CSE P503:
Principles of Software Engineering

David Notkin
Spring 2009

Tonight's agenda

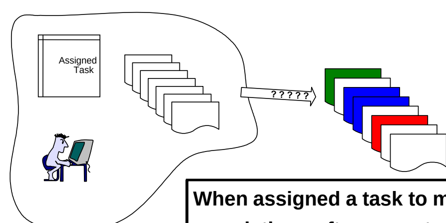
- Software reverse engineering, visualization, etc.
- "So, what happened in Vancouver BC at ICSE 2009?"
- Dynamic invariants
- "What is the remaining work for p503 this quarter?"

Reverse engineering & visualization

- Do you use any tools for these?
- If so, which, and what is your experience?
- If not, why not?

Discussion

A view of maintenance

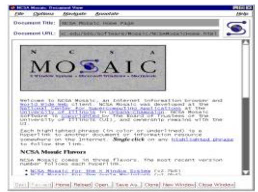


When assigned a task to modify an existing software system, how does a software engineer choose to proceed?

A task: isolating a subsystem

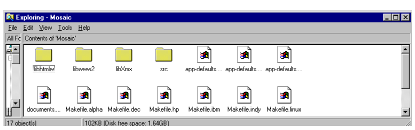
- Many maintenance tasks require identifying and isolating functionality within the source
  - sometimes to extract the subsystem
  - sometimes to replace the subsystem

Mosaic



- A task might (have been) to isolate and replace the TCP/IP subsystem that interacts with the network with a new corporate standard interface
- First step in task is to estimate the difficulty

Mosaic source code



- After some configuration and perusal, determine the source of interest is divided among 4 directories with 157 C header and source files
- Over 33,000 lines of non-commented, non-blank source lines

Some initial analysis

- The names of the directories suggest the software is broken into
  - code to interface with the X window system
  - code to interpret HTML
  - two other subsystems to deal with the world-wide-web and the application (although the meanings of these is not clear)

How to proceed?

- What source model – information extracted from the code – would be useful?
  - calls between functions (particularly calls to Unix TCP/IP library)
  - references to global variables
- How do we get this source model?
  - statically with a tool that analyzes the source or dynamically using a profiling tool
  - these differ in information characteristics
    - False positives, false negatives, etc.

More...

- What we have
  - approximate call and global variable reference information
- What we want
  - increase confidence in source model
- Action:
  - collect dynamic call information to augment source model

Augment with dynamic calls

- Compile Mosaic with profiling support
- Run with a variety of test paths and collect profile information
- Extract call graph source model from profiler output
  - 1872 calls
  - 25% overlap with CIA (an old tool)
  - 49% of calls reported by gprof not reported by CIA

Are we done?

- We are still left with a fundamental problem: how to deal with one or more “large” source models?
  - Mosaic source model:

static function references (CIA)	3966
static function-global var refs (CIA)	541
dynamic function calls (gprof)	1872
Total	6379

One approach

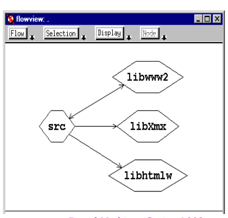
- Use a query tool against the source model(s)
  – maybe grep?
  – maybe source model specific tool?
- As necessary, consult source code
  – "It's the source, Luke."
  – Mark Weiser. Source Code. *IEEE Computer 20*,11 (November 1987)

Other approaches

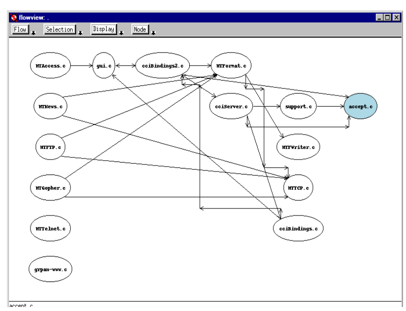
- Visualization
- Reverse engineering
- Summarization

Visualization

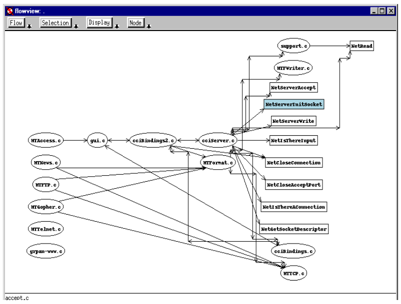
- e.g., Field, Plum, Imagix 4D, McCabe, etc. (Field's flowview is used above and on the next few slides...)
- Note: several of these are commercial products



Visualization...



