Software Engineering

Introduction

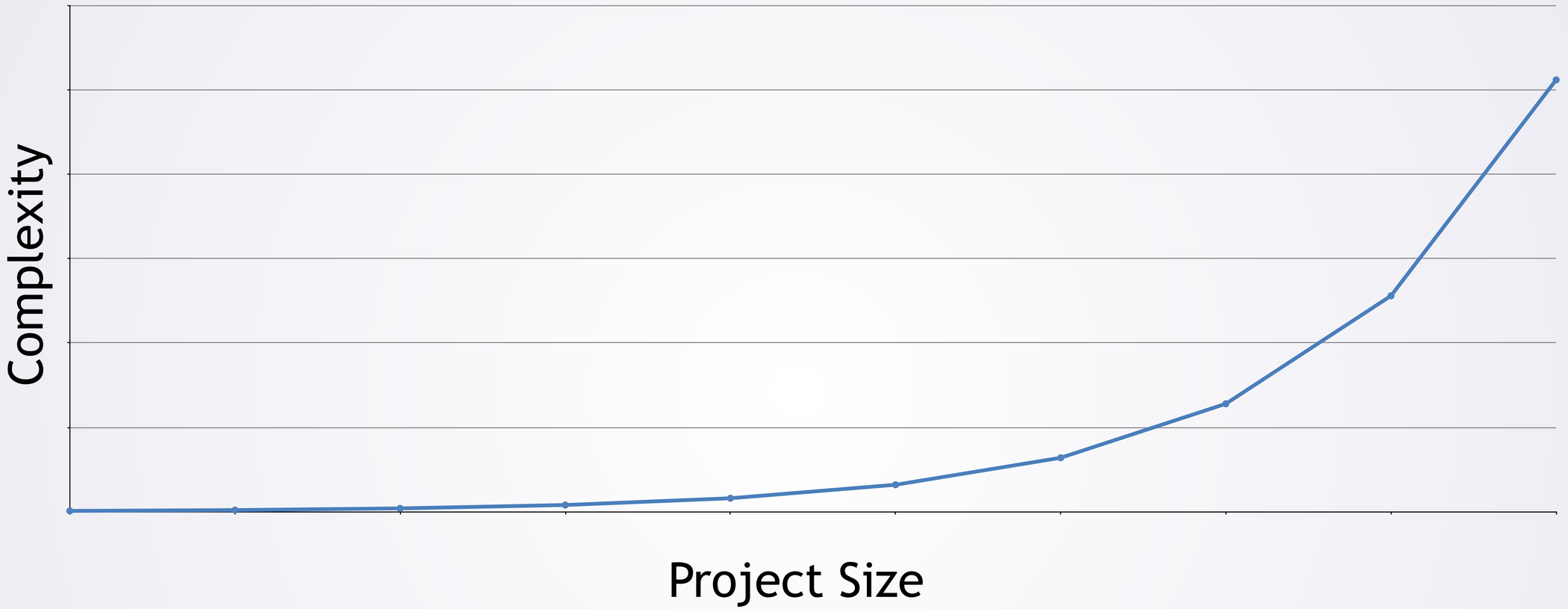
Duke
UNIVERSITY

Welcome to 651: Software Engineering

- Professors (will also be posted on Sakai)
  - Dr. **Drew** Hilton (Office Hours: 1:30-3:00 PM, Wednesdays)
  - Dr. Steven Noyce (Office Hours:)
- TAs (Office Hours to be posted on Sakai):
  - ....

Duke UNIVERSITY

What is Software Engineering?

