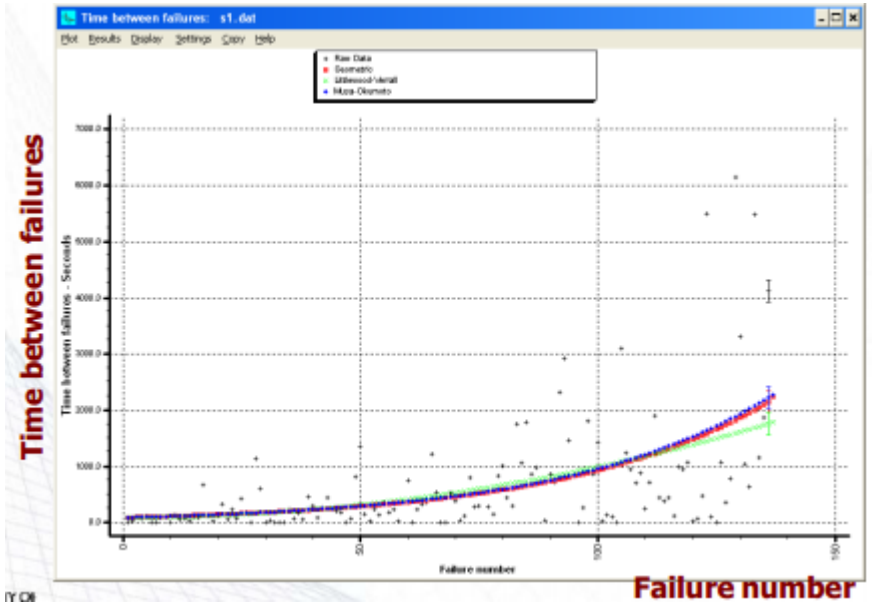


Display: Cumulative Failures

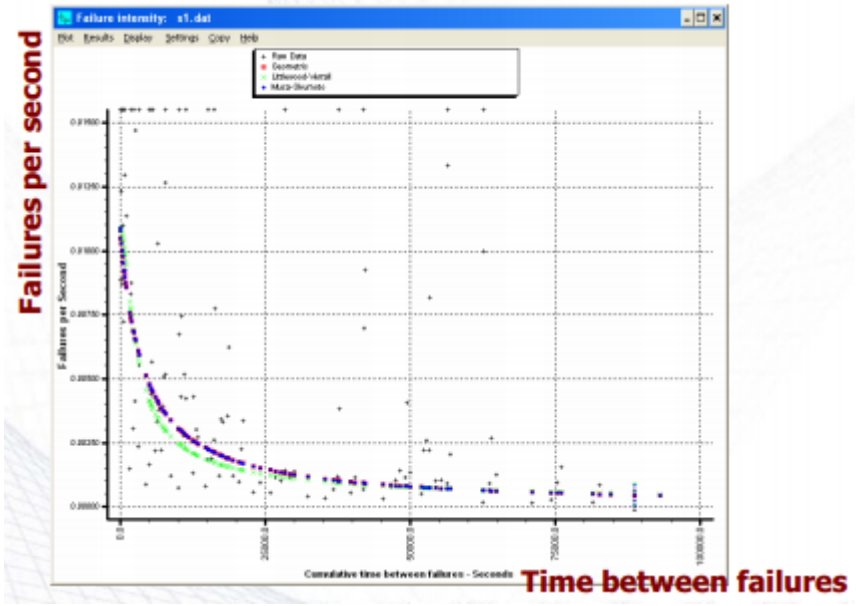


Display: Time Between Failures

Display: Time Between Failures

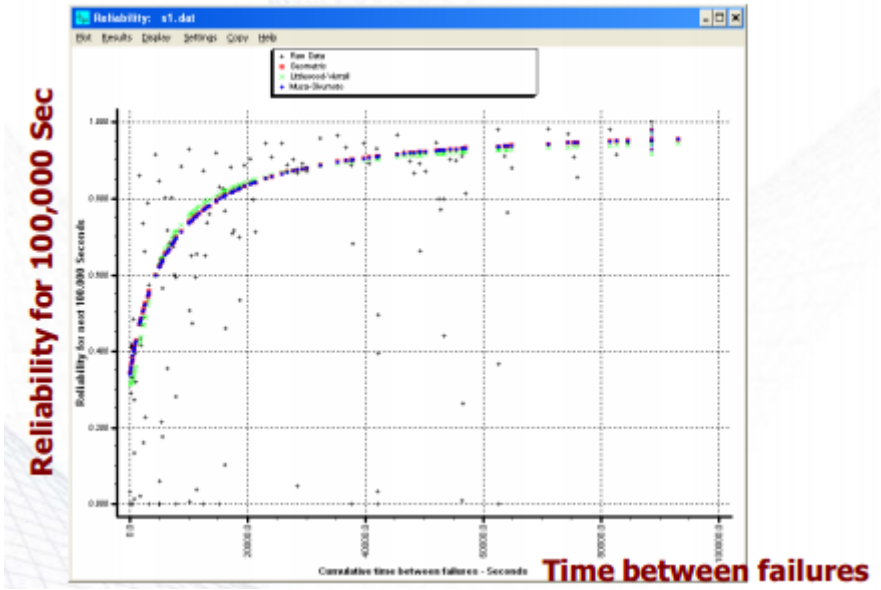


Display: Failure Intensity



Display: Reliability

Display: Reliability



Interpreting Results /1

- Accuracy of estimation of the $\lambda/\lambda F$ ratio depends on the number of failures experienced (i.e., the sample size).
- Good results in estimating failure intensity are generally experienced for programs with 5,000 or more developed source lines and at least 20 failure data points.

Interpreting Results /2

- When the λ/λ_F ratio is very large and the trend indicates little chance of achieving the λF by the scheduled release date, consider taking one or more of the following actions:
  - Adding additional test and debugging resources
  - Adjusting the balance among the objectives for failure intensity, development time, and development cost
  - Deferring features

SRE: Process

- There are 5 steps in SRE process (for each system to test):

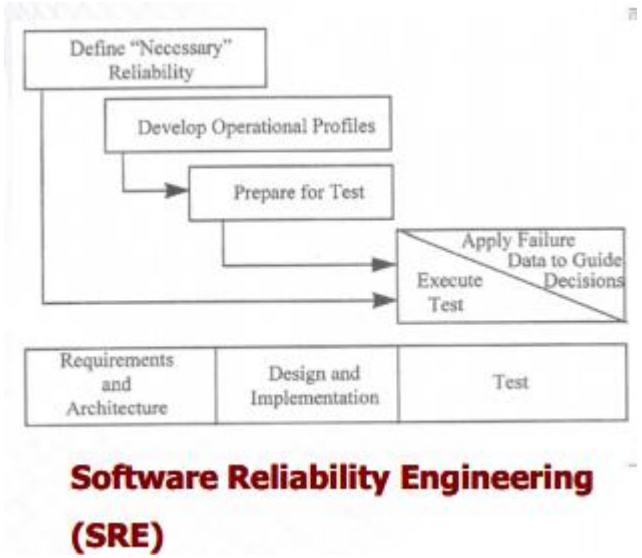
  1) Define necessary reliability

  2) Develop operational profiles

  3) Prepare for test

  4) Execute test

  5) Apply failure data to guide decisions



1. Failure Intensity Objective (FIO)

- Failure intensity (λ) is defined as failure per natural units (or time), e.g.
  - 3 alarms per 100 hours of operation.
  - 5 failures per 1000 print jobs, etc.
- Failure intensity of a system is the sum of failure intensities for all of the components of the system.
- For exponential distribution:

$$z_{system}(t) = \lambda_1 + \lambda_2 + \cdots + \lambda_n = \sum_{i=1}^{n} \lambda_i$$

How to Set FIO?

- Setting FIO in terms of system reliability (R) or availability (A):

$$\lambda = \frac{-\ln R}{t} \quad or \quad \lambda \approx \frac{(1-R)}{t} \quad for \quad R \geq 0.95$$

$$\lambda = \frac{1-A}{t_m A}$$

λ is failure intensity

R is reliability

t is natural unit (time, etc.)

t_m is downtime per failure

Operational Mode / Profile

- Operational mode is a distinct pattern of system use and/or set of environmental conditions that may need separate testing due to likelihood of stimulating different failures.
- Example:
  - Time (time of year, day of week, time of day)
  - Different user types (customer or user)
  - Users experiences (novice or expert)
- The same operation may appear in different operational mode with different probabilities.
- System operational profile must be developed for all of its important operational modes.

