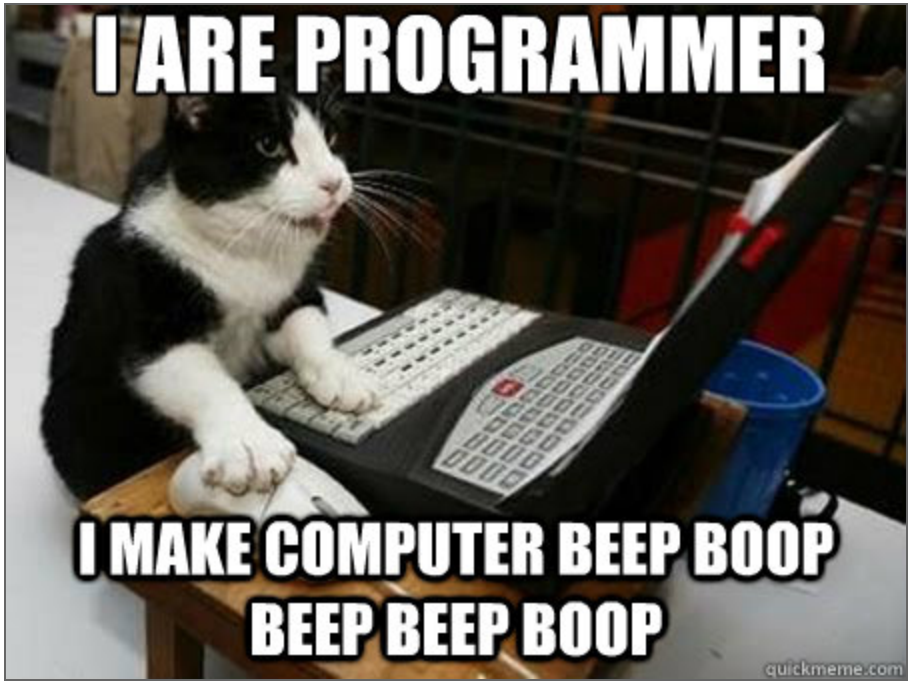
Welcome to INFO6250

Web Development Tools and Methods



Who is this guy?

Brett Ritter `<b.ritter@northeastern.edu>`

- He/Him
- Currently in Seattle
- WebDev since 1995
- Multiple languages, frameworks, platforms
- Both frontend and backend
- Part time instructor since 2017
    - Tell me where to improve my teaching

Not Perfect

I have a truly terrible memory.

Terrible

You have my permission to remind me, and keep reminding me, until something is done or I explicitly say "stop".

Funny

I tell jokes

Fortunately, they are all hilarious and you will laugh

Out loud

Try it now

Get Better

We will keep practicing on that

Accommodations

- This course is challenging, very busy
- Require/benefit from accommodations?
    - Let me know

Falling Behind

- This Course has a lot of work
- Intended to build mental pathways
- Needs time to not just complete
    - Need to make common mistakes
    - Need to build pathways in your brain
- Falling behind is BAD
    - Hard to catch up!
    - Not just a matter of dedication
- Let me know ASAP if it happens

What does "Tools and Methods" mean?

Goal: A foundation in practical Web Development

- How to break down a problem for the web
- Strengths and weaknesses of front/back end
- How to code for changing needs
- How to communicate with future team/self
- How to debug web issues
- Know enough to continue growing

We will USE languages/frameworks

- But we LEARN more general concepts

Northeastern INFO6250 Courses have Differences

This course:

- Includes a bit of front end as well as backend
- NodeJS used, not Java

Other 6250 courses may differ:

- Boston Example: Java/Spring/Hibernate focused

What we will cover

- Web server/client request/response model
- What a webserver does
- Serving static content
- Writing backend generated content
- How the front end works
    - But minimal HTML/CSS/front end JS
- Calling Web services from front end (AJAX)
- Writing RESTful services on backend
- Single Page Applications (SPA)
    - Using "Vanilla" JS and using React
    - Also the benefits/costs of SPAs

Important Topics we don't cover (or not enough)

- Automated Testing
    - Trying to squeeze in
    - Just too much
- Databases
    - Complex topic
    - Not impacted by using for web
    - If you know, this course will be enough
    - If not, this course would never be enough
- Accessibility (a11y)
- Internationalization (i18n)

But I'm not a JS Dev!

