Software Engineering

Future of SE / Dynamic Analysis in SE

CS 1530 Software Engineering Fall 2004

Reviewing Wasserman’s recommendations
- Abstraction
- Analysis and design notations
- User interface prototyping
- Software architecture
- Software process
- Reuse
- Measurement
- Tools and integrated environments

How to move Research into Practice: Adopter types
- Innovators
- Early adopters
- Early majority
- Late majority
- Laggards
Sold on Ideas: types of evidence

- Tangible evidence
- Testimonial evidence
- Equivocal testimonial evidence
- Missing evidence
- Accepted facts

Speed of tech transfer influenced by:

- The nature of the communication channels used to increase awareness and knowledge of the technology
- The nature of the social system in which the potential user operates
- The extent of efforts to diffuse the technology throughout an organization
- The technology’s attributes
  - relative advantage
  - compatibility
  - complexity
  - trialability
  - observability

Question 1: You have a 50% chance of losing $200 and a 50% chance of losing nothing. Would you be willing to pay $100 to avoid this situation?

Question 2: You can pay $100 to avoid a situation where you may lose $200 or nothing. Would you pay if there were a 50% chance of losing?
First framing:

Program A: Exactly 200 lives will be saved.

Program B: 1/3 chance of saving all 600, and 2/3 chance of saving none.

Alternate framing:

Program C: Exactly 400 lives will be lost.

Program D: 1/3 chance that no one will die, and 2/3 chance that 600 will die.

SE Research

- Many, many directions
- One area: tools to aid in the design, coding and maintenance of programs
  - tools that are always right vs. tools that are useful
  - "dynamic analysis"
- Work in next slides: joint research with D. Atkinson, C. Chambers & S. Eggers

Program Slicing

- Promise: look only at the relevant parts of a program
  - Backward slicing: what may have influenced <statement,variable>
  - Forward slicing: what is affected by a change to <statement>
A Backward Slice

```c
sum = 0;
prod = 1;
i = 0;
while (i<N) do {
  sum += i;
  prod *= i;
}
printf("%d
", sum);
printf(%d
, prod);
```

Another Backward Slice

```c
*sump = 0;
*prodp = 1;
i = 0;
while (i<N) do {
  *sump += i;
  *prodp *= i;
}
printf("%d
", *sump);
printf(%d
, *prodp);
```

Identifying Data Dependences

- Pointer analysis crucial
- Slice size depends on pointer analysis precision
- Reason why slices of C programs often very large?
What size are slices with “Optimal” Pointer Information?

- How much could slice size be reduced with perfect pointer information?
- Static pointer analysis results are approximate
  - Compared to run-time behavior very conservative

Static Pointer Analysis

- Computes conservative approximation of actual run-time behavior
  - E.g. as points-to analysis: *p may point to {x,y,z}
  - Conservative results make dependent analyses potentially more imprecise

Dynamic Points-to Sets

- **Observe** what pointers point to at run-time
  - Lower bound on actual “true” points-to sets, i.e., Dynamic-pts(p) ⊆ static-pts(p)
  - Lower bound on pointer-induced data dependences
  - Upper bound on slice size improvements from more precise pointer analysis
**Dynamic vs. Static Points-To Sets**

![Dynamic vs. Static Points-To Sets](image)

**Experiments**
- Sprite slicing tool for C
  - Computes a backward slice
  - Modified to use either
    - Steensgaard’s points-to analysis
    - Dynamic points-to sets
- Slices
  - All possible criteria from executed statements
  - Varied several parameters

**Benchmark Suite**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Lines</th>
<th>Criteria</th>
<th>Origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>art</td>
<td>1,270</td>
<td>837</td>
<td>SPEC2000</td>
</tr>
<tr>
<td>equake</td>
<td>1,513</td>
<td>1,111</td>
<td>SPEC2000</td>
</tr>
<tr>
<td>mcf</td>
<td>1,909</td>
<td>880</td>
<td>SPEC2000</td>
</tr>
<tr>
<td>bzip2</td>
<td>4,639</td>
<td>1,579</td>
<td>SPEC2000</td>
</tr>
<tr>
<td>gzip</td>
<td>7,757</td>
<td>1,546</td>
<td>SPEC2000</td>
</tr>
<tr>
<td>ispell</td>
<td>8,020</td>
<td>1,617</td>
<td>GNU</td>
</tr>
<tr>
<td>parser</td>
<td>10,924</td>
<td>6,223</td>
<td>SPEC2000</td>
</tr>
<tr>
<td>diff</td>
<td>11,755</td>
<td>2,110</td>
<td>GNU</td>
</tr>
<tr>
<td>ammp</td>
<td>13,263</td>
<td>5,146</td>
<td>SPEC2000</td>
</tr>
<tr>
<td>vpr</td>
<td>16,973</td>
<td>7,993</td>
<td>SPEC2000</td>
</tr>
<tr>
<td>less</td>
<td>18,305</td>
<td>1,879</td>
<td>SPEC2000</td>
</tr>
<tr>
<td>twolf</td>
<td>19,748</td>
<td>13,816</td>
<td>SPEC2000</td>
</tr>
<tr>
<td>vortex</td>
<td>52,633</td>
<td>31,324</td>
<td>SPEC2000</td>
</tr>
<tr>
<td>grep</td>
<td>13,084</td>
<td>3,520</td>
<td>SPEC2000</td>
</tr>
<tr>
<td>find</td>
<td>13,122</td>
<td>740</td>
<td>GNU</td>
</tr>
<tr>
<td>mesa</td>
<td>49,701</td>
<td>7,272</td>
<td>SPEC2000</td>
</tr>
<tr>
<td>burlap</td>
<td>49,845</td>
<td>5,293</td>
<td>FELT</td>
</tr>
<tr>
<td>gap</td>
<td>59,482</td>
<td>15,245</td>
<td>SPEC2000</td>
</tr>
</tbody>
</table>
Results: Dereference Size

Results: Flow-Set Size

Results: Slice Size
Conclusions

- The Good
  - More compact call graph \(\Rightarrow\) smaller slices

- The Bad
  - Programs with little function pointer use benefit insignificantly

- The Ugly
  - Useful slices for C programs may be hard to achieve in practice