Operational Profile

- Operational Profile is the set of operations (operation names and frequencies) and their probabilities of occurrences.
- There are four principal steps in developing an operational profile:
  - Identify the operation initiators (i.e., user types, external systems, and the system itself)
  - List the operations invoked by each initiator
  - Determine the occurrence rates
  - Determine the occurrence probabilities by dividing the occurrence rates by the total occurrence rate

SRE: 3. Prepare for Test

- The Prepare for Test activity uses the operational profiles to prepare test cases and test procedures.
- Test cases are allocated in accordance with the operational profile.
- A Test Case is a partial specification of a run through the naming of its direct input variables and their values.
- Test case is specified with its direct input variables.

Types of Test

- Certification Test: Accept or reject (binary decision) an acquired component for a given target failure intensity.
- Feature Test: A single execution of an operation with interaction between operations minimized. Sometimes called Unit test.
- Load Test: Testing with field use data and accounting for interactions.
- Regression Test: Feature tests after every build involving significant change, i.e., check whether a bug fix worked.

4. Execute Test

- Allocate test time among the associated systems and types of test (feature, load, regression, etc.).
- Invoke the test cases at random times, choosing operations randomly in accordance with the operational profile.
- Identify failures, along with when they occur.
- Use this information to decide whether release the software or not.

Release Criteria

Consider releasing the product when:

1. All acquired components pass certification test

2. Test terminated satisfactorily for all the product variations and components with the λ/λ_F ratios for these variations don't appreciably exceed 0.5 (Confidence factor)

Software Reliability Metrics: Reliability Management Models and Trend Models

The Goal

- It is important to be able to assess the status of quality of a software product, predict the number of failures or estimate the mean time to the next failure during the development.
- Such tasks are the target of the quality and reliability management models.

Trend Analysis: Questions …

- Is the system reliability increasing, decreasing or stable?
- Which reliability growth model fits best the gathered failure data?
- Can the same reliability model be used in all the cases of reliability growth, decrease and stable?

And The Answers …

- Reliability trends can be analyzed by “trend tests”.
- Trend tests can be used to help determine whether the system undergoes reliability growth, decrease or stable reliability.
- Trend analysis also helps select appropriate reliability model for each phase.

Failure Data for Trend Analysis

- The trend tests work with two forms of failure data.
  - Inter-failure times (failure in intervals)
  - Number of failures per unit of time (failure intensity)
- The trend can be analyzed using inter-failure times data or failure intensity data.

Types of Trend Tests

- Two types of trend tests:

Graphical tests and Analytical tests

  - Graphical tests consist of plotting some observed failure data such as the inter-failure times or the number of failures per unit of time versus time in order to visually obtain the trend displayed by the data.
  - Analytical tests are based on statistical analysis of the raw failure data to derive trend factors.

- Two trend tests are commonly carried:
  - Arithmetical mean test
  - Laplace tests

1. Inter-failure Times Data /1

- The arithmetical mean of the inter-failure times (failure in intervals) consists of calculating arithmetical mean τ(i) of the observed inter-failure times θj.

$$\tau(i)=\frac{1}{i}\sum_{j=1}^{i}\theta_j$$

- An increasing series of τ(i) indicates reliability growth and a decreasing series suggests decrease of reliability
- For N(T) as the cumulative number of failures over the time period [0, T], the Laplace factor u(T) is derived:

$$u(i)=\frac{\frac{1}{i-1}\sum_{n=1}^{i-1}\sum_{j=1}^{n}\theta_j-\frac{\sum_{j=1}^{i}\theta_j}{2}}{\sum_{j=1}^{i}\theta_j\sqrt{\frac{1}{12(i-1)}}}$$

- For the case that T is equal to the time of occurrence of failure i.

Inter-failure Times Data /2

- Negative values of the Laplace factor u(i) indicate a decreasing failure intensity, i.e., reliability growth.
- Positive values of the Laplace factor u(i) indicate an increasing failure intensity, i.e., reliability decrease.
- Values between –2 and +2 indicate stable reliability.

Inter-failure Times Data /3

- Midpoint of the observation interval: T/2
- Statistical center of data: $\frac{1}{N(T)\sum_{n=1}^{N(T)}\sum_{j=1}^{n}\theta_j}$
- For the failure intensity decrease (i.e., reliability growth), the inter failure times θ_j tend to occur before the midpoint; hence the statistical center tends to be smaller than the mid-interval.
- For the failure intensity increase, the statistical center tends to be larger than the mid-interval.

