Focus

1. Measuring reliability
2. Improving reliability

What is Reliability?

Assuming that an item is performing its intended function at time equals zero, reliability can be defined as the probability that an item will continue to perform its intended function without failure for a specified period of time under stated conditions.

Product can be -

1. Electronic or mechanical hardware product
2. Software product
3. A manufacturing process or even a service.

Time between failures or its reciprocal failure rate.

Why is Reliability Important?

There are a number of reasons why reliability is an important product attribute, including.

- **Reputation.**
- The more reliable a product is, the more likely the company is to have a favorable reputation.
- **Customer Satisfaction.** While a reliable product may not dramatically affect customer satisfaction in a positive manner, an unreliable product will negatively affect customer satisfaction severely.
- **Warranty Costs.** If a product fails to perform its function within the warranty period, the replacement and repair costs will negatively affect profits, as well as gain unwanted negative attention.
- **Repeat Business.** A concentrated effort towards improved reliability attitude has a positive impact on future business.
- **Cost Analysis.** This life cycle cost analysis can prove that although the initial cost of a product might be higher, the overall lifetime cost is lower than that of a competitor's because their product requires fewer repairs or less maintenance.
- **Customer Requirements.** Many customers in today's market demand that their suppliers have an effective reliability program. These customers have learned the benefits of reliability analysis from experience.

- **Competitive Advantage.** Many companies will publish their predicted reliability numbers to help gain an advantage over their competitors who either do not publish their numbers or have lower numbers.

Difference Between Quality and Reliability?

Even though a product has a reliable design, when the product is manufactured and used in the field, its reliability may be unsatisfactory.

The reason for this low reliability may be that the product was poorly manufactured. So, even though the product has a reliable design, it is effectively unreliable when fielded, which is actually the result of a substandard manufacturing process.

As an example, cold solder joints could pass initial testing at the manufacturer, but fail in the field as the result of thermal cycling or vibration.

This type of failure did not occur because of an improper design, but rather it is the result of an inferior manufacturing process.

So while this product may have a reliable design, its **quality is unacceptable because of the manufacturing process.**

Just like a chain is only as strong as its weakest link, a highly reliable product is only as good as the **inherent reliability of the product** and the **quality of the manufacturing process.**

Quality versus reliability

Reliability is "quality changing over time"	The everyday usage term "quality of a product" is loosely taken to mean its inherent degree of excellence. In industry, this is made more precise by defining quality to be "conformance to requirements at the start of use". Assuming the product specifications adequately capture customer requirements, the quality level can now be precisely measured by the fraction of units shipped that meet specifications.

A motion picture instead of a snapshot	But how many of these units still meet specifications after a week of operation? Or after a month, or at the end of a one year warranty period? That is where "reliability" comes in. Quality is a snapshot at the start of life and reliability is a motion picture of the day-by-day operation. Time zero defects are manufacturing mistakes that escaped final test. The additional defects that appear over time are "reliability defects" or reliability fallout.
Life distributions model fraction fallout over time	The quality level might be described by a single fraction defective. To describe reliability fallout a probability model that describes the fraction fallout over time is needed. This is known as the life distribution model.

Software Product Reliability (Pankaj Jalote)

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2004-145.pdf
For many practical situations, **reliability of a system is represented as the failure rate**.

For measuring the failure rate of a software product, we can have **N installations** of the software under observation.
If the **total number of failures** in all the **N** installations in a time period **T** is **F**, then the best estimate for the failure rate of the software is [18]

$$\lambda = F / (N * T) .$$

This approach for measuring failure rates has been widely used [1, 19].

This straightforward approach for quantifying reliability has **assumptions**.

1. All failures have “equal” reliability impact
2. There is a single number that captures the reliability of the product under all usage scenarios.
3. All the F failures can be recorded
4. the population size N is known.
5. By **normalizing by T*N** (and T is generally measured in **days**,) it is assumed that the system is in use for **same amount of time each day** (generally assumed to be 24 hours.)
6. The **operational profile** is **consistent** across the user base

Reliability formula for massively used software

We view reliability of a product as a vector comprising of failure rates for diff. failure types.

