

C++ FRench User Group

Développeurs C++ de tous les pays, rencontrez-vous!



Understanding Large and Unfamiliar Codebases

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

60 minutes | Intermediate Audience
19:00 - 20:00 Thur, June 19, 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://cpponseas.uk/2025/session/understanding-large-and-unfamiliar-codebases>

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 ever 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 ever worked on? (3/3)

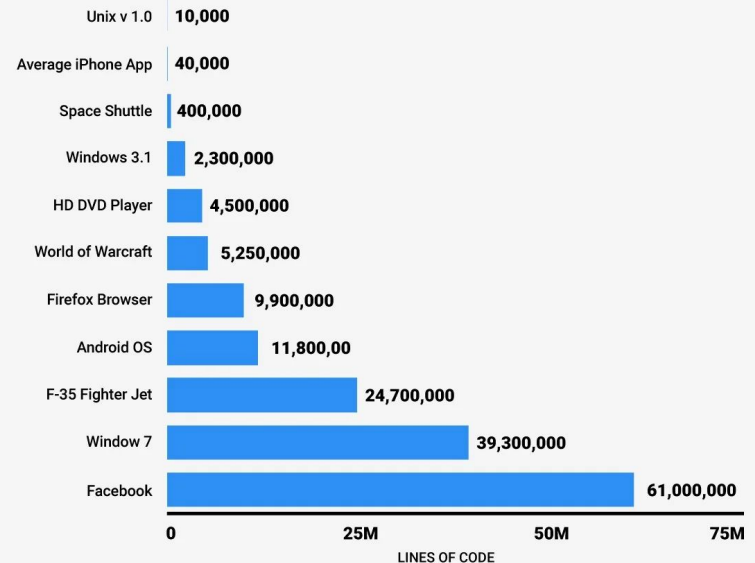
Follow-up question(s):

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 MANY LINES OF CODE MAKE UP THESE POPULAR TECHNOLOGIES



SOURCE: NASA, Quora, Ohloh, Wired

BUSINESS INSIDER

Figure source:
https://external-preview.redd.it/RP36x-Dy-3iZn_CEMh_mEfs7GlyvhlKlasyWAOk2v_M.jpg?auto=webp&s=b6177113412f10a5c9a26727387364e8647883d
Wired Article for more on code sizes: <https://www.wired.com/2015/09/google-2-billion-lines-code-and-one-place/>

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!

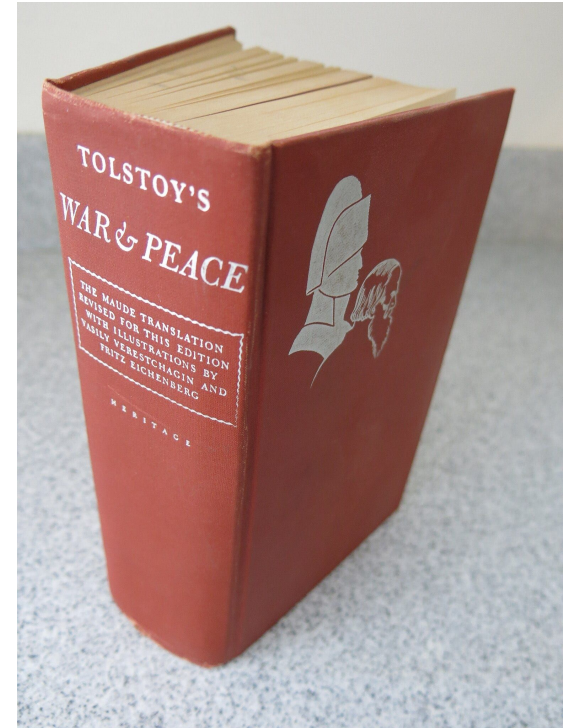


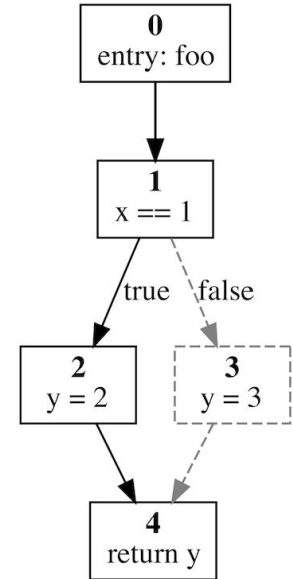
Figure source:
https://stackoverflow.com/image/fetch/?auto=q_auto&good_fit=progressive&steep=https%3A%2F%2Fsubstack-media.s3.amazonaws.com%2Fpublic%2Fimages%2F46832694-698e-468f-840b-4c7f89c6503c_1538x2048.jpeg

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

What makes a codebase hard to understand? (2/4)

- **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? (3/4)

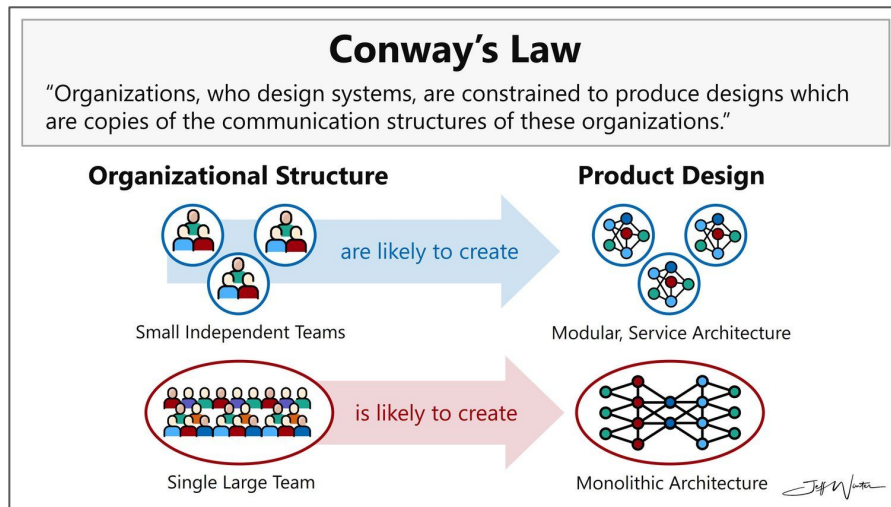
- The **pure scale** of software and its relation to complexity
 - i.e. The number of lines of source code
- Different variations of code
 - e.g. Legacy C++ mixed with C++26
 - Different ‘design patterns’ or architecture design within the source code

```
1 // ...
2
3 //<div data-bbox="616 512 875 889" data-label="Image">
```

‘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? (4/4)

- **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.



Every heard of Conway's law? Organizational structure can potentially impact the very structure of code itself!