2. Failure Intensity Data /1

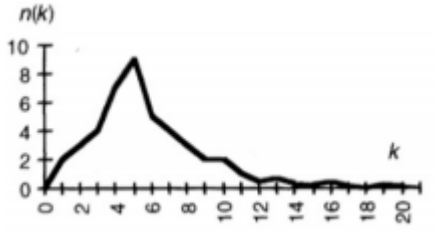
- For the time period [0, T], divided into k units of equal length and for n(i) be the number of failures observed during the time interval i, the Laplace factor u(k) is derived by:

$$u(k)=\frac{\sum_{i=1}^{k}(i-1)n(i)-\left(\frac{k-1}{2}\right)\sum_{i=1}^{k}n(i)}{\sqrt{\frac{k^2-1}{12}\sum_{i=1}^{k}n(i)}}$$

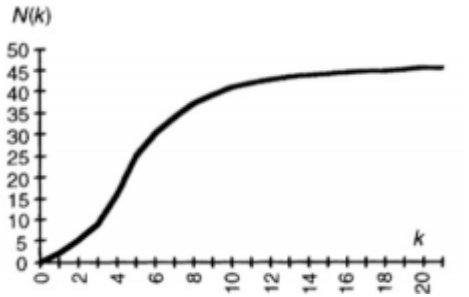
- Negative values of the Laplace factor u(k) indicate a decreasing failure intensity, i.e., reliability growth.
- Positive values of the Laplace factor u(k) indicate an increasing failure intensity, i.e., decrease of reliability.

Typical Plots /1

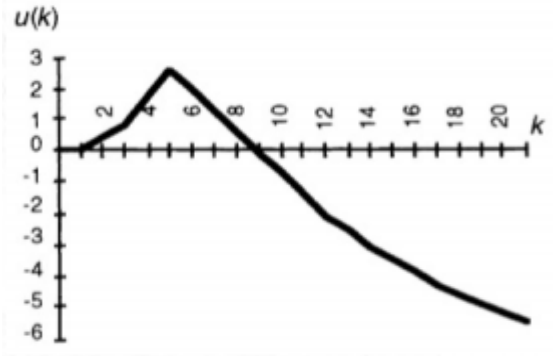
- Typical graphs for failure intensity n(k) and



- cumulative failure intensity N(k)



- Typical plot for the Laplace factor u(k)