Product Reliability = [λ1, λ2 , λ3 , ……, λn]

Note:

1. From this reliability vector, we can get a **single reliability number** for a product by taking a **weighted sum of the failure rates for different types of failures**.
2. The **weights** will represent the **relative importance** for the product **of the different failure types**.
3. If all **failures are equal (highly unlikely)**, then the overall failure rate is the sum of all the failure rates.

This **failure classification** will have to be from the **user's perspective**, as we are trying to **capture the reliability experience of the user.**

Some of the examples of the types of events that can be included under these categories are given in Figure 1.

- Unplanned Events
  - Crashes
  - Hangs
  - Functionally incorrect response
  - Untimely response – too fast or slow
- Planned Events
  - Updates requiring restart
  - Configuration changes requiring a restart
- Configuration failures
  - Application/System incompatibility error
  - Installation/setup failures

Figure 1: A failure classification

Unplanned Events

o Crashes o Hangs o Functionally incorrect response o Untimely response – too fast or slow

Planned Events

o Updates requiring restart o Configuration changes requiring a restart

Configuration failures

o Application/System incompatibility error o Installation/setup failures Figure 1: A

We need to measure the **usage time very clearly to calculate the failure rate**.

Comparing reliability across different products

If we want to compare reliability of **different products, uniform policies** for recording events will be imposed.
Only when one is sure that different products are recording all the same events can comparisons be meaningfully done

Other factors -> user perception of reliability

The reliability measurements provided by the systems can have other uses as well.

As many studies have shown, **technical reliability** is only one of many factors that **impact** the end users **perception** of reliability.

Examples of other factors that impact the users perception of the product reliability

1. Installation issues (configuration problems, driver incompatibility)
2. Product learning (failure rate decrease following product installation)
3. Patch distribution - complexity, communications, time
4. Ease of management
5. Interoperability etc.

Reliability measurements can be used to understand these factors.

Examples

1. For instance, the **ease of product installation** can be measured based on the time it takes for a product reliability to reach steady state following its installation.
2. **Ease of management** may be measured as the number of actions/reboots it takes to perform any management action.

Obviously the comparisons are more relevant when applied to products within the same class

e.g. **application to application** would be valid but application to operating system would require much **greater interpretation**.

Mass market product

that runs

in **different operational profiles**

with **different user groups**

attaching different levels of importance to different types of failures, this simple approach is not suitable and throws new challenges.

For mass market products,
reliability = vector of failure rates for different types of failures.

This definition allows **different reliability experience** of **different groups** to be quantified in a manner consistent with their environment.
In this paper, we discussed some of the **key issues that arise when measuring reliability of mass market products**, and **suggested a failure classification framework** that can be used for capturing data on failures.

As only specified servers are monitored, the size of the **observed group** is known

The report gives the total number of sessions, the total session length, and the count of crash and hangs failures for some observation period (scaled).

Product Name	No of Sessions	No of Crash Failures	No of Hang Failures	Session Length (mts)	Crash Failure Rate (per hr)	Hang Failure Rate (per hr)
Access	33,000	300	1,000	3,140,000	0.0057	0.0191
Excel	422,000	1200	8,700	46,450,000	0.0015	0.0112
FrontPage	20,000	100	700	2,540,000	0.0023	0.0165
OneNote	24,000	100	1,000	5,940,000	0.0010	0.0101
PowerPoint	153,000	600	3,300	12,920,000	0.0027	0.0153
Publisher	12,000	100	200	900,000	0.0066	0.0133
Word	648,000	2600	29,900	183,530,000	0.0008	0.0097

Table 1: An example CEIP report

Failure rate for these two failure types is then computed from the data – it can be computed as number of failures per session or number of failures per hour of usage

Reliability growth in software products (Pankaj Jalote)

https://www.iiitd.edu.in/~jalote/papers/RelGrowth.pdf
Most of the **software reliability growth models** work under the

Assumption

that reliability of software grows due to the bugs that cause **failures being removed** from the software.

Observation - failure rate improve with time

While correcting bugs will improve reliability, another phenomenon has often been observed – the failure rate of a software product, as observed by the user, improves with time irrespective of whether bugs are corrected or not.

