Improving testing Efficiency using Cumulative Test Analysis

Ian Holden

Purpose

- Introduce the concepts of CTA
- Show prototype implementation
- Share experiences
- Encourage future involvement

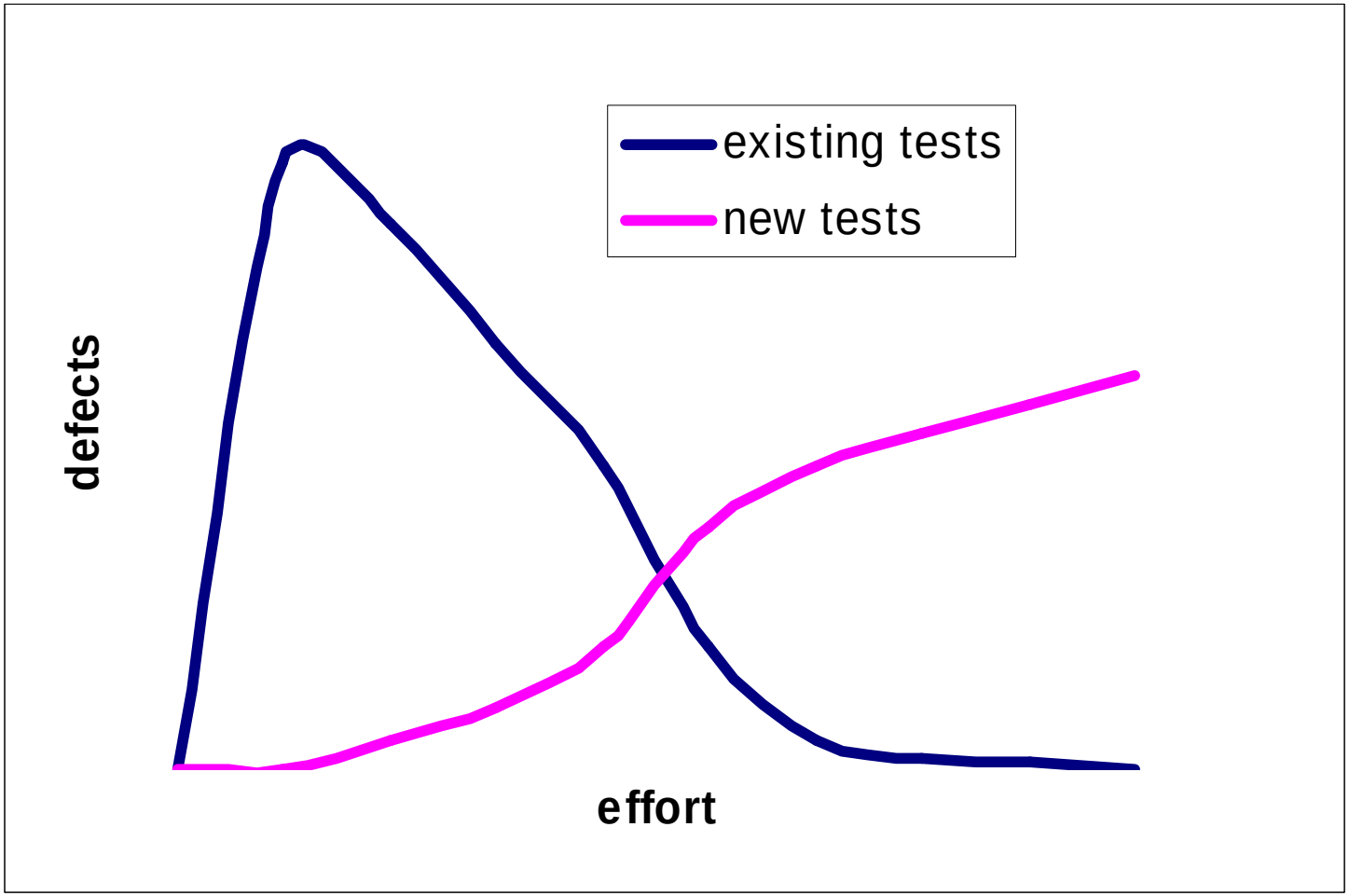
Introduction – the problem

- Impossible to fully test
  - Large number of tests
  - Large number of environments
  - Long setup/execution times
  - High build rate
  - Legacy tests not well understood
  - Automation
    - Setup / teardown times and problems
    - Demoralising work

Introduction - need

- Practical technique/tool
- Flexible and adaptable
- Run a prioritised set of tests
  - Target the areas at most risk
  - Select the best tests
  - Best use of time available
    - Run as few as necessary
    - Free up resources
- New mind set (away from 100% pass)
  - Test effectiveness
  - Product quality
  - Risk of customers experiencing problems

