ECE 458
Engineering Software for Maintainability

Introduction and Course Overview

Tyler Bletsch

Spring 2022

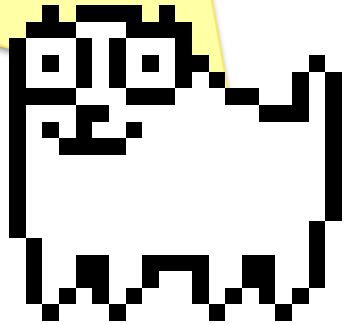
(Adapted from work by Drew Hilton)

Welcome!

Welcome to ECE 458:
Engineering Software for Maintainability

Your Senior Design Course!

Camera On, Please!



MEET YOUR TAS!

Anshu Dwibhashi

&

Arjun Sridhar

Evolution!

- You four years ago:

Fig. 1: Freshman year

- You now:

Fig. 2: You now have freaky robot hands I guess??

What this class is about

- Real software has a long lifespan
  - In industry, you might work **the same** code base for years or decades
- Contrast with code you write in school:
  - Turn it in, forget about it.
- Real world software's requirements **evolve**
  - New features
  - Changing requirements
  - …

Fig. 3: Software is like pokeymans

- How do we design software to ease later changes?
  - Goal of this class: learn this by **doing** and reflection

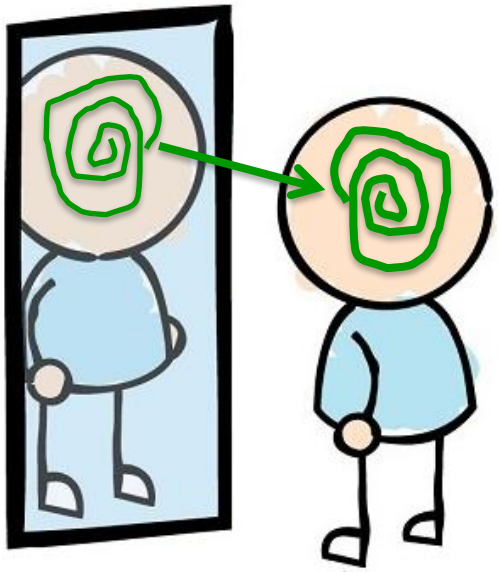
What this class is not

- This class is not about learning to program, you know that (well, you better know that...)
- This is not a lecture class
  - These are the first, last, and about the only slides I've prepared
  - You've been taught some software engineering skills, but...
- **You learn by doing!**

THE FUNDAMENTAL TEACHING MECHANISM OF THE COURSE: REFLECTION

Reflection: To take time to think on what you've done, critically evaluate how it went, and extract lessons (both positive and negative) from it.

Other courses:
I vomit green swirlies at you.



This course:
You produce your *own* green swirlies.

What are we doing?

- One semester long project:
  - Requirements staged into **4 evolutions**.
    - Changes will usually be substantive restructuring of core ideas:
      Not "add this form", but "change what this concept means"
- After each evolution, submit a report. Major parts:
  - Retrospective (analysis of past design choices):
  - Forward looking evaluation (analysis of current design):

Where's all the stuff?????

It's here:

https://people.duke.edu/~tkb13/courses/ece458/

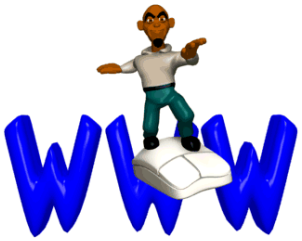
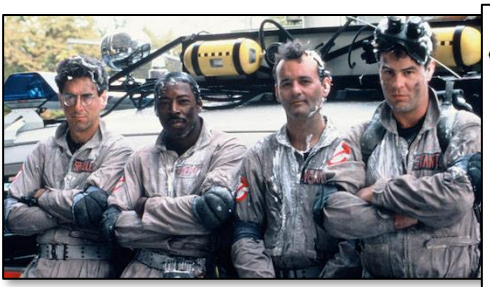


Fig. 4: internet.

Project groups

- You will do your project in groups of 4
  - Pick carefully: fixed for the semester
- Considerations:
  - Language/framework choices
    - Note: subject of next discussion
  - Other tool choices
    - Revision control, …
  - Skills and expertise
    - Ideal: strong skills, complimentary expertise
- End of class: find groups, start planning ev1