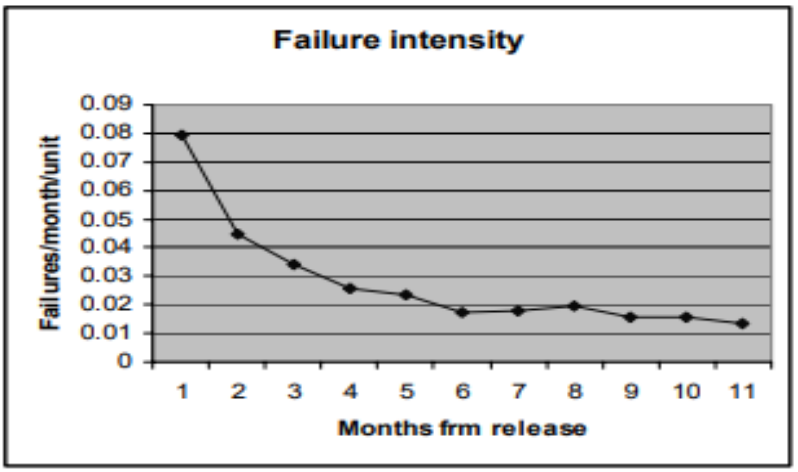


Figure 1: Overall failure rate of a product

Reliability of a product, as observed by users, **varies, depending on the length of time** they have been **using** the product.

Reason for this reliability growth

1. **users gain experience** with the product, they
2. **learn to use** the product correctly and
3. **find work-around** for failure-causing situations

A simple model to represent this phenomenon – we **assume** that the **failure rate** for a product **decays** with a **factor α per unit time**.

Applying this failure rate decay model to the data collected on reported failures and number of units of the product sold, it is **possible to determine** the

1. initial failure rate
2. the decay factor
3. the steady state failure rate of a product.

The paper provides a number of examples where this model has been applied to data captured from released products.

Formula

1. N installations of the software, total
2. F failures are reported by all the installations in
3. time period T

then the failure rate of the software can be computed as

$\lambda = F / (N * T)$ [9].

Using this approach to represent the failure rates of the product assumes that the f**ailure rate -> only on the number of faults in the software**, and
=/ independent of other factors such as
the experience level of the people managing the system.

Stable Reliability after initial rise

It should be clear that this **reliability growth for products** is different from the growth modeled in most of the software reliability growth models – it is due to large number of factors and **not primarily due to fault removal**

Reason for less failure over period of time

The immediate question is why the reliability of a software product improves with time for a user. One reason for this improvement is that the

1. user learns to properly use the product and
2. avoids the situations, commands, and actions that cause failure.

By doing this, the user stops experiencing those failures that are repeatable, and is left to **face** only those that are **random** and **unpredictable**. That is, after initially experiencing a high failure rate, the user reaches a steady state failure rate for the software product.

Modeling failures without fixes

We assume that the failure rate for a user decreases by a factor α every month, till it reaches the steady state.
Using an **initial failure rate**, the **decay factor** α, and t**he steady state failure rate**, if we know the number of total units of the product in the field every month, we can model the aggregate failures.

For a product unit, the failure rate experienced month i after it is purchased is modeled.

$\lambda(i) = \lambda_0 * \alpha * i + \lambda_f$

the steady state failure rate λ_f ,
the initial failure rate λ_0, and
the decay factor α.

For multiple products, since each months many units are being sold.

$F1 = \lambda_0 N1 + \lambda_f N1$
$F2 = \lambda_0 N2 + \lambda_0 N1 \alpha + \lambda_f (N1 + N2)$
$F3 = \lambda_0 N3 + \lambda_0 N2 \alpha + \lambda_0 N1 \alpha 2 + \lambda_f (N1 + N2 + N3)$
.
.
.
$Fi = Fi\text{-}1\ \alpha + \lambda_0 Ni + \lambda_f (Ni + (1- \alpha)(N1+ N2 +\ldots+ Ni- 1))$

the steady state failure rate λ_f ,
the initial failure rate λ_0, and
the decay factor α.
number of units sold each month, starting from the first month, is N1, N2, N3, …, Ni.

