ECE 458
Engineering Software for Maintainability

Introduction and Course Overview

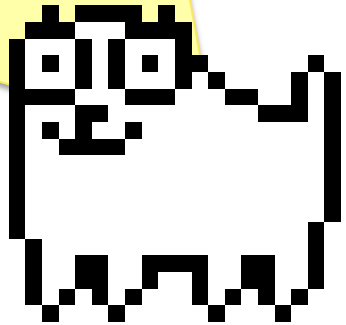
Tyler Bletsch

(Adapted from work by Drew Hilton)

Welcome!

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
  - ...

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

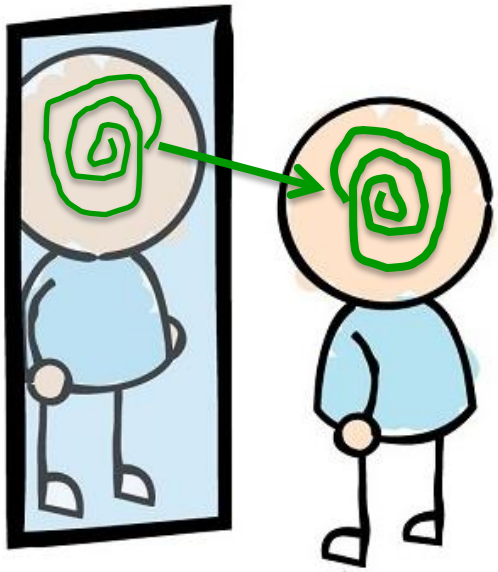
What this class is not

- 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!**

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:
  - An inventory and production system for a large scale food manufacturer
    - Based on real requirements from practicing scientists and managers at a major food corporation
  - 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):
    - How did your past designs set you up to win or struggle?
    - How did these outcomes align with your prior analyses?
  - Forward looking (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?

Where's all the stuff?????

It's here:

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

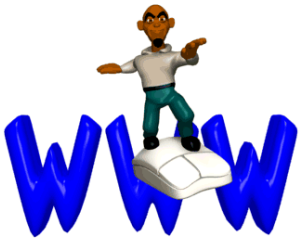
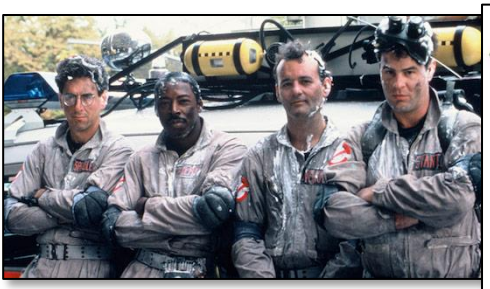


Fig. 4: internet.

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)

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

Oral Presentations

- Day that evolutions are due: oral presentations
  - Each group members 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)

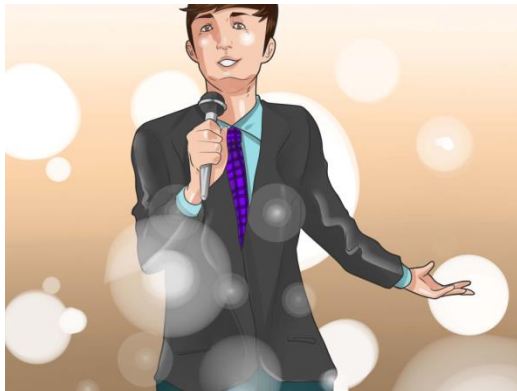


Fig. 6b: Please don't look like this when presenting.

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.).
   - Two **discussion leads** per topic (sign up soon!)
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 Tuesday except for presentation days
- Noted on the class schedule

