12. Metrics and evaluation

Stopping criteria
Testing effort
Testing efficiency
Testing effectiveness
Planning, costing, documentation
Functional vs structural testing

The fundamental problem of testing software

- We cannot test for everything
- No system can be completely tested

When should testing stop?

"There is no single, valid, rational criterion for stopping.

Furthermore, given any set of applicable criteria, how each is weighted depends very much upon the product, the environment, the culture and the attitude to risk."

Boris Beizer

When should testing stop?

- Possible answers

Things to measure

  - When you run out of time
  - When you run out of resources
  - When continued testing causes no new failures
  - When continued testing reveals no new faults
  - When you cannot think of any new test cases
  - When you reach a point of diminishing returns
  - When mandated coverage has been attained
  - When all faults have been removed

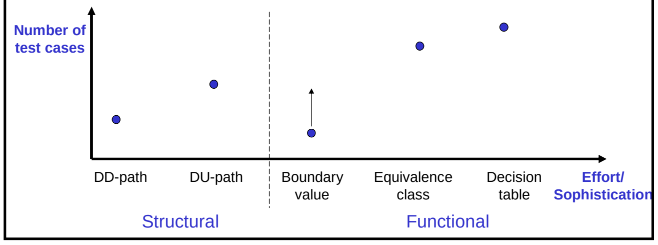
Paul C Jorgensen

Stopping criteria

- Based on
  - Testing effort
  - Testing efficiency
  - Testing effectiveness
- Supported by metrics
  - Quantitative measures
- Multi-dimensional problem
  - Combining the metrics and considering other semi-quantitative criteria

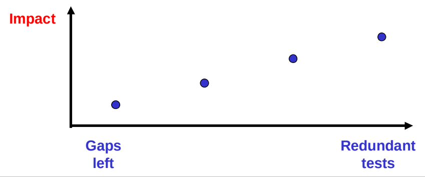
Testing effort

- Testing techniques vary in
  - Number of tests generated
  - Effort to develop the test cases
- There is always a trade-off between the two



Testing efficiency

- Intuitively, the number of tests to ensure the maximum detection of failures and faults
- Relies on balance between
  - Leaving gaps in untested functionality / coverage
  - Executing redundant tests
- Difficult to quantify
- Balance should be decided based on the risk analysis



Testing efficiency

- Pragmatic approach to estimating the degree of redundancy

Functional testing:
- Annotate each test case with the "purpose of testing"
- If several tests have the same purpose, they are likely to be redundant

Structural testing:
- If the same path is traversed more than once, there is likely to be a redundancy

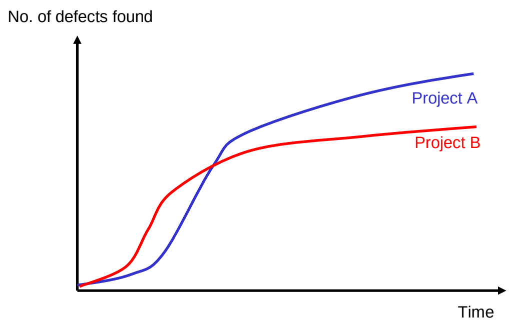
- Detecting gaps is much harder
  - For high impact software components, accept redundancy as a necessary cost for achieving completeness

Testing Effectiveness

- Tells us how effective are test cases in discovery of the failures and faults
  - Difficult to measure in general, because it requires that we know *a priori* about all the bugs

"Because we don't know all the faults in the program, we could never know if the test cases from a given method revealed them."

Defect discovery graph



Is it useful?