Identifying the parameters

Through the failure data from the field each month, and the monthly sales information, it is possible to compute the parameters of the model, using statistical techniques like maximum likelihood principle.
In other words, from the actual data on failures and number of units sold for many months, it is possible to determine the three model parameters used above.

it is estimated that only a fraction of the users actually download the patches.
if a SP has been released, then the total failure rate of a product is the failure rate of two different versions of the product, with the newer version presumably having a lower failure rate. Hence, **to apply the model**, we should **not use the aggregate failure data** of after a SP has been released, or should **find some method of distinguishing failures** of the two **versions**

The informal estimates are that **less than 1% of the users bother to download the patches.** One of the reasons is that patches sometimes solve the problem to which the user has already found a workaround. We assume that the impact of patches on the failure rate is minimal – that is too few users update their copy with the patches, and even if they do, their failure rates do not change substantially as due to the workaround they would have experienced the same failures. If this assumption does not hold, the impact of the patches will have to be estimated and the failure data suitably enhanced to remove that effect.

Applying the model

It is difficult to get the failure data from systems and no of units sold for a consistent amount of time. Units sold might not be units operational as they may be just sold to distributors and lying unused until some end user buy them.
Through the customer support groups (CSG), failures data is collected and this data is also given to the product developers, who then fix the bugs for later releases of the product.

Discussion about collection of failure data.

1. CSG normally don't distinguish between reports from new users and old users, they just report the issues, and provide the workaround to the users.
2. CSG are often the main source of failure data on products, and their data has been used in other reliability and availability computations [1, 5, 7, 10].
3. Automated approaches for recording certain types of failures (raw or processed) also exist for server-type of systems (like L1, L2, L3 events in MRI machines).
4. Not all users report failures, anda user does not report all the failures. Additionally only a **certain class of failures** will be **reported** and annoyances are ignored normally. In other words, the failure data with CSG is not likely to be complete.
5. **Organizations often assume that a fixed percentage of failures are being reported (e.g. the analysis in [1] assumes that 10% failures are being reported through this mechanism.) We also assume that the percentage of failures being reported to PSO is a fixed percent of the total failures being experienced by all the users. Note that as long as the percentage of failures getting reported remains by-and-large the same, regardless of what percentage is used in the model, the trends and ratios will remain the same.**

Available Data

Months after release	1	2	3	…	i
Monthly sales	N1	N2	N3		Ni
Total failures reported	F1	F2	F3		Fi

Even with the assumption about the percentage of failures reported, there are still issues with this data.

Typically bugs - solved -> patches (web, less people use) - many patches -> sp (cds) -> released

a **SP** release of a product may also include **new features**, therefore as well as correcting known bugs it may also create a **new** category of **bugs**.

In most setups, it is not possible to know how many users use the patches, estimated at 1%.

Example:

Data after release month below, no info about patches. SP was not installed till 11th month after the release.

