Reliability Engineering

Reliability Engineering Practices

- Define reliability objectives
- Use operational profiles to guide test execution
  - Preferable to create automated test infrastructure
- Track failures during system tests
- Use reliability growth curves to track quality of product
- Release when quality of product meets reliability objectives

- Paper on “Software reliability engineered testing”
  - http://www.stsc.hill.af.mil/crosstalk/1996/06/Reliabil.asp
  - Fairly easy to read, good overview

Defining reliability objectives

- Quantitative targets for level of reliability ("failure intensity": failures/hour) that makes business sense
  - Remember for defects it was "not discussable"?

Impact of a failure	FI Objective	MTBF
100's deaths, >10^9 cost	10^{-9}	114,000yrs
1-2 deaths, around 10^6 cost	10^{-6}	114 yrs
$1,000 cost	10^{-3}	6 weeks
$100 cost	10^{-2}	100 h
$10 cost	10^{-1}	10 h
$1 cost	1	1 h

From John D. Musa

Testing based on operational profiles

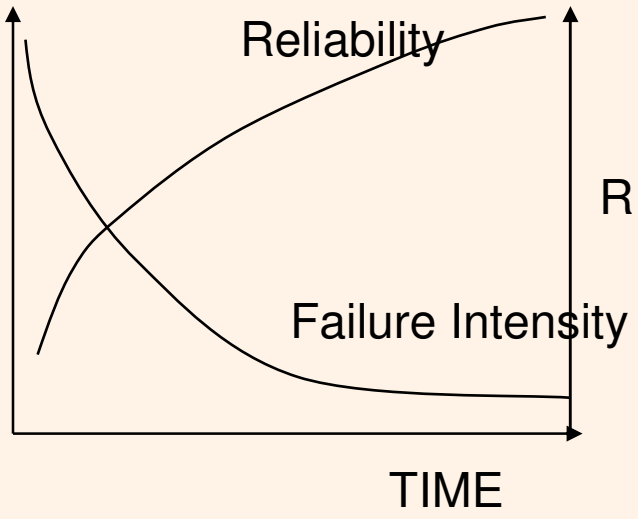
- Done during black-box system testing
- Preferably mix of test cases that match operational profile
- If possible, create automated test harness to execute test cases
  - Need to run large numbers of test cases with randomized parameters for statistical validity
- Execute test cases in randomized order, with selection patterns matching frequencies in operational profile
  - Simulating actual pattern of usage

Builds and code integration

- Most large projects have periodic builds
  - Development team integrates a new chunk of code into the product and delivers to test team
- Test team does black box system testing
  - Identifies bugs and reports them to dev team
- Track pattern of defects found during system testing to see how reliability varies as development progresses
  - Defects found should decrease over time as bugs are removed, but each new chunk of code adds more bugs
  - Pattern of reliability growth curve tells us about the code being added, and whether the product code is becoming more stable
- Pattern can also be used to statistically predict how much more testing will be needed before desired reliability target reached
  - Useful predictions only after most of the code integrated and failure rates trend downward

