

Understanding Large and Unfamiliar Codebases

**Mike Shah &
Chris Croft-White**

2025



The international
C++ conference
in the UK, by the sea

Understanding Large and Unfamiliar Codebases

Mike Shah and Chris Croft-White

Chris Croft-White
Staff Solutions Architect
chris@undo.io



Web : mshah.io
YouTube : www.youtube.com/c/MikeShah
Social : mikeshah.bsky.social
Courses : courses.mshah.io
Talks : <http://tinyurl.com/mike-talks>

60 minutes | Intermediate Audience
09:15 - 10:15 Tuesday, June 24, 2025

Abstract (Which you already read :))

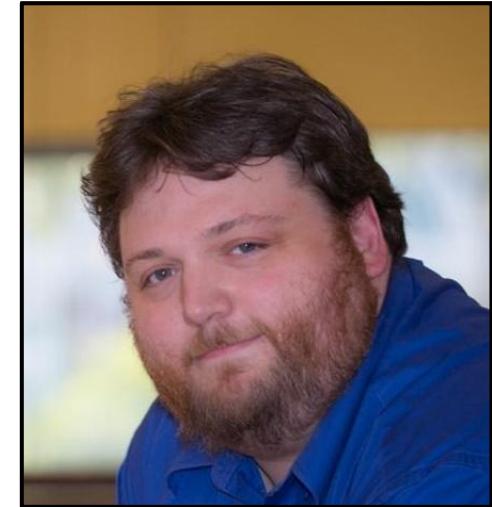
Talk Abstract: It's your first day on the job as a new employee. You set up your workstation and then download a repository of over 1,000,000 lines of code. Even more intimidating, the code has been around for 20 years and has parts of it in legacy C++ and also using tentative C++ 26 features from various libraries. You feel overwhelmed! Don't fret however! In this talk, I provide you a collection of tools to help software engineers of all levels understand what is going on in large unfamiliar codebases. The audience will leave this talk with a few simple and advanced tricks for navigating large and complicated codebases.

<https://cpponsea.uk/2025/session/understanding-large-and-unfamiliar-codebases>

Your Tour Guide(s) for Today

Chris Croft-White

- **Current Role:** Staff Solutions Architect at [Undo.io](#) (5 years)
 - Supporting new customers and creating content to help people debug software
 - Background: ~20 years a pre-sales engineer
- **Available for:**
 - **Time Travel Debugging**, happy to explain the technology, its uses, and get you started
 - **Sleep**, it's dreadfully lacking and top of the list if I ever get any of this spare time I keep hearing about
- **Fun:**
 - Computer games, travelling, food & drink, learning cool new technologies, meeting new people.



Your Tour Guide(s) for Today

Mike Shah

- **Current Role:** Teaching Faculty at **Yale University** (Previously Teaching Faculty at Northeastern University)
 - **Teach/Research:** computer systems, graphics, geometry, game engine development, and software engineering.
- **Available for:**
 - **Contract work** in Gaming/Graphics Domains
 - e.g. tool building, plugins, code review
 - **Technical training** (virtual or onsite) in Modern C++, D, and topics in Performance or Graphics APIs
- **Fun:**
 - Guitar, running/weights, traveling, video games, and cooking are fun to talk to me about!



Web

www.mshah.io

YouTube

<https://www.youtube.com/c/MikeShah>

Non-Academic Courses

courses.mshah.io

Conference Talks

<http://tinyurl.com/mike-talks>

Question to the Audience:

What's the biggest codebase you have ever worked on?

(i.e. the approximate lines of code)

What's the biggest codebase you have every worked on? (2/3)

Follow-up question(s):

1. Did that codebase ever get smaller?
2. Did that codebase become less complex over time?
3. How well was that codebase documented?
 - a. i.e. Could I go find tutorials, internal wiki pages, or internal/external videos explaining the code or architecture?
4. Did anyone say “Scrap the project -- let’s rebuild it from scratch”?

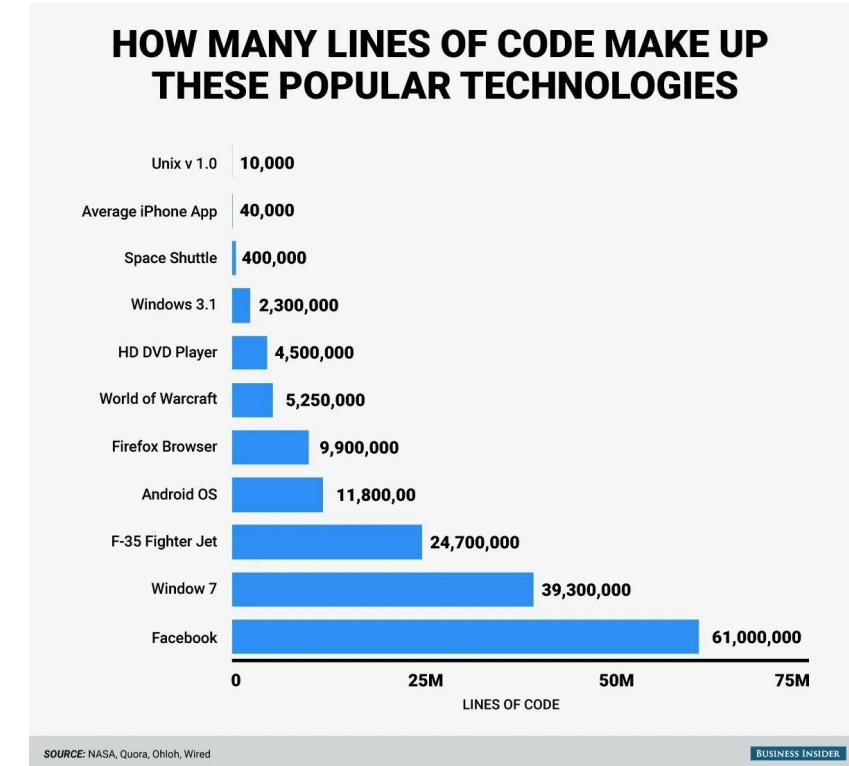
What's the biggest codebase you have every worked on? (3/3)

Follow-up question(s):

1. Did that codebase ever get smaller?
2. Did that codebase become less complex over time?
3. How well was that codebase documented?
 - a. i.e. Could I go find tutorials, internal wiki pages, or internal/external videos explaining the code or architecture?
4. Did anyone say “Scrap the project -- let’s rebuild it from scratch”?
 - a. ^Sometimes we feel this way -- but it can be hard to convince a business to throw away thousands/millions of lines of code :)

How Big are Codebases (1/2)

- Today we are talking about **large** and **unfamiliar** codebases
- I think we have an idea of what this means, but the scale I'm interested in is anything from tens of thousands of lines, to billions of lines of code (e.g. monorepo)
 - i.e. Big enough you cannot keep the whole design in your head
- Note:
 - The figure on the right gives a very *rough* approximation of various codebases.



How Big are Codebases (2/2)

- For a reference of scale --
 - “**A million lines of code**, if printed, would be **about 18,000 pages of text**. That’s **14x the length of War and Peace**.”
 - <https://www.visualcapitalist.com/millions-lines-of-code/>
- That’s a lot of ‘text’, architecture, subsystems, etc. to keep in our head!

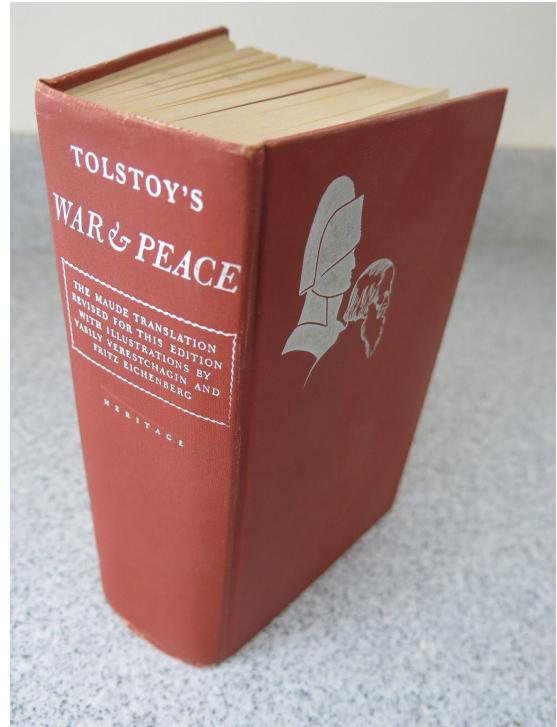


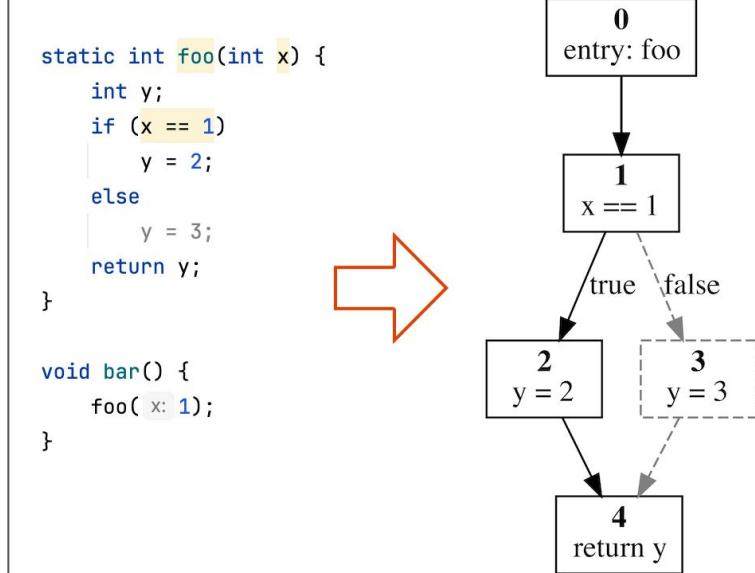
Figure source:
https://substackcdn.com/image/fetch/f_auto,q_auto,g_auto,fl_progressive,steep/https%3A%2F%2Fsubstackcdn.com%2Fpublic%2Fimages%2Fd6b32b44-199e-498f-840b-4cf789c6503c-1536x2048.jpg

Question to the Audience:
What makes a codebase hard to understand?

What makes a codebase hard to understand? (1/3)

- **Understanding the logical control flow** (and/or data flow)
 - Where are the entry points?
 - Are there threads / fibers / coroutines
 - What are the ‘hot spots’ and important subsystems
 - Where does ‘state’ change in the program
 - What is a specific function doing?
 - Is the name of the function reflective of what it does?
 - Are the parameters named well and consistent?

```
static int foo(int x) {  
    int y;  
    if (x == 1)  
        y = 2;  
    else  
        y = 3;  
    return y;  
}  
  
void bar() {  
    foo( x: 1);  
}
```



To the right is a visualization of a ‘Data Flow Analysis’. This is useful once you know the isolated fragment(s) of code you want to look at (but this visualization technique otherwise does not work on the entire codebase). <https://blog.jetbrains.com/clion/2023/11/striving-for-better-cpp-code-part-i-data-flow-analysis-basics/>

What makes a codebase hard to understand? (2/3)

- The **pure scale** of software and its relation to complexity
 - i.e. The number of lines of source code
- Different variations of code
 - e.g. Legacy C++ mixed with C++26
 - Different ‘design patterns’ or architecture design within the source code

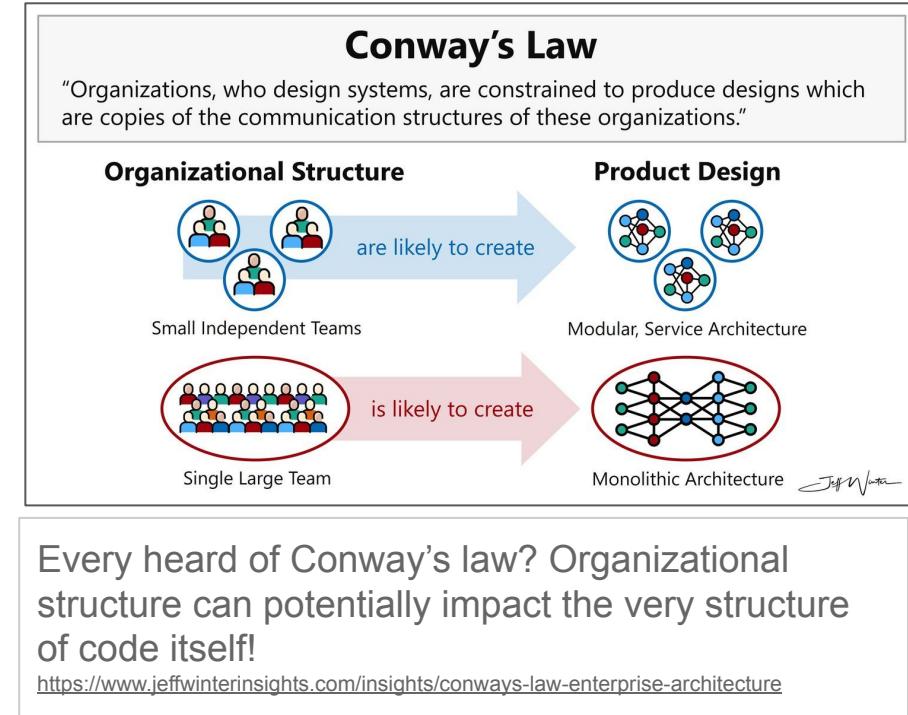
```
1 <div class="row">
2   <div class="col-md-6 col-sm-12">
3     
4     <p>School Excellence Award: Presented to SMIT for outstanding achievement in academics, extracurricular activities, and community service. This award recognizes the hard work and dedication of the entire school community in creating a positive and successful learning environment. (P)
5   </div>
6 
7   <div class="col-md-6 col-sm-12">
8     
9     <p>OECD School Grant is offered by the Philippine government that provides financial assistance to students pursuing higher education in colleges and universities in the Philippines. (P)
10  </div>
11 
12 </div>
13 
14 </div>
15 
16 </div>
17 
18 </div>
19 
20 </div>
21 
22 </div>
23 
24 </div>
25 
26 </div>
27 
28 </div>
29 
30 </div>
31 
32 </div>
33 
34 </div>
35 
36 </div>
37 
38 </div>
39 
40 </div>
41 
42 </div>
43 
44 </div>
45 
46 </div>
47 
48 </div>
49 
50 </div>
51 
52 </div>
53 
54 </div>
```



‘source code’ we visualize as text -- printing out the code will look something like the bottom-right image.

