CS 130 Notes

W 1 M Lec 4-3-17

- Problem in the sense that this theory keeps getting revamped
- Practice - closer to the code where you are actually building stuff and build things more accurately and reliably
- Software construction
- What were the software construction practices at Microsoft (McConnell)
- He hasn't written a book to describe what Microsoft has done now so let's do what we have now
- More practice - learning by doing
- Do a project and we want to make this project more like real world software projects than it is normally possible than it is at undergraduate courses
- Recruit projects from the outside world and make presentations to you and try to recruit you.
- **Groups of 4 or 5 people**

- One of the hardest parts of doing software engineering isn't the coding or construction, but rather what you are supposed to do.
- The art of doing requirements
- Professor tells you what to do and you sit down and solve the problem
- Coming up with the problem statement is often the most difficult part of the entire project
- Revise the problem statement and this is something Eggert hopes we get.
- Teamwork is vital!
- Takes a while to build up a team of 100 people and we aren't quite up to that level, but Eggert will be happy if we get a functioning team of 5 with everyone contributing.
- If we can do that, we will be doing better than most commercial developers.

Paul Eggert's background

- UCLA student
- CS professor for 3 years
- Spent too much time writing grant proposals rather than doing interesting work
- Startup (Logic programming) (3 years)
- Unisys
- Another startup
- Had 35 people and there are firefighters on a project
- 15 years Lecturer at UCLA
- **Write free software**
- **Emacs, grep, …**

This course is overly ambitious

- CS departments in this country in which you can get a B.S. in software engineering
- UC Irvine (B.S. in Software Engineering)
- Take 10 courses in software engineering!
- They have one course just how to do requirements and another course on teamwork
- Each week in this course corresponding to an entire quarter course at Irvine.
- We can only cover a subset of software engineering
- Which subset?
- What's left out?

- There can be an overemphasis on things and keep an eye out for stuff that Eggert is underemphasizing

Course catalog

- structured programming
- Have a discipline for writing code that is clear and easy to understand
- gotos get in the way of that!
- Have the structure of your program reflect the structure of your problem
- Are you some relic from the 1970s?
- This part of the catalog has been written in the 1970s and no one has bothered to update it since
- program specification
- How will a program behave and what will it do without necessarily having the program it had?
- Have an independent way of knowing what a program will do besides its source code.
- Coming up with a good spec is harder than it may seem
- program proving
- You write proofs about your program - mathematical notion
- Prove that your program will work and it won't work
- Automobile proving grounds
- Testing your program (QA is analogous in the real world)
- **modularity - splitting software up into pieces and this runs into a lot of the courses)**
- **Covered in programming languages**
- **abstract data types**
- **composite design - build an application out of designs elsewhere (specs)**
- **Make your specs modular as well as your code**
- Software tools - programs that let you write other programs (development)
- Software control systems - programs that let you manage other programs while they are running (operations)

- Have one phrase that people think is very important
- program testing
- team programming

To some extent, this list is very old-fashioned with structured programming
- To some other extent, the problems we ran into 40 years ago are still problems we run into today
- We will look at some classic problems and the basic underlying issues will come up over and over again

Grading
- **7.5%** quizzes (starting next Wednesday, there will be quizzes everyday)
- 4 questions, closed book
- Take the assigned reading
- Take a random number generator, and write a question about that
- 15 quizzes and the total weight is 7.5%
- **2.5%** requirements homework
- When you look at the most famous failed software projects, they often had bad requirements!
- Some of your customers are quite computer literate and they will help you by coming up with the requirements for you
- In order to make sure you get to do requirements, this is for people whose main projects were already specified for!
- **40%** Main project
- Have a requirements document for your project and code that is deliverable
- The details will be discussed by the TAs
- (details - see TAs)
- **15% Midterm (Wednesday of Week 5, May 3rd)**
- **35% Final**
- open book & notes
- Pain to grade an English paper because to some extent, the real world is kind of messy for our exams
- Give out sample exams