Fig. 5: All the best groups have four members



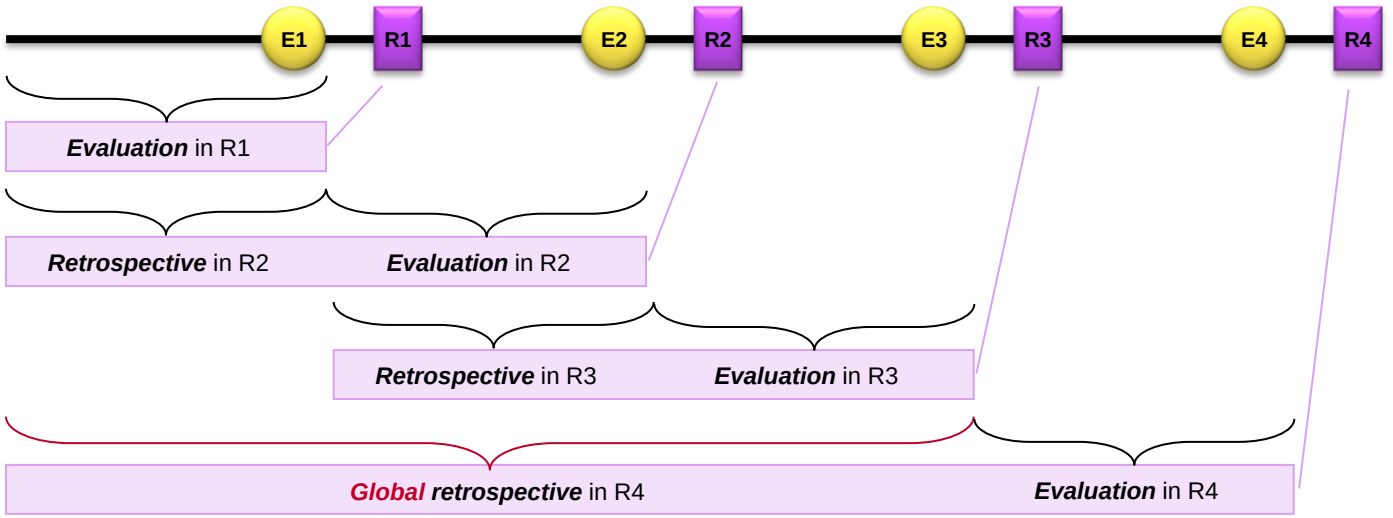

Project reports

- No specific page limit/requirement
  - Say what you need to say. Don't say more, don't say less.
- Expect document to be
  - Well-written: Organized, clear, precise. Include figures if they help
  - Analytical:
    - Delve into **why** your design is/was good/bad
    - Tell me what was bad, and how it could have been better (Hindsight is 20/20)
    - Include discussion of testing plan (part of design)
- Include:
  - Retrospective (analysis of past design choices):
    - How did your past designs set you up to win or struggle?
    - How did these outcomes align with your prior analyses?
  - Forward looking evaluation (analysis of current design):
    - What are its key features? Why did you design it this way?
    - What do you see as its strengths?
    - How about its weaknesses?

See course site for full details and rubric!

Project reports in time

- A report R_n for evolution **n** gives an evaluation of the design in evolution E_n *and* a retrospective on evolution E_{n-1}



- Evolution 4's report is special: it has a retrospective that covers the whole project in addition to just evolution 3.

Oral Presentations

- Day that evolutions are due: oral presentations
  - Each group member presents once
- 10 minutes per group
  - The *seemingly least prepared* group goes first!
  - Have your AV and laptop crap sorted out!
- Rough outline (2-3 min each)
  - Quick demo of working project
  - Retrospective from previous evolution
  - Overview of current design
  - Analysis of current design (include: why, strengths, weaknesses)

See course site for full details and rubric!

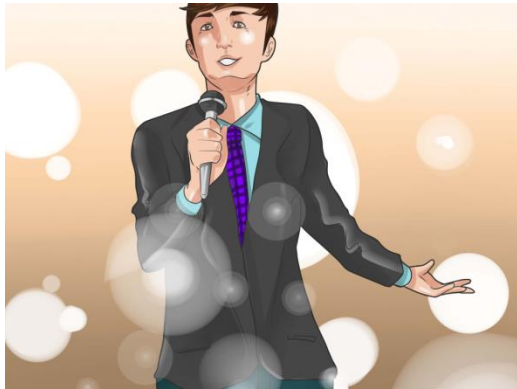


Fig. 6: WikiHow illustrations will never stop being hilarious.

Class Time: Four flavors

1. Class discussions

- Topics posted on class webpage (all posted now)
- Some topics have readings – you need to read it before you come
- Prepare ~1-2 pages of outline/notes on discussion, turn in at end
  - The notes are NOT a summary of the reading, but your thinking on the whole of the topic (reactions, opinions, questions, etc.).

2. Workdays

- Work with your group on your project
- I'll circulate around, answer questions, offer advice, etc.
- Includes **standup meeting**: teams go over status, blockers, and open questions publicly.

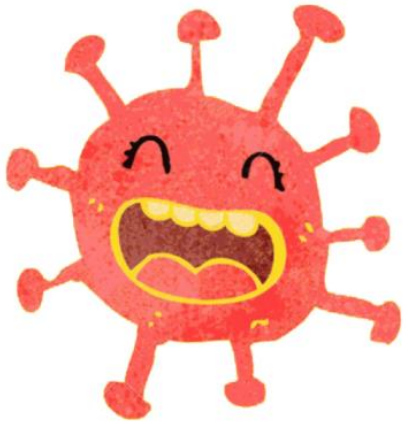
3. Presentation day

- If presenting: present. Else: support your group!