<https://www.jeffwinterinsights.com/insights/conways-law-enterprise-architecture>

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)

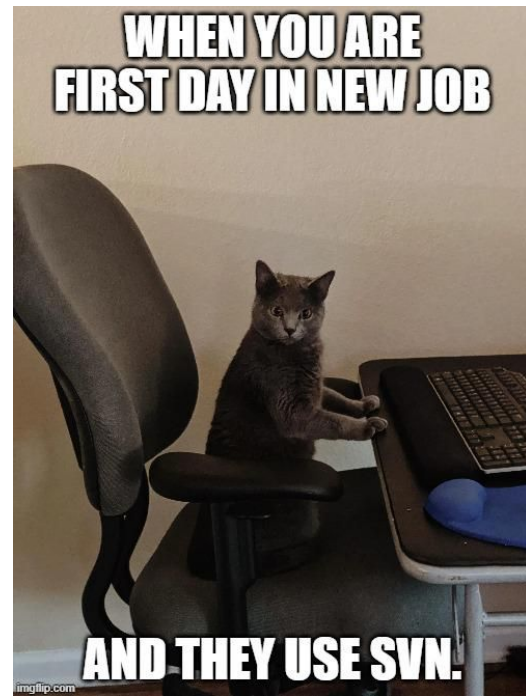


Ah, I have found memories of TortoiseSVN -- it's still active as far as I know!

<https://preview.redd.it/whatdididont-vd-3v70uo8hdvc1-lineo2width=320&crop=smart&auto=webp&s=5d56747737e31e0dfba5ca86ce9fde9a9173a>

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/whatdididont-vd-3v79uo8hdvc1-lineo2width=320&crop=smart&auto=webp&s=456747737e31e0wfbascan6ceh4dew9f173a>

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 there 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 or 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 talks 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!

```
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$
```

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 -irln "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/_CMakeLT0Test-CXX/src/main.cpp:3:int main()
./build/CMakeFiles/3.28.3/CompilerIdC/CMakeCCompilerId.c:848:void main() {}
grep: ./build/CMakeFiles/openttd lib.dir/src/video/opengl.cpp.o: binary file matches
./os/emscripten/cmake/FindLibZMA.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/_CMakeLT0Test-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?**



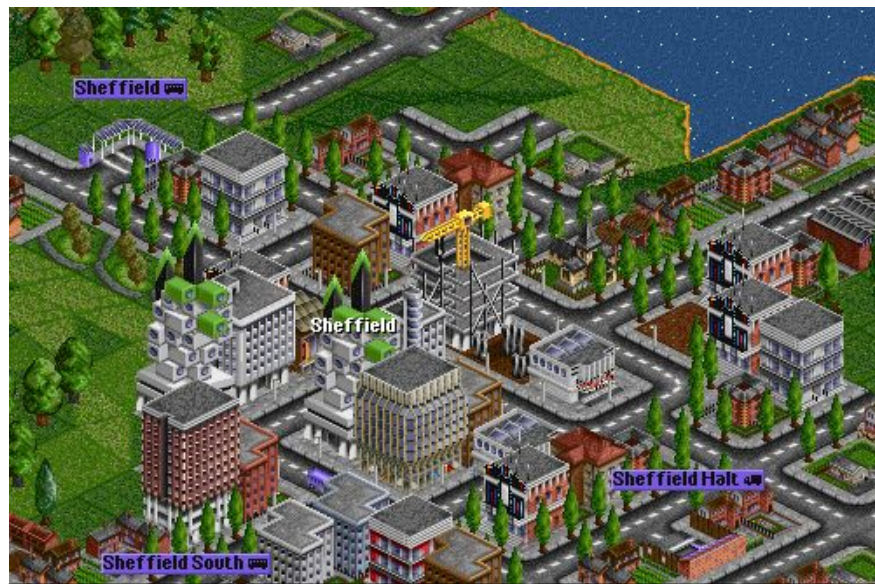
A nice write up on Debug information is here:

- <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

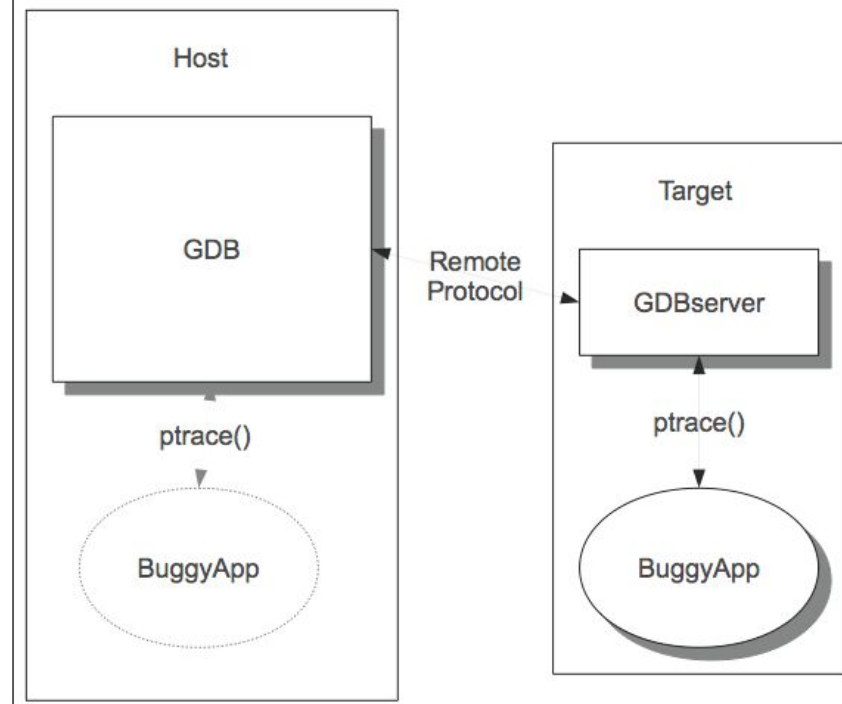


A nice write up on Debug information is here:
<https://developers.redhat.com/articles/2022/01/10/gdb-developers-gnu-debugger-tutorial-part-2-all-about-debuginfo>

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 image shows a terminal window with a dark background. The top of the window has a title bar that reads "mike@mike-MS-7B17: ~/Downloads/openttd/openttd-14.1-source/openttd-14.1". The terminal content shows the following sequence of commands and output:

```
mike@mike-MS-7B17:openttd-14.1$  
  
mike@mike-MS-7B17:build$ ./openttd  
  
[0] 0:bash*
```

At the bottom right of the terminal, there is a status bar that reads: "mike@mike-MS-7B17: ~/ " 17:07 09-Jun-25".