Typical Plots /2

- Typical plot for Laplace factor during various project phases

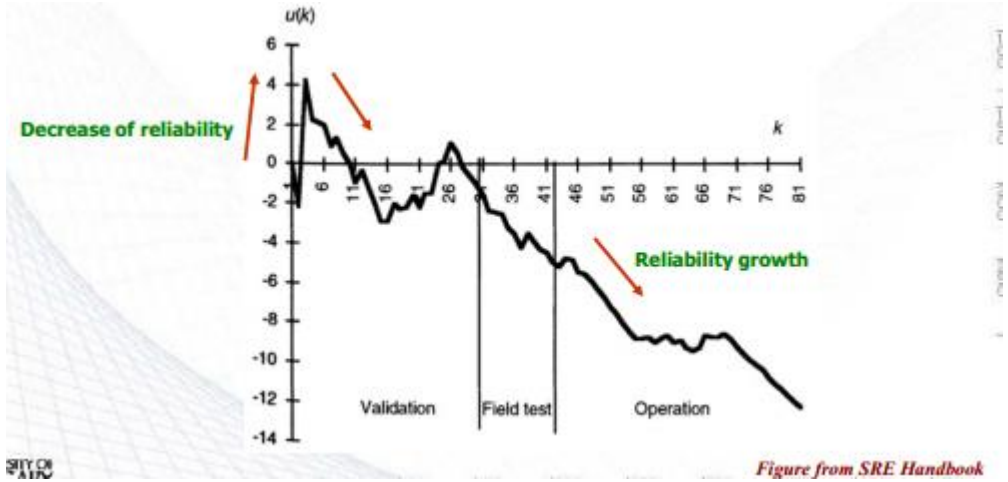


Figure from SRE Handbook

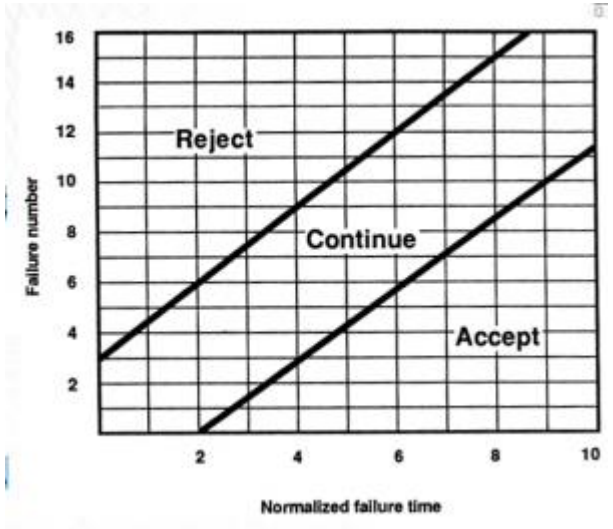
Decisions Based on Testing

- What will be the outcome of certification testing?
  - Accept/reject an acquired component
- What will be the outcome of feature, regression and load testing?
  - Guiding software development process
  - Releasing the product

Certification Test Using Reliability Demonstration Chart

- A way of checking whether the Failure Intensity Objective (λ_F) is met or not.
- It is based on collecting failure data at time points.
- Vertical axis: failure number
- Horizontal axis:

  normalized failure data, i.e., failure time × λ_F



How to Make a Reliability Demo Chart?

- Boundary between reject and continue region:

$$\frac{\ln\frac{\beta}{1-\alpha} - n\ln\gamma}{1-\gamma}$$

- Boundary between accept and continue region:

$$\frac{\ln\frac{1-\beta}{\alpha} - n\ln\gamma}{1-\gamma}$$

Parameters Involved /1

- Discrimination ratio (γ):Acceptable error in estimating failure intensity.
- Consumer risk (β) :) : Probability that the developer is willing to accept of falsely saying the failure intensity objective is met when it is not.
- Supplier risk (α) :) : Probability that the developer is willing to accept of falsely saying the failure intensity objective is not met when it is.

For α =10% and β= 10% and γ =2

- There is 10% risk of wrongly accepting the software when its failure intensity objective is actually equal or greater than twice the failure intensity objective.
- There is 10% risk of wrongly rejecting the software when its failure intensity objective is actually equal or less than half the failure intensity objective.

Example /1

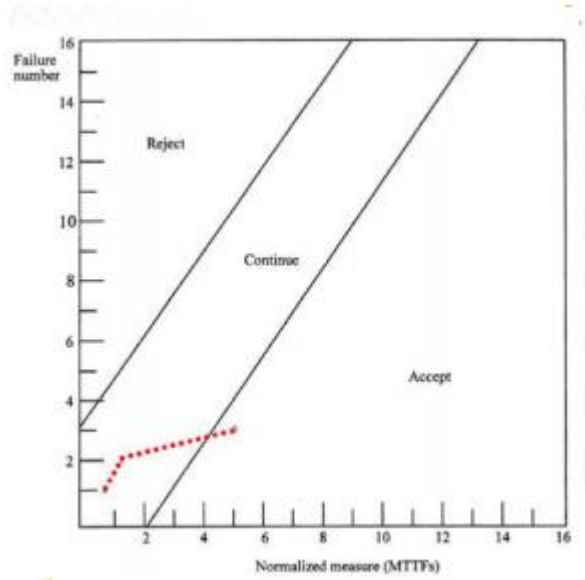
Failure no.	Measure (million transactions)	Normalized Measure (MTTF)
1	0.1875	0.75
2	0.3125	1.25
3	1.25	5