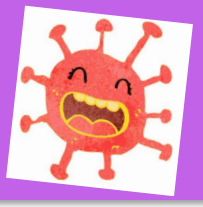
4. Reflection day

- Session after a due date
- Reflect on work so far, discuss newly released requirements

Stuff that's different for COVID-19



Planning for failure or planning for success?



- How to definitely be miserable, depressed, and fail

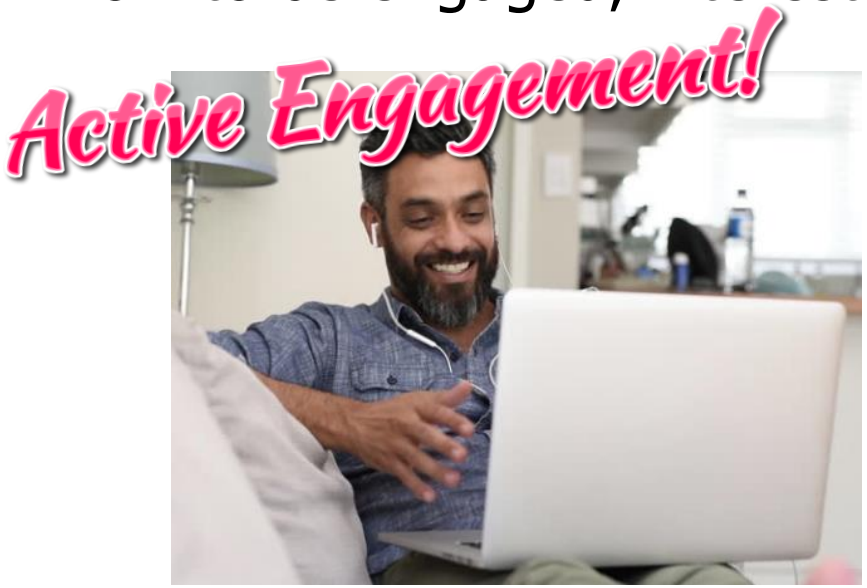
welp time to "attend" my "class" 😒

ughhhhhhhhhhhhhhhhhhhhhhhhhh

(Later)

oh no discussion grade low???

- How to be engaged, interested, and pass



I have a comment or question that I'm saying out loud in real time with real people! I'm engaging my brain, getting valuable socialization, and establishing a rapport with my peers!

Also I have a beard now!

(Later)

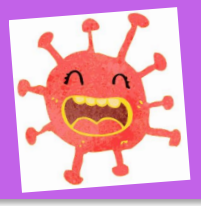
im fine here too

Important rule!

Camera On!

"You don't have a choice!"

Hybrid course policies for online people!



- The course is **synchronous**
  - **Video must be *<u>on</u>***: You were going to wear clothes and sit upright if this were in person, why change that?

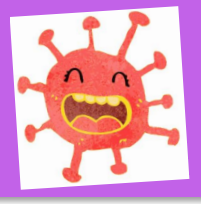
- Live attendance is required

- No recordings except for the first 2 sessions (nobody watches recordings of discussions and workdays)

CRITICALLY IMPORTANT TO GOOD HAPPY SUCCESS

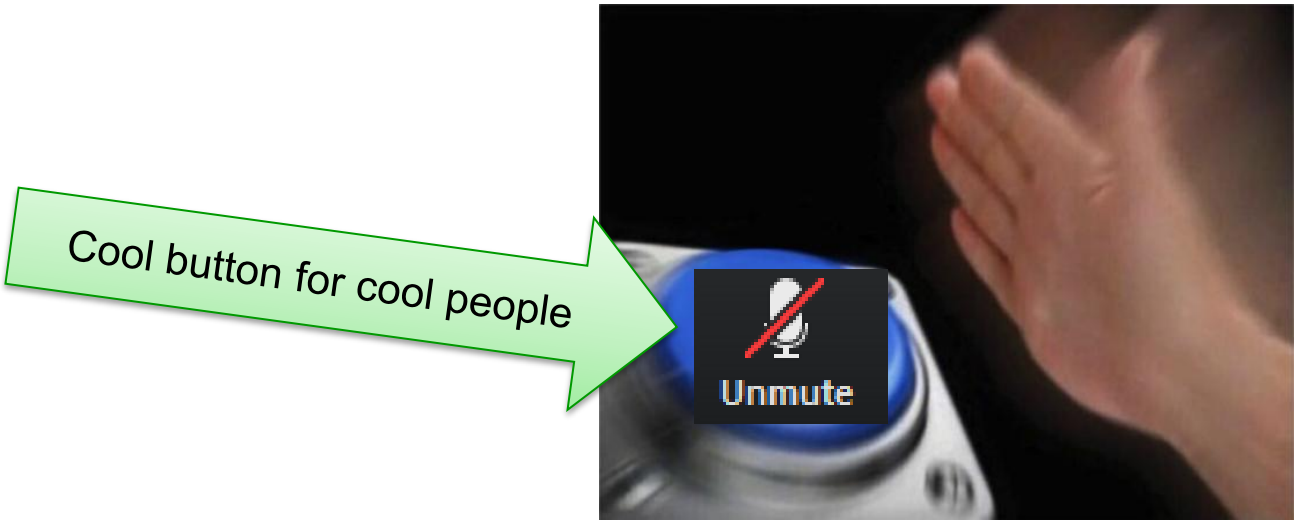
TALK OUT LOUD

How to interrupt



1. Preferable: **JUST START TALKING**
2. If it's very busy: use "**Raise hand**" button

BE DOING THIS CONSTANTLY:



