Recovering System Specific Rules from Software Repositories

Chadd Williams
Jeff Hollingsworth
Problem

• How much do you know about your 10 year old code base?
  - didn’t someone rewrite the matrix objects?
    • how do you transform an image now?

• Implicit rules build up over time
  - little or no documentation
  - failure to understand implicit rules causes bugs
    • 32% of bugs detected during maintenance\(^1\)

• We can discover implicit rules by looking at code changes

Implicit Rule

- **Function Usage Pattern**
  - how functions are invoked with respect to each other in the source code
  - describe *relationships* between functions
  - static analysis - intraprocedural

```c
HDC hdc = BeginPaint(hwnd, &ps);
if (hdc)
    DrawIcon(hdc, x, y, hIcon);
EndPaint(hwnd, &ps);
```

```c
mdi = HeapAlloc(GetProcessHeap());
if (!mdi)
    HeapFree(GetProcessHeap(), 0, cs);
```

**Called After**

**Conditionally Called After**
Function Usage Pattern Miner

- Find new instances of relationships
  - where that instance was not found in the revision immediately prior

  ```
  int foo(){
    open();
  }
  ```

  ```
  int foo(){
    open();
    read();
  }
  ```

  **new instance of read() called after open()**

- Preliminary filtering heuristic
  - function calls within 10 source lines of code
    - many APIs contain functions that are called in quick succession
    - error handling is near error producing function
Classification of Mined Data

- Each mined instance is classified by how it entered the source code:
  - both of the function calls were added
    • instance added in full
  - one function call was added
    • the added function completed the pairing
    • bug fix? refactoring?
  - neither of the function calls were added
    • deleted code? control flow change?

```c
int foo(){
    open();
    read();
}
```

Change

```c
int foo(){
    open();
    read();
    close();
}
```

Change
Rating Mined Relationships

- **Determine support and confidence for each mined relationship**
  - *confidence of foo() -> bar()*
    - in what percent of instances that start with foo(), is foo() follow by bar()?
  - *support of foo() -> bar()*
    - what percent, of all instances found, are foo() -> bar()?
  - present a sorted list to the user
    - sort on support then confidence
Preliminary Case Study

- **Mined Wine CVS repository**
  - 15,666 unique relationships added > 9 times
  - 862 unique relationships added > 99 times

- **What relationships are found in CVS?**
  - how was it added to the source code?
  - compare to relationships in the latest version of the source code

- **How can this help us find bugs?**

- **Can we mine data for a specific API?**
How do the Top 25 of the lists differ?

- **Most similar to latest version**
  - added both function calls
  - sum of differences in ranking: 91
  - items unique to one list: 8

- **Least similar to latest version**
  - added one function call
  - sum of differences in ranking: 41
  - items unique to one list: 28

### Relationships found in the Latest Version of the Source Code

<table>
<thead>
<tr>
<th>Called After Relationship</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>fprintf</td>
<td>12671</td>
</tr>
<tr>
<td>VariantChangeTypeEx</td>
<td>6700</td>
</tr>
<tr>
<td>GetProcAddress</td>
<td>3605</td>
</tr>
<tr>
<td>HeapFree</td>
<td>3577</td>
</tr>
<tr>
<td>printf</td>
<td>3098</td>
</tr>
<tr>
<td>HeapAlloc</td>
<td>2851</td>
</tr>
<tr>
<td>memcmp</td>
<td>2294</td>
</tr>
<tr>
<td>GetProcessHeap</td>
<td>1985</td>
</tr>
<tr>
<td>GetProcAddress</td>
<td>1747</td>
</tr>
<tr>
<td>GetDlglItem</td>
<td>1742</td>
</tr>
</tbody>
</table>

### Relationships Created By Adding One Function Call

<table>
<thead>
<tr>
<th>Called After Relationship</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>fprintf</td>
<td>2606</td>
</tr>
<tr>
<td>RtlFreeHeap</td>
<td>1782</td>
</tr>
<tr>
<td>RtlAllocateHeap</td>
<td>1251</td>
</tr>
<tr>
<td>HeapFree</td>
<td>1200</td>
</tr>
<tr>
<td>GetProcAddress</td>
<td>1100</td>
</tr>
<tr>
<td>HeapAlloc</td>
<td>816</td>
</tr>
<tr>
<td>GetProcessHeap</td>
<td>768</td>
</tr>
<tr>
<td>HeapFree</td>
<td>480</td>
</tr>
<tr>
<td>memcmp</td>
<td>342</td>
</tr>
<tr>
<td>GetProcessHeap</td>
<td>233</td>
</tr>
</tbody>
</table>
What relationships were found?

- `EnterCriticalSection` -> `LeaveCriticalSection`
  - in latest version: 939 times

How were the instances created?

- add both function calls: 1,277 times
- add one function call: 5 times
- added one function but did not complete the pairing: 82 times

- 78 of these uncompleted pairings were because of the 10 line heuristic

```c
EnterCriticalSection( &(This->lock) );
uRef = ++(This->ref);
if (This->driver)
    IDsCaptureDriver_AddRef(This->driver);
LeaveCriticalSection( &(This->lock) );
```
How can this help us find bugs?

- **Profile of a bug plagued relationship**
  - created often by adding one function call
  - rarely created by adding two function calls

- **Possible bug**
  - `TREEVIEW_UpdateScrollBars` -> `TREEVIEW_Invalidate`
  - update the scroll bars after adding items
  - invalidate the Treeview so it gets redrawn

```c
for ( Each Item In the List ) {
    TREEVIEW_DrawItem(infoPtr, hdc, wineltem);
}

TREEVIEW_UpdateScrollBars (infoPtr);
... return;
```
Mining Relationships for an API

- What relationships are found between functions declared in an API?
- msiquery.c - database access API
  - two sets of functions:
    - MsiFoo( , LPCSTR, ) and MSI_Foo( , LPCWSTR, )
    - MsiDatabaseOpenViewA -> MsiViewExecute
    - MSI_DatabaseOpenViewW -> MSI_ViewExecute
- Heap access functions
  - HeapAlloc(GetProcessHeap(), . . . )
  - HeapAlloc() -> HeapFree()
Future Work

- **Apply our tool to more projects**
  - projects that use a common external library
- **Track removed usage patterns**
- **Better filtering heuristic**
  - control flow based
  - data flow based
- **How do we use the patterns we find?**
  - documentation
  - feed patterns to static source code checkers to find violations

```c
hdc = BeginPaint( hwnd, &ps );
if( hdc )
  DrawIcon( hdc, x, y, hIcon );
EndPaint( hwnd, &ps );
```
Backup Slides
How do the Top 25 of the lists differ?

- **Difference metric**
  - distance between rankings of common items
  - number of items unique to each list

- **Most similar to latest version**
  - Added both function calls
    - sum of differences in ranking: 50
    - items unique to one list: 18

- **Least similar to latest version**
  - Added one function call
    - sum of differences in ranking: 12
    - items unique to one list: 48
Source Code Change History

- We can discover implicit rules by looking at code changes
  - every change is committed
  - changes highlight misunderstood code
  - changes highlight new code

- Studying each commit gives fine-grain knowledge
  - how quickly does a rule emerge?
  - how fast is a rule adopted?
  - how often is it used later?
Debug functions in Wine

- Many of the relationships involve a debug statement
  - overwhelmed the rest of the results
  - filtered from the data
  - future work:
    - what can we determine about the proper use of debug statements?

```c
if (RegOpenKeyA(HKEY, name, &key)) {
    RegCloseKey(key);
    TRACE(message);
}
```
Relations highlighted by CVS mining

- **Data Flow Functionality**
  - `GetDlgItem -> EnableWindow`

```c
case WM_USER:
    EnableWindow (GetDlgItem(...), FALSE);
    EnableWindow (GetDlgItem(...), FALSE);
    EnableWindow (GetDlgItem(...), FALSE);
    SetFocus (GetDlgItem(hwnd, IDC_TOOLBARBTN_LBOX));
    return TRUE;
```
Conditionally Called After

- 3,872 unique patterns added 10 or more times

- **Error handling code**
  - conditionally report error
  - which functions need errors handled

- **Debug code**
  - conditionally call a debug function

```c
if (!(hModule = LoadLibraryExA(fileName, 0, LLDF)))
    WINE_ERR("LoadLibraryExA (%s) failed, %ld\n", fileName, GetLastError());
```
Transitive Patterns

- *called after* may be a transitive pattern
  - only a binary pattern
  - allow larger patterns to be built
  - may need to add more context information

Patterns Identified
- SelectObject called after BeginPaint
- SetTextColor called after SelectObject
- TextOutA called after SetTextColor
- DeleteObject called after TextOutA
- EndPaint called after DeleteObject
Chains of relationships

- Search through the relationships
  - relationships created by adding two functions
  - find relationships of high confidence and support such that:

```c
case WM_USER:
    EnableWindow (GetDlgItem (..), FALSE);
    EnableWindow (GetDlgItem (..), FALSE);
```

```
GetDlgItem() -> EnableWindow ()
```

```
GetDlgItem() -> EnableWindow () -> EnableWindow ()
```

```
GetDlgItem() -> EnableWindow () -> EnableWindow () -> EnableWindow ()
```
- Data flow functionality
  - LoadCursorA -> RegisterClassA
    - in latest version: 42 times
    - add both function calls: 43 times

```c
wClass.hCursor = LoadCursorA (...);
RegisterClassA (&wClass);
```
RtlHeapFree Called After RtlHeapAlloc
Value: 8
dlls/kernel/heap.c
dlls/ntdll/loader.c