Balancing act

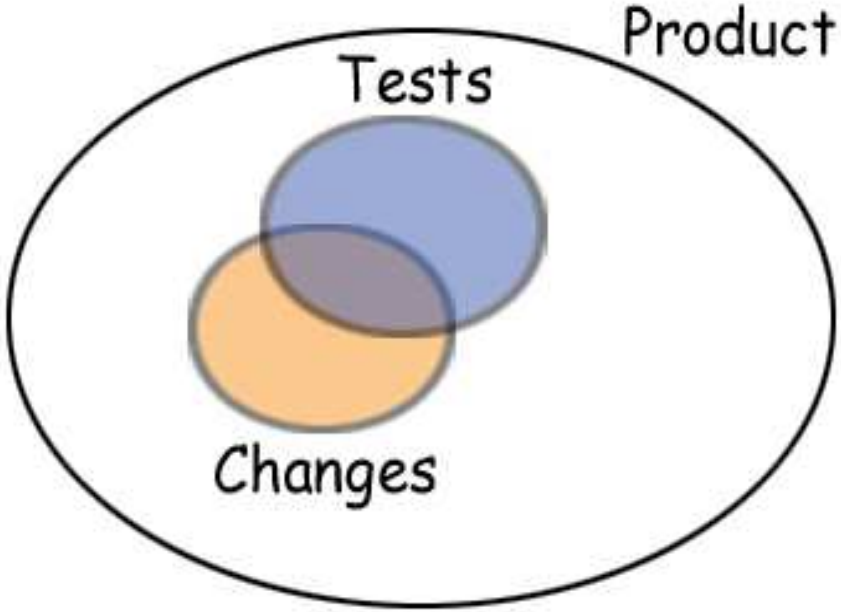


- Endless testing vs improving tests

Basic idea

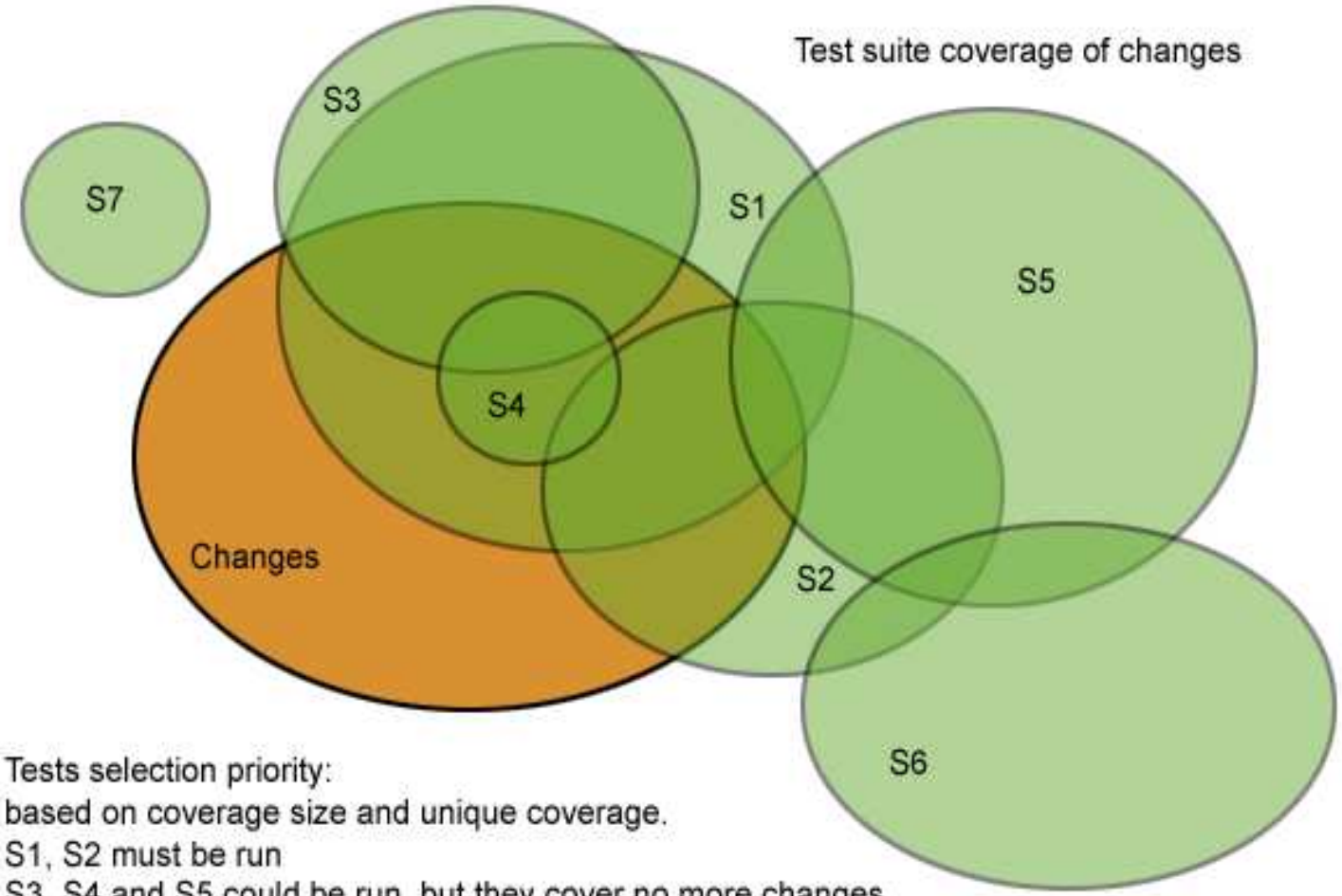
- Choose a baseline/reference build
- Target tests on untested changes since the reference
- Factor in risk, test effectiveness, test coverage, environments
- Accumulate results
- Produce useful reports for testers and managers

Targeting info



- Basic information for targeting
  - Code coverage for tests.
  - How good are the tests?
  - What is the impact of each change?

