ECE 458
Engineering Software for Maintainability

Introduction and Course Overview
Tyler Bletsch

(Adapted from work by Drew Hilton)
Welcome to ECE 458: Engineering Software for Maintainability

Your Senior Design Course!

Camera On, Please!
MEET YOUR TAS!

Anshu Dwibhashi
&
Lucas Liu
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 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:
      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
• After groups have formed, they will be **randomly permuted**, with one member from each group rotating to another.

• Why?
  1. In industry, you don’t always get to pick who you work with.
  2. Reduces “musical chairs” effect where students rush to form groups and last group to form is at a disadvantage.
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!

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.
   - Often 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 because we're online during COVID-19
• How to definitely be miserable, depressed, and fail

**Passive Watching**

• How to be engaged, interested, and pass

**Active Engagement**

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)

(Later)

*lecture.mp4*

Welp time to “attend” my “class”
ughhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh

Oh no discussion grade low???

I’m 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 on**: You were going to wear clothes and sit upright if this were in person, why change that?
- Live attendance is expected
- Recordings will be made available for reference later, but in a class like this, that’s not worth much
  (They will always be private to students in the course)

- What if you have **real life reasons** why you can’t do the above?
  - You may request to take the course **asynchronously** by emailing me directly.
  - Valid reasons: crazy timezone, severely complicated living/working situation
  - Invalid reason: preference
CRITICALLY IMPORTANT TO GOOD HAPPY SUCCESS

TALK OUT LOUD
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.”
These are extraordinary times.

The course conditions are extraordinary.

The Duke response has been extraordinary.

The logistics of running a course like this hybrid will be extraordinary.

Let our commitment to each another and to ourselves also be extraordinary!
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 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](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: 2018

- In your first evolution: **get automated testing done** for one thing on your backend. Then in your second evolution: **Extend that automated testing to everything else**. "I don't automate because I don't have time. I don't have time because I don't automate!" My group fell into this loop, and it would have been really nice if we hadn't.

- **Don't procrastinate, communicate with your team** effectively, **don't be afraid to ask for help** from your teammates if you don't know what's happening. Meet your **code freezes** and spend a lot of time testing, **test extensively**. **Start the evolutions early**, just because it seems like you have a lot of time, doesn't mean you actually have a lot of time. **Don't be afraid to re-write or re-factor** if you think it'll be worth it; saying we already started that way so let's keep going is a bad excuse. Meet with your team frequently, even if it is a quick meeting. **Break down the evolutions and assign tasks** to specific member instead of letting members pick up tasks as they go, and have a schedule. **Finish the blockers first**, or else it's a waste of time.

- **Logistics** is a really big part of this course, so **spend at least a day each evolution planning**. Good planning will lead to success in the evolution.

- For each evolution, **set a code freeze date**. But make sure this code freeze date does not include testing and then allow a few days for testing because you will find unexpected bugs right before the deadline. Setting a code freeze date too close to the deadline does not allow enough debugging time.

- **Plan out your design** well. Although it may feel like you're not making a lot of progress by planning it will pay back over the course of the evolution.

---

Read all 2018 advice here
Advice from past students: 2019 (1)

- 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 2019-2020 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 ...
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 2019-2020 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] …

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 2019-2020 advice [here](#)
so just do all that stuff
and you'll 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
Specifics of the Project

Measurement instrument inventory and calibration tracker for an industrial backup power validation and test company

- 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 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 Piazza thread 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 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!*
    • We test on this deployment when you’re not around!
• Recommend VM from OIT: [https://vcm.duke.edu](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
QUESTIONS?
Minor logistics

• For next time, need to select:
  • Four to present a programming language + framework
    • See course site for details
Meet your customer

Hypothetical Power Testing

Keeping the lights on and the building not on fire

- Field testing validation of industrial power backup facilities at customer sites (cell tower sites, industrial control facilities, nuclear power plants).
- In-house lab validation of power equipment.
A brief primer on industrial power

- Power from utility company is AC (120, 240, 480, or even 600V AC)
- Some industrial systems use AC, some use DC. 48V DC is common for telcos.
- Power backup systems often deployed on the DC power bus.