Fig. 8: Hahaha look at these blob people

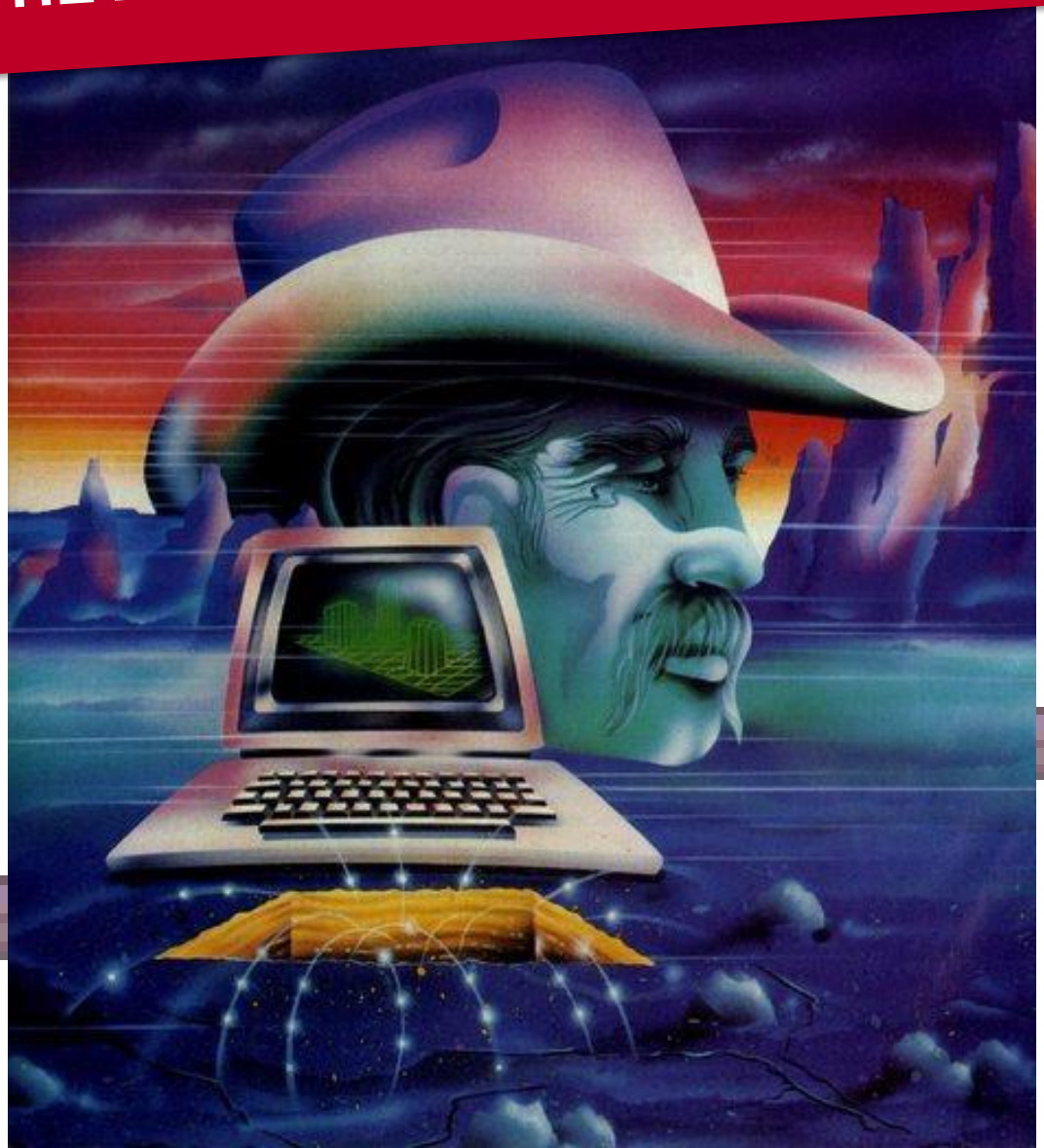
Grading (1)

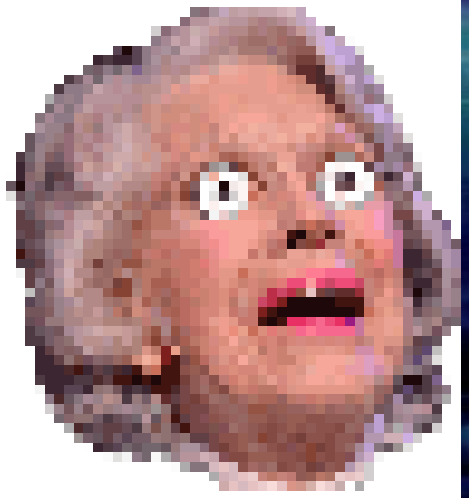
- 45% software deliverables:
  - How well did your system meet requirements?
  - Based on a **demo session** and **instructor functional testing**
  - We want to be as objective as possible, but in assessing "quality" without the benefit of a giant spec doc, there will be some subjectivity.
    - The 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!
- 25% written deliverables:
  - Technical/analytical content: how well did you describe/analyze?
  - Writing: how well written are your documents?

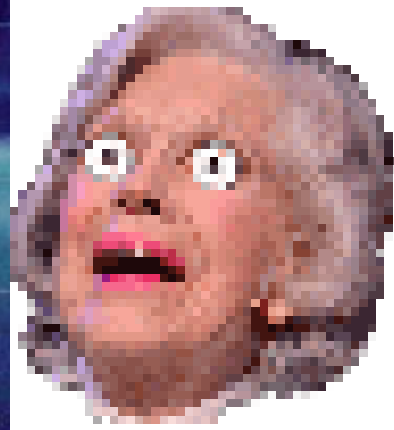
Grading (2)

- 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: 2016

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

From a survey given at the end of Spring 2016.