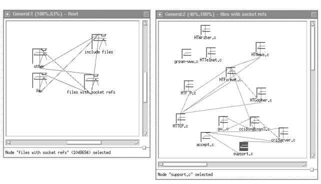
Visualization...



Visualization...

- Provides a “direct” view of the source model
- View often contains too much information
  - Use elision (…)
  - With elision you describe what you are not interested in, as opposed to what you are interested in

Reverse engineering



- e.g., Rigi, various clustering algorithms (Rigi is used above)

Reverse engineering...

Clustering

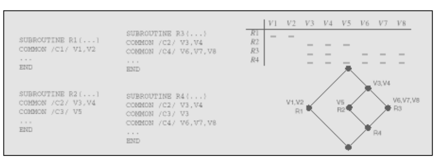
- The basic idea is to take one or more source models of the code and find appropriate clusters that might indicate “good” modules
- Coupling and cohesion, of various definitions, are at the heart of most clustering approaches
- Many different algorithms

Rigi’s approach

- Extract source models (they call them resource relations)
- Build edge-weighted resource flow graphs
  - – Discrete sets on the edges, representing the resources that flow from source to sink
- Compose these to represent subsystems
  - – Looking for strong cohesion, weak coupling
- The papers define interconnection strength and similarity measures (with tunable thresholds)

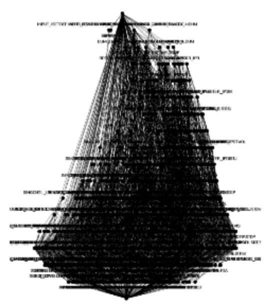
Mathematical concept analysis

- Define relationships between (for instance) functions and global variables [Snelting et al.]
- Compute a concept lattice capturing the structure
  - – “Clean” lattices = nice structure
  - – “ugly” ones = bad structure



An aerodynamics program

- 106KLOC Fortran
- 20 years old
- 317 subroutines
- 492 global variables
- 46 COMMON blocks

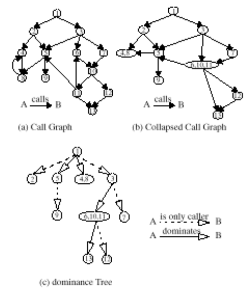


Other concept lattice uses

- File and version dependences across C programs (using the preprocessor)
- Reorganizing class libraries

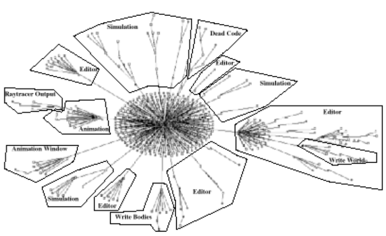
Dominator clustering

- Girard & Koschke
- Based on call graphs
- Collapses using a domi
- Heuristics for putting va



Aero program

- Rigid body simulation; 31KLOC of C code; 36 files; 57 user-defined types; 480 global variables; 488 user-defined routines



Other clustering

- Schwanke
  – Clustering with automatic tuning of thresholds
  – Data and/or control oriented
  – Evaluated on reasonable sized programs
- Basili and Hutchens
  – Data oriented

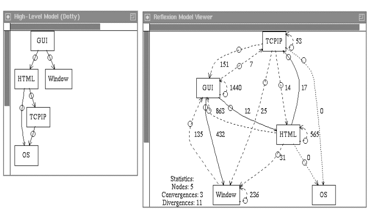
Reverse engineering recap

- Generally produces a higher-level view that is consistent with source
  - Like visualization, can produce a “precise” view
  - Although this might be a precise view of an approximate source model
- Sometimes view still contains too much information leading again to the use of techniques like elision
  - May end up with “optimistic” view

More recap

- Automatic clustering approaches must try to produce “the” design
  - One design fits all
- User-driven clustering may get a good result
  - May take significant work (which may be unavoidable)
  - Replaying this effort may be hard
- Tunable clustering approaches may be hard to tune; unclear how well automatic tuning works

Summarization



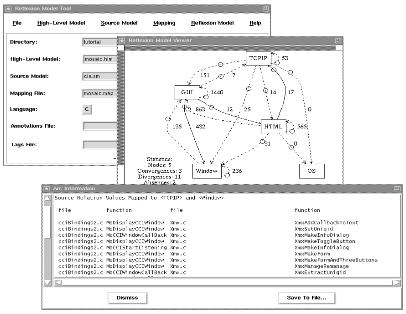
- e.g., software reflexion models

Summarization...

