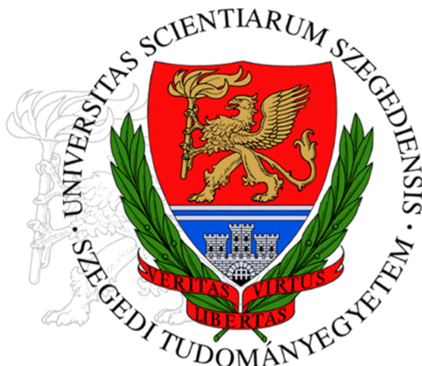
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

UNIVERSITY OF SZEGED

Department of Software Engineering

Fault Detection and Localisation in Reduced Test Suites



Árpád Beszédes

University of Szeged, Hungary
Department of Software Engineering

The 29th CREST Open Workshop, London November 2013

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Overview

- University of Szeged, Department of Software Engineering
- Part I – Redundancy in testing
- Part II – Research on fault detection and fault localisation. Test reduction

Redundancy in testing

Avoiding redundancy is the basis for practical testing

- Testing is about partitioning the testing space based on various assumptions

Yet, there are many repetitions

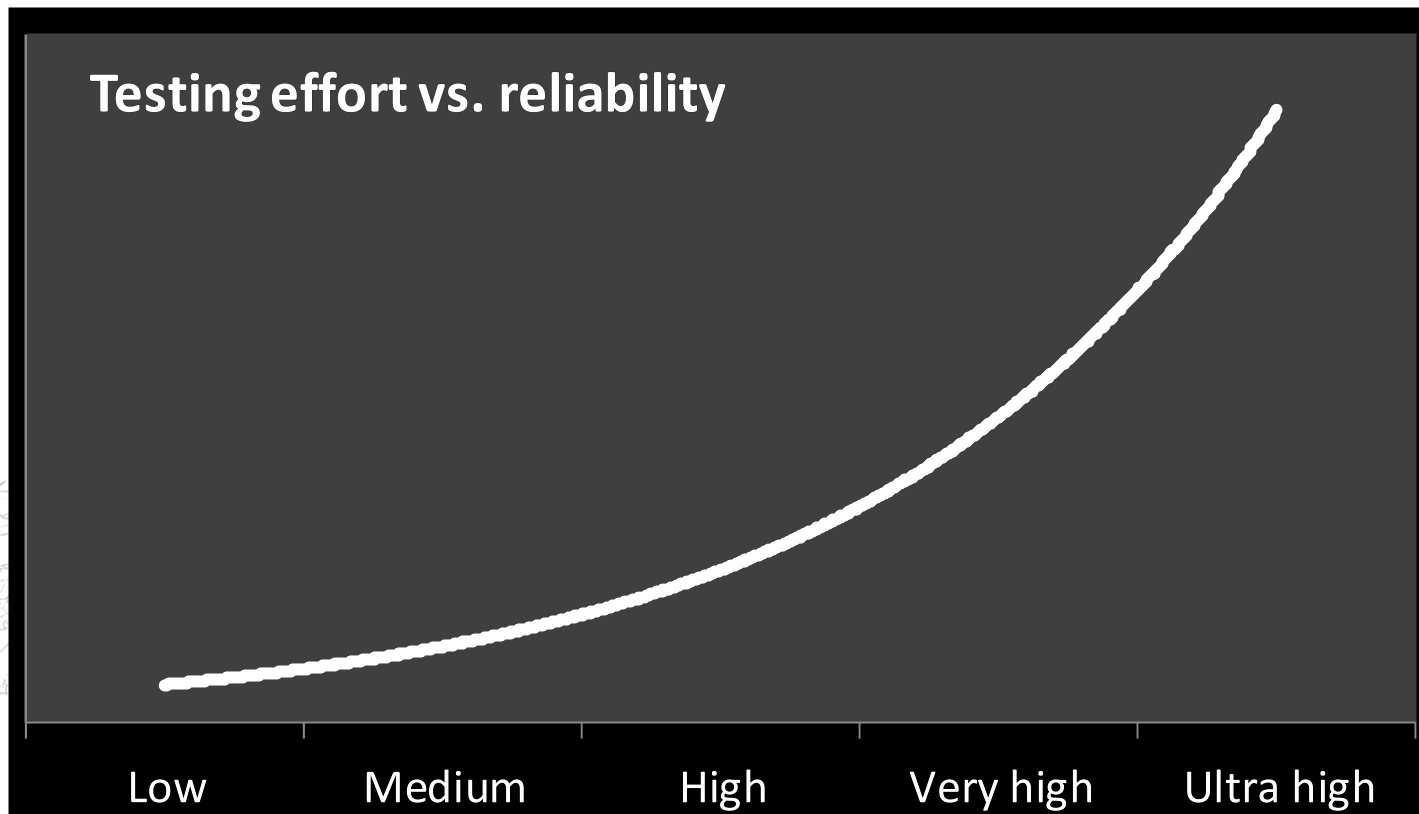
- Test automation, regression testing, etc.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Goal of testing