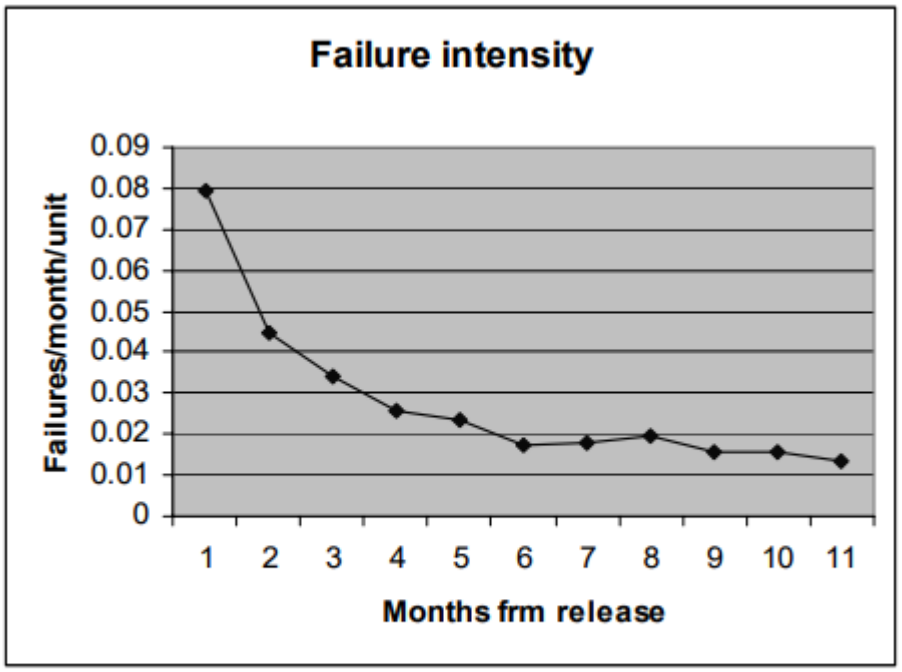


Figure 1: Overall failure rate of a product

Month	Total failures	Monthly Sales
1	367	4618
2	853	14385
3	835	5608
4	791	6186
5	956	9829
6	805	5584
7	967	8240
8	1218	7656
9	1031	4914
10	1144	5295
11	1058	7418

Table 1: Failure and sale data for a product

determine the parameters using the method of **least squares**
λ0 as 0.04 failures/month,
λf as 0.008 failures/month
decay factor α as 0.4.

This is the value that should be compared to any failure rates that may be predicted using the reliability growth models not the results by taking average.

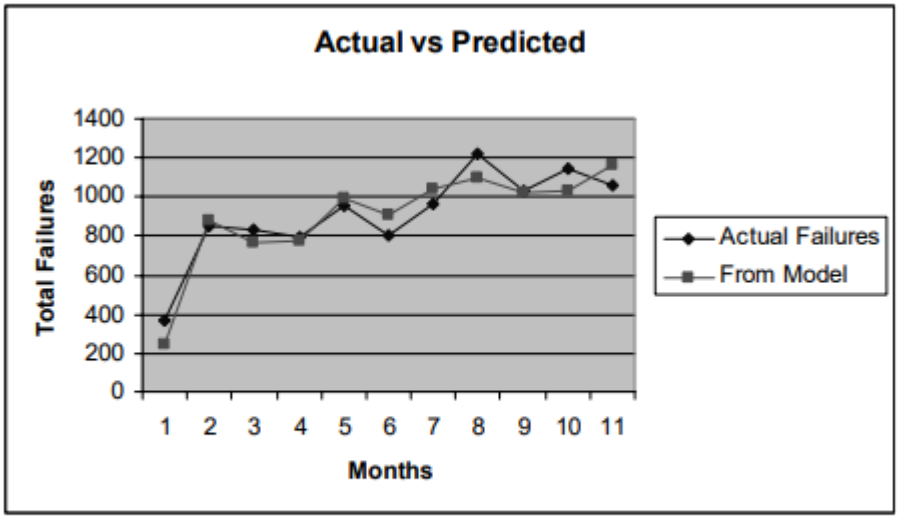


Figure 3: Total failures predicted and actual

Comparison of products

with more complex products (B, C, C is most complex), to tell complexity, internal information about the products and the relative team sizes of the products was used.

	Product A	Product B	Product C
Initial transient failure rate, λ_0 (failures/month)	0.04	0.026	0.177
Steady state failure rate, λ_f (failures/month)	0.008	0.0066	0.067
Decay factor, α	0.4	0.24	0.10

Table 2: Model parameters for three different products

The average error in case of Product B product is about 16%, and in the case of Product C is about 29%.

When the transient failure rate decays rapidly, the accuracy of the model becomes lesser. A model where the decay factor alpha itself is a function of time and reduces with time may reflect reality better in these cases and may give a closer fit to the real data

Alan Wood Tandem Computers

https://sci-hub.tw/https://ieeexplore.ieee.org/document/544240/

After the software is shipped, software vendors receive customer feedback about software reliability. However, **by then it is too late**; software vendors need to know whether their products are reliable before they are delivered to customers. **Software reliability growth models** help provide that information

Literature - Unfortunately, **very little real data** and **models** from commercial applications have been published, possibly because of the proprietary nature of the data

At Tandem,
a major software release - substantial modifications to many products, contain several million lines of code
It follow a well-defined development process and involve a coordinated quality assurance effort. We applied software reliability modeling to **a subset of products for four major releases**

Learnings

collected defect occurrence times during system test and statistically **correlated** the test data with known mathematical functions, called software reliability growth models.

experimented with different mbdels, data, and statistical techniques wth tHe followmg results:

Operational Profiles

Prob distributions of different inputs as used in real scenarios.

