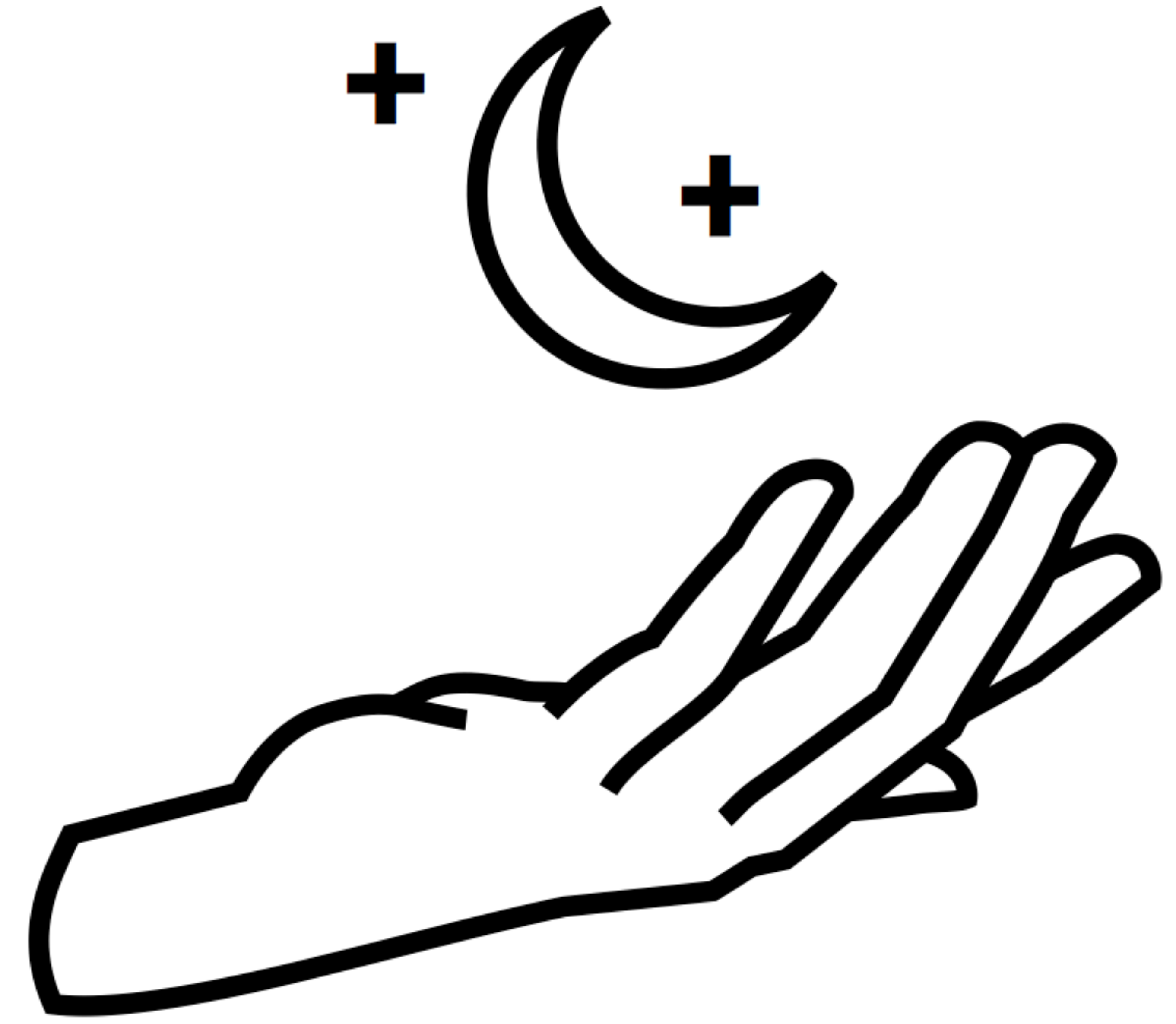
Practical Software Engineering

ADC 2022

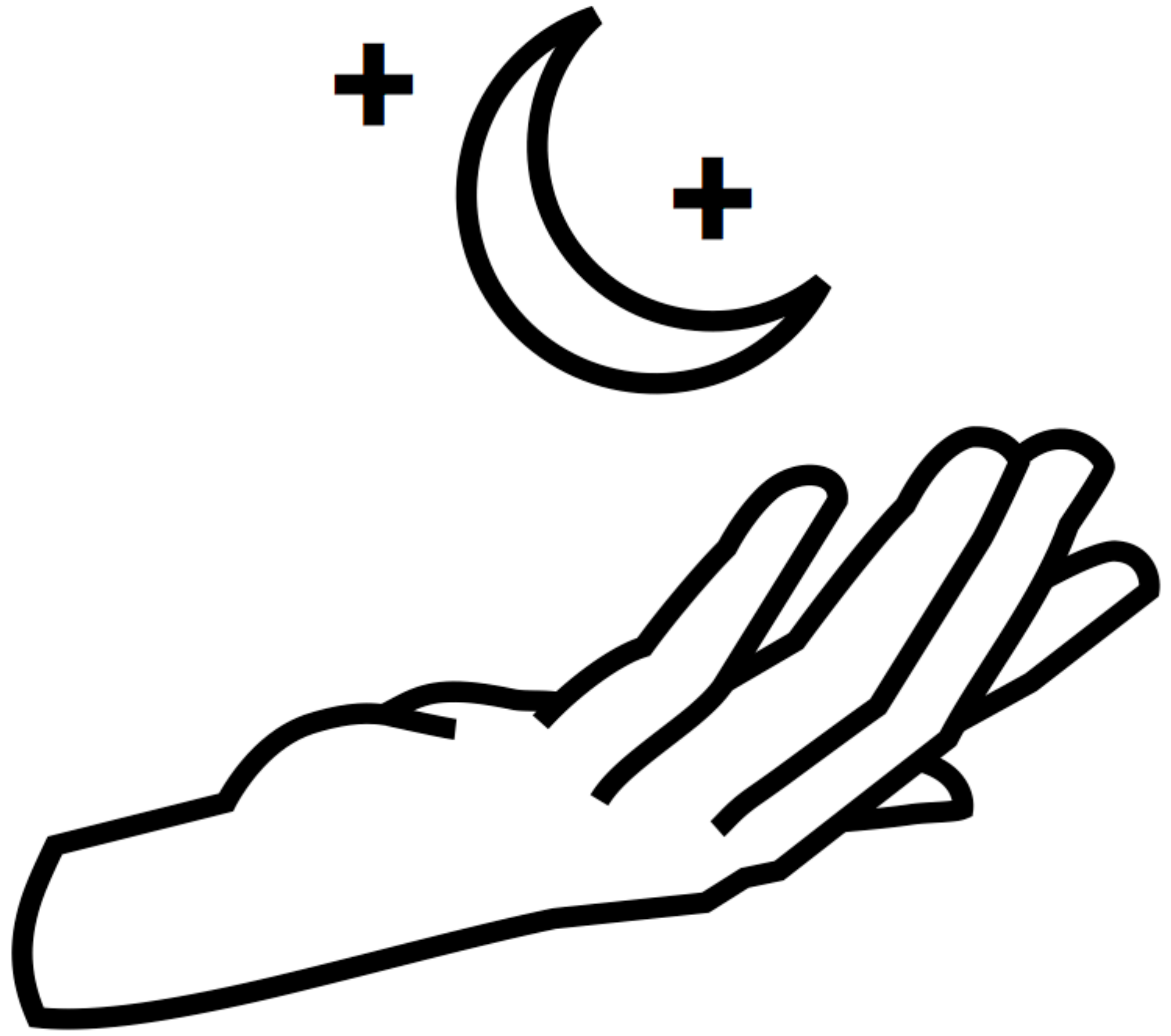


@dynamic_cast

Harriet Drury @drury_harriet

Rachel Locke @Madammodular

Anna Wszeborowska @aniawsz

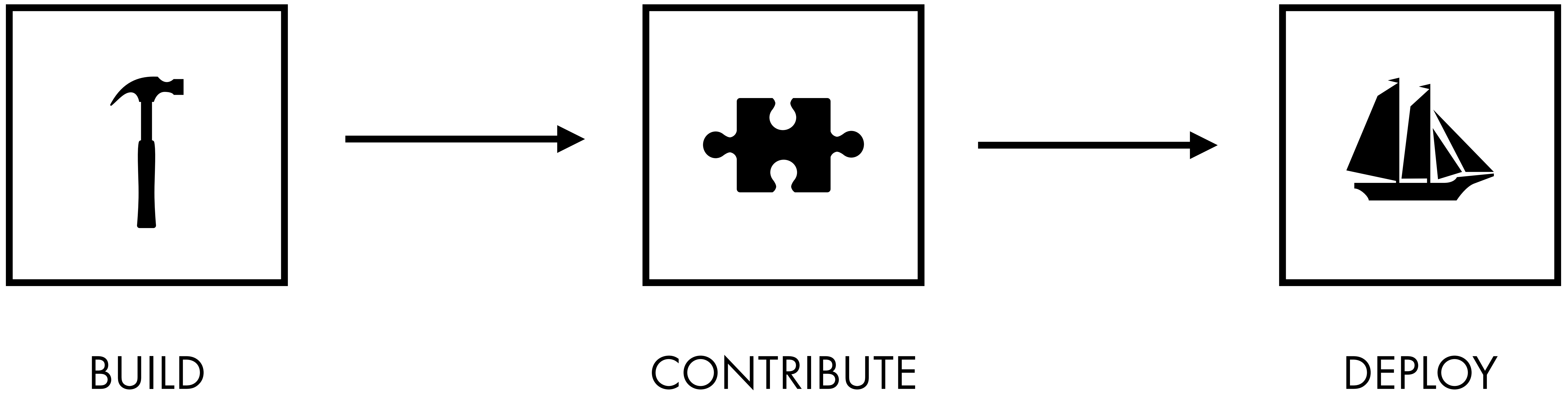


@dynamic_cast

Presentation slides

https://github.com/dynamic-cast/aquila-workshop/wiki

Stages of working with existing code

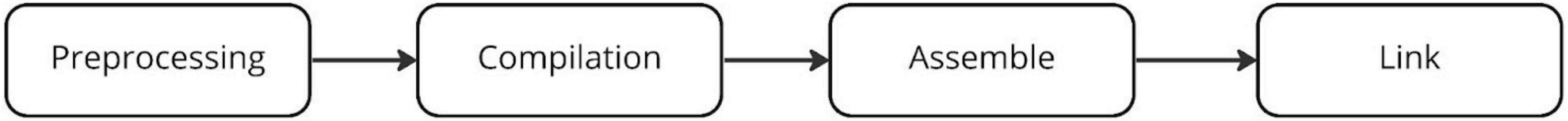


BUILD

From .cpp to .exe…… What does it mean to build?

To build a C++ program means to compile source code from one or more files and then link those files into an executable file, a dynamic-load library or a static library.

The C++ Compilation Model



Preprocessing

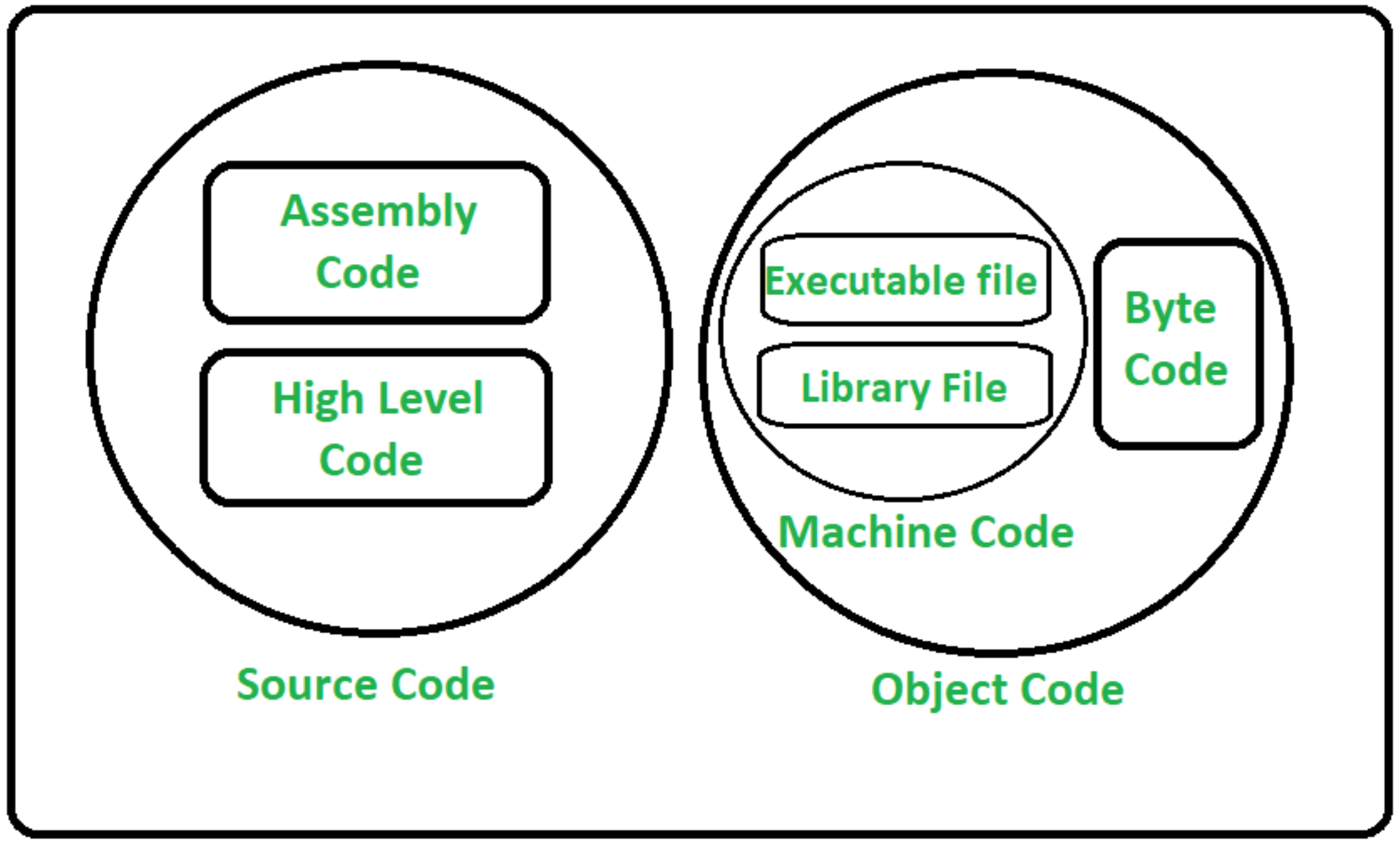
The preprocessing stage takes our source files (.cpp, .h) and deals with the #includes and #defines, user-defined macros, etc.

Compilation

The expanded source code is compiled into assembly code to output and assembler file

Assemble

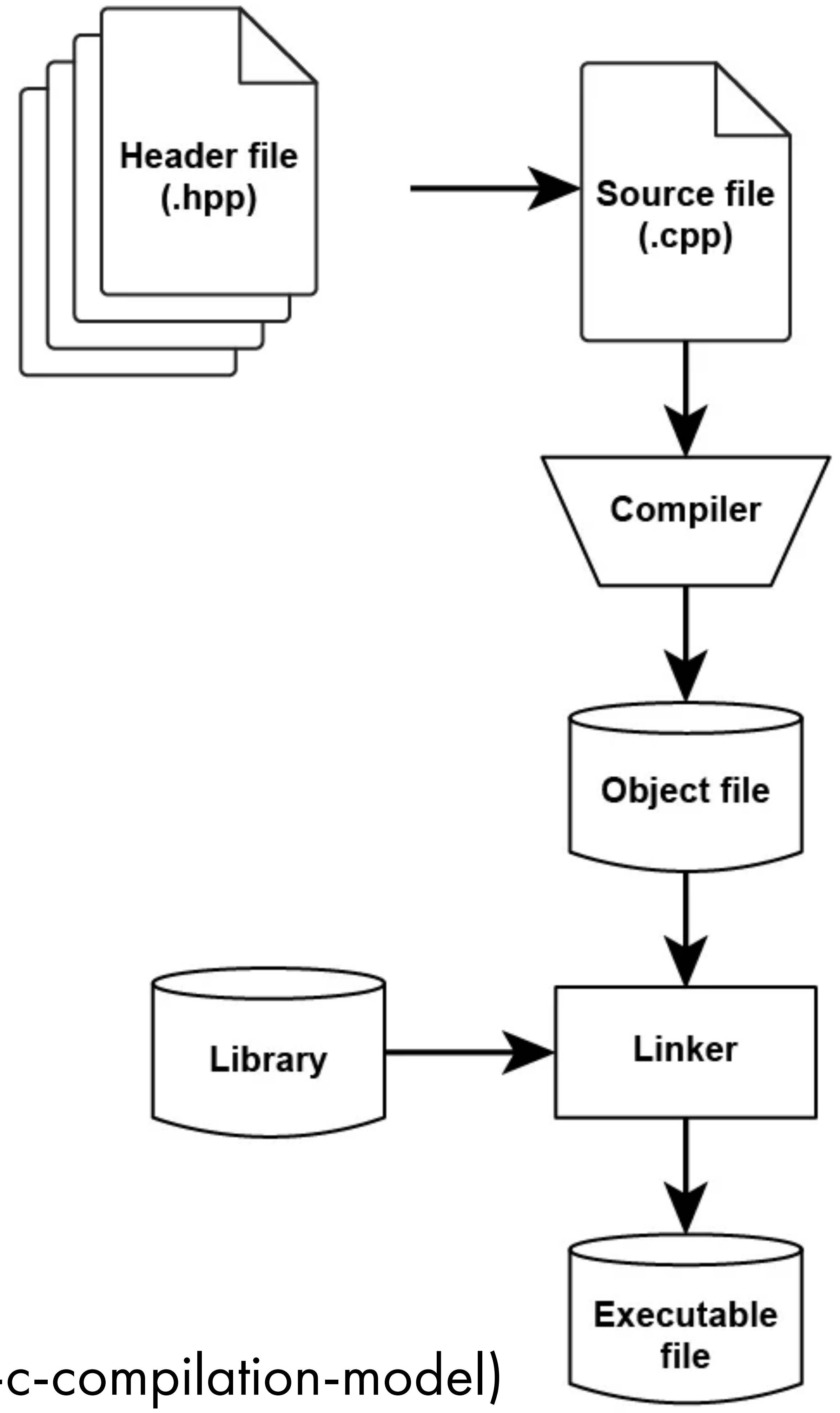
Converts the compiled assembly code into object files



Source (https://www.geeksforgeeks.org/difference-between-source-code-and-object-code/)

Link

This object code file is linked together with the object code files for any library functions to produce an executable file



Source:
(https://subscription.packtpub.com/book/programming/9781789801491/1/ch01lvl1sec03/the-c-compilation-model)

Build systems - CMake

We use build systems to control builds across multiple platforms, it automates the compiling process.

CMake is a generator of build systems. It generates makefiles, this includes instructions on how to build your code. It can be used to target multi platform build systems such as:

- Visual Studio
- Xcode
- KDEvelop

CMake

CMake Benefits:

- Has a scripting language that can be easy to learn and integrate into your workflow (CMakeLists.txt)
- Good documentation
- Widespread usage

Building an open source project with CMake

Software/prerequisites to have:

- Git
- CMake
- IDE of choosing (Visual Studio, XCode, etc)

Install these to continue with the Aquila build!

Nice (Free!) software to consider:

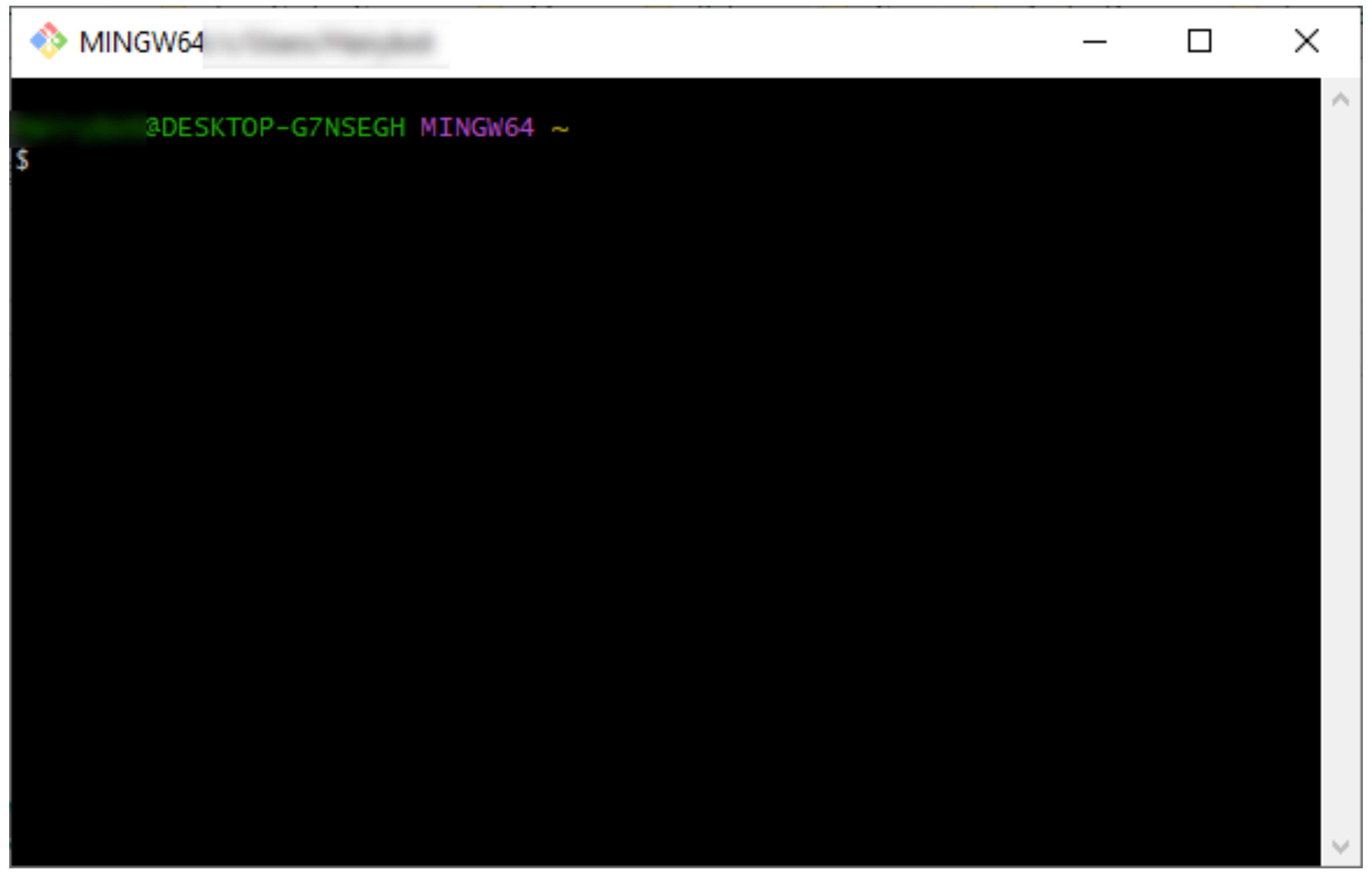
- Sourcetree (Git GUI Client)

Git

https://git-scm.com/

A free, open source, versioning control system

On Windows ,once installed, you'll see the git bash, gui and cmd applications. We'll be using the bash.



```
[redacted]@DESKTOP-G7NSEGH MINGW64 ~
$
```

Aquila

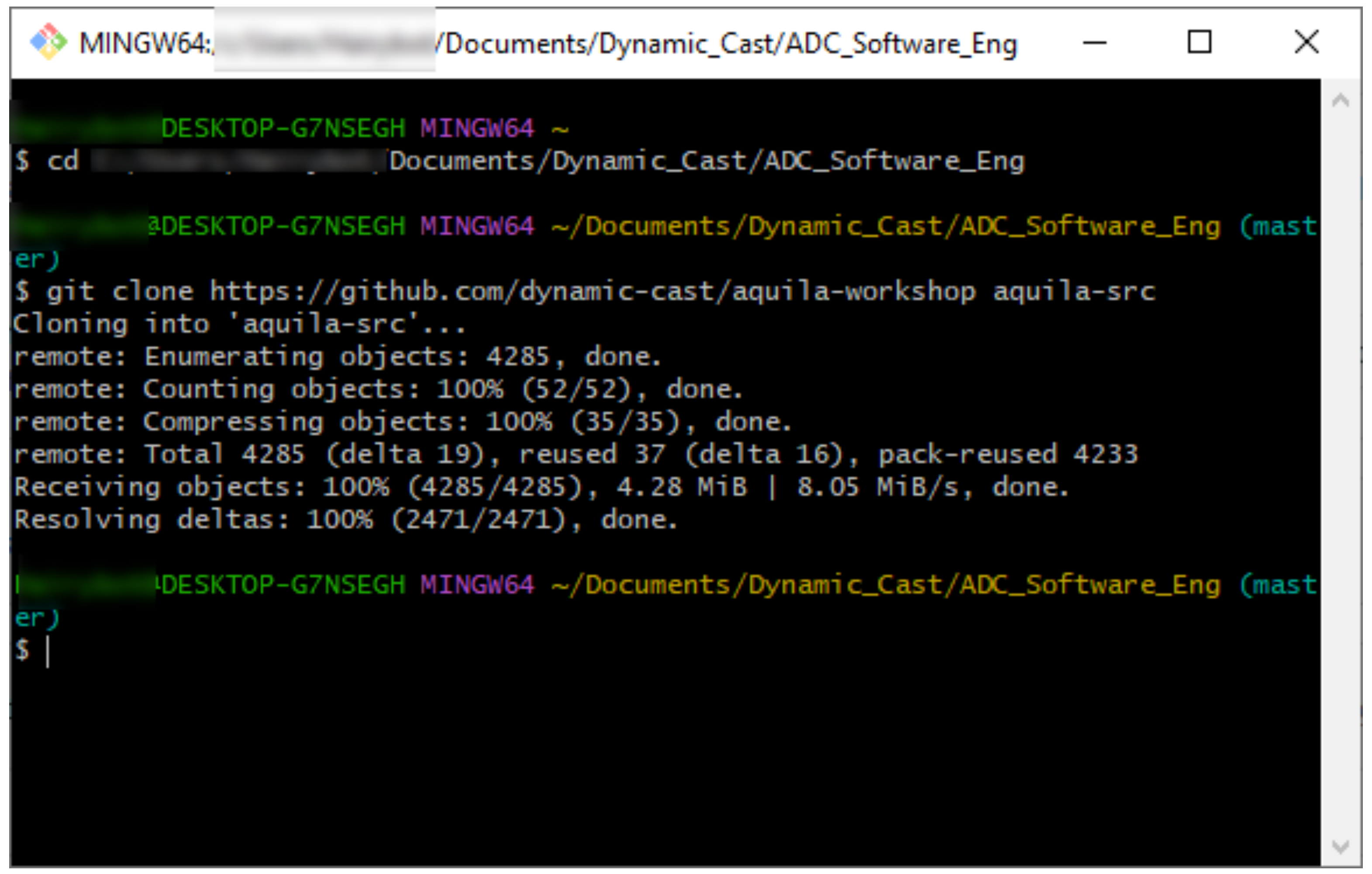
Aquila is an open source and cross-platform DSP (Digital Signal Processing) library written in C++.

https://aquila-dsp.org/

Clone the repository

https://github.com/dynamic-cast/aquila-workshop

1. Create a new folder on your computer and change your directory to point to it
2. Run: git clone https://github.com/dynamic-cast/aquila-workshop aquila-src

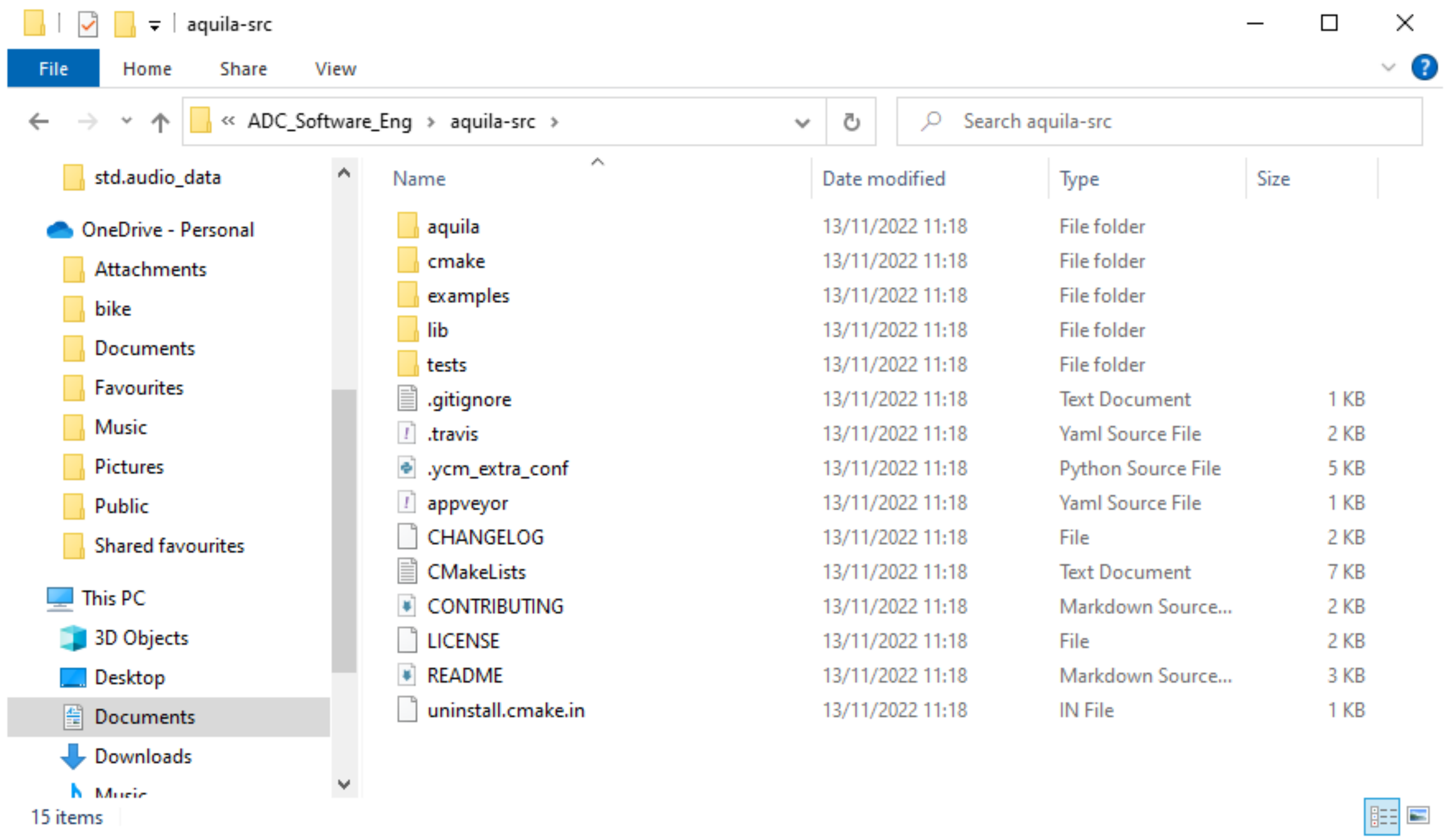


MINGW64:[redacted]/Documents/Dynamic_Cast/ADC_Software_Eng

```
[redacted]DESKTOP-G7NSEGH MINGW64 ~
$ cd [redacted]Documents/Dynamic_Cast/ADC_Software_Eng

[redacted]@DESKTOP-G7NSEGH MINGW64 ~/Documents/Dynamic_Cast/ADC_Software_Eng (mast
er)
$ git clone https://github.com/dynamic-cast/aquila-workshop aquila-src
Cloning into 'aquila-src'...
remote: Enumerating objects: 4285, done.
remote: Counting objects: 100% (52/52), done.
remote: Compressing objects: 100% (35/35), done.
remote: Total 4285 (delta 19), reused 37 (delta 16), pack-reused 4233
Receiving objects: 100% (4285/4285), 4.28 MiB | 8.05 MiB/s, done.
Resolving deltas: 100% (2471/2471), done.

[redacted]DESKTOP-G7NSEGH MINGW64 ~/Documents/Dynamic_Cast/ADC_Software_Eng (mast
er)
$ |
```



Name	Date modified	Type	Size
aquila	13/11/2022 11:18	File folder	
cmake	13/11/2022 11:18	File folder	
examples	13/11/2022 11:18	File folder	
lib	13/11/2022 11:18	File folder	
tests	13/11/2022 11:18	File folder	
.gitignore	13/11/2022 11:18	Text Document	1 KB
.travis	13/11/2022 11:18	Yaml Source File	2 KB
.ycm_extra_conf	13/11/2022 11:18	Python Source File	5 KB
appveyor	13/11/2022 11:18	Yaml Source File	1 KB
CHANGELOG	13/11/2022 11:18	File	2 KB
CMakeLists	13/11/2022 11:18	Text Document	7 KB
CONTRIBUTING	13/11/2022 11:18	Markdown Source...	2 KB
LICENSE	13/11/2022 11:18	File	2 KB
README	13/11/2022 11:18	Markdown Source...	3 KB
uninstall.cmake.in	13/11/2022 11:18	IN File	1 KB

CMake Build - command line on Unix platforms

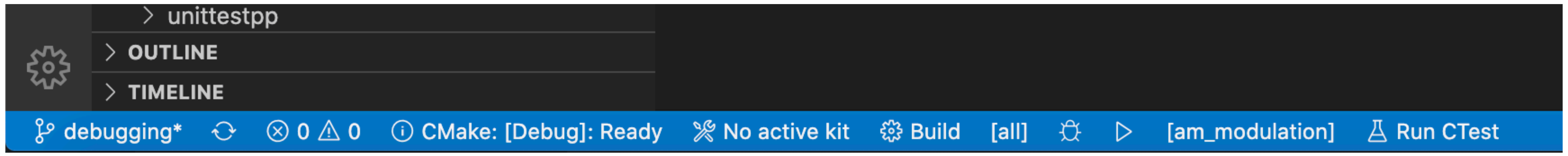
```
mkdir build; cd build

cmake .. -DCMAKE_BUILD_TYPE=Debug
```

CMake Build - VS Code

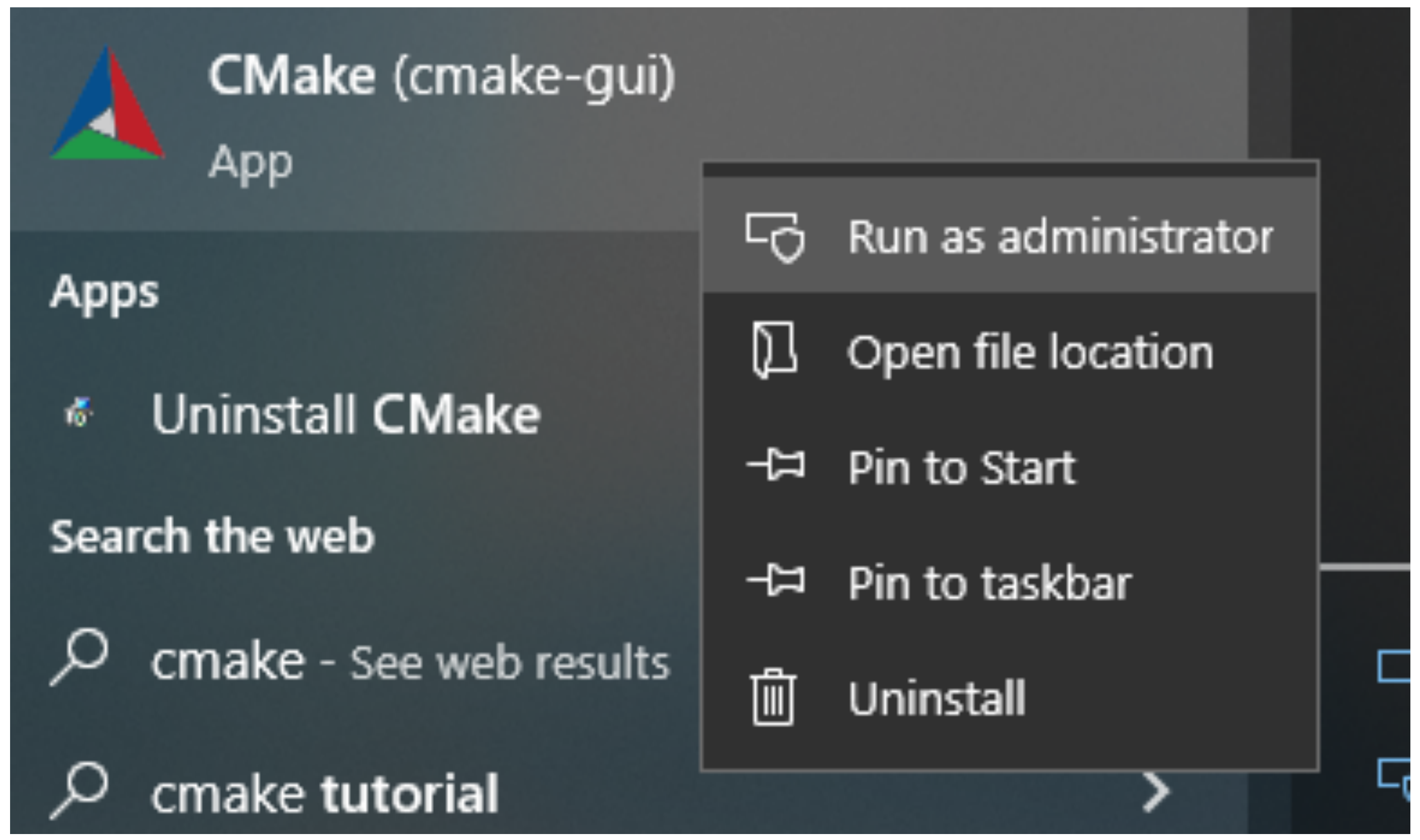
C++ Extension Pack installed

Open the folder where you checked out the repository



CMake Build

We're using the CMake GUI to keep things easy! Windows users, remember to run as administrator. This allows us to write files to locations needing admin access.



CMake Build

We'll use the cmake GUI for this.

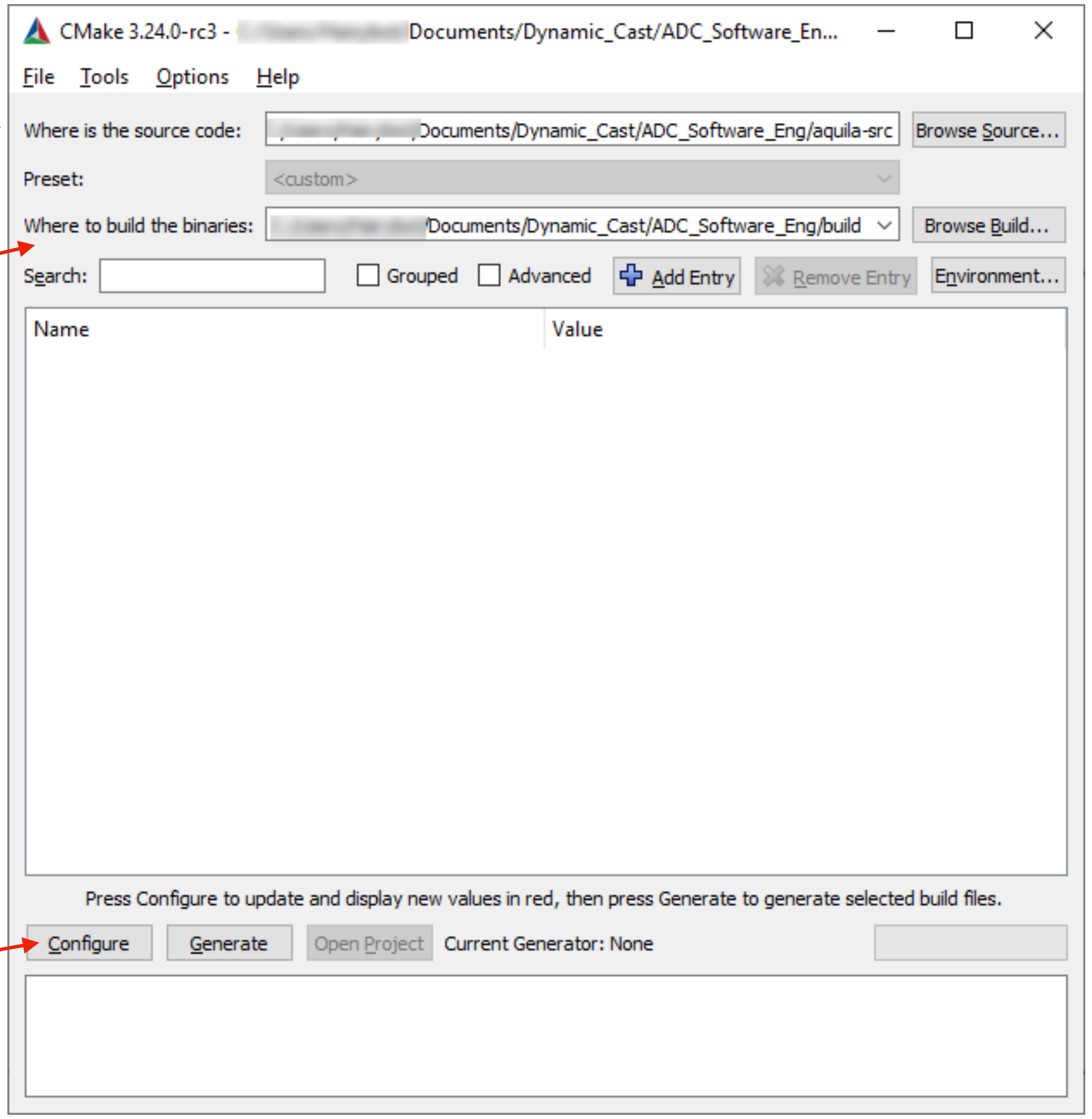
Point Cmake to the source code location

Create a new folder for the build location

Your folder Hierarchy should look like:

workshop-dir/
|_build
|_aquila-src

Click configure in the bottom left



CMake 3.24.0-rc3 - [redacted]Documents/Dynamic_Cast/ADC_Software_En...

Where is the source code: [redacted]Documents/Dynamic_Cast/ADC_Software_Eng/aquila-src

Preset: <custom>

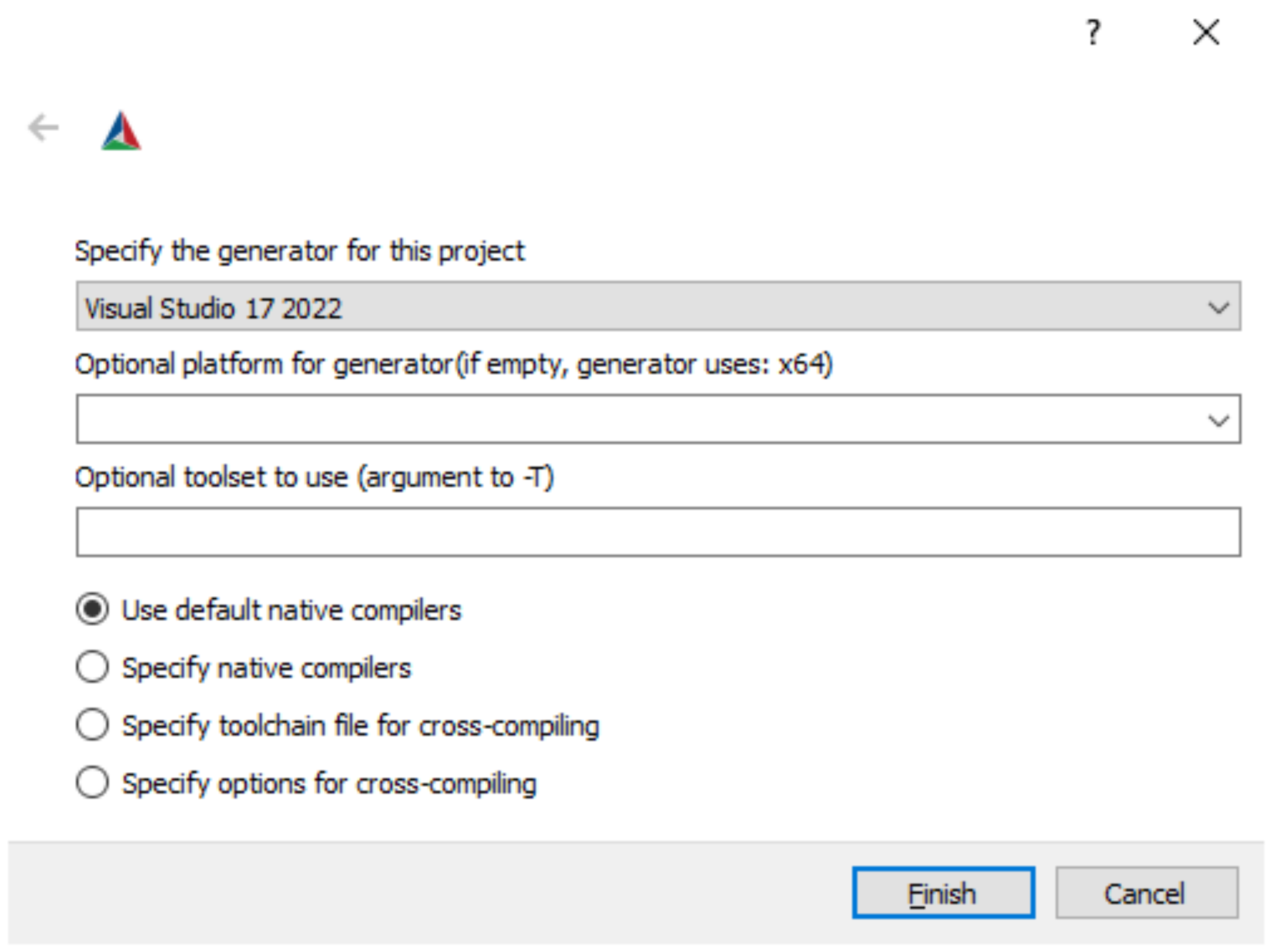
Where to build the binaries: [redacted]Documents/Dynamic_Cast/ADC_Software_Eng/build

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: None

CMake Build

You'll need to specify your generator for your project



Specify the generator for this project

Visual Studio 17 2022

Optional platform for generator(if empty, generator uses: x64)

Optional toolset to use (argument to -T)

[✓ Use default native compilers]

Specify native compilers

Specify toolchain file for cross-compiling

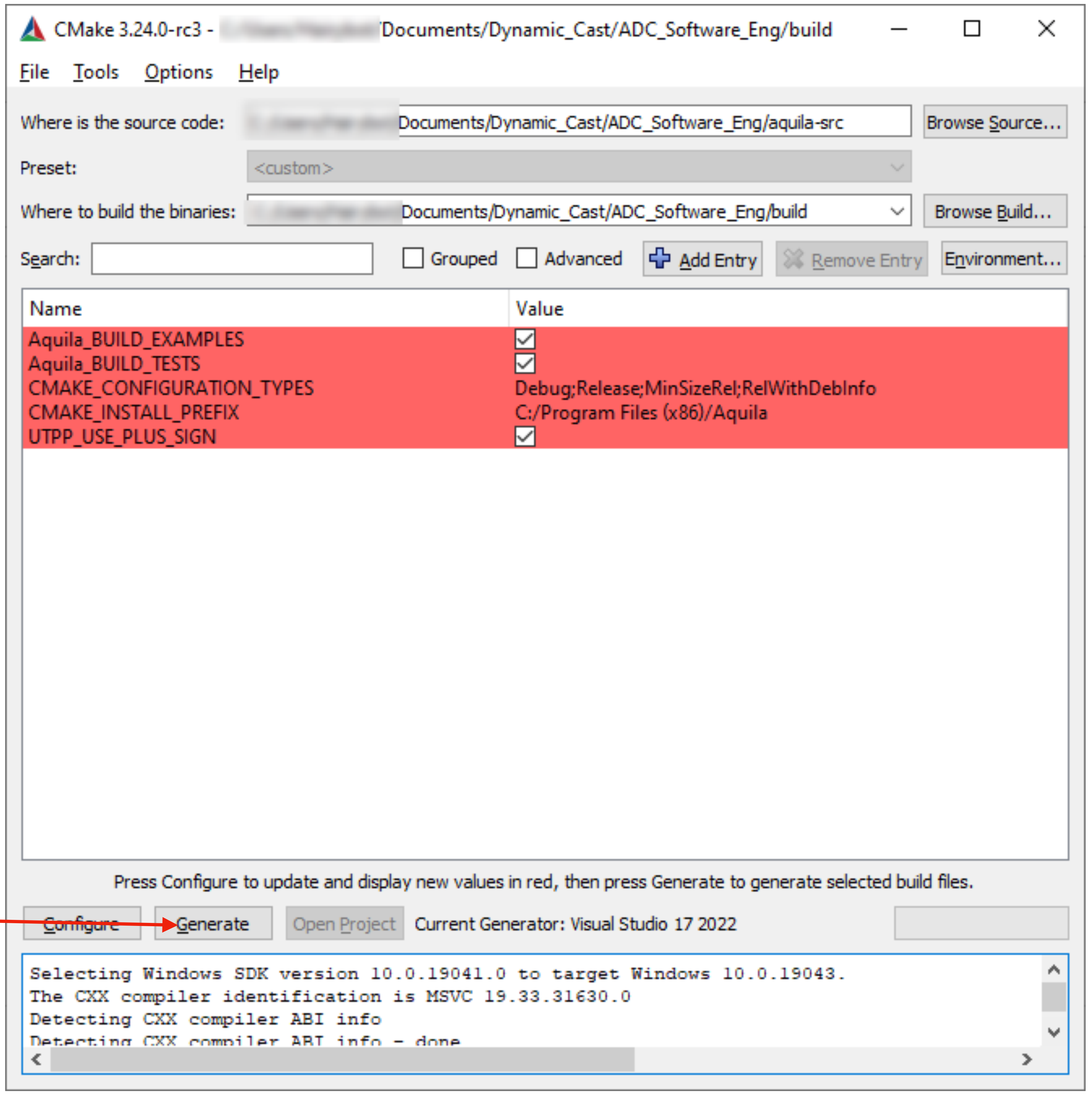
Specify options for cross-compiling

CMake Build

You'll see this:

Settings can be changed here, including install preferences

Click generate



Where is the source code: [redacted]Documents/Dynamic_Cast/ADC_Software_Eng/aquila-src

Preset: <custom>

Where to build the binaries: [redacted]Documents/Dynamic_Cast/ADC_Software_Eng/build

Search:

Name	Value
Aquila_BUILD_EXAMPLES	✓
Aquila_BUILD_TESTS	✓
CMAKE_CONFIGURATION_TYPES	Debug;Release;MinSizeRel;RelWithDebInfo
CMAKE_INSTALL_PREFIX	C:/Program Files (x86)/Aquila
UTPP_USE_PLUS_SIGN	✓

Press Configure to update and display new values in red, then press Generate to generate selected build files.

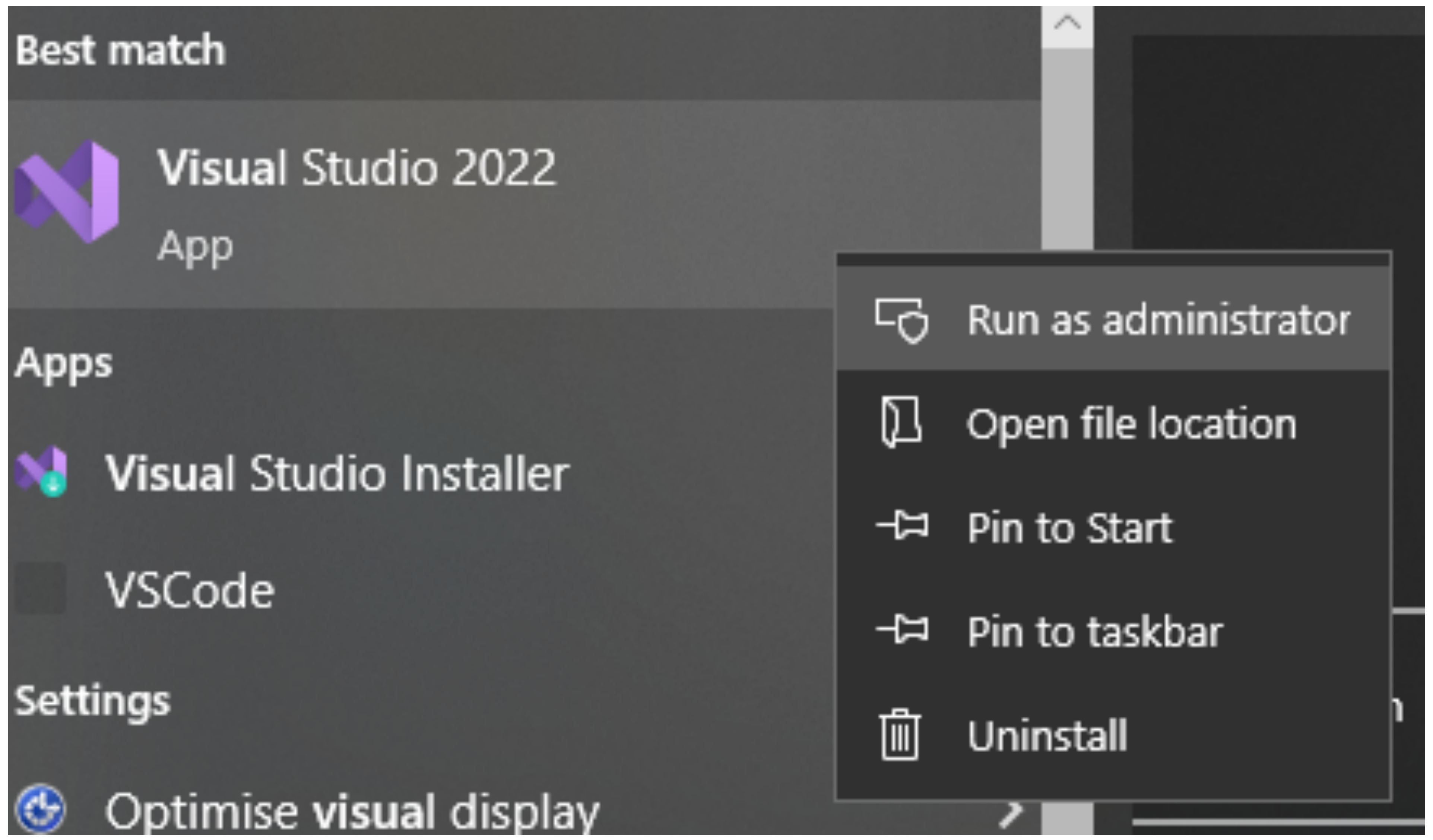
Current Generator: Visual Studio 17 2022

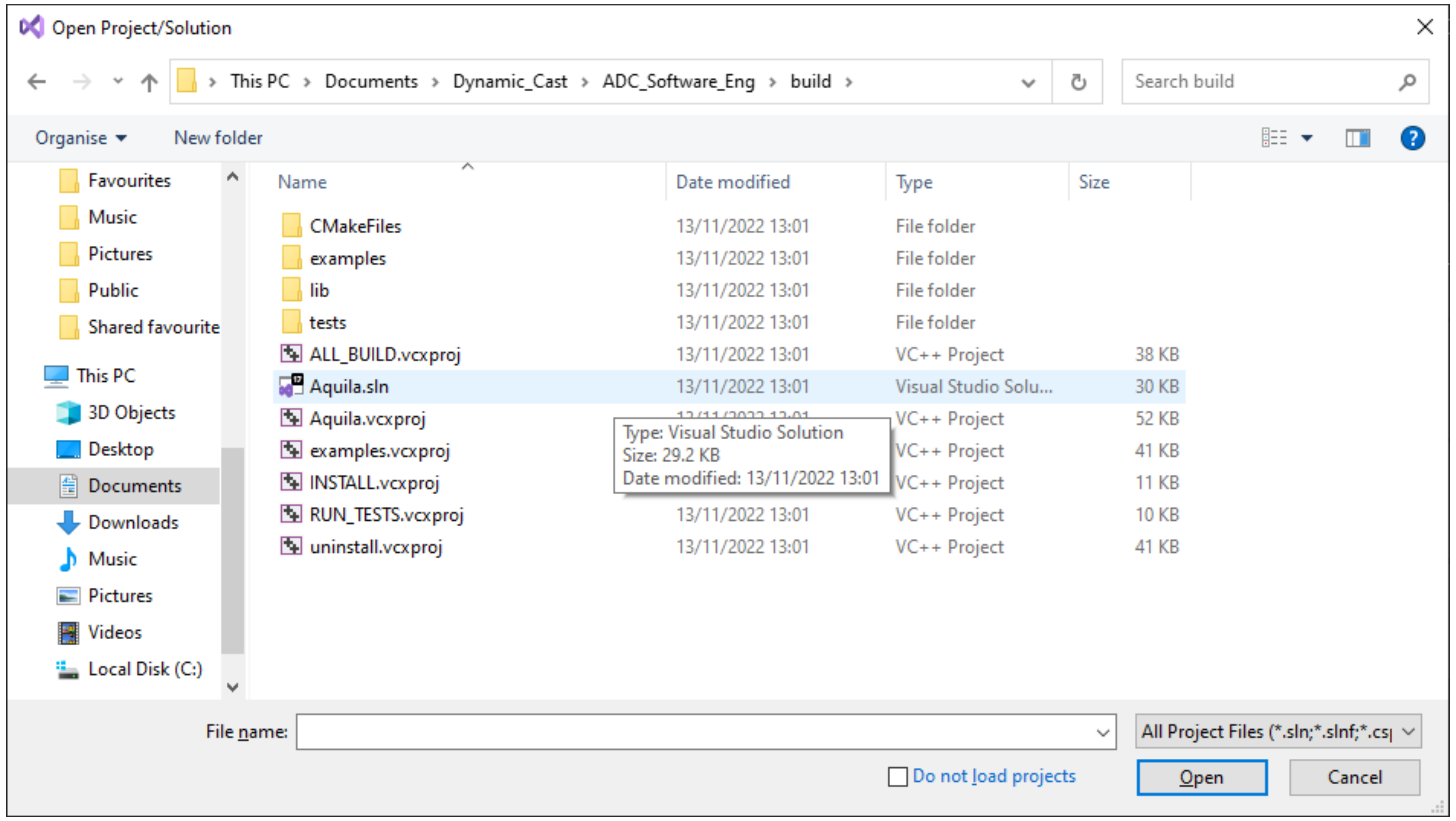
```
Selecting Windows SDK version 10.0.19041.0 to target Windows 10.0.19043.
The CXX compiler identification is MSVC 19.33.31630.0
Detecting CXX compiler ABI info
Detecting CXX compiler ABI info - done
```

Build system

You'll see the build folder has been populated with items. We now open our IDE for the next step.

Note:- Windows users! Open Visual Studio as an administrator:





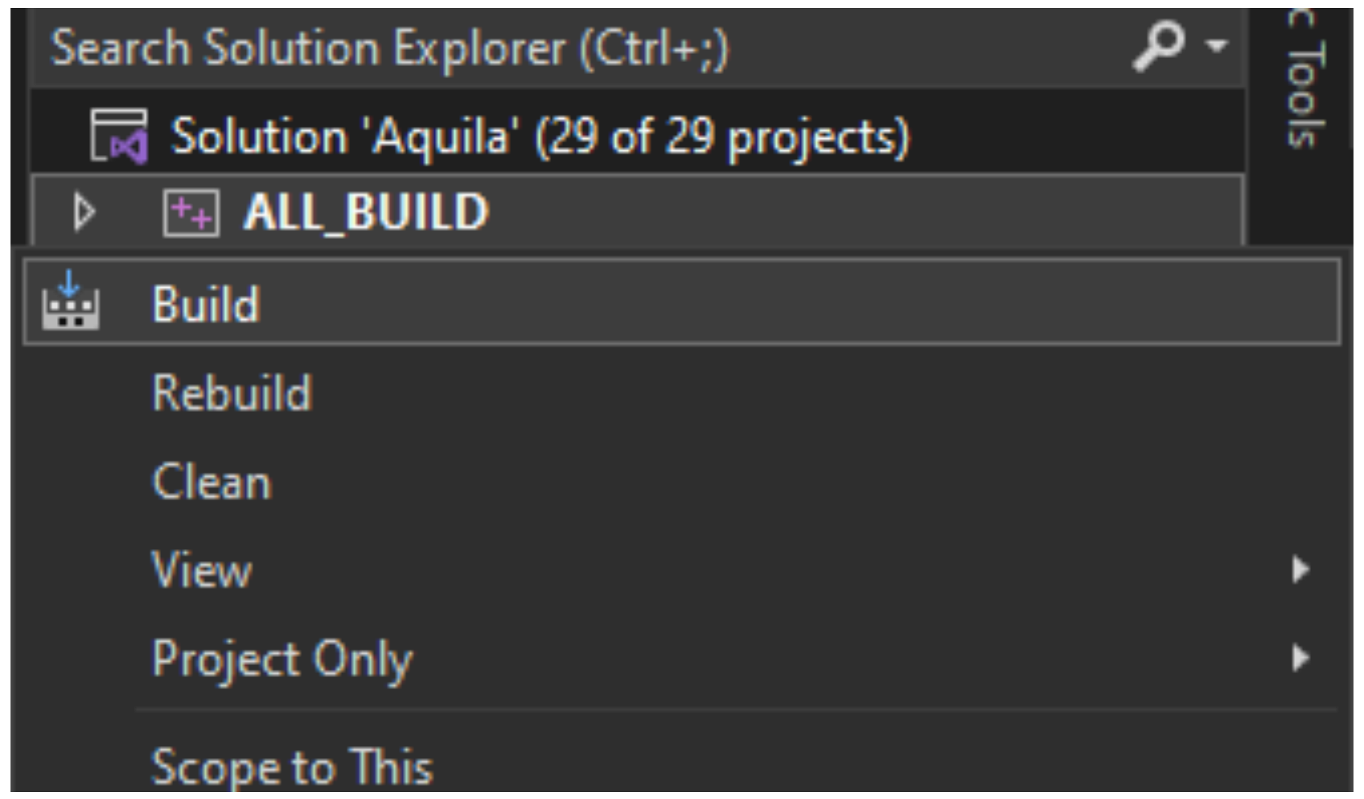
Name	Date modified	Type	Size
CMakeFiles	13/11/2022 13:01	File folder	
examples	13/11/2022 13:01	File folder	
lib	13/11/2022 13:01	File folder	
tests	13/11/2022 13:01	File folder	
ALL_BUILD.vcxproj	13/11/2022 13:01	VC++ Project	38 KB
Aquila.sln	13/11/2022 13:01	Visual Studio Solu...	30 KB
Aquila.vcxproj	13/11/2022 13:01	VC++ Project	52 KB
examples.vcxproj		VC++ Project	41 KB
INSTALL.vcxproj		VC++ Project	11 KB
RUN_TESTS.vcxproj	13/11/2022 13:01	VC++ Project	10 KB
uninstall.vcxproj	13/11/2022 13:01	VC++ Project	41 KB

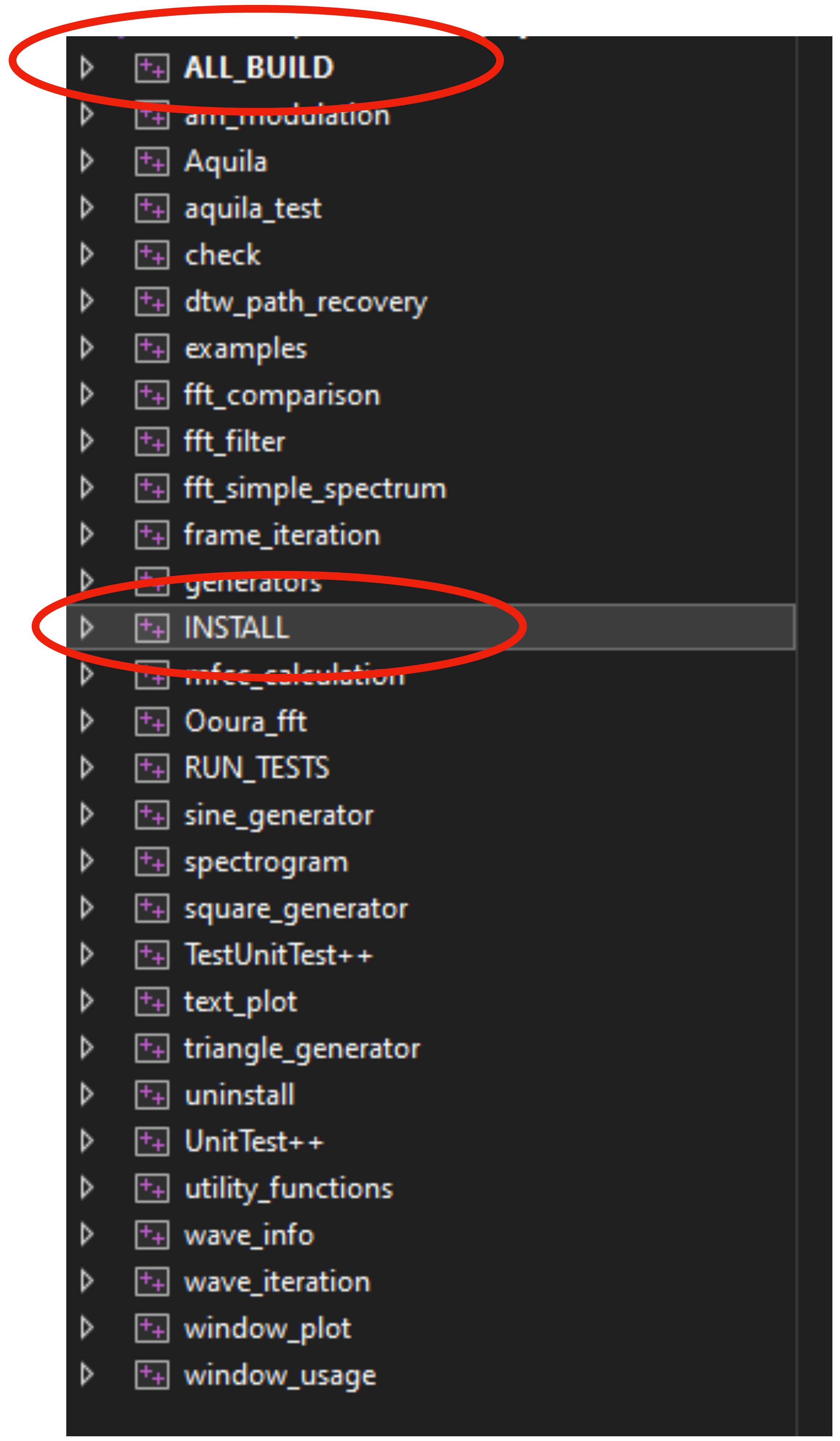
Type: Visual Studio Solution
Size: 29.2 KB
Date modified: 13/11/2022 13:01

Build system

Once we've targeted the project, you can open it with your IDE and perform a debug build.

Right click on the ALL_BUILD and build a solution,
Right click on INSTALL and build


Search Solution Explorer (Ctrl+;)
Solution 'Aquila' (29 of 29 projects)
ALL_BUILD
Build
Rebuild
Clean
View
Project Only
Scope to This



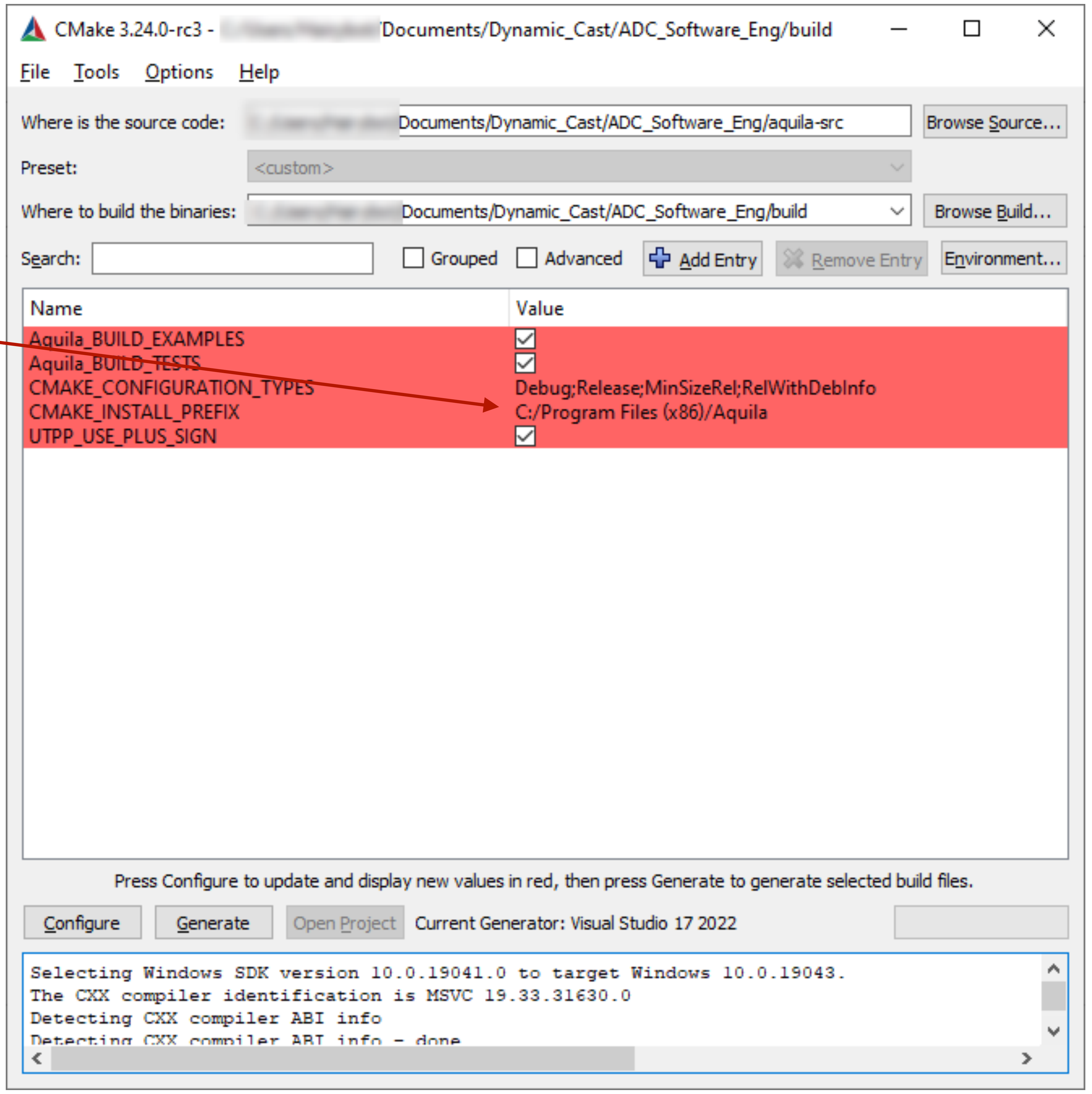
Successful build

Once built, navigate to your chosen install location. This was set in CMake.

You'll find:

- Lib (for library files)
- Include (Containing header files)

We can now move to understand this project further



CMake 3.24.0-rc3 - [redacted]Documents/Dynamic_Cast/ADC_Software_Eng/build

Where is the source code: [redacted]Documents/Dynamic_Cast/ADC_Software_Eng/aquila-src

Preset: <custom>

Where to build the binaries: [redacted]Documents/Dynamic_Cast/ADC_Software_Eng/build

Search:

Name	Value
Aquila_BUILD_EXAMPLES	✓
Aquila_BUILD_TESTS	✓
CMAKE_CONFIGURATION_TYPES	Debug;Release;MinSizeRel;RelWithDebInfo
CMAKE_INSTALL_PREFIX	C:/Program Files (x86)/Aquila
UTPP_USE_PLUS_SIGN	✓

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 17 2022

```
Selecting Windows SDK version 10.0.19041.0 to target Windows 10.0.19043.
The CXX compiler identification is MSVC 19.33.31630.0
Detecting CXX compiler ABI info
Detecting CXX compiler ABI info - done
```

CONTRIBUTE

Contribute

READ

WRITE

DELETE

Deleting code

https://github.com/huggingface/diffusers/pull/218/commits/9583ab730e4bcb948c920a15832f3f7027b76d78

Contribute

Contributing starts with comprehending

READ
TO UNDERSTAND

almost 60% of time*

WRITE

DELETE

* *"on average developers spend ~58 percent of their time on program comprehension activities";*
Measuring Program Comprehension: A Large-Scale Field Study with Professionals (Xia et al., 2017)

Contribute

Contributing starts with comprehending

The ability to read code is a prerequisite to contributing code

READ
TO UNDERSTAND

WRITE

DELETE

Contribute

Contributing starts with comprehending

The ability to read code is a <u>prerequisite</u> to contributing code

Contrary to common advice: code more in order to get better at programming

READ
TO UNDERSTAND

WRITE

DELETE

Contribute

Contributing starts with comprehending

The ability to read code is a prerequisite to contributing code

Contrary to common advice: code more in order to get better at programming

It gets easier as we get more familiar with the concepts* used in the codebase

READ
TO UNDERSTAND

WRITE

DELETE

* *"Code can be read in different dimensions: structure, domain, concepts, context, and collaboration."*
Code Reading in Practice, Felienne Hermans

Contribute

Contributing starts with comprehending

The ability to read code is a prerequisite to contributing code

Contrary to common advice: code more in order to get better at programming

It gets easier as we get more familiar with the concepts* used in the codebase

Practice reading and reasoning about code often

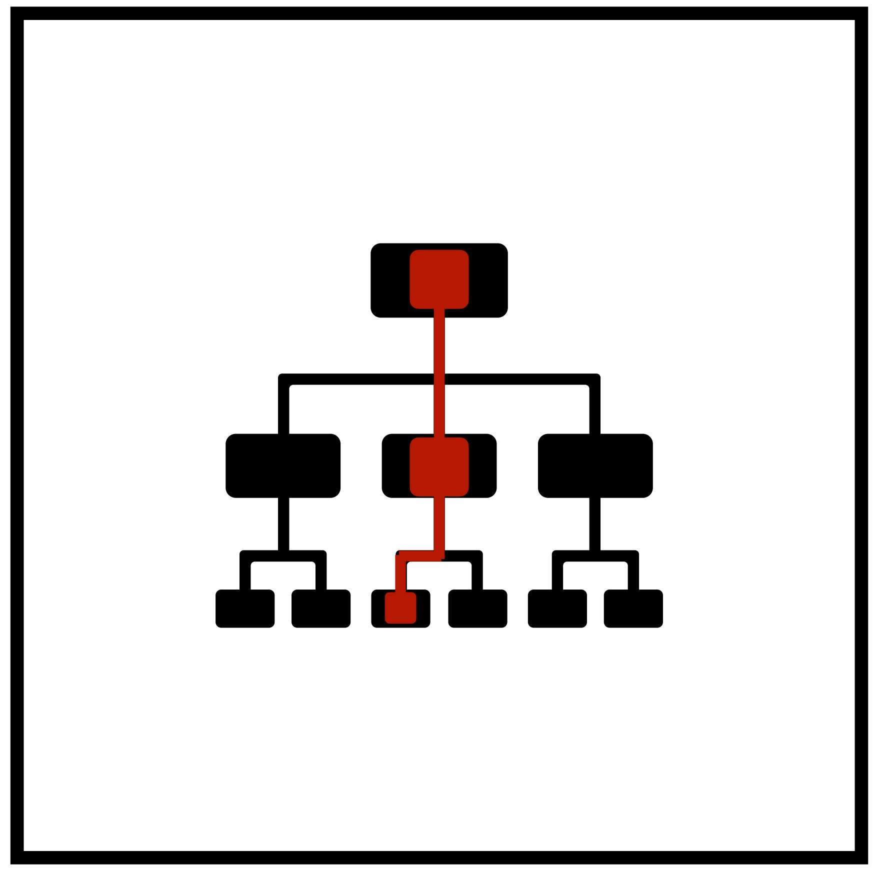
READ
TO UNDERSTAND

WRITE

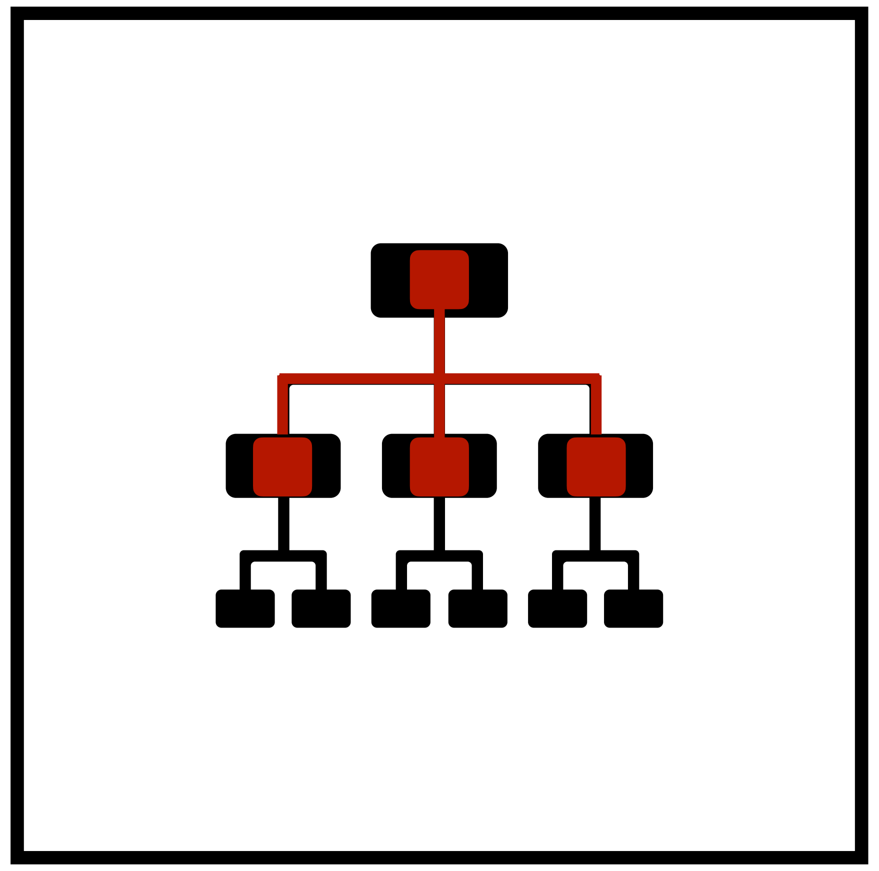
DELETE

* *"Code can be read in different dimensions: structure, domain, concepts, context, and collaboration."*
Code Reading in Practice, Felienne Hermans

Deliberate reading



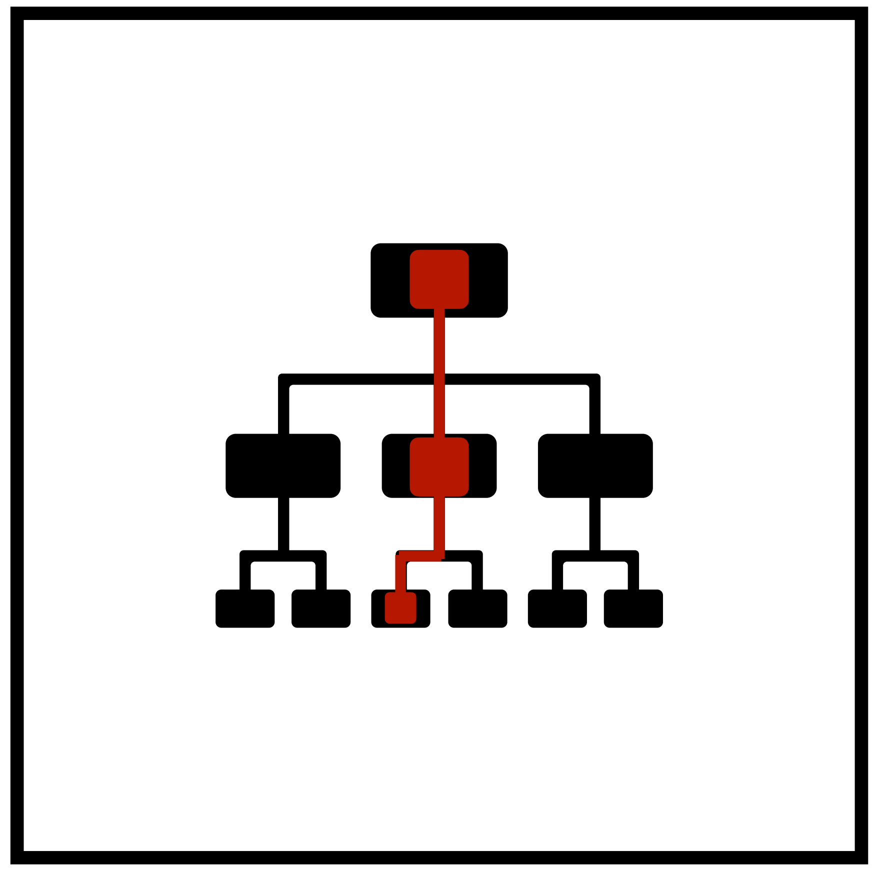
DEPTH FIRST



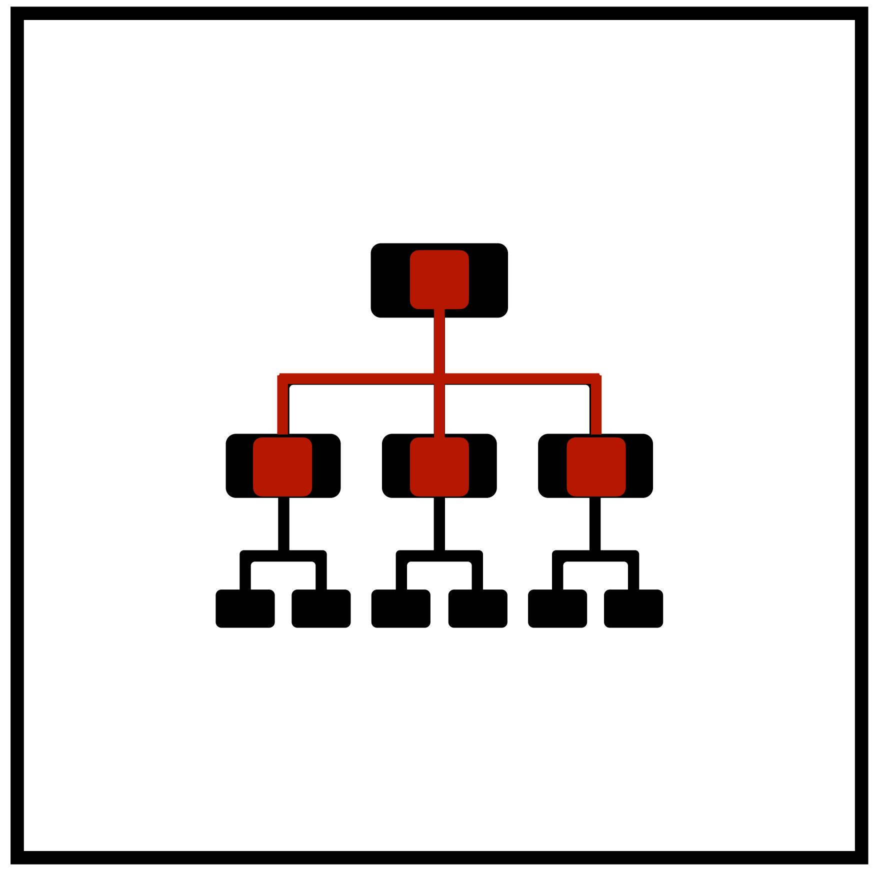
BREADTH FIRST

Deliberate reading

Narrowed focus



DEPTH FIRST

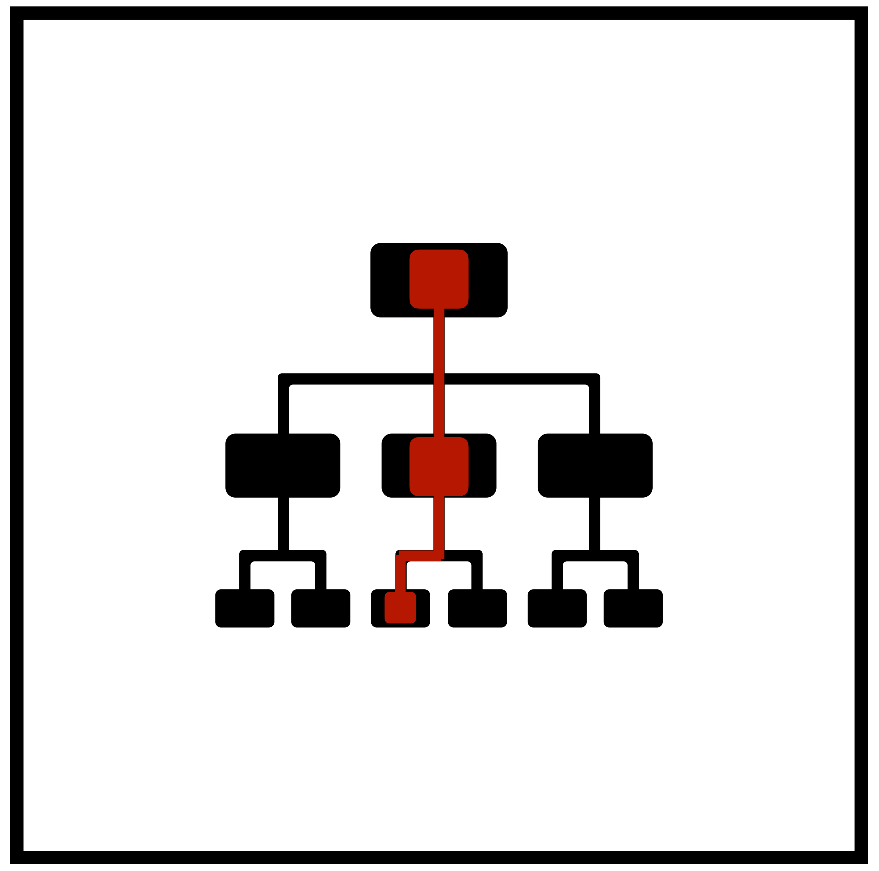


BREADTH FIRST

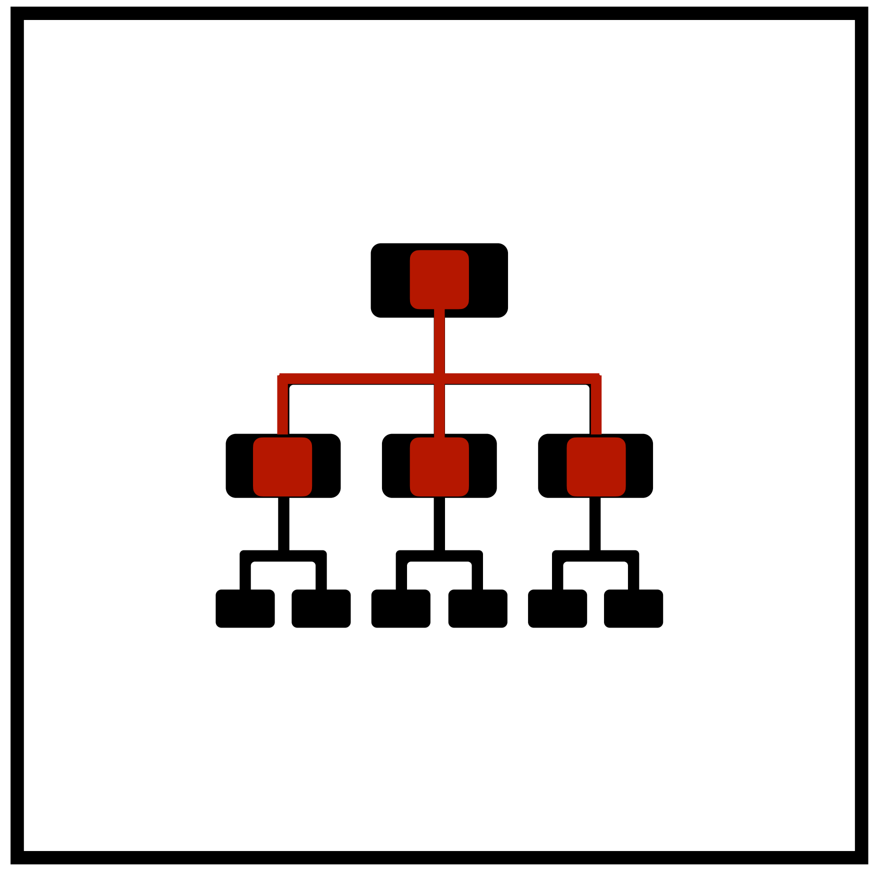
Deliberate reading

Narrowed focus

No need to understand the whole system

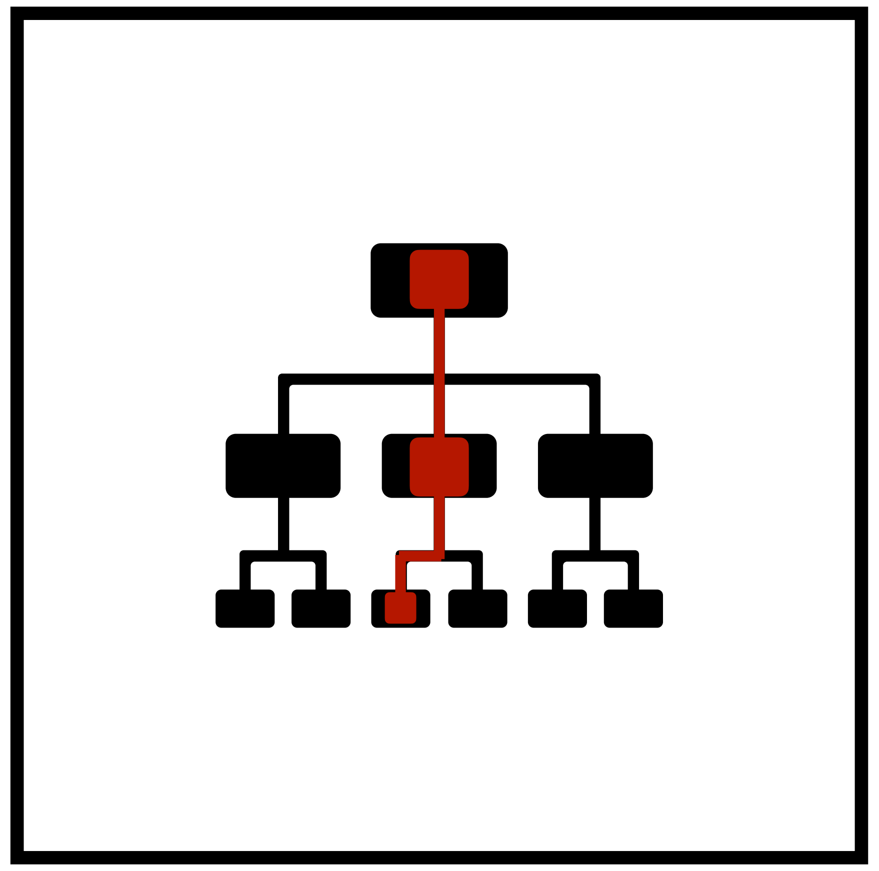


DEPTH FIRST



BREADTH FIRST

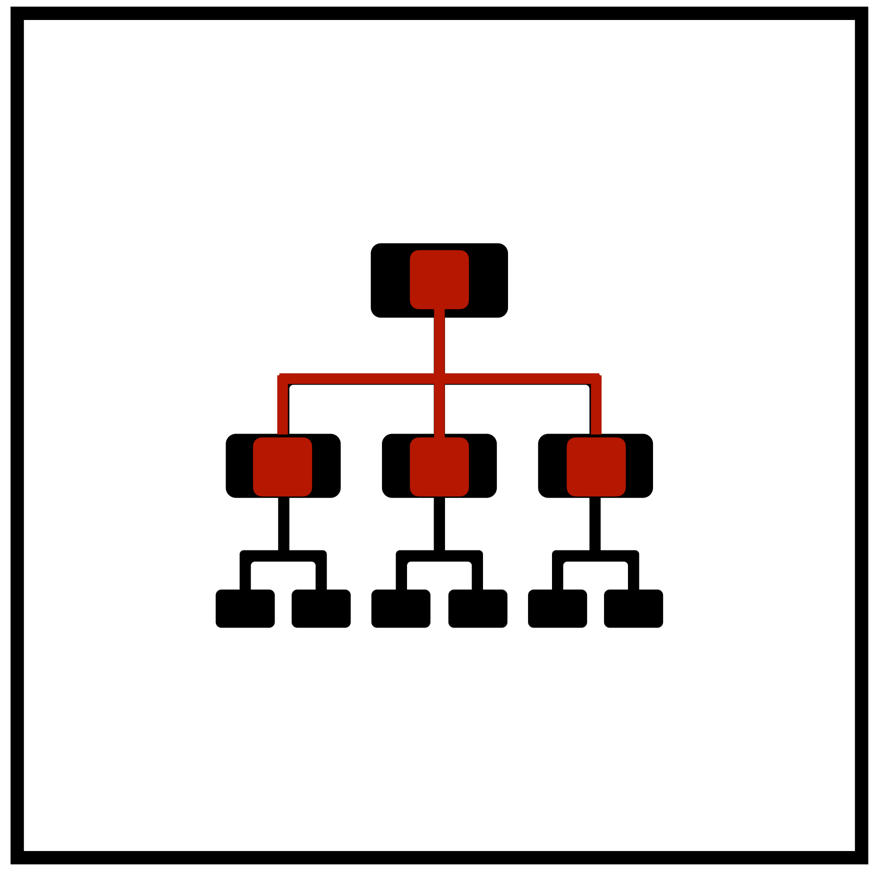
Deliberate reading



DEPTH FIRST

Narrowed focus

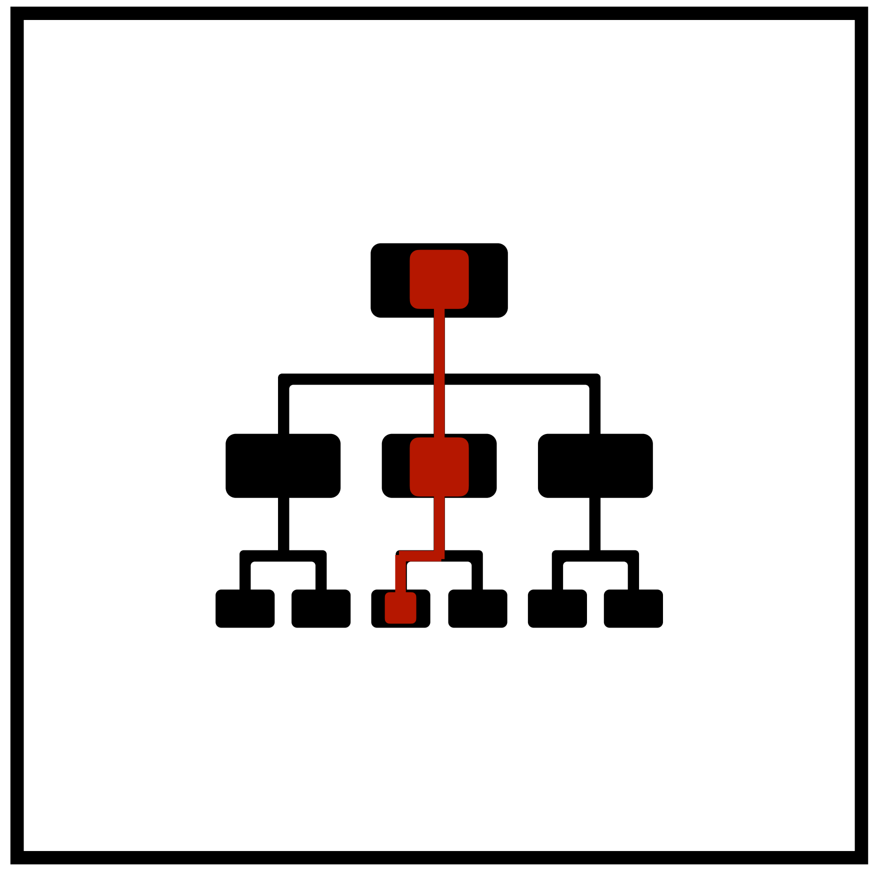
No need to understand the whole system



BREADTH FIRST

Focus on higher-level components

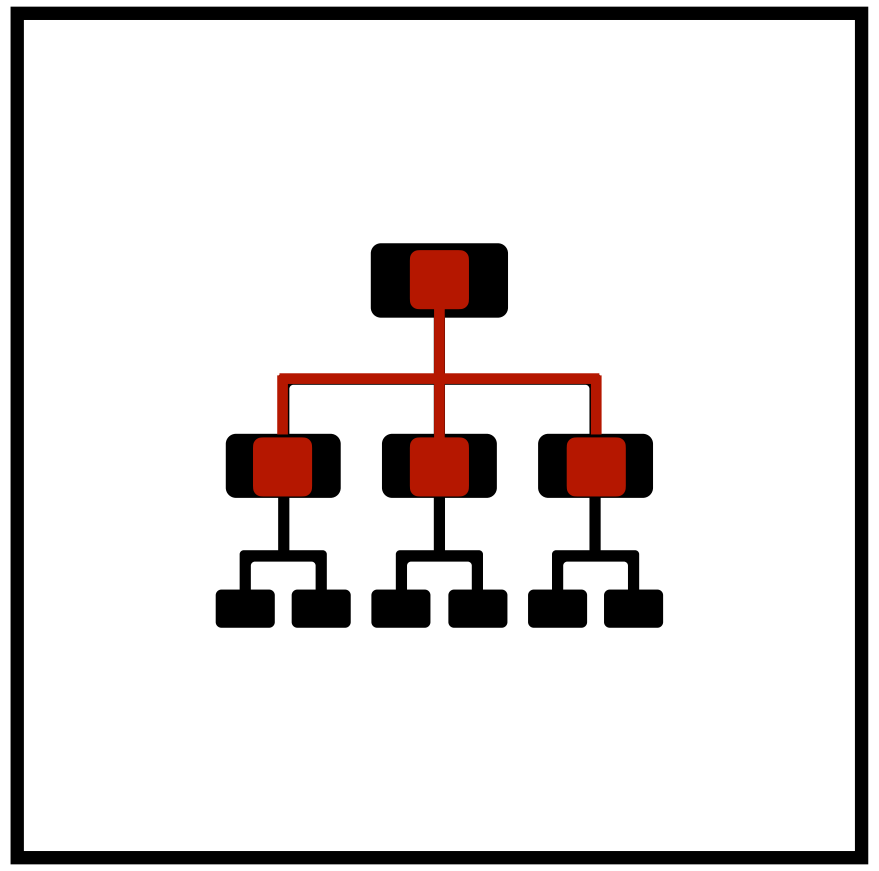
Deliberate reading



DEPTH FIRST

Narrowed focus

No need to understand the whole system



BREADTH FIRST

Focus on higher-level components

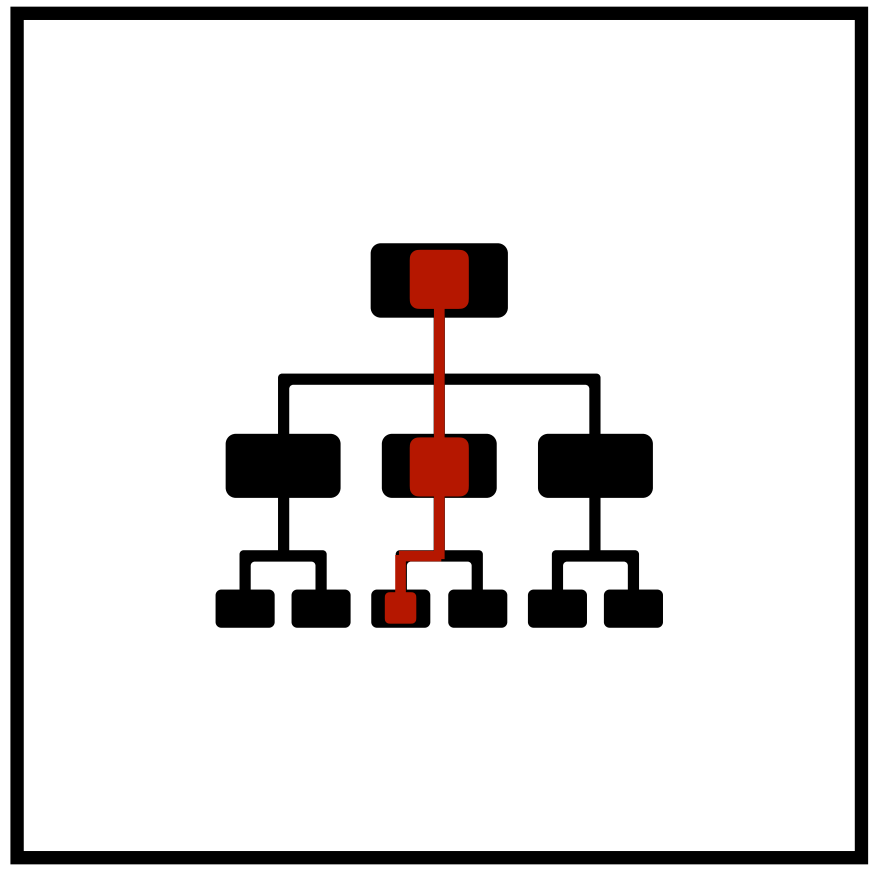
No need to understand all the details

Deliberate reading

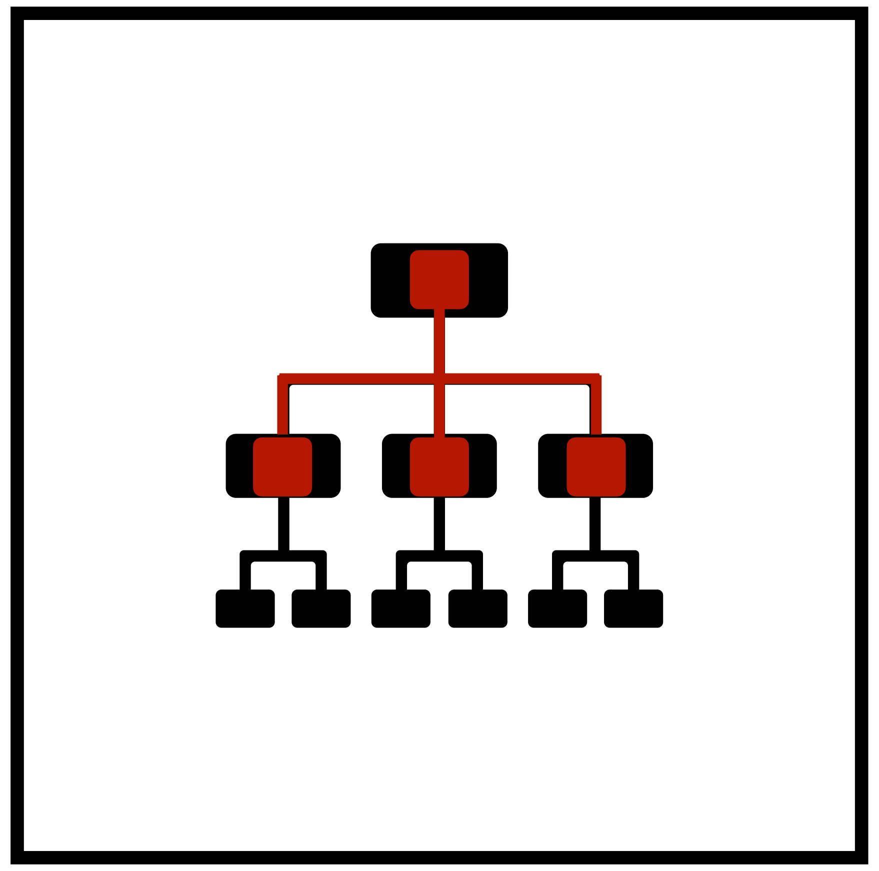
Understanding what causes a particular problem

Narrowed focus

No need to understand the whole system



DEPTH FIRST



BREADTH FIRST

Focus on higher-level components

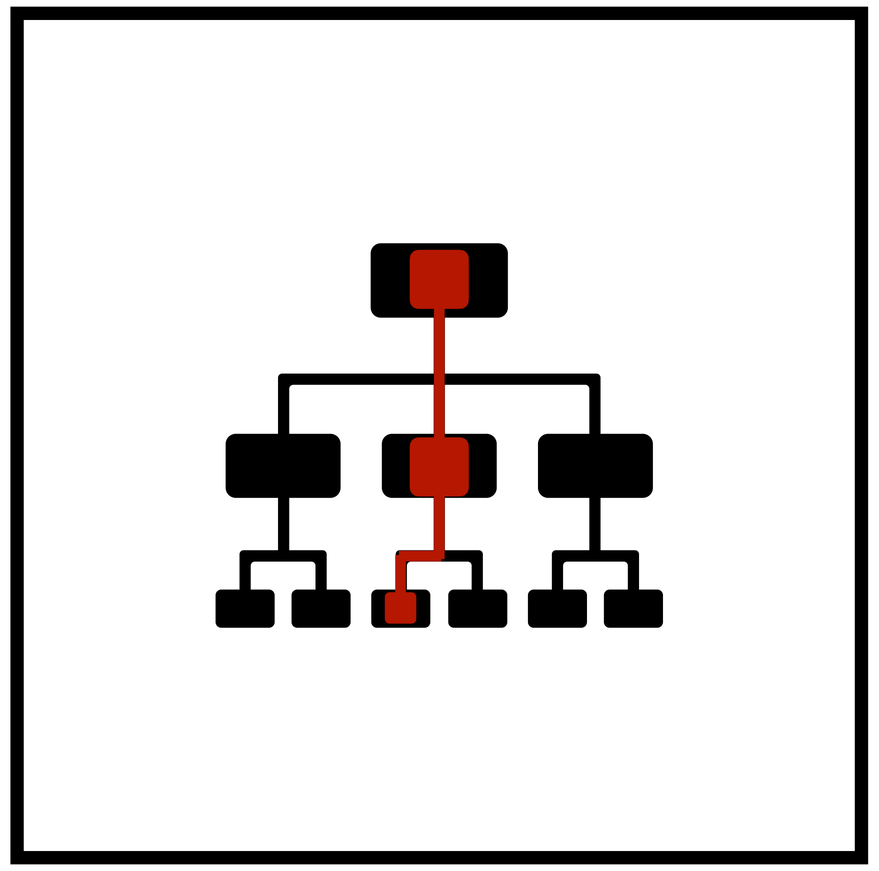
No need to understand all the details

Deliberate reading

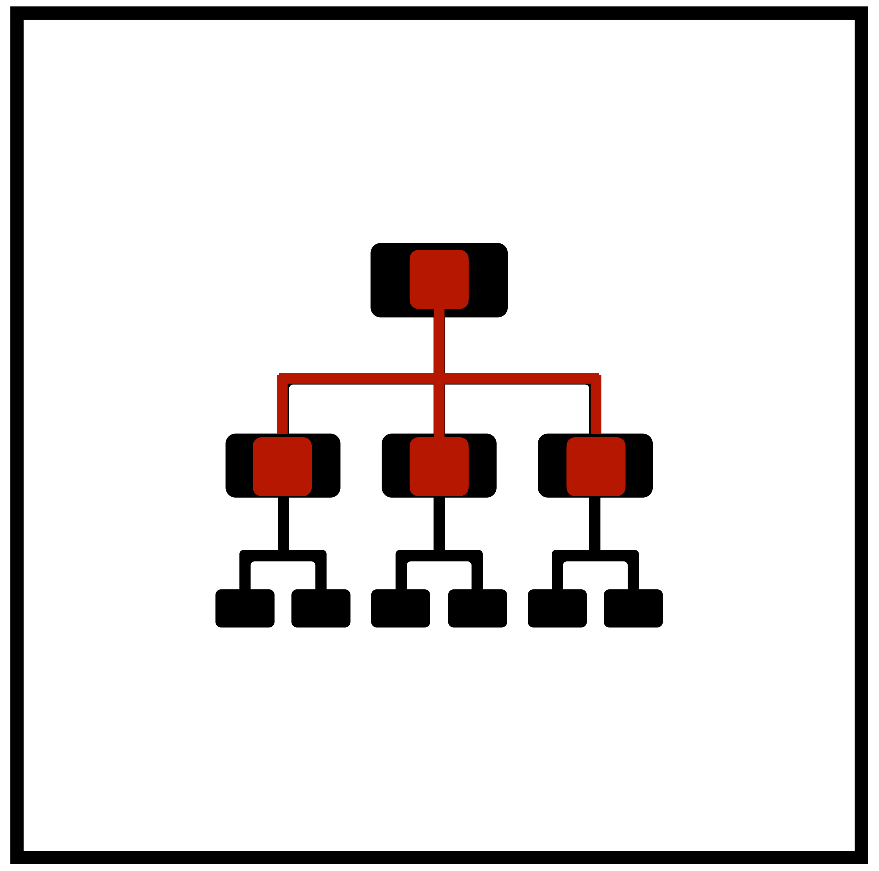
Understanding what causes a particular problem

Narrowed focus

No need to understand the whole system



DEPTH FIRST



BREADTH FIRST

Trying to integrate a new component with the existing system

Focus on higher-level components

No need to understand all the details

Reading code in preparation for the task

FIX

INTEGRATE

MODIFY

Debug an existing problem

https://github.com/zsiciarz/aquila/issues/55

1. check out the branch `debugging`
2. build the `workshop` target
3. run the `workshop` example

   pass `tone.wav` from the workshop folder as an argument

Debug an existing problem

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "(msvc) Launch",
            "request": "launch",
            "type": "cppvsdbg",
            "program": "${command:cmake.launchTargetPath}",
            "args": ["examples/workshop/tone.wav"],
            "stopAtEntry": false,
            "cwd": "${workspaceFolder}",
            "environment": [],
            "externalConsole": false,
        },
        {
            "name": "(lldb) Launch",
            "type": "cppdbg",
            "request": "launch",
            "program": "${command:cmake.launchTargetPath}",
            "args": ["examples/workshop/tone.wav"],
            "stopAtEntry": false,
            "cwd": "${workspaceFolder}",
            "environment": [],
            "externalConsole": false,
            "MIMode": "lldb",
        },
    ]
}
```

Debug an existing problem

1. Reproduce the problem
2. Find meaningful places to put breakpoint in
3. Have a conversation with the debugger / other tools

Debug an existing problem

Don't try to keep everything in the working memory

Use tools that support distributed cognition (eg. pen and paper); they extend the working memory

What is there to hold on to?

Higher level: names, concepts, structure, git history

Lower level: values, relations

Don't read the code linearly, follow the call stack

Aim to understand the relations between values

Testing

Example types of tests

UNIT

INTEGRATION

ACCEPTANCE

Testing frameworks

GoogleTest (https://github.com/google/googletest)

Catch2 (https://github.com/catchorg/Catch2)

UnitTest++ (https://github.com/unittest-cpp/unittest-cpp)

UnitTest++

https://github.com/unittest-cpp/unittest-cpp/wiki/Macro-and-Parameter-Reference

Testing

Setup

Action

Check

Modifying

Move the responsibility to assure the correct data length from the caller to Wave Source

Modifying

Move the responsibility to assure the correct data length from the caller to Wave Source

Keep your focus on what matters at all times; ask yourself often: what am I trying to do? why am I looking here / what am I looking for?

Test Driven Development

Red - introduce a failing test

Green - add necessary changes to make the test pass

Refactor - clean up your solution

Test Driven Development

1. Find a group of matching tests
2. State your end goal in a test
3. Let the process guide you to the solution and end implementation

Test Driven Development

Potential benefits

- Detailed tests serving as documentation and providing entry points to the codebase

- Simpler and cleaner implementation (no premature abstractions, only necessary code) and interfaces

Code design

What are your (or the project's) priorities?

MAINTAINABILITY

PERFORMANCE

Code design

What are your (or the project's) priorities?

Long-lived projects, new contributors joining often

MAINTAINABILITY

PERFORMANCE

Critical real time applications, limited hardware resources

Code design

MAINTAINABILITY

PERFORMANCE

Reducing cognitive load

↓

Complex or unusual solutions overload your short-term memory

READABILITY

SCALABILITY
EXTENSIBILITY

COHERENCE

Meaningful names

Simplicity over quirkiness

"Self-documenting code"

Code design

MAINTAINABILITY

PERFORMANCE

Reducing cognitive load

↓

Complex or unusual solutions overload your short-term memory

READABILITY

Meaningful names

Simplicity over quirkiness

"Self-documenting code"

SCALABILITY
EXTENSIBILITY

Ease of adding / removing code

Loosely / tightly coupled code

Can accept different size of data

COHERENCE

Code design

MAINTAINABILITY

PERFORMANCE

Reducing cognitive load

↓

Complex or unusual solutions overload your short-term memory

READABILITY

Meaningful names

Simplicity over quirkiness

"Self-documenting code"

SCALABILITY EXTENSIBILITY

Ease of adding / removing code

Loosely / tightly coupled code

Can accept different size of data

COHERENCE

Coherent practices

Coherent code style

Code design

MAINTAINABILITY

PERFORMANCE

Try to hide the complexity

If you can't avoid cryptic code, prioritize documentation

Reducing cognitive load

↓

Complex or unusual solutions overload your short-term memory

READABILITY

Meaningful names

Simplicity over quirkiness

"Self-documenting code"

SCALABILITY
EXTENSIBILITY

Ease of adding / removing code

Loosely / tightly coupled code

Can accept different size of data

COHERENCE

Coherent practices

Coherent code style

DEPLOY

Deployment

1. Commit your changes on a branch
2. Create a PR
3. Get a code review and merge your changes!

Thank you