"It's not rude if I'm literally telling you to do it."

Ask questions, please!

- Discussions are a great place to ask questions!
- In the past, students reported that they felt intimidated to ask about concepts that were introduced in discussion
  - But it turns out it was MOST students who felt this way!

- **Imposter Syndrome:** The phenomenon wherein "high-achieving individuals are marked by an inability to internalize their accomplishments and a persistent fear of being exposed as a 'fraud'. "

- Affects 90% of Duke I think... (including myself)

Fig. 7: Human psychology

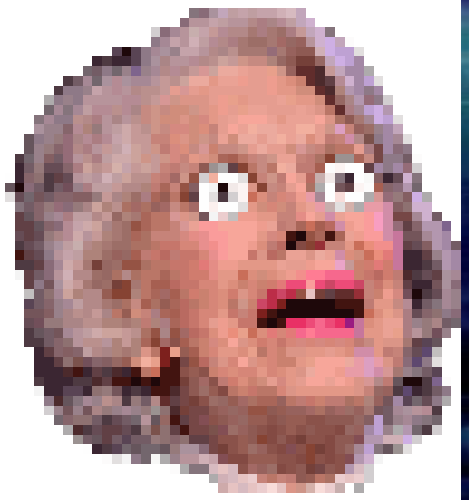
Grading (1)

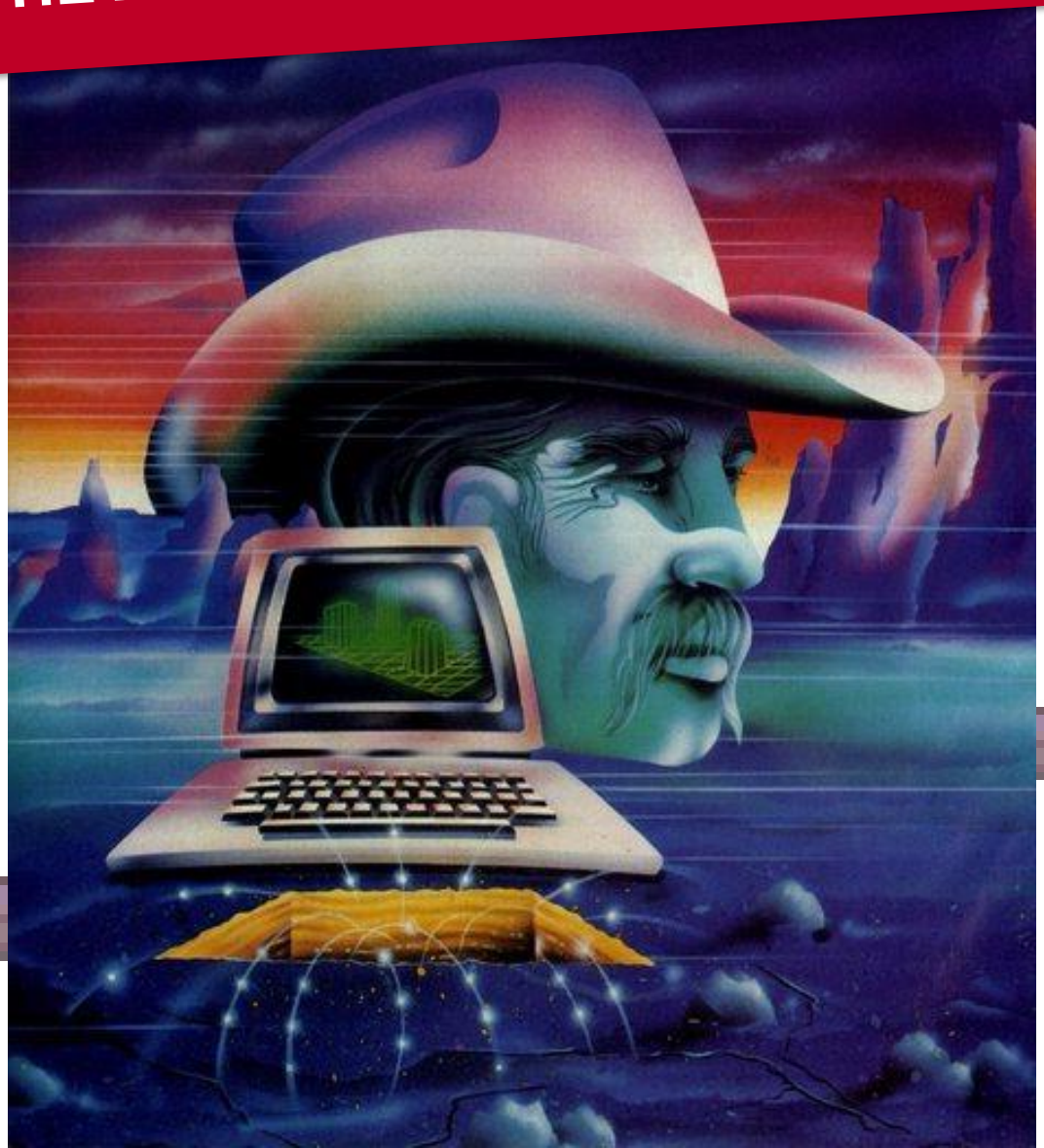
- 45% software deliverables:
  - How well did your system meet requirements?
  - Based on a **eval session** and **instructor functional testing**
  - We want to be as objective as possible, but in assessing "quality" without the benefit of a giant spec doc, there will be some subjectivity.
    - The system must actually be *good* from a customer perspective, not merely tick all the boxes.
    - Especially true for problems reported to you that you do not fix!
    - In other words, don't try to "Air Bud" us.
      - https://www.youtube.com/watch?v=Jvf0WWxrYRM
- 25% written deliverables:
  - Technical/analytical content: how well did you describe/analyze?
  - Writing: how well written are your documents?
  - Rubric is on course site

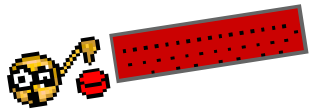
Grading (2)

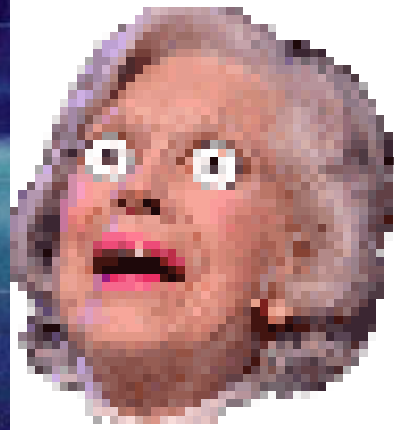
- 10% oral presentation:
  - Each group member does one evolution's presentation
  - Rubric is on course site posted
- 20% class attendance/participation:
  - Come to class regularly (2 free absences).
  - For discussion:
    - Have your discussion notes prepared (grading: 0, 70, or 100)
    - Actively participate in the discussions
  - For workdays/presentations/reflections:
    - Attend and participate as appropriate
- No exams! 

NOW IS THE PART OF THE PRESENTATION WHERE I GIVE YOU ALL THE ANSWERS ABOUT HOW TO DO WELL









Advice from past students: 2019 (1)

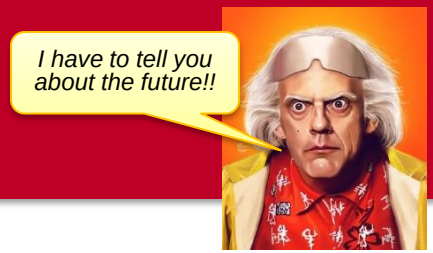


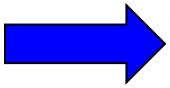
Fig. 9: Dr. E. L. Brown, ca. 2015

- Figure out **team dynamics** early on and be **upfront with your team if you're struggling** to compete something on time. They'd rather know I week in advance and by able to help than find out the night before an evolution is due.
- When choosing your project, do some **research into the features** of your framework before implementing things, and do some research into SQL and NoSQL databases before choosing to use Mongo because it seems easy and popular.
- Good use of your **framework's prebuilt components** and a **well chosen backend database** will make your project better and easier.
- Try to **use warroom coding** whenever you can because it makes integration easier and faster.
- I think **code integration should not be a second step** in the development process after initial creation of features but rather something that's a constant part of all components as you build them.
- You **can't view a feature as working but not integrated** with the backend/frontend, because if it isn't integrated it just doesn't work yet.
- 1. **Know your framework**…
  2. **Communicate with your team**. …
  3. **Start things early and leave several days for testing and debugging**. …

Read all 2019-2021 advice here

Advice from past students: 2019 (2)

- This class isn't solely about how well you can code, how cool your features are, how comprehensively you know your stack, or even how good your final product is. This class is about **communication and organization with your group**. There's **no chance you get through this project even without one member of your group**
- You'll notice in later evolutions that decisions you made at the beginning that seemed isolated become very important (and often screw you over), so **take the big picture decisions seriously** and make those decisions as a group.
- **Don't slow down after evolutions**. Those 3 weeks go by much faster than you'd think.
- **Work hard on evolution 1** so that you won't have to play catch up on other evolutions.
- **Test to break, not to validate!**
- 1) **DO NOT MISS YOUR CODE FREEZE**
  2) **DO NOT FINISH THINGS 95% OF THE WAY AND LEAVE THE LAST 5% FOR THE CODE FREEZE**
  3) **DO NOT PROCRASTINATE CORE ASPECTS OF PROJECT** (things that are blocking …)
  4) **USE A TESTING PLAN/AUTOMATED TESTING DURING CODE FREEZE…**
  5) … **DO NOT CODE WHILE DRUNK**
  6) **Be nice and respectful to your team** …