Test Suite Selection



Tests selection priority:
based on coverage size and unique coverage.
S1, S2 must be run
S3, S4 and S5 could be run, but they cover no more changes
others do not need to be run

Identifying Risk Areas

- Component at risk of containing or exposing a defect
- A change
- Typically a class or method
- Code coverage data obtainable
- External dependency change

Assigning Impact to Risk Areas

- Impact – an estimate of the probability of a serious defect in a single risk area
  - Introduced by developer
  - Found by customer
  - Likely severity of problem
- Use defaults: per component/package
  - defect rates, complexity, importance …
- Other change specific considerations
  - Developer skill, size of change, dependency analysis …
- Change increases, testing decreases
- Assess each factor as a probability and combine

  $I = P(A \text{ and } B \text{ and } C \ldots)$

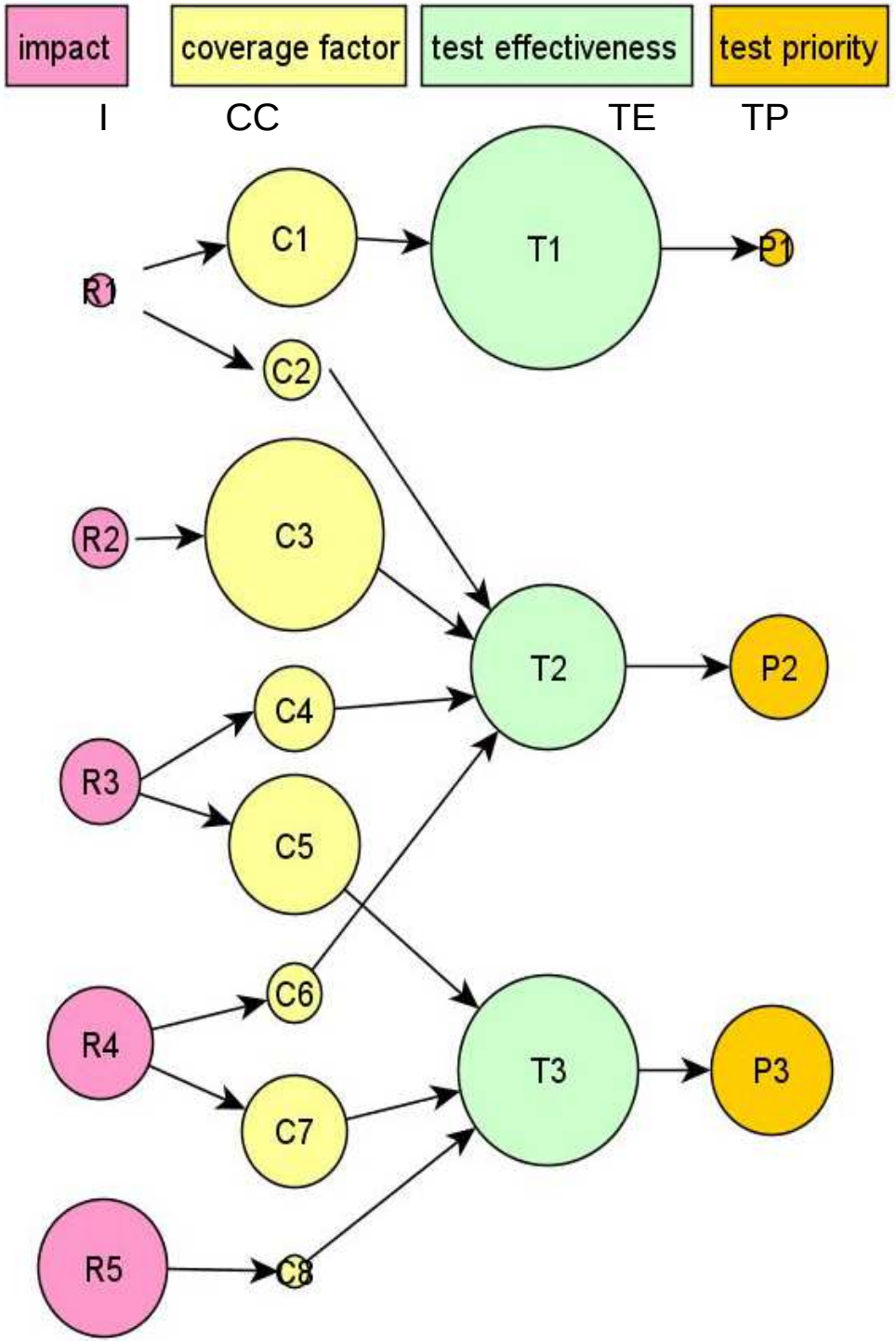
Evaluating Test Effectiveness

- Probability of running the test successfully and completely
- Can we trust the test results?
- Analysis of previous run history
- Consider age of test
- Early runs – lower
- Long time since last run – lower