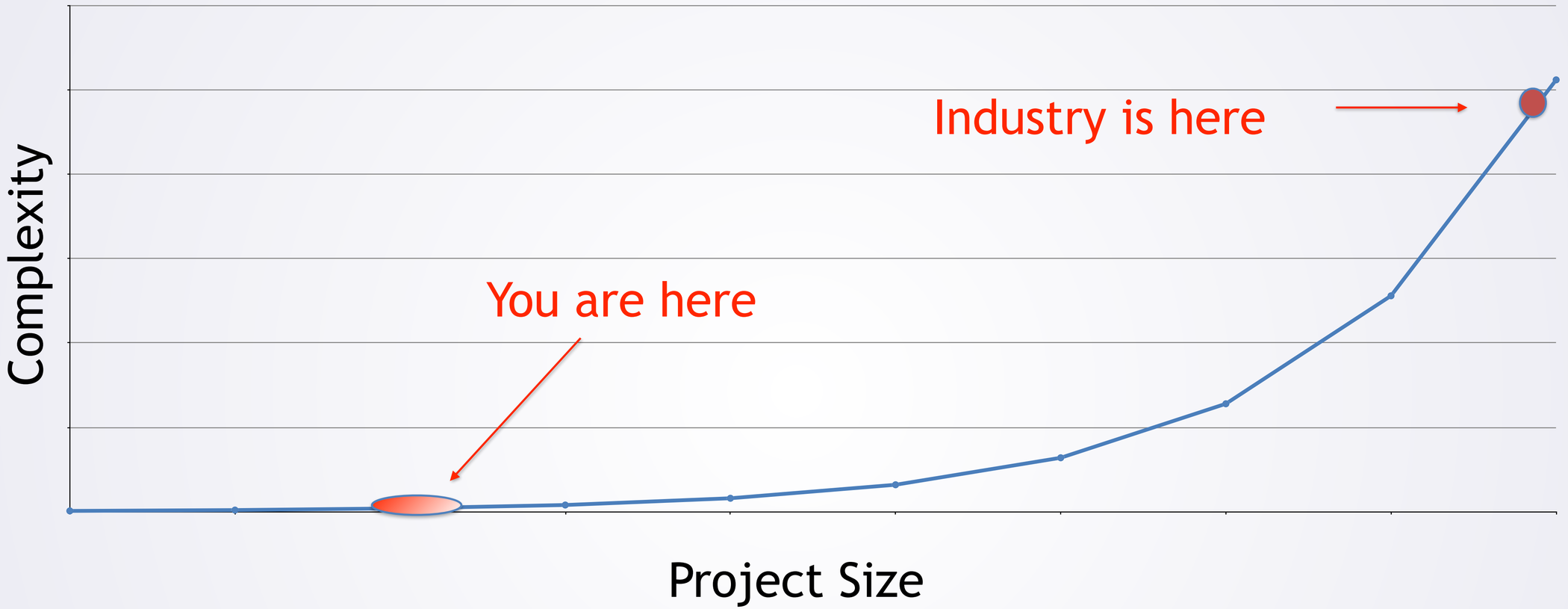


- Software Engineering is about **Managing Complexity**
  - Then again, so is pretty much everything in Computer Engineering..

What is Software Engineering?

- Current skill set: small projects, low complexity, one developer
  - A few classes with simple interfaces
  - Working from well-defined specs, often with design given

What is Software Engineering?



- Need to handle **a couple orders of magnitude** more complexity
  - Much larger projects, many developers
  - Specifications need refinement, must do significant design

Complexity: Big Hammer

Complexity

- Fortunately 551 + 550 have given you a big hammer to attack complexity…
  - Remind me what it is called?

Complexity: Big Hammer

Complexity

- Fortunately 551 + 550 have given you a big hammer to attack complexity…
  - Remind me what it is called? **Abstraction**

Abstraction

- Break big problems into small problems
  - Separate interface from implementation
- Tools you are familiar with for this:
  - Functions
  - Classes
- Now need to think about how to break large problems down
  - Into many classes
  - Possibly multiple programs (maybe on multiple computers)
    - May communicate by things other than function call (e.g., http)

OO Design

- One major topic of this course: OO Design
  - How do we split the task into (good) classes?
  - What are the interfaces between classes?
  - How do we make the project resilient to changes?
    - Real code changes.
    - Change is hard
    - Most reasons for what makes good vs bad code is change
- …but design is not the only aspect of software engineering…

Facets of Software Engineering

- Requirements Definition
- Design
- Implementation
- Testing
- Maintenance
- Working in Teams
- Process/Project Management

Facets of Software Engineering

- Requirements Definition
- Design
- Implementation
- Testing
- Maintenance
- Working in Teams
- Process/Project Management

I'm going to overview each briefly
As I do so, I want you all to think
about how **abstraction helps complexity**
in each topic.

Requirements Definition

- Customers often have a **vague** idea of what sw should do
  - "I need a program that lets students register for courses"
- However, you need a very **specific** specification with details
  - Should it be a web app? Mobile?
  - What rules does it need to enforce?
  - How does it handle full classes?
  - …

Design

- Design: determining what the pieces are and what they do
  - Pieces may be..
    - Services/programs
    - Classes
    - Functions
- Hierarchy: (also popular in 550, right?)
  - May do high level design (HLD) then refine
    - Split into services now, then design each of those
- Key: getting the right interfaces!
- Note: design does not generally involve writing code!

Implementation

- Implementation: given a small enough"piece" make it work
  - This is what you all are good at from 551
  - Here is where you write code.
- Piece too large? Refine design
  - Break into more pieces

Testing

- Remind us about testing from 551?
  - Think pair share about what you remember...

Testing

- Find **presence** of bugs.
  - Become more confident that software is correct as bugs harder to find.
- In 551, you did **unit testing**
  - Testing individual functions/classes
- Other kinds of testing we'll learn about
  - **Regression testing**: did you break it with this change?
  - I**ntegration testing**: do the pieces fit together?
  - **System testing**: does the whole thing work?
  - **Acceptance testing**: should the customer say "you are done"?

Maintenance

- After we are "done" we aren't really done.
- Changes, monitoring, and support after "done" are maintenance
  - Bug fixes
  - New features
  - Changes to how features should work
  - Monitoring behavior
  - Recovering from outages
  - …

Working In Teams

- So far: develop individually
- Real software: 10s to 100s (or 1000s..) of developers
  - 15,600 developers have contributed to Linux since 2005.
  - Internet estimates about 1000 developers on Windows 7.
- How do you work on a team of 20? 100? 500? 1000?

Duke UNIVERSITY

Process/Project Management

- Need to not just make software…
  - But make it **on time**
  - And correct.
- What process do you follow to get all this stuff done?
  - Especially with your team of 100 people…
- We'll talk about some common models, e.g.
  - Waterfall
  - Agile

