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!
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 first, last, and 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/
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 $E_n$ 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.

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)

Fig. 7: Human psychology
Stand-up meetings

• A brief meeting where teams go over status, blockers, and open questions.
• Every Wednesday except for presentation days
• Noted on the class schedule
Grading (1)

- 45% software deliverables:
  - How well did your system meet requirements?
  - Based on a **demo 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?
Grading (2)

• 10% oral presentation:
  • Each group member does one evolution’s presentation
  • Rubric will be 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: 2017 (1)

From a survey given at the end of the semester.

- 3 useful lessons:
  1. Use a web **framework** that helps you hit the ground running, lets you learn quickly (good docs really help here), and is flexible enough to tweak as needed (look for 3rd party packages / plugins).
  2. Good **organization of tasks**, who is working on them, and when they need to be completed is essential (full-stack is your friend).
  3. **TESTING** IS EVERYTHING. IT MAKES ALL THE DIFFERENCE. DO IT.

- Take the time to actually **design your UI** with the same care that you do your backend, otherwise you will end up with a messy, unintuitive interface.

  **Set milestones**, both individually and as a team, and hold each other accountable to them. Procrastinating work in this class will screw you, arguably even more so than in CS 308.

  **Keep it simple**, especially early in the project. The teams that are best at adapting for requirements late in the semester are the ones who didn't try anything too crazy early on.

  Take a look at when the first evolution is due, subtract at least three days from it, and go ahead and set in stone that you will have your code "done" by that time. You will need the **extra days to actually test your software and fix bugs.**
Advice from past students: 2017 (2)

- Stick to technologies that are either (a) familiar to you or (b) really well documented and supported. **Don't choose tech just because it sounds cool** -- the class is too short to learn how to do things the right way so use something that is easy to pick up.

Set aside a good amount of time for **testing**. This really really really really really really really really really really really really really really really really really really helped our team a lot. I literally cannot stress how important it is to do this. There was a huge, visible difference between teams that took the time test their software and those that didn't. Set aside a weekend just for testing and then fixing the bugs that you found during that period. Do this even if you haven't completely finished all the dev yet -- unless you are super far behind, it's definitely better to test most of your software than to wait to test all of your software only to realize that you don't have time to do any testing.

**Don't wait until the last day to deploy to prod.** Prod is messy. Shit breaks. You don't want late nights in dev ops hell.

**Pick a good team.** You are going to be spending a lot of time talking to/with each other.

- If learning a development framework feels overwhelming, **create a specific plan for learning** it. Ex: rather than "spend 2 hours today, 2 hours tomorrow" use "watch this tutorial series today, read through this documentation tomorrow, implement xyz the next day".
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.
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 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 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

IT Asset Management System for a software company

• Many specifics are left up to you
  • Web based? Desktop application? Mobile app? Your choice

• 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 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

• 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

- Midsize software company focused on niche business application development: bespoke back-end software for small/medium businesses (e.g. dentist scheduling, drycleaner recordkeeping, etc.)
- Maintains their own small **datacenter**; management of this datacenter is currently run by spreadsheets and manual effort of the IT team
- They want a system to organize all their gear (servers, network, etc.)
- Each team is a contractor vying for their business
A brief primer on IT management

- Systems are standardized to the **rackmount** form factor
  - Standard width (19”), height is an integer multiple of 1.75” – this measure is called a **rack unit (U)**
  - Example: This is an HP DL380 Gen6 server, it is 2U.

![Racked 2U HP DL380 Gen6 server]

- Gear is identified in a **datacenter** by a **row letter, rack number, and U location** vertically within the rack, e.g. “West lab rack C19, U22”

- It’s common to show a **rack elevation** (diagram) of one or more racks

![Rack elevation diagram]
Examples in IT asset management

Questions:
- Where is server X located?
- What make/model is server X?
- What CPU/RAM/cards are in server X?
- How is the network cabled up?
- What is the IP address/hostname of server X?
- Who owns server X? Who is using server X?
- Is server X up and working?
- More...

Kinds of hardware:
- Servers
- Network switches and routers
- Storage controllers
- Appliances: Load balancers, firewalls, etc.
- Battery backup systems (UPS)
- Interfaces: Keyboard, display, KVM switch, etc.
- More...
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?

Time permitting: Field trip!!
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