What makes a codebase hard to understand? (3/3)

- **Human factors** can additionally contribute to making your life harder
 - Team size / structure
 - Accessibility to code that you may not own.
 - Is all the code in the same repo, or do you need permission to view other libraries?)
 - No version history or documentation of the ‘why’ something evolved
 - Undocumented code that only a ‘senior engineer’ understands, and everyone is afraid to touch.

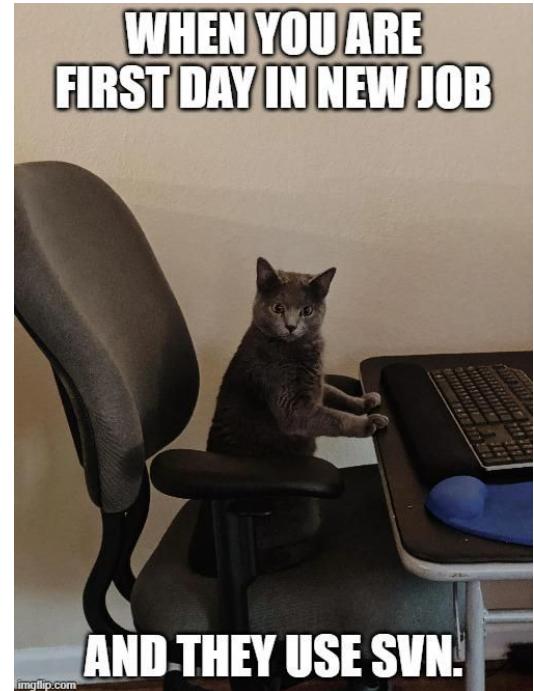


**Question to the Audience:
Why do folks think I'm talking about this topic?**

Why do folks think I'm talking about this topic? (2/3)

- **Answer:** It was an honest difficulty of mine when I started out as a software engineer!

- (more on next slide)



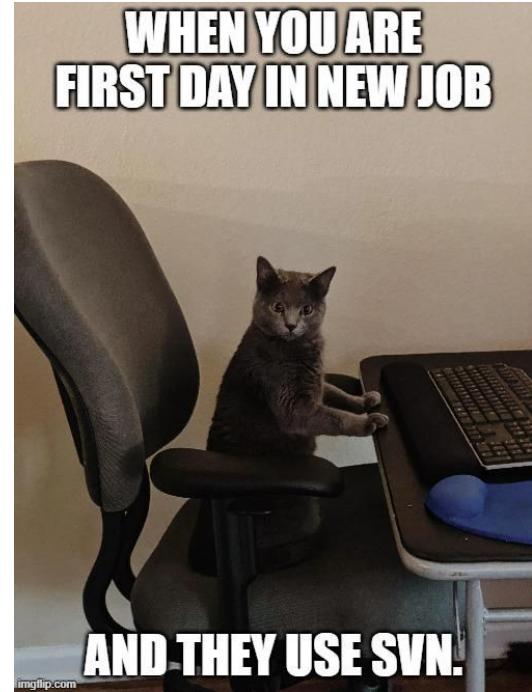
Ah, I have found memories of TortoiseSVN -- it's still active as far as I know!

<https://preview.redd.it/whatididdone-v0-tv7l0uo8hdvc1.jpeg?width=320&crop=smart&auto=webp&s=d56747737e31e0cf8ba5caa6cebfbfebb9f173e>

Why do folks think I'm talking about this topic? (3/3)

- **Answer:** It was an honest difficulty of mine when I started out as a software engineer!

- I remember the difficulty and overwhelming feeling as an intern learning new tools & code!
 - (On occasion I consult on short contracts -- so this is still relevant today!)
 - I suspect there are others in the audience who remember this daunting feeling too
- It probably feels a bit embarrassing to admit that feeling on the first day of work!
 - We should not be afraid to ask questions however!



Ah, I have found memories of TortoiseSVN -- it's still active as far as I know!

<https://preview.redd.it/whatididdone-v0-tv7l0uo8hdvc1.jpeg?width=320&crop=smart&auto=webp&s=d56747737e31e0cf8ba5caa6cebfbfefbb9f173e>

Where do we learn to read code?

- The other reason is that we are not always taught many tools or how to read large codebases in school or our workplace
 - **Let me be clear -- I'm not necessarily blaming schools** -- there is a lot of computer science to cover.
 - Some large projects in school are maybe 2-5k lines of code -- but again not quite the scale we are talking about for millions of lines of code
 - **Note:** I (and many other professors) do often expose students to large codebases at various points -- probably some professors out their doing an excellent job as well!

[./missing-semester](#) | [lectures](#) | [about](#)

The Missing Semester of Your CS Education

Classes teach you all about advanced topics within CS, from operating system to machine learning, but there's one critical subject that's rarely covered, and instead left to students to figure out on their own: proficiency with their tools. We'll teach you how to master the command-line, use a powerful text editor, use fancy features of version control systems, and much more!

Students spend hundreds of hours using these tools over the course of their education (and thousands over their career), so it makes sense to make the experience as fluid and frictionless as possible. Mastering these tools not only enables you to spend less time on figuring out how to bend your tools to your will, but it also lets you solve problems that would previously seem impossible complex.

Read about the [motivation behind this class](#).

MIT, Tufts, Yale, UMich, etc. have various courses in this spirit for helping students learn tools that I know of.

<https://missing.csail.mit.edu/>

Understanding Large and Unfamiliar Codebases

- Now from this talk's title, the word 'understanding' is important
- We might also mean:
 - The **performance** of some aspect
 - The **memory safety** or **security** of a system
 - Or really any other metric about a piece of software.
- That said -- today we are going to focus more on understanding --
what our code is logically doing -- and what tools can help!

Today we'll focus on these 3 categories

1. **Understanding the control flow**
2. The **pure scale** of software and its relation to complexity
3. **Human factors** can additionally contribute to making your life harder

Given:
An unknown (open-source) codebase



Real World Example: Transport Tycoon (1/2)

- Today I'll give be giving some tips on understanding larger source code projects.
- We will look at the **video game** -- **Transport Tycoon**
 - Some of you may be familiar with it, and it is perhaps better if you are not!
 - (Although, I will say it's a wonderfully charming game...)



Real World Example: Transport Tycoon (2/2)

- If you are following along at home, you can follow along by picking Transport Tycoon (<https://www.openttd.org/>) or a similar project.
- A good candidate project to practice with is:
 - Open Source
 - Many collaborators, perhaps of different levels of seniority/experience
 - Long-lived, and lots of lines of code!
 - Ideally with some ‘git’ history, but not strictly necessary (i.e. may be fun to practice applying patches, fixing bugs, etc.)
 - And again -- perhaps even better if you’re not 100% familiar with the project
 - (but perhaps have some domain experience or interest in the domain)



Project Scale

- The ‘**cloc**’ tool will count the lines of code in a project
 - This may give some insights as well if there are multiple languages used
- Our stats for the Open Transport Tycoon game for example are:
 - 188,428 C++
 - 123,108 C/C++ headers
 - 670 lines of C
 - etc.
- Let’s agree this is a large size codebase to investigate!

3138 text files.				
2402 unique files.				
1371 files ignored.				
github.com/AlDanial/cloc v 1.98 T=1.86 s (1293.4 files/s, 534526.5 lines/s)				
Language	files	blank	comment	code
Text	86	53266	0	308781
C++	499	40695	47958	188428
C/C++ Header	768	23071	41636	123108
D	477	0	0	93912
make	59	6122	4985	16050
CMake	304	1380	1520	11791
INI	21	614	149	5023
JSON	5	1	0	3140
HTML	5	278	1	3106
Squirrel	33	364	238	3050
Markdown	26	966	32	3011
YAML	25	480	117	2819
Objective-C++	4	499	432	1742
SVG	2	1	2	1353
C	1	149	61	670
awk	1	41	32	217
XML	4	3	12	199
diff	2	3	31	174
Python	2	68	55	168
TypeScript	61	0	0	122
PowerShell	5	34	27	113
JavaScript	1	16	24	86
Tcl/Tk	1	22	32	52
Bourne Shell	3	17	32	41
reStructuredText	1	5	0	22
DOS Batch	5	4	0	18
Dockerfile	1	1	0	3
SUM:	2402	128100	97376	767199

1. **Understanding the control flow**
2. The pure scale of software and its relation to complexity
3. Human factors can additionally contribute to making your life harder

Understanding Control Flow: Run our program!

- The **very first thing to do**, is run whatever game/application/service/system/etc you are building
 - This sounds silly and is obvious, but you should have a high level understanding of what your program actually does is key.
- **I will assume** that you can build the software using whatever build system you are provided.
 - And I fully acknowledge, this it is not often trivial to build!



Understanding Control Flow: Entry Points (1/3)

- One of the first things I like to do -- is to anchor around the ‘main()’ function
- From the main() function, you will often get hints as to the overall software architecture
 - (i.e. are there plugins loaded, is there an event loop, etc.)
 - Some things you can actively log are:
 - What happens before main (any initialization code)
 - What happens after (cleanup code)

```
21 int CDECL main(int argc, char *argv[])
22 {
23     /* Make sure our arguments contain only valid UTF-8 characters. */
24     for (int i = 0; i < argc; i++) StrMakeValidInPlace(argv[i]);
25
26     CrashLog::InitialiseCrashLog();
27
28     SetRandomSeed(time(nullptr));
29
30     signal(SIGPIPE, SIG_IGN);
31
32     return openttd_main(argc, argv);
33 }
```

In my opinion, a good software engineering practice is to keep your ‘main’ relatively clean. Observe here, the main entry point returns to the ‘real main (openttd_main)’

In this particular software, you’ll observe there are also different ‘main’ functions per operating system.

Understanding Control Flow: Entry Points (2/3)

- Here's an example of finding 'main()' with grep
- grep is a tool for searching for patterns within files

- e.g.
 - grep -irn "main"
 - This does a case insensitive(i) search, recursively through the directory (r), and provides line numbers (n) with the filenames
- This gives us around **37 places to look**

```
mike@mike-MS-7B17:openttd-14.1$ grep -irn "main()".
grep: ./build/openttd: binary file matches
grep: ./build/openttd test: binary file matches
./build/CMakeFiles/_CMakeLTOTest-CXX/src/main.cpp:3:int main()
./build/CMakeFiles/3.28.3/CompilerIdC/CMakeCompilerId.c:848:void main() {}
grep: ./build/CMakeFiles/openttd_lib.dir/src/video/opengl.cpp.o: binary file matches
./os/emscripten/cmake/FindLibLZMA.cmake:8:    int main() { return 0; }
./github/unused-strings.py:184:def main():
./github/unused-strings.py:220:    main()
./github/script-missing-mode-enforcement.py:53:def main():
./github/script-missing-mode-enforcement.py:71:    main()
./src/table/opengl_shader.h:17: "void main() {}",
./src/table/opengl_shader.h:32: "void main() {}",
./src/table/opengl_shader.h:45: "void main() {}",
./src/table/opengl_shader.h:56: "void main() {}",
./src/table/opengl_shader.h:67: "void main() {}",
./src/table/opengl_shader.h:80: "void main() {}",
./src/table/opengl_shader.h:115:           "void main() {}",
./src/table/opengl_shader.h:138:           "void main() {}",
./src/table/opengl_shader.h:164:           "void main() {}",
./src/table/opengl_shader.h:191:           "void main() {}",
./src/os/macosx/macos.mm:211: * These are called from main() to prevent a _NSAutoreleaseNoPool error when
./src/openttd.cpp:123:   * openttd main() is never executed. */
```

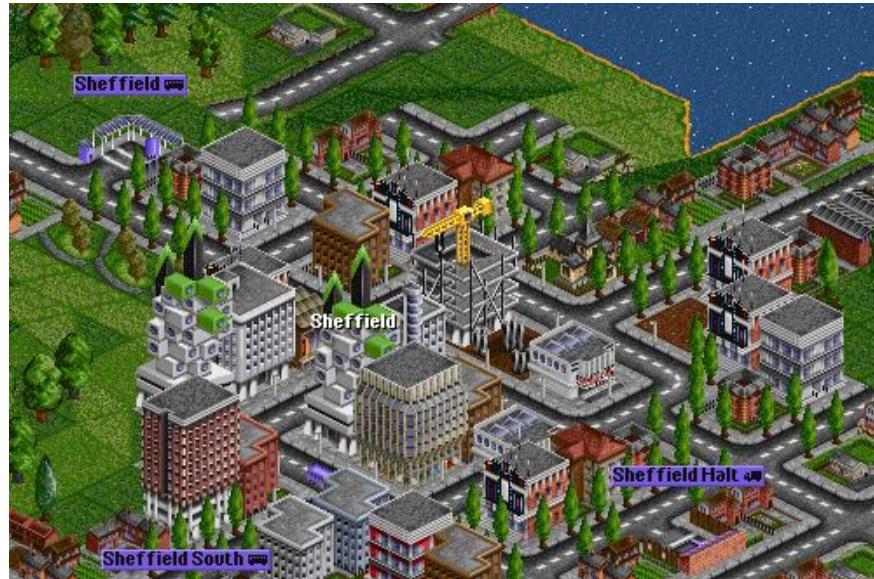
Understanding Control Flow: Entry Points (3/3)

- **grep** is quite powerful, so you will want to spend some time playing with it
- Here's a more specific search that gives us around 7 **places to look**
 - `grep -irn -I --include=*.cpp "main()" .`
 - `-I` tells us to ignore binary files
 - `--include=*.cpp` tells us to look at .cpp files

```
mike@mike-MS-7B17:openttd-14.1$ grep -irn -I --include=*.cpp "main()" .
./build/CMakeFiles/_CMakeLTOTest-CXX/src/main.cpp:3:int main()
./src/openttd.cpp:123: * openttd_main() is never executed. */
./src/landscape.cpp:1322:         finder.Main();
./src/video/sdl2_v.cpp:623:             * openttd_main() is never executed. */
./src/3rdparty/md5/md5.cpp:46: 2002-03-11 lpd Corrected argument list for main(), and added int return
./src/pathfinder/npf/aystar.cpp:245:int AyStar::Main()
./src/pathfinder/npf/npf.cpp:1044:     [[maybe_unused]] int r = _npf_aystar.Main();
```

Pro Tip: Debug Symbols (1/2)

- grep is going to be quite useful, but it can sometimes be a ‘treasure hunt’ if we do not know what we’re looking for
- **Remember -- we can simply run our program**
 - And you did compile with debug symbols (i.e. ‘-g’) right?



