COMP 3711

(OOA and OOD)

Software Testing 7

Measurements & Maintenance

(chapters 7, 8)

- *Software Test Automation,* Mark Fewster, Dorothy Graham, Addison-Wesley

Two Big Questions

- Are we building the software product "right"?

$\Rightarrow$ Verification

Specifications conformance

- Are we building the "right" software product?

$\Rightarrow$ Validation

Requirements compliance

F.I.R.S.T.

- Clean tests have the following charateristics:
  - Fast
    - Tests should run quickly
  - Independent
    - Tests should not depended on another test
  - Repeatable
    - Tests should be repeatable in any environment (e.g. QA, production, client)
  - Self-Validating
    - Tests should have boolean output (pass/fail)
  - Timely
    - Tests should be written before coding

Why Measure?

- True project status is a major problem in the software development
- Role best assumed by QA
- Measurements make it much easier to accurately determine the true status of a project
- Critical number is the cost to complete

Cost Benefit Comparison Example

	Manual Testing	Automated Testing
Cost to design test cases	$6000	$6000
Cost of tool		$5000
Cost to implement automation of test cases		$11000
Total cost of automation		$16000
Cost to execute a full cycle of test cases	$5000	$1000
Number of cycles per release	3	3
Cost of testing per release	$21000	$9000
Savings per release		$12000
Releases per year	4	4
Benefit per year		$48000
Savings per year (benefit – investment)		$32000
ROI (savings / investment)		200%

What to Measure

- Certainly not everything
- Measure what will illuminate progress toward objectives
- DDP
- Time to automate test set
- Total maintenance time
- Some measure of benefit

Some Measurable Software Indices

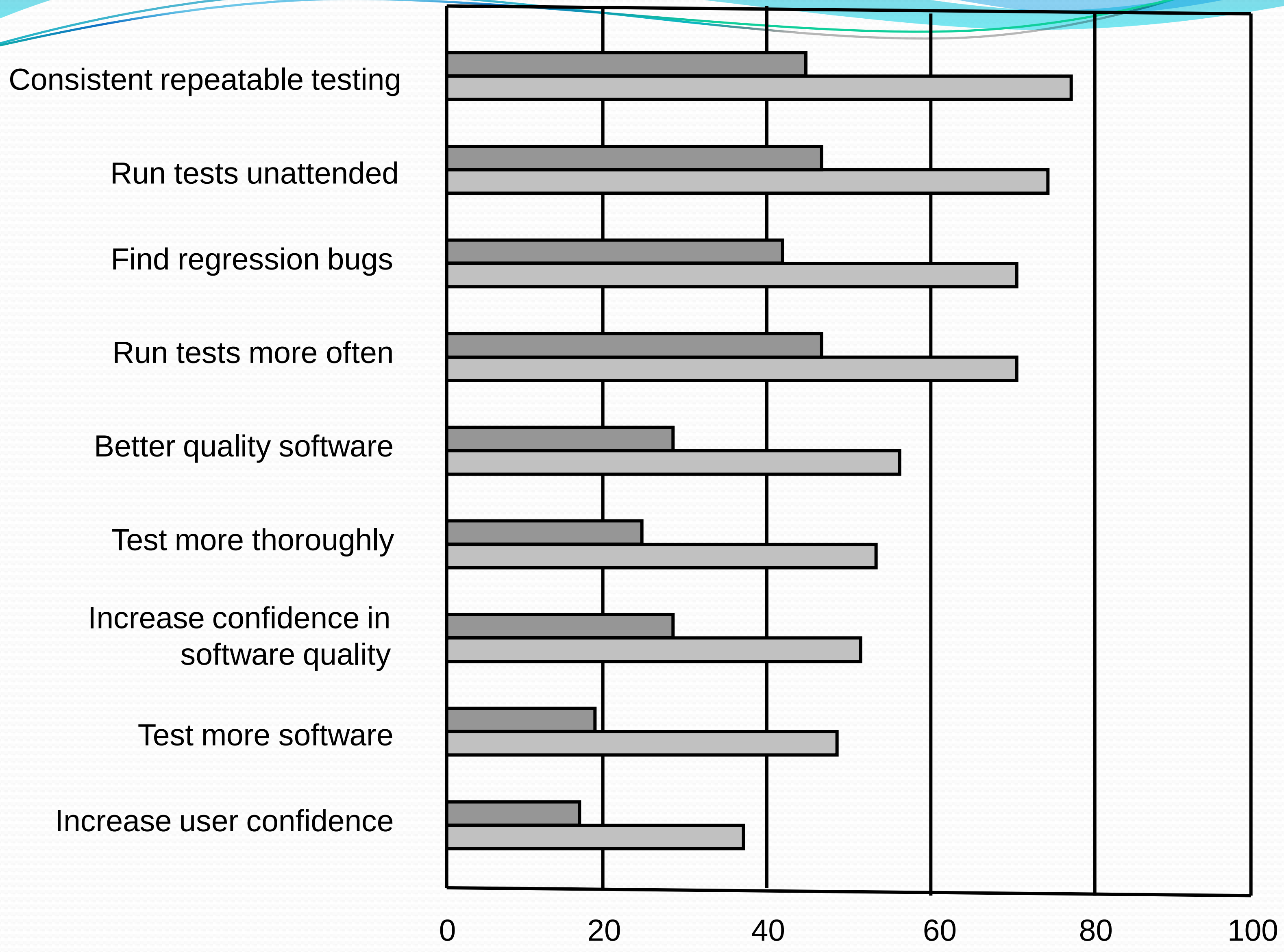
1. Lines of source code
2. Function points
3. Bytes of object code
4. Number of decisions
5. Development cost
6. Number of bugs
7. Number of developers