An industrial-scale battery room
A high-level power schematic (1)

Main generator(s)

Utility power

Portable generator (To patch in in case of generator failure)

Manual transfer switch

Automatic transfer switch

Primary distribution panels

480V AC

Transformer

120V AC

AC panels for office, lights, etc.

AC panel for rectifier plant

AC panels for HVAC (chillers, air handlers, etc.)

To next slide
A high-level power schematic (2)

- **AC power for rectifier** (From previous slide)
- **480V AC**
- **48V DC**
- **Primary DC panel**
- **Secondary DC panels**
- **Main DC bus: rated for 10000A!**
- **48V battery array**
- **Inverter**
- **120V AC**
- **Critical AC gear (security locks, monitoring, etc.)**
- **Servers/switches/etc**
Testing the system (1)

- **The customers** of HPT are companies that deploy and use industrial backup power systems (cell tower sites, industrial control facilities, etc.).
- These customers had **vendors** come in and deploy or upgrade their power system.
- Because of the critical nature of these systems, the customer then hires a neutral third party, **HPT**, to inspect and validate such systems.
- **Why?**
  - Remember in ECE 250 when you hooked up wires wrong in Logisim & they’d turn red?

![Diagram](image)
• In fact, sometimes they even turn white...

• This can have outcomes ranging from low voltage to worse...
Testing the system (3)

- How do conduct testing?

_LOTS OF INSTRUMENTS!!_

- Multimeters
- Amp clamps
- Custom-designed load bank with integrated current testing
- IR Thermal imagers
- Conductivity meters
- Specific gravity instruments (for battery chemistry)
- Thermocouples
- Temperature and humidity loggers
- More...

A small fraction of the equipment inventory
Customer problem: Inventory

- Field teams keep much of this in mobile vans for field work; in-house lab teams have some as well
- They currently track this gear with a big spreadsheet
- As the company grows, the spreadsheet gets harder and harder to keep updated and reliable
- After driving 12 hours to a job site only to be missing critical gear, they’ve decided they need an **inventory system** to better organize things
Customer problem: Calibration

- Most equipment needs to be **calibrated** to ground truth on a set schedule
  - Meters can drift as components age
  - Undetected damage can invalidate measurements
- It’s critical that HPT be able to prove that all instruments used were calibrated according to the manufacturer’s schedule
- Types of calibration now:
  - In-house with a calibrator (special instrument)
    - May fill out a form to note variances as you go
  - Some equipment shipped to manufacturer for formal calibration
  - Custom-built gear has special & complex calibration procedures
Why is calibration so important?

- Two true stories from the inspiration behind HPT:

- A power backup system was underperforming, and HPT determined that a malfunctioning generator was out of spec. The vendor who installed it used a cheapo meter and claimed it was in spec, and initially refused warranty service. HPT was able to show it was out of spec and that the meter that showed this was in calibration within 6 months.

- A bolt broke off during installation, and the equipment vendor blamed the customer for installing wrong. HPT tested with their own torque wrench (a tool that tightens to a configurable torque), and the bolt broke again. The vendor claimed HPT used too much torque, but HPT showed their wrench was set to the given value and was in calibration, and it turns out the documented torque value from the vendor was wrong!
Examples in instrument inventory management

Questions:
- What make/model is it?
- How does it get calibrated?
- When was it last calibrated?
- When will it need to be calibrated next?
- What data was recorded at its last calibration?
- If it was calibrated with another instrument, when was *that* last calibrated?
- Have we validated that it still works recently?

(Calibration ensures it works, validation checks if it works)
- What instances of model X do we own?
- Where is the equipment assigned (e.g., a certain van or lab)?
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?
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