To find (as many as possible) bugs

Using (as least as possible) effort



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

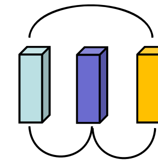
Exhaustive testing is impossible

- Theoretically, each execution is unique
  - You wouldn't try all 2^64 combinations of summing up two integers
  - Try out the "interesting" cases
- Testers are practical: don't do "redundant" testing because that will cost much and will bring little value
- But what are the "interesting" cases?

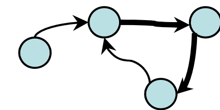
Testers make assumptions

- Equivalence partitioning

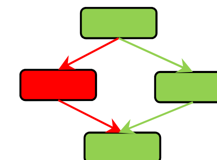
- Pairwise testing



- n-switch state-based testing



- Statement / branch / path coverage



- Etc.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Repetitions in testing

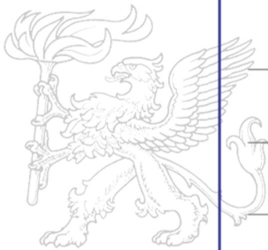
- What is a unique/redundant test case?
- Definition phase:
  - Two test cases with common steps
  - Test execution procedure or script
  - “Data/keyword-driven” testing
- Execution phase:
  - Confirmation testing
  - Regression testing

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Regression test selection

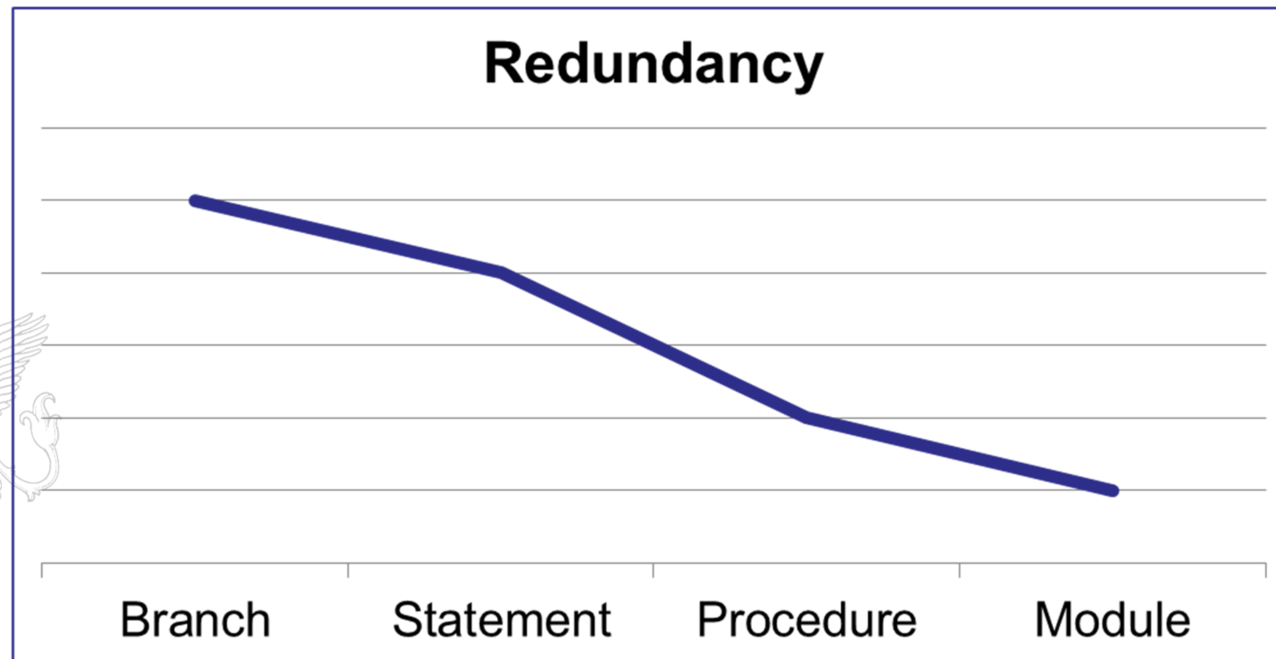
- If used properly, there should be many regression tests
  - Test suite increases and evolves with the system
- Test selection based on changes and/or risks
- When are two regression tests redundant?