Read all 2019-2021 advice here

Advice from past students: 2020 (1)

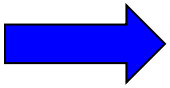
- As soon as something unexpected happens, ***immediately* reach out to your group because you will most likely need to re-plan** or shuffle some things around. The later this happens, the more screwed you get.

- **JUST GET STARTED.** Some of the requirements can be so intimidating, but **the sooner you start, the more questions** you can ask yourself, your team, the internet, etc., and you'll get stuff done faster. **I was so afraid to not know what I was doing, this really held me back in the first two evolutions.**

  ...

  Don't let **imposter syndrome** get the best of you, especially if you aren't in the group you anticipated to be in. Honestly, for me, I learned so much more than I would have if I were in said group, and retrospectively, I'm glad I had the chance to be in this uncomfortable-at-first experience in order to **grow as a human and as a developer**.

- **Take pride** in the work you do in this class... Once I forced myself to view the project less as "homework" and more as something that I wanted to be proud of, I found myself able to enjoy it way more.



Read all 2019-2021 advice here

Advice from past students: 2020 (2)

- (1) **Figure out task tracking ASAP**. …
  (2) **Review each other's code**! …
  (3) **Be honest about what skills you want to learn**. …
  (4) **Do your research, but don't over-research**. …
  (5) **If you're not going to do automated testing, then set up a detailed procedure for end to end testing, and stick to it**. …
  (6) **Over-communicate with your team**. …
  (7) **Ask Dr. Bletsch for guidance when you think it might be useful**. …
  (8) **Be ambitious about your code freeze deadline**. [ambitious=earlier] …

This student wrote a detailed paragraph for each of these bullets! Check it out in the link below!

- **If there's a global pandemic**, make sure to communicate and make efficient use of your time!

- **How you start is the best predictor of how you'll finish.**

Read all 2019-2021 advice here

Advice from past students: 2021

- **Effective communication/project management are almost as important as writing clean code**.

- YOU WILL LEARN SO MUCH IN THIS CLASS, **don't worry if you feel lost/stuck in the beginning**.
- **Don't be shy** to ask a teammate or two to jump on a quick call to address an issue that is blocking you.

- Get friends/family to [use] your service and give **usability feedback**.
- "Cheating" (in the form of **getting preliminary feedback before an evaluation**) is accepted - even encouraged.

- Make **informed decisions** [e.g., on hosting platform].
- Database **primary keys should be gibberish** (random hash/ID) - unrelated to the underlying data.
- **Lint your code**.

- Do not use Heroku! Do not use Heroku! Do not use Heroku! Do not use Heroku! [repeated 15 more times]

Advice from past students: 2021

- **Create a schedule**. Have at least one **weekly meeting** to plan out the week ahead. **Break down tasks** so that they are as granular as possible, assign every task to the appropriate team members, and **set deadlines** for every task.
- **Beware of over-planning**. Divide up responsibilities early on and **trust your teammates** to take care of their responsibilities.
- Don't save **deployment** until the end of the first evolution.
- This could very likely be the most **time-intensive class** you take, but you will **learn an insane amount**, just by doing.

- I would say learn how to **turn the requirements into a test plan** as soon as possible... **Test as a team**.
- Write a detailed **written test plan** and use it…**manual UI testing** is … important.
- Always **test with realistic data**.
- Set a **code freeze** several days before the deadline so that you have plenty of time to test. Create a **test plan based on the requirements** so that you are testing your software methodically and exhaustively.
- Use work days as "**bug bashes**": aim to have some features integrated that everyone can test and try to break together.

Advice from current professors of this class (me)