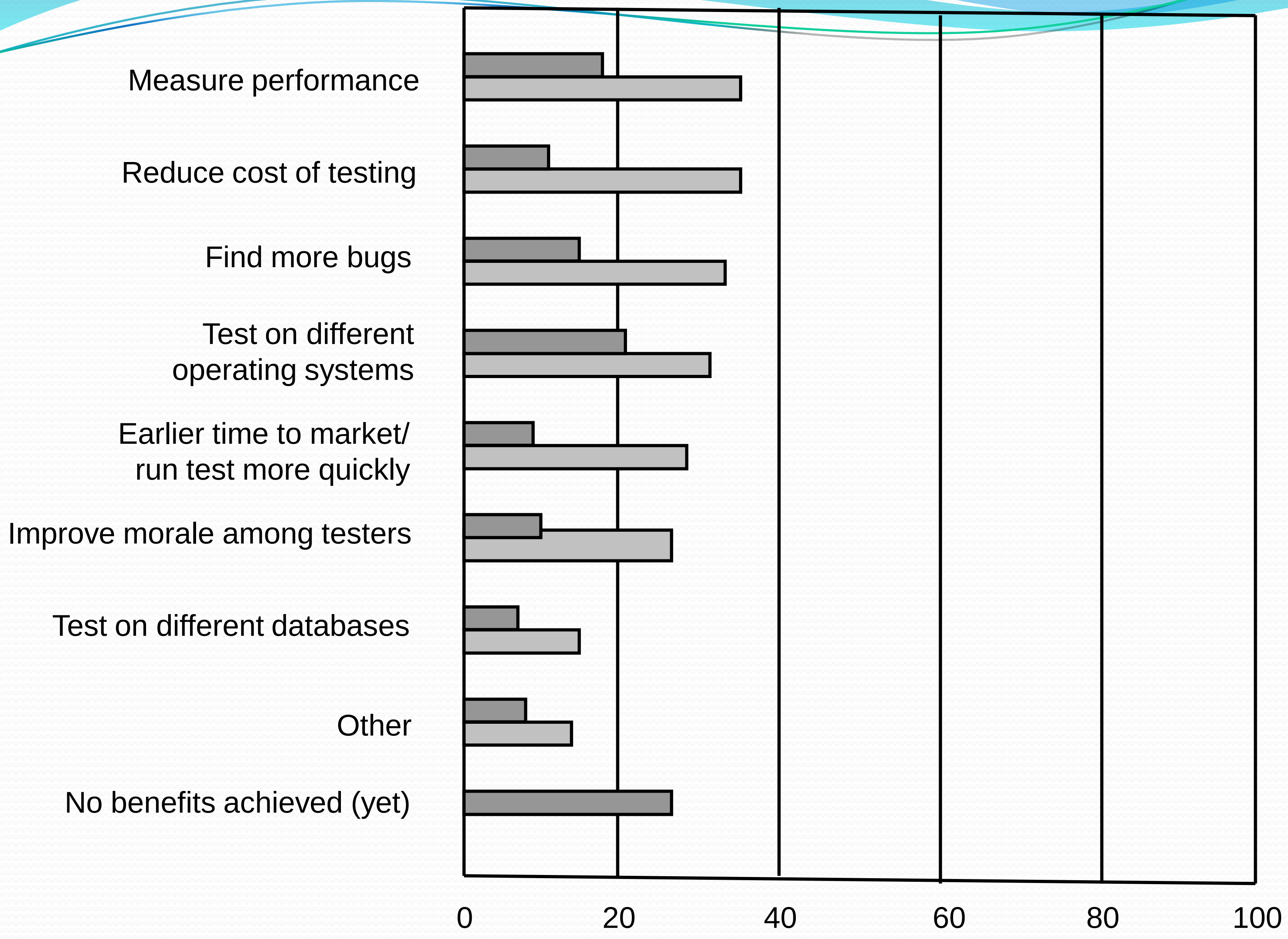
Measurable Testing Attributes

1. Number of tests in suite
2. Number of tests:
    1. Planned
    2. Run
    3. Passed
3. Time and effort spent on testing
4. Number of defects found
    1. Testing
    2. Use
5. Coverage

Measurable Test Automation Attributes

- Number of automated scripts
- Number of automated tests
- Time to run automated tests
- Time or effort to maintain the tests
- Number of test failures caused by a defect


Consistent repeatable testing
Run tests unattended
Find regression bugs
Run tests more often
Better quality software
Test more thoroughly
Increase confidence in software quality
Test more software
Increase user confidence
0
20
40
60
80
100