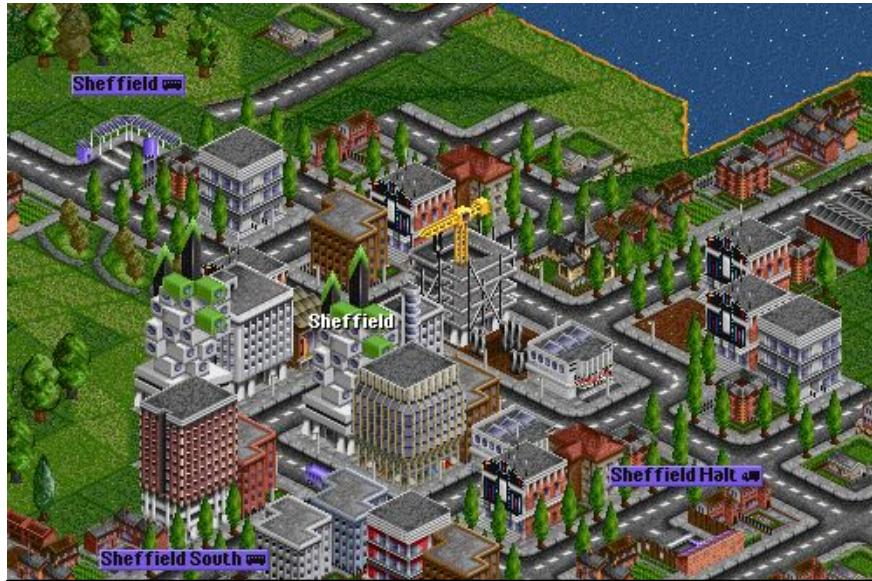
A nice write up on Debug information is here:

- <https://developers.redhat.com/articles/2022/01/10/gdb-developers-gnu-debugger-tutorial-part-2-all-about-debuginfo>
- <https://undo.io/resources/guide-to-symbols-debug-info/>

Note: If you’re in a live system, you can ‘gdb -ex ‘attach \$PID’ app.debug’ add debug info as well -- but let’s assume we have our debugging symbols in order!

Pro Tip: Debug Symbols (2/2)

- **Note:** A ****key**** that will make your life easier
 - **Build/Compile with debug information!**
 - `cmake -DCMAKE_BUILD_TYPE=Debug ..`
 - In some instances, this may also include making sure to build 3rd party libraries in debug as well
- **Debug information** is useful for helping you inspect and gather more information about your programs execution with the tools we will use

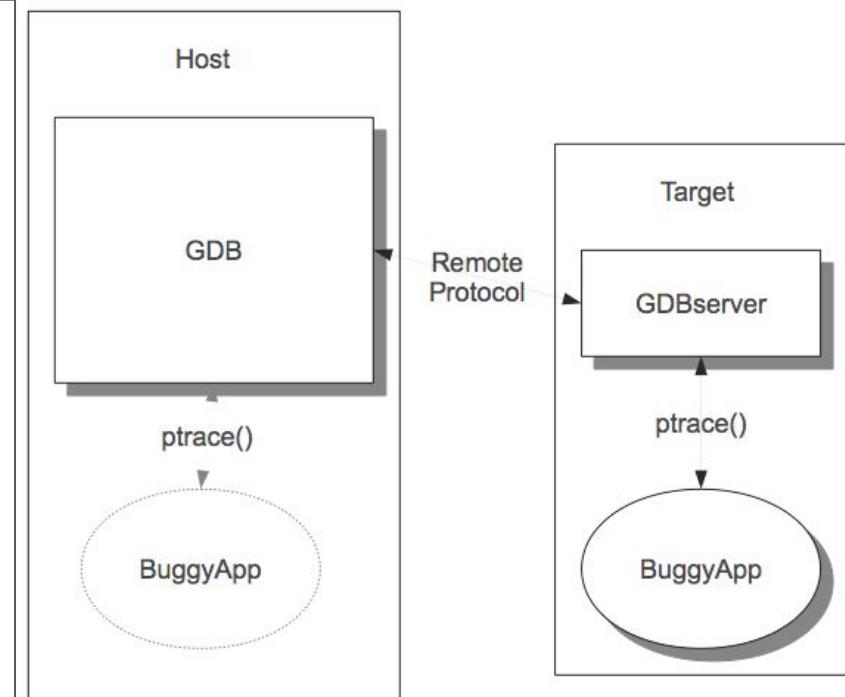


A nice write up on Debug information is here:
<https://developers.redhat.com/articles/2022/01/10/gdb-developers-gnu-debugger-tutorial-part-2-all-about-debuginfo>

Note: If you're in a live system, you can 'gdb -ex 'attach \$PID' app.debug' add debug info as well -- but let's assume we have our debugging symbols in order!

Step-by-Step debugging (1/3)

- So now we have a ‘large unfamiliar codebase’ that we have compiled in ‘debug’
- So we can use a ‘debugger’ to step through the execution
 - **And of course** use a ‘debug build’ of the source.
- **Let’s repeat our first task of finding `main()` that we did with ‘grep’, but this time with a debugger**



You can learn more about how debuggers work here: <https://aosabook.org/en/v2/gdb.html>

Step-by-Step debugging (2/3)

- [GDB](#) is a good free debugger to start with
- In this video, I attach a debugger to a running process
 - ps aux | grep 'openttd'
 - This gives me the name
 - Then I attach to the running process
 - sudo gdb -p pid
- **next slide for more!**

The screenshot shows a terminal window with two main sections. The top section is titled "mike@mike-MS-7B17:~/Downloads/openttd/openttd-14.1-source/openttd-14.1\$". It contains the command "ps aux | grep 'openttd'" which is run to find the process ID of openttd. The bottom section is titled "mike@mike-MS-7B17:build\$./openttd". It shows the openttd executable being run. The terminal window has a dark background with light-colored text. The bottom status bar shows the user's name, the date and time ("17:07 09-Jun-25"), and the command prompt "[0] 0:bash*".

Step-by-Step debugging (3/3)

- Once you have the ‘`main()`’ function, you can start stepping through the code.
- Some important commands in GDB
 - `Ctrl+C` // Pause running program
 - `br main()` // breakpoint in main
 - `c` // continue
 - `s` // to step into a function
 - `n` // next line
 - `f` // finish the stack frame
 - Useful if you’re in library code you do not own
 - `bt` // print out the backtrace
 - `kill` // to exit the process, but stay in GDB

GDB cheatsheet - page 1	
Running <code># gdb <program> [core dump]</code> Start GDB (with optional core dump).	<where> <code>function_name</code> Break/watch the named function.
<code># gdb --args <program> <args...></code> Start GDB and pass arguments	<code>line_number</code> Break/watch the line number in the current source file.
<code># gdb --pid <pid></code> Start GDB and attach to process.	<code>file:line_number</code> Break/watch the line number in the named source file.
<code>set args <args...></code> Set arguments to pass to program to be debugged.	
<code>run</code> Run the program to be debugged.	Conditions <code>break/watch <where> if <condition></code> Break/watch at the given location if the condition is met.
<code>kill</code> Kill the running program.	<code>Conditions</code> Conditions may be almost any C expression that evaluate to true or false.
	<code>condition <breakpoint#> <condition></code> Set/change the condition of an existing break/ or breakpoint.
Breakpoints <code>break <where></code> Set a new breakpoint.	Examining the stack <code>backtrace</code> Show call stack.
<code>delete <breakpoint#></code> Remove a breakpoint.	<code>where</code> Show call stack.
<code>clear</code> Delete all breakpoints.	<code>backtrace full</code> Show call stack, also print the local variables in each frame.
<code>enable <breakpoint#></code> Enable a disabled breakpoint.	<code>where full</code> Show call stack, also print the local variables in each frame.
<code>disable <breakpoint#></code> Disable a breakpoint.	<code>frame <frame#></code> Select the stack frame to operate on.
	Stepping <code>step</code> Go to next instruction (source line), diving into function.
Watchpoints <code>watch <where></code> Set a new watchpoint.	
<code>delete/enable/disable <watchpoint#></code> Like breakpoints.	

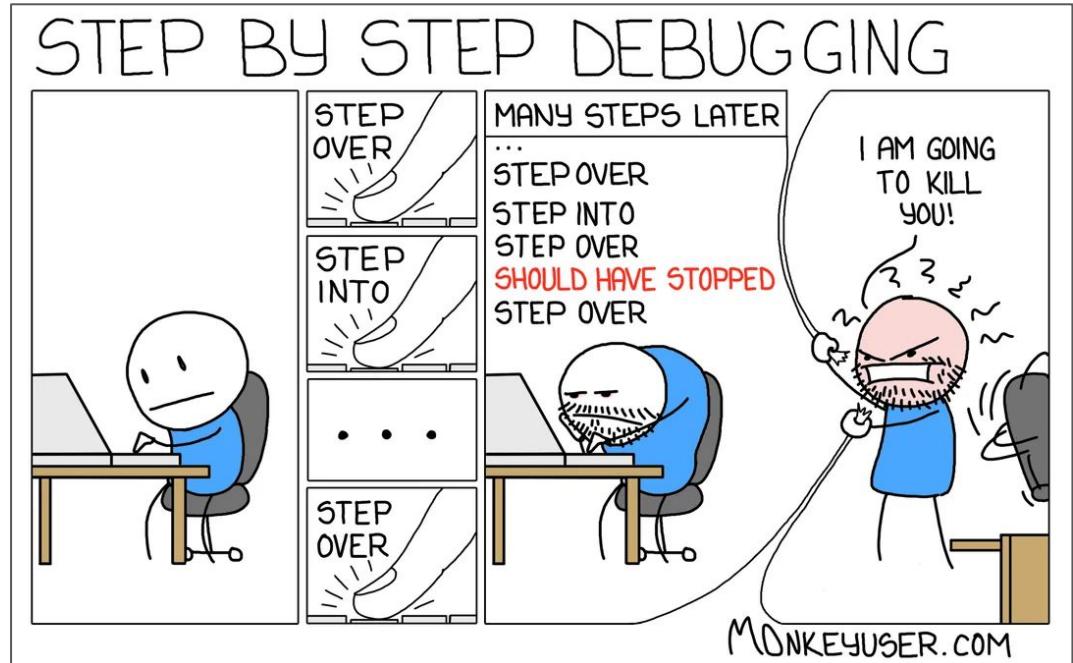
GDB Cheat sheet (page 1 of 2)

<https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf>

Step-by-Step: Backwards (i.e. Time Travel)

Reverse debugging (1/5)

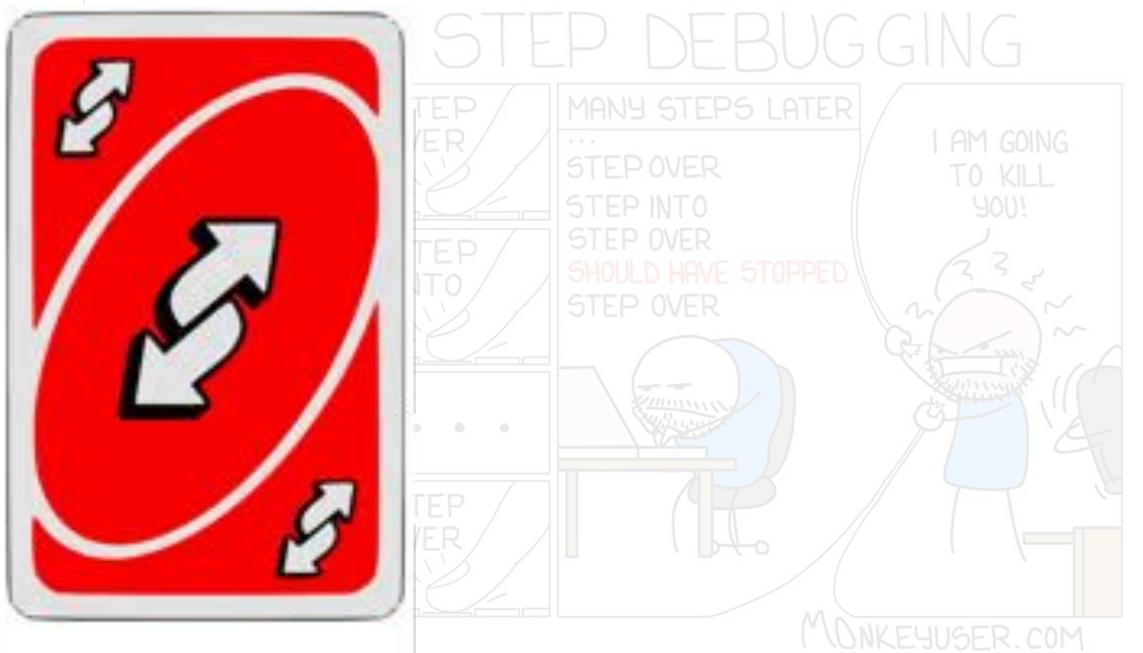
- Just a brief aside on debuggers
- At some point you will probably have this experience.
 - So it's worth mentioning 'reverse-debuggers' or 'time travel debuggers' to you :)



<https://preview.redd.it/e4b0jejmlxxz.png?width=1080&crop=smart&auto=webp&s=5b03b0d54dbe429bace5820da37a2345682eaada>

Reverse debugging (2/5)

- Just a brief aside on debuggers
- At some point you will probably have this experience.
 - So it's worth mentioning 'reverse-debuggers' or 'time travel debuggers' you :)



<https://preview.redd.it/e4b0jejmxxz.png?width=1080&crop=smart&auto=webp&s=5b03b0d54dbe429bace5820da37a2345682eaada>

Reverse debugging (3/5)

- Reverse debuggers (since GDB 7.0) allow you to ‘reverse-step’ or ‘reverse-next’ where you are.
- This is very useful during your code exploration!