- "Make sure that you **set individual milestones during an evolution**, this is one of the classes where actually trying to do everything the weekend before will not work regardless of how smart you think you are."
- "Take a long time **deciding which framework, database and front end** to use as it really affects future evolutions. Really look at the data and the requirements in your design decisions (especially databases), rather than going with just what you know or a 'shiny new tool'."
- "**Don't switch frameworks** at any point during the project. It is a fucking nightmare. Not worth it. Beware picking a framework that is cool and trendy that you know nothing about...don't do it "just to learn" - make sure it's got good documentation and support."
- "Understand that this is a class about **satisfying software requirements**. This is not a class about writing modular, clean code. Yes, doing that will make it easier to adapt to changes in the specifications, but it will not be a part of your grade and **no one will look at your code**."

Advice from past students: 2017 (1 of 3)

From a survey given at the end of Spring 2017.

- 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.

- You're learning a new language (most likely). remember how shitty your java code from 201 was? That's how your code is going to look to you in the future unless you **read up on your frameworks** and **start practicing good quality**.

- 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 of 3)

From a survey given at the end of Spring 2017.

- 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 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: 2017 (3 of 3)

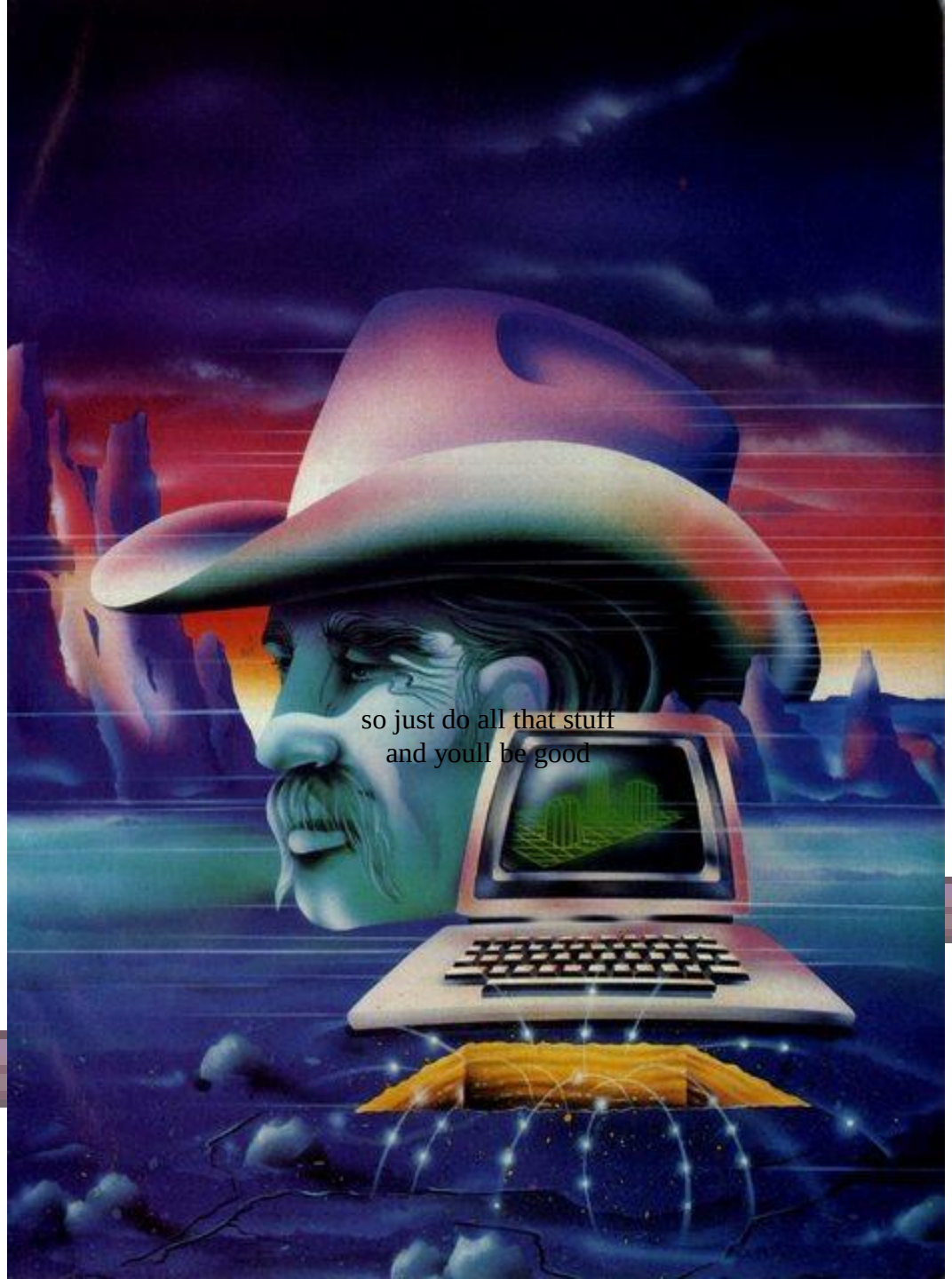
From a survey given at the end of Spring 2017.