Facets of Software Engineering

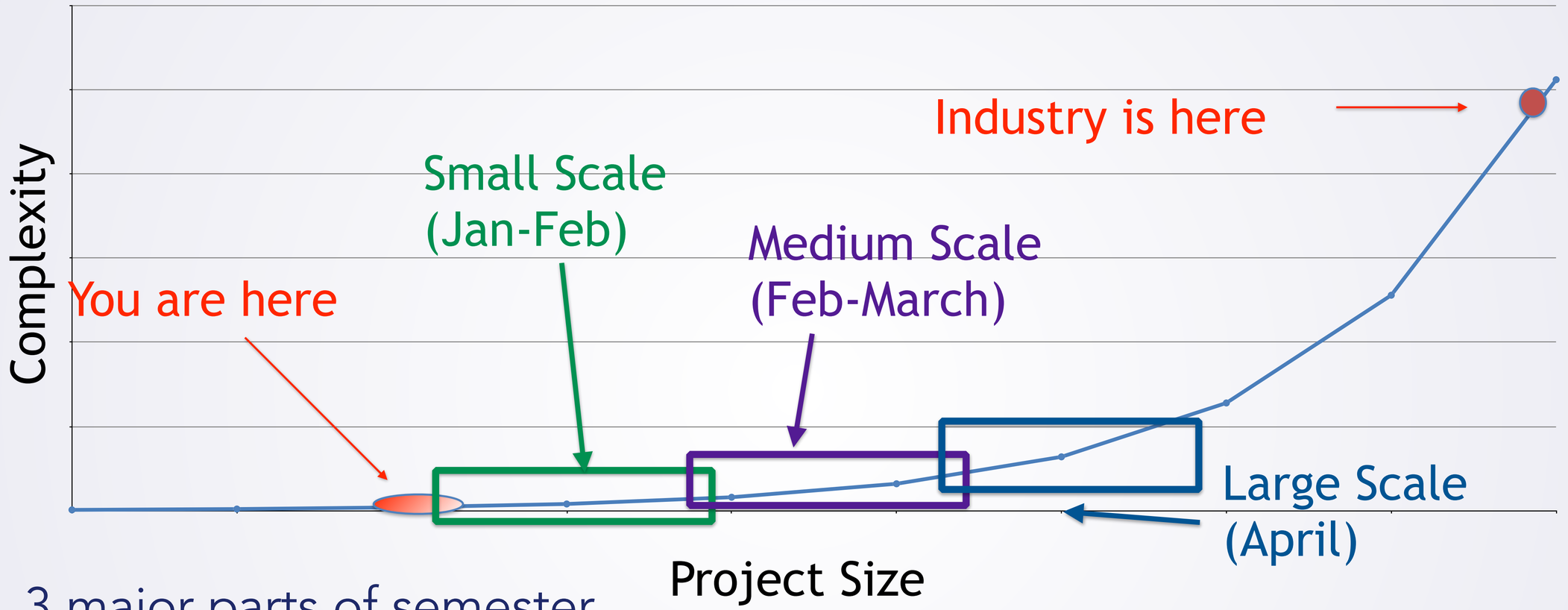
- Requirements Definition
- Design
- Implementation
- Testing
- Maintenance
- Working in Teams
- Process/Project Management

Think, pair, share!

You all thought about how abstraction helps in each, discuss your thoughts with the person next to you.

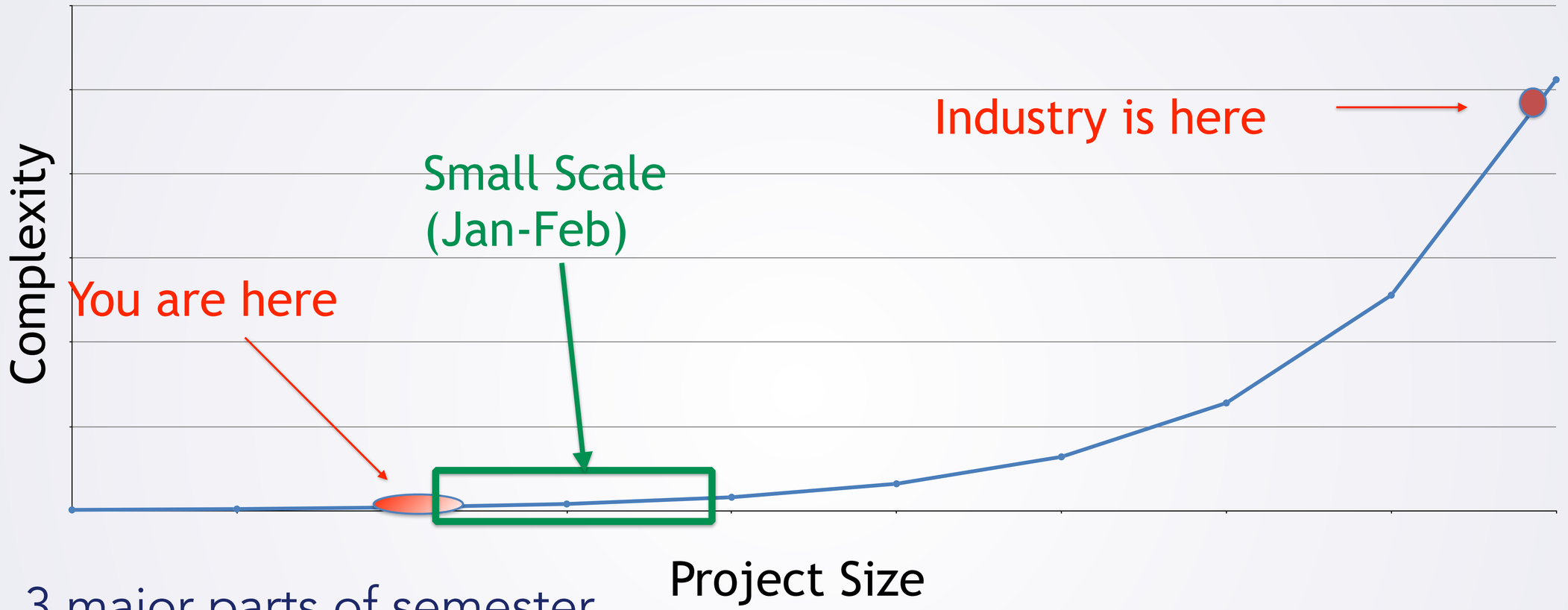
In a few minutes, we'll have people report back…

Roadmap



- 3 major parts of semester
  - Small Scale: A few classes (1-~10)
  - Medium Scale: Modules: many classes (10+)
  - Large Scale: Systems: multiple components/programs interacting

Roadmap



- 3 major parts of semester
  - Small Scale: A few classes (1-~10)
  - Medium Scale: Modules: many classes (10+)
  - Large Scale: Systems: multiple components/programs interacting