$\lambda_F = 4$ failures/million transactions

$\alpha = 10\%$

$\beta = 10\%$

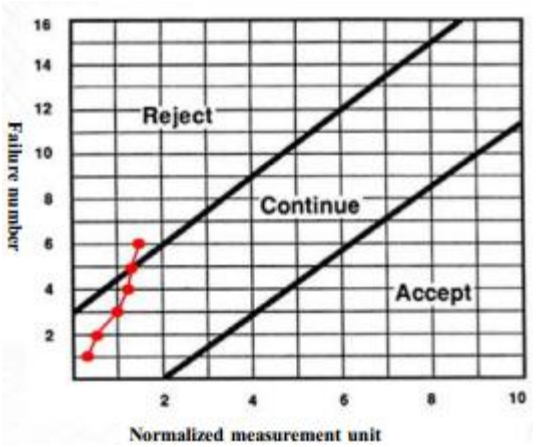
$\gamma = 2$



Entering Accept region, therefore Certification test passed

Example /2

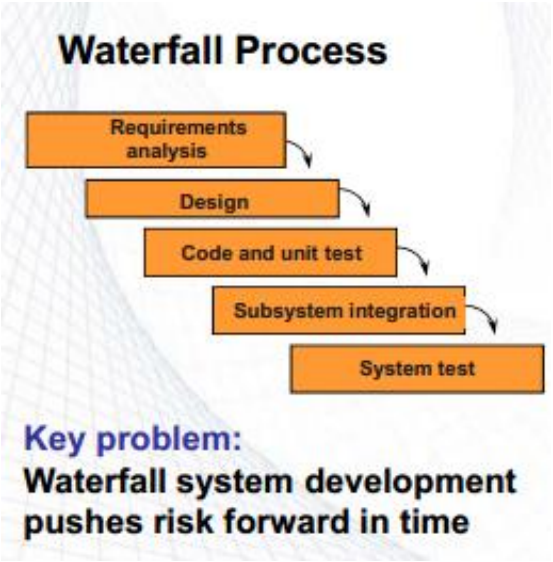
- We are going to buy a new laser printer. We want to have only one failure for 10,000 pages of output (λ_F = 1/10000 pages)
- We observe that failures occur at 4,000 pages, 6,000 pages, 10,000 pages, 11,000 pages, 12,000 pages and15,000 pages of output.
- What can we conclude about this printer?



Entering Reject region. Because of failing the certification test we will reject the printer.

Waterfall Development Characteristics

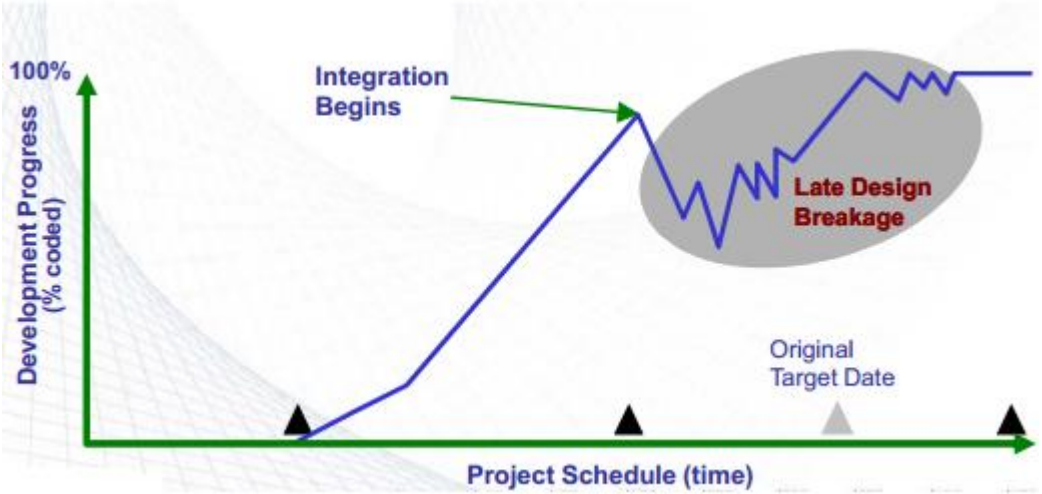
- Delays confirmation of critical risk resolution
- Measures progress by assessing work-products that are poor predictors of time-to completion Delays and aggregates integration and testing
- Precludes early deployment
- Frequently results in major unplanned iterations