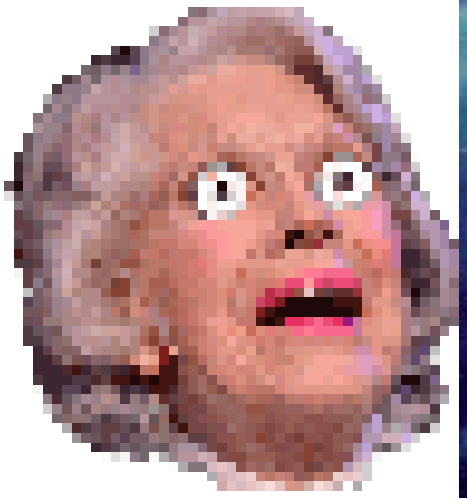
- This class is a consistent amount of work throughout the semester. However, if you **schedule wisely** and have a group that **communicates well**, you won't be stuck pulling an all nighter the night before!

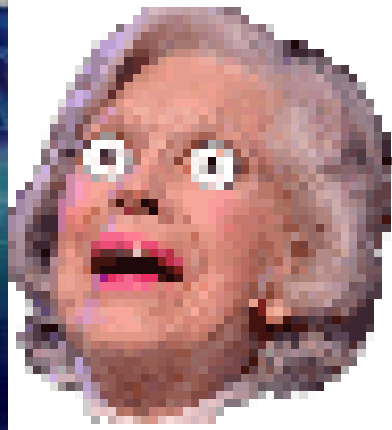
- For each evolution, take the time to set up a **firm schedule that is reasonable** (key word reasonable). Don't be too ambitious about the dates, but at the same time, don't save everything for the last minute. Make sure you leave enough time for testing at the end. It's really easy to fall behind, leading to a loss in testing time and subsequently a lower quality product. **Testing is super important** and in many cases, catches a lot of bugs that wouldn't have otherwise been caught. During each evolution, **make sure everyone is on track to finish by the corresponding dates** in the schedule. If adjustments have to be made, make them earlier rather than later.

- Make sure to **schedule a code freeze before each deadline** so that you have at least 2 days to **test and fix bugs**. A **fully functional app with no bugs and a few lacking features is better than having all the features but a lot of bugs**. This class is a lot of work so start early and work efficiently, but also make sure to enjoy senior spring and try not to let this class stress you out too much.

- **Plenty of testing** is the key to a perfect demo. **Code freeze at least a week prior** to the ddl.


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**

Specifics of the Project

Food Manufacturing Production Lifecycle Manager (PLM)

- 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!)

Group number: ___
Group name: ___

Requirement number(s) affected: ___

Requested change: ___

Rationale: ___

Submission

- Submission of projects by repository pull
  - Your choice of revision control system
  - Coordinate with TA on submission
- Server:
  - Have *at least* a 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: http://vm-manage.oit.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*
  - Two to act as discussion leads for the programming discussion

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:
  - The food company's logistics staff (as impersonated by me)

Fig. 10: Release planning

Meet your customer

Hypothetical Meals

Not at all organic and full of GMOs

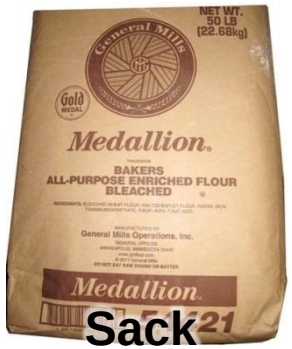
- Large, industrial-scale food manufacturer
- Production currently run by spreadsheets and manual effort
- They want a system to organize everything
- Each team is a contractor vying for their business

A brief primer on industrial food manufacturing

- The company **buys ingredients** in bulk
- Company **stores ingredients** until time for use
- Ingredients used on production line to make **food products**
- Food products sent to **distributors** around world
  - Distributors send products to customers (Food Lion, Dominoes, etc.)
  - Consumers buy end products

Ingredients

- Ingredients may be **frozen**, **refrigerated**, or **room temperature**
  - Must keep at this temperature state: store in **freezer**, **refrigerator**, or **warehouse/tank**
- Ingredients may come in various packages:



Sack

Pail


Drum


“Supersack”
(Flexible Intermediate Bulk Container)


Tanker truck

Railcar

Food production practices

- Need to rotate stock (use ingredients in order of acquisition) and mind expiration dates
- Logistics challenge: Need the right things in stock when we want to use them
  - Can’t keep tons of ingredients in stock we’re not using (wasteful of money/food)
  - Can’t let the production line stall (wasteful of time; and time is money)
- Need good bookkeeping to maintain efficiency

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.