Deadline policy
- Typical Eggert late policy
- June 9 is the drop dead day
- This is the absolute last day to submit your assignments

Start the project early and do work even if you are not sure it is right
- Learning a lot from each other rather than learning from Eggert

Plagiarism

- We have situations in this course where we have competitive teams with the same project and they are trying to solve a problem in different ways
- Can be tempting sometimes to get in the other team's way or copy work from the other team that you get from your client.
- **Please avoid these temptations**
- Conduct things in an ethical way
- CS 131 - write 100 line program and all of those have to be your own
- Among those thousands of lines of code, maybe 1000 lines were written by someone else
- That is okay as long as you give proper credit to those who wrote your sources
- Look at all this code that I wrote and make it very clear when you submit your work which part you wrote and which part other people wrote.
- Put a by-log in each of the modules that you write!
- Written by Matthew Lin

- **All work submitted must be your own, except for items explicitly labeled as imported**
- They are smart people and everything will be good!
- Do NOT wait until the end to submit the work and they won't pull a fast one

Software
- A function where you have an input that will give an output
- Counterexample: **Pocket watch**: an output has a couple of hands here when it tells you to do a break
- NOT software
- Computers are complex finite state machines
- Changing states from input to output
- Variable part of your system
- Important word that belongs here
- When you write a C program and you have local variables in your program, those local variables values vary
- **Easily reprogram your device in order to do something else**
- The "soft" is software because it isn't hardwired!
- In particular, the consequence of this is that software is NOT manufactured in the traditional sense!
- Nor does it "wear out" in the usual sense
- A lot of funny things in your car that you drive to school are NOT designed to make your car faster but they want it so that it is easier to build
- A lot of problems in traditional engineering that are NOT present in software engineering

Failure rate (Hardware)
- A traditional curve over time

- Tend to come from an accumulation of small things (flash drive slowly wears out)
- Eventually, it starts to wear down

Failure rate (Software)
- More like a roller coaster ride!
- Come from bug discoveries and discrete events

Software
- Most software is custom-built
- Someone will write the software just to solve a particular problem
- Most people are NOT working for Facebook-central or Google-central
- They are running their own custom-built software
- In contrast, our cars and cell phones are mostly mass produced
- A lot of the engineering happens to deal with cars and these don’t work with software
- We are starting to see the custom-built approach in hardware nowadays!
- Software isn’t going to take over the world this week, but maybe in a few years.

Economics Question
- $100 in 1994, how much is it now? Presumably a greater number in 2017
- About $400 today!
- A software failure in 1994 DMV
- California Department of Motor Vehicles
- Spent $44 million which is about $220 million today on a computer system that would never work
- The software maker isn’t responsible

1998 project started
- Tandem Cycle DB runtime
- Fault-tolerant model that was highly available
- **Central system became overloaded!**
- Specified a machine that was big enough to handle the load and when they actually tried to run it, it did NOT work and there was just too much of a gap between them and the design was built on a single central server and the whole system design was wrong!
- Eggert had to pay for it because he is a California taxpayer!

Shellshock bug reported 2014-09-24

```
env cat=‘(){echo bar;}’ sh
$ cat /etc/passwd
bar
```

- This bug had been in the Shell for 25 years!
- Allowed hackers to compromise systems!

There are still many websites out there that are vulnerable to this bug

- Software development process that wastes a huge amount of money
- Most software projects fail!

GNU Emacs
~ 4 million lines
~ 400 thousand lines of C
19,000-line pitch

- Go into the C interpreter and change key components of emacs
- Should you install the patch under the following constraints?

Decisions like this can lead to these kinds of problems!

- When you are doing software engineering, you can make practical decisions that lead into these very visible failures later on and decide using some algorithm later on
- **Eggert decided NOT to put the patch in**!
- Eggert didn't want to risk it!

