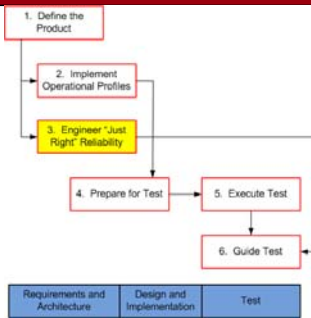


SRE: Engineering "Just Right" Reliability



This lecture provides reference material for Chapter 3 of the book entitled "Software Reliability Engineering: More Reliable Software Faster and Cheaper" by John D. Musa © 2004

This lecture material is copyrighted by Laurie Williams (2007). However, you are encouraged to download, forward, copy, print, or distribute it, provided you do so in its entirety (including this notice) and do not sell or otherwise exploit it for commercial purposes.

For PowerPoint version of the slides, contact Laurie Williams at williams@csc.ncsu.edu.

Faults vs. Failures



A programmer makes a **mistake**.

The mistake manifests itself as a **fault** [an incorrect step, process, or data definition in a program] in the program.

A **failure** [the inability of a system or component to perform its required function within the specified performance requirement] is observed if the fault is made visible. Other faults remain **latent** in the code until they are observed (if ever).

Definitions - Reprisal

- **Reliability:** probability that a system will operate without failure for a specified number of natural units or a specified time
- **Natural unit:** unit that is related to the output of a software-based product and hence the amount of processing done
- **Failure intensity:** failures per natural or time unit
- **Availability:** average (over time) probability that a system is currently functional in a specified environment OR ratio of uptime to the sum of uptime plus downtime

Four steps to engineering “just right” reliability for your product

1. Define “failure” for the product
2. Choose a common reference unit for all failure intensities
3. Set a system failure intensity objective
4. For any software you develop:
 - a) Find the developed software intensity objective
 - b) Choose software reliability strategies to optimally meet the developed software failure intensity objective

Step One: Define “failure” for the product

- Establishing negative requirements on program behavior, as defined by users [what the system should NOT be doing]
- Define failure severity classes [set of failures with same degree of per-failure impact on users]

Table 3.1 Fone Follower failure severity classes

Failure severity class	System capability impact	Example
1	Basic service interruption	Calls misforwarded or not forwarded
2	Basic service degradation	Phone number entry inoperable
3	Inconvenience, correction not deferrable	Graphical user interface for administrators inoperable
4	Minor tolerable effects, correction deferrable	Date missing from display

Step Two: Choose a common reference unit for all failure intensities

- Preferable: failures per natural unit from a user perspective [rather than execution time or operation time]
 - Expressed in user terms
 - Measure of failure-inducing stress on the product
 - Does not change with the throughput of the specific computer you execute the software on
 - Easier to instrument

Step Three – Part a: Set a system failure intensity objective

- Determine whether your users need reliability [nonperformance has the greatest impact] or availability [downtime has the greatest impact] or both
 - 99% effective availability
 - Service interruption takes 14 minutes
 - Machine downtime of 1 minute
 - Failure intensity objective (λ_F):

$$A(t) = \frac{1}{1+t_m\lambda_F} \text{ or } \lambda_F = \frac{1-A(t)}{t_m A(t)}$$

 - A = availability; t_m = average downtime per failure
 - $\lambda_F = (1-0.99)/(.25 \times .99) = .04$ failures/hour or about 4 failures per 100 hr.

Step Three – Part b: Set a system failure intensity objective

- Determine the overall (for all operations) reliability and/or availability objectives
 - Who are the users?
 - What do they want?
 - Which objective is best for users in their particular competitive environment?
 - Might be established via contractual, warranty, or regulatory requirement
 - Analyze profit vs. reliability or availability objective
 - Analyze experience with previous or similar systems
 - Analyze the competition
- Usually take the minimum failure intensity objective based on different users or customers.

Table 3.2 Estimated product profit vs failure intensity

Failure intensity (failures per million transactions)	Profit (millions of dollars)
1000	0.6
500	1.5
250	5.8
100	17.2
50	21.4
25	19.8
10	16.3

System failure intensity guidelines

Table 3.4 System failure intensity objective guidelines

Failure impact	Typical failure intensity objective (failure / hr)	Time between failures
Hundreds of deaths, more than \$10 ⁹ cost	10 ⁻⁹	114,000 years
One death, around \$10 ⁶ cost	10 ⁻⁶	114 years
Around \$1000 cost	10 ⁻³	6 weeks
Around \$100 cost	10 ⁻²	100 hours
Around \$10 cost	10 ⁻¹	10 hours
Around \$1 cost	1	1 hour

Step Three – Part c: Set a system failure intensity objective

- Find the common failure intensity objective (FIO) for the reliability and availability objectives [when both exist]
 - Convert availability objective to FIO
 - Example: availability objective of 0.9999 and downtime of 0.01 hr.
 - $\lambda_F = (1-A)/(At_m) = 0.00001/(.9999)(0.01) = 0.01$ failures/hr
 - Change to natural units
 - We have 1000 transactions/hour so 0.01 failures/KTransactions
 - Usually classify all failures that cause interruption of service (availability problems) as Sev 1, but FIO is for all failures
 - $\lambda_F = \lambda_{F1}/R$ where λ_{F1} is Sev 1 failure intensity and R is the expected ratio of Sev 1 failures to total failures
 - If R is 0.05, $\lambda_F = (0.01)/(0.05) = .2$ failures/KTransactions
 - The ratio R is typically 0.05 to 0.1.
- Take smaller of FIO for reliability and availability objectives for common FIO

Step Three – Part c: Set a system failure intensity objective (cont'd)

