ECE 458
Engineering Software for Maintainability

Introduction and Course Overview
Tyler Bletsch

Spring 2024

(Adapted from work by Drew Hilton)
Welcome!

Welcome to ECE 458: Engineering Software for Maintainability

Your Senior Design Course!
MEET YOUR TA!

Yifan Jiang
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
  - ...

- 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 about the only slides I’ve prepared
  • You’ve been taught some software engineering skills, but...

• You learn by doing!
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:
      Less “add this form”,
      more “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/ece458-2024sp/
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 $E_n$ 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!
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
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)
Grading (1)

- 45% software deliverables:
  - How well did your system meet requirements?
  - Based on an **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](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
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: 2022

• i literally cannot emphasize this enough. **ev1 is the most important ev** and if you have a strong base set up, it will pay off SO MUCH in the long run. definitely put in the work in the beginning, otherwise you'll just be playing catch up for the rest of the semester.

• **Think critically about your ORM.** If they all look the same, look at bug forums to actually get a read on how strong the community is and how the community feels about the ORM management responsiveness.

• If your group mates are not communicating what they are working, they are probably not working/working effectively. **Vague communication about progress means that there is no progress or the progress is not going to be useful** since they cannot communicate it. All of your work is interconnected.

• Try to decide on a **decision making process and accountability process early.**

• Do not procrastinate and try to do everything on the last day. You'll be surprised **how many shit** could happen in that one day (or even last 12 hours).

• **SPRING BREAK RUINS THINGS.** For that one evolution, if you scrape the week of spring break out, there's actually less time to complete the project compared to ALL other evolutions.

• Really, seriously, **do start early in ALL the evolutions (especially the first tho).** You will have to learn a lot at the start of the course, and you want to start on that quick so you have time to actually do the coding.

• **Making a schedule of some kind for your team is crucial,** even if its not uber-detailed. If you don't, some teammates are going to end up doing a disproportionate amount of work and you potentially won't have a working product, and no one likes that.
Advice from past students: 2022

- Make **intentional design decisions** where you can. Have a **comprehensive test plan**. All of this melts away; don't let the course listing fool you, this is a **communication course, not a coding one**.

- **Try to have *everyone* read *all* of the requirements.** We got into the bad habit of only reading the new stuff each evolution, and only reading the stuff we had deemed as relevant to our particular scope of the project. This led to us missing a few small things, or interpreting wording incorrectly.

- **Talk to other groups if you get stuck.** They can help give you ideas to get moving again!

- **Don't let issues within your team fester.** Try to pay attention to how people are handling things, and offer or ask for help when you (undoubtedly) need it.

- **Consider pair programming, or war room coding.** It can stop you from spending hours going down the wrong path, and help you work more efficiently!

- Don't use a group chat to communicate with your team. Find some tool that's a little more powerful to help do some of the work for you. **Consider a slack, or a discord.**

- **Know when to abandon (replace) something you've done.** Some times trying to hold onto a choice or code feels right (because you've already put time into it), but if you are recognizing that it's going to slow you down or cause pain and fixing it isn't going to solve your problem, don't be afraid to start over.

*Read all 2019+ advice [here](#)*
Advice from past students: 2023

- **It truly is necessary to begin evolution 1 right now.** Getting up and running from scratch poses unique challenges and you need all the time to develop a workflow and ensure that the team is capable in your stack. Taking on feature/technical debt this evolution will make it a tough game of catch-up for the rest of the semester.

- **War room code as much as possible.** Task management software, constant communication etc. is great, but nothing beats meeting up in-person as much as possible and just working alongside each other and chatting.

- **Create a systematic approach to solving problems from the very beginning.** Be organized, be thoughtful, and always assume that tasks will take longer than you expect them to be. ... **Also, make sure to test in production!** Don't assume that all the features will work without testing them in the real production environment!

- Though the course is daunting at first, **you will come out to be a better developer after this.**

- Please just chose a framework that is widely used/documented...

- Treat DevOps as a portion of the project and allocate time for it...

- Adopt a project planning strategy in the first evolution...

- Meet regularly in person with your team. We had twice weekly meetings, and found this worked well for us...

- **Make a test plan.** ...even if it's just a rough framework/list of things that you need to test for and not a rigorous test plan, it's better than nothing.

Read all 2019+ advice [here](#)
This student even wrote you a little guide to how they approached the course!

Here is a short list that you can use.

1. Have a system design meeting to layout how you will deploy your software.
2. Are you going to have your frontend/backend use a different tech stack? (This depends on individual experience of groupmates)
3. If frontend and backend is separated, make a communication channel between the two as fast as possible.
4. If you need to use HTTPS to encrypt your deployment you can do it using docker, nginx, certbot without using cloud software.
5. Duke VCM is a great way to deploy your software.
6. Use Jenkins to auto-deploy your main branch to production (use github webhooks). This makes life so much easier.
7. Have a test deployment machine ready so you can actually test your software.
8. Do not delay frontend development because it is dependent on backend. Have dummy data ready and endpoints sorted out as soon as possible so frontend can start developing early.
9. Spend excessive amount of time on the first evolution because those time will be rewarded in the future if you set up proper pipelines for development and deployment.
10. Making technical debt in the first evolution WILL propagate.
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 you'll be good
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

Experiment management system for a food shelf stability lab

- Probably web based, but many specifics are left up to you

- 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 on the course site
  - 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!)*

<table>
<thead>
<tr>
<th>Group number: ___</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group name: ___</td>
</tr>
<tr>
<td>Requirement number(s) affected: ___</td>
</tr>
<tr>
<td>Requested change: ___</td>
</tr>
<tr>
<td>Rationale: ___</td>
</tr>
</tbody>
</table>
Submission

• Submission of projects by Canvas

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?