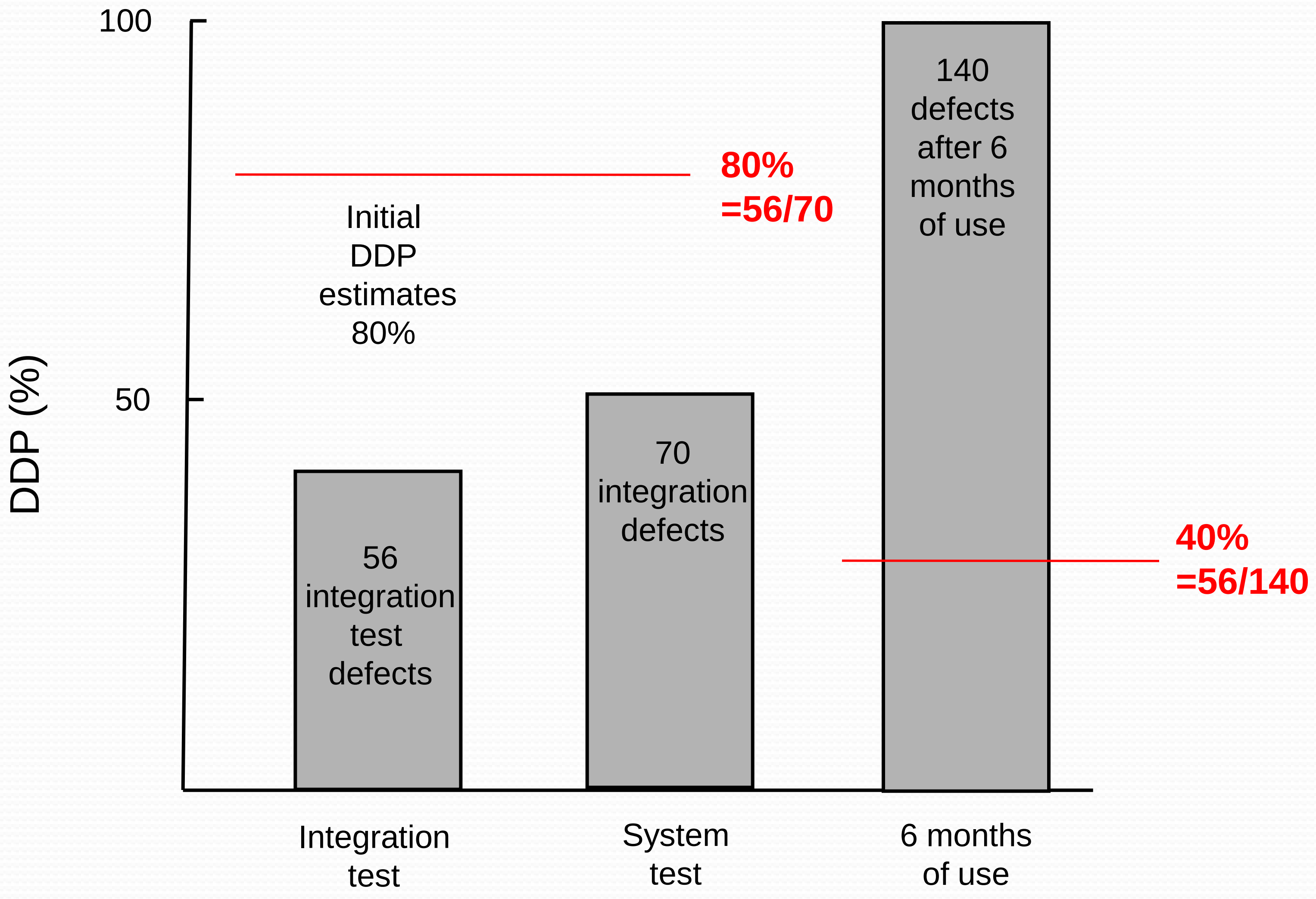

Measure performance
Reduce cost of testing
Find more bugs
Test on different operating systems
Earlier time to market/ run test more quickly
Improve morale among testers
Test on different databases
Other
No benefits achieved (yet)
0
20
40
60
80
100

Example: Measuring test effectiveness

$$DDP = \frac{defectsfoundbytesting}{totalknowndefects}$$

- DDP = Defect Dectection Percentage
- Totalknowndefects=number of defects found by this test + number of defects found afterwards.
- Measurement of how effective test process is in finding bugs
- DDP index will decline as more bugs are found in service (i.e. more effective testing captures those defects that had escaped earlier detection)