This course also for Java/C#/Python/Ruby/etc devs

- Languages/Frameworks have same result
- Languages/Frameworks regularly change
- Front end remains essentially JS-only
    - Defeated all comers so far
    - Plan to avoid JS is risky

What about the Lab class?

- Reflects the additional time for the assignments
  - Seriously, set aside a lot of time!
  - I hate homework too
  - But you learn by doing
- I am not teaching during the lab time

Class is challenging but worthwhile

- We cover a lot of material
- Based on my experiences in the industry
- You must learn to apply new concepts
- A lot of time and work
- Goal is maximum preparation

Cannot teach it all

Too much to cover

- You end empowered to continue your education
- Understand WHY things work
- Options to do things differently
- NOT learning "The one way to do it"

I teach this course so you can learn

- Last few semesters have been rough
    - People relied on ChatGPT
    - Stopped actually using my material
    - Focused on interviews
    - Surprised to be behind by mid-semester
- Do not plan on just finishing the assignments
    - My lessons build on each other
    - Common Feedback:
        - Class is very good prep for industry
        - But learning curve ramps up quickly

Core Class Concepts

The Concepts you'll hear from me repeatedly

1. Programming is Communication
2. More Changes than is New
3. Accessibility is Functionality
4. Complexity is the Enemy
5. Working is Not Enough
6. Avoiding "tutorial hell"

Programming is Communication

All Programmers tell the computer what to do

Most Programmers get the right result

Good Programmers have solutions that are:

- Understood
- Reliable
- Easy to change (successfully)

Most of Programming isn't computer code

- **Programming is Communication**

More Changes than is New

- School assignments usually all new work
- Real tasks almost always changes to existing
    - Rarely "new" work
- Devs won't read every existing line

Work must make it easy to:

- Find relevant spot
- Understand the concepts
- Make changes using those concepts

Accessibility is Functionality

- Accessibility = Content available to all
    - "ability to access"
- Web successful because content is accessible
    - Not just to humans
    - Humans are important, though!
- Web predates smartphones, smart watches, IoT
    - But still works on them
- Improving Accessibility
    - Improves Functionality

"It looks fine to me" isn't how Web changed the world and is still here 30 years later

Complexity is the Enemy

Complexity is a paradox

- To do things requires complexity
- Complexity makes it harder to do things

> *"Everything should be as simple as possible, but not simpler"*

Coding is choosing **where to put the complexity**

We must be aware of our constant enemy, Complexity

Working is Not Enough

Code that runs and gives a correct answer is "working"

- That is not enough
- Is it understood?
- Can it be changed?
- Efficiency?
- What is it dependent on?
- Works with different browsers?
- Works on different devices?
- Accessibility?

Avoiding Tutorial Hell

Tutorial Hell: When new students study tutorials, but find they can't generalize any of their knowledge to solve real problems

I emphasize the opposite - **why** we make choices

I encourage you to wrestle with problems rather than look up a solution

I provide examples and have you practice using those in new ways

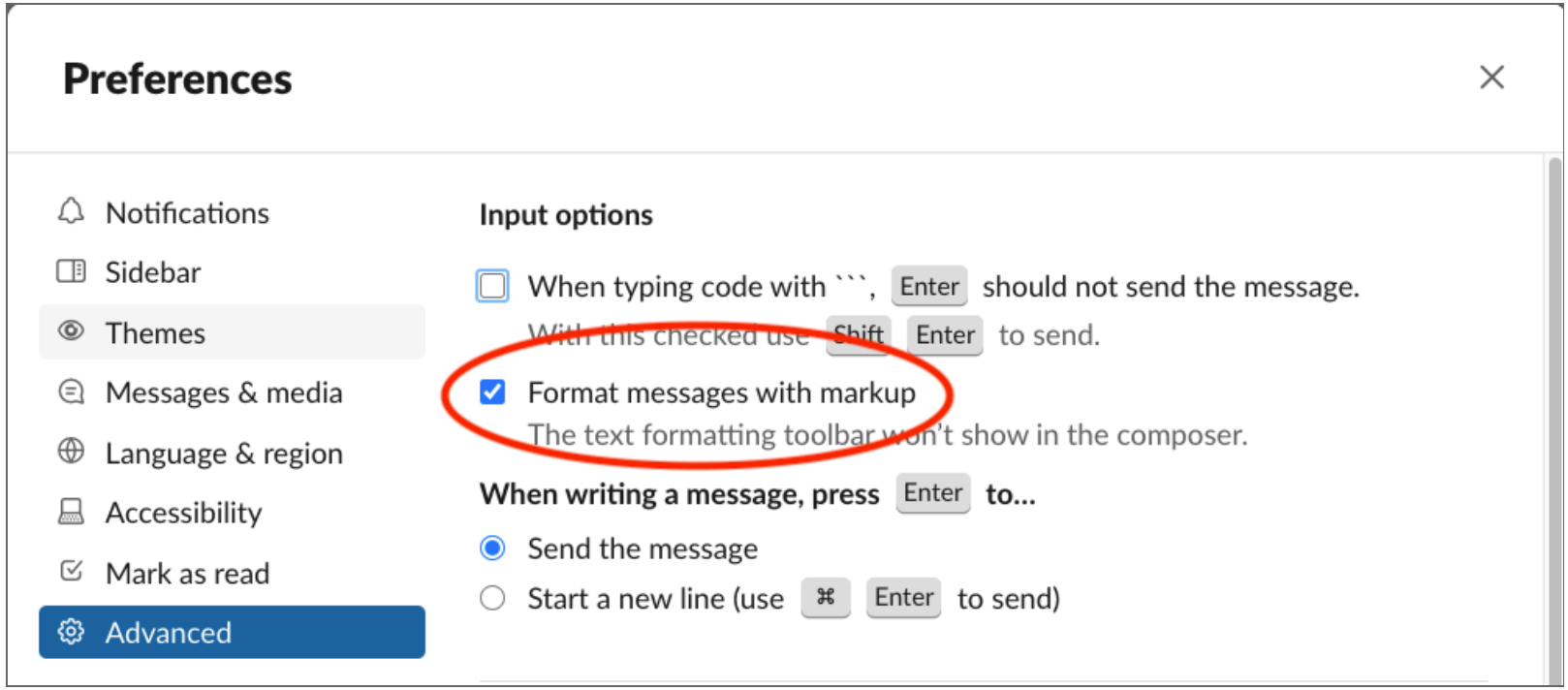
Slack

We communicate via Slack (not Teams)

- **https://rebrand.ly/vtlinfo6250-slack**
- You can email me, BUT
    - slower
    - terrible to talk about code
- Use it web, desktop, or phone
    - Notifications recommended!
- Slack is a useful job skill
    - Managing many messages/channels
    - Searching for past messages
    - Reminders

Configuring Slack for code

Update your Slack Preferences:



Mentioning code in Slack

When sending messages in Slack:

(Notice the *backtick* characters)

`this has *bold*` but `` `this does *not*` ``

Use backticks for a line with code in it.

Code blocks in Slack

Triple backticks ``` before and after your message - multiple lines

Text often better than screenshots

- Can copy and run!
- Can grab individual lines!

If using screenshots

- Use built in screenshot abilities!
- No need for screen selfies!

These are job skills!

Slack is a Job Skill

- You know how to use Chat apps, but…
  - Most coders will be in 10-20 **channels**
  - Search to find previous answers
- You need to tweak your notifications
  - I announce changes to assignments/classes!
  - Find sounds that inform without disruption
- You need to find information posted in the past
- Will need to **join** channels of interest
  - Such as **#articles** or **#funny**

Class Github

We use git and github.com

Each of you get a personal repository

- **https://rebrand.ly/vtlinfo6250-github**

You must have/get a github.com account

- Students new to coding may be unfamiliar
- git is a vital job skill for any programmer
- Devs will encounter github often
    - Even if not at their job

Git vs Github

`git` is the source control/version control system.

- tracks files
- changes to files
- many devs can have repos
- can pass files/changes between many repos

`github.com` provides a central place for repos. It has competitors (example: gitlab.com)

github uses git, git does not require github or a github competitor, though we will use github.

Github flow

See `readings/git/` in your repo

- Make edits in **feature branch based off of main**
- You **push** (send) that branch to github
- You create a **Pull Request (PR)** to **merge** your branch into `main` **on github**
- I/TA **review** and **approve** your request
    - We might request changes first
- I/TA merge your branch into `main` on github
    - On the job you will probably do this step
- You update your local `main` branch

Other git-based flows

Other flows of changes and branches exist

- This one (github flow) is the one we will use

Not all version control systems (VCS) are decentralized the way git is

Github acts as a central point for communications

- Your future job uses Github or similar

Local and Remotes

Local means your computer

When I give notes or assignments:

- I **pull** latest `main` from github to my local copy
- I update my local copy (**add**ing and **commit**ting)
- I **push** my `main` branch to github copy of your repo

You submit the same way:

- You **pull** changes from github to your local copy
- You make changes in a **feature branch**
- You **push** feature branch to github and create **PR**

Key Git Notes

- Always do work in the correct branch
- Always check `git status` before `git commit`
- Always check `git status` before `git push`
- When creating a PR, always check the file list
- Before creating a new feature branch
    - Always switch to `main` and pull latest

If you follow these instructions

- Each assignment is distinct and will not conflict

I will use the Command Line

I use the "command line" (terminal) frequently

- You don't NEED to learn command line
    - Outside of a few commands
    - A lot will be easier if you do
        - Not just this course
- Command Line Interfaces (CLI) are a powerful tool
    - Common in documentation/tutorials
- I don't *teach* the command line itself
    - Just the specific commands I use
- See `readings/the-command-line.md` for more info

You may use the text editor (IDE) of your choice

- VSCode is most common in industry (free)
    - Many tutorials, plugins
    - My recommendation in general
- Other options include Webstorm, IntelliJ
- I use neovim
    - I'm ~~really old~~ awesome
    - Less distraction on screen for you

How the Class works

- Assignment each class (15% total)
- Quiz each class (10% total)

3 Projects (25% each)

- Server-side Project
- Vanilla JS + REST Services Project
- Final Project

Common Grading questions

- No curve
- 90% is A-
- 95% (!) and higher is A (Department scale)

How a Class works

- Lectures, sometimes labs (ungraded)
- Bio break 2/class (10 minute ea)
- Recorded online for review (attempted!)
    - University requires attendance!
- Slides as PDF added to repos
    - Usually before class
- Sometimes samples to repos
- Assignment due night before next class
    - Via github
- Canvas Quiz due night before next class
    - Open notes

How Quizzes work

In Canvas

- Multiple choice
- Covers materials from the class lecture
- NOT timed!
- You are welcome to consult notes, recordings, etc
    - Do NOT copy from other people
    - Do NOT google answers (Can mislead!)
- Due night before next class

Worst Quiz score is ignored for final grade

How Assignments work

- Build from skills shown in class
    - Leave time!
- Due night before next class
- Added to repos under `/work` (see README)
    - Each assignment is a different subdirectory
    - Remember: **branch** NOT the same as **folder**
- Submitted via github **Pull Request**
- TA/I will review and merge
    - May request changes
- No changes unless requested

Worst Assignment score is ignored for final grade

How Projects work

- Like Assignments
    - Pull request, Due date
    - Done at home, not in class
- In repos under `/project1`, `/project2`, `/final`
    - NOT `/work`
- Big chunk of grade each!
    - VERY important to do well
    - DO NOT copy work
        - "referencing" may be copying
    - "Demonstrate Skills from class"
- Minimal outside libraries

How the Final Project works

- Guidelines given
- Full React SPA + REST services
    - Minimal outside libraries
- Potential Showcase for NEU
- Submitted as Pull Request in repo
    - in `/final`
- Limited time! No extensions!
- Chance to raise grade
    - "Demonstration of skills from class"

Instructor Virtual Office Hours

- Mon: 2pm-3pm (ET) / 11am-noon (PT)
- Tue: 2pm-3pm (ET) / 11am-noon (PT)
- Wed: No Office Hours
- Thu: No Office Hours
- Fri: 2pm-3pm (ET) / 11am-noon (PT)
- Other times by appointment
- Available on Slack for quick questions

Other Details

- No video requirement
- No breakout rooms
- Canvas for grades and quizzes only
- Slack preferred over Email
- TA Office Hours TBD

Important Details

- Request Assignment Extensions
    - No excuse required, just ask
    - No grade penalty if granted
    - Request BEFORE day it is due!
        - Job skill
- Do not copy/generate work you submit
- Demonstrate skills from this course
    - Not just working code

Final Projects: NO extensions

- Except unquestionable emergencies

Primary Sources/Tools

- **MDN - https://developer.mozilla.org**
  - Closest the Web has to user documentation
- **caniuse.com**
  - Works in which browsers and since when?
- Browser DevTools (we use Chrome)
  - Used daily by all webdevs
  - Students tend to neglect
  - In whichever browser, tools are similar
  - Firefox often preferred by Devs

A Unique Warning about the Web

You WILL be expected to use online sources

BUT a **lot** of info about web tech is outdated

- Many updates in possible + best practices

Don't use any sources older than 3 years

Or **extra work** and **wrong results**

- One reason ChatGPT often gives poor advice
- "Works" is not the same as "Good"
    - Esp. for Web (diverse browsers/systems)

"Meh" Sources

- W3 Schools
    - Prefer MDN to learn HOW, not just one way
- StackOverflow
    - Some great answers
    - But often out of date
    - Older answers dealt with Internet Explorer
    - Many newer options much easier

These "meh" sources may give you good answers

- But you need to confirm with better sources

LLMs aren't good sources for now

ChatGPT, Claude, Gemini, Copilot, etc

- Can have great answers
- Can have awful answers
- How can you tell which is which?
- "Works on my machine" is just a sliver of the web
- "Sounds reasonable" doesn't matter

While learning, you build the skills used to validate

- You are limited if you use LLMs instead of building
- All your competitors can use ChatGPT too

DO NOT COPY/GENERATE WORK

- I prefer learning to grades
    - But grades should be fair
- Most learning is practice
    - Finding little lessons in the doing
- Copying/Generating reduces practice
    - Whatever you call it ("referencing")
- NOT WORTH THE RISK
    - Suspected problematic work is penalized
    - Confirmed problematic work gets a 0
    - Use my generous extension policies
- See "do-not-copy-work" in your repository

Large Language Models (ChatGPT etc)

I love tech, I get the appeal

- Natural language is great
- "Prompt engineering" gives better answers
- Straight answers

But:

- LLMs have major limitations and costs
- "Confidently Wrong"
- You lack the skills to verify
    - "it runs" isn't enough
- Harms your learning because you aren't doing

LLMs and Students

I've been teaching this course for a long time

- Over 30 instances over 20 semesters so far

Since ChatGPT was made public

- Code has LOOKED better
- and BEEN worse
- Steep drop in quality mid-semester
    - When complexity exceeds ChatGPT
    - Students hadn't built their skills
- Don't assume you'll be the exception

Summary

- Class is challenging (Worth it!)
- Get on course Slack
    - Set to get notifications from #announce
- Do weekly quiz on Canvas (open notes)
- Request copy of Git repository (repo)
- Do browser/slack configuration per class repo
- Submit Pull Request (PR) for weekly assignments
    - Request extensions before day they are due
- Don't trust sources over 3 years old
- Don't copy/generate work (Seriously!)
- Ask questions! (Really!)