Tracking failures during testing

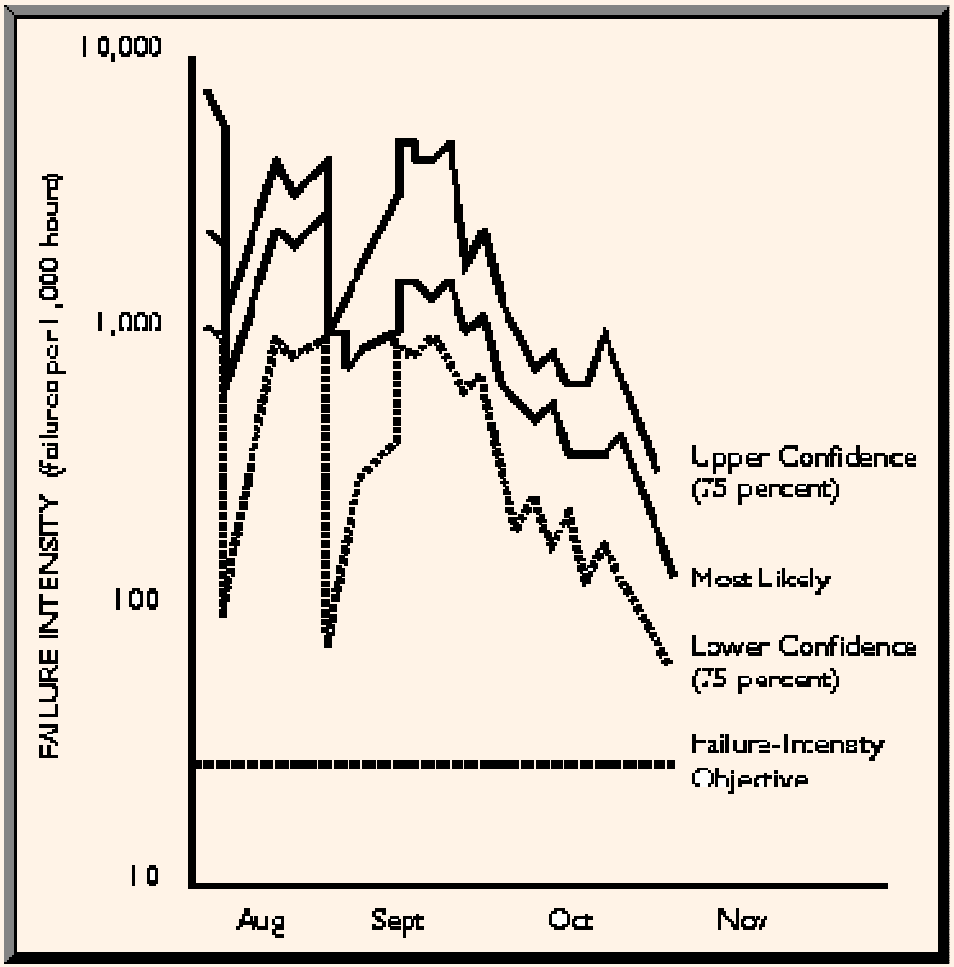
- Enter data about when failures occurred during system testing into reliability tool e.g. CASRE
  - Plots graph of failure intensity vs. time



(in concept)

From netserver.cerc.wvu.edu/numsse/Fall2003/691D/lec3.ppt

A more realistic curve



From http://www.stsc.hill.af.mil/crosstalk/1996/06/Reliabil.asp

Reliability Models

- Assumes a particular reliability model
- Different reliability models proposed
  - Differ in statistical model of how reliability varies with time
  - Fitting slightly different curves to the data
  - Built into the tool
- Another statistical model, the Rayleigh model, can be used to predict remaining defects
- My take on this:
  - Trying to get too mathematically precise about the exact level of reliability is not valid!
  - Just models, use to get a reasonable estimate of reliability

Reliability Metric

- Estimated failure intensity
  - (Reliability = 1 / failure intensity)
  - Tool shows statistical estimates of how failure intensity varies over time
- The curve is referred to as the “reliability growth curve”
  - Note that the product being tested varies over time, with fixes and new code
  - In-process feedback on how quality is changing over time

Interpreting reliability growth curves

- Spikes are normally associated with new code being added
- Larger volumes of code or more unreliable code causes bigger spikes
  - The curve itself tells us about the stability of the code base over time
- If small code changes/additions cause a big spike, the code is really poor quality or impacts many other modules heavily
- The code base is stabilizing when curve trends significantly downward
  - Release (ideally) only when curve drops below target failure intensity objective … indicates right time to stop testing
  - Can statistically predict how much more test effort needed before target failure intensity objective needed.
- Shows up “adding a big chunk of code just before release”
  - Common occurrence! … Getting code done just in time

- Note that there is definitely random variation!
  - Hence “confidence intervals”
  - Avoid reading too much into the data

