

EECS 498 System Design of a Search Engine

Winter 2026

Lab 3: Basics of Developer Workflow/Productivity

GSI: Christopher Davis
IA: Rickey Dong

Nicole Hamilton
[https://web.eecs.umich.edu/~nham/
nham@umich.edu](https://web.eecs.umich.edu/~nham/nham@umich.edu)

agenda

1. Announcements
2. Upcoming assignments
3. Debugging in VSCode
4. Shared Codebase Patterns

agenda

1. Announcements
2. Upcoming assignments
3. Debugging in VSCode
4. Shared Codebase Patterns

Office Hours

Chris:

Tues/Thurs 12:00-3:00 (1010 DOW)

By appointment (virtually) most other times M-F.

Just email chcdavis@umich.edu with **EECS498W26** in title.

Rickey:

Mon 3:30-4:30 (1637 BBB Table #1)

Wed 2:30-4:00 (1270A Leinweber)

Reminders

- String/Vector and UTF-8 autograder assignments due 1/25, this Sunday

Reminders

- String/Vector and UTF-8 autograder assignments due 1/25, this Sunday
- Group picture also due 1/25
 - Tip: put some effort into this...

Announcements

- Animal mascot voting finished!
- We will have a cartoon design version of it, and hopefully print a t-shirt
- If you have any ideas for the design, let us know!

agenda

1. Announcements
2. Upcoming assignments
3. Debugging in VSCode
4. Shared Codebase Patterns

Upcoming Assignments

- HTML Parser: 2/1
- HashTable: 2/1
- Project Plan: 2/1

New Assignments

- HTML Parser: 2/1
- HashTable: 2/1
- Project Plan: 2/1
- HashBlob/HashFile: 2/8
- Multithreaded File I/O: 2/8

New Assignments

- HTML Parser: 2/1
- HashTable: 2/1
- Project Plan: 2/1
- HashBlob/HashFile: 2/8
- Multithreaded File I/O: 2/8
- Linux GetURL: 2/15

New Assignments

- HTML Parser: 2/1
- HashTable: 2/1
- Project Plan: 2/1
- HashBlob/HashFile: 2/8
- Multithreaded File I/O: 2/8
- Linux GetURL: 2/15
- Linux GetSSL: 2/22
- Linux TinyServer: 2/22

New Assignments

- HTML Parser: 2/1
- HashTable: 2/1
- Project Plan: 2/1
- HashBlob/HashFile: 2/8
- Multithreaded File I/O: 2/8
- Linux GetURL: 2/15
- Linux GetSSL: 2/22
- Linux TinyServer: 2/22
- **Midterm: 2/26**

New Assignments (Tips)

- You'll have to start working on them alongside your project!
 - Plan things out; split work, do them concurrently
 - Even if you don't do some of the assignment, you should still have an understanding of how it works!

New Assignments (Tips)

- You'll have to start working on them alongside your project!
 - Plan things out; split work, do them concurrently
 - Even if you don't do some of the assignment, you should still have an understanding of how it works!
- Speaking of planning things out and working together, I'll cover some of that in a later section

agenda

1. Announcements
2. Upcoming assignments
3. Debugging in VSCode
4. Shared Codebase Patterns

Debugging

agenda

1. Announcements
2. Upcoming assignments
3. Debugging in VSCode
4. **Shared Codebase Patterns**

Foreword

- I'm about to dump out a bunch of stuff
- For brevity, can't go in-depth into all of it
- But, I will post resources covering all of this to EdStem
 - (And at the end for Canvas)
- Feel free to talk or email me if you have any questions, or otherwise want a discussion

Version Control Systems (VCS)

- You need to be using a VCS...
 - Organizes your work
 - Makes collaboration easier
 - Safeguards against AI/cheating if you have reasonable commits
- A couple options: Git, Mercurial, etc.
 - Just use Git
 - Hopefully you've used it for an EECS course already
 - Even if you have, there's still a lot to learn; especially for collaborative projects

Git Refresher

1. Edit `my_work.cpp`
2. `git add my_work.cpp`
3. `git commit -m "This is a commit message"`
4. `git push [remote] [branch]`

Git Refresher

1. Edit `my_work.cpp`
2. `git add my_work.cpp`
3. `git commit -m "This is a commit message"`
4. `git push [remote] [branch]`

Can use the CLI, or a GUI like *Github Desktop*

Git Refresher

The screenshot shows the GitHub web interface for a repository named `gsi-eecs498-scratch`. The current branch is `main`. A commit is being made to the `Utf8Starter/Utf8.cpp` file. The commit message is "Added ReadUtf16" and the description is "Implemented definition of the ReadUtf16 function." The commit is being made to the `main` branch. The file `Utf8.cpp` is shown with a diff view. The diff shows that the file was added with the following code:

```
@@ -230,4 +230,8 @@ Utf8 *WriteUtf8( Utf8 *p, Unicode c, Utf8 *bound = nullptr ) {  
    }  
    return p + len;  
+ }  
+  
+ Unicode ReadUtf16( const Utf16 **p, const Utf16 *bound = nullptr ) {  
+  
+ }
```

The interface includes a "Changes" tab with a filter input and a list of changes. The "History" tab is also visible. The "Commit" button is labeled "Commit 1 file to main". The "Publish repository" button is also visible.

Collaboration

- This is easy for solo projects, but what about collaboration?
- Probably the first time many of you will work on a codebase of >2-3 people (or maybe even >1)
- Effective collaboration is a well-thought out and debated problem
- I will go over some common patterns
- Opinionated, and you should do what you feel works best, but make sure it's a conscious decision to maximize productivity

The Basics (Branching)

Instead of working off of main, use feature branches!

```
git checkout -b <new-branch-name>
```

```
git checkout <existing-branch>
```

The Basics (Branching)

Instead of working off of main, use feature branches!

```
git checkout -b <new-branch-name>
```

```
git checkout <existing-branch>
```

You can work safely in your own isolated workspace!

You don't want to type in the same space as someone in a Google Doc – so don't do it in code!

The Basics (Branching)

Instead of working off of main, use feature branches!

```
git checkout -b <new-branch-name>
```

```
git checkout <existing-branch>
```

You can work safely in your own isolated workspace!

You don't want to type in the same space as someone in a Google Doc – so don't do it in code!

Of course, this introduces other complexities...

Merges

If two people have different branches, then the code they've written could still conflict

- **What to do?**

Merges and PRs

If two people have different branches, then the code they've written could still conflict

- **What to do?**
 - Try and avoid in the first place (more on this later)
 - Pull Requests!

Merges and PRs

If two people have different branches, then the code they've written could still conflict

- What to do?
 - Try and avoid in the first place (more on this later)
 - Pull Requests!
- **Why do pull requests?**

Merges and PRs

If two people have different branches, then the code they've written could still conflict

- What to do?
 - Try and avoid in the first place (more on this later)
 - Pull Requests! (aka PRs or CLs)
- **Why do pull requests?**
 - Have someone else review your code...

Merges and PRs

If two people have different branches, then the code they've written could still conflict

- What to do?
 - Try and avoid in the first place (more on this later)
 - Pull Requests! (aka PRs or CLs)
- **Why do pull requests?**
 - Have someone else review your code...
 - Protect the release version
 - Don't want to have someone break a bunch of stuff in "prod", then have to try and cherry-pick rollbacks...

Merges and PRs

PRs should be *atomic*



Merges and PRs

PRs should be *atomic*, and cover only what's required for the feature.

- I.e., only change one module at a time
- Your bug fixes to the crawler shouldn't be include in a PR to add a new **map** structure
- Why?

Merges and PRs

PRs should be *atomic*, and cover only what's required for the feature.

- I.e., only change one module at a time
- Your bug fixes to the crawler shouldn't be include in a PR to add a new **map** structure
- Why?
 - Easier to roll-back or cherry pick discrete, atomic changes
 - Smaller, discrete changes much easier to review

Merges and PRs

PRs should be *atomic*, and cover only what's required for the feature.

- I.e., only change one module at a time
- Your bug fixes to the crawler shouldn't be include in a PR to add a new **map** structure
- Why?
 - Easier to roll-back or cherry pick discrete, atomic changes
 - Smaller, discrete changes much easier to review

Ideally, your PRs are also working and pass testing, but with greenfield projects it's a bit different, since there's so much setup, as opposed to incremental features or bug fixes.

Collaboration

- Decide on a coding standard/style
 - Can be automated! (More on this later)

Collaboration

- Decide on a coding standard/style
 - Can be automated! (More on this later)
- Review each others code!
 - Can require x number of reviewers to merge into main

Collaboration

- Decide on a coding standard/style
 - Can be automated! (More on this later)
- Review each others code!
 - Can require x number of reviewers to merge into main
- Most importantly...

Collaboration

- Decide on a coding standard/style
 - Can be automated! (More on this later)
- Review each others code!
 - Can require x number of reviewers to merge into main
- Most importantly...
 - Split work up **publically** and fairly
 - Sucks to do duplicate work
 - Track urgency and progress
 - Also sucks to be blocked

Collaboration

- Decide on a coding standard/style
 - Can be automated! (More on this later)
- Review each others code!
 - Can require x number of reviewers to merge into main
- Most importantly...
 - Split work up **publically** and fairly
 - Sucks to do duplicate work
 - Track urgency and progress
 - Also sucks to be blocked
 - Many different “ticket” systems to do this
 - I’ll share one of my favorites

Collaboration (Github)

- Need to host code somewhere everyone can access
 - Many options: Github, Gitlab, etc.
 - Github has a LOT of other QOL dev features...
 - Actions, Github Desktop, Issues, Projects, Orgs
 - So, I recommend Github
 - I'll talk through some of these features as we get to the relevant section

Collaboration (Github)

- Need to host code somewhere everyone can access
 - Many options: Github, Gitlab, etc.
 - Github has a LOT of other QOL dev features...
 - Actions, Github Desktop, Issues, Projects, Orgs
 - So, I recommend Github
 - I'll talk through some of these features as we get to the relevant section
 - First, **Github Issues and Organizations**

Github

- Create an **organization** for your team
 - Allows you to group multiple repos together
- Can also make **projects**, which allow you to track and assign **issues** for repos
 - Github issues / projects are very feature rich, can do quite a lot with them
- Make your issues detailed
 - Are they a bug? Feature request?
 - How urgent is it? How large is it?
 - Who is assigned to it?
 - What do we need to deliver?

Github

DEMO

Repository Patterns

- How should we organize our codebases?

Repository Patterns

- How should we organize our codebases?
- Monorepo
 - Everything all in one repo
 - Why?

Repository Patterns

- How should we organize our codebases?
- Monorepo
 - Everything all in one repo
 - Why?
 - Makes dependency management easier
 - But can have more merge conflicts
 - This is what Google does

Repository Patterns

- How should we organize our codebases?
- Monorepo
 - Everything all in one repo
 - Why?
 - Makes dependency management easier
 - But can have more merge conflicts
 - This is what Google does
- Git submodules / “microservices”
 - Different repo for different components
 - Why?

Repository Patterns

- How should we organize our codebases?
- Monorepo
 - Everything all in one repo
 - Why?
 - Makes dependency management easier
 - But can have more merge conflicts
 - This is what Google does
- Git submodules / “microservices”
 - Different repo for different components
 - Why?
 - Less merge conflicts
 - But, submodule updates are trickier
 - This is what Amazon / AWS does

Build Systems

I'm sure many of you had just been hand-compiling with gcc/clang in the terminal. Or maybe you even had a Makefile.

But now...

Build Systems

I'm sure many of you had just been hand-compiling with gcc/clang in the terminal. Or maybe you even had a Makefile.

But now...

- Lots of different people on your team
 - Different OS, different hardware, (maybe) different compilers
 - Meeting all of these targets with scripts or Makefiles is tricky
 - Also doesn't have a nice way of dealing with huge nested projects and repos

Build Systems

I'm sure many of you had just been hand-compiling with gcc/clang in the terminal. Or maybe you even had a Makefile.

But now...

- Lots of different people on your team
 - Different OS, different hardware, (maybe) different compilers
 - Meeting all of these targets with scripts or Makefiles is tricky
 - Also doesn't have a nice way of dealing with huge nested projects and repos
 - **Build systems can help this**

Build Systems

Think of them as an abstraction above Makefiles.

They transparently support multiple OSes and compilers.

They also intuitively support projects with large directory structures.

Build Systems

- Think of them as an abstraction above Makefiles.
- They transparently support multiple OSes and compilers.
- They also intuitively support projects with large directory structures.

Unfortunately, C++ does not have a canonical build system, unlike, e.g., Cargo for Rust

Build Systems

- Think of them as an abstraction above Makefiles.
- They transparently support multiple OSes and compilers.
- They also intuitively support projects with large directory structures.

Unfortunately, C++ does not have a canonical build system, unlike, e.g., Cargo for Rust

Many different options.

- Bazel (Google's)
- CMake (industry standard)
- Meson, Premake, VS, etc.

Build Systems

Many different options.

- Bazel (Google's)
- CMake (industry standard)
- Meson, Premake, VS, etc.

I recommend CMake or Bazel, as they will have the most documentation, and I can help. Also, good VSCode support.

Unfortunately (especially for CMake), they can be pretty old, esoteric, and uncomfortable. But, with some practice (and references), make development A LOT easier.

Build Systems (Example)

- I will release a build system lab assignment on Canvas next week
 - Can do in either Bazel or CMake
 - Make a toy repo build

Build Systems (Example)

- I will release a build system lab assignment on Canvas next week
 - Can do in either Bazel or CMake
 - Make a toy repo build
- But let's give a quick example so you can see the benefits.

Build Systems (Example: Library)

- I will give an example Library project structure

Unit Testing

- Very important (at least for your library methods)

Unit Testing

- Very important (at least for your library methods)
- Unfortunately, EECS does not prepare you much for the real world...

Unit Testing

- Very important (at least for your library methods)
- Unfortunately, EECS does not prepare you much for the real world...
 - 280/281/etc. use custom scripts or raw CTest
 - In industry, almost everyone uses **Google Test**

Unit Testing (Google Test)

- Open source unit testing framework from Google

```
TEST (TestSuiteName, TestName) {  
    EXPECT_EQ (foo(), true);  
    EXPECT_GE (2, 1);  
}
```

Unit Testing (Google Test)

- Open source unit testing framework from Google

```
TEST (AlgorithmTests, Max) {  
    int i1 = 1, i2 = 2;  
    EXPECT_EQ (core::max (i1, i2), i2);  
    EXPECT_EQ (core::max (-1, -2), -1);  
  
    EXPECT_EQ (core::max (i1, i2, std::greater{}), i1);  
    EXPECT_EQ (core::max (1, 2, std::greater{}), 1);  
  
    EXPECT_EQ (core::max ({1, 2, 3, 4}), 4);  
    EXPECT_EQ (core::max ({1, 2, 3, 4}, std::greater{}), 1);  
}
```

Unit Testing (Google Test)

It's also natively supported by CMake (and CTest)!

Can make the dev workflow pretty nice. Personally, I like GUIs. I'll show you what my setup is.

Unit Testing (Google Test)

DEMO

Microbenchmarks

- What is a microbenchmark? Why do it?

Microbenchmarks

- What is a microbenchmark? Why do it?
 - Benchmark performance of small sample of code
 - Like Unit Tests, but for performance
 - Performance is unintuitive – why?

Microbenchmarks

- What is a microbenchmark? Why do it?
 - Benchmark performance of small sample of code
 - Like Unit Tests, but for performance
 - Performance is unintuitive – why?
- For our search engines, what might we want to test?

Microbenchmarks

- What is a microbenchmark? Why do it?
 - Benchmark performance of small sample of code
 - Like Unit Tests, but for performance
 - Performance is unintuitive – why?
- For our search engines, what might we want to test?
 - Posting list decompression, HTML parsing, etc.

Microbenchmarks

- What is a microbenchmark? Why do it?
 - Benchmark performance of small sample of code
 - Like Unit Tests, but for performance
 - Performance is unintuitive – why?
- For our search engines, what might we want to test?
 - Posting list decompression, HTML parsing, etc.

Industry standard framework is **Google Benchmark**

Microbenchmarks

- What is a microbenchmark? Why do it?
 - Benchmark performance of small sample of code
 - Like Unit Tests, but for performance
 - Performance is unintuitive – why?
- For our search engines, what might we want to test?
 - Posting list decompression, HTML parsing, etc.

Industry standard framework is **Google Benchmark**

More on this in later labs (when we get lower level)...

Miscellaneous

- Will have lots of dev artifacts
 - Builds, `.DS_Store`, `.vscode/`, etc.
 - You probably don't want this in your VCS!!!
- Luckily, can ignore it with a `.gitignore` file
 - Just put in regex for files you want to ignore, and they won't get tracked by git
 - They work recursively, so you can have one in each component of your project
 - E.g., `foo/.gitignore`, `foo/bar/.gitignore`

Miscellaneous (.gitignore)

```
build/*  
.vscode/*  
.DS_Store  
*.dat  
*.idx
```

Example .gitignore file

Miscellaneous (Formatting)

- You can also automate formatting with `.clang-format` and `.clang-tidy` files!
 - You can even require that someone passes them before merging...
 - This way, you don't need to worry about hand-perfecting all your tabs and whitespace – the script does it for you!

For example...

- Can set indent and column width
- Type name cases
 - E.g., pascal case for enums, lower case for local variables

Miscellaneous (Analysis)

- It also does some static analysis!
 - What is static analysis?

Miscellaneous (Analysis)

- It also does some static analysis!
 - What is static analysis?
 - Analyzes source code for bugs without running it

Speaking of analysis...

- Address sanitizers `-fsanitize=address`
- Memory analyzers (valgrind, etc.)

CI/CD (Github)

- **Github Actions:** Github's native CI/CD platform
- Basically, allows scripts to run whenever you update main
- Can run unit tests, integration tests, static analyzers, linters
- Can rollback on failure, send alerts, etc.

“But it builds on my machine!”

- Nice way of verifying cross-platform

Conclusion

Lots of information...

- Don't worry if this is overwhelming
- Just a broad overview to expose you to some of these ideas / terms early
 - Can't really look up something you've never heard of
- Will follow up in more detail in future labs / lab assignments
- Again, feel free to come to OH, email me, or ask on EdStem with any questions

Addendum

References

Google Test: <https://google.github.io/googletest/>

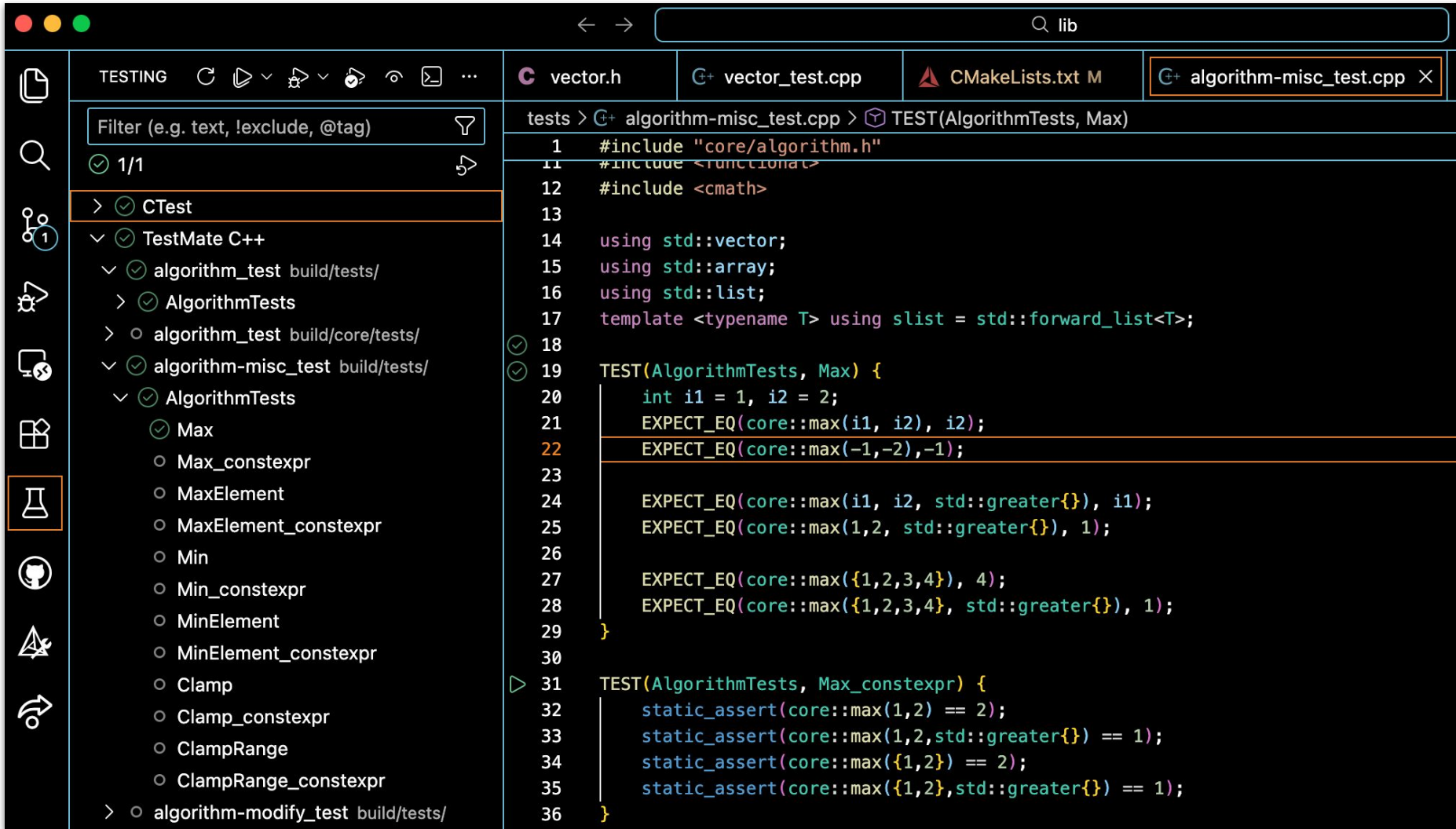
The testing VSCode extension I use is *C++ TestMate*
(matepek.vscode-catch2-test-adapter)

- Very nice testing GUI

CMake (next lab will cover it more)

I use Microsoft's *CMake Tools* VSCode extension.

Google Benchmark: <https://github.com/google/benchmark>



TestMate UI & Google Test Example