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

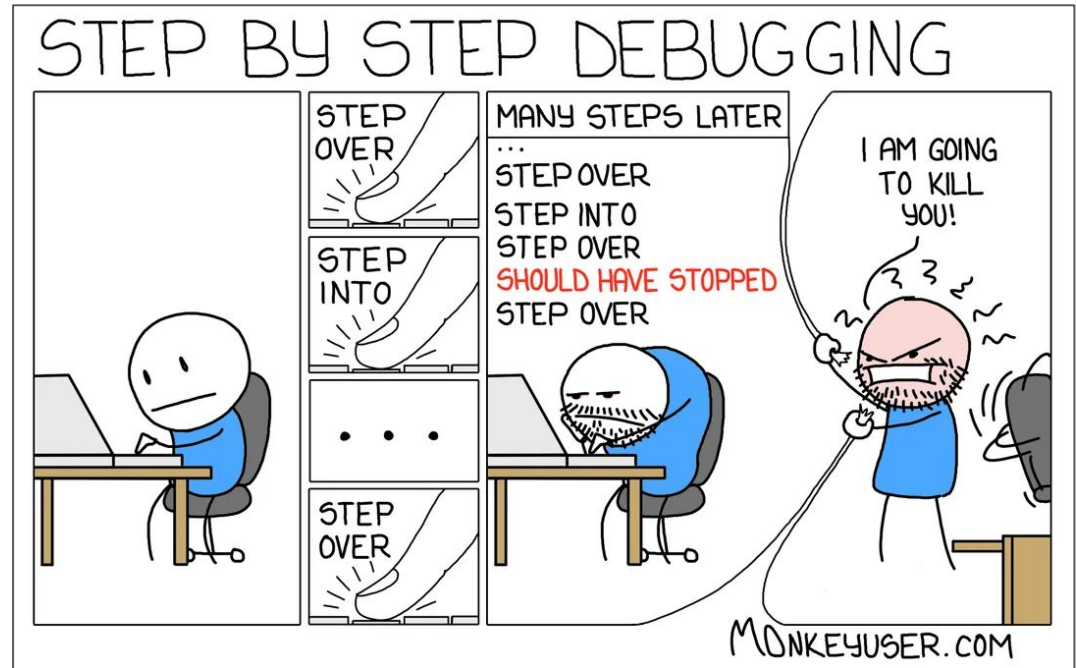
GDB cheatsheet - page 1	
Running	<where>
# gdb <program> (core dump) Start GDB (with optional core dump).	function_name Break/watch the named function.
# gdb --args <program> <args...> Start GDB and pass arguments	line_number Break/watch the line number in the current source file.
# gdb --pid <pid> Start GDB and attach to process.	file:line_number Break/watch the line number in the named source file.
set args <args...> Set arguments to pass to program to be debugged.	Conditions
run Run the program to be debugged.	break/watch <where> if <condition> Break/watch at the given location if the condition is met. Conditions may be almost any C expression that evaluate to true or false.
kill Kill the running program.	condition <breakpoint#> <condition> Set/change the condition of an existing break- or watchpoint.
Breakpoints	Examining the stack
break <where> Set a new breakpoint.	backtrace where Show call stack.
delete <breakpoint#> Remove a breakpoint.	backtrace full where full Show call stack, also print the local variables in each frame.
clear Delete all breakpoints.	frame <frame#> Select the stack frame to operate on.
enable <breakpoint#> Enable a disabled breakpoint.	Stepping
disable <breakpoint#> Disable a breakpoint.	step Go to next instruction (source line), diving into function.
Watchpoints	
watch <where> Set a new watchpoint.	
delete/enable/disable <watchpoint#> Like breakpoints.	
	next Go to next instruction (source line) but don't dive into functions.
	finish Continue until the current function returns.
	continue Continue normal execution.
	Variables and memory
	print/format <what> Print content of variable/memory location/register.
	display/format <what> Like 'print', but print the information after each stepping instruction.
	undisplay <display#> Remove the 'display' with the given number.
	enable display <display#> disable display <display#> En- or disable the 'display' with the given number.
	x/nfu <address> Print memory. n: How many units to print (default 1). f: Format character (like 'print'). u: Unit. Unit is one of: b: Byte, h: Half-word (two bytes) w: Word (four bytes) g: Giant word (eight bytes)).
	© 2007 Marc Haisenko <marc@darkdust.net>

GDB Cheat sheet (page 1 of 2)

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

Reverse debugging (1/5)

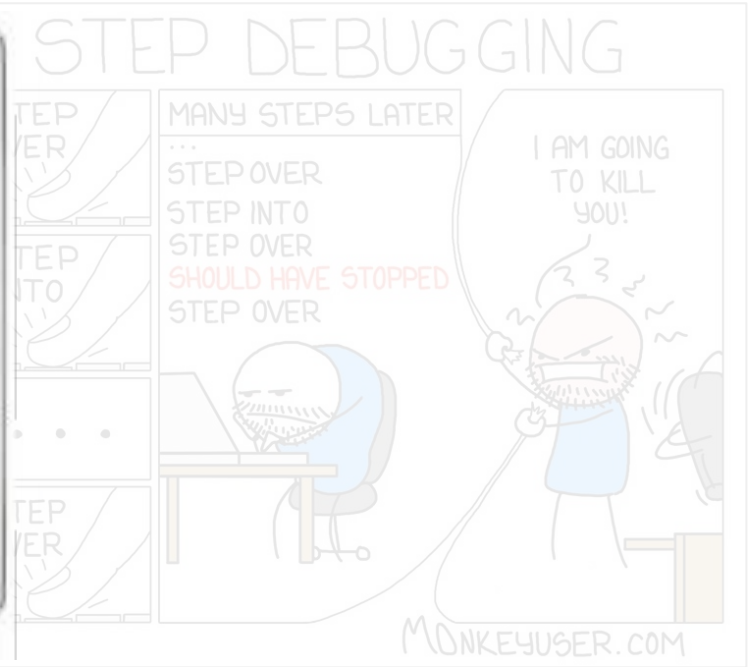
- Just a brief aside on debuggers
- At some point you will probably have this experience.
 - So it's worth mentioning 'reverse-debuggers' or 'time travel debuggers' to you :)



<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/e4b0jejmlxxz.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!

GDB Wiki [Login](#)

Search

Self: [ReverseDebug](#)

[HomePage](#) [RecentChanges](#) [FindPage](#) [HelpContents](#) [ReverseDebug](#)

[Immutable Page](#) [Info](#) [Attachments](#) [More Actions:](#)

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:

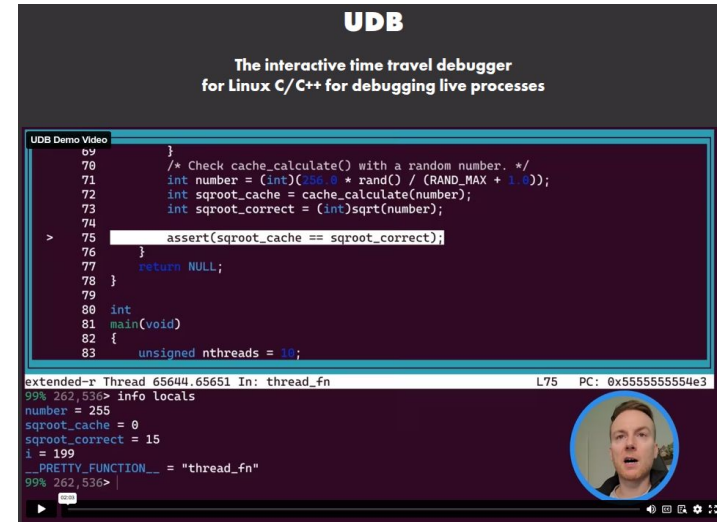
- Certain remote targets including the Simics and SID simulators, and "Undo-db"
- The [Process Record and Replay](#) target for native linux.

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

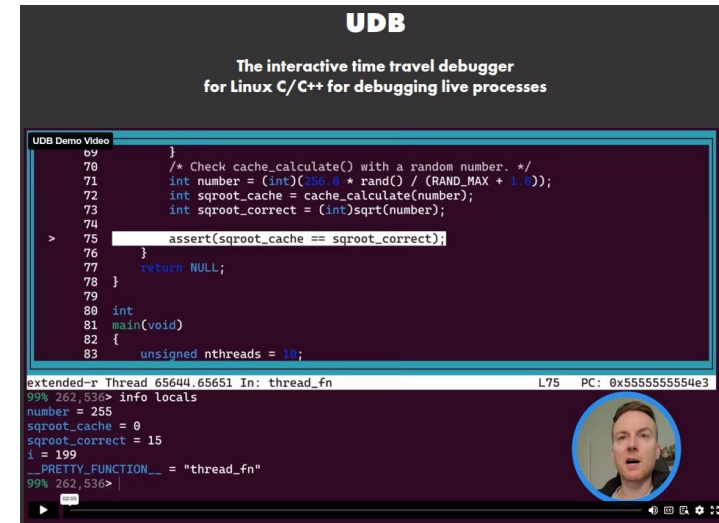


Reverse debugging (5/5)

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/udb-free-trial/>
- Okay -- so let's see how reverse debugging can help us

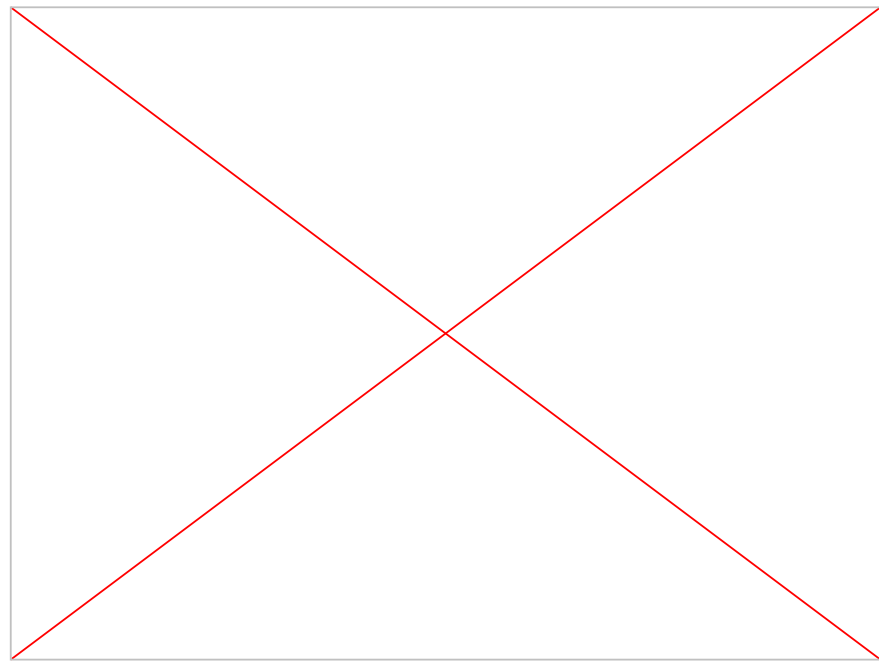


Reverse debugging (UDB) (1/3)

- 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/udb 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.



Time Travel Debugging - Greg Law - Meeting C++ 2023

<https://www.youtube.com/watch?v=qyGdk6QMpMY>

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=SW1fzrB-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

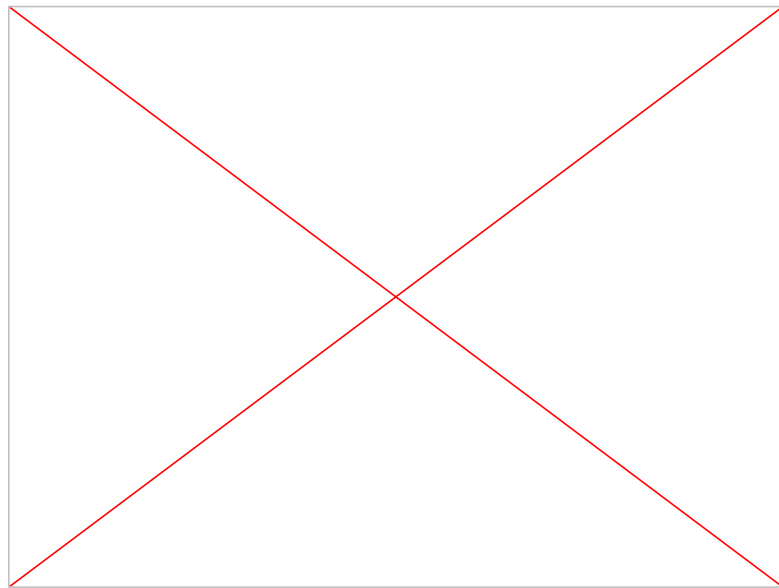
```
sudo perf record ./openttd  
sudo perf report
```

```
Samples: 11K of event 'cycles:P', Event count (approx.): 12994629772  
Overhead Command Shared Object Symbol  
8.23% openttd libgomp.so.1.0.0 [.] 0x0000000000256c0  
6.61% openttd libgomp.so.1.0.0 [.] 0x0000000000258a0  
4.17% openttd openttd [.] Sprite* Blitter_32bppOptimized::EncodeInternal<false>(std::a  
2.71% openttd openttd [.] void Blitter_40bppAnim::Draw<(BlitterMode)0>(Blitter::Blitte  
2.34% openttd openttd [.] unsigned char const* std::max_element<unsigned char const*  
2.02% openttd openttd [.] ResizeSpriteIn(std::array<SpriteLoader::Sprite, 6ul>&, ZoomL  
1.69% openttd openttd [.] bool __gnu_cxx::ops::Iter_less_iter::operator()<unsigned  
1.35% ottd:game openttd [.] Sprite* Blitter_32bppOptimized::EncodeInternal<false>(std::a  
1.33% openttd openttd [.] Sprite* Blitter_32bppOptimized::EncodeInternal<true>(std::a  
0.99% openttd openttd [.] std::initializer_list<unsigned char>::begin() const  
0.91% openttd openttd [.] void Blitter_40bppAnim::Draw<(BlitterMode)1>(Blitter::Blitte  
0.88% openttd openttd [.] unsigned char std::max<unsigned char>(std::initializer_list<  
0.84% openttd openttd [.] GetTileSlope(StrongType::Typedef<unsigned int, TileIndexTag,  
0.78% openttd openttd [.] std::array<SpriteLoader::Sprite, 6ul>::operator[]<unsigned l  
0.70% ottd:game openttd [.] IsValidTile(Tile)  
0.67% ottd:game openttd [.] unsigned char const* std::max_element<unsigned char const*  
0.65% openttd openttd [.] std::initializer_list<unsigned char>::end() const  
0.57% ottd:game openttd [.] StrongType::Typedef<unsigned int, TileIndexTag, StrongType::  
0.56% ottd:game openttd [.] RunTileLoop()  
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  
0.47% openttd openttd [.] ResizeSpriteOut(std::array<SpriteLoader::Sprite, 6ul>&, Zoom  
0.45% ottd:game openttd [.] CallVehicleTicks()  
0.45% openttd openttd [.] Colour::Colour(unsigned int)  
0.45% ottd:game openttd [.] ResizeSpriteIn(std::array<SpriteLoader::Sprite, 6ul>&, ZoomL  
0.44% openttd [kernel.kallsyms] [k] nv044975rm  
0.40% ottd:game openttd [.] TileLoop_Water(StrongType::Typedef<unsigned int, TileIndexTa  
0.35% openttd libc.so.6 [.] memmove_avx_unaligned_erms  
0.34% openttd openttd [.] DecodeSingleSprite(SpriteLoader::Sprite*, SpriteFile&, unsi  
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, BlitterMo  
0.31% openttd openttd [.] AllocSprite(unsigned long)  
0.31% ottd:game openttd [.] unsigned char std::max<unsigned char>(std::initializer_list<  
0.28% openttd libc.so.6 [.] memset_avx2_unaligned_erms  
0.28% ottd:game openttd [.] GetTileSlope(StrongType::Typedef<unsigned int, TileIndexTag,  
0.28% ottd:game openttd [.] AllocSprite(unsigned long)  
0.26% openttd openttd [.] ViewportSortParentSpritesSSE41(std::vector<ParentSpriteToDra  
0.25% openttd libc.so.6 [.] std::vector<char const*, std::allocator<char const*> >::back  
0.25% openttd openttd [.] malloc  
0.24% openttd openttd [.] StrongType::Typedef<unsigned int, TileIndexTag, StrongType::  
[.] Blitter_40bppAnim::DrawRect(void*, int, unsigned char)
```

Attaching to a Running Process

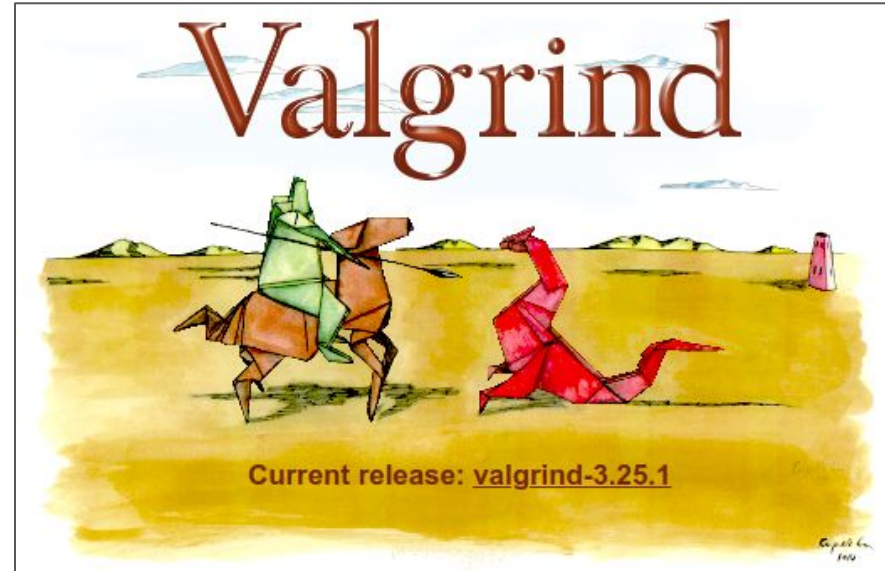
- Note: With most of the tools I'm showing you today (GDB, UDB, etc.) it's possible to 'attach to a running process'
 - This allows us to collect information from a specific point and time.

```
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

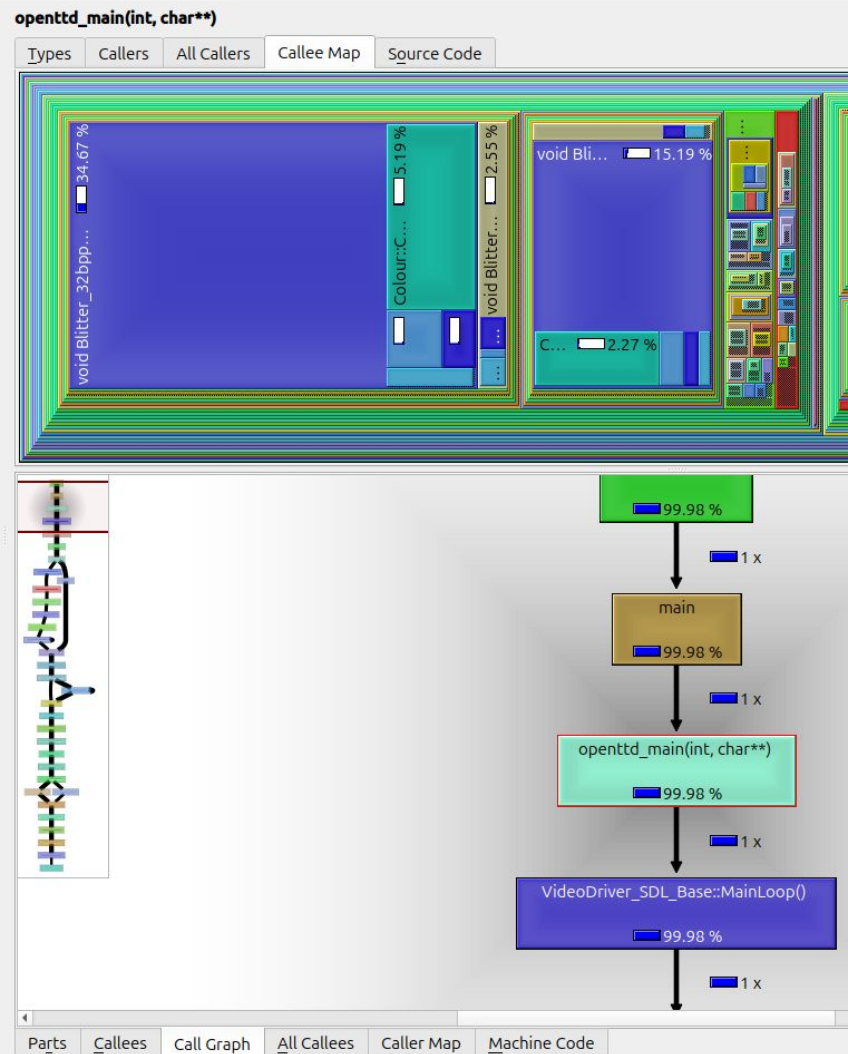


Visualization: callgrind and kcachegrind (2/2)

- 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

F&lat Profile

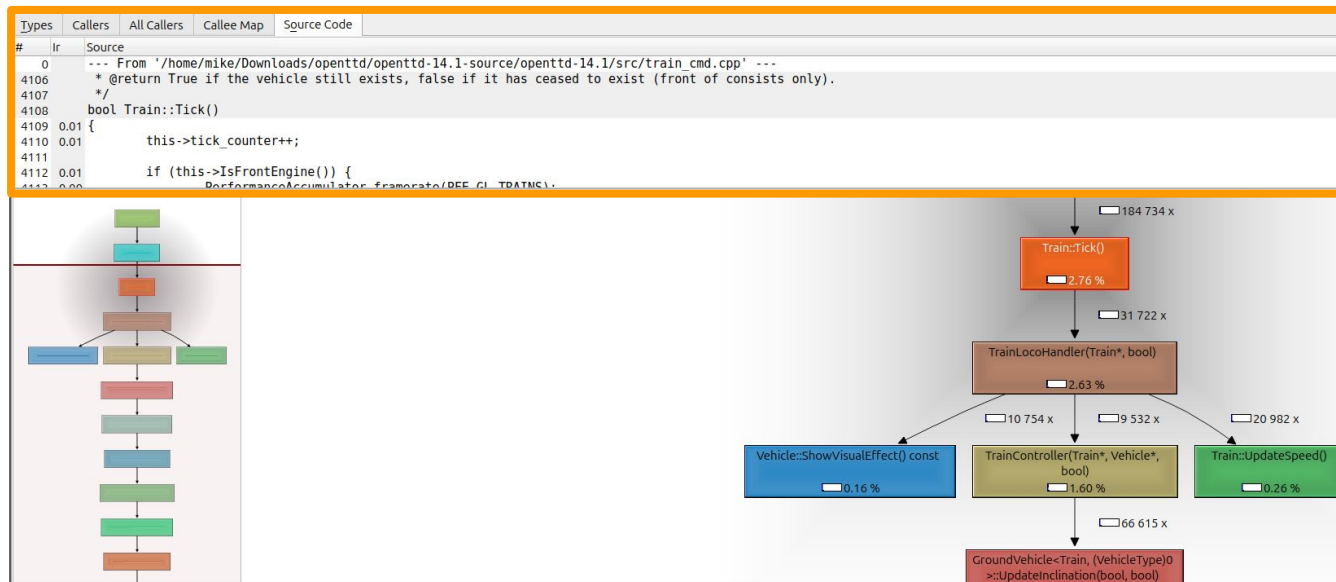
Search: tick (No Grouping)

Incl.	Self	Called	Function	Location
49.59	0.00	25	VideoDriver::Tick()	openttd: video_driver.cpp
31.92	0.00	18	CallWindowRealtimeTic...	openttd: window.cpp
31.92	0.00	18	SelectGameWindow::On...	openttd: intro_gui.cpp
8.78	0.18	311	CallVehicleTicks()	openttd: vehicle.cpp, vehicle_base.h, bitmath...
2.92	0.00	5 598	Ship::Tick()	openttd: ship_cmd.cpp
2.76	0.07	184 734	Train::Tick()	openttd: train_cmd.cpp, vehicle_base.h, bitma...
1.89	0.02	49 138	RoadVehicle::Tick()	openttd: roadveh_cmd.cpp, vehicle_base.h, bi...
0.36	0.00	5 598	Aircraft::Tick()	openttd: aircraft_cmd.cpp
0.23	0.00	311	CallLandscapeTick()	openttd: landscape.cpp
0.19	0.01	311	OnTick_Station()	openttd: station_cmd.cpp
0.11	0.00	23 243	EffectVehicle::Tick()	openttd: effectvehicle.cpp
0.10	0.00	22 112	DieselSmokeTick(Effect...	openttd: effectvehicle.cpp
0.09	0.00	20 837	StationHandleSmallTick...	openttd: station_cmd.cpp
0.07	0.00	67	StationHandleBigTick(B...	openttd: station_cmd.cpp
0.03	0.00	311	OnTick_Industry()	openttd: industry_cmd.cpp
0.01	0.00	1 131	SmokeTick(EffectVehicl...	openttd: effectvehicle.cpp
0.00	0.00	311	OnTick_Town()	openttd: town_cmd.cpp



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

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

50

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
 - Given build / test is often remote, a good screen or screens can be hugely helpful too (although this has a cost, so I understand if you'd prefer not to recommend it to everyone)



https://i.pcmag.com/imagery/roundups/01Y9bqNdRmGOzHcetHOG2FW-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'

```
// A material that allows for multiple textures
module multitexturematerial;
import pipeline, materials, texture;
import glBindc, OpenGL;

// Represents a material with multiple textures
class MultiTextureMaterial : Material {
    Texture mTexture1;
    Texture mTexture2;
    Texture mTexture3;
    Texture mTexture4;

    // Construct a new material for a pipeline, and load a texture for that pipeline
    this(string pipelineName,
         string textureFilename1,
         string textureFilename2,
         string textureFilename3,
         string textureFilename4) {
        // Delegate to the base constructor to do initialization
        super(pipelineName);

        mTexture1 = new Texture(textureFilename1);
        mTexture2 = new Texture(textureFilename2);
        mTexture3 = new Texture(textureFilename3);
        mTexture4 = new Texture(textureFilename4);
    }

    // TextureMaterial.update()
    override void update() {
        // Set our active Shader graphics pipeline
        PipelineID pipelineID;

        // Set any uniforms for our mesh if they exist in the shader
        if ("sample1" in mMiniFormMap) {
            glUniform1i(mMiniFormMap["sample1"], mTexture1.mTextureID);
        }
        if ("sample2" in mMiniFormMap) {
            glUniform1i(mMiniFormMap["sample2"], mTexture2.mTextureID);
        }
        if ("sample3" in mMiniFormMap) {
            glUniform1i(mMiniFormMap["sample3"], mTexture3.mTextureID);
        }
        if ("sample4" in mMiniFormMap) {
            glUniform1i(mMiniFormMap["sample4"], mTexture4.mTextureID);
        }
    }
}

// An example of a normal map material
module normalmapmaterial;
import pipeline, materials, texture;
import glBindc, OpenGL;

// Represents a normal mapped material
// Some notes on 'albedo' vs 'diffuse map' for naming: https://www.a2d3.co/
// load diffuse between albedo and diffuse map
class NormalMapMaterial : Material {
    Texture mTexture1;
    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);

        mTexture1 = new Texture(textureFilename);
        mTexture2 = new Texture(normalmapFilename);
    }

    // TextureMaterial.update()
    override void update() {
        // Set our active Shader graphics pipeline
        PipelineID pipelineID;

        // Set any uniforms for our mesh if they exist in the shader
        if ("normalmap" in mMiniFormMap) {
            glUniform1i(mMiniFormMap["normalmap"], mTexture2.mTextureID);
        }
    }
}

// #version 410 core
// in vec2 vTexCoords;
// out vec4 fFragColor;
// uniform sampler2D albedoMap; // colors from texture
// uniform sampler2D normalMap; // The normal map

void main() {
    vec3 colors = texture(albedoMap, vTexCoords).rgb;
    vec3 normals = texture(normalMap, vTexCoords).rgb;

    fragColor = vec4(colors, 1.0)/2 + vec4(normals, 1.0)/2;
}
```

I primarily use tmux and VIM to split my 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 **udb**, **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

Code Documentation.
Automated.

🔗 Software documentation consulting


Free, open source, 🔍 cross-platform.

<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=tLPHQMosF9M>

```
doxygen -g  
doxygen Doxyfile
```

A terminal window screenshot showing a command prompt. The prompt is 'mike@mike-MS-7B17:openttd-14.1\$' followed by the command 'doxygen'. The terminal output is mostly black, indicating the command is running or has completed. The bottom status bar shows '0: bash*' and the system time '18:09 10-Jun-25'.

```
mike@mike-MS-7B17:openttd-14.1$ doxygen  
[0] 0: bash* "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 >
```

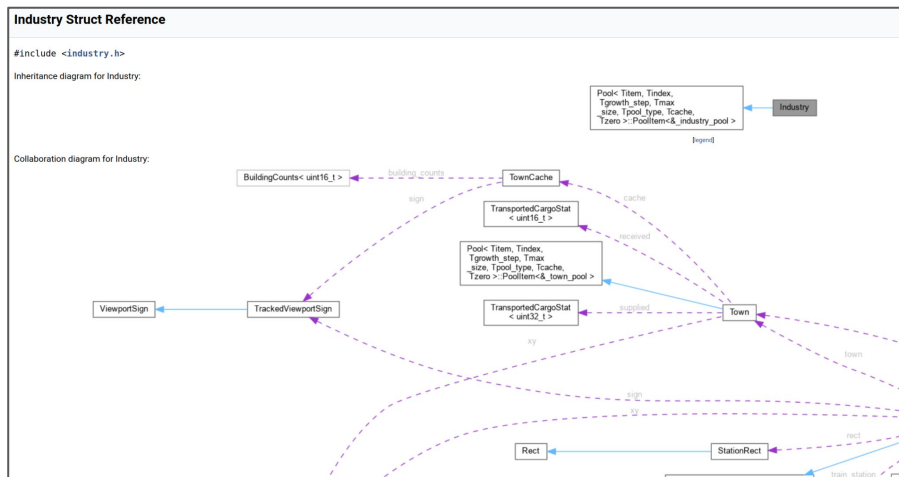
Public Member Functions

```
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)
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

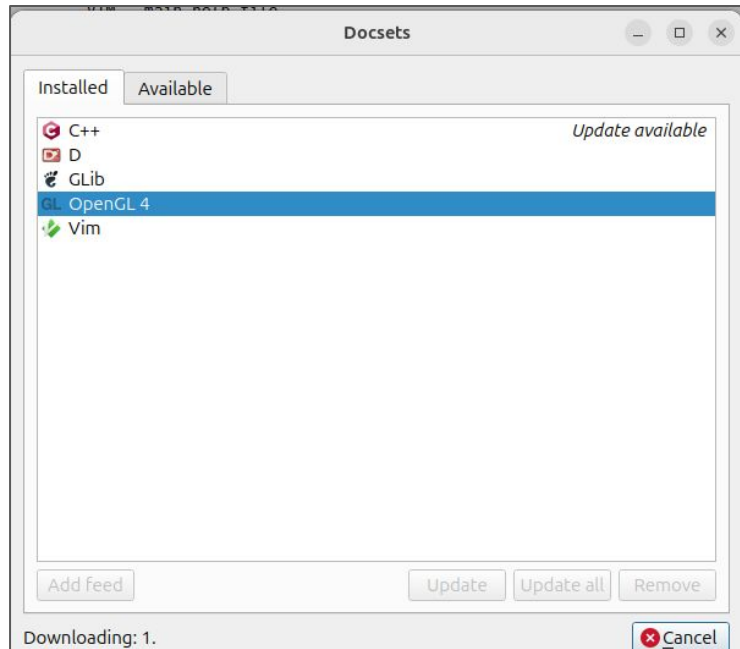
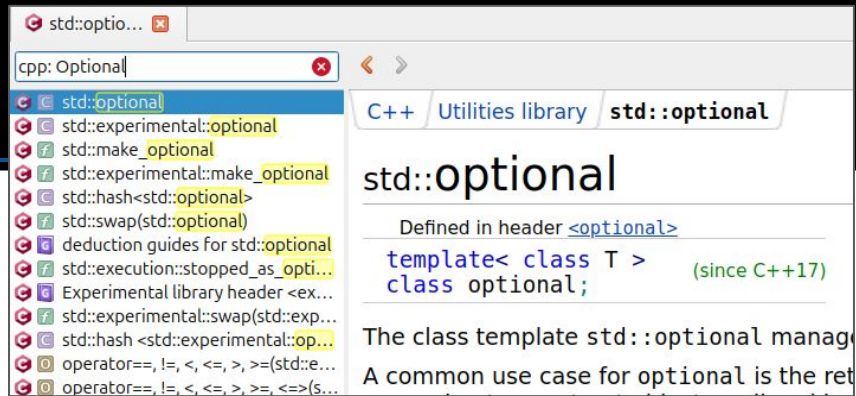
- In any big project, there will also be code that you **do not own**
- For example, shared libraries
- Finding shared libraries is easy
 - Just run a tool like ldd ([Dependency Walker](#) on Windows is also quite nice)
- You'll often get some hints of what 'graphics libraries' to otherwise pay attention 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)
libicu18n.so.74 => /lib/x86_64-linux-gnu/libicu18n.so.74 (0x0000705b97200000)
```

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



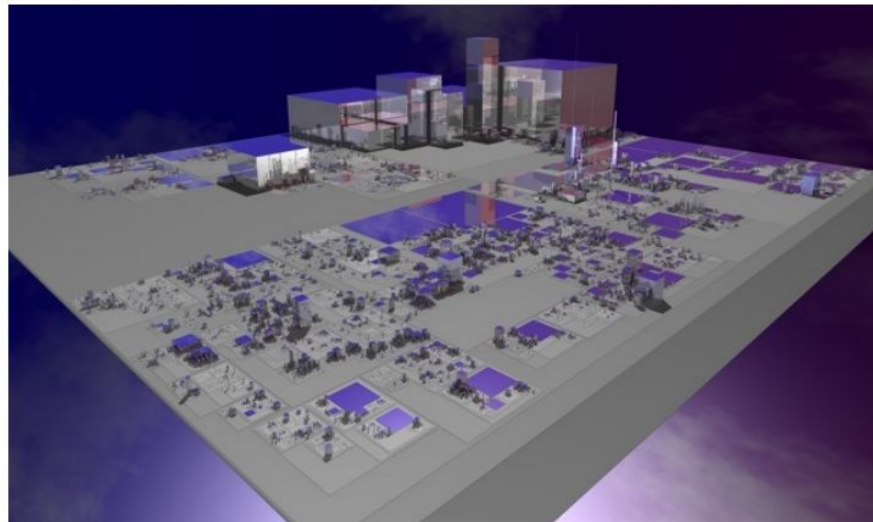
Still too much? Try Running Examples/tests

- 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.

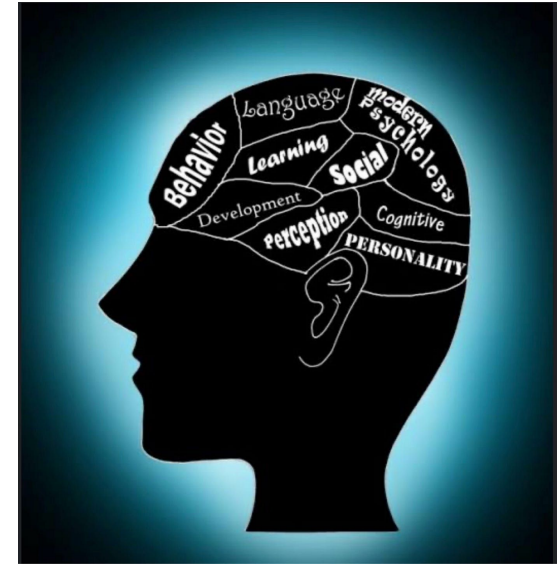
```
int main() {  
    auto lang = "C++";  
    std::cout << "Hello and welcome to " << lang << "!\n";  
  
    std::string str;  
    str.append("Hello friend");  
}
```

local variable
const char *lang

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

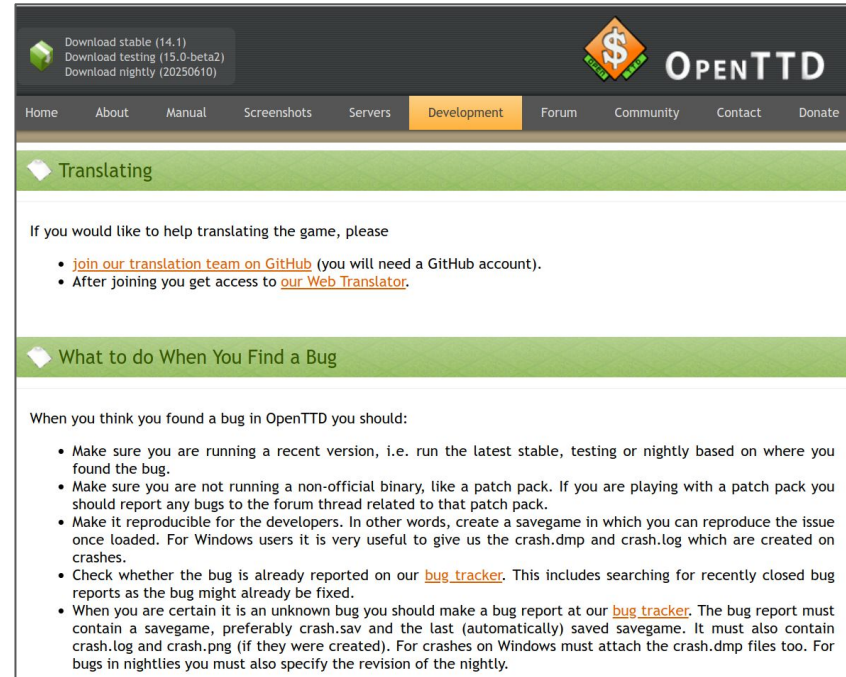
- We have some tools now to help us **understand the control flow** and **working at scale**.
- But sometimes the hardest problems in software are sometimes -- i.e. **human factors**
- What I mean by this is -- how do I get motivated or focused to actually learn a codebase?
 - **My best answer** is to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



https://miro.medium.com/v2/resize:fit:1400/1*nOZxRBAQwg8FZ-lINA630w@2x.jpeg

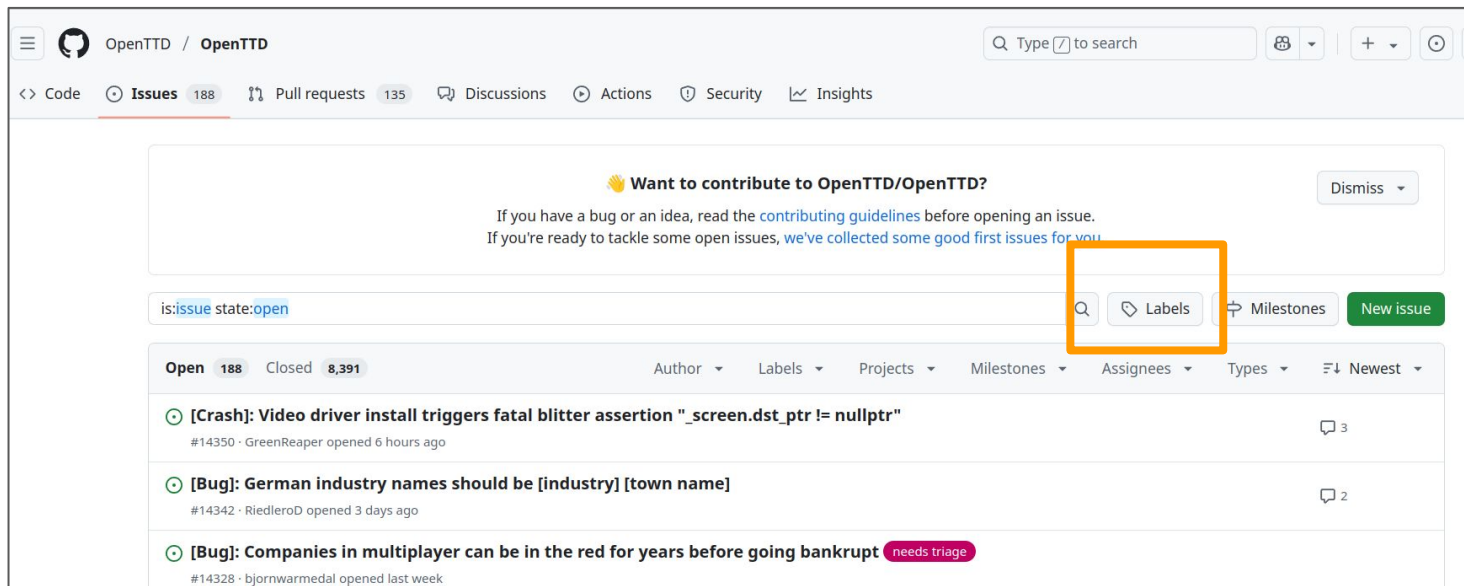
Find a first good task (1/3)

- 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>



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



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

size: large

size: small

size: trivial

stale

waiting on author

waiting on runner update

wip

won't fix

won't implement

work: minor details

work: missing intention

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/midfile.cpp:1069: std::string tmpdirname = FioGetDirectory(Searchpath::SP_AUTODOWNLOAD_DIR);
./src/strgen/strgen_base.cpp:31: int _errors, _warnings, _show_todo;
./src/strgen/strgen_base.cpp:363: if ((_show_todo & 2) != 0) StrgenWarning("{}");
./src/strgen/strgen_base.cpp:753: /* Abusing _show_todo to replace "warning" with "info" for translation
./src/strgen/strgen_base.cpp:754: _show_todo &= 3;
./src/strgen/strgen_base.cpp:755: if (!this->translation) _show_todo |= 4;
./src/strgen/strgen_base.cpp:929: if (_show_todo > 0 && !s->translated.empty()) {
./src/strgen/strgen_base.cpp:930: if ((_show_todo & 2) != 0) {
./src/strgen/strgen_base.cpp:933: if ((_show_todo & 1) != 0) {
./src/strgen/strgen_base.cpp:934: const char *s = "<TODO> ";
./src/strgen/strgen.cpp:38: if (_show_todo > 0) {
./src/strgen/strgen.cpp:318: GETOPT_NOVAL('t', "--todo"),
./src/strgen/strgen.cpp:371: _show_todo |= 1;
./src/strgen/strgen.cpp:375: _show_todo |= 2;
./src/strgen/strgen.cpp:381: " -t | --todo replace any untranslated
./src/strgen/strgen.cpp:462: if ((_show_todo & 2) != 0) {
./src/newgrf_engine.cpp:287: /* @todo Need to find which terminal (or hangar) we've chosen
./src/newgrf_engine.cpp:297: /* @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;
./src/fileio.cpp:98: 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: FioCreateDirectory(_searchpaths[SP_AUTODOWNLOAD_DIR]);
./src/fileio.cpp:1087: FioCreateDirectory(FioGetDirectory(SP_AUTODOWNLOAD_DIR, dirs[i]));
./src/economy.cpp:1815: /* TODO: Regarding this, when we do gradual loading, we
./src/economy.cpp:2038: * @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 the 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 [udb](#) or [gdb](#) to 'track' what function is called when you click on something.
 - udb 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 they 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

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=xSnetY3eolK>
 - 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

C++ French User Group

Développeurs C++ de tous les pays, rencontrez-vous!



Thank you!

Understanding Large and Unfamiliar Codebases

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

60 minutes | Intermediate Audience
19:00 - 20:00 Thur, June 19, 2025

Thank you!