Software Engineering

- Implies that the proper viewpoint for addressing this bug and that problem is an engineering question
- You should be able to go up to the 6th floor of Boelter to the Office of the Dean of Engineering at UCLA and ask Dean Murthy questions to help you engineer your software

Battle between engineers vs the hackers

- Engineers want to take meticulous planning to make sure everything works!
- We need to solve problems and hack things out the door and worry about engineering later

- This sort of battle has been going on for decades and the term software engineering was invented in 1968 or 1969
- This was sort of in response to the first software crisis and the hackers had ruled the day and people wrote code and built great stuff with it
- Look with disdain upon traditional engineers and bureaucrats who couldn't get work done.

Problem was first recognized in the defense community to learn how to control radars and the problem of the hackers as a severe one to try to figure out how to bring software development under control

- Come up with a definition of software engineering

- 1960 NATO conference: about both development time and runtime!

- These goals coincide where writing software cheaply will cause it to run fast
- More often than not, though, development time and runtime will be **competing goals!**

• Software engineering is about coming up with abstract principles where you learn from mistakes and write it down to tell other people.
• “Sound” principle vs “unsound” principle
• Proof is just a way of leveraging off of testing and you might want a mixture of these two!

Establishment
• Why did they bother to put that word in the definition?
• It means we want to have a bureaucracy and we have a building with Greek pillars on the front and a bunch of paperwork on it
• Make sure that we write down what all these principles are

Use
• Get everyone to use these principles and we have a police force!

Real machines
• Computer Science Theory
• Talk about fun stuff - Turing machines and that sort of thing
• What this phrase is supposed to say is that we are NOT doing theory
• If it is theoretical, it is NOT engineering!
• We want our stuff to run on real machines, and if it is NOT running on real machines, we are NOT interested
• Advocating bureaucracy and stepping away from theory is to be real practical and insist on this bureaucracy and police so that engineering principles will work

Alternatives to software engineering
• Add people to a late project
• Outsourcing - send it to people in Bangalore (shoutout to my man Ashvin Vinodh!)
• Engineering process is done in Bangalore rather than the U.S.
• Probably more complicated than before because you now have more people involved

Goal is that after every lecture Eggert gave, Eggert talked about obvious stuff today
• Lecture should be obvious!

W 1 W Lec 4-5-17

Project 1
The Aerospace Corporation

Mustafa Alammar
Web Applications Developer

- A lot of the work involves R&D
- Made in the 1960s to work with the Air Force
- First to launch GPS satellites and get a global positioning system set up.
- Big achievement in their history
- R&D on space systems, spectral imaging, and small satellites
- Discover failures before they occur in a real launch because satellites can be really expensive
- Responsible for helping SpaceX with their certification process as a launch platform and they succeeded to get certification
- NRO satellites and U.S. Space Systems

Headquarters is in El Segundo - acts as a secondary room for scientists and engineers help launch their rockets

- Sensor measurements, check weather patterns, testing before the launch
- Watch a laugh and have a giant screen and you feel like you are at a launch site

Civil & Commercial arm - work with SpaceX and NASA

- Attach it to a Go Pro and this can help them run costly inspections for way faster and way cheaper.

Application Development

- Little group within Aerospace that works on internal projects like the Internet and External site
- Web development related to Aerospace

Last Year's Project

- Many sub-projects
- iOS client
- website
- server
- Integrate touchID
- Made everything secure an open source!
- Didn't rely on hiding of technology but it was still secure!
- Helped to maintain security because people are nice enough to reveal any vulnerabilities

This Year's Project

- Internet of Things (IoT) - smart devices and popular media
- Smart houses are becoming much easier and much more widespread idea and a lot of people are implementing these ideas day to day.
- Network of devices
- Cool, convenient, and efficient!
- Program your thermostat to use your energy much more efficiently

Your Mission

- Find a unique or novel way of connecting the physical world to the Internet
- Build a smart device in anyway you want
- Very open-ended and take a piece of technology to be useful
- Made an innovation lab at Aerospace and this is what I would like to do.
- Why?
- For whom, if you have a customer in mind, how would we improve people's lives?
- Look at the tools we have available and we have a great device that is great for learning!
- **Good for showing it off on your resume**

The (Optional) Arsenal

- Given a bunch of equipment like Arduinos, micro controllers, accelerometers, Alexa, etc.
- An array of stuff to play around with
- Anything you want on the Internet - so much out there that is available to use!
- **You can build something on your phone - smart device**

What if we need more?