…

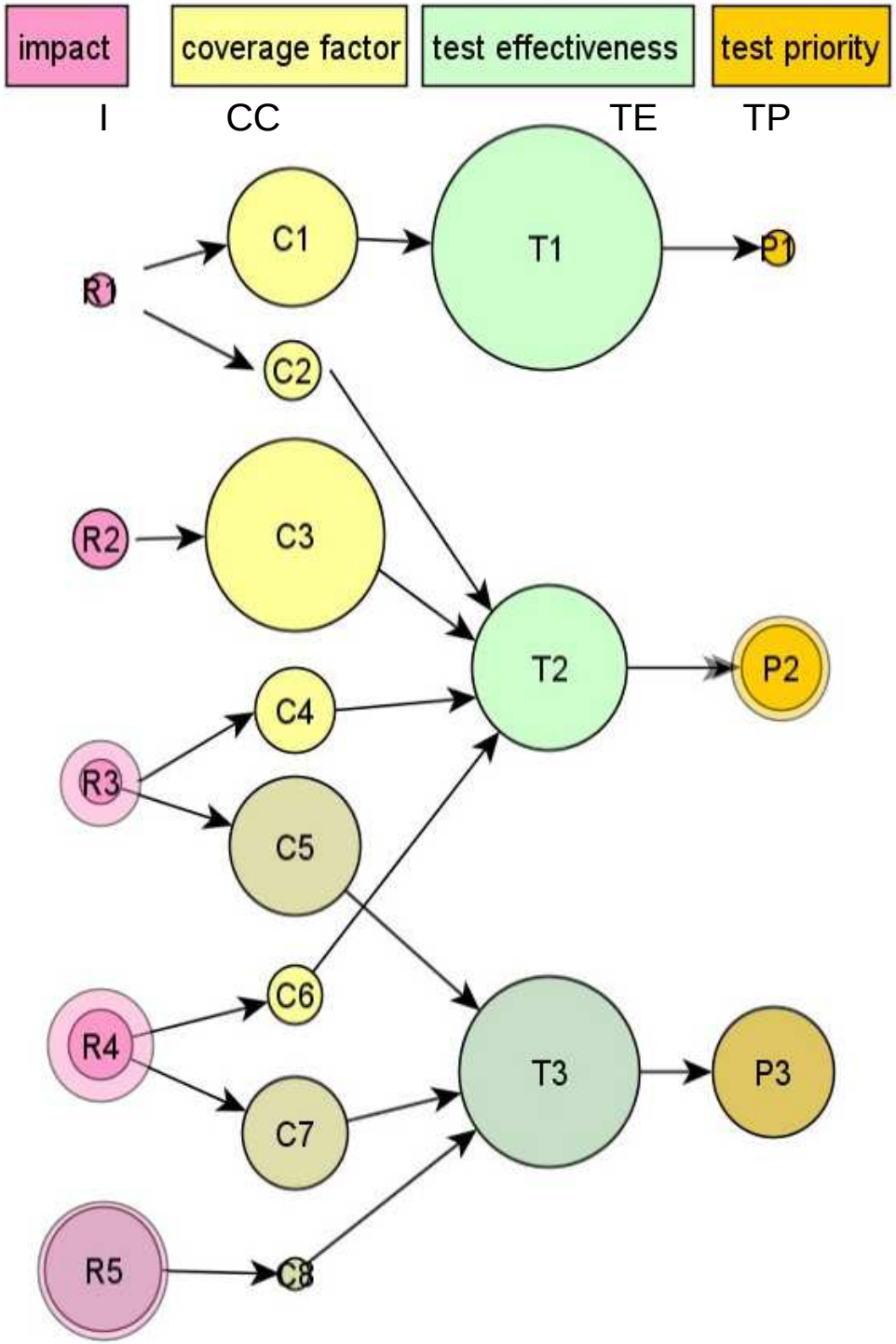
Selection of Tests to Run

- I = probability of serious defect in single risk area
- I*CC = probability of each (perfect) test finding defect in a risk area
- Combine probabilities to find TD for a test over all risk areas it covers
- Factor in test effectiveness. TP = TE*TD to get the probability of this test finding a defect. Use this to prioritise.