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering



White-box testing

- If test cases exercise the same statements/paths/functions are they redundant?
  - Probability decreases as we relax coverage criterion



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Some redundancy data

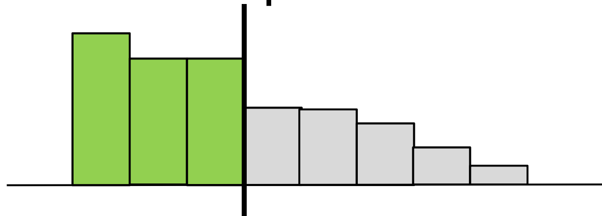
Program	kLOC	# TCs	Unique statement-level coverage vectors
space (SIR)	6.2	13 570	88%
tcas (Siemens)	0.1	1 608	4.9%
tot_info (Siemens)	0.5	1 052	31.8%

Program	kLOC	# TCs	Unique procedure-level coverage vectors
space (SIR)	6.2	13 570	30%
tcas (Siemens)	0.1	1 608	0.4%
tot_info (Siemens)	0.5	1 052	0.6%
GCC	6 200	128 230	44.6%
WebKit	4 500	21 987	88.4%

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

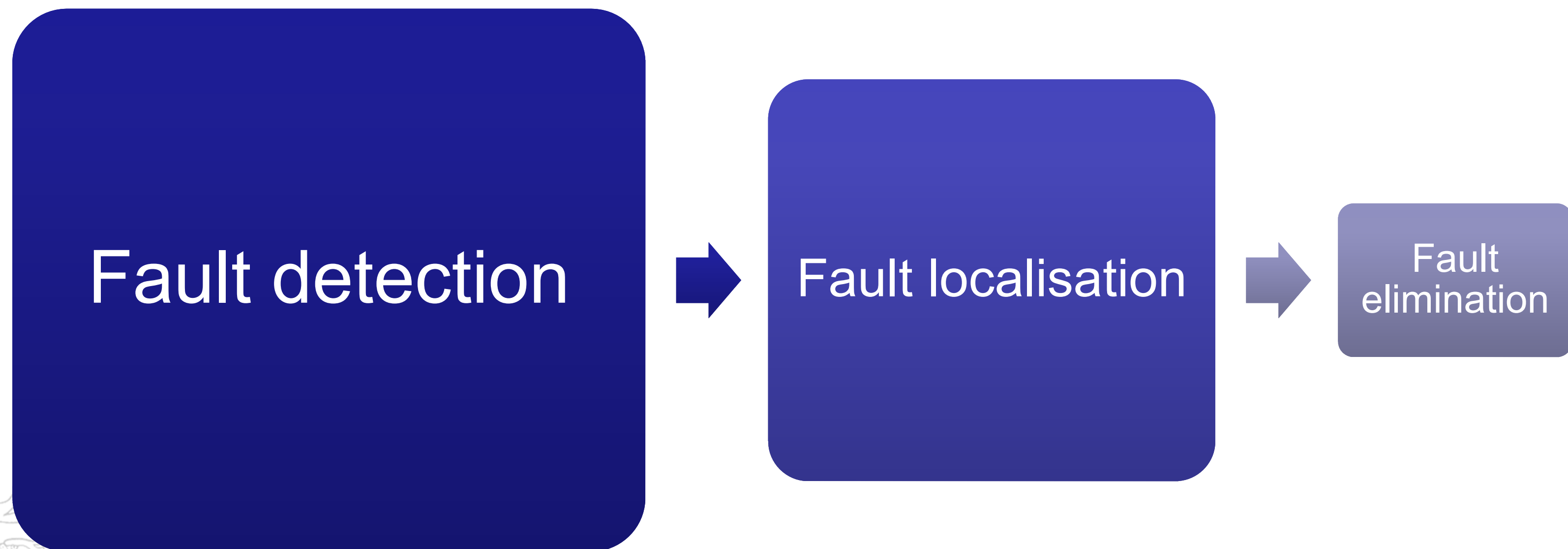
Test reduction

- Test suite reduction
  - Find a representative subset satisfying certain properties of the full suite
  - Also called minimisation (finding the minimal subset is NP-complete)
  - Either permanently or temporary eliminate the test cases
- Test case prioritisation
  - Finding the optimal sequence permutation
- Test case selection: Which test cases to use instead of the full suite?
  1. Satisfy the desired property (e.g. 100% statement coverage)
  2. Select the first N in the prioritised list



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Detect-fix lifecycle



Fault detection and -localisation

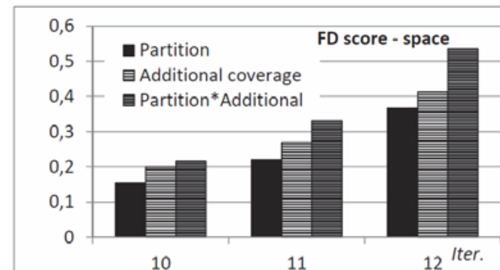
- Fault detection:
  - Use failing tests to shed light on defects
  - Prime task of testing
- Fault localisation:
  - Diagnose the actual cause of the defective behaviour, i.e. pinpoint program elements containing defects
  - Prime task of debugging
- Fault detection capability:
  - ***~high (code) coverage***
- Fault localisation capability:
  - ***~???***

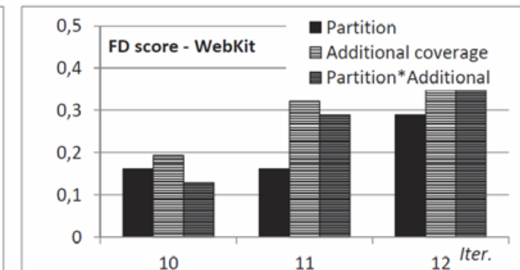
Test reduction

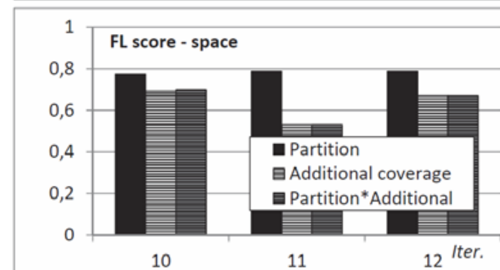
- Traditionally investigated in the context of fault detection
  - For instance, aim at maximal code coverage
- However, this strategy could be bad for fault localisation

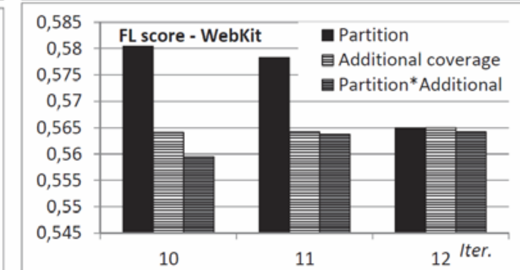
print_tokens	4.934
print_tokens2	4.597
replace	4.747
schedule	8.805
schedule2	6.081
space	0.024
tcas	6.854
tot_info	4.895
Summary	4.767

