User Interaction Issues in Defect Detection Tools

Jon Pincus
PPRC, Reliability Tools
Microsoft Research

My first program analysis tool

- Implemented in Prolog
- Input: syntax tree for the program
  - (manually translated into Prolog representation)
- Output: complexity metrics
- Sample session:
  > compute_metrics(Program, Metrics)
  no.

Some more recent background:
PREfix

- C/C++ defect detection via static analysis
- Powerful inter-procedural analysis
- Incomplete
- Unsound
- Useful in practice
- Typically run as part of a centralized build
- V1.0: command-line tool
  - ... no significant adoption
- V2.0 ... V4.0: whole system (UI, DB)
  - ... broad adoption

Even more recent background:
PREfast

- Lightweight, “desktop” defect detection
- Simple intra-procedural analyses
- Compared to PREfix:
  - fewer defects
  - higher noise
  - fast
- V1.0: UI, XML log; fast adoption
- Infrastructure: plugins
  - New defects (e.g., I18N)
  - Platform for other tools

PREfast as an experiment

- What had we learned from PREfix?
  - Which defects people care about
  - Build integration techniques
  - User interaction techniques
  - Scalable, powerful analyses
- Replace the analyses with weak, fast ones
  - Leverage build integration, user interaction, knowledge
- Result: surprisingly well accepted
  (yeah, yeah, I know, it's not a controlled experiment ...)

“Analysis is necessary, but by no means sufficient”

- Actual analysis is only a small part of any “program analysis tool”.
  - In PREfix, < 10% of the “code mass”
- No matter what the power of the analyses, make sure to consider other aspects as well
Stepping back a little …

- Why do people use a tool? If
  - it helps them get their work done …
  - … more efficiently than they would otherwise
  - … without making them look (or feel) bad.

*User interaction is key to all of these points*

Aside: See Alan Cooper's books, e.g. *About Face*

---

Successful tools

- Provide value …
- … at sufficiently low cost
- … and work in the target environment
  - … which typically means with “average” developers

*User interaction is once again key*

---

From an internal presentation: Progress in “static analysis”

Integration
Use Model
Comprehension
Noise
Scalability
Usability
Barriers

Defects
Capability

---

Apologies in advance

- My examples are by no means exhaustive
- I’m a tools junkie, not a research junkie
  - Please don’t be offended if I fail to cite you!
- Examples are typically drawn from tools I’ve worked on or used
  - It’s just familiarity, not any belief that these are better!

---

Key areas of user interaction for defect detection tools

- Specifying what properties to check
- Controlling the analysis
- Dealing with the results of the analysis
  - Viewing individual defects
  - Dealing with noise
  - Managing sets of defects

---

This presentation’s focus

- Static analysis tools
- Dealing with the results of the analysis
  - Viewing individual defects
  - Dealing with noise
  - (Managing sets of defects?)
PREfast, SLAM, ESP: Viewing Results

Viewing Results

Source Code

Defect Log

prefast View/Bebop

prefast list

PREfix: Viewing Results

Source Code Management Server

SQL Server

Defects

IIS pf/cgi

Function Info

Sample compiler-reported defect

```c
#include <set>
std::set<const int> set;

memory(57) : error C2535: 'const int *__thiscall xmemory(57) : error C2535: 'const int *__thiscall
std::set<const int> set;std::set<const int> set;
xmemory(57) : error C2535: 'const int *__thiscall xmemory(57) : error C2535: 'const int *__thiscall
std::allocator<int const >::address(const int &) const': member std::allocator<int const >::address(const int &) const': member
function already defined or declared
memory(54): see declaration of 'address'
memory(54): see declaration of 'address'
cmp(33): see reference to class template instantiation
ncmp(33): see reference to class template instantiation
std::less<int const >,class std::allocator<int const > >::_Kfn,struct
std::less<int const >,class std::allocator<int const > >'_being compiled
std::less<int const >,class std::allocator<int const > >'_being compiled
const int.cpp(2) : see reference to class template instantiation
const int.cpp(2) : see reference to class template instantiation
std::set<int const ,struct std::less<int const >,class std::allocator<int const > >' being compiled
std::set<int const ,struct std::less<int const >,class std::allocator<int const > >' being compiled

Sample compiler-reported defect

```
Sample compiler-reported defect

```c
void f(char *p)
{
    float f;
    f = p;
}
```

Example.c(4) : error C2115: '=' : incompatible types

```c
#include <set>
std::set<const int> set;
```

example.cpp(57) : error C2535: 'const int *__thiscall xmemory(57) : error C2535: 'const int *__thiscall
```

f function already defined or declared
df function already defined or declared
```
exmemory(54): see declaration of 'address': see declaration of 'address'
xtree(51): see reference to class template instantiation : see reference to class template instantiation 'std::allocator<int const >::address(const int &) const': member std::allocator<int const >::address(const int &) const': member
```

SymbolInfo *Find(ISymbolPtr id);
SymbolInfo *Find(char *name);
main() {
const char* n = "a constant";
Find(n);
}
```

comip.h(690) : error C2227: left of 'comip.h(690) : error C2227: left of '->QueryInterface' >QueryInterface'
must point to class/struct/union
```
Viewing individual defects

- A more relevant question – and answer
- Why don't static analysis tools (including the compiler) report complex defects as clearly and concisely as easy ones?
- Because they're complex, of course!
```
Sample defect #1

```c
int f(int i)
{
    int n;
    if (i == 0)
        n = 1;
    return n;
}
```

uwmsrsi1.c(6) : warning C4701: local variable 'n' may be used without having been initialized

Sample defect #1: compiler warning

```c
int f(int i)
{
    int n;
    if (i == 0)
        n = 1;
    return n;
}
```

uwmsrsi1.c(6) : warning C4701: local variable 'n' may be used without having been initialized
Sample defect #1: PREfix warning

```c
int f(int i)
{
    int n;
    if (i == 0)
        n = 1;
    return n;
}
```

<table>
<thead>
<tr>
<th>uwmsrs1.c(6): warning 1: using uninitialized memory 'n'</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem occurs when the following condition is true:</td>
</tr>
<tr>
<td>uwmsrs1.c(4): when 'i != 0' here</td>
</tr>
<tr>
<td>Path includes 2 statements on the following lines:</td>
</tr>
<tr>
<td>4 6</td>
</tr>
</tbody>
</table>

August 16, 2001 Jon Pincus, Microsoft

Sample defect #1: Comments

- Compiler warning doesn't give enough info
- When is n uninitialized?
- As a result, often not fixed
- Key techniques
  - Additional detailed information
  - Path
  - Link to additional information
- There's an information management problem
  - Even in simple example, output is very verbose

Sample defect #2

```c
extern int phase_of_moon(void);

/* initialize the buffer; or return failure */
static int initialize(char *buff)
{
    if (phase_of_moon())
        return 0;
    buff[0] = 0;
    return 1;
}
```

<table>
<thead>
<tr>
<th>uwmsrs2.c(17): warning 1: using uninitialized memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>problem occurs in function 'uwmsrs2'</td>
</tr>
<tr>
<td>uwmsrs2.c(15): stack variable declared here</td>
</tr>
<tr>
<td>Path includes 2 statements on the following lines:</td>
</tr>
<tr>
<td>16 17</td>
</tr>
</tbody>
</table>

August 16, 2001 Jon Pincus, Microsoft

Sample defect #2 (cont.)

```c
void uwmsrsr2(void)
{
    char buff[100];
    initialize(buff);
    if (buff[0] != 0)
        return;
    /* ... */
}
```

August 16, 2001 Jon Pincus, Microsoft

Sample defect #2: PREfix message

```c
void uwmsrsr2(void)
{
    char buff[100];
    initialize(buff);
    if (buff[0] != 0)
        return;
    /* ... */
}
```

August 16, 2001 Jon Pincus, Microsoft
Sample defect #2: PREfix GUI

Comments
- Where’s the defect?
  - The failure to initialize `buff` before calling `initialize`?
  - `initialize`'s failure to initialize `buff` in the error case?
  - The fact that `initialize` can fail?
  - The failure to check the return value of `initialize`?
  - What if the caller of `uwmsrsi2` has previously verified that `phase_of_moon() == 0`?
- Key technique:
  - Navigation across functions
  - (But how does the user know where to look?)
- Is there a way to present this in a single screen?

Sample defect #3

original PREfix message

```c
CHAR buff[MAX_PATH];
GetWindowsDirectory(buff,sizeof(buff));
SetCurrentDirectory(buff,sizeof(buff));
```

Warning: using uninitialized memory `buff[0]`
Problem occurs in call to `SetCurrentDirectory`

Sample defect #3: explanation

```c
CHAR buff[MAX_PATH];
GetWindowsDirectory(buff,sizeof(buff));
SetCurrentDirectory(buff,sizeof(buff));
```

GetWindowsDirectory can fail, in which case it doesn’t initialize its output

But most people don’t think of this, so think the message is noise
Sample defect #3: revised PREfix message

... CHAR buff[MAX_PATH];
GetWindowsDirectory(buff, sizeof(buff));
Warning: Failure to check return value
SetCurrentDirectory(buff, sizeof(buff));

---

Sample defect #3: explanation

... CHAR buff[MAX_PATH];
GetWindowsDirectory(buff, sizeof(buff));
SetCurrentDirectory(buff, sizeof(buff));

GetWindowsDirectory can fail,
So the return value must be checked
But most people don’t BELIEVE this, so still think the message is noise

---

Sample defect #3: re-revised PREfix message

... CHAR buff[MAX_PATH];
GetWindowsDirectory(buff, sizeof(buff));
Warning: Failure to check return value
GetWindowsDirectory can fail in low-memory situations
SetCurrentDirectory(buff, sizeof(buff));

---

Sample defect #3: What I’d really like to see

PREfix’ display mechanisms don’t yet support this,
but I’d rather see something like ...
CHAR buff[MAX_PATH];
GetWindowsDirectory(buff, sizeof(buff));
Warning: Failure to check return value
GetWindowsDirectory can fail in low-memory situations
SetCurrentDirectory(buff, sizeof(buff));
Warning: using uninitialized memory buff[0]
Problem occurs in call to SetCurrentDirectory
Due to previous failure to check return value

---

Sample defect #3: Comments

- Key technique:
  - Iterative, detailed, defect-specific info
- But can the tool know all of that up front?
  - I tend to think not - instead, provide hooks for people to provide this information

---

PREfast “defect description”

- An XML description of each defect, with
  - Brief description (mandatory; everything else is optional)
  - Additional details
  - Effect of the defect
  - Hypothesis about cause (phrased as question)
  - Severity
  - One or more examples (erroneous and corrected code)
  - Documentation (as XHTML, RTF, or reference)
  - Help URL for more information
  - Owner’s e-mail address
  - GUID
Sample defect #4

```c
void uwmsrsi4(LPCTSTR in) {
    TCHAR buff[100];
    _tcsncpy(buff, in, sizeof(buff));
    /* ... */
}
```

_TCHAR is typedef'd as either char or wchar_t, depending on whether UNICODE is defined

_tcsncpy expands to either strncpy or wcsncpy

---

Sample defect #4: Sample defect #4:

Comments

- Key technique:
  - Additional precision in the warning message
- The fact that it's a stack buffer is important for prioritizing the warning
- There are well-known techniques for exploiting stack buffer overruns

---

Sample defect #5

```c
#define CHECK(hr) \ 
    {if (! SUCCEEDED(hr)) goto bail; }

int uwmsrsi_macro(void) {
    int i;
    CHECK(initialize(&i));
    return ++i;

bail:    return 0;
}
```

---

Sample defect #6

```c
int uwmsrsi_loop(void) {
    int arr[5];
    int i, total = 0;

    arr[0] = 3; arr[1] = 1;

    for (i = 0; i < 5; i++)
        total += arr[i] * i;
    return total;
}
```

---
Additional complexities

- Templates
- Value derivation
- Recursion
- Long functions
- Source code changes

What about more complex properties?

- Race conditions
- Deadlocks
- Arbitrary properties

SLAM “debugger” UI

Summarize vs. navigate?

- Ideally, want to summarize and hide unimportant details
  - But which details are important?
- Developers often think sequentially ...
  - ... even when this is less efficient

Noise

- Noise = “messages people don’t care about”
  - (not just “bogus” messages)
- Usually, noise is worse than missing a defect

Too much noise
  => people won’t use the tool
  == missing all the defects

Noise can result from

- Incorrect requirements
- Integration issues
- Usability issues (e.g., unclear messages)
- Parser incompatibilities
- Analysis inaccuracies
  - ...
Dealing with noise

- Improving analysis is usually not sufficient
  - Sometimes, it's necessary

- Successful user interaction techniques:
  - Prioritization
  - History
  - Filtering
  - Improving presentation, navigation
  - Providing more detail

Message Prioritization

- Which messages correspond to defects that will actually be fixed?
- "Rank": a synthetic metric of a message's "goodness"
  - Better-ranking messages are more likely to identify defects that will actually get fixed
- Multiple dimensions:
  - Severity of consequences
  - Likelihood that message is correct
  - Comprehensibility of message
  - ...

Noise and history

- Noise naturally increases over time
  - People fix the real defects
- A history mechanism avoids these problems
  - Distinguish newly-occurring messages
  - Goal: avoid re-examining noise messages
  - "Fuzzy" notion of equality survives code changes
- Code modification is also possible

Noise Example

- In 1999, 30% of PREfix-reported bugs in the "RAID" bug tracking system were being misdiagnosed
  - Real defects rejected as bogus
  - Unnecessary fixes for false messages
- Why? A surprising reason:
  - The URL describing a PREfix message wasn't self-descriptive, so couldn't be stored in RAID
  - Most people weren't even seeing the PREfix UI!
- Solution:
  - Make the URL's self-descriptive (duh)
  - Automatically store specific information and the URL as part of the RAID entry
- Result: misdiagnosis rate dropped below 5%

Approaches to sets of defects

- List
- GUI
  - List
  - Tabular
  - Tree
  - Filtering, Sorting, ...
- Database and query language
- Statistical analyses

Example: sets of defects

- Textual compiler output
- Compiler output in a dev environment
  - (emacs, Visual Studio, ...)
- Typo.pl: tabular view
- PREfix: tree/list, with filtering and sorting
  - Note: PREfix 4.0 moves to a tabular view
- PREfast: tabular, filtering
Observations

- Others have looked at this problem
  - Spreadsheets
  - OLAP/Data mining
  - "Information agent" work
- Improvements here are very highly leveraged
  - Reducing the number of defects to examine by an order of magnitude is much easier than reducing the time to examine a single defect by an order of magnitude ...
- Prioritization and statistical analysis can make a big difference

User Interaction Issues in Defect Detection Tools

Jon Pincus
PPRC, Reliability Tools
Microsoft Research