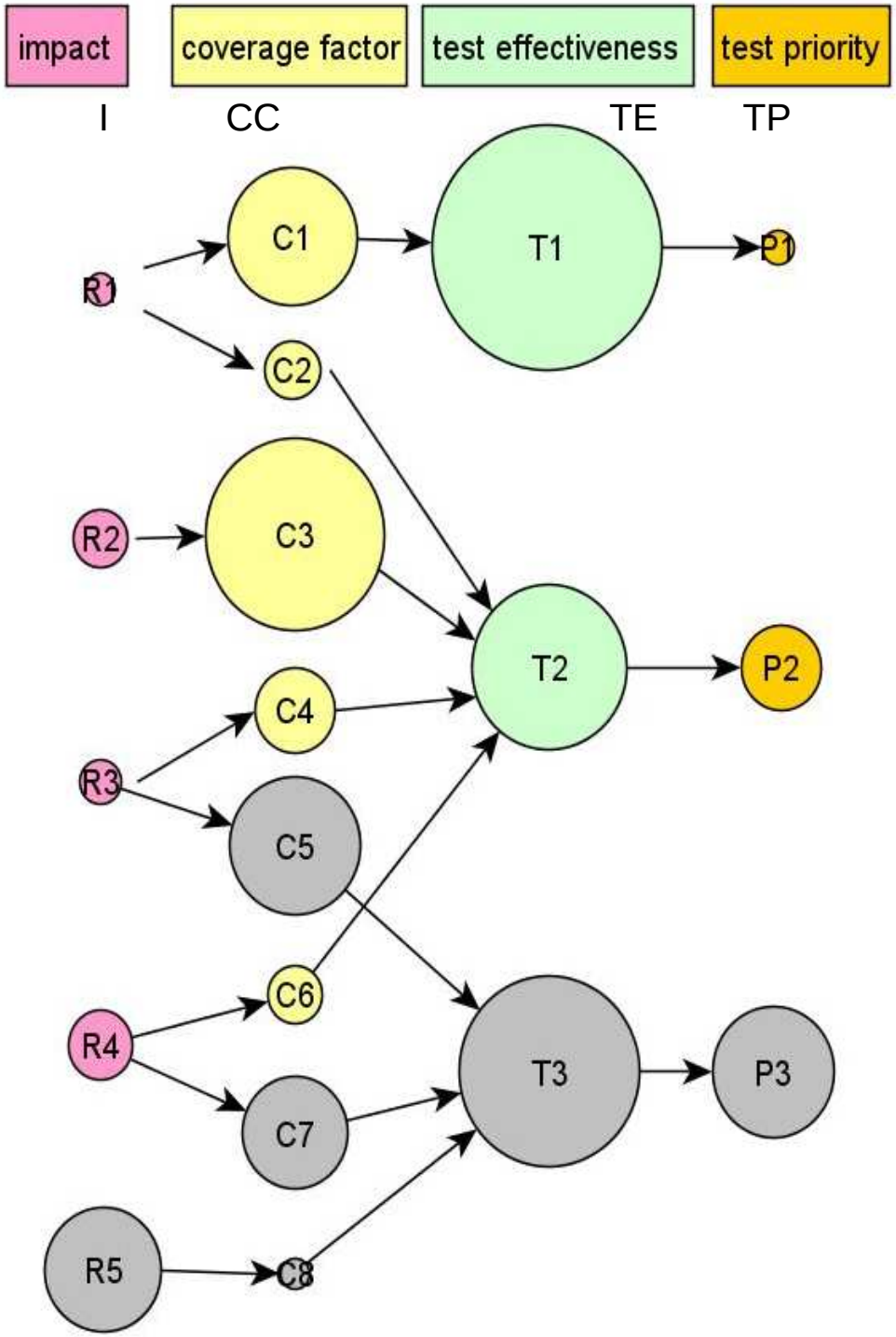


Simple example

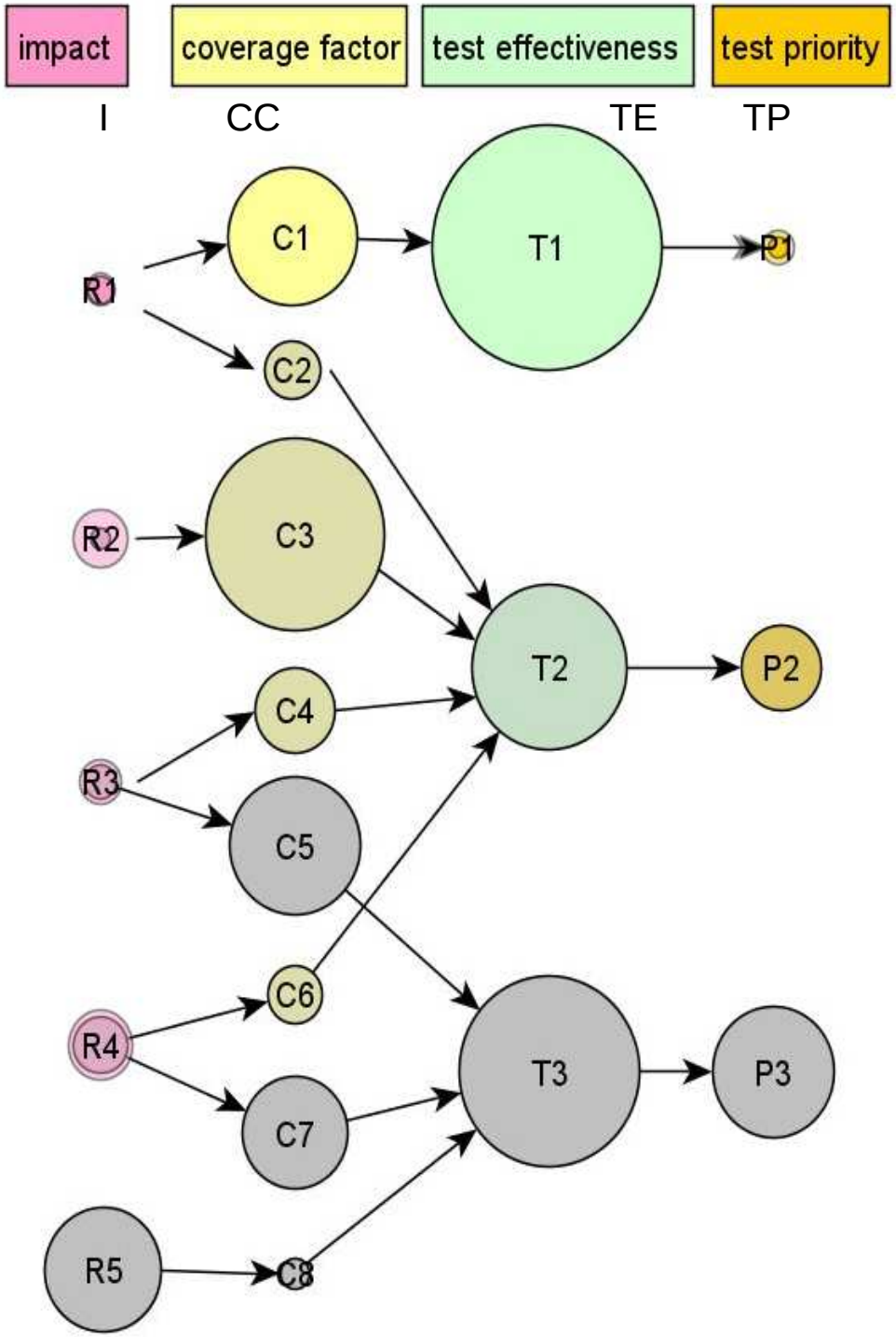
- Sizes : values
- T3 has highest priority (P3)
- Now reduce impact of R3, R4 and R5 to allow for the effect of running the test