- A map file specifies the correspondence between parts of the source model and parts of the high-level model

```
[ file=HTTCP        mapTo=TCPIP ]
[ file=^SGML        mapTo=HTML ]
[ function=socket   mapTo=TCPIP ]
[ file=accept       mapTo=TCPIP ]
[ file=cci          mapTo=TCPIP ]
[ function=connect  mapTo=TCPIP ]
[ file=Xm           mapTo=Window ]
[ file=^HT          mapTo=HTML ]
[ function=.*       mapTo=GUI ]
```

Summarization...



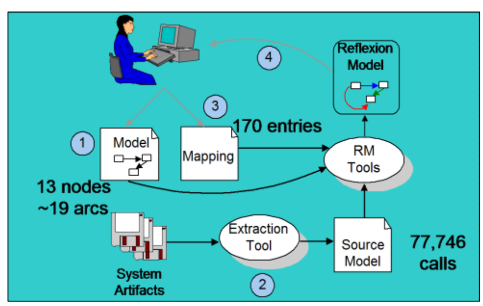
Summarization...

- Condense (some or all) information in terms of a high-level view quickly
  - In contrast to visualization and reverse engineering, produce an “approximate” view
  - Iteration can be used to move towards a “precise” view
- Some evidence that it scales effectively
- May be difficult to assess the degree of approximation

Case study: A task on Excel

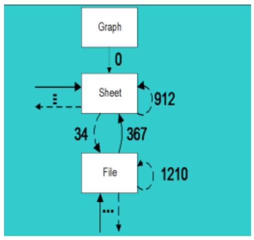
- A series of approximate tools were used by a Microsoft engineer to perform an experimental reengineering task on Excel
- The task involved the identification and extraction of components from Excel
- Excel (then) comprised about 1.2 million lines of C source
  - About 15,000 functions spread over ~400 files

The process used



An initial Reflexion Model

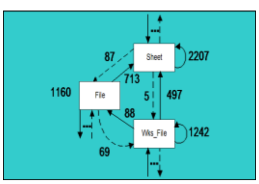
- The initial Reflexion Model computed had 15 convergences, 83, divergences, and 4 absences
- It summarized 61% of calls in source model

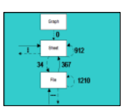


An iterative process

- Over a 4+ week period
- Investigate an arc
- Refine the map
  – Eventually over 1000 entries
- Document exceptions
- Augment the source model
  – Eventually, 119,637 interactions

A refined Reflexion Model





- A later Reflexion Model summarized 99% of 131,042 call and data interactions
- This approximate view of approximate information was used to reason about, plan and automate portions of the task

Results

- Microsoft engineer judged the use of the Reflexion Model technique successful in helping to understand the system structure and source code

  "Definitely confirmed suspicions about the structure of Excel. Further, it allowed me to pinpoint the deviations. It is very easy to ignore stuff that is not interesting and thereby focus on the part of Excel that I want to know more about." — Microsoft A.B.C. (anonymous by choice) engineer

Open questions

- How stable is the mapping as the source code changes?
- Should reflexion models allow comparisons separated by the type of the source model entries?
- ...

ICSE?

- What is it?
- When is it?
- What happens?
- How does it work?

ICSE 2009: semi-random tidbits

- Michael Jackson tribute
  - Tony Hoare, Daniel Jackson and others
  - Michael Jackson on contrivances
- ICSE N-10 most influential paper: "N Degrees of Separation: Multi-Dimensional Separation of Concerns" by P Tarr, H Ossher, W Harrison, SM Sutton Jr.
- Steve McConnell keynote: 10 Most Important Ideas in Software Development
- Two example research results
  - The Secret Life of Bugs: Going Past the Errors and Omissions in Software Repositories (Jorge Aranda, Gina Venolia)
  - Invariant-Based Automatic Testing of AJAX User Interfaces (Ali Mesbah, Arie van Deursen)