Roadmap From Here

- First: **key principles**
  - What guides our design?
  - How do we know if something is good or bad?
- These will underpin everything else we do
  - They are your **vocabulary** for **discussing** software engineering ideas
  - Discussion is key. I expect you all to talk
    - Why? **Think pair share**…

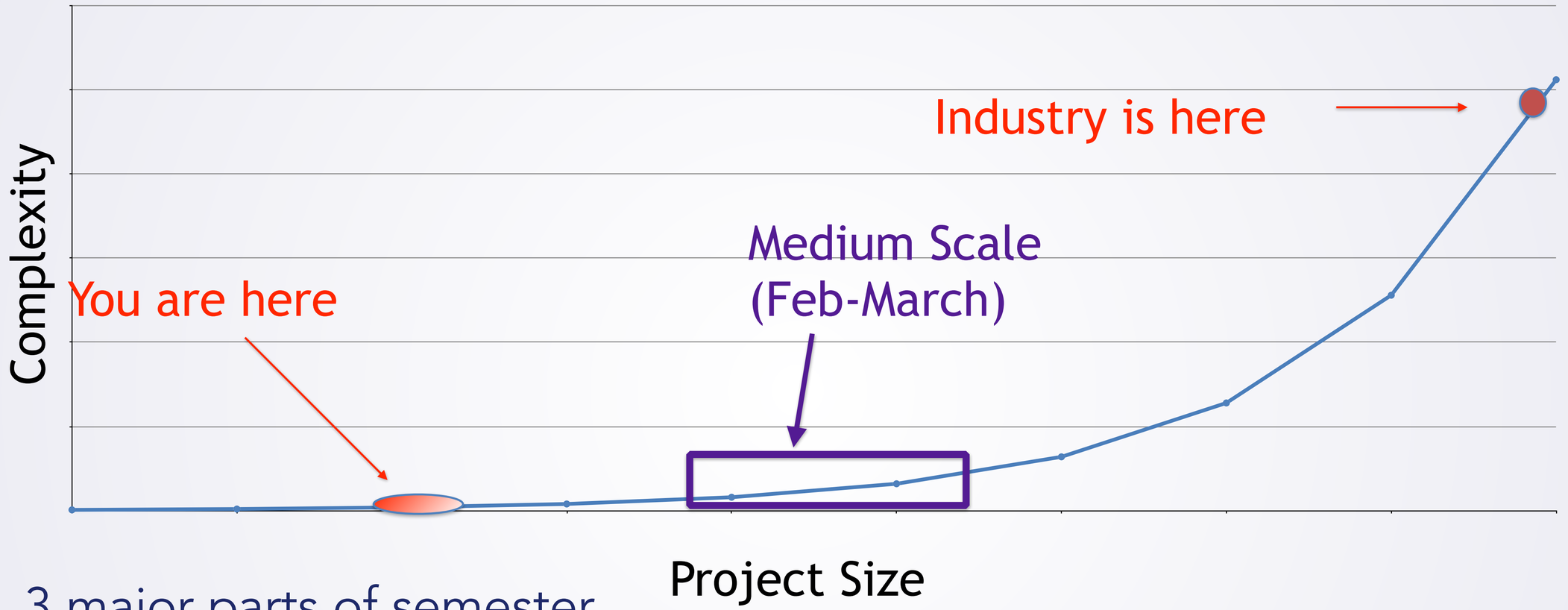
Discussing Software Engineering

- This is a key skill for your jobs
  - Advocate for your design. Give feedback on your co-worker's
  - Interview? Design questions…
- Analytical skills -> deep understanding
  - Nothing in CE is about memorization
  - Deep understanding: how, why?
  - Contemplate new things never seen before

Roadmap Cont'd

- **After principles**: Java for C++ programmers
  - Analyze language differences in framework of our design principles
  - Java came after C++
    - Why did they consider changes an improvement?
- **Then**: "small scale" software engineering
  - Process/project management
  - Design (especially design patterns)
  - Quality (testing, code review, technical debt, refactoring