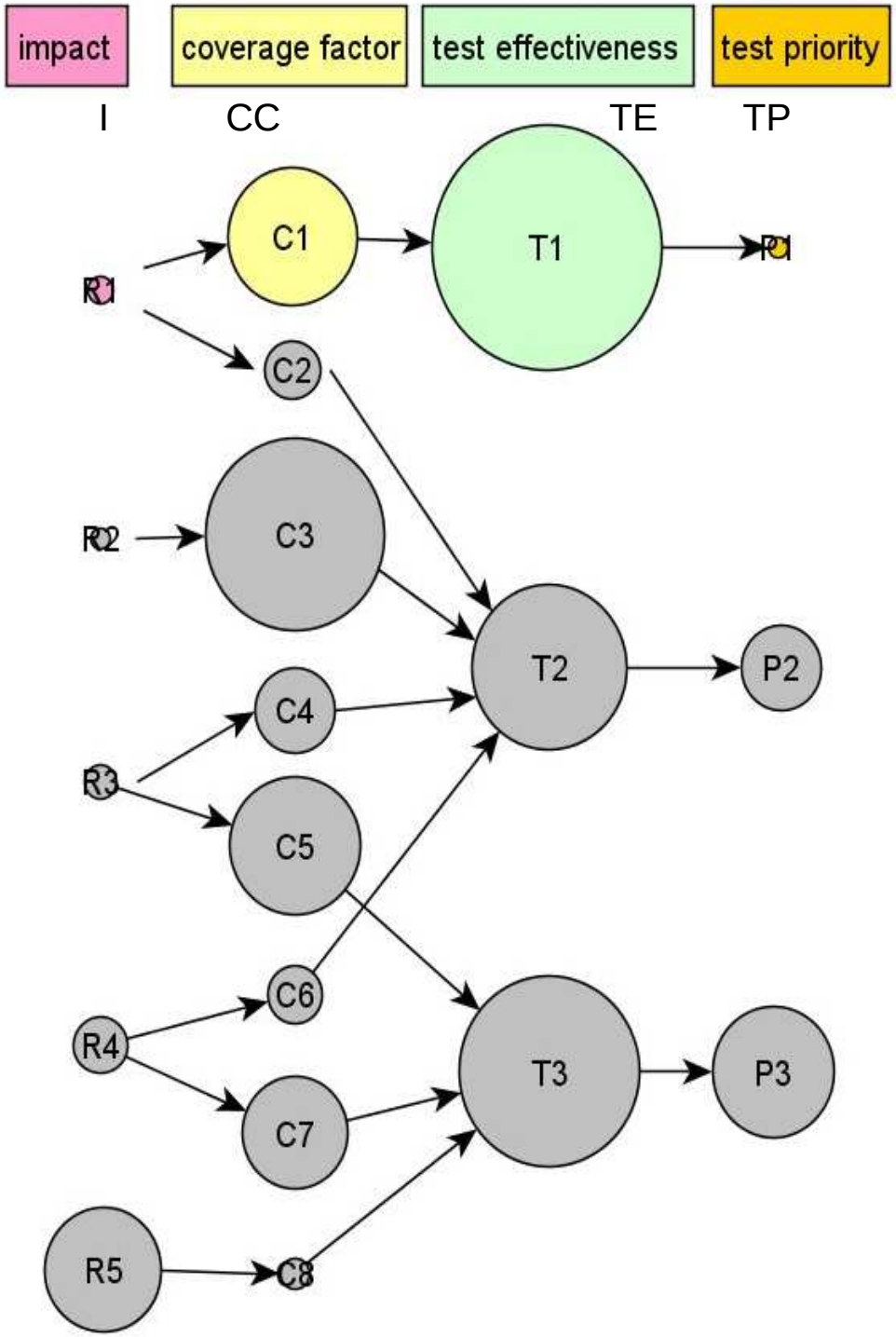


- I = I – CC*TE for each risk area covered by the chosen test



- Now remove T3 and repeat the process
- T2 is the next test


impact
coverage factor
test effectiveness
test priority
I
CC
TE
TP
C1
T1
P1
R1
C2
C3
R2
T2
P2
C4
R3
C5
C6
R4
T3
P3
C7
R5
C8



- Finally select T1
- Test priorities typically reduce rapidly
- they represent the probability of the test finding a serious defect
- To minimise the set of tests to run, choose a priority threshold.

Handling Test Environments

- Hardware and software configurations e.g. OS, database etc.
- Prioritise
- Select environment for test
  - 1. environment specific test
  - 2. highest priority environment
- Re-prioritise environments
- Minimal approach, better techniques?