Measuring Reliability - http://www.hpl.hp.com/techreports/tandem/TR-96.1.pdf

During system test, we would like the **model to predict the additional ;test effort** required to achieve a suitable quality lek1 (as measured by number of remaining defects)

At the end of system test. we would like the model to predict the number of remaining defects that could be experienced as failures in the field.

Evaluation of model

Stable model

It must become and remain stable, e.g., 10% variation over each week rather than 50 in week 1 and 200 in week 2.

It would be nice if the model was immediately stable, but stability requires a reasonable amount of data. The literature' ! and our experience indicate that the model will not become stable until about 60 percent of testing is complete

Accurate

If the defects discovered after delivery is within the 90 percent confidence interval developed from the model. In experience, 90% confi intervals are > twice the defects predicted.

We also attempted to repre t test time using **number of tests or calendar time** instead of execution e This would be a good measure if all tests had a similar probability of detecting a defect, but often that is not the case in our environment We have some test suites tha execute 100 tests an hour and more sophisticated tests hat execute one test every 24 hours.

Defect data lags TPR (tandem prob report) data - TRPs are also good

We found that TPRs are an excellent surrogate for defects. As Table 5 shows, the predictions based on TPRs become stable earlier than predictions based on defects because there is more data. The ratio of TPRs to defects (column 2 versus column 4) was used to predict a total number of defects from the TPR model. This ratio is usually about 60 percent. During system test, we used the TPRs and defects from a few weeks preceding the current test week to predict a ratio of defects to TPRs. We then used this ratio and the current test week TPR prediction to predict expected defects. The results for Release 3 show that, despite a change in the defect-to-TPRratio from 64 percent in Week 9 to 60 percent in Week 12, this technique still provides a good prediction of total defects.

At tandem

Each defect is fixed when words, the length of time between defect discoveries should increase. When the defect-discovery rate reaches an acceptably he software is **deemed suitable to ship.**

3q
e Goellkumoto (and weibull also) model was significantly better than the other nodels at predicting the number of residual defects.

two-stage models

All of these models **assume that the code being tested does not change during test**, except for defect repair, and they assume that the **effects of defect repair are minimal**.

STATISTICAL TECHNIQUE

The two most common techniques for estimating a mathematical function's parameters from test data are
maximum likelihood and
least squares.

Measuring software rel

http://paris.utdallas.edu/IEEE-RS-ATR/document/2010/8-Predicting%20and%20Improving%20the%20Reliability%20of%20Software%20Intensive%20Systems_Bernstein.pdf

life data analysis

http://www.barringer1.com/Contents.shtml

Failure analysis techniques - http://www.barringer1.com/pdf/Chpt1-5th-edition.pdf

1. Weibull
2. Log Normal analysis

Use cases

1. medical and dental implants,
2. warranty analysis,
3. life cycle cost,
4. materials properties and

5. production process control.

Other models - quantitative models

1. binomial,
2. Poisson,
3. Kaplan-Meier,
4. Gumbel extreme value
5. Crow-AMSAA

Weibull's claim

Data could **select** the **distribution** and fit the **parameters**

Leonard Johnson at General Motors improved on Weibull's plotting methods. Weibull used mean ranks for plotting positions. Johnson suggested the use of median ranks which are slightly more accurate than mean ranks.

Small Samples

Weibull method works with extremely small samples, even **two or three failures** for engineering analysis.
For **statistical relevance**, **larger** samples are needed