- If you need to . . . convert from FIO to reliability

$$R = e^{-\lambda t}$$
 - where R is reliability, λ = failure intensity, and t = number of natural or time units
 - if λt is less than .05, $R \approx 1 - \lambda t$ with less than 2.5% error
- If you need to . . . convert from reliability to FIO

$$\lambda = \frac{-\ln(R)}{t}$$
 - If R is larger than 0.95 than $\lambda \approx (1-R)/t$ with less than 2.5% error
- Example:
 - Example: R = 0.992 for 8 hr, then $\lambda = (.008)/8 = .001$ failures/hour or 1 failure per 1000 hours.
 - Example: λ of 1 failure / Kpages or 1 failure/KTransactions, R = $1 - (.001)(8) = 0.992$ for 8 pages or 8 transactions

Step Four-Part a: Find the developed software intensity objective

- A component failure is a behavior that would cause a system failure. Add all of the expected component failure intensities together to get the system expected acquired failure intensity.
- Developed software FIO = System FIO - hardware FIO + acquired FIO
- Example (Fone Follower):
 - Fone Follower base product “allows” 1 failure/10,000 calls or 100 failures/Mcalls.
 - Vendor field data gives failure intensities of 1 failure/Mcalls for the hardware and 4 failures/Mcalls for the operating system
 - Developed software failure intensity = $100 - (4+1) = 95$ failures/Mcalls
- We will use the developed software intensity objective to choose the software reliability strategy and to track reliability growth during system test with the failure intensity to failure intensity objective ratio.

Step Four-Part b: Choose software reliability strategies to optimally meet the developed software failure intensity objective

- A *software reliability strategy* is a development activity that reduces failure intensity, incurring development cost and perhaps development time
- Plan software reliability strategy in the requirements phase, focusing on new operations of release.
- A software reliability strategy may be *selectable* (requirements, design, or code reviews) or *controllable* (amount of system test, amount of fault tolerance).
- Assumes the team is using a good development process and conducts unit test to a high level of path coverage.

Software reliability strategies (cont'd)

- Current view of possible reliability strategies:
 - Use of requirements review
 - Use of design review
 - Use of code review
 - Degree of fault tolerance designed into system
 - *Fault tolerance* is the ability of a system or component to continue normal operation despite the presence of hardware or software fault. [IEEE]
 - Amount of system test

Choosing software reliability strategies

- **Basic failure intensity (λ_0)** is the failure intensity that would exist at the start of system test for a project without reviews or fault tolerance.
 - Estimated for new operations only, assuming the FIO of release operations has already been achieved
 - Similar across many products given that no software reliability strategies have been applied
- **Failure Intensity Reduction Objective (FIRO)** is the ratio of the basic failure intensity to the failure intensity objective and the failure intensity reduction that must be obtained through software reliability strategies
- We choose software reliability strategies so that we can achieve our FIRO, and therefore, our FIO.

Determine FIRO . . . Developed software FIO

- Allocate the FIRO among the software reliability strategies
- Reviews yield failure intensity reductions of about 2 each (each type of review removes no more than 50% of the faults)
 - Order of cost effectiveness: requirements, design, and code
- Determine FIRO
 - Express the developed software FIRO in execution time
 - Compute the basic failure intensity (λ_0)
 - Compute FIRO by dividing basic failure intensity by developed software FIO
- Fone Follower
 - Developed software FIO= 95×10^{-6} failures/calls.
 - Execution time/call = 2×10^{-6} exec hr/call.
 - Developed software FIO= 47.5 failures/exec hr.

Determine FIRO: Compute basic failure intensity and FIRO

- Compute basic failure intensity (λ_0)
 - $\lambda_0 = K \omega_D Nr / Q_x$
 - Determine fault exposure ratio K [failures/fault] and fault density ω_D [faults/source instruction] at the start of system test based upon:
 - A previous release of the product
 - A similar product with a similar development environment
 - Industry values. Capers Jones: $K = 4.2 \times 10^{-7}$ failures/fault and $\omega_D = 6 \times 10^{-3}$ faults/source instruction.
 - N is fraction of code that is new
 - Q_x is the ratio of object to source instructions,
 - r is throughput
- Fone Follower:
 - use industry average. $N=1$, $Q_x = 3$ object instructions/source instruction, and $r = 3 \times 10^8$ object instructions/second
 - $\lambda_0 = (4.2 \times 10^{-7})(6 \times 10^{-3})(1)(3600 \times 3 \times 10^8) / 3 = 907.2$ failures/ exec hr
 - Where the numerator was multiplied by 3600 to obtain failure intensity/hour rather than per second.
 - $FIRO = \lambda_0 / \text{developed software FIO} = 907.2 / 47.5 = 19.1$

Plan the strategies

- Allocate FIRO among the software reliability strategies in order of decreasing cost effectiveness until FIRO is met.
- Order:
 - Early system test (driven by operational profile)
 - Requirements review
 - Design review
 - Code review
 - Fault Tolerance
 - + late system test

Table 3.7 Fone Follower - allocation of FIRO to software reliability strategies

Step	Reliability strategy	FIRO allocation	Remaining FIRO
	Start		19.1
1	Early system test	8	2.4
2	Requirements reviews	2	1.2
3	Design reviews	2	0.6

Choosing between fault tolerance and late system test

- **Fault tolerance fraction (FTF)** is the proportion of remaining FIRO, after reviews and early system test, that is to be obtained through fault tolerance (as contrasted to late system test)
 - Product specific
 - Set based on experience with first release or similar product
 - If no data is available use $FTF = 0.5$. [equal cost between fault tolerance and late system test]

Summary

- Defining what we mean by “just right” reliability in quantitative terms is one of the key steps in achieving the benefits of software reliability engineering.
- Through examining “just right” reliability in quantitative terms makes it possible for us to balance customer needs for reliability, delivery date, and cost and to develop and test the product more efficiently.