Program invariants

- Invariants can aid in the development of correct programs
  - The invariants are defined explicitly as part of the construction of the program
- Invariants can aid in the evolution of software as well
- In particular, programmers can easily make changes that violate unstated invariants
  - The violated invariants are often far from the site of the change
  - These changes can cause errors
  - The presence of invariants can reduce the number of or cost of finding these violations

But…

- …most programs have few invariants explicitly written by programmers
- Ernst's idea: trace multiple executions of a program and apply machine learning to discover likely invariants (such as those found in assert statements or specifications)
  - *x > abs(y)*
  - *x = 16*y + 4*z + 3*
  - *array a contains no duplicates*
  - *for each node n, n = n.child.parent*
  - *graph g is acyclic*
  - *…*

Example: Recover formal specification

```
// Sum array b of length n into
// variable s
i := 0; s := 0;
while i ≠ n do
  { s := s + b[i];  i := i + 1 }
```

- Precondition: $n \geq 0$
- Postcondition: $S = \Sigma_{0 \leq j < n}\, b[j]$
- Loop invariant:
  $0 \leq i \leq n$ and $S = \Sigma_{0 \leq j < i}\, b[j]$

Test suite: first guess

- 100 randomly-generated arrays
  - length uniformly distributed from 7 to 13
  - elements uniformly distributed from –100 to 100

Inferred invariants

```
ENTRY:
   N = size(B)
   N in [7..13] ♦
   B: All elements in [-100..100]
EXIT:
   N = I = orig(N) = size(B)
   B = orig(B)
   S = sum(B) ♦
   N in [7..13]
   B: All elements in [-100..100]
```

Inferred loop invariants

```
LOOP:
   N = size(B)
   S = sum(B[0..I-1]) ♦
   N in [7..13]
   I in [0..13] ♦
   I <= N ♦
   B: All elements in [-100..100]
   B[0..I-1]: All elements in [-100..100]
```


Example: Code without explicit invariants

- 563-line C program: regular expression search & replace [Hutchins][Rothermel]
- Task: modify to add Kleene +
- Complementary use of both detected invariants and traditional tools (such as grep)

Programmer use of invariants

- Helped explain use of data structures
  - regexp compiled form (a string)
- Contradicted some maintainer expectations
  - anticipated *lj* < *j* in `makepat`
  - queried for counterexample
  - avoided introducing a bug
- Revealed a bug
  - when *lastj* = **j* in `stclose`, array bounds error

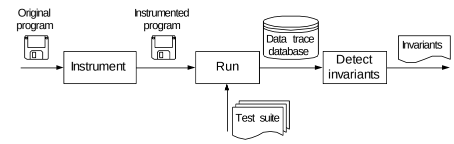
More invariant uses

- Showed procedures used in limited ways
  - `makepat`
    start = 0 and *delim = '\0'*
- Demonstrated test suite inadequacy
  - *#calls(in_set_2) = #calls(stclose)*
- Changes in invariants validated program changes
  - `stclose`: **j = orig(*j)+1*
  - `plclose`: **j ≥ orig(*j)+2*

Experiment 2 conclusions

- Invariants
  - effectively summarize value data
  - support programmer's own inferences
  - lead programmers to think in terms of invariants
  - provide serendipitous information
- Additional useful components of Daikon
  - trace database (supports queries)
  - invariant differencer

Dynamic invariant detection



- Look for patterns in values the program computes
  - Instrument the program to write data trace files
  - Run the program on a test suite
  - Invariant engine reads data traces, generates potential invariants, and checks them
- Roughly, machine learning over program traces

Requires a test suite

- Standard test suites are adequate
- Relatively insensitive to test suite (if large enough)
- No guarantee of completeness or soundness
- Complementary to other techniques and tools

Sample invariants

- *x,y,z* are variables; *a,b,c* are constants
- Invariants over numbers
  - unary: $x = a$, $a \leq x \leq b$, $x \equiv a (\mathrm{mod}\ b)$, …
  - n-ary: $x \leq y$, $x = ay + bz + c$, $x = \max(y, z)$, …
- Invariants over sequences
  - unary: sorted, invariants over all elements
  - with sequence: subsequence, ordering
  - with scalar: membership

Checking invariants

- For each potential invariant:
  - Instantiate
    - That is, determine constants like *a* and *b* in $y = ax + b$
  - Check for each set of variable values
  - Stop checking when falsified
- This is inexpensive
  - Many invariants, but each cheap to check
  - Falsification usually happens very early

