CodeCompass
an Open Software Comprehension Framework

Zoltán Porkoláb¹², Dániel Krupp¹, Tibor Brunner², Márton Csordás²

https://github.com/Ericsson/CodeCompass

Motto: If it was hard to write it should be hard to understand
-- unknown programmer

¹Ericsson Ltd, ²Eötvös Loránd University, Budapest, Hungary
Agenda

• Comprehension as a cost factor
• Why development tools are not perfect for comprehension?
• Requirements
• Architecture
• A few workflows
• Restrictions
• Experiences
• Further planes
# Comprehension is a major cost factor

<table>
<thead>
<tr>
<th>Research</th>
<th>Effort for comprehension</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM (Corbi, 1989)</td>
<td>Over 50% of time</td>
</tr>
<tr>
<td>Bell Labs (Davison, 1992)</td>
<td>New project members: 60-80% of time, drops to 20% as one gains experience</td>
</tr>
<tr>
<td>National Research Council in Canada (Singer, 2006)</td>
<td>Over 25% of time either searching for or looking at code</td>
</tr>
<tr>
<td>Microsoft (Hallam, 2006)</td>
<td>Equal amount of time as design, test</td>
</tr>
<tr>
<td>Microsoft (La Toza, 2007)</td>
<td>Over 70% of time</td>
</tr>
<tr>
<td>Microsoft (Cherubini, 2007)</td>
<td>95%~ significant part of job</td>
</tr>
<tr>
<td></td>
<td>65%&lt; at least once a day</td>
</tr>
<tr>
<td></td>
<td>25%&lt; multiple times of a day</td>
</tr>
</tbody>
</table>
Using tools
Using tools
Using tools
Using tools
Comprehension requires specific toolset

<table>
<thead>
<tr>
<th>Development of code</th>
<th>Understanding code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Writing new code (support: code completion, etc.)</td>
<td>Reading and navigating inside code</td>
</tr>
<tr>
<td>Intentions are clear</td>
<td>Intensions are weak</td>
</tr>
<tr>
<td>Editing only a few files at the same time</td>
<td>Frequently jumping between different files</td>
</tr>
<tr>
<td>Working on the same abstraction level for a while</td>
<td>Jumping between various abstraction levels (Google map of code)</td>
</tr>
<tr>
<td>Edit, compile, fix</td>
<td>Visualize</td>
</tr>
</tbody>
</table>
Some existing tools

- **Web-based**
  - OpenGrok
  - Woboq (deep analysis)
  - ...

- **Fat-client**
  - Understand (+edit)
  - CodeSurfer
  - ...

- **IDE-based**
  - Eclipse
  - NetBeans
  - QtCreator
  - VisualStudio
  - ...
Required features

- Deep analysis + build information -> using a real parser
- Fast text based feature location
- Architectural information
- Textual summaries (types, variables, functions, macros)
- Various (interactive) visualizations
- Scalable (>10 million LOC)
- Most actions should be fast (< 1-2 sec)
- Permalinks for communication with fellow developers
- Gathering all available information: code history, metrics, ...
- Open, extensible platform
First experimental version: store AST

- AST contains most of the required information
- Natural output of Clang
- Problem: size!
  - 40GB for LLVM project AST dump + indexes, etc... ->100 GB
  - 1:500 ratio between source and CodeCompass DB size
- Not scalable
- Future work:
  - Detecting identical sub-trees (e.g. of headers)
  - NoSQL database?
- Fat client
Final approach: Store named entities

- Names: the most natural target of user actions
- We store
  - Class/function/variable declarations, definitions, usage
  - References to names are stored as hash values
  - Source file as it is (keeping original formatting)
  - Build information
- Scalable
  - 1:30-50 ratio between source and CodeCompass DB size
  - Full LLVM CodeCompass DB with indexes 13 GB in postgres
- A few addition was required
  - Assignment, parameter lists: detecting read/write relations of variables
  - Inheritance, pointer indirections, typedefs, etc...
- Web-based client
## Performance

<table>
<thead>
<tr>
<th></th>
<th>Tiny XML 2.6.2</th>
<th>Xerces 3.1.3</th>
<th>CodeCompass v4</th>
<th>Ericsson TSP product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source code size [MiB]</td>
<td>1.16</td>
<td>67.28</td>
<td>182</td>
<td>3 344</td>
</tr>
<tr>
<td>Search database size [MiB]</td>
<td>0.88</td>
<td>37.93</td>
<td>139</td>
<td>7168</td>
</tr>
<tr>
<td>PostgreSQL DB size [MiB]</td>
<td>15</td>
<td>190</td>
<td>2144</td>
<td>7729</td>
</tr>
<tr>
<td>Build time [s]</td>
<td>2.73</td>
<td>361</td>
<td>2024</td>
<td>-</td>
</tr>
<tr>
<td>CC Parse time [s]</td>
<td>21.98</td>
<td>517</td>
<td>6409</td>
<td>-</td>
</tr>
<tr>
<td>Text/definition search [s]</td>
<td>0.4</td>
<td>0.3</td>
<td>0.43</td>
<td>2</td>
</tr>
<tr>
<td>C++ get usage of a type [s]</td>
<td>1.4</td>
<td>2</td>
<td>2.3</td>
<td>3.1</td>
</tr>
</tbody>
</table>
Architecture
How to use?

• Fast feature location using text/definition/log search
• Explore the environment of the focus point
  – Info tree
  – Interactive call graphs
  – Virtual functions and function pointers
• Understand the code history
• Understand higher level architecture
• Explore related static analysis results/code metrics
DEBUG INFO: TSTHan: sys_offset=-0.019821, drift_comp=-90.4996, sys_poll=5

conprint(" -------------- NTP hourly stats --------------\n");
ntpq_p();
conprint(" sys_offset=%s, drift_comp=%s, sys_poll=%d\n",
  fptoa(&sys_offset, 6), fptoa(drift_comp, 4), sys_poll);
conprint(" --------------\n");
// For now we only support one workspace

#include "model/cxx/cppastnode.h"

namespace cc {
namespace service {

namespace language {

CppServiceHandler::CppServiceHandler(const CppServiceHelper& helper) :
    helper(helper) {

    void CppServiceHandler::getAstNodeInfoByPosition(AstNodeInfo& _return,
        const core::FilePosition& fps, const std::vector<std::string>& filters)

        _return = helper.getAstNodeInfoByPosition(fps, filters);

    void CppServiceHandler::getInfoBox(InfoBox& _return,
        const core::AstNodeId& astNodeId) {

        _return = helper.getInfoBox(astNodeId);

    void CppServiceHandler::getInfoBoxByPosition(InfoBox& _return,
        const core::FilePosition& fps, const std::vector<std::string>& filters)

        _return = helper.getInfoBoxByPosition(fps, filters);

    void CppServiceHandler::getAstNodeInfo(AstNodeInfo& _return,}
// MMCreateDisasmCPU

LLVMCreateDisasmCPUFeatures(const char *TT, const char *CPU,  
const char *Features, void *DisInfo, int TagType,  
LLVMInfoCallback GetOpInfo,  
LLVMSymbolLookupCallback SymbolLookup) {  

// Get the target.
std::string Error;
const Target *TheTarget = TargetRegistry::lookupTarget(TT, Error);  
if (TheTarget)  

  Assuming 'TheTarget' is non-null
  return nullptr;

const MCRegisterInfo *MRI = TheTarget->createMCRegInfo(TT);  
if (MRI)  

  Assuming 'MRI' is non-null
  return nullptr;

// Get the assembler info needed to setup the MCContext.
const MCAsmInfo *MAI = TheTarget->createMCAsmInfo(MRI, TT);  
if (MAI)  

  Assuming 'MAI' is non-null
  return nullptr;

const MCInstrInfo *MII = TheTarget->createMCInstrInfo();  
if (MII)  

  Assuming 'MII' is non-null
  return nullptr;

const MCSUBTargetInfo *STI =  
TheTarget->createMCSUBTargetInfo(TT, CPU, Features);  
if (STI)  

  Assuming 'STI' is non-null
  return nullptr;

// Set up the MCContext for creating symbols and MCExpr's.
MCContext *ctx = new MCContext(MAI, MRI, nullptr);  

Memory is allocated
```cpp
#include <boost/program_options.hpp>
#include <boost/algorithm/string.hpp>

LogCommand::LogCommand()
    : compiler()
    , objects()
    , params()
{
}

LogCommand::LogCommand(const LogCommand & arg_1)
    : compiler(arg_1.compiler)
    , objects(arg_1.objects)
    , params(arg_1.params)
{
}

LogCommand::LogCommand(LogCommand && arg_1)
    : compiler(std::move(arg_1.compiler))
    , objects(std::move(arg_1.objects))
    , params(std::move(arg_1.params))
{
}

LogCommand& LogCommand::operator=(LogCommand && arg_1)
{
    this->compiler.operator=(std::move(arg_1.compiler));
    this->objects.operator=(std::move(arg_1.objects));
    this->params.operator=(std::move(arg_1.params));
    return *this;
}

LogCommand::~LogCommand()
{
}
```
Experiences with CodeCompass

- Open source since summer 2016
- Mainly used inside Ericsson and in University
- Replacing/extending OpenGrok
- Voluntary-based: No policy to enforce using CodeCompass
- ~15 million LOC parsed inside Ericsson
- ~300 users
- Frequently used investigate CodeChecker results
- ... and by architects to get a system level view
Experiences with CodeCompass

Tool usage in the four most active teams

- Function definitions and calls
- Variable binding and references
- Class or object relationships
- Code change responsibility, version control
- Higher-level models of software

Legend:
- CodeCompass
- OpenGrok
- Low-level tools (e.g.: grep)
- By browsing the code
- By asking colleagues
Future plans

• Incremental parsers: from “Snapshot” view to editable
  – Pointer analysis
  – Reparse: source + build info -> rebuild AST on demand
• Complex query language
• User specific information
  – Review notes, reminders, comprehension map
  – Personal “Comprehension map” (incl. internal links)
• Ideal for starting a Clang-based server implementing C/C++
  LSP (Language Server Protocol), like ClangD
• Feel free to contribute
  – New language parsers
  – New GUI functionality
• Language Server Protocol (LSP) interface
Summary

• Scalable (up to 10 million LOC)
• Most actions are completed ( < 1-2 sec)
• Textual summaries (types, functions, variables, macros)
• Various (interactive) visualizations on the code
• Architectural information (based on build info)
• GIT history
• Permalinks to communicate with other developers
• CodeChecker integration to show Clang SA results
• Java, Python support (less mature)
• Easy to extend