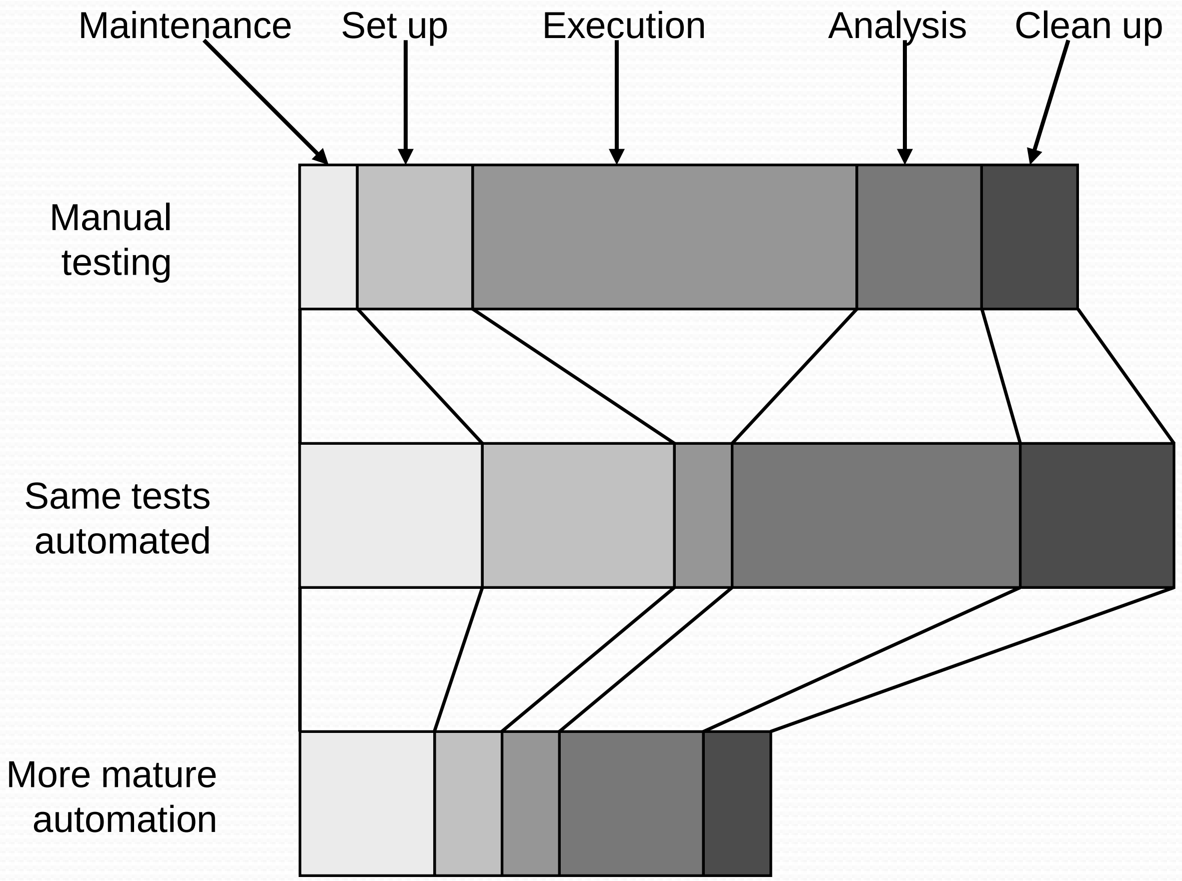
Example: DDP At Different Stages



DDP Weaknesses

- Never know full number of bugs
- Information is always incomplete
- Not all bugs are equal in significance
- Great danger of being over optimistic
- Existence of at least one fault
- Variation in testing effort
- Slow bug reporting - after the fact
- Dependent on amount of testing and amount of usage

Efficiency - Test Automation



Relationship between test activities in manual testing, early automation and more mature automation

Measuring Efficiency - How

- Elapsed time
- Effort
- Use of test elements
- Percentage of scripts used

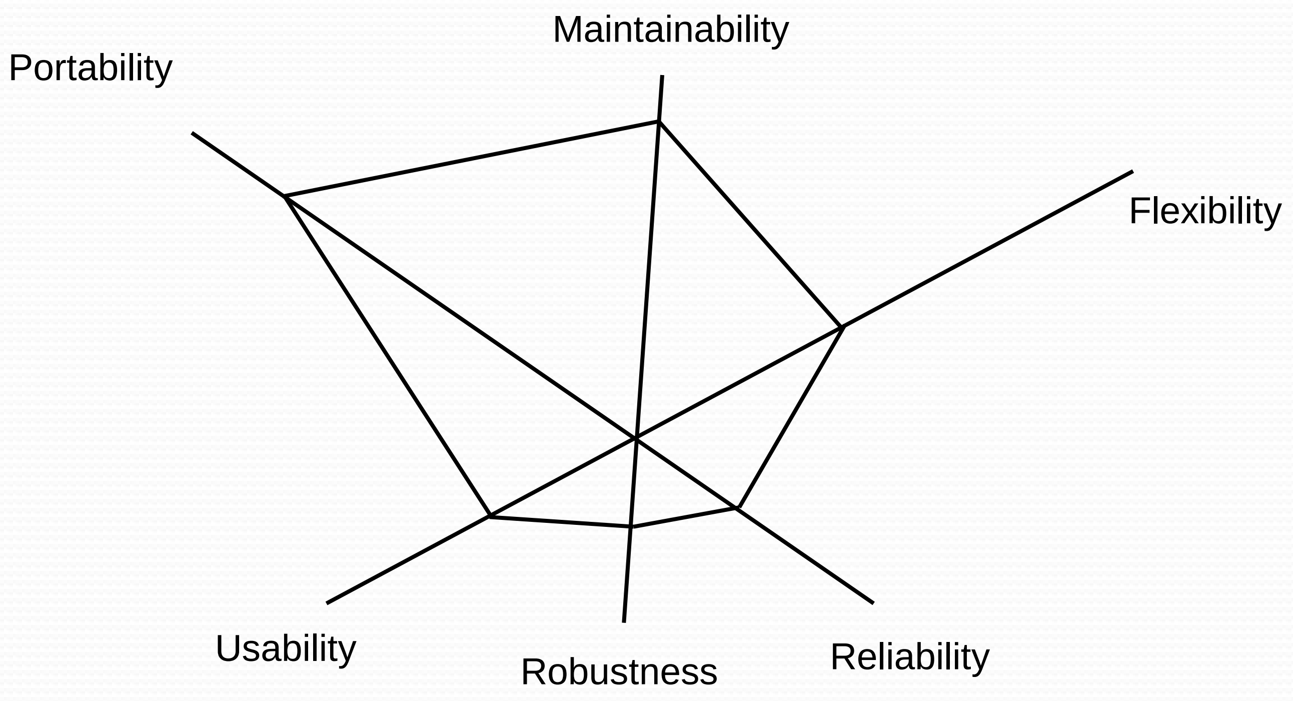
Measuring Efficiency - What

- Adding new set of automated test cases
- Setting up a set to run
- Cleaning up after running a set
- Selecting and running a subset
- Running time
- Monitoring
- Determining results
- Debugging test sets

Changes That Affect Tests

- Screens
- Business rules
- File formats
- Reports
- Communication protocols
- Simple changes in functionality
- Major changes in functionality

Various Measurements



Reliability

- Percentage of failures due to tests
- Additional test cycles needed because of failed tests
- False negatives
- False positives

Flexibility

- Time to test fixes on old releases
- Time to select specific test cases
- Selection criteria used to identify subset
- Time to retrieve test case that has been archived

Usability

- Time to add new test cases
- Time to determine results of test run
- Time to train new people
- Time to screen out irrelevant defects
- User feeling

Robustness

- Failures due to single defect
- Frequency of unexpected failures
- Mean time to unexpected failure
- Time to investigate unexpected failure

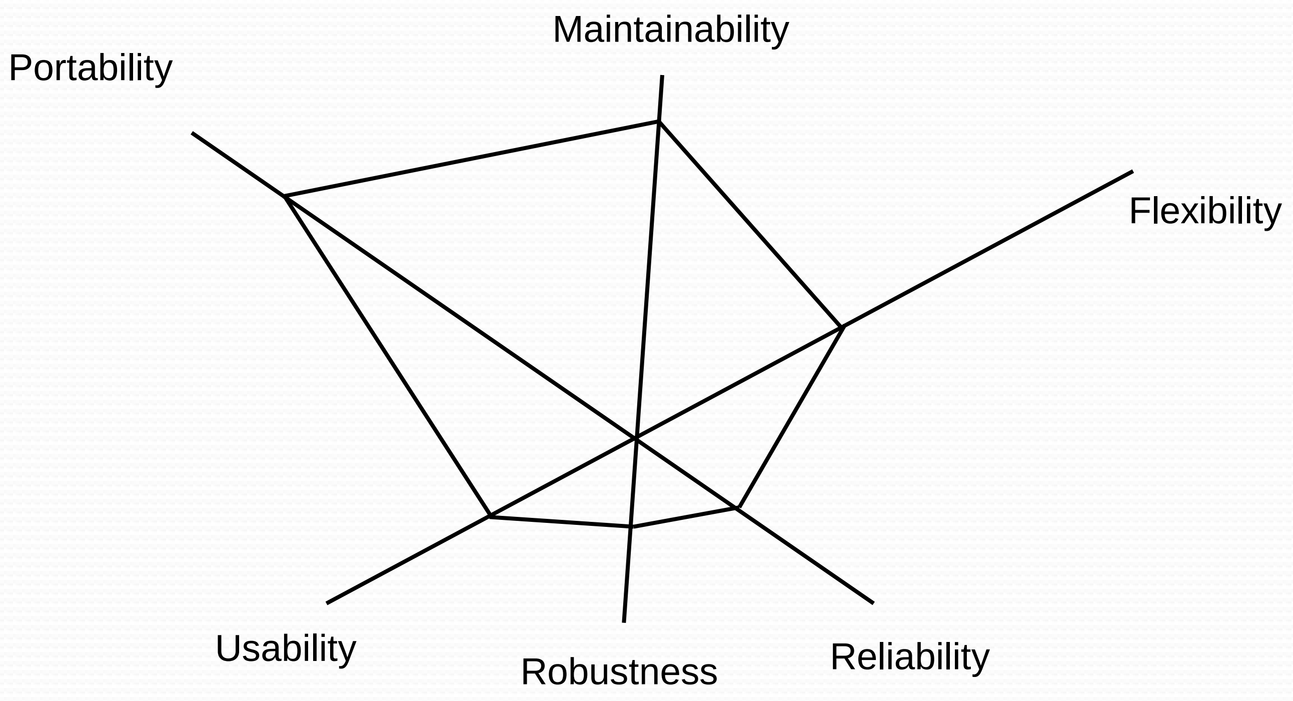
Portability

- Time to port system to new OS environment
- Time to port system to different test tool
- Number of different environments in which system runs

Maintainability

- How long?
- How often?
- Which changes require the most maintenance?
- Minimize maintenance to areas changed most frequently

Which Measurements are Most Important?