- $30 budget provided per team
- There should be enough for each team

Dos and Don'ts

- Do
  - Use version control like git
  - Do use GitHub, Trello, or Asana for project tracking
  - Document as you go
  - Regularly (virtually) meet up and share progress
- Agile/Scrum
  - Use slack or other team chat
  - Use open source libraries

- Don't
  - Wait until the last minute
  - Build everything yourself
  - Forget about the customer!

Project 2
Jens Palsberg

- The textbooks are pretty good and the lectures are pretty good

- Once people go home and they have to do the homework, it is like sand between the fingers

LL(1) Academy
Khan Academy for LL(1)

- LL(1) concept is something that can be online
- Tailors questions that people use to learn the concept and this should be fully automatic
- This whole process of making and presenting questions and generating everything can be automated
- This is what Palsberg does automatically when he teaches the class
- Hopefully, people can get a much better learning experience

Khan Academy has a huge bank of questions and this is fantastic when Palsberg's son comes across something he doesn't know well

- They wrote them and store them in a bank

For LL(1), which has to do with compilers and grammars, you can automatic this process!

Mathematics II

- For the purpose of learning what is needed, we need to start small
- For the purpose of LL1 Academy, we need small grammars to begin with!

Color coding scheme

- Make people excited about their progress

Midterm Question

- Good way of testing the material because you cannot guess your way through

LL(1) Academy
Launch date: June 7, 2017

Q. How much of it is student's coming up with problems themselves?

Project 3
Blue Stream

- Startup that is taking ideas from the Anderson Accelerator
- Bring medical devices to the U.S. market faster

Experienced Team

- Soumya Mahaptra, Founder and CEO
- Manages IT and Business Transformation portfolio for Fortune 100 Healthcare Clients
- Tae Macias, CFO

- All of them are in the Anderson MBA program
- 8-20 year experience in respective verticals
- Got her CPA for small to midsize businesses

Who Are We

- Blue Stream provides software platform to reduce the time and cost to navigate regulatory pathway for launching Medical Devices.

Why is it complicated?

- We can save a lot of money and if there are 500 new medical device startups every year, we can make it a more sustainable model

Product Suite

- Content submission manager and exchanging information to automate the process of this application
- Analagous to Turbo Tax!

Users are NOT tech-savvy!

- Things take about 2-4 weeks and we need to do some initial proof of concept and feasibility tests to less than a week.

Product Development Principles

- In terms of designing, there will be components of designing the architecture solution
- These users are not technically savvy

Product Execution Methodology

- 0 - Share storyboards from users that Soumya is working with
- 1 - Immerse
- 2 - Plan & Prepare - build a high-level proof of concept framework where you can test things easily
- Stock programs and give it inputs and see if this is how you want it or do you want to refine the model
- 3 - Innovate/Design - create a product that fits the user's needs
- 4 - Prototype
- 5 - Assess & Reflect
- 6 - Pilot & Learn

A Full-Stack Product Development

- UI screen design
- Front end development
- Black end development
- Core functionality
- Analytics/Machine Learning
- Databases

- Access to Prototyping Tools and AWS for the best learning experience!

What to expect from the Project

- Solve real-world problems in healthcare in U.S.
- Gain hands-on expertise on emerging tech
- Full-scale software development lifecycle
- Real-time user feedback from primary research

Product Development Principles

- Discover - the content discovery phase
- When you are initiating a project and what is doable (feasible) and what is NOT!