Accumulating Results

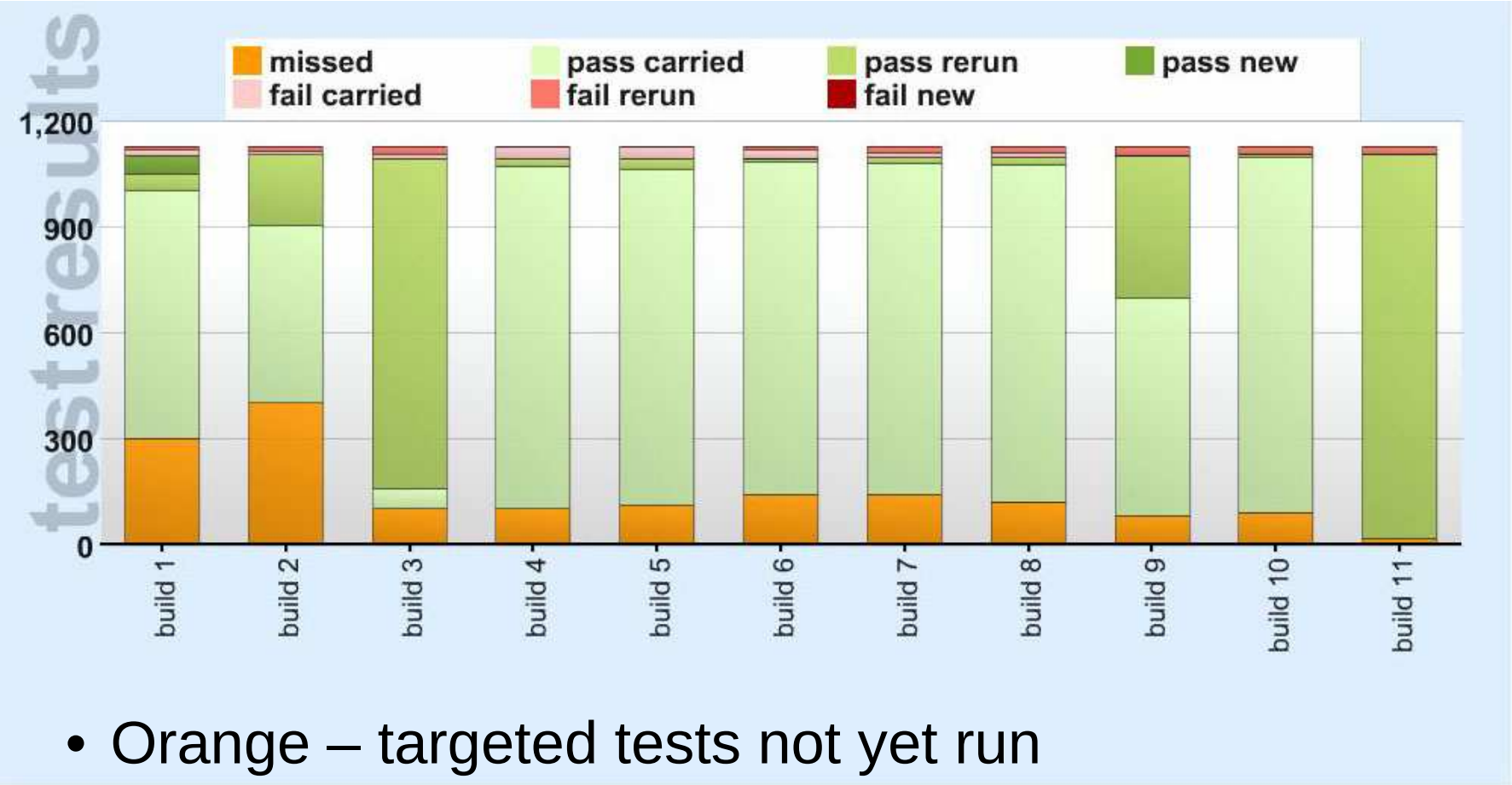
- Maintain knowledge over builds of:
  - What needs testing
  - What tests have been run
- Accumulate:
  - Un-tested changes (risk areas)
  - Test results when no re-run is required
- Each build:
  - Reset Impact for new risk areas
  - Remove test results for targeted tests
  - Replace results for all tests run

Calculating Build Quality

- A function of:
  - Cumulative test results
  - Test effectiveness
  - Remaining risk (combination of remaining Impact for all risk areas)
  - Proportion of environments tested