Increase in fault-localisation expense %
(Yu, Jones and Harrold. An empirical study of the effects of test-suite reduction on fault localization, ICSE '08)









FD=fault detection rate, FL=fault localisation rate
…for different reductions sizes (1k, 2k and 4k test cases)

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Fault localisation using Spectra

- "Spectrum-based" approach
  - Spectrum: signature of program behaviour on the test cases (count or hit)
  - Idea: look at which tests fail/pass and compare this to the spectrum
  - Program elements that usually produce failing tests when executed and passing tests when not are "suspicious"

$$CFM(p_i) = \begin{cases} 1 & \text{if } \mathbf{C}(p_i) = \mathbf{e}, \\ 0 & \text{otherwise.} \end{cases}$$

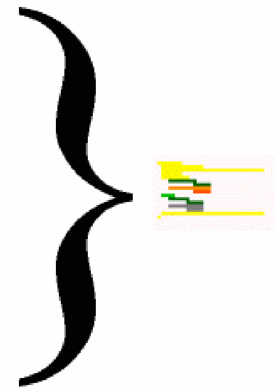
$$\mathbf{C} = t_j \left\{ \overbrace{\begin{bmatrix} 0/1 & 0/1 & \cdots & 0/1 & 0/1 \\ 0/1 & 0/1 & \cdots & 0/1 & 0/1 \\ & & \ddots & & \\ 0/1 & 0/1 & \cdots & 0/1 & 0/1 \\ 0/1 & 0/1 & \cdots & 0/1 & 0/1 \end{bmatrix}}^{p_i} \right. \quad \mathbf{e} = \begin{bmatrix} 0/1 \\ 0/1 \\ \vdots \\ 0/1 \\ 0/1 \end{bmatrix}$$

Suspiciousness assessment

- "Consistent Faults Method"
  1. Only one fails assumption
  2. Always fails assumption
  - Find the matching coverage vector with the fault vector
  - Rarely satisfied, especially the second assumption
- Similarity-based methods:
  - Tarantula: failing test vs. all hitting test
  - Ochiai: a cosine similarity

```
mid() {
  int x,y,z,m;
  read("Enter 3 numbers:",x,y,z);
  m = z;
  if (y<z)
    if (x<y)
      m = y;
    else if (x<z)
      m = y;
  else
    if (x>y)
      m = y;

  print("Middle number is:", m);
}
```



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Fault Localisation Capability

- How “variable” are test cases in terms of their coverage vectors?
- Partitioning based on coverage vectors
- Number and sizes of partitions

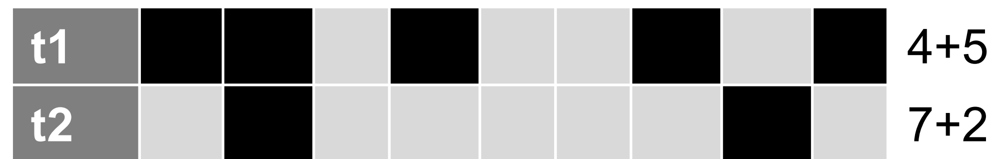
	p1	p2	p3	p4	p5
t1	1	1	1	1	1
t2	1	1	0	0	1
t3	0	0	1	1	1

π_1 (p1, p2) π_2 (p3, p4) π_3 (p5)

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Reduction for fault localisation

- A possible metric: $$FL\ metric = \sum_{i=1}^{K} |\pi_i|(|\pi_i| - 1)$$
  - Or entropy
- Greedy algorithm
  - Selects test cases that improve the fault localization capability best
  - Repeat iteratively until a fixed size or the original localisation capability metric is achieved
- I.e. tests that separate procedures best into different partitions



- t1 should be preferred because it produces smaller partitions

Results

→1 is good

FD - space

Partition
Naive coverage
Additional coverage
Partition*Naive
Partition*Additional

→0 is good

FL - space

Iterations

1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 0,1 0,2 0,3 0,4 0,5 0,6 0,7 0,8 0,9 1

Reduction sizes: 1,3,7,15, …

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering