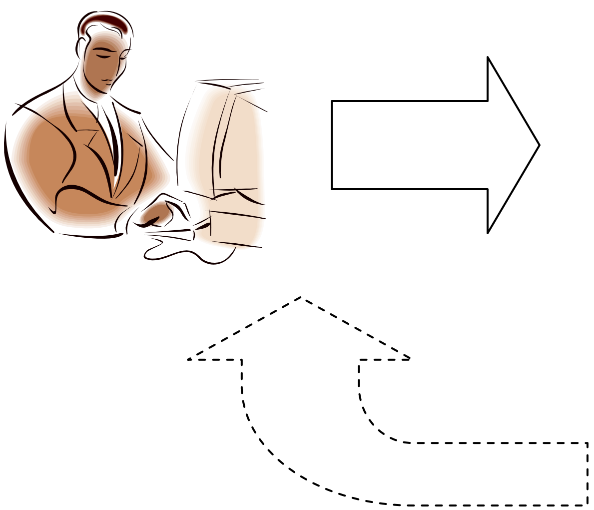
Profile-Guided Program Simplification for Effective Testing and Analysis

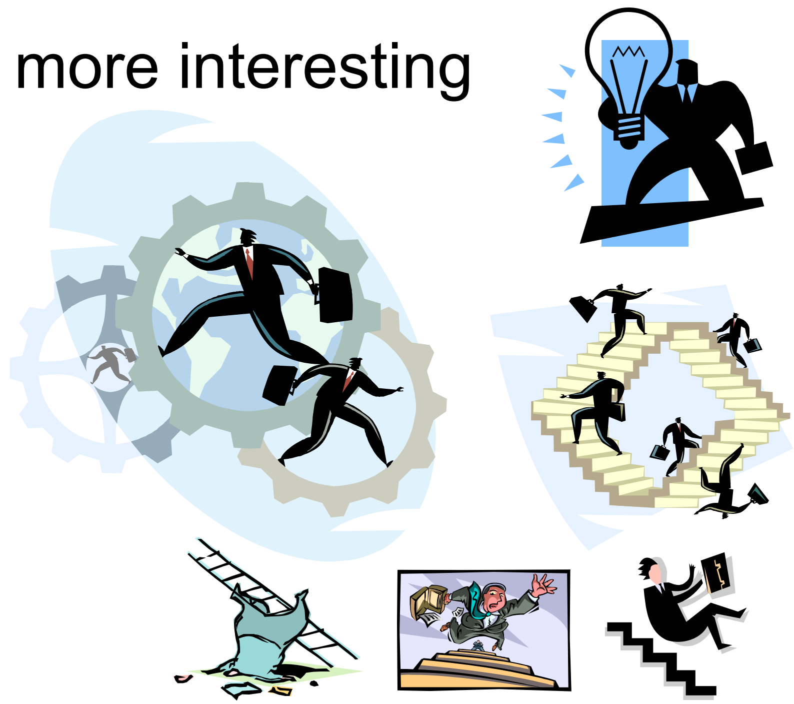
Lingxiao Jiang Zhendong Su

Program Execution Profiles

- A profile is a set of information about an execution, either succeeded or failed
- Profiles from users are more interesting



Random testing with unlimited resources



privacy, security, overhead

Profile-Collecting Infrastructure

- To reduce users' concern
- The Cooperative Bug Isolation Project
- Low-overhead
- Sparse information
- Statistical analysis on profiles from **many users** for bug Isolation
- Low costs on profile triage at the developers' side

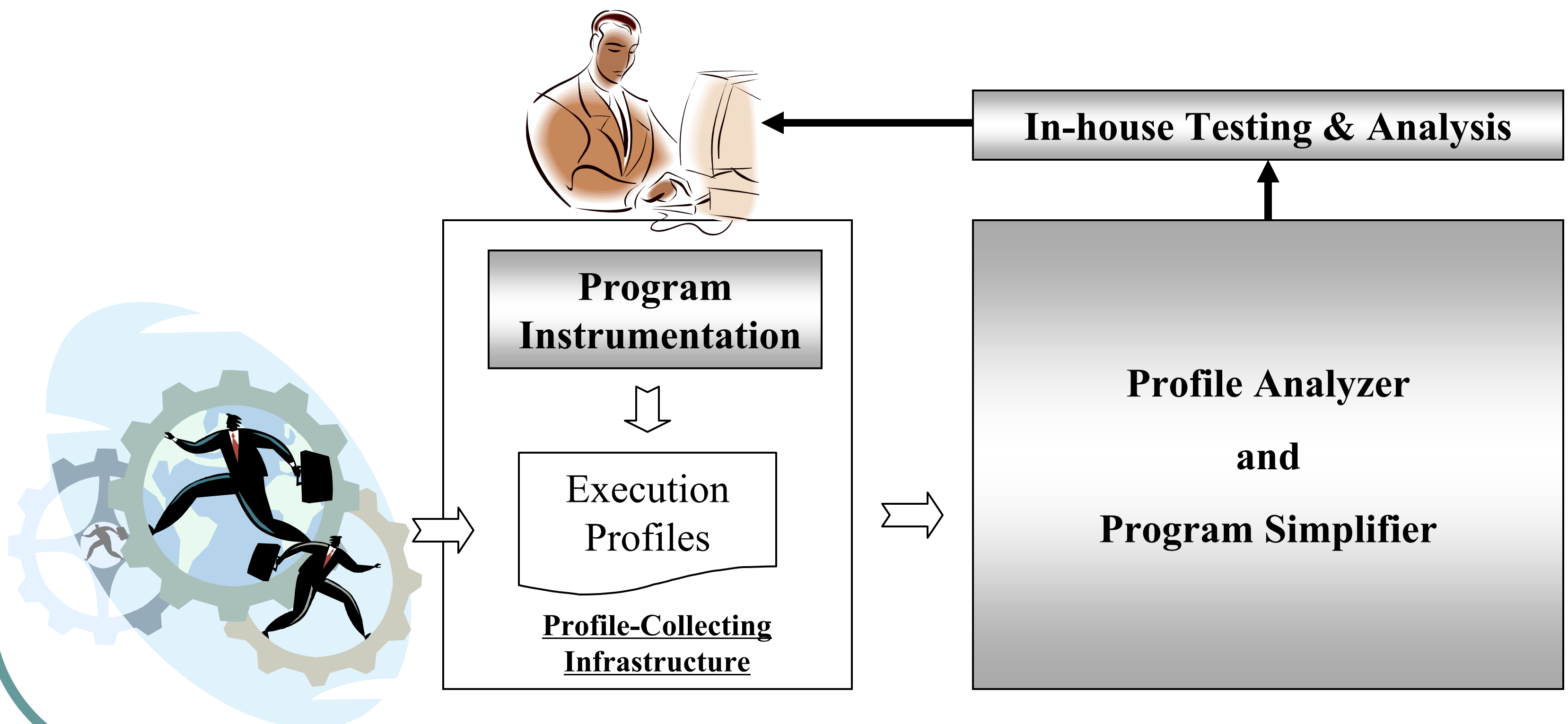
Lack of many details that **developers** desire

Bridge the Information Gap

- An insight: Many in-house development tools, as already used by developers for testing and debugging, can provide the desired information, if the program is "simple" enough

Bridge the Information Gap

- Our realization: profile-guided program simplification



Bridge the Information Gap

- A question: why not use in-house techniques on the original programs directly

In-house Testing & Analysis

Testing & Analysis - Incapabilities

- Dynamic techniques, e.g., random testing
  - Incomplete coverage → false negatives
- Static techniques, e.g., BLAST
  - Undecidable problems → false positives
- Also, implementation limitations
- **Developers** are used as an expensive remedy:
  - Manual filtering
  - Manual annotations/specifications

Example of Incapability

- Failure occurs when (**`I*J`**) is within the range of [991, 1000]

```
I = 0;
J = 1;
......
I = library_func(J);
J = library_func(I);
if ( I * J > 990 &&
     I * J <= 1000 )
   assert( FALSE );
```

Library functions are not easy to track/control

Non-linear multi-variable constraints are difficult to solve

Example Revisited: Benefit of Program Simplification

- Failure occurs when (**`I*J`**) is within the range of [991, 1000]

```
I = 0;
J = 1;
......
I = library_func(J);
J = library_func(I);
if ( I * J > 990 &&
     I * J <= 1000 )
   assert( FALSE );
```

simplification

```
I = 0;
J = 1;
......
I = library_func(J);
J = library_func(I);

if ( TRUE )
   assert( FALSE );
```

This is possible based on statistical analysis on large amount of execution profiles

Approach Overview

In-house Testing and Analysis

Program Instrumentation

Execution Profiles

Profile-Collecting Infrastructure

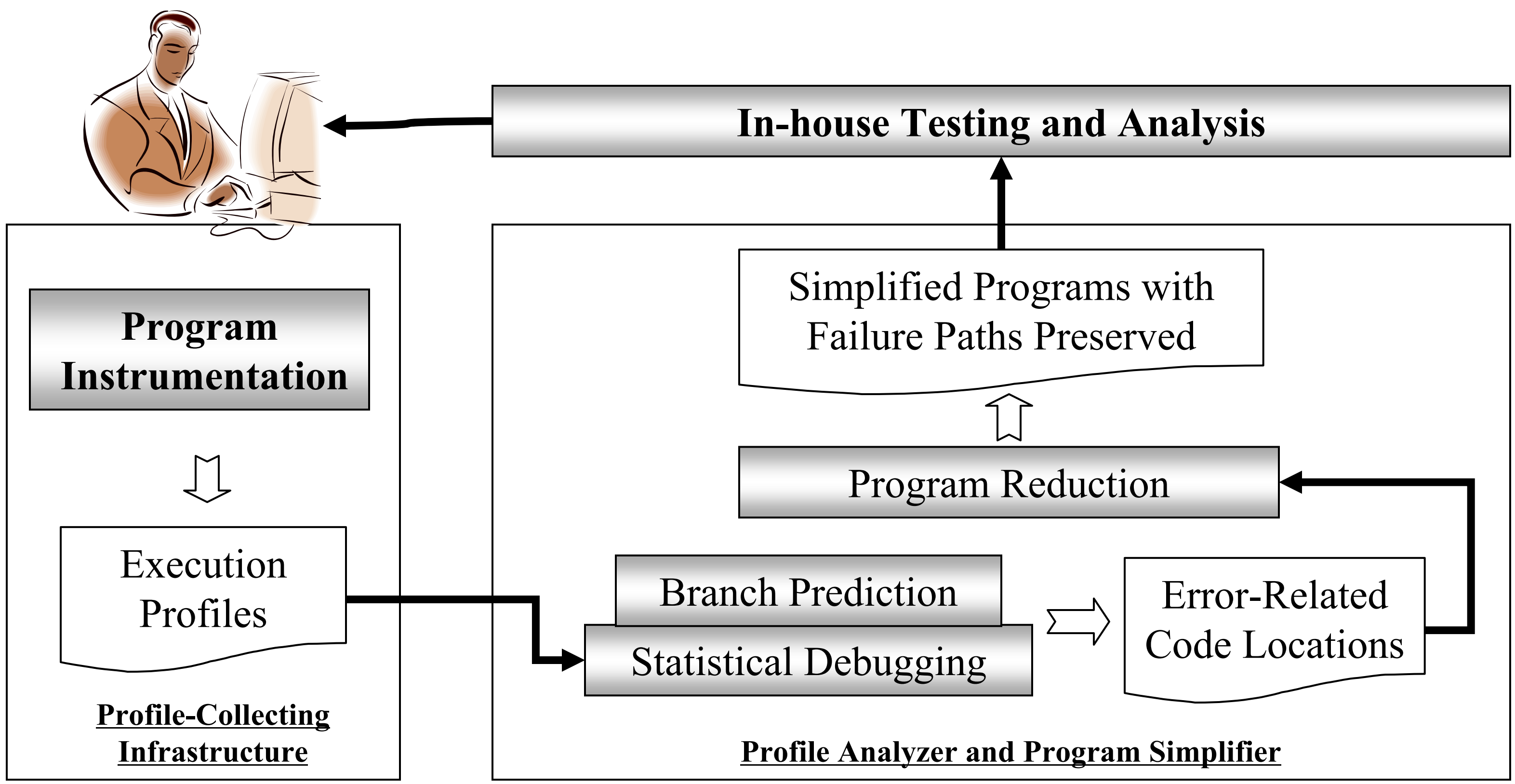
Profile Analyzer

and

Program Simplifier

Fully automatable and focus naturally on failures experienced more often by users

Approach Overview



Profile-Collecting Infrastructure

- For each program execution, CBI collects:
  - Outcome: exit status of the execution
  - List of predicate counters: e.g., the numbers of times branching conditions are observed to be true

$(p_{11}, \ldots, p_{n1}, \text{label}_1)$,

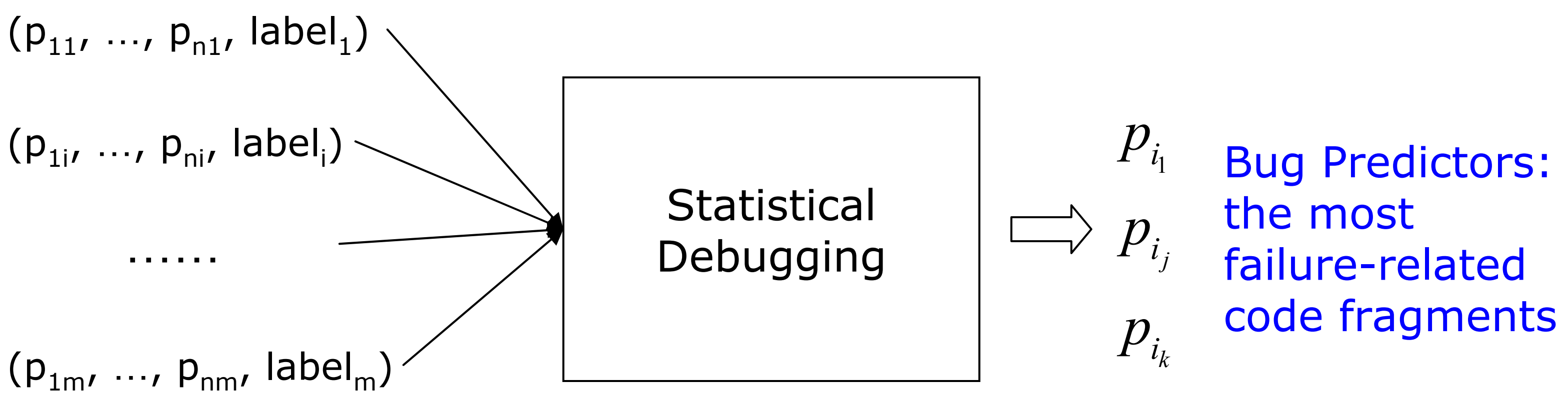
$(p_{1i}, \ldots, p_{ni}, \text{label}_i)$,

……

$(p_{1m}, \ldots, p_{nm}, \text{label}_m)$

Statistical Debugging

- A black-box view



Branch Prediction

- For each conditional C, suppose its *then branch* be C_t and its *else branch* be C_f.
- Assign a truth value for every C_i, e.g.,
  - `True` for bug predictors
  - `False` for C_i never taken in failed executions
  - `Unknown` for C_i taken in failed *and* succeeded executions
- Decide branching directions

Direction for C		Truth Value of C_t		
		True	False	Unknown
Truth Value of C_f	True	Both	False	Both
	False	True	Neither	True
	Unknown	Both	False	Both

Branch Reduction

- For each conditional $IF(B_p, B_t, B_f)$ and its prediction P, if
  - P==`Neither`, replace it with `exit(0)`
  - P==`True`, replace it with $IF($`TRUE`$, B_t, B_f)$
  - P==`False`, replace it with $IF($`FALSE`$, B_t, B_f)$
  - P==`Both`, keep it

Almost Ready

- For dynamic testing & analysis tools, the simplified code may exhibit additional failures

```
if ( p != NULL )
{
   p->next = ...
   ......
   state = ...
   assert ( state != 0 );
}
```

```
if ( TRUE )
{
   p->next
   ......
   state = ...
   assert ( state != 0 );
}
```

Segfault

Branch Validation – `Assumes`

```
if ( p != NULL )
{

   p->next = ...
   ......
   state = ...
   assert ( state != 0 );
}
```

```
if ( TRUE )
{
   assume ( p != NULL );
   p->next = ...
   ......
   state = ...
   assert ( state != 0 );
}
```

- Require supports of testing or analysis tools
- May still be faster than on original programs, but better not to add it arbitrarily

Ready for Empirical Evaluation

The current branch validation strategy: ***none*** or ***all***

- Simplify code with *no* `assume` first
- If segfaults, add in *all* `assume`s and apply back-end tools again

Experimental Setup

- Subject programs
  - **Siemens Test Suite** from the Aristotle Analysis System
  - **Gzip** from the Software-artifact Infrastructure Repository (SIR)
- CBI profiling and statistical bug localization [ASE 2007]
- Testing & analysis tools
  - **CUTE**, a testing engine that combines both concrete and symbolic execution, [PLDI 2005, FSE 2005]
  - **BLAST**, a software model checker based on predicate abstraction, [POPL 2002, SPIN 2003]

Results on CUTE

Program	# of Faults Found in		# of Conditionals		Reduction	# of Iterations Taken for	
	Original	Simplified	Total	Reduced	Percentage	Original	Simplified
PRINT_TOKENS	1	1	231	46	19.9%	4418	4414
PRINT_TOKENS2	9	9	809	41	5.1%	156	149
REPLACE	27	28	2683	881	32.8%	1892290	731954
SCHEDULE	8	8	262	104	39.7%	137675	126200
SCHEDULE2	6	6	371	123	33.2%	786918	301581
TCAS	27	41	1348	926	68.7%	674	73
TOT_INFO	4	6	1011	296	29.3%	88195	283764
GZIP	3	6	5237	3635	69.4%	14	865
Total	85	105	11952	6052	50.6%	2910340	1449000
Relative Improvements:		**+20**					**-50.2%**

Extra Bugs Detected: Examples

- Complement Implementation Limitations

```
if ( (float)input >= 1.0 )
{
    ......
    assert( FALSE );
}
```

```
if ( TRUE )
{
    ......
    assert( FALSE );
}
```

Extra Bugs Detected: Examples

- Simplify Path Constraints

```
int longNumArray[] = { 13, 25, 48,
    ..., /* a lot of numbers */...,
    255, ... };

if ( longNumArray[ input ]==255 )
{
    ......
    assert( FALSE );
}
```

```
int longNumArray[] = { 13, 25, 48,
    ..., /* a lot of numbers */...,
    255, ... };

if ( TRUE )
{
    ......
    assert( FALSE );
}
```

```
*(longNumArray + input) == 255
```

Results on CUTE

Program	# of Faults Found in		# of Conditionals		Reduction	# of Iterations Taken for	
	Original	Simplified	Total	Reduced	Percentage	Original	Simplified
PRINT_TOKENS	1	1	231	46	19.9%	4418	4414
PRINT_TOKENS2	9	9	809	41	5.1%	156	149
REPLACE	27	28	2683	881	32.8%	1892290	731954
SCHEDULE	8	8	262	104	39.7%	137675	126200
SCHEDULE2	6	6	371	123	33.2%	786918	301581
TCAS	27	41	1348	926	68.7%	674	73
TOT_INFO	4	6	1011	296	29.3%	88195	283764
GZIP	3	6	5237	3635	69.4%	14	865
Total	85	105	11952	6052	50.6%	2910340	1449000
Relative Improvements:		**+20**					**-50.2%**

May Be Slower Without `Assumes`

- Assertions may not be violated with *invalid* inputs

Suppose (`sum==0`) iff `input` is in the range of `[99900, 99998]`

```
if ( input > 99899 &&
     input < 99999 ) {
   ......
   for (i=0; i<input; i++)
      sum += foo(i);
   assert ( sum != 0 );
}
```

```
if ( TRUE ) {
  //assume (input > 99899 && input < 99999);
   ......
   for (i=0; i<input; i++)
      sum += foo(i);
   assert ( sum != 0 );
}
```