The screenshot shows a Wikipedia-style page for "ReverseDebug". The header includes "GDB Wiki" and "Login". Below the header, the page title is "Self: ReverseDebug". The navigation bar contains links for "HomePage", "RecentChanges", "FindPage", "HelpContents", and "ReverseDebug" (which is highlighted). A search bar and a "Titles" link are also present. The main content area has a blue header with the title "Reverse Debugging with GDB".

Reverse Debugging with GDB

Beginning with the 7.0 release in September 2009, gdb now includes support for a whole new way of debugging called "reverse debugging" -- meaning that gdb can allow you to "step" or "continue" your program backward in "time", reverting it to an earlier execution state.

Reverse debugging is only supported for a limited (but growing) number of gdb targets, including:

- Certain remote targets including the Simics and SID simulators, and "Undo-db"
- The [Process Record and Replay](#) target for native linux.

Reverse debugging (4/5)

QR code to UDB trial

- UDB from [Undo](#) is another invaluable tool I'll demonstrate shortly.
- UDB effectively uses the same GDB commands -- so if you know one, you know both!
- More information
 - Performance Differences between UDB and GDB [[link](#)]
 - If you can't [use a reversible debugger?](#), then click this link for the GDB equivalents to try



UDB
The interactive time travel debugger
for Linux C/C++ for debugging live processes

UDB Demo Video

```
69 }  
70 /* Check cache_calculate() with a random number. */  
71 int number = (int)(0.5 * rand() / (RAND_MAX + 1.0));  
72 int sqroot_cache = cache_calculate(number);  
73 int sqroot_correct = (int)sqrt(number);  
74  
> 75 assert(sqroot_cache == sqroot_correct);  
76 }  
77 return NULL;  
78 }  
79  
80 int  
81 main(void)  
82 {  
83     unsigned nthreads = 10;
```

extended-r Thread 65644.656651 In: thread_fn
99% 262,536> info locals
number = 255
sqroot_cache = 0
sqroot_correct = 15
i = 199
_PRETTY_FUNCTION_ = "thread_fn"
99% 262,536> |

A screenshot of the UDB debugger interface. At the top, there's a title bar with the name 'UDB' and a subtitle 'The interactive time travel debugger for Linux C/C++ for debugging live processes'. Below that is a 'UDB Demo Video' section showing a code editor with some C code and assembly instructions. The code includes assertions for a square root calculation. At the bottom, there's a terminal window showing the interaction with the debugger, including commands like 'extended-r', 'info locals', and assembly dump commands. A small circular video player icon is visible in the bottom right corner of the demo video area.

Reverse debugging (5/5)

QR code to UDB trial

- I'll give you a minute to scan the QR code, or otherwise you may consider visiting this link
 - <https://undo.io/edb-free-trial/>
- Okay -- so let's see how reverse debugging can help us



UDB
The interactive time travel debugger
for Linux C/C++ for debugging live processes

UDB Demo Video

```
69 } /* Check cache_calculate() with a random number. */
70 int number = (int)(256.0 * rand() / (RAND_MAX + 1.0));
71 int sqroot_cache = cache_calculate(number);
72 int sqroot_correct = (int)sqrt(number);
73
74 > assert(sqroot_cache == sqroot_correct);
75 }
76 return NULL;
77 }
78
79 int
80 main(void)
81 {
82     unsigned nthreads = 10;
83 }
```

extended-r Thread 65644.656651 In: thread_fn
99% 262,536> info locals
number = 255
sqroot_cache = 0
sqroot_correct = 15
i = 199
_PRETTY_FUNCTION_ = "thread_fn"
99% 262,536>

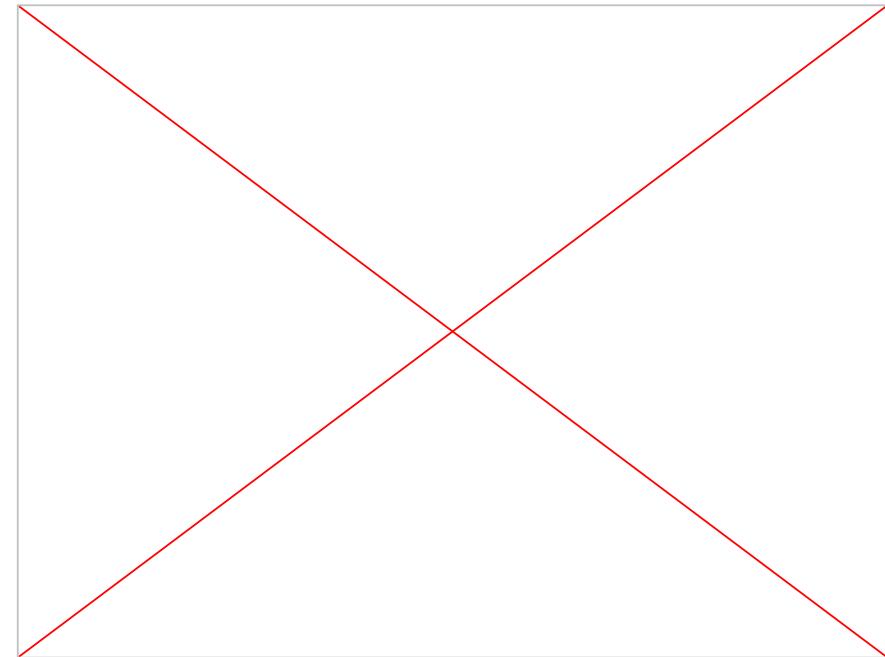
A screenshot of the UDB debugger interface. It shows a code editor with a snippet of C code for testing square root calculations. Below the code is a terminal window displaying the output of the debugger commands. A small video camera icon in the bottom right corner indicates a demo video is being recorded.

Reverse debugging (UDB) (1/3)

- The UDB Suite of tools is quite powerful -- so I will show a sample of ‘recording’ and then traversing that recording
- **First- capture the recording**
 - `~/Downloads/Undo-Suite-x86-8.3.0/live-record --record-on symbol:_Z12PostMainLoopv ./build/openttd`
 - A few neat things
 - You can trigger the recording to start on a particular symbol
 - Tools like ‘nm’ can be helpful here to find such symbols
 - `nm build/openttd | grep MainLoop`
- **Then- replay the recording**
 - `~/Downloads/Undo-Suite-x86-8.3.0/ldb openttd-466718-2025-06-10T19-18-20.511.undo`

Reverse debugging (UDB) (2/3)

- Here's an example of loading up a recording and stepping forward and backward
- Pretty neat!



Reverse debugging (UDB) (3/3)

- One more demo worth showing -- this one uses **watchpoints**
- **So instead of hopping around with ‘next’** -- we can **pause** on data that changes
- You can find interesting data (e.g. a ‘train data structure that might get modified’) as shown in this example
 - *hint* ‘ptype’ can help here in GDB once you find an object and want to look at the struct.



Pro Tip: Keep a Journal

- Now that you have done a few investigations -- take some notes!
 - Making lists about what you learned or need to revisit will be useful!
- I recommend using whatever journaling tool is the lowest friction thing for you
 - e.g. physical notebook perhaps, but a ‘Google Doc’ or text file that is ‘greppable’ may be best
 - More on note taking systems (e.g. .plan files):
 - https://www.youtube.com/watch?v=S_W1fzrB-UEg

[idsoftware.com]

Login name: johnc In real life: John Carmack

Directory: /raid/nardo/johnc Shell: /bin/csh

Never logged in.

Plan:

This is my daily work ...

When I accomplish something, I write a * line that day.

Whenever a bug / missing feature is mentioned during the day and I don't fix it, I make a note of it. Some things get noted many times before they get fixed.

Occasionally I go back through the old notes and mark with a + the things I have since fixed.

--- John Carmack

= feb 18 =====

* page flip crap
* stretch console
* faster swimming speed
* damage direction protocol
* armor color flash
* gib death
* grenade tweaking
* brightened alias models
* nail gun lag
* dedicated server quit at game end
+ scoreboard
+ optional full size
+ view centering key
+ vid mode 15 crap
+ change ammo box on sbar
+ allow "restart" after a program error
+ respawn blood trail?
+ -1 ammo value on rockets
+ light up characters

How else can we find what's important?

- Sampling profilers can be another good tool to see ‘what is going on’
 - A profiler like `perf` will show you what may be interesting.
 - i.e. the ‘hot paths’ in a codebase

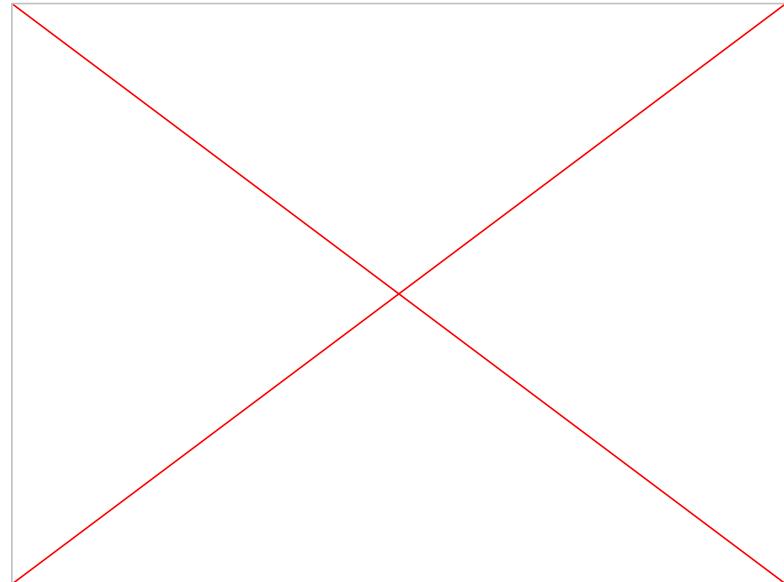
```
sudo perf record ./openttd  
sudo perf report
```

Samples: 11K of event 'cycles:P', Event count (approx.): 12994629772	Overhead	Command	Shared Object	Symbol
8.23%	openTTD	libgomp.so.1.0.0		[.] 0x000000000000256c0
6.61%	openTTD	libgomp.so.1.0.0		[.] 0x000000000000258a0
4.17%	openTTD	openTTD		[.] Sprite* Blitter_32bppOptimized::EncodeInternal<false>(std::array<unsigned char const*, unsigned int> const&, std::vector<BlitterMode> const&)
2.71%	openTTD	openTTD		[.] void Blitter_40bppAnim::Draws(BlitterMode)> (Blitter::Blitter_40bppAnim::Draws(BlitterMode)>::operator()<const BlitterMode> const)
2.34%	openTTD	openTTD		[.] unsigned char const* std::max_element<unsigned char const*>(std::vector<unsigned char const*> const)
2.02%	openTTD	openTTD		[.] ResizeSpriteIn<std::array<SpriteLoader::Sprite, 6ul>, ZoomLandscape> (openTTD::SpriteLoader::Sprite, 6ul)>::operator()<const TileIndexTag>
1.60%	openTTD	openTTD		[.] bool __gnu_cxx::ops::Iter_less_iter::operator()<unsigned int>(std::vector<unsigned int> const)
1.35%	ottd:game	openTTD		[.] Sprite* Blitter_32bppOptimized::EncodeInternal<false>(std::array<unsigned char const*, unsigned int> const)
1.33%	openTTD	openTTD		[.] Sprite* Blitter_32bppOptimized::EncodeInternal<true>(std::array<unsigned char const*, unsigned int> const)
0.99%	openTTD	openTTD		[.] std::initializer_list<unsigned char>::begin()
0.91%	openTTD	openTTD		[.] void Blitter_40bppAnim::Draws(BlitterMode)> (Blitter::Blitter_40bppAnim::Draws(BlitterMode)>::operator()<const BlitterMode> const)
0.88%	openTTD	openTTD		[.] unsigned char std::max<unsigned char>(std::vector<unsigned char> const)
0.84%	openTTD	openTTD		[.] GetFileSlope(StrongType::Typedef<unsigned int, TileIndexTag> const)
0.78%	openTTD	openTTD		[.] std::array<SpriteLoader::Sprite, 6ul>::operator()<const TileIndexTag>
0.70%	ottd:game	openTTD		[.] IsValidTile(Tile)
0.67%	ottd:game	openTTD		[.] unsigned char const* std::max_element<unsigned char const*>(std::vector<unsigned char const*> const)
0.65%	openTTD	openTTD		[.] std::initializer_list<unsigned char>::end()
0.57%	ottd:game	openTTD		[.] StrongType::Typedef<unsigned int, TileIndexTag, StrongType>::operator()<const RunFileLoop>
0.56%	ottd:game	openTTD		[.] RunFileLoop()
0.51%	openTTD	openTTD		[.] ViewportAddLandscape()
0.51%	openTTD	libc.so.6		[.] __strcmp_avx2
0.48%	ottd:game	openTTD		[.] bool __gnu_cxx::ops::Iter_less_iter::operator()<unsigned int>(std::vector<unsigned int> const)
0.47%	openTTD	openTTD		[.] ResizeSpriteOut<std::array<SpriteLoader::Sprite, 6ul>, ZoomLandscape> (openTTD::SpriteLoader::Sprite, 6ul)>::operator()<const TileIndexTag>
0.45%	ottd:game	openTTD		[.] CallVehicleTicks()
0.45%	ottd:game	openTTD		[.] Colour::Colour(unsigned int)
0.45%	ottd:game	openTTD	[Kernel\kallsyms]	[.] ResizeSpriteIn<std::array<SpriteLoader::Sprite, 6ul>, ZoomLandscape> (openTTD::SpriteLoader::Sprite, 6ul)>::operator()<const TileIndexTag>
0.44%	openTTD	openTTD	[Kernel\kallsyms]	[k] nv044975m
0.40%	ottd:game	openTTD		[.] TileLoop_Water(StrongType::Typedef<unsigned int, TileIndexTag> const)
0.35%	openTTD	libc.so.6		[.] memmove_avx unaligned_erm3
0.34%	openTTD	openTTD		[.] DecodeSingleSprite(SpriteLoader::Sprite*, SpriteFile&, unsigned int)
0.33%	openTTD	libc.so.6		[.] int malloc
0.33%	openTTD	openTTD		[.] AddSortableSpriteToDraw(unsigned int, unsigned int, int, int)
0.33%	openTTD	openTTD		[.] void GfxBlitter<4, false>(Sprite const*, int, int, BlitterMode)
0.31%	openTTD	openTTD		[.] AllocSprite(unsigned long)
0.31%	ottd:game	openTTD		[.] unsigned char std::max<unsigned char>(std::vector<char> const)
0.28%	openTTD	libc.so.6		[.] memset_avx2 unaligned_erm3
0.28%	ottd:game	openTTD		[.] GetFileSlope(StrongType::Typedef<unsigned int, TileIndexTag> const)
0.28%	ottd:game	openTTD		[.] AllocSprite(unsigned long)
0.28%	openTTD	openTTD		[.] ViewportSortParentSpritesSSE41(std::vector<ParentSpriteToDraw> const)
0.26%	openTTD	openTTD		[.] std::vector<char>::operator<char const*> (std::allocator<char const*> const)
0.25%	openTTD	libc.so.6		[.] malloc
0.25%	openTTD	openTTD		[.] StrongType::Typedef<unsigned int, TileIndexTag, StrongType>::operator()<const Blitter_40bppAnim::DrawRect(void*, int, int, unsigned int)>
0.24%	openTTD	openTTD		[.] Blitter_40bppAnim::DrawRect(void*, int, int, unsigned int)