Testing Effectiveness

$$\text{Effectiveness} = \frac{N}{N + S}$$

N - Number of faults / failures found by defect removal

S - Number of faults not found

Impossible to know

Testing Effectiveness

- Measuring effectiveness for functional methods of testing
  - The theoretical formula cannot be applied
- A practical approach
  - Track the types and frequencies of bugs WHILE developing software
  - Work "backwards", using post-hoc measures of DECREASE in errors

Testing Effectiveness

Theoretical formula:

$$\text{Effectiveness} = \frac{N}{N + S}$$

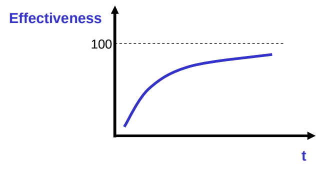
N - Number of faults / failures found by defect removal

S – Number not found

Impossible to know

Testing Effectiveness

Practical formula:



$$\text{Effectiveness} = \frac{N}{N + \hat{S}}$$

N - Number of faults / failures found by defect removal

$\hat{S}$ - Bugs found subsequently

e.g. during first year of operation
or between Unit and Integration Testing

Typical performance: 85% pre-release, 15% post-release

Other measures of effectiveness

- Coverage metrics
  - Used for structural methods of testing
  - Measure a ratio of the tested components of the program to the possible total number of components
  - e.g. a ratio of the tested DU-paths in the program to all the DU-paths in the program (data flow testing)
  - e.g. a ratio of C_1 to C_∞ coverage (path testing)
- Note: Coverage metrics do NOT measure testing effectiveness as such

Other measures of effectiveness

- Combining strengths of functional and structural methods
  - Functional methods for generating test cases
  - Structural methods for obtaining countable quantities (number of program paths, DD-paths, DU-paths etc)

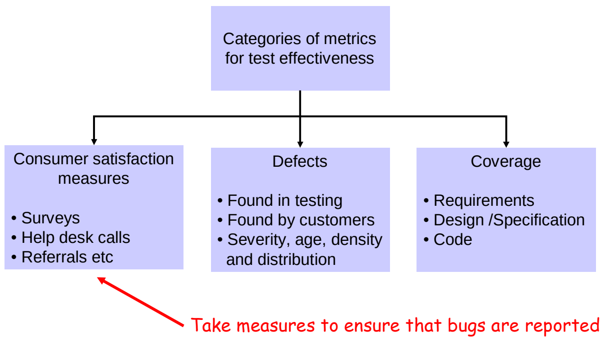
Other measures of effectiveness

- Definitions
  - Functional technique M generates **m** test cases
  - Metric S identifies **s** structural elements (e.g. DU-paths), in total, in the unit being tested
  - When **m** cases are executed, they traverse **n** of **s** structural elements
- Measures
  - **Coverage** of methodology M w.r.t. metric S : $\frac{n}{s}$
  - **Redundancy** of methodology M w.r.t. metric S : $\frac{m}{s}$
  - **Net redundancy** of methodology M w.r.t. metric S : $\frac{m}{n}$

Other measures of effectiveness

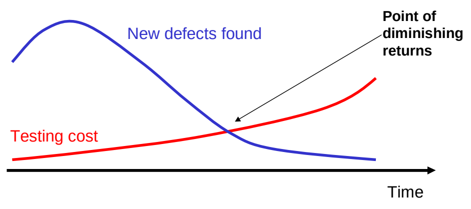
- The above measures give some insight into the effectiveness of testing
- They cannot truly measure the effectiveness
- Essential for demonstrating the mandated coverage

Testing Effectiveness


Categories of metrics for test effectiveness
Consumer satisfaction measures
• Surveys
• Help desk calls
• Referrals etc
Defects
• Found in testing
• Found by customers
• Severity, age, density and distribution
Coverage
• Requirements
• Design /Specification
• Code
Take measures to ensure that bugs are reported

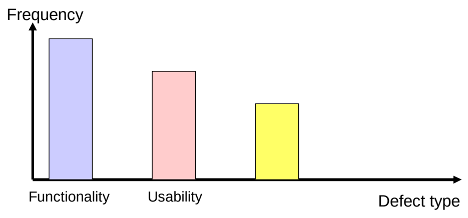
Effort / time factors

- The number of faults / failures detected must always be measured against testing effort or cost (measured in time or money)
- Typical pattern