Relevance

- Our first concern was whether we could find any invariants of interest
- When we found we could, we found a different problem
  - We found many invariants of interest
  - But most invariants we found were not relevant

Find relationships over non-variables

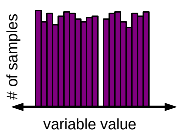
- array: *length, sum, min, max*
- array and scalar: element at index, subarray
- number of calls to a procedure
- …

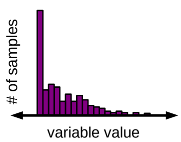
Unjustified properties

- Given three samples for **`x`**:
  - **`x = 7`**
  - **`x = -42`**
  - **`x = 22`**
- Potential invariants:
  - $\mathbf{x \neq 0}$
  - $\mathbf{x \leq 22}$
  - $\mathbf{x \geq -42}$

Statistically check hypothesized distribution

- Probability of no zeroes (to show $x \neq 0$) for v values of x in range of size r $\left(1-\frac{1}{r}\right)^{v}$
- Range limits (e.g., $x \leq 22$)
  - same number of samples as neighbors (uniform)
  - more samples than neighbors (clipped)





Duplicate values

- Array sum program:

```
i := 0; s := 0;
while i ≠ n do
  { s := s + b[i];  i := i + 1 }
```

- *b* is unchanged inside loop
- Problem: at loop head
  - ***–88 ≤ b[n – 1] ≤ 99***
  - ***–556 ≤ sum(b) ≤ 539***
- Reason: more samples inside loop

Disregard duplicate values

- Idea: count a value only if its variable was just modified
- Result: eliminates undesired invariants

Redundant invariants

- Given
  0 ≤ i ≤ j
- Redundant
  a[i] ∈ a[0..j]
  max(a[0..i]) ≤ max(a[0..j])
- Redundant invariants are logically implied
- Implementation contains many such tests

Suppress redundancies

- Avoid deriving variables: suppress 25-50%
  - equal to another variable
  - nonsensical
- Avoid checking invariants:
  - false invariants: trivial improvement
  - true invariants: suppress 90%
- Avoid reporting trivial invariants: suppress 25%

Unrelated variables

```
bool b;
int *p;
```

b < p

```
int myweight, mybirthyear;
```

myweight < mybirthyear

Limit comparisons

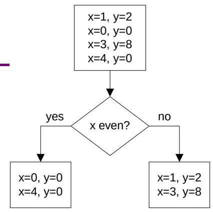
- Check relations only over comparable variables
  - declared program types: 60% as many comparisons
  - Lackwit [O'Callahan]: 5% as many comparisons; scales well
- Runtime: 40-70% improvement
- Few differences in reported invariants

Richer types of invariant

- Object/class invariants
  - `node.left.value < node.right.value`
  - `string.data[string.length] = '\0'`
- Pointers (recursive data structures)
  - `tree is sorted`
- Conditionals
  - `if  proc.priority < 0 then`
`    proc.status = active`
  - `ptr = null  or  *ptr > i`

Conditionals mechanism

- Split the data into parts
- Compute invariants over each subset of data
- Compare results, produce implications




```
if even(x) then
  y = 0
else
  y = 2x
```


Data splitting criteria

- Static analysis
- Distinguished values: zero, source literals, mode, outliers, extrema
- Exceptions to detected invariants
- User-selected
- Exhaustive over random sample

Summary

- Dynamic invariant detection is feasible
- Dynamic invariant detection is accurate & useful
  – Techniques to improve basic approach
  – Experiments provide preliminary support
- Daikon can detect properties in C, C++, Eiffel, IOA, Java, and Perl programs; in spreadsheet files; and in other data sources.
- Easy to extend Daikon to other applications
- http://groups.csail.mit.edu/pag/daikon/ (but http://www.cs.washington.edu/homes/mernst/)

So, what work is left for p503?

- Staff: grading of Alloy and research papers
- You
  – I didn't provide assignment #4, which is "Due 6:00PM on Monday June 8, 2009"
  – Here it is (soon on web page): a *choice* of
    - Proposed curriculum per last week's email
    - A shorter (5 page) additional research paper on a different topic (no approval is needed, but be reasonable)
    - An Alloy model for something you work on and want to understand better (no need to break NDA)

See you next week…