Conventional Software Process

Sequential activities

Requirements → Design → Code → Integration →Test

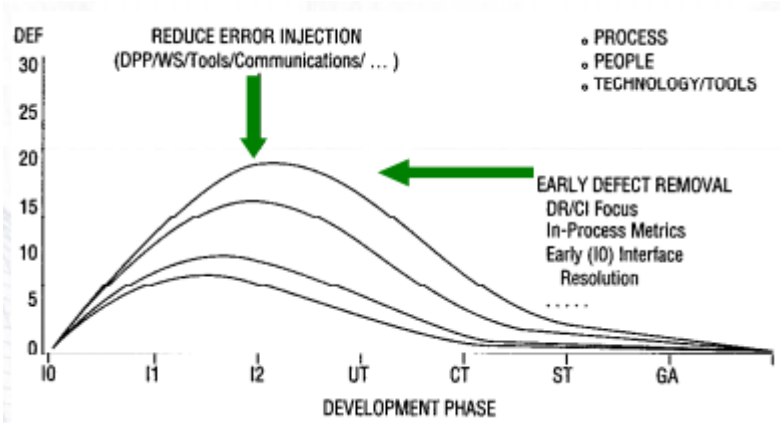


Prevention Methods & Models

- The most important principle in software engineering is “do it right the first time.”
- This may be done via:
  - Preventing errors from being injected into the development process.
  - Improving the front end of the development process to remove as many failures as early as possible; e.g., for waterfall development process, rigorous design reviews and code inspections are needed.
  - In the integration phase, unit test or pre-integration test (the development phase prior to system integration) is the last chance to catch the bugs before they being integrated into the system.

Bidirectional Quality Strategy

- The Rayleigh model is an overall model for quality management in successive development versions of software.
- If the error injection rate is reduced, the area under Rayleigh curve becomes smaller.
- The earlier the peak comes, the smaller the area under the curve.
- Interpretation of "do it right the first time": shift the peak of the Rayleigh model to the left while lowering its peak.



Defect Rate vs. Inspection Effort /1

- Is failure count (defect rate) a good metrics for software quality?
- The more bugs found and fixed doesn’t necessarily imply better quality because the fault injection rate may be different.
- From the defect removal data how do we assess the error injection rate or defect removal?
- Using inspection effort vs. detect rate chart

Defect Rate vs. Inspection Effort /2

- Best case scenario high effort/low defect rate: an indication that the design/code was cleaner before inspections, and yet the team spent enough effort in design review/code inspection that better quality was ensured.
- Good/not bad scenario high effort/high defect rate: error injection may be high, but higher effort spent is a positive sign and that may be why more defects were removed. If effort is significantly higher than the target, this situation may be a good scenario.
- Unsure scenario low effort/low defect rate: not sure whether the design and code were better. Less time in inspection was needed or inspections were hastily done, hence finding fewer defects.
- Worst case scenario low effort/high defect rate: an indication of high error injection but inspections were not rigorous enough. Chanc ined in the design or code at the exit of the inspection process

		Inspection Effort	
		Higher	Lower
Defect Rate	Higher	Good/ Not bad	Worst Case
	Lower	Best Case	Unsure

Reliability Models: Variations

- Evolving programs
  - Dealing with small increments of program size and/or stepwise evolution
- Unreported failures
  - How unreported failures affect failure intensity estimation?

Evolving Programs /1

- Reliability models used in estimating failure intensity (λ) assume that the executed program is stable (not changing, except for those changes that result from failure correction).
- Programs can evolve due to:
  - Requirements changes
  - Integration of new parts
  - Necessity to adapt to changing hardware and software environment
  - Necessity for system performance improvement
  - Evolution as part of the development process

Evolving Programs /2

Possible scenarios:

- Ignoring changes for programs evolving slowly and in small size increments
- Ignoring old data on the old phase and base estimates on the new phase
- Stepwise evolution
  - Applying changes by components

- The effects of evolution can be ignored for programs evolving slowly and in small size increments (increments of less than %5 of total code per week).
- Advantages: No extra testing and data collection is required.
- Disadvantages: Estimates of model parameters and λ / λ_F ratios will lag the true situation and will have error, but the range of errors may be acceptable.

Evolving Programs /3

- In component by component approach, add the component failure intensities separately to obtain the system failure intensity at each stage.
- In operation group approach, add the weighted failure intensities of operation groups to obtain the system failure intensity at each stage.
- stepwise evolution approaches generally work best when having a small number of large changes, each resulting from the addition of an independent element.
- Advantages and disadvantages of stepwise evolution approaches:
- Advantages: System operational profile can be applied directly.
- Disadvantages:
  - The extra data collection required because of the multiple elements
  - Greater estimation error (or later achievement of a specified degree of accuracy) due to smaller sample sizes

… End …