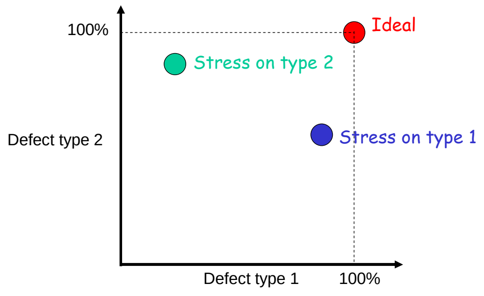
Pareto analysis

- A single metric is unlikely to characterise the state of the program
- Weighted frequency of the defects of various types can be plotted to target areas of weakness



Pareto analysis

- Risk analysis can help to direct testing efforts



Documentation

Example for requirements and design coverage

Attribute	TC1	TC2	TC3	TC4	**Coverage**
Requirement 1	X	X		X	**75%**
Requirement 2		X			**25%**
Requirement 3			X	X	**50%**
Design 1	X	X			**50%**
Design 2			X		**25%**
Design 3		X			**25%**

Documentation

Example for code coverage

Statement	Test Run			Covered?
	TR1	TR2	TR3	
A	X	X		Yes
B	X		X	Yes
C	X			Yes
D				No
E			X	Yes
Total	60%	20%	40%	80%

Practical tips

Bugs tend to cluster

- Physical clustering
  - Some modules will be particularly affected
  - Don't stop testing until the bug level subsides
- Logical clustering
  - Particular solutions tend to be coded in the same way and thus likely to have similar logical bugs
  - Having found a logical fault in one module, (re-)test all logically similar parts of code

Practical tips

Keep analysing bug detection performance of your testing team

- Whenever your testing team missed a bug
  - Analyse why the bug was missed
  - Implement a change in testing procedure so that this particular kind of bug would not be missed again

Practical tips

Testing Effectiveness metric

$$\text{Effectiveness} = \frac{\Sigma\alpha_i N_i}{\Sigma\alpha_i N_i + \Sigma\beta_j \hat{S}_j}$$

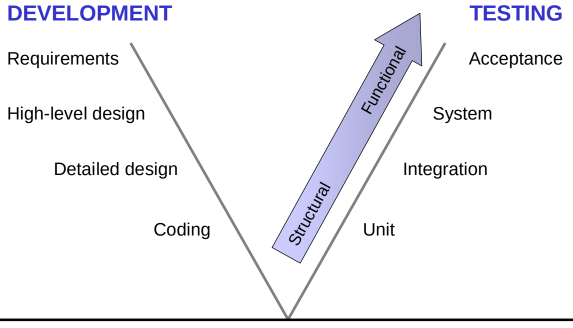
- It is useful to weight the defects by
  - Severity of their effect (Impact): α
  - Cost and time to fix: β

Practical tips

Use likelihood estimates to select test cases

- Testing for faults that are not likely to be present is not effective

Functional vs structural testing: When?



Functional vs structural testing Limitations

- Functional testing
  - Can exercise statistically insignificant portion of possible processing paths in the program
  - Weak at finding undefined variables and initialisation faults
  - Weak at testing alternative outcomes of computations (e.g. real and imaginary roots in the Quadratic Equation example)

Functional vs structural testing Limitations

- Structural testing
  - Weak at detecting logical faults

    we may be satisfied with the correct answer even though it has been incorrectly computed, e.g.
    - Implementation: a = 2 * b
    - Should be: a = b * b
    - Test case for b = 2 will not detect the problem
  - Cannot find missing parts of the program
    (it does not make use of the specification)

Functional vs structural testing

- It has been empirically shown that either type of testing, individually, can recover between 1/3 and 2/3 of defects
- It is best to use both

Next lecture

Integration and System testing

Homework

- Study the following metrics and the associated methodologies
  (Systematic Software Testing,R Craig & SP Jaskiel)
  - Surveys
  - Help desk calls
  - Number of defects found in testing
  - Number of defects found by customers
  - Severity
  - Age
  - Density
  - Distribution