Roadmap

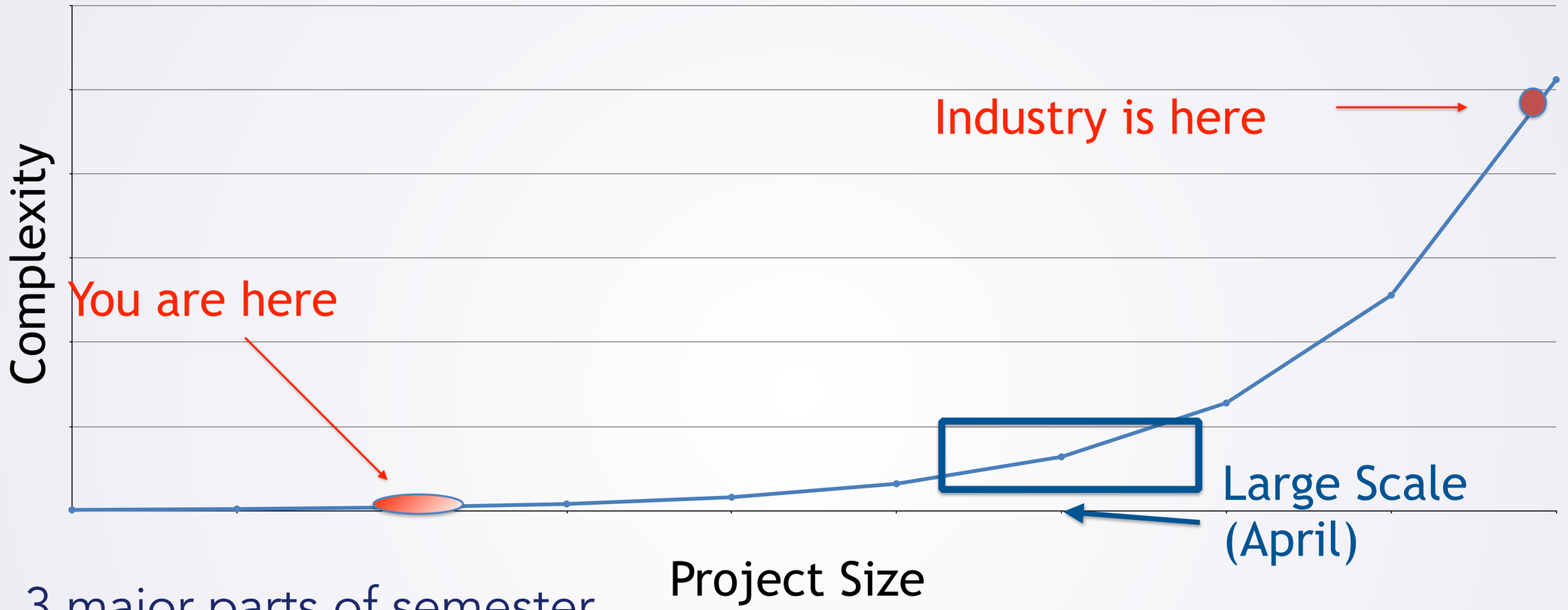


- 3 major parts of semester
  - Small Scale: A few classes (1-~10)
  - Medium Scale: Modules: many classes (10+)
  - Large Scale: Systems: multiple components/programs interacting

Roadmap Cont'd

- **After that**: "medium scale" software engineering
  - Process revisited
    - Teamwork
    - CI/CD
  - UI/UX
  - Designing modules + the interfaces between them
  - More testing!
    - Including breaking serialization across teams

Roadmap

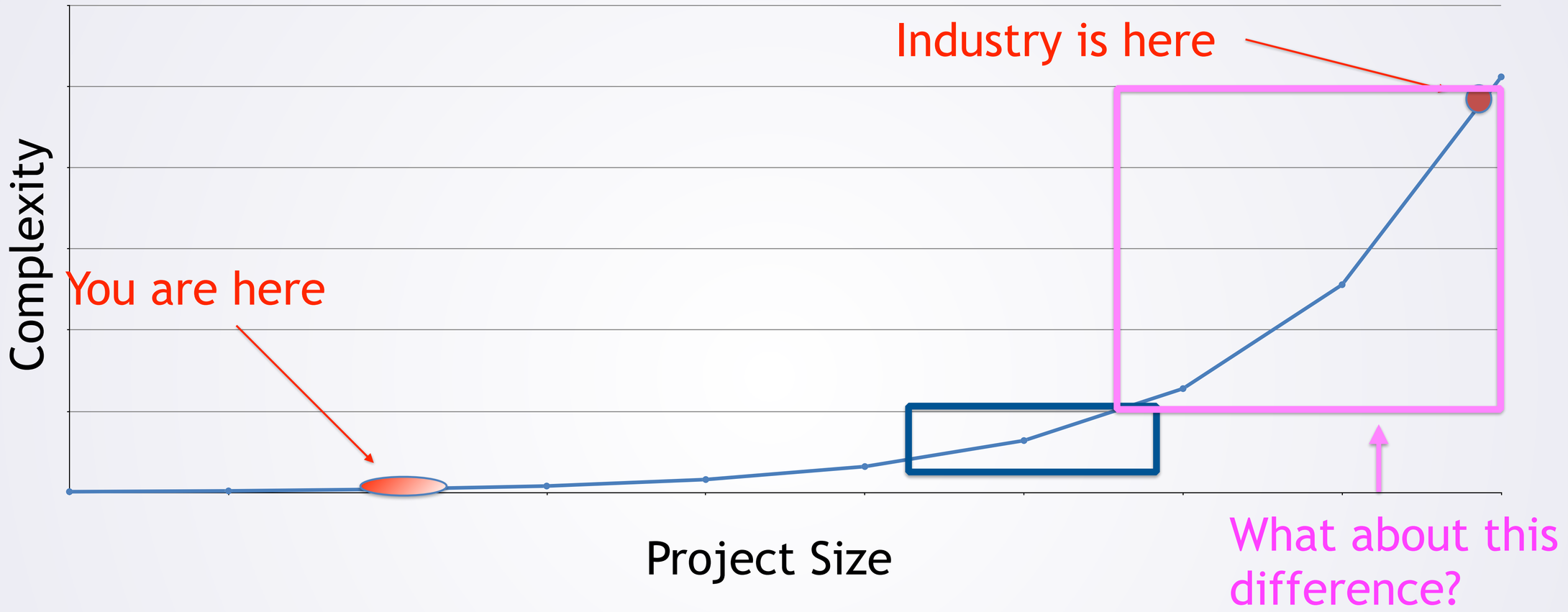


- 3 major parts of semester
  - Small Scale: A few classes (1-~10)
  - Medium Scale: Modules: many classes (10+)
  - Large Scale: Systems: multiple components/programs interacting

Roadmap Cont'd

- **Last**: "large scale" software engineering
  - System architectures
    - Monolith? Micro services? Event driven?
  - Components of large scale systems
  - Security
  - Maintenance + monitoring

Roadmap



- What about the remaining gap?
  - Don't really need any new techniques.. Same ideas, just at larger scale
  - Can't really have you all write a million lines of code this semester..

Logistics: Assignments

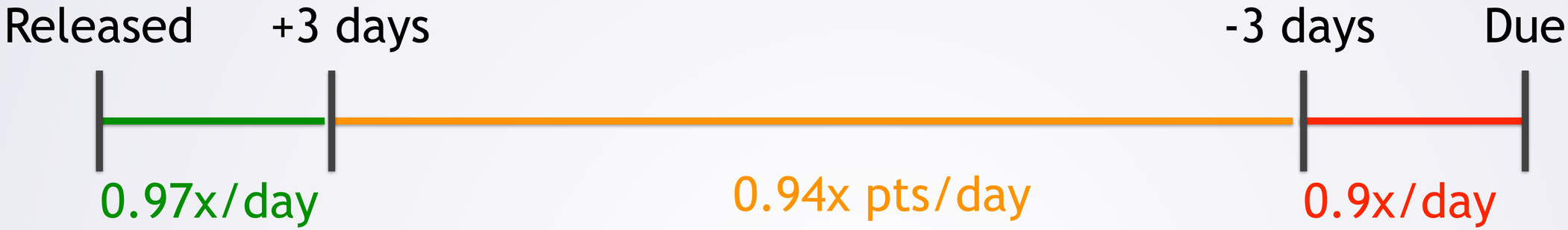
- You will have the following assignments:
- **Homework 1**: Individual Programming (Battleship)
  - Two halves: walkthrough + on your own
- **Homework 2**: Interview Potential Teammates for project
  - Industry panel on interviewing **TBD**
- **Team Programming**: larger software project in 3 parts
  - Teams of 3—4
  - Teams assigned, informed by interviews
- **Exams**: midterm + 3 small quizzes (two pages of notes)
  - 3x 20 minute quizzes instead of final exam

Duke UNIVERSITY

Programming Assignments: Start with planning

- **Homework 1 (Battleship):** we will model this planning for you
- **Team project**:
  - Plan at the start: explicit deliverable
  - …as a team!

Late Policy



- Late days requested within 3 days of assignment release
  - 0.97x multiplier per day requested
- Late days requested on day 4 or later, but before (due date - 3)
  - 0.94x multiplier per day requested
- Late days requested within 3 days of due date
  - 0.9x multiplier day requested

Other rules about late policy

- Does not apply to **exceptional** situations
  - In hospital, death in family, etc.
  - Documentation may be required
  - Contact professor ASAP
- Once you (or your group) "buy" a late day you can **NOT** undo it.
  - Can't ask for 5 late days on day 1, then later say "nevermind, only need 3"

Why This Policy?

- **You all tell me**: why do I have this policy?

Why This Policy?

- **You all tell me**: why do I have this policy?
- Plan carefully and accurately!
  - If you realize you need more time on day 1, late days are "cheap"
  - If you ask at the last minute, they are expensive.
- Start early!
  - Realize you are behind? Make a plan to catch up or ask for late day **now**.
- Gives some flexibility
  - "Oh my gosh but that is due at the same time as [….]"
  - Plan!

Duke UNIVERSITY

Student Model of Project Management

1/14 650 Assn Released

1/15 568 Assn Released

1/24 651 Assn Released

2/4 650 Due

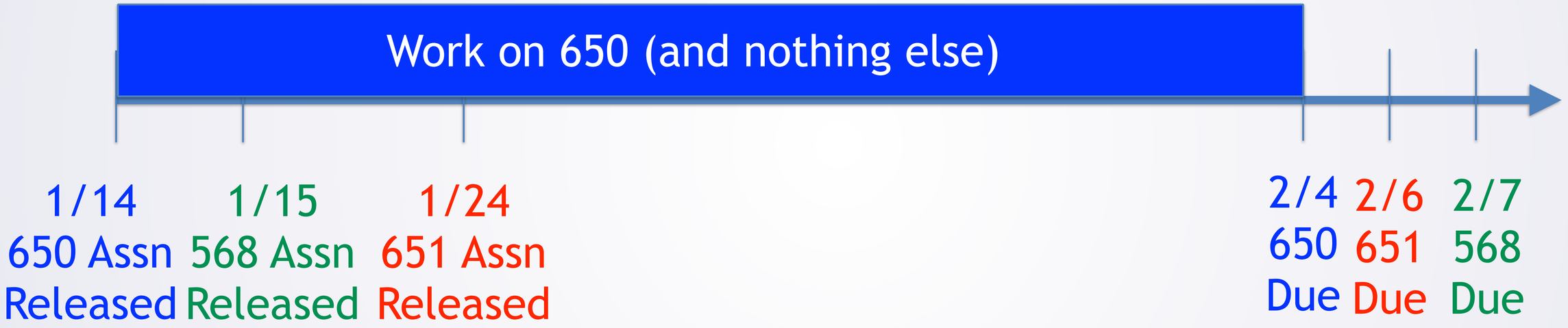
2/6 651 Due

2/7 568 Due

(Hypothetical dates)

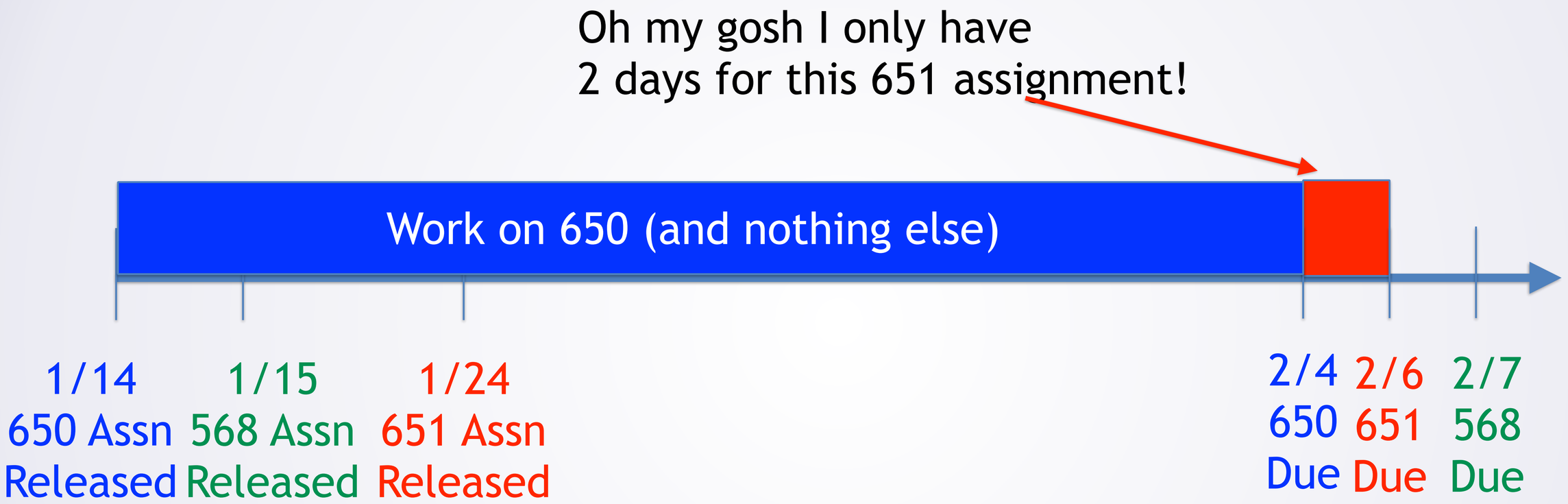
- I've learned this is how you all manage your time.

Student Model of Project Management



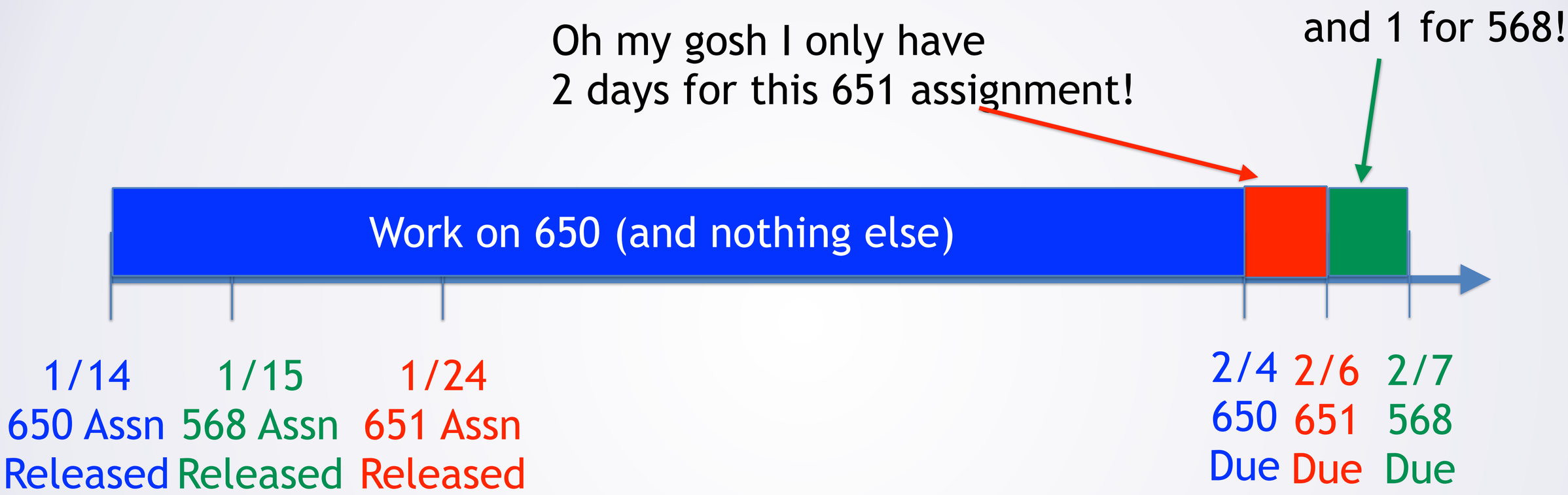
- I've learned this is how you all manage your time.

Student Model of Project Management



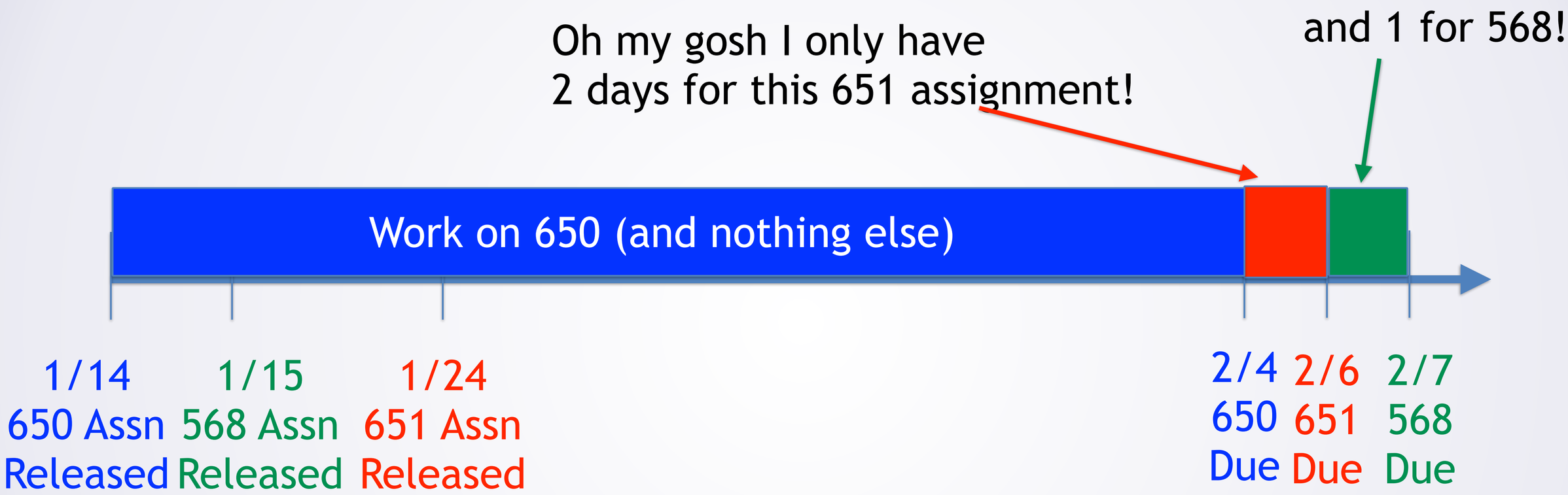
- I've learned this is how you all manage your time.

Student Model of Project Management



- I've learned this is how you all manage your time.

Student Model of Project Management



- I've learned this is how you all manage your time.

DO NOT DO THIS

What You Need To Do Instead

DO THIS INSTEAD

1/14 650 Assn Released

1/15 568 Assn Released

1/24 651 Assn Released

2/4 650 Due

2/6 651 Due

2/7 568 Due

- Why should you do this?
  - After all, EDF scheduling and context switch overhead make other seem good, right?

Student Model of Project Management

Oh I need to go to OH in 2 days because I realized I don't understand X…

I'm stuck until my group member does Y…

1/14 650 Assn Released

1/15 568 Assn Released

1/24 651 Assn Released

2/4 650 Due

2/6 651 Due

2/7 568 Due

I just have a mental block about this bug and need to step away for a bit.

- You will encounter delays..
  - Switching between tasks is good.

Other Things About Time Management

- If you plan for milestones across weeks, you can recover
  - Get a bit behind, time to fix
- If you plan for everything to happen in 2 days, you cannot recover
  - Falling one hour behind is a catastrophe
- In the Real World, you will need to handle multiple project at a time
  - Probably at least 3.

Tips for This Class

- Many students make this class take MUCH more time than it should...
- Here are three tips to avoid these problems

Tips for This Class : 1

- Incremental Testing.
  - Write a small amount of code (**< 10 minutes**)
  - Test it
  - Repeat
- Why does this save time?
-

Tips for This Class : 1

- Incremental Testing.
  - Write a small amount of code (<10 minutes)
  - Test it
  - Repeat
- Why does this save time?
  - The less untested code you have, the easier it is to debug
  - Writing large piles of broken code = wasted time
- 100% test coverage should not be an "after thought"
  - Write, test, get 100% coverage, write some more..
  - We model this for you in Battleship part 1

Duke UNIVERSITY

Tips for This Class : 2

- Start with a **minimal** working system
  - We'll talk about this in more detail in a couple weeks
  - Start with the smallest piece of software that you can
    - **Battleship**: read one coordinate, place ship, print board
      - That is all
  - Now you have a working program
    - Missing many features + complexity
  - Everything else is a feature you add --> keep it working
- How does this save time?

Tips for This Class : 2

- Start with a **minimal** working system
- How does this save time?
  - Avoid team members writing code that does not fit together
    - Everyone has to fit code into working system
  - Keep it working = keep it tested
    - Avoid writing large piles of broken code (as before)

Tips for This Class : 3

- Ask for help
  - Don't stay stuck: maximum of 15 minutes stuck
- **Bad**: "I spent 5 hours Googling this error before I gave up and posted on Ed Discussions"

- **Note**: if you do good time management, you can switch between tasks (other classes) while waiting for an answer!

Academic Integrity

- You are expected to you own work in this class.
  - After all, you are here to learn.
  - If you can't do this, you can't do the job you want.
  - Your friends/the internet won't do it for you…
- 4 policies…
  - **Individual Programming**: **free to discuss, but write own code**
  - **Interviews**: **free to discuss, but do your own interviews + write-ups**
  - **Team Programming**: **team**
  - **Exams + Quizzes**: **individual**

Duke UNIVERSITY

Academic Integrity

- **Individual Programming**: **discuss, but write own code**
- I expect you to write your own code.
- Do not show your code to other students or look at anyone else's code
  - Includes finding code on Internet
- Can have discussions such as
  - "I don't understand Factory pattern, can you explain it?"

Academic Integrity

- **Interviews**: **discuss, but write own code**
- Feel free to discuss ideas with each other
- …but you really need to do your own interviews

Academic Integrity

- **Team Programming**: **team**
- Work done entirely by your team
  - Discuss with team mates
  - Share code with team mates
  - Do not look at other team's code, or on Internet
- Think of this like a company:
  - Do not expose your company to IP infringement lawsuit!

Academic Integrity

- **Exams**: **individual**
- Do not discuss at all with other students
  - Can ask professor/TA clarifying questions
- Open notes

Assignment Particulars

	Percent	From	To
Class Participation	11	All semester	
Individual Programming	15	1/20	2/10
Interviews	10	2/10	2/21
Midterm	10	3/2	
Team Project Evolution 1	15	2/24	3/24
Team Project Evolution 2	15	3/24	4/10
Team Project Evolution 3	15	4/10	4/28
Quiz 1	3	3/9	
Quiz 2	3	4/4	
Quiz 3	3	4/18	

Letter Grades

- Letter grades follow the standard 10 point scale with 3 points for - and 3 for +
- A+: [97, ∞) A: [93, 97) A-: [90, 93)
- B+: [87, 90) B: [83,87) B-: [80,83)
- C+: [77,80) C: [73,77) C: [70, 73)
- F: [0,70)

First Thing To Do

- Read All of Programming, Chapter 31: Java
  - Please do by 2 classes from now