Advanced Topics
Combining S. M. T

an empirical study of predicate dependence levels and trends

Icse 2003

david binkley
mark harman

Or …
How to use slicing to measure testability
overview

- evolutionary testing
- variable dependence
- empirical study programs
- results
- implications

DaimlerChrysler approach to test data generation

the starting point for the study was evolutionary test data generation
fitness = approximation level + local distance

if A = B  local distance = | A - B |

search space reduction

variable d is not mentioned
so we must keep y
so c is killed
these are the inputs
x, y, and z seem to be killed
search space reduction is exponential

but wait...

but suppose we target this predicate … which variables matter?
of the original 7 variables only 3 matter
an empirical study

the question
for a typical predicate
how much of input space is relevant?

why care?
how much do inputs affect?
reduce search space
implications for slicing
... slice size for the predicate is closely related

comprehension impact analysis measurement
inputs

formal parameters

globals in scope

du-globals

transitively defined or used in the predicate’s procedure

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replace</td>
<td>Regular expression string replacement</td>
</tr>
</tbody>
</table>

the programs studied
data collection

modified HRB slicing algorithm
implemented using CodeSurfer
formals and du-globals become parameters

terminology

max parameters
the maximum number of parameters a predicate could depend upon

parameters used
the number of parameters a predicate actually depends upon
result data

three results to summarise in one data point
- max parameters
- parameters used
- number of predicates summarised

we adopted two diagramatic techniques

dependence skyline diagram

notice the trend for dependent proportion to drop
8 predicates have max formals of 3 and use 2.2 on average

evident savings for prepro

replace

listed in ascending order of max formals
... but precisely ranking is lost for size reduction
dependence bubble chart

replace \hspace{0.3cm} formal parameters \hspace{0.3cm} prepro

angle shows the size reduction

these is a bubble chart showing how many formal parameters are available to a predicate and how many are actually used.

over all predicates in all programs
good news

as max formals increases
the dependent proportion falls well almost
good news for evolutionary testing
and also for all the other applications

perhaps it is not good for cohesion

declining dependent proportion
all predicates where max formals < 11

all predicates where max formals > 10

as a function of max formals available
declining dependent proportion

max formals < 11

max formals > 10

bad news

no such correlation for globals
globals could entail untestability using search
also bad for other applications
is this more evidence that globals are bad
dependence trends

formals:

as the problem gets worse
the solution gets better
implications

globals:
- large global variable lists present problems
- hard to generate test data
- hard to understand
- high levels of dependence

implications

cohesion:
- functions with large numbers of formals may not be so cohesive
algorithm performance

some possible interpretations

what do you read into these diagrams?
flex evolution of du globals

version 2.4.7

version 2.5.4

profiles

big bubbles top right  big bubbles bottom left

sendmail du globals  findutils du globals
conclusions

falling formal dependent proportion
invariant global dependent proportion

analysis is worth performing

diagrams may be an aid to
  understanding
  evaluation
  monitoring evolution

Testability Transformation
Overview
or…
How to use slicing and transformation to improve testing

Test Data Generation
  Evolutionary Test Data Generation
  The Flag Problem
Flag Removal Algorithm
Initial Results
Testability Transformation
Other ‘non meaning-preserving’ transformations

If time permits
Automatic Test Generation

We know that generating good quality test data is hard
and knowing what **good quality** means is hard
I do not propose to answer that question today

Starting point: structural test adequacy criterion
Specifically: that some branch is to be **covered**
and that we are going to use **evolutionary testing**

Distance Based Fitness

1. **Approximation level**

   Relevant branching statements can lead to a miss of the desired target

2. **Local distance calculation in the branching statements with undesired branching**

   Evaluation of predicate in a branching condition in the same manner as described for safety testing, e.g.
   if \( A = B \) \( \iff \) \( \text{Local} \_\text{Distance} = \max - (| A - B |) \)

\( \iff \) Fitness = \text{Approximation Level} + \text{Local Distance}
The Flag Problem

if (A==0) ...

Suppose we want to make this true

Max - (| A-0 |)

flag = A==0;

if (flag) ...

Transform program to transform fitness function to transform landscape

Better Flag Landscape
Informally

A **transformation** is a partial function on programs which preserves meaning, of some kind or other.

We need to pair the program and test adequacy criterion – call this the **test pair**

A **testability transformation** is a partial function on test pairs such that...

---

Testability Transformation

Test data which is **adequate** for the **transformed** test pair is **adequate** for the **original** test pair.
Testability Transformation Paradox

We are testing to cover **structure** … but the **structure is the problem**
So we **transform** the program … and this **alters the structure**

So we need to be careful:
Are we **still testing according to the same criterion?**

Our transformations will preserve coverage of

- Statements
- branches
- MC/DC

Future work: define a semantics to verify this

This is **not** abstract interpretation

To preserve branch coverage:

```
if (e) skip; else skip;
```

**Cannot** be transformed to `skip;`

But the **program**

```
if (e) x=1; else x=2;
```

**Can** be transformed to
```
if (e) skip; else skip;
```
Flag TT

Transform the program to remove flags
Not always possible
but worth doing where possible

Our Approach uses

Simple amorphous slicing
Substitute flag use with definition

Brief overview of amorphous slicing...

Flag Removal Transformation

Suppose \( n \) is an unsigned integer
What initial values of \( n \) will achieve this?
Once we have this
We can keep the original flag assignment code

Claim:
Adding the new flag assignment leaves adequacy criteria invariant
Simple Flag Removal Algorithm

For loop free flag definition code

Bush
Blossom
Slice leaf sequences
Convert to conditional assignment
Add temporary variables
Substitute definition for use
Blossoming

Moves all actions to the leaves

All internal nodes will be predicates

Original predicates may be altered

Blossoming is repeated for all internal action nodes

Some leaf assignments are not to the flag variable

… these can be removed using slicing

Now all internal nodes are predicates
Amorphous slicing gives single assignment at leaves

Sometimes syntax-preserving slicing doubles the number of single assignments

PQ: Some predicates change several times during blossoming

We assumed freedom from side effects

Fortunately we have a side effect removal transformation

Now all leaves are single assignments

... and they all assign to the flag variable

Initial Empirical Analysis

Daimler ran their Evolutionary Test Data Generator on both versions

Collected coverage information for 6 runs
Special Values

/* date correction for september 1752 */
if(special_days)
    result = "Day did not exist.";
else
    if (leapflag && is_september && day>13)
        result = dayName((addMonths(month,year)+
                        (--day)+firstJanuary(year)+10)%7);
    else ...

Nothing special flag

returnflag = (a==0 || b==0 || c==0) ||
            a>10000 || b>10000 || c>10000 ||
            (c>=a+b) || (a>=b+c) || (b>=a+c);
...
if (returnflag) return;

Is it hard to find values which make this flag false?
Is it hard to find values which make this flag true?
To have a problem we need flag for which few inputs make it true/false
Let's look at a bad flag problem ...
returnflag = (a==99999 || b==99999 || c==99999);
...
if (returnflag) return;

So the flag only fires if a or b or c is the special value

... at less cost

The flag true

So we get better coverage

---

Disposable Transformations

We generate test data using the transformed program because it is easier

... then throw away the transformed program

Transformation as a means to an end not an end in itself

Do the transformations even need to preserve meaning?

This is a radically new form of transformation
Conclusion

Test data generation is hard
  … anything which helps is good

Test data generation can be impeded by structure
  … so **transform the structure**

We have to be sure to preserve **branch** coverage
  … but **not traditional meaning**

This suggests a new kind of transformation

**Testability Transformation**

References

• David Binkley and Mark Harman.
  *Analysis and Visualization of Predicate Dependence on Formal Parameters and Global Variables.*

• Mark Harman, Lin Hu, Rob Hierons, Joachim Wegener, Harmen Sthamer, Andre Baresel and Marc Roper.
  *Testability Transformation.*

• David Binkley and Mark Harman.
  *An Empirical Study of Predicate Dependence Levels and Trends*

Electronic copies of these papers are available on my website.