    …

Traditional - Cumulative Results

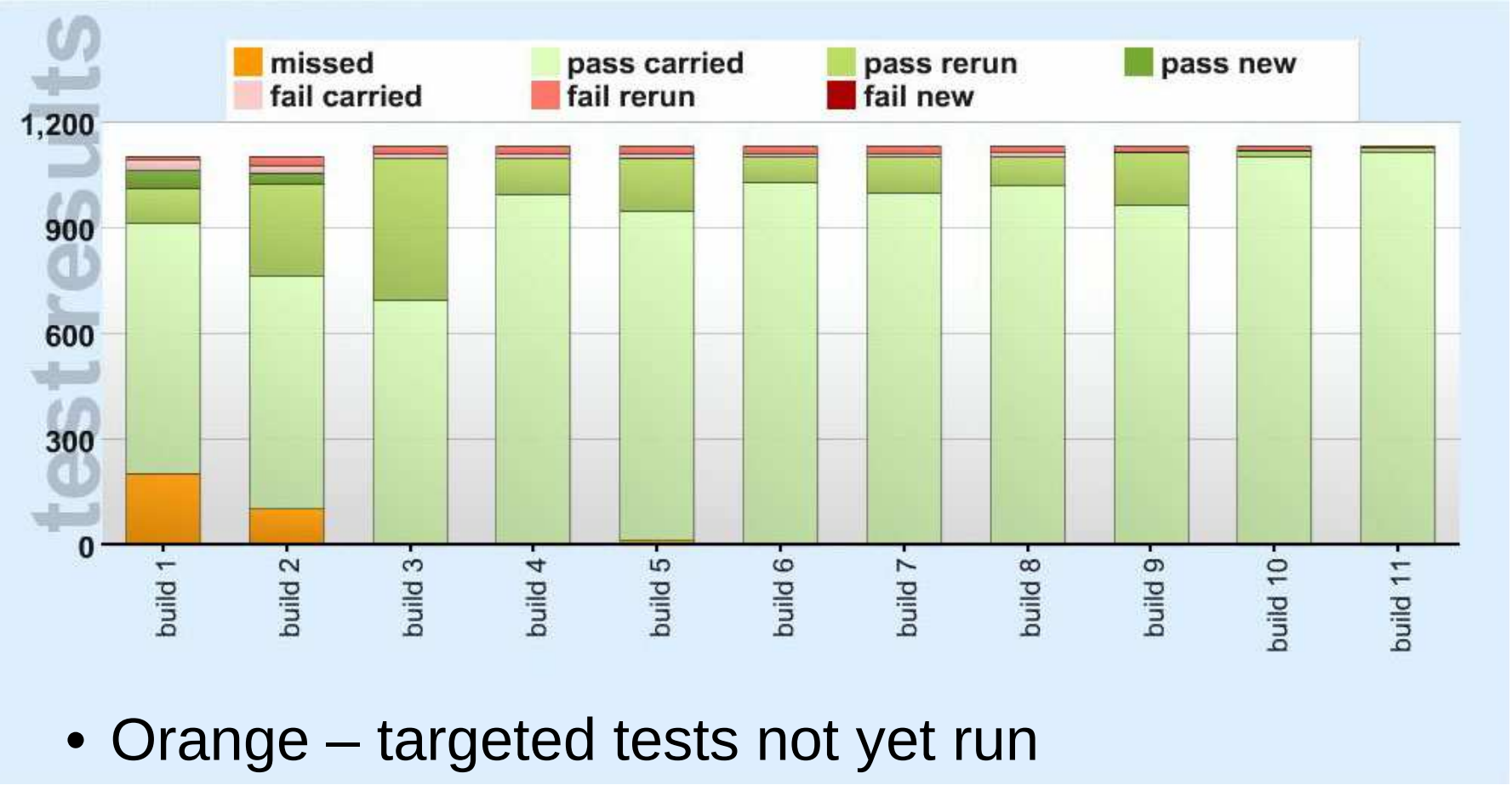


- Orange – targeted tests not yet run
- New – first results of running a test
- Rerun – results or re-running a test
- Carried – results carried over from a previous run

Traditional - Changes versus testing completed

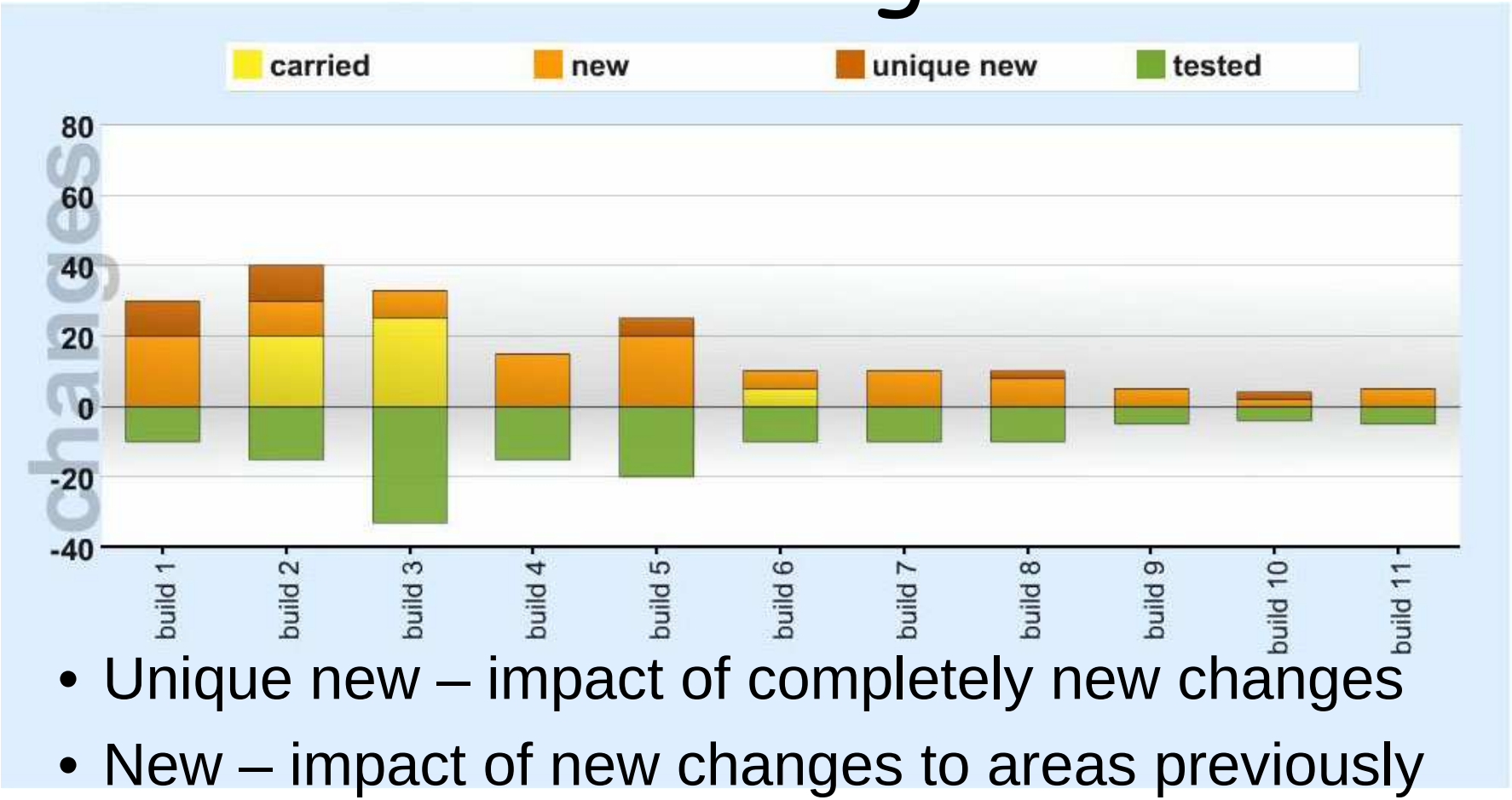
- Unique new – impact of completely new changes
- New – impact of new changes to areas previously changed
- Carried – impact of untested changes carried over
- Tested – the amount of impact tested in this build

Using CTA



- Orange – targeted tests not yet run
- New – first results of running a test
- Rerun – results or re-running a test
- Carried – results carried over from a previous run

Earlier and better coverage of the changes



- Unique new – impact of completely new changes
- New – impact of new changes to areas previously changed
- Carried – impact of untested changes carried over
- Tested – the amount of impact tested in this build

Summary

- Focus on risk areas and their impact
- Match tests to risk areas (code coverage)
- Prioritise and minimise tests to run
- Accumulate results
- Benefits
  - Shortens time to find bugs
  - Better understanding of test effectiveness
  - More time to develop new tests