Attaching to a Running Process

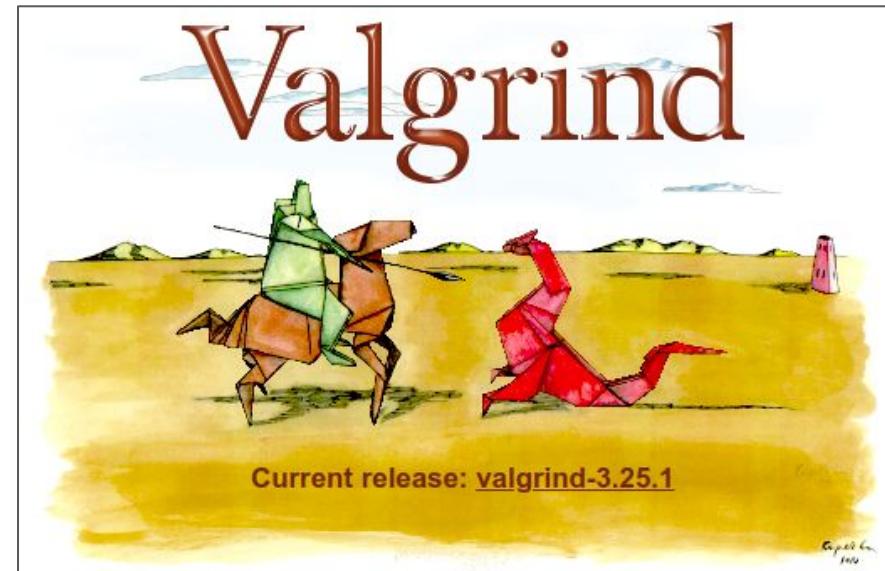
- Note: With most of the tools I'm showing you today (GDB, UDB, etc.) it's possible to 'attach to a running process'
 - This allows us to collect information from a specific point and time.

```
sudo perf record -p <pid> -g  
sudo perf report
```



Visualization: callgrind and kcachegrind (1/2)

- There's a good number of tools as well that can help you visualize the profile -
- The 'Valgrind' toolsuite (pronounced: val-grinn) is another dynamic analysis tool

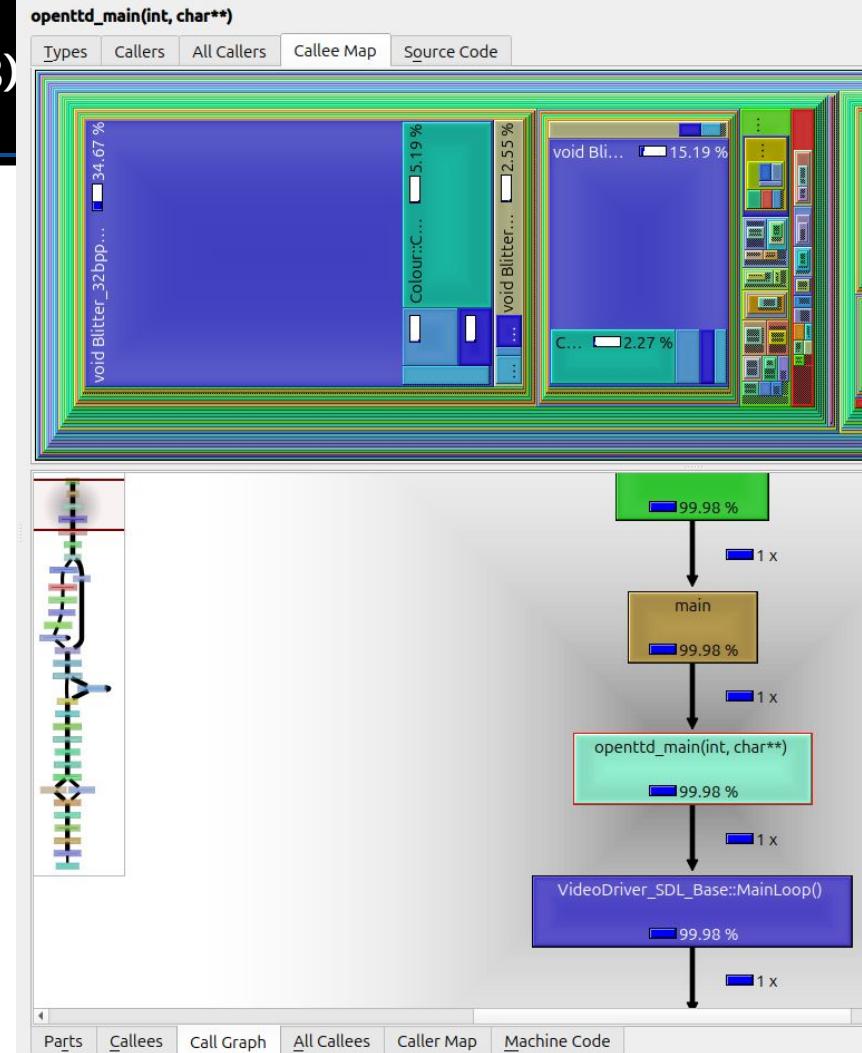


Visualization: callgrind and kcachegrind (2/2)

- The goal here is to again ‘sample’ where we execute code
 - (The call graphs will look pretty in your journal too)
- Run:
 - **valgrind --tool=callgrind -v --dump-every-bb=10000000 ./openttd**
 - This grabs a sample every 10000000 basic blocks executed
 - Note again: There are some mechanisms to turn on/off sampling if things are too slow
 - e.g. `callgrind_control --instr=on`
 - Documentation for usage:
 - <https://kcachegrind.sourceforge.net/html/Usage.html>

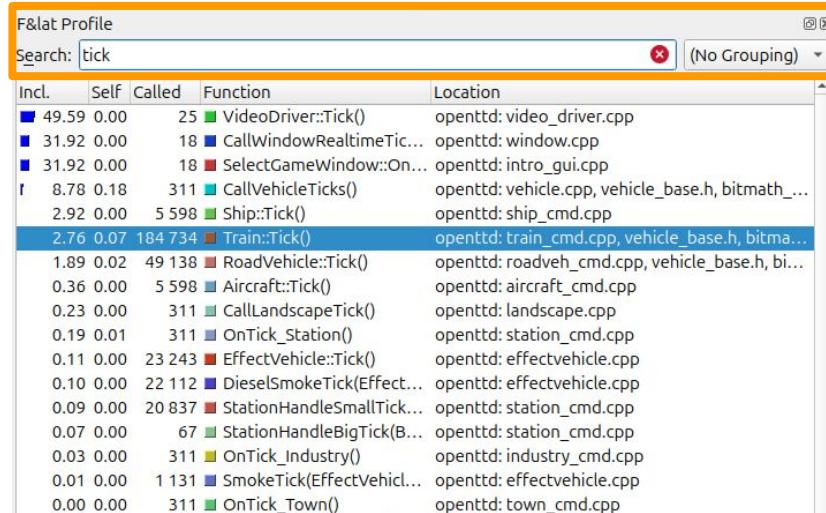
Visualization: callgrind and kcachegrind (1/3)

- Here's an example of the output from 'callgrind' visualized in 'kcachegrind'
- Simply run:
 - **kcachegrind**
 - (from the directory where the output of callgrind is dumped)



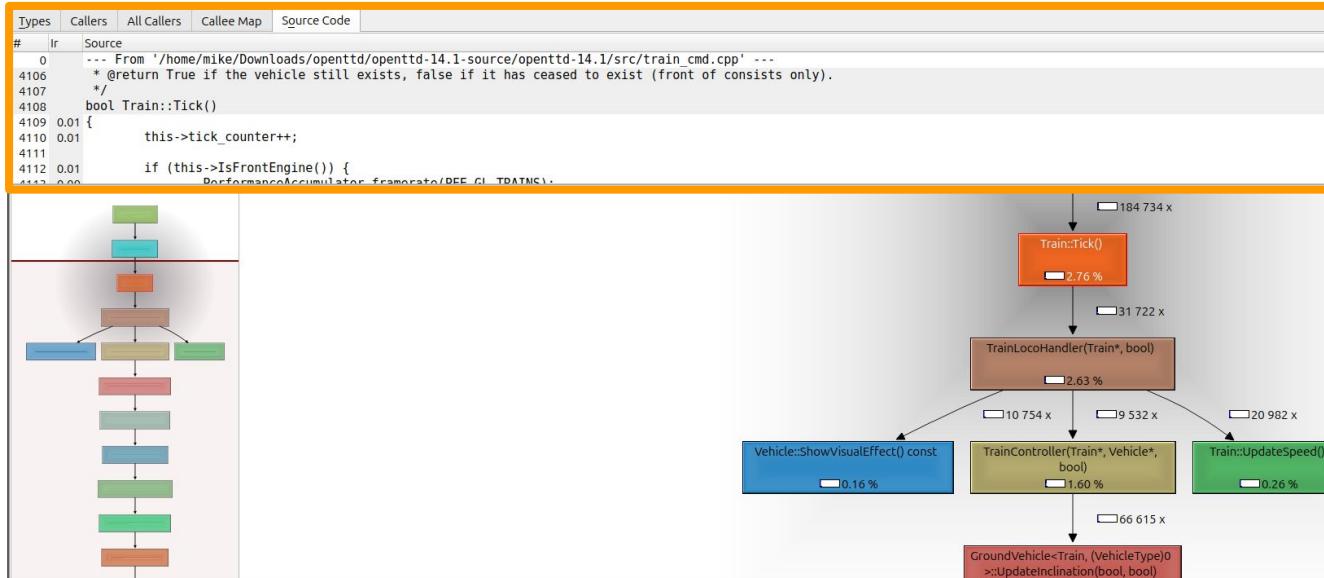
Visualization: callgrind and kcachegrind (2/3)

- Again, as you find things interesting, you can explore the call graph
- kcachegrind has a way to ‘search’ for functions
 - Using the ‘called’ column, you can find what may be important



Visualization: callgrind and kcachegrind (3/3)

- Note: You can directly view the source from this tool as well
- **However** -- I'd recommend doing a debug session with **udb or gdb** and setting a breakpoint at these points of interest



1. Understanding the control flow
2. The **pure scale** of software and its relation to complexity
3. Human factors can additionally contribute to making your life harder

C

Note: By **pure scale** that is an expression to mean: “The vast size” of the number of files and lines of code in a project. 52

Dealing with Pure Scale of Software

- Once you have some idea of the project flow, it may be time to ‘dive into’ the project into more specifics
- But since we cannot again look at all of the source, we’ll focus on **reducing scale** to the things you care about
 - i.e. You don’t necessarily have to understand about all of the codebase at once -- often just smaller subsystems

```
mike@mike-MS-7B17:openttd-14.1$ cloc .
 3138 text files.
 2402 unique files.
 1371 files ignored.

github.com/AlDanial/cloc v 1.98  T=1.86 s (1293.4 files/s, 534526.5 lines/s)
-----
```

Language	files	blank	comment	code
Text	86	53266	0	308781
C++	499	40695	47958	188428
C/C++ Header	768	23071	41636	123108
D	477	0	0	93912
make	59	6122	4985	16050
CMake	304	1380	1520	11791
INI	21	614	149	5023
JSON	5	1	0	3140
HTML	5	278	1	3106
Squirrel	33	364	238	3050
Markdown	26	966	32	3011
YAML	25	480	117	2819
Objective-C++	4	499	432	1742
SVG	2	1	2	1353
C	1	149	61	670
awk	1	41	32	217
XML	4	3	12	199
diff	2	3	31	174
Python	2	68	55	168
TypeScript	61	0	0	122
PowerShell	5	34	27	113
JavaScript	1	16	24	86
Tcl/Tk	1	22	32	52
Bourne Shell	3	17	32	41
reStructuredText	1	5	0	22
DOS Batch	5	4	0	18
Dockerfile	1	1	0	3
SUM:	2402	128100	97376	767199

```
mike@mike-MS-7B17:openttd-14.1$
```

Remember, 300,000+ lines of code -- wow

Source code uses ‘text’ as a visualization

- The **pure scale** (i.e. the size) of codebases is a challenge, and since we’ll be looking at a lot of text...
 - If at all possible beg / borrow / steal a large 4K+ screen, or two screens...
 - This is one of the few real hardware advantages these days for developers as build / test is often remote.
 - Dell's 27" 4K screen start £270 - £300 (inc. 20% VAT)



https://i.pcmag.com/imagery/roundups/01Y9baNdRmGOzHcetHQG2FW-36.fit_lim.size_1050x.webp

Windows/Screen Management

- Another less technical (but important) suggestion
- Find a good window manager
 - Either for your operating system, or a multiplexer (e.g. [tmux](#)) if you're working in terminal.
- The number of times I've needed to compare code side-by-side can be very helpful to see 'what have I missed'

The screenshot shows a terminal window with two code files open in split panes. The left pane contains the code for `multitexturematerial.d`, which defines a material that supports multiple textures. The right pane contains the code for `normalmapmaterial.d`, which defines a material that uses a normal map. Below the code panes, the terminal displays OpenGL setup information and a log of OpenGL commands being executed.

```
// A material that allows for multiple textures
module multitexturematerial;
import pipeline, materials, texture;
import bindobj.openGL;

// Represents a material with multiple textures
class MultiTextureMaterial : IMaterial{
    Texture mTexture;
    Texture mTexture2;
    Texture mTexture3;
    Texture mTexture4;
    Texture mTexture5;
    Texture mTexture6;
    Texture mTexture7;
    Texture mTexture8;
}

// Construct a new material for a pipeline, and load a texture for that
pipeline
this(string pipelineName,
    string textureFile1Name,
    string textureFile2Name,
    string textureFile3Name,
    string textureFile4Name,
    string textureFile5Name,
    string textureFile6Name,
    string textureFile7Name,
    string textureFile8Name)
{
    // delegate to the base constructor to do initialization
    super(pipelineName);
    mTexture = new Texture(textureFile1Name);
    mTexture2 = new Texture(textureFile2Name);
    mTexture3 = new Texture(textureFile3Name);
    mTexture4 = new Texture(textureFile4Name);
    mTexture5 = new Texture(textureFile5Name);
    mTexture6 = new Texture(textureFile6Name);
    mTexture7 = new Texture(textureFile7Name);
    mTexture8 = new Texture(textureFile8Name);
}

// TextureMaterial.update()
override void update()
{
    // Set uniforms for our mesh
    Pipeline* pipeline = Pipeline::getPipeline();
    Pipeline* pipelineUpdate = Pipeline::getPipeline("update");
    Pipeline* pipelineNormalMap = Pipeline::getPipeline("normalmap");

    // Set any uniforms for our mesh if they exist in the shader
    if("mTexture" in uniformMap){
        glUniform1i(uniformMap["mTexture"], 0);
        glBindTexture(GL_TEXTURE_2D, mTexture.textureID);
        uniformMap["mTexture"].Set(0);
    }
    if("mTexture2" in uniformMap){
        glUniform1i(uniformMap["mTexture2"], 1);
        glBindTexture(GL_TEXTURE_2D, mTexture2.textureID);
        uniformMap["mTexture2"].Set(1);
    }
    if("mTexture3" in uniformMap){
        glUniform1i(uniformMap["mTexture3"], 2);
        glBindTexture(GL_TEXTURE_2D, mTexture3.textureID);
        uniformMap["mTexture3"].Set(2);
    }
    if("mTexture4" in uniformMap){
        glUniform1i(uniformMap["mTexture4"], 3);
        glBindTexture(GL_TEXTURE_2D, mTexture4.textureID);
        uniformMap["mTexture4"].Set(3);
    }
}

// An example of a normal map material
module normalmapmaterial;
import pipeline, materials, texture;
import bindobj.openGL;

// Represents a normal mapped material
// Normal maps are often called 'diffuse map' for naming: https://www.a23d.co/p/difference-between-albedo-and-diffuse-map
class NormalMapMaterial : IMaterial{
    Texture mTexture;
    Texture mTexture2;
}

// Construct a new material for a pipeline, and load a texture for that
pipeline
this(string pipelineName, string textureFileName, string normalMapFileName)
{
    // delegate to the base constructor to do initialization
    super(pipelineName);
    mTexture = new Texture(textureFileName);
    mTexture2 = new Texture(normalMapFileName);
}

// TextureMaterial.update()
override void update()
{
    // Set uniforms for our mesh
    Pipeline* pipeline = Pipeline::getPipeline();
    Pipeline* pipelineNormalMap = Pipeline::getPipeline("normalmap");

    // Set any uniforms for our mesh if they exist in the shader
    if("albedomap" in uniformMap){
        glUniform1i(uniformMap["albedomap"], 0);
        glBindTexture(GL_TEXTURE_2D, mTexture.textureID);
        uniformMap["albedomap"].Set(0);
    }
    if("normalmap" in uniformMap){
        glUniform1i(uniformMap["normalmap"], 1);
        glBindTexture(GL_TEXTURE_2D, mTexture2.textureID);
        uniformMap["normalmap"].Set(1);
    }
}

// Set any uniforms for our mesh if they exist in the shader
if("mTexture" in uniformMap){
    glUniform1i(uniformMap["mTexture"], 0);
    glBindTexture(GL_TEXTURE_2D, mTexture.textureID);
    uniformMap["mTexture"].Set(0);
}
if("mTexture2" in uniformMap){
    glUniform1i(uniformMap["mTexture2"], 1);
    glBindTexture(GL_TEXTURE_2D, mTexture2.textureID);
    uniformMap["mTexture2"].Set(1);
}
if("mTexture3" in uniformMap){
    glUniform1i(uniformMap["mTexture3"], 2);
    glBindTexture(GL_TEXTURE_2D, mTexture3.textureID);
    uniformMap["mTexture3"].Set(2);
}
if("mTexture4" in uniformMap){
    glUniform1i(uniformMap["mTexture4"], 3);
    glBindTexture(GL_TEXTURE_2D, mTexture4.textureID);
    uniformMap["mTexture4"].Set(3);
}
```

SDL_Setup SDL_Setup SDL_Setup

Searching for SDL on Linux
SDL version(2, 30, 0)
Your SDL version loaded was: 2.30.0
Note: If SDL 2 was loaded, it's very likely that some SDL3 function calls, but some functions different.

OpenGL_Setup OpenGL_Setup OpenGL_Setup

OpenGL_SupportedVersion(4.1)

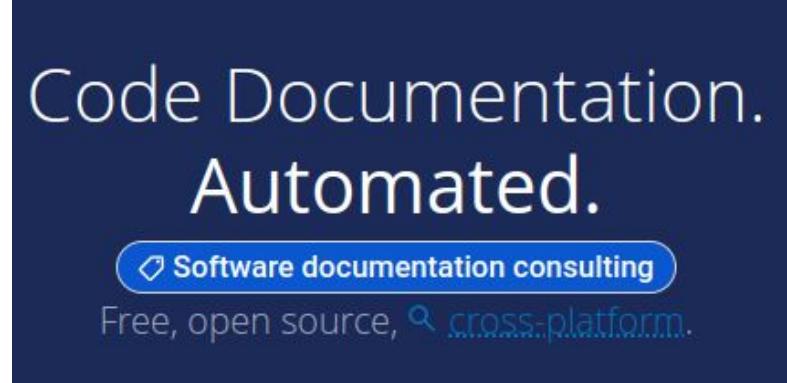
Vendor: NVIDIA Corporation
Renderer: NVIDIA TITAN Xp/C10/SSE2
Version: 4.1.0 NVIDIA 365.77
Shading Language: 4.10 NVIDIA via Cg Compiler

GL_LINK_STATUS=0
GL_ATTACHED_SHADERS=0
GL_ACTIVE_ATTRIBUTES=2
GL_ACTIVE_UNIFORMS=5
GL_ACTIVE_UNIFORM_BLOCK=1
GL_ACTIVE_TEXTURES=2
GL_ACTIVE_VARYINGS=0
GL_ACTIVE_VERTICES=0
GL_CURRENT_PROGRAM=0
GL_NUM_EXTENSIONS=1
Uniform automatically parsed from: /pipelines/normalmap/vert.glsL
("uniform mat4 mModel"; "uniform mat4 mView"; "uniform mat4 mProjection";)
Uniform automatically parsed from: /pipelines/normalmap/frag.glsL
("uniform sampler2D albedomap"; "uniform sampler2D normalmap";)
Dimensions12x512x1
Camera Position: <0,0,1>
Progressed to step 0 now exiting..
Camera Position: <0,0,1>
Shutting Down

Tools like tmux and VIM can split and manage all your code windows. Find some tools you are otherwise comfortable with

Generate your Documentation (1/3)

- By now hopefully you've found a few useful functions
 - Perhaps using **ldb**, **kcacheGrind**, etc.
- But how to keep notes of what was important?
- No docs in your project?
- No problem -- generate them yourself
 - Doxygen, Doxypress, or other documentation generators can be helpful here



<https://www.doxygen.nl/>

Generate your Documentation (2/3)

- *Example Video* in 1-minute
- For more configuration options check:
 - <https://www.youtube.com/watch?v=tLPHQMoF9M>

```
doxygen -g  
doxygen Doxyfile
```



A terminal window showing the execution of the doxygen command. The command 'doxygen -g' is run first, followed by 'doxygen Doxyfile'. The terminal shows a black background with white text. The prompt is 'mike@mike-MS-7B17:~/Downloads/openTDD/openTDD-14.1\$'. The command 'doxygen Doxyfile' is shown in blue. The output of the command is completely blacked out. The bottom of the terminal shows the prompt '[0] 0:bash*' and the date/time 'mike@mike-MS-7B17: ~/" 18:09 10-Jun-25'.

Generate your Documentation (3/3)

- Some more examples of what is generated
 - left: documentation extracted from functions and members of types
 - right: inheritance diagrams

Classes

```
struct AcceptedCargo
struct ProducedCargo
struct ProducedHistory
```

Public Types

```
using ProducedCargoArray = std::array< ProducedCargo, INDUSTRY_NUM_OUTPUTS >
using AcceptedCargoArray = std::array< AcceptedCargo, INDUSTRY_NUM_INPUTS >
```

» Public Types inherited from Pool< TItem, Tindex, Tgrowth_step, Tmax_size, Tpool_type, Tcache, Tzero >::PoolItem<& _industry_pool >

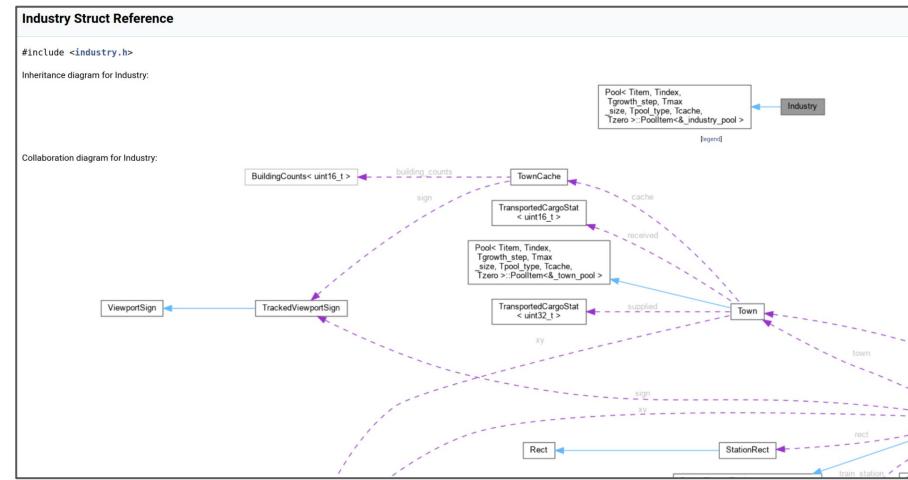
Public Member Functions

```
    Industry (TileIndex tile=INVALID_TILE)
    void RecomputeProductionMultipliers ()
    bool TileBelongsToIndustry (TileIndex tile) const
    ProducedCargoArray::iterator GetCargoProduced (CargoID cargo)
    ProducedCargoArray::const_iterator GetCargoProduced (CargoID cargo) const
    AcceptedCargoArray::iterator GetCargoAccepted (CargoID cargo)
    AcceptedCargoArray::const_iterator GetCargoAccepted (CargoID cargo) const
    bool IsCargoAccepted () const
    bool IsCargoProduced () const
    bool IsCargoAccepted (CargoID cargo) const
    bool IsCargoProduced (CargoID cargo) const
    const std::string & GetCachedName () const
```

» Public Member Functions inherited from Pool< TItem, Tindex, Tgrowth_step, Tmax_size, Tpool_type, Tcache, Tzero >::PoolItem<& _industry_pool >

Static Public Member Functions

```
static Industry * GetByTile (TileIndex tile)
```



Recap and Some Observations

- No documentation -- no problem
 - Generate your own using Doxygen, Doxypress, or other tools to extract information out.
 - Can often use your compiler to see the dependencies at the least
- Viewing the ‘doxygen’ files (or any documentation) gives you hints as a developer to a few things
 - How are things named?
 - i.e. the naming convention
 - How are ‘errors’ propagated (return codes or exceptions?)
- Take a few notes of this -- it will help you when it does become time to submit for code review!