Project 4

Chris St. Jacques

- Participated in this class for about 6 years
- Graduated in 2012 and worked with Shopzilla for the last 5 years

What its Connexity

- Technology-driven marketing solutions company
- Probably second to only Amazon to drive sales to our different partners

Social Shopper

- Use the Connexity Catalog API to build your own social focused shopping website
- Utilize social APIs to supplement data
- Instagram
- Pinterest
- Tumblr
- Develop a back-end and front-end component of shopping website

Catalog API Sample

- XML and a dump of data and there are targets and images

Technologies

- Enterprise level technologies
- Java
- Spring MVC

Logistics

- Mentorship from former CS 130 students
- Direct experience with enterprise level engineering practices
- Setup guidelines
- On site development workshops
- Git/Github
- Various methods of communication between teams and Connexity

Mustache

- Integrates nicely with Spring Boot but you can use it to iterate through objects and some deeper functionality than HTML
- Looks and feels just like HTML and this is going to be an easy transition to Mustache

W 1 Dis 4-7-17
Week 1
CS 130: Software Engineering

TA: Muhammad Ali Gulzar
WR - 1:30 PM - 2:30 PM

Finish up by 11 AM!

Software Requirement Specification

- Not really talking about technical details but rather the high level overview of the software
- Functionalities
- What the function does when the user responds to it
- High-level agreement between client and developer

Homework is also on this slide, so it will be at the end!

- What will the software do in certain cases?
- This kind of information is very detailed information, so we won't miss out something that is ambiguous!
- Define the constraints of the software application you are building!
- Define that upfront in this document and the developer will know what he or she will be implementing for those features
- NOT really offering technical or business issues!
- Technical details is done in the later part of these issues
- There is something that I don't know and go back to the same document for the feature that I am implementing
- A list of all the requirements that the software would do!

Benefits of Writing an SRS Document

- Whatever you are implementing, you have a base document that you always go back to and figure out the feature to implement.
- Organizes the information to write software
- Places boundaries around the problem
- Don't implement something that isn't really necessary
- Solidify ideas here
- An app that you want to build for interior design
- Come up with a technical solution for that particular functional feature

Things that can be avoided by a well written SRS Document

- Once you have these precise sets of requirements, there is no ambiguity between what you are building and what your client wants you to build
- Once you are writing this document, you would know these are the requirements I need to implement
- 5-6 developers can work on a software and you would spend 2-3 weeks on something and you need to iterate

Components in SRS Document

- NOT the exact template recommended by IEEE but it is pretty solid.

0. Application Overview

- Define what are the roles and responsibilities of the users
- Important part of the whole document
- In the end, you have these users who interact with the software and the application
- Yelp - user entering the data into the app and a DB administrator who is a business owner and you have a different set of rules
- Liking or commenting on some reviews are features you can do
- These are the roles of the users you want to define in your document
- Define interactions with other systems and these all the time interact with something
- Part of a big picture, so define those interactions

0. External Interface Requirements

- If a user interacts with the UI, you need to have a complete flow of the UI
- Once you have a deeper understanding, you know what the requirements of your project is
- Workflow of the whole interface - how you interact with the hardware is also a big component
- Hardware won't be a big part of this, and **it certainly won't be a part of the homework!**
- Software Interface
- APIs and libraries i.e. Android
- Communication Interface
- Interaction with other network devices such as a network protocol
- This tries to be as generic as possible because software communicates with hardware

0. Functional Requirements

- Gist of whole document
- What are the functions or requirements for this?
- You need to define how a user can reach to that point if he or she likes something
- Define what the user roles should be in these cases

0. Non-Functional Requirements

- Performance requirements
- How responsive your software output should be
- Need some ranges for which the final throughput should lie
- Reliability requirements
- Most of these online services like AWS have these service heavy requirements will always be available
- Security requirements
- You want the best protection to ensure your users don't get fucked!
- You could have a function requirement for implementing an encryption scheme
- Software quality attributes (Maintainability)
- Related to how the client would want his software to be written

Example SRS
Software Requirements Specification
- Amazing Lunch Indicator

- Build a GPS based mobile application and this will be based on user's current position
- This document is writing too many pages and it is a nightmare to create a document for everyone

Restaurant owner can add or delete his restaurants in the application

User Interfaces
- If he or she clicks certain buttons, what would the response be?
- Don't need really graphical pