Using Clang to Visualize Large Codebases

Nathan Hawes and Ben Barham
Oracle Labs Australia
October 2014
Safe Harbour

The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract.

It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle.

Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.
Code Comprehension

The truth is in the source!
But what if that source is large?

10 million lines
But what if that source is large?

10 million lines

50 lines
But what if that source is large?

10 million lines

50 lines
Code Comprehension in IDEs

• Go to definition, find uses, class overview, type hierarchy, etc.

• IDEs impractical to use for large C/C++ codebases
  – Imprecise language recognition
  – Issues with custom build systems
Current Practice
For large C/C++ codebases

• Text editors and text-search tools
  – vim, emacs
  – grep, sed, cscope
• Fast and simple
• But imprecise →
  – Symbol types, scopes, linking information, preprocessor
• Low-level focus

static VALUE mnew(...) {
  ...
  data->id = rid;
  ... 
}

Find definition
method.h:70
node.h:244
thread_pthread.c:594
(+ 17 more)

Actual definition (14th)
proc.c:21
Frappé Aims

• Provide precise dependency information
  – With easy build integration
• Allow users to specify higher-level queries directly
  – Not just defs or refs
• Show users the broader context of the system
Frappé Overview

Source Code → Dependency Graph → User

EXTRACT → QUERY → DISPLAY
Dependency Graph

• Natural representation of the code
  – Call graphs, type hierarchies, control flow graphs, etc.

• Nodes and edges
  – Build system: modules, files, and linking information between them
  – File system: directories and files
  – Preprocessor: includes, macros, their expansion and interrogations
  – Symbols: functions, locals, types, and relations between them

• High-level questions become graph queries
Go to Definition
Go to Definition

main() \(\rightarrow\) .buf \(\rightarrow\) scope{}

writes \(\rightarrow\) contains

reads \(\rightarrow\)

foo() \(\rightarrow\) .buf \(\rightarrow\) bar()
Go to Definition

```
main() writes .buf

buf contains scope{}

foo() reads buf

bar() reads .buf
```

Copyright © 2014 Oracle and/or its affiliates. All rights reserved. | Oracle Confidential - Restricted
Find References

main() writes -> .buf contains -> scope{}

foo() reads -> .buf

bar() reads -> .buf
Find References

main() \rightarrow \text{writes} \rightarrow.\text{buf} \rightarrow \text{contains} \rightarrow \text{scope{}}

\rightarrow \text{reads} \rightarrow \text{foo()}

\rightarrow \text{reads} \rightarrow \text{bar()}

contains \rightarrow \text{buf} \rightarrow \text{reads}

Copyright © 2014 Oracle and/or its affiliates. All rights reserved. | Oracle Confidential - Restricted
Find References
Impact Estimation

```
insert()  pop()  first()  init()

#PREV  #FIRST  #NULL
```

- expand
- expand
- expand
Impact Estimation
Impact Estimation

- `insert()`
- `pop()`
- `first()`
- `init()`

- `#PREV`
- `#FIRST`
- `#NULL`
Impact Estimation

- `insert()`
- `pop()`
- `first()`
- `init()`

- expands
- `#PREV`
- `#FIRST`
- `#NULL`
Frappé Architecture

Source Code → .fo → Server → User

- EXTRACTOR
- IMPORTER
- EDITOR PLUGINS
- CLI SCRIPTS
- WEB UI
Frappé Architecture

Source Code -> EXTRACTOR -> .fo -> IMPORTER -> Server -> User

- EDITOR PLUGINS
- CLI SCRIPTS
- WEB UI
Extractor
Simple Build Integration

Compiler Wrappers

Native Compiler

Clang + Plugin

.o

.fo
Extractor
Simple Build Integration

Compiler Wrappers

Native Compiler

Clang + Plugin

.o

.fo
Extractor

Simple Build Integration

Compiler Wrappers

Native Compiler

Clang + Plugin
Extractor
Clang Plugin

• PPCallbacks
  – Includes, macros, their expansions and interrogations

• RecursiveASTVisitor
  – Visit all declarations, types, and expressions

• Easy to use interface
  – Provides detailed location information
Frappé Architecture

Source Code -> EXTRACTOR -> .fo -> IMPORTER -> Server

User

EDITORS PLUGINS
CLI SCRIPTS
WEB UI
Frappé Architecture

Source Code → .fo → Server → User

- EXTRACTOR
- IMPORTER
- EDITOR PLUGINS
- CLI SCRIPTS
- WEB UI
Frappé Architecture

Source Code → EXTRACTOR → .fo → IMPORTER → Server → User

- EDITOR PLUGINS
- CLI SCRIPTS
- WEB UI
Code Maps

Visualising large codebases
Code Maps
Using a cartographic map metaphor

• Continent/country/state/city → module/sub-module/file/function
• Distinctive shape and positions serve as landmarks
• Can overlay a variety of information
Overlay Search Results

Visual filtering

Contextual search
java.lang.RuntimeException: bad
at Foobar.setup(Foobar.java:74)
at Foobar.launch(Foobar.java:43)
at Bar.launch(Bar.java:39)
at Bar.bar(Bar.java:97)
at Foo.foo(Foo.java:35)
at Main.main(Main.java:104)
Overlay Metrics

Fine granularity

Coarse granularity
But how?
Existing Approaches

Deline, R. *Staying oriented with software terrain maps* In proc. of the workshop of visual languages and computation, 2005

Kuhn, A.; Erni, D.; Loretan, P.; Nierstrasz, O. *Software cartography: thematic software visualization with consistent layout* Journal of Software Maintenance and Evolution, 2010
Input

• Abstraction hierarchy
  – Abstracts files into higher level groupings
  – Use directory structure by default

• Dependency graph
  – Represents dependencies between files as a weighted edge
  – Use references

Frappé: a code comprehension tool for large codebases
Map Generation

Graph layout

Implicit surface generation

Surface subdivision

Recursive subdivision
Map Generation

Graph layout


Recursive subdivision

Demonstration

Linux x86_64-linux-gnu

phash_insert
of hashmd.c

Called by hash_builtin
d of ./buildconf/./buildconf/hashdef on line 176

Called by add_hashed_command
d of ./buildconf/./buildconf/sh.sh on line 208

Called by search_for_command
d of findmd.c on line 360

Declared by phash_insert
d of hashmd.h

Called by assign_hashcmd
do variables.c on line 1587

Called by hashmd.c
do hashmd.h on line 92

Has parameter filename
do hashmd.h

Has parameter full_path
do hashmd.h

Has parameter check_dir
do hashmd.h

Has parameter found
do hashmd.h

ps_exec_preferred(ps_nodirs);

/* Place FILENAME (key) and FULL_PATH (data->path) into the
hash table. CHECK_DOT if non-null is for future calls to
phash_search(); it means that this file was found
in a directory in $PATH that is not an absolute pathname.
FOUND is the initial value for times_found. */

void
phash_insert (filename, full_path, check_dot, found)
{

}

else
command = find_user_command (pathname);
if (command && hashing_enabled && temp_path == 0 && (flags & 1))
Future Work

• More detailed dependency graph
  – Find calls where third argument is macro FLAG
  – Find all functions the pointer fptr could point to

• More overlays
  – Test coverage, profiling data

• Store multiple versions
  – Impact estimation
  – Code map evolution (stability)
Frappé

Nathan Hawes and Ben Barham
nathan.hawes@oracle.com
ben.barham@oracle.com

Oracle Labs Australia
http://labs.oracle.com/locations/australia

Research Director
cristina.cifuentes@oracle.com