Public Attributes

uint32_t	name_2
	Parameter of name_1 .
StringID	name_1
	Name of the company if the user did not change it.
std::string	name
	Name of the company if the user changed it.
StringID	president_name_1
	Name of the president if the user did not change it.
uint32_t	president_name_2
	Parameter of president_name_1 .
std::string	president_name
	Name of the president if the user changed it.
CompanyManagerFace	face
	Face description of the president.
Money	money
	Money owned by the company.
byte	money_fraction
	Fraction of money of the company, too small to represent in money .
Money	current_loan
	Amount of money borrowed from the bank.
Money	max_loan
	Max allowed amount of the loan or COMPANY_MAX_LOAN_DEFAULT.

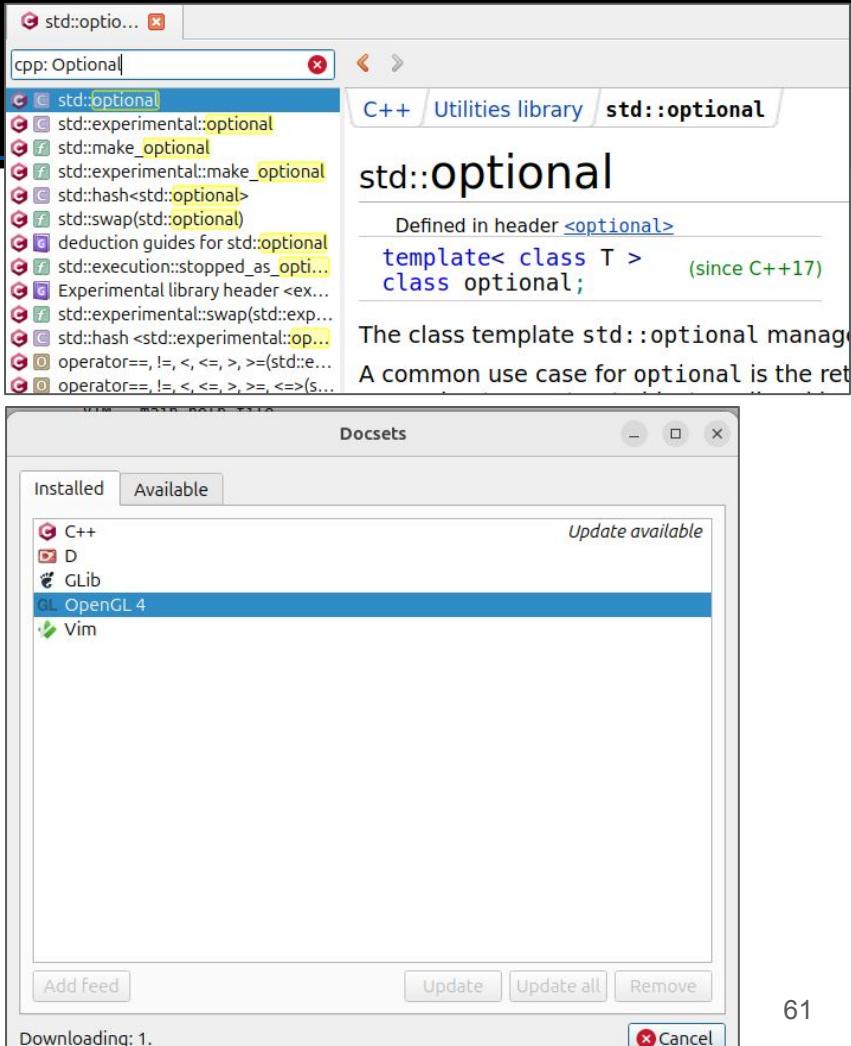
Code you do not own

- In any big project, there will also be code that you **do not own**
- For example, shared libraries
- Finding shared libraries is easy
 - Just run a tool like ldd ([Dependency Walker](#) on Windows is also quite nice)
- You'll often get some hints of what 'graphics libraries' to otherwise pay attention to.

```
mike@mike-MS-7B17:build$ ldd openttd
    linux-vdso.so.1 (0x00007ffcc2dec000)
    libpng16.so.16 => /lib/x86_64-linux-gnu/libpng16.so.16 (0x0000705b9a3b1000)
    libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x0000705b9a395000)
    liblzma.so.5 => /lib/x86_64-linux-gnu/liblzma.so.5 (0x0000705b9a363000)
    libfluidsynth.so.3 => /lib/x86_64-linux-gnu/libfluidsynth.so.3 (0x0000705b9a27c000)
    libSDL2-2.0.so.0 => /lib/x86_64-linux-gnu/libSDL2-2.0.so.0 (0x0000705b97824000)
    libfreetype.so.6 => /lib/x86_64-linux-gnu/libfreetype.so.6 (0x0000705b97758000)
    libfontconfig.so.1 => /lib/x86_64-linux-gnu/libfontconfig.so.1 (0x0000705b97707000)
    libharfbuzz.so.0 => /lib/x86_64-linux-gnu/libharfbuzz.so.0 (0x0000705b975fa000)
    libicui18n.so.74 => /lib/x86_64-linux-gnu/libicui18n.so.74 (0x0000705b97200000)
    libicuuc.so.74 => /lib/x86_64-linux-gnu/libicuuc.so.74 (0x0000705b97100000)
    libicudata.so.74 => /lib/x86_64-linux-gnu/libicudata.so.74 (0x0000705b97000000)
```

Zeal Docs (or Dash)

- For these function calls, I recommend finding fast offline documentation
 - no latency, and usually easier to organize and keep tabs
 - Sometimes more powerful search options as well
 - e.g. fuzzy search
- Note:
 - These tools in combination with code you own can also be useful



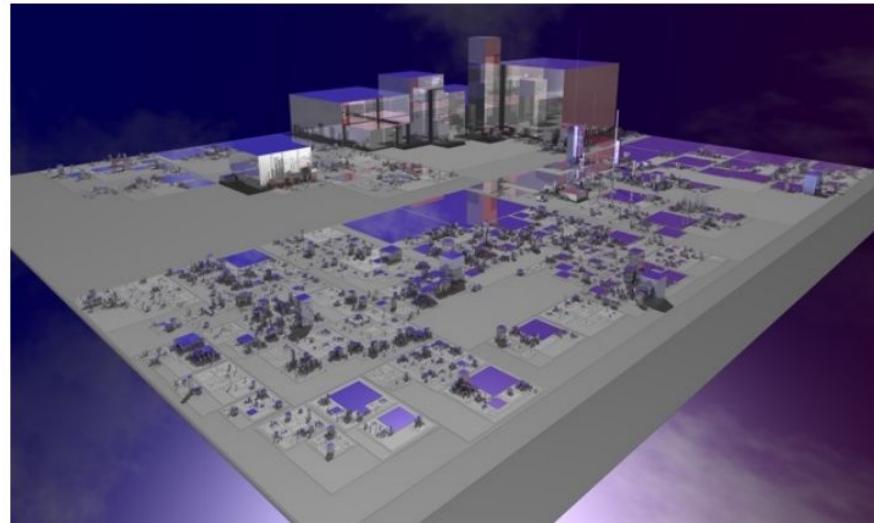
Still too much? Try Running Examples/tests

- Try to reducing the scale of the project further
- If you're working in a framework, then running some of the smaller samples
 - e.g. a graphics library likely has tutorials out there
- If you're lucky, there may also be some ‘tests’ that mock or showcase behavior.

```
mike@mike-MS-7B17:openttd-14.1$ tree src/tests/  
src/tests/  
└── bitmath_func.cpp  
    ├── CMakeLists.txt  
    ├── landscape_partial_pixel_z.cpp  
    ├── math_func.cpp  
    ├── mock_environment.h  
    ├── mock_fontcache.h  
    ├── mock_spritecache.cpp  
    ├── mock_spritecache.h  
    ├── string_func.cpp  
    ├── strings_func.cpp  
    └── test_main.cpp  
        └── test_script_admin.cpp  
            └── test_window_desc.cpp
```

Some Other Tips (1/3)

- Some of you have the fortune of working on nicely formatted codebases
 - If your codebase is ‘ugly’ -- consider running [clang-tidy](#), [indent](#), or similar formatting tools to reformat and unify the code.
- There’s a world coming soon where we’ll need different abstractions to visualize our code -- perhaps you’ll come up with other neat abstractions (e.g. “software cities”)
 - <https://ieeexplore.ieee.org/document/4290706>
 - <https://www.cs.nmt.edu/~jeffery/city-surv.pdf>



A ‘software city’ is a metaphor for the source code. Blocks may represent ‘namespaces’ or ‘directories’ and buildings ‘files’ for instance.

Some Other Tips (2/3)

- Study Design Patterns
 - Yes, they're not perfect, but traditional patterns like 'observer', 'visitor', etc. tend to show up.
 - These are found in the '[Design Patterns](#)' or '[Gang of Four](#)' book
 - Other sorts of software design things (e.g. 'event-loop', 'component-pattern', 'plugin-system') are also good to search for examples.
 - Often times writing a toy version of these patterns can help you understand the context in a larger system.
- In some cases, you can try to find an application that may be similar to what you're developing, and learn about how that system is documented.
 - There may be some hints within software case studies otherwise if your software has absolutely no documentation on its architecture by looking at related software.
 - See: <https://aosabook.org/en/>

Some Other Tips (3/3)

- Text editor and IDE Support continues to improve.
 - i.e. inference of types and other information often available as needed.

The screenshot shows a code editor displaying the following C++ code:

```
int main() {
    auto lang = "C++";
    std::cout << "Hello and welcome to " << lang << "!\n";
    std::string str;
    str.append("Hello friend");
```

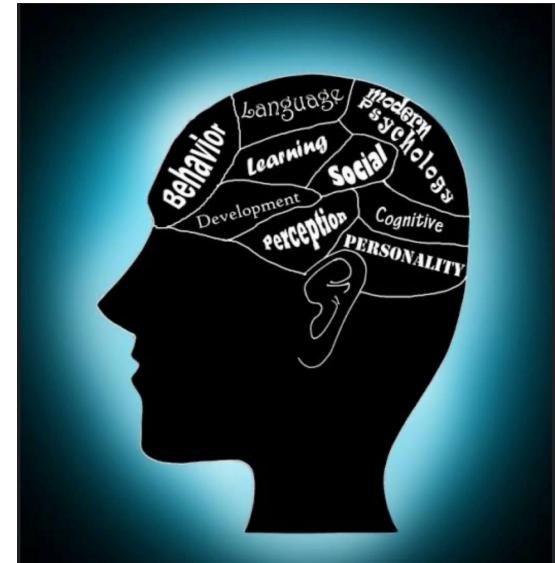
Two annotations are present on the right side of the code:

- A tooltip for the variable `lang` indicates it is a `local variable` of type `const char *lang`.
- A tooltip for the variable `str` indicates it is a `local variable` of type `std::string str`.

1. Understanding the control flow
2. The pure scale of software and its relation to complexity
3. **Human factors** can additionally contribute to making your life harder

Human Factors

- We have some tools now to help us **understand the control flow** and **working at scale**.
- But sometimes the hardest problems in software are sometimes -- i.e. **human factors**
- What I mean by this is -- how do I get motivated or focused to actually learn a codebase?
 - **My best answer** is to put a real problem in front of you that you will learn from!



https://miro.medium.com/v2/resize-fit:1400/1*nOZxRBAQwq8FZ-lINA630w@2x.jpeg

Find a first good task (1/3)

- With a large codebase, we might get some direction by literally just solving a problem.
 - If you're lucky, your code base will be labeled with 'good first tasks'
 - Good first tasks are 'byte sized' -- i.e. small, and picked out by engineers on your team
- For our project today -- observe there is a **Development** tab on the website.
 - <https://www.openttd.org/development>

The screenshot shows the OpenTTD website's navigation bar at the top, featuring links for Home, About, Manual, Screenshots, Servers, Development (which is highlighted in orange), Forum, Community, Contact, and Donate. To the right of the navigation is a logo consisting of a green diamond shape with a white dollar sign inside, and the text "OPENTTD" next to it. Below the navigation, there are two main sections: "Translating" and "What to do When You Find a Bug". The "Translating" section contains instructions for joining the translation team on GitHub and using the web translator. The "What to do When You Find a Bug" section provides guidelines for reporting bugs, including instructions for running a recent version, reporting to the forum, making issues reproducible, checking bug trackers, and attaching crash logs.

Download stable (14.1)
Download testing (15.0-beta2)
Download nightly (20250610)

Home About Manual Screenshots Servers **Development** Forum Community Contact Donate

Translating

If you would like to help translating the game, please

- join our translation team on GitHub (you will need a GitHub account).
- After joining you get access to our Web Translator.

What to do When You Find a Bug

When you think you found a bug in OpenTTD you should:

- Make sure you are running a recent version, i.e. run the latest stable, testing or nightly based on where you found the bug.
- Make sure you are not running a non-official binary, like a patch pack. If you are playing with a patch pack you should report any bugs to the forum thread related to that patch pack.
- Make it reproducible for the developers. In other words, create a savegame in which you can reproduce the issue once loaded. For Windows users it is very useful to give us the crash.dmp and crash.log which are created on crashes.
- Check whether the bug is already reported on our bug tracker. This includes searching for recently closed bug reports as the bug might already be fixed.
- When you are certain it is an unknown bug you should make a bug report at our bug tracker. The bug report must contain a savegame, preferably crash.sav and the last (automatically) saved savegame. It must also contain crash.log and crash.png (if they were created). For crashes on Windows must attach the crash.dmp files too. For bugs in nightlies you must also specify the revision of the nightly.