When CUTE is applied: three iterations vs. tens of thousands of iterations

Results on BLAST

Program Name	# of Faults Found in Original	# of Faults Found in Simplified	Reduction Rate	# of Calls to Theorem Prover & Reachability Analysis for Original		# of Calls to Theorem Prover & Reachability Analysis for Simplified		Relative Improvements	
PRINT_TOKENS	6/1	7	19.9%	0	5695	0	4147	0.0%	27.2%
PRINT_TOKENS2	10	10	5.1%	14424226	1481178	14309606	1474237	0.8%	0.5%
REPLACE	2/30	18/13	32.8%	0	662	0	265	0.0%	60.0%
SCHEDULE	4/1	7/1	39.7%	136238	11896	0	2428	100%	79.6%
SCHEDULE2	6/1	6/1	33.2%	39638	13799	29100	7447	26.6%	46.0%
TCAS	41	41	68.7%	0	111514	0	13734	0.0%	87.7%
TOT_INFO	19/2	20/1	29.3%	0	39429	0	28693	0.0%	27.2%
GZIP	7	7	69.4%	0	11164	0	4452	0.0%	60.1%
Total	95/35	116/16	50.6%	14600102	1675337	14338706	1535403	1.8%	8.4%
Relative Improvements:		**+21/-19**							

Discussion

- Not a direct testing or debugging tool itself
- Need different interpretation of bug reports

```
I = 0;
J = 1;
......
I = library_func(J);
J = library_func(I);

if ( TRUE )
   assert( FALSE );
```

Error Trace + Reduced Path Constraints

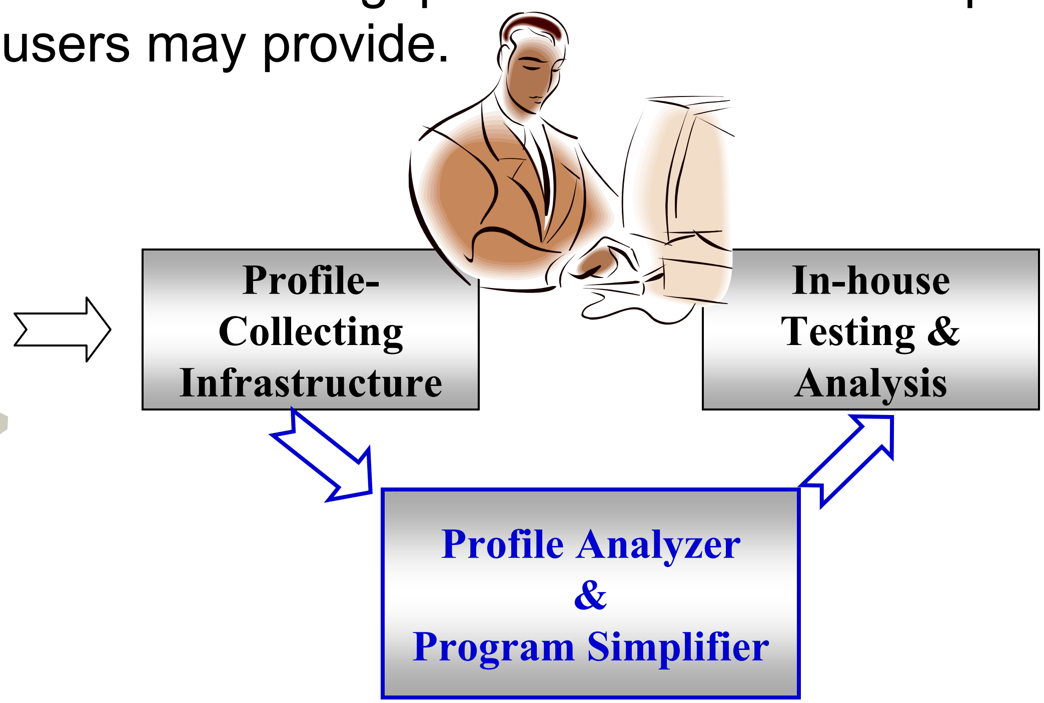
```
I * J > 990 &&
I * J <= 1000
```

Discussion

- Not a direct testing or debugging tool itself
- Need different interpretation of bug reports
- May not be faster without selective `assume`s
- Reply on the accuracy of statistical debugging
- Evaluation for larger-scale programs

Take-away Message

- Profile-Guided Program Simplification:
  a way to combine many benefits of two existing research areas and help bridge the information gap between what developers desire and what users may provide.



Thank you!

Questions?

jiangl@cs.ucdavis.edu