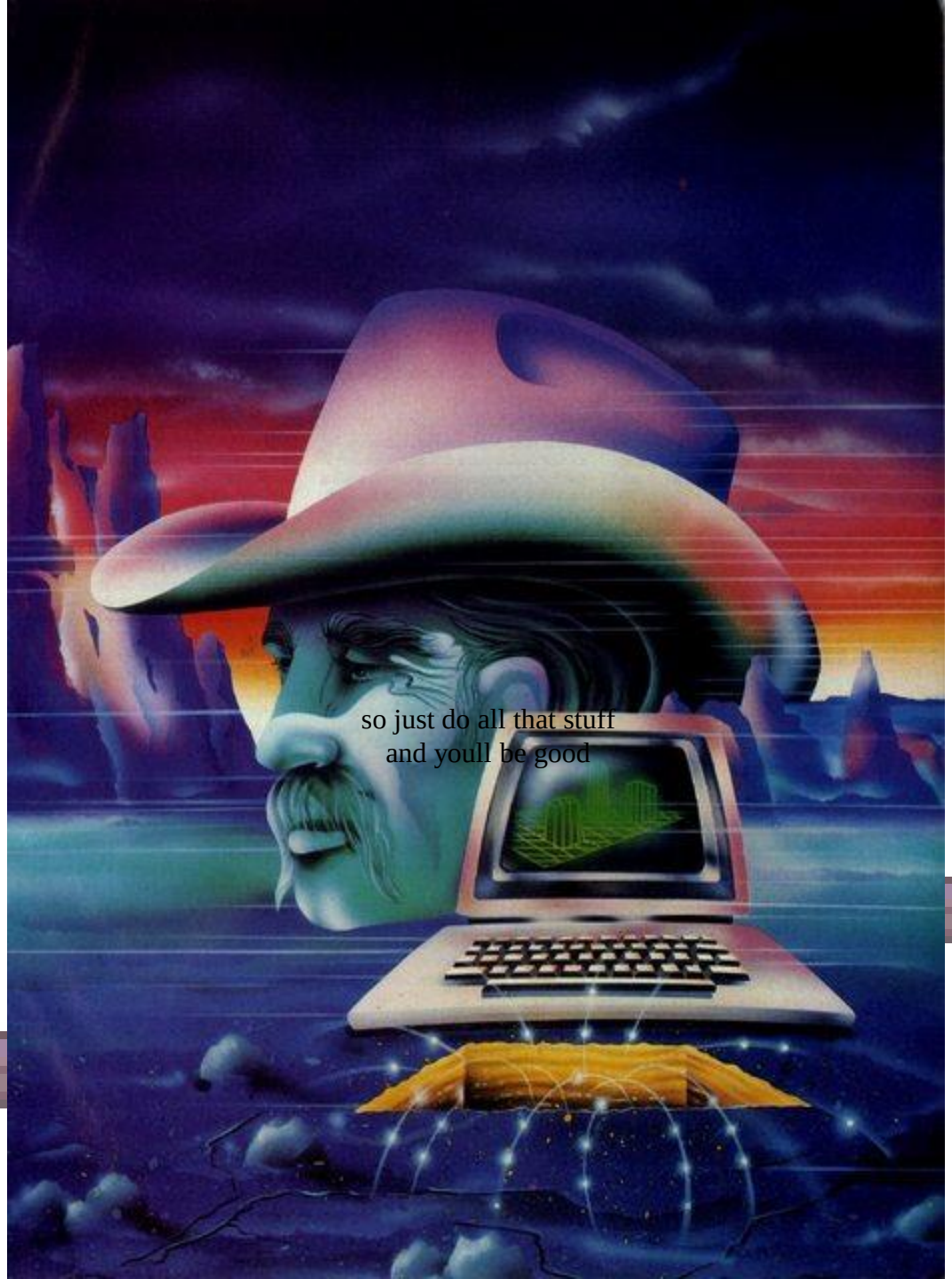
- Most common *first* mistake: waiting too long to "really" start

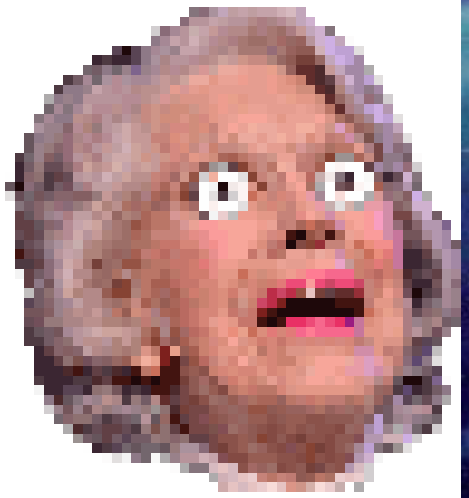
- You ALREADY know what you need to know (including how to learn what you don't)
- The requirements are POSTED

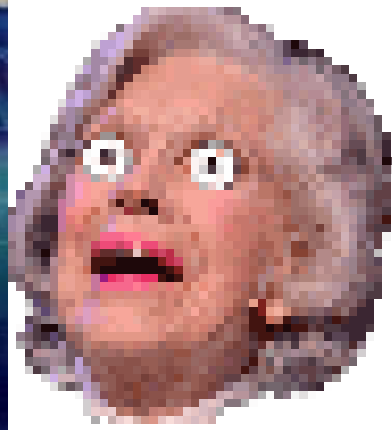
→ YOU ARE NOT BLOCKED ON ANYTHING. THEREFORE, ***GO!***

- Symptoms of not really starting:
  - Waiting for someone on your team to take the lead
  - Spending tons of time discussing little minutiae
  - Fiddling with automation and infrastructure for more than a few days
  - Being paralyzed by not knowing exactly how to proceed

    (Cure: Just try crap. Build something. *Go*. It beats paralysis.)


so just do all that stuff
and youll be good





Academic Integrity

- Expect academic integrity from all of you
  - Duke community standard
    - I will not lie, cheat, or steal in my academic endeavors, nor will I accept the actions of those who do
    - I will conduct myself responsibly and honorably in all my activities as a Duke student.
- Concrete rules:
  - Discuss anything you want
    - Give credit where its due if you use other groups' ideas
  - All code should be produced within your group
    - Don't share code outside your group
    - Can use libraries for graphics, sound, etc (e.g., SDL) as needed
- Not sure? **ASK**

Equitable labor rule

- Triggers in the event of a gross imbalance of team member effort (<50% of their share as estimated by the instructor)
  - Instructor may multiply that student's score on group assignments by their estimated contribution.
  - Instructor may distribute points lost in this manner among the other members of the group.

- For cases where a student's contribution is at or near 0%, this may be done without warning, otherwise this action will only be undertaken after significant discussions and a sustained failure to change trajectory over time.

- More details on course site
- I hope to never invoke this rule ☹

Specifics of the Project

School Bus Logistics Management System

- Probably web based, but many specifics are left up to you
  - UI design? Framework/libraries? Data model? Devops? Test plan?

- All 4 evolutions are (almost) already written
  - I will not change them in response to your status

    (Some students worried that I'd put in their worst fears)

Requirements

- Requirements will be distributed as PDFs
  - New requirements in blue
  - Changed requirements have old requirements in grey
    - (replacements in blue)

- Unclear on requirements? Ask
  - Happy to clarify anything
- Unspecified requirements/behavior?
  - Do anything reasonable
- Contradiction? Is a requirement stupid?
  - Discuss it in class or on the forum; changes may be possible
- Don’t need to be artistic
  - But it does need to be efficient and usable!

Variance requests

Variance request: A formal process for requesting a change in the requirements for your group.

To submit a variance request, post a *question* in the Ed forum with the "variance" tag as well as the appropriate evolution tag. In the body, fill out the following template.

This way we can track variances on a per-group basis easily and get your precise change down in writing. (CAREER TIP: Always get stuff in writing!)

Group number: ___
Group name: ___

Requirement number(s) affected: ___

Requested change: ___

Rationale: ___

Submission

- Submission of projects by Sakai

- Server:
  - Have *at least* a dev/test server and a production server
  - **Production server** should only be touched in the week before the evolution is due – *otherwise it's frozen!*
    - More details on next slide…
  - Recommend VM from OIT: https://vcm.duke.edu
  - NOT recommended: the various fly-by-night "free" hosting providers
    - Students have been screwed by this before...
- A note on platform:
  - You must document ALL environmental pre-requisites and instructions for setup of your product
  - If you do anything mobile, please include instructions for emulator

Production server freeze policy

- Production server is *frozen* from the moment of an evolution's deadline until 7 days before the next evolution's deadline. **This is true even after your eval session and after an evolution's grades are posted!**
- *Frozen* means:
  - No code changes, no configuration changes, no content changes.
  - Exception: In-class demo, informal use in the presence of instructor
- While frozen, the server must stay up
  - May be accessed by the myself or the TAs at any time.
  - We may revisit past tests or do ad hoc testing.

- If you inadvertently violate the production server rules (e.g. accidentally updating it), just let us know so we can be aware.

QUESTIONS?

Minor logistics

- For class after next, need to select:
  - Four to present a programming language + framework
    - *See course site for details*

Meet your customer

- Provider of school bus services to a handful of school districts in your area.
- Operates a fleet of buses to transport K-12 children to/from school as well as to special activities and events.

A brief primer on school bus logistics

- Basics: Bus starts in depot, travels on a route to collect kids, drops them at school; does the reverse in the afternoon
- The school bus system can be **complicated**
  - Routing issues:
    - Administrators have to design routes to maximize efficiency; want the computer to assist (traveling salesman problem)
    - Drivers can get sick and buses can break down; need to re-route on the fly to adapt
    - Parents have multiple kids in different schools, and want a single bus stop location
  - Communication issues:
    - Drivers need to know the most up-to-date route
    - Parents/students want to know the ETA to bus arrival
    - Parents want to tell driver ahead of time to skip them
    - Schools can suspend students, thus suspending bus service too
    - Administrators may need to contact everyone on a route
  - More…

Evolution 1: Go!

- **Find your groups**

- Start trying to get the **requirements** out of the customer (me)

- Maybe even talk about your **design**?
  - What are the key objects to model?
  - Decide how to split up the work?
  - What do you think the main challenges will be?
  - How should you design to accommodate whatever changes I throw at you?
  - What programming language do you want to use?
    - Detailed discussion on Monday.
  - What procedures and tools to use?
    - Code control? Scheduling? Milestones? Task tracking? Etc.

A realistic beginning

- The formal requirements have been published, but they may not be clear yet...

- To get started, I recommend you interview the customer:
  (as impersonated by me)

Fig. 10: Release planning