Application example

1. Engineer reports 3 failures of a component in service operations during a three-month period.
   The Program Manager asks,
   "How **many** failures will we have in the next quarter, - **Predictions**
   six months,
   and year?"
   What will it cost?
   What is the best corrective action to reduce the risk and losses?
1. To **order** spare parts and schedule maintenance labor,
   how many units will be returned to depot for overhaul for
   each failure mode month-by-month next year?
   The program manager wants to be 95% confident that he will have enough spare parts and labor available to support the overall program.
1. A state Air Resources Board requires a fleet recall when any part in the
   a. emissions system exceeds a 4% failure rate during the warranty period.
   b. Based on the warranty data, which parts will exceed the 4% rate and on what date?
2. After an engineering change,

    a. how **many units** must be tested for **how long**, without any failures, to verify that the old failure mode is eliminated,
    b. or significantly improved with **90% confidence**?
3. An electric utility is plagued with outages from superheater tube failures.
    a. Based on inspection data forecast the **life of the boiler** based on **plugging failed tubes.**
    b. The boiler is replaced when 10% of the tubes have been plugged due to failure.
4. The cost of an **unplanned** failure for a component, subject to a wear out failure mode, is **twenty times** the cost of a planned replacement.
    a. What is the **optimal replacement interval**?

Data, Discrete Versus Life Data

Tests would be **categorized** as success or failure - good parts versus defective parts
modeled with the binomial and Poisson distributions
imprecise unless enormous sample sizes
Crow-AMSAA - more precise and accurate for success-failure / go-no go data when the sample size is large

Data like **age-to-failure** data is much more precise because there is more information in each data point, smaller samples are acceptable.

Ideally, each Weibull plot depicts a **single failure mode**

3 requirements - determine failure time precisely

1. A time origin must be unambiguously defined, •
2. A scale for measuring the passage of time must be agreed to and finally, •
3. The meaning of failure must be entirely clear."

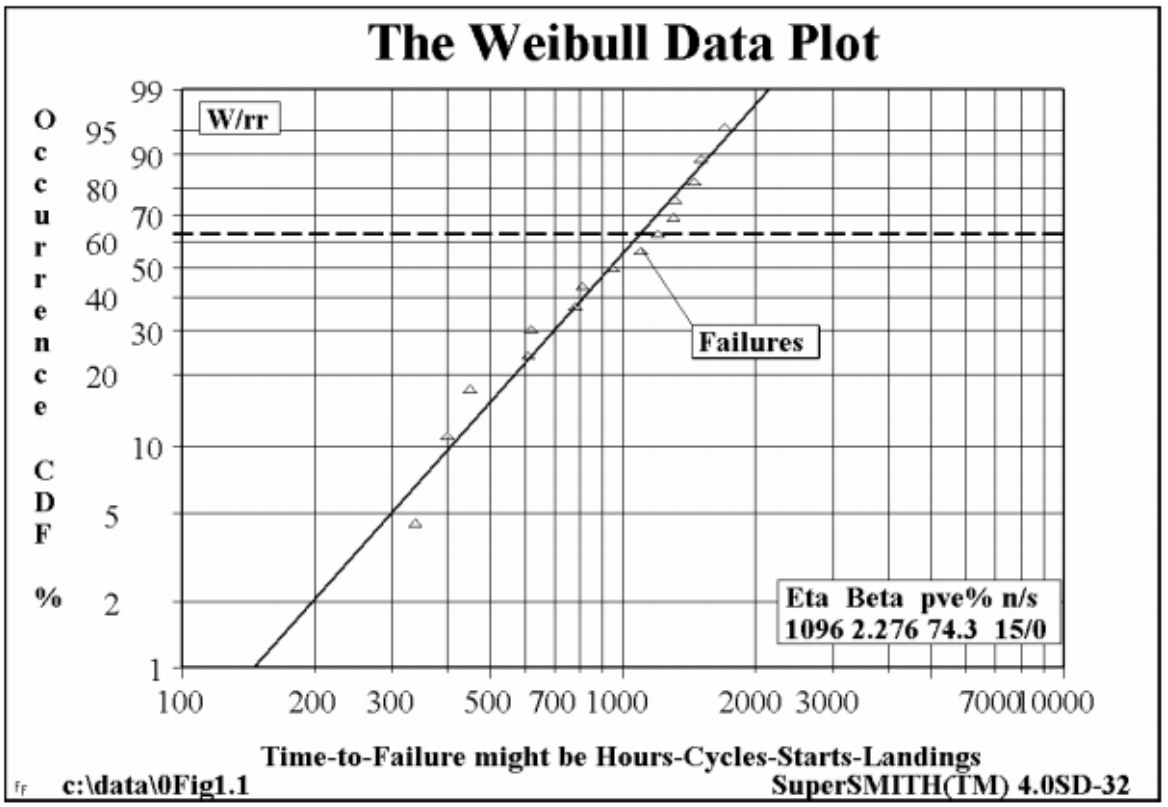


Figure 1-1. The Weibull Data Plot

Age Scale

knowledge of the **physics-of-failure** will provide the **age scale**
The "best" aging parameter data may not exist and substitutes are tried.

