Test Metrics

Software Requirements & Project Management
CITS 3220
Lecture Overview

- What is software reliability?
- How can we test for reliability?
- Can we measure reliability?
- Defect Density
- When to stop testing
Software Reliability
What is SW reliability?

- A key high level attribute of software; the most studied of all quality attributes
- The basic problem of reliability theory is to predict when a system will eventually fail
- All attempts to measure reliability are examples of PREDICTION: use the data we have available (past history) and a prediction model to make accurate predictions about the future
- Reliability is probabilistic in nature
Failures, Faults, Defects
(IEEE Std 729, 1983)
(beware B&D ch 8 definitions differ)

- A failure is the departure of a system from its required behaviour
- A fault occurs when a human error results in a mistake in some SW product
- Defects normally refer collectively to faults and failures
- *An error may lead to faults may lead to failures*
SW vs HW reliability

- HW components fail due to physical wear: corrosion, shock or over-heating
- HW components can be replaced after failure
- There are many reasons for SW to fail, but none involves wear and tear
- Usually, software fails because of a design problem
- SW failures also occur when code is written or changes introduced – new failures may not be manifest immediately
Reliability Testing
also called Statistical Testing
Operational Profiles of Usage

- In order to predict future reliability, we need observed data of failure times in the past
- *but* data from system testing (defect testing) is unlikely to reproduce “normal” usage patterns
- So, we create operational profiles to anticipate typical user interaction (J.Musa 87)
- and use such operational profiles as test cases for reliability testing
Reliability testing procedure

- Determine operational profile of SW use
- Generate a set of test data corresponding to this profile
- Apply tests, measuring amount of execution time between each failure
- After a statistically valid number of tests have been executed, reliability can be measured
Advantages of Reliability Testing

- Concentrates on the parts of the system most likely to be used
- Gives confidence that reliability predictions based on test results will be accurate
Defect Density Measures of SW Quality
Defect Density Definition

$$\text{defect density} = \frac{\text{number of known defects}}{\text{product size}}$$

Product size is usually measured in lines of code (or FPs).
We count known defects but the product will still contain latent defects – those not yet discovered.
DD Advantages

- De facto standard measure of SW quality
- Provides useful information
- Not too hard to collect the necessary data
DD disadvantages

- There is no general consensus on
  - what constitutes a defect
  - how to measure SW size
- DD is a *product measure* but it is derived from the *process* of finding defects
  - May say more about quality of defect finding and reporting, than about the quality of the product itself
- Should measure number of defects with respect to time (e.g. MTBF) rather than product size
DD Cautions

- Even if we knew the exact number of residual faults in a system, it is still
  - Difficult to determine in advance the seriousness of a fault
  - There is much variability in the way the system is used by different users; a system is not always used as expected
Other quality measures based on defect counts

- System spoilage = \[ \frac{\text{time to fix post-release defects}}{\text{total system development time}} \]
- Cumulative fault density
- Total serious faults found
- Mean time to serious faults
- High level design review errors per KNCSL
- etc. etc.
## Failure Class Descriptions

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transient</td>
<td>Occurs only with certain inputs</td>
</tr>
<tr>
<td>Permanent</td>
<td>Occurs with all inputs</td>
</tr>
<tr>
<td>Recoverable</td>
<td>System can recover without operator intervention</td>
</tr>
<tr>
<td>Unrecoverable</td>
<td>Operator intervention is needed to recover</td>
</tr>
<tr>
<td>Non-corrupting</td>
<td>Failure does not corrupt data</td>
</tr>
<tr>
<td>Corrupting</td>
<td>Failure corrupts system</td>
</tr>
</tbody>
</table>
When to Stop Testing
When to Stop Testing

- The intuition that testing discovers first the trivial, easy to correct, faults and the more difficult ones later is WRONG

- How can you know when you’ve discovered “most of them” or the “most important” ones?
Fault Seeding

- Used measure testing effectiveness
- **Idea:** in order to measure how thorough the testers have been, first insert some dummy faults into the program (fault seeding) and then count how many of these are detected.
- Previous project experience can also identify likely problem areas
Fault Seeding

- **Assumption:**
  \[
  \frac{\text{detected seeded faults}}{\text{total seeded faults}} = \frac{\text{detected non-seeded faults}}{\text{total non-seeded faults}}
  \]

- **Problem:** it is difficult to make seeded faults representative of the real ones.

- **Conclusion:** this approach is most useful for testing systems which are similar to ones we have built before
A Balanced Method

Idea: Test Group 1 and Test Group 2 test the same program. Let F1 be the set of faults detected by group 1, F2 those by group 2 and

- $x$ is size of $F_1$ and $y$ is size of $F_2$
- $q$ is size of $F_1 \cap F_2$ (the common faults)
- $n$ is unknown: the total number of faults in the program
Balanced method (cont)

- So, the **effectiveness** of Groups 1 and 2 is given by,

\[
E_1 = \frac{x}{n} = \frac{q}{y} \quad \text{and} \quad E_2 = \frac{y}{n} = \frac{q}{x}
\]

and thus

\[
n = \frac{x \cdot y}{q}
\]

- **Conclusion**: So far we have found \((x+y-q)/n\) proportion of the faults.

- **Conclusion**: Group 1 is better than group 2 (or vice versa)

- **Problem**: probably assumes too much uniformity of distribution and detection
Worked Example

- During testing group A detect 20 faults and group B detect 25 faults. 10 of the faults detected are common to both groups. Estimate the total number of indigenous faults in the program and thus number of indigenous faults remaining undetected in the program.

- \( x=20 \) \( y=25 \) \( q=10 \)
- Effectiveness of A = \( q/y = 10/25 = 0.4 \)
- Effectiveness of B = \( q/x = 10/20 = 0.5 \)
- Estimated total faults is \( n = x.y/q = 20.25 / 10 = 50 \)
- Faults found so far = \( 20 + 25 - 10 = 35 \)
- So remaining faults est. 15 (ie 30%) still to find