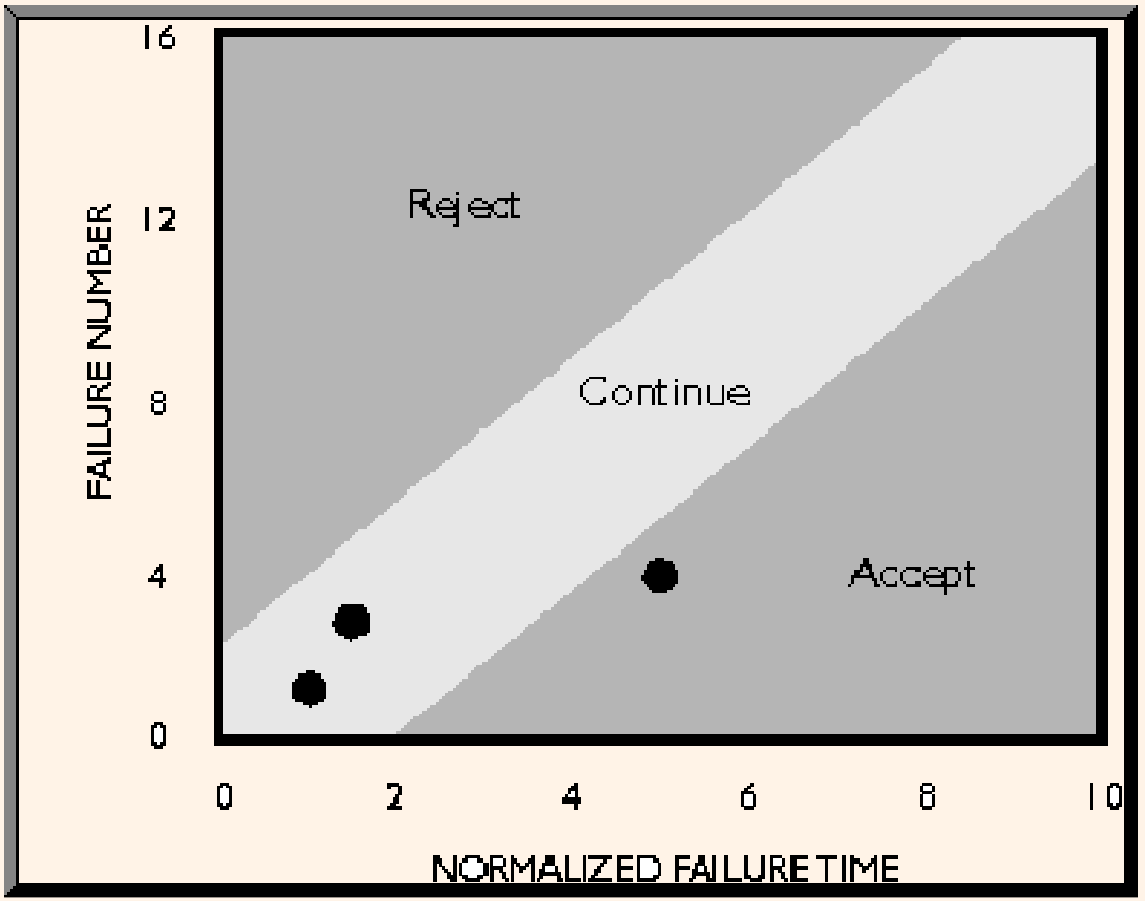
Limitations of reliability curves

- Operational profiles are often “best guesses”, especially for new software
- The reliability models are empirical and only approximations
- Failure intensity objectives should really be different for different criticality levels
  - Results in loss of statistical validity!
- Automating test execution is challenging (particularly building verifier) and costly
  - But it does save a lot over the long run
  - More worthwhile when reliability needs are high
- Hard to read much from them till later stages of system testing … very late in the development cycle

Reliability Certification

- Another use for reliability engineering is to determine the reliability of a software product
  - E.g. you are evaluating web servers for your company website – reliability is a major criterion
- Build test suite representative of your likely usage
  - Put up some pages, maybe including forms
  - Create test suite that generates traffic
  - Log failures e.g. not loading, wrong data received
  - Track failure patterns over time
- Evaluate multiple products or new releases using test suite, to determine reliability
  - Avoids major problems and delays with poor vendor software
- Note that this applies the analysis to a fixed code base
  - Fewer problems with statistical validity

Example certification curve



From http://www.stsc.hill.af.mil/crosstalk/1996/06/Reliabil.asp

Summary

- Software reliability engineering is a scientific (statistical) approach to reliability
- Vast improvement over common current practice
  - "Keep testing until all our test cases run and we feel reasonably confident"
- Avoids under-engineering as well as over-engineering ("zero defects")
- When done well, SRE adds ~1% to project cost
  - Musa's numbers, my experience: ~10% for medium-sized projects if you include cost of automated testing
  - Note that as the number of builds and releases increases, automated testing more than pays for itself