For example, the only data on air conditioner compressors may be the date shipped and the date returned. The "best" data, operating time or cycles, is unobtainable, so based on the dates above, a calendar interval is used as a substitute.

Data fit will tell us if the Weibull is good enough.

Weibull Industry Usage - Demonstration Test Plan

http://blog.minitab.com/blog/understanding-statistics/will-the-weibull-distribution-be-on-the-demonstration-test

Weibull distribution is frequently used to model time-to-failure data , how long it will take to fail

You can use it prove the reliability of a component manufactured by you. A good way to do this is to follow a **Demonstration Test Plan**.

Minitab's test planning commands make it easy to to determine the sample size and testing time required to show that you have met reliability specifications.

Your test plan will include:

- The number of units or parts you need to test
- The stopping point, which is either the amount of time you test must each part or the number of failures that must occur
- The measure of success, which is the number of failures allowed in a passing test (for example, every unit runs for the specified amount of time and there are no failures)

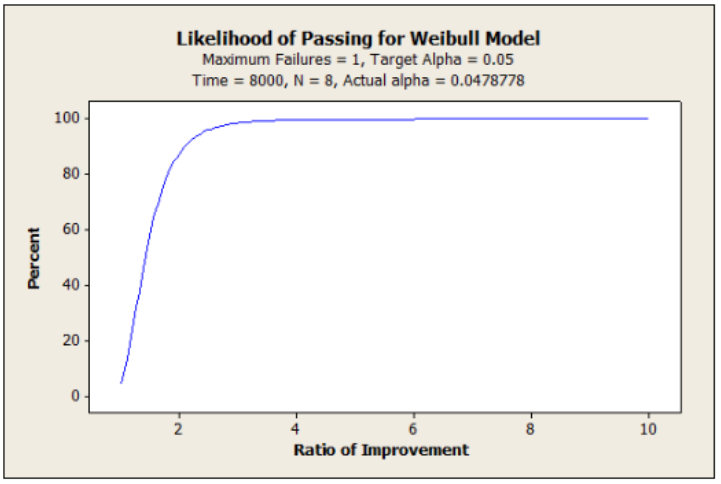
Demonstration Test Plans

```
Reliability Test Plan
Distribution: Weibull, Shape = 3
Percentile Goal = 2000, Target Confidence Level = 95%
```

Failure Test	Testing Time	Sample Size	Actual Confidence Level
1	8000	8	95.2122

Looking at the Sample Size column in the output above, we can see we'll need to test 8 combustors for 8000 cycles to demonstrate with 95.2% confidence that the first percentile is at least 2000 cycles.

When it generates your Demonstration Test Plan, Minitab also creates this graph:



The graph gives us a visual representation of the likelihood of actually passing the demonstration test. Here,

- The very sharp rise between 0 and 2 indicates that the probability of your 1-failure test passing increases steadily as the improvement ratio increases from zero to two.
- If the improvement ratio is greater than about two, the test has an almost certain chance of passing.

Based on this information, if the (unknown) true first percentile was 4000, then the improvement ratio = 4000/2000 = 2, and the probability of passing the test will be about 0.88. If you reduced the value to be demonstrated to 1600, then the improvement ratio increases to 2.5 and the probability of passing the test increases to around 0.96. Reducing the value to be demonstrated increases the probability of passing the test. However, it also makes a less powerful statement about the reliability of the turbine engine combustor.

http://www.indium.com/blog/weibull-analysis-ii-the-curse-of-the-early-first-failure.php

Demonstration of effect of first failure on the reliability calculation. First failure can be **removed** if proper **root cause analysis** is done to get the root cause was not the product quality but process issue, still it can be very risky to remove data point.