Test Maintenance

- Important but tends to be neglected
- Much more important for automated testing
    - Automated tests are much more fragile than manual tests
    - As necessary as maintenance on an application

Need for Test Maintenance

- As the software under test changes, the test system must change
- New tests must be added
- Existing tests must be modified
  - If new fields are added to a screen or a data record, the tests that handle these records or screens must be modified

Maintenance Tasks

1. Updating scripts to match new UI
2. Updating test data to match new formats
3. Updating expected results
4. Updating comparisons
5. Analyzing and correcting causes of test failures

Antipatterns in Test Maintenance

1. Idea
   - Appears to be good initially
2. Problem
   - Manifests itself in the long term
3. Solution
   - What can be done to address the situation

Number of Test Cases

- Idea
  - More test mean better coverage, fewer bugs and higher software quality
- Problem
  - More tests mean a higher maintenance burden
  - Uncontrolled growth in the number of tests can result in tests that cost more than they create in value

- Solution
  - Think before adding tests
  - Make sure each new test adds something to the team
  - Periodically review and weed out tests
  - Difficulty is in accurately assessing the benefits of a particular test.
  - Not quite as difficult to assess cost of a particular test

Quantity of Test Data

- Idea
  - More is better, results in more complete testing
- Problem
  - Managing the data takes more effort
- Solution
  - Limit the disk space a test case may use
  - Think about how much data is required to test a particular area
  - Restrict large datasets to area requires it

Format of Test Data

- Idea
  - Always store test data in the format required by the application
- Problem
  - Specialized tools may be required to update this data
- Solution
  - Store test data in common/standard format
  - ASCII text if possible
  - Convert as required to do testing

Test Case Running Time

- Idea
  - Long test save on set up and clean up time
- Problem
  - If a test must be modified, the time required is not a linear function of the test case length
  - Longer tests are harder to analyze
- Solution
  - Keep test cases short and focused

Debugability of Test Cases

- Idea
  - We only need to know whether test case passed or failed
- Problem
  - If a test case fails, it may be a problem with the test case
  - If not then there is the problem of figuring out what went wrong with the application

- Solution
  - Design with debugging in mind
  - More information will make fixing the test case easier
  - Anticipate failures and what information would be required to pin down and analyze them

Dependencies Between Tests

- Idea
  - Run many test one after the other using output from one as input to another
- Problem
  - If one has to be modified, it impacts other test cases
  - The domino effect
- Solution
  - Make test cases as independent as possible
  - Use the sausage process with caution

Naming Convention

- Idea
  - Not needed, slow down and limit creativity
- Problem
  - As volume of test cases grows it becomes more difficult to identify items
- Solution
  - Adopt a convention at the start before it becomes a problem
  - Have a "good" naming convention

Test Case Complexity

- Idea
  - With automation complex test cases are now possible
- Problem
  - Complexity restricts understanding
- Solution
  - Complexity only as necessary, otherwise, keep it simple

Test Documentation

- Idea
  - No need to document scripts as only the computer reads them
- Problem
  - To make changes people have to read them
- Solution
  - Maintain at least a minimal level of documentation

Detailed Test Specification

1. Test name
2. Test purpose
3. Test method
    - Detailed enough to do manual test
4. Pass/Fail criteria

The Trick

1. Tools could lead you do the wrong thing
2. Taking the easy way results in high maintenance costs
3. Initial enthusiasm wears off after making a mess
4. Benefits back end loaded

Strategy

- Identify attributes that are going to have the largest impact on test maintenance
- Do something to reduce the impact of each one
- Need to track where effort is spent on test maintenance

Tactics

1. Define preferred values and standards
2. Provide tool support
3. Automate updates
4. Schedule periodic weeding
5. Maintenance utilities

Maintenance Timing

- Best done on a regular systematic basis
- Can be started based on accepted changes to the application
- Built entirely new scripts each time a new release of the product required testing

Maintenance Costs

- Happen every time the software is released
- Much more critical than the original cost to create the test
- Record and playback makes test creation cheap at the expense of high maintenance costs
- Scuttled more than one automation project

Test Requirements for Low Maintenance

- Maintainable
- Modular
- Robust
- Well documented
- Reusable