Find a first good task (2/3)

- Searching for ‘bugs’ or ‘issues’ are a great place to start
- Often time these issues will be labeled

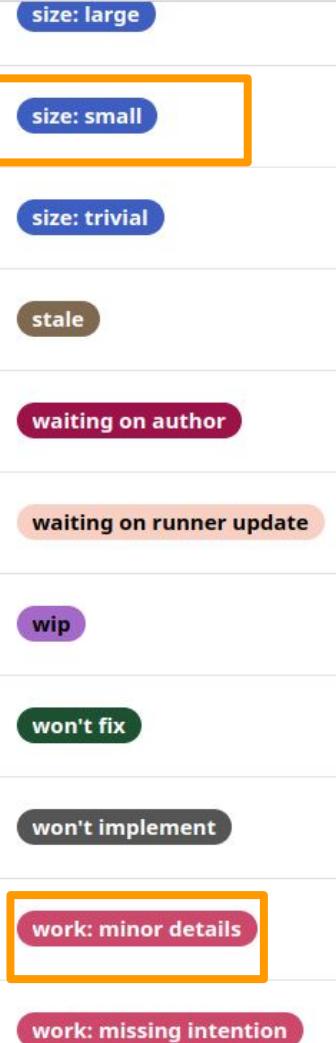
The screenshot shows the GitHub interface for the OpenTTD repository. The top navigation bar includes links for Code, Issues (188), Pull requests (135), Discussions, Actions, Security, and Insights. A search bar with placeholder text "Type to search" is located at the top right. Below the navigation, a modal window titled "Want to contribute to OpenTTD/OpenTTD?" provides instructions: "If you have a bug or an idea, read the [contributing guidelines](#) before opening an issue. If you're ready to tackle some open issues, [we've collected some good first issues for you](#)". This modal has a "Dismiss" button in the top right corner. The main content area shows a search bar with the query "is:issue state:open". To the right of the search bar are buttons for Labels (which is highlighted with an orange box), Milestones, New issue, and other filters like Author, Projects, Milestones, Assignees, Types, and Newest. Below the search bar, there are three issue cards:

- [Crash]: Video driver install triggers fatal blitter assertion "_screen.dst_ptr != nullptr" by GreenReaper (opened 6 hours ago) with 3 comments.
- [Bug]: German industry names should be [industry] [town name] by RiedleroD (opened 3 days ago) with 2 comments.
- [Bug]: Companies in multiplayer can be in the red for years before going bankrupt by bjornwarme (opened last week) with a "needs triage" label.

<https://github.com/OpenTTD/OpenTTD/issues>

Find a first good task (3/3)

- Of the labels for this particular project, I found two that are probably good ‘starting places’
 - i.e. they imply a small amount of work



If you are not assigned a good ‘first task’:

- Don’t forget about our tool ‘grep’
 - Try searching for ‘TODO’, ‘FIXME’ or ‘Later’ as ‘keywords’
 - If you’re lucky, you may find notes something like:
 - “This is ‘hard-coded’ and should be fixed later”
 - “FIXME, this is a ‘magic constant’ that should be loaded from a config file...”
 - etc.
- `grep -irn -I --include=*.cpp "TODO" ./src/`
 - TODO Show an example of how to query for small ‘git diff’ messages in the git log.
 - Ideally can do this in a ‘subsystem’ you’re working in or interested in.
 - Thegithub messages should give some hints about what is going on, or at the least the code change.

```
mike@mike-MS-7B17:openttd-14.1$ grep -irn -I --include=*.cpp "TODO" ./src/
./src/music/midifile.cpp:1069:    std::string tempdirname = FioGetDirectory(Searchpath::SP_AUTODOWNLOAD_DIR);
./src/strengr/strengr_base.cpp:31:int _errors, _warnings, show_todo;
                                         if ((show_todo & 2) != 0) StrengrWarning("{}");
                                         if ((show_todo & 1) != 0) {
                                         const char *s = "<TODO> ";
                                         if (_show_todo > 0 && ls->translated.empty()) {
                                         if ((show_todo & 2) != 0) {
                                         if ((show_todo & 1) != 0) {
                                         if (_show_todo > 0) {
                                         GETOPT_NOVAL('t', "--todo"),
                                         show_todo |= 1;
                                         show_todo |= 2;
                                         " -t | --todo" replace any untransl
                                         if ((show_todo & 2) != 0) {
                                         /* @todo Need to find which terminal (or hangar) we've c
                                         /* @todo Need to check terminal we're landing to. Is it
                                         .src/townname.cpp:192: * TODO: Perhaps we should use it for all the name generators? --pasky */
                                         .src/tunnelbridge cmd.cpp:11: * @todo separate this file into two
                                         .src/os/windows/win32.cpp:374:     _searchpaths[SP_AUTODOWNLOAD_PERSONAL_DIR] = tmp;
                                         case SP_AUTODOWNLOAD_DIR: // Otherwise we cannot download
                                         .src/fileio.cpp:862:     _searchpaths[SP_AUTODOWNLOAD_PERSONAL_DIR_XDG] = tmp;
                                         .src/fileio.cpp:872:     _searchpaths[SP_AUTODOWNLOAD_PERSONAL_DIR_XDG] = tmp;
                                         .src/fileio.cpp:875:     _searchpaths[SP_AUTODOWNLOAD_PERSONAL_DIR_XDG].clear();
                                         .src/fileio.cpp:891:     _searchpaths[SP_AUTODOWNLOAD_PERSONAL_DIR] = tmp;
                                         .src/fileio.cpp:894:     _searchpaths[SP_AUTODOWNLOAD_PERSONAL_DIR].clear();
                                         .src/fileio.cpp:1078: /* If we have network we make a directory for the autodownloading of content */
                                         .src/fileio.cpp:1079: _searchpaths[SP_AUTODOWNLOAD_DIR] = personal_dir + "content_download" PATHSEP;
                                         .src/fileio.cpp:1080: Debug(misc, 3, "{} added as search path", _searchpaths[SP_AUTODOWNLOAD_DIR]);
                                         .src/fileio.cpp:1081: Fio.CreateDirectory(_searchpaths[SP_AUTODOWNLOAD_DIR]);
                                         .src/fileio.cpp:1087: Fio.CreateDirectory(FioGetDirectory(SP_AUTODOWNLOAD_DIR, dirs[i]));
                                         /* TODO: Regarding this, when we do gradual loading, we
                                         .src/economy.cpp:1815:                                     /* TODO: currently this only works for AI companies
                                         .src/order_gui.cpp:1521:                                     * TODO: give a warning message */
                                         .src/order_gui.cpp:1807: * TODO: Rewrite the order GUI to not use different WindowDescs.
```

More ideas for ‘first’ tasks

- Another idea is to try fixing some 'warnings' in the code.
 - Not glorious, but at least can give you some practice.
- Probably more exciting is to try to ‘add’ something
 - This might be adding another item to a ‘listbox’ user interface widget
 - That will teach you if this data is hard-coded, comes from a configuration file, is downloaded, etc.
 - Again, it will probably give you an idea of how some system works.
- **Exercise:** Try using ldb or gdb to ‘track’ what function is called when you click on something.
 - ldb has live recorder for tracking changes, and otherwise we can use our grep skills to search the codebase

Human Factors: Asking for Help (on a team)

- Don't be arrogant, and be always be nice to your teammates.
 - Empathy matters **a lot** when working on teams, and it helps when you want to ask each other questions!
- "If you don't know, just ask"
 - If you're a junior engineer, then bring what you have tried/learned for the problem you have solved
 - If you're a senior engineer, make a note of where reoccurring questions may be coming from in the project.
 - Generally, don't put juniors in uncomfortable positions either -- sit down with them when they first join your team to help them get acquainted!
 - Screen record your session if appropriate, or let junior engineers take a picture of the whiteboard

1. **Understanding the control flow**
2. The pure scale of software is often one factor.
3. **Human factors** can additionally contribute to making your life harder

Bonus Round: **AI**

Does AI Solve this problem?

- I'm not sure -- yet (sorry!)
 - But if you have access to enterprise AI tools (i.e. where it's safe to paste some code in if you're working at a company) -- they may help summarize what code is doing.
- Some spaces to watch out for
 - Code/context summarizers (e.g. <https://sourcegraph.com/>)
 - Debugging assistants [chatDBG](#)
 - All of these tools will likely continue to improve, so keep an eye on this space!

Does AI Solve this problem?

- It's getting there, but there's a way to go
 - But if you have access to enterprise AI tools (i.e. where it's safe to paste some code in if you're working at a company) -- they can help summarize what code is doing, answer your questions about the codebase, help with basic tasks.
- Some spaces to watch out for
 - Code/context summarizers (e.g. <https://sourcegraph.com/>)
 - Debugging assistants [chatDBG](#)
 - [Claude Code](#) terminal based collaborative coder
 - Undo's AI integration coming soon (see the lightning talk by Rashmi *time TBC*)
- Things to watch out for
 - AIs get distracted if something looks like something it have seen before
 - They can be very convincing, especially when you don't know the codebase either
 - They rarely challenge you or defend their positions with evidence
 - "Trust but verify" is the safest approach today

Time Travel debugging & AI Vs A MySQL Issue (1/2)

- I recently started looking at an open issue with MySQL
- I've no knowledge of the codebase, so I had to get started somewhere
- Again, searching code and setting breakpoints is the way to start

The screenshot shows a debugger interface with several windows:

- Editor:** Displays MySQL source code for `item.cc`. The code includes a temporary table creation, insertions, and a complex SELECT query involving coalesce, convert, and if functions.
- Terminal:** Shows a MySQL session with the following logs:

```
This CDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) [answered N; input not from terminal]
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.

Thread 17 "ib log_files" received signal SIGSTOP, Stopped (signal).
[Switching to Thread 60646.60665]
[Switching to Thread 60646.60665]
Have reached end of recorded history.
0x00007f7505508d71 in _futex_abstimed_wait_common64 (private=32628, cancel=true, abstime=0x7f74b37fd8b0, op=157, expected=0, futex_word=0x7f74b37fd8b0) [/usr/lib/internal.c:57]
Have reached start of recorded history.
[Switching to Thread 60646.60665]
{void (int, const char *, ...)} 0x7f7505499a12 <err>
start >[]
```
- Bottom Status Bar:** Shows WSL:Ubuntu-24.04, 0 errors, 0 warnings, 0 info, 0 UDBs, Replay an Undo recording (mysql-8.0.3), Build, and MS SQL.

Time Travel debugging & AI Vs A MySQL Issue (2/2)

- AI isn't really the topic today
- AI (Claude and ChatGPT) was hallucinating at first and running off after any vaguely plausible root cause with intense conviction
- Time Travel debugging allowed me to constrain the AI by making it confirm with evidence every hypothesis

```
chris@hyperion:~/tmp/mysql-8.0.35$ cloc .
 35707 text files.
 12819 unique files.
 21827 files ignored.

github.com/AlDanial/cloc v 1.98 T=9.82 s (1305.6 files/s, 650515.5 lines/s)
-----
Language           files    blank   comment      code
-----
C++                  4467   437653   561364  2384569
C/C++ Header        5211   200309   511638  895020
C                     635    59973    75534   341537
Text                  186    10159      0     164174
JSON                   70      3      0     115239
Pascal                 260    17800   35508   103489
Bourne Shell          122    15026   17801   94354
Java                   534    12915   21441   55213
m4                      20    3950     962   36727
CMake                  451    6030    14147   36434
SQL                     203    2307    6319   26634
Perl                   122    7794    5270   25205
----- Snip -----
SUM:                12819   781666  1261752  4343511
```

Article documenting my experience analysing this issue:

<https://undo.io/resources/time-travel-debugging-tames-complex-codebases-and-overconfident-ai/>

More Resources

Talks on Tools / Debuggers (Linux Focus)

- Debugging and Tools
 - Time Travel Debugging - Greg Law - Meeting C++ 2023
 - <https://www.youtube.com/watch?v=qyGdk6QMpMY>
 - Note: Example of following ‘data’
 - Back to Basics: Debugging in Cpp - Greg Law - CppCon 2023
 - <https://www.youtube.com/watch?v=qgszy9GquRs>
 - Back to Basics: Debugging in C++ - Mike Shah - CppCon 2022
 - <https://www.youtube.com/watch?v=YzIBwqWC6EM>
 - Cool New Stuff in Gdb 9 and Gdb 10 - Greg Law - CppCon 2021
 - <https://www.youtube.com/watch?v=xSnetY3eoIk>
 - CppCon 2018: Greg Law “Debugging Linux C++”
 - <https://www.youtube.com/watch?v=V1t6faOKjuQ>
 - CppCon 2016: Greg Law “GDB - A Lot More Than You Knew”
 - <https://www.youtube.com/watch?v=-n9Fkq1e6sg>
- Time Travel Case Studies
 - Quake 2 <https://www.jwhitham.org/2015/05/review-undodb-reversible-debugger.html>
 - Doom - Reviving a zombie: <https://www.youtube.com/watch?v=tjJLZ1da6xs>

Summary

We have many tools to help us!

- For Understanding the control flow
 - Use debuggers and profilers to find what is important and ‘slow down’
 - ‘attaching’ to live running processes is incredibly helpful
- The pure scale of software is often one factor.
 - Try to reduce the scale
 - Use documentation tools and visualizations
 - docs are also easy to ‘bookmark’ and recall over time as you understand software
 - Take your own notes as you learn!
- Human factors can additionally contribute to making your life harder
 - Find ‘small’ problems to understand first
 - Don’t be afraid to ask for help



The international
C++ conference
in the UK, by the sea

Thank you C++ on Sea 2025!

Understanding Large and Unfamiliar Codebases

Chris Croft-White
Staff Solutions Architect
chris@undo.io

undo

Web : mshah.io

60 minutes | Intermediate Audience

YouTube www.youtube.com/c/MikeShah

09:15 - 10:15 Tuesday, June 24, 2025

Social: mikeshah.bsky.social

Courses: courses.mshah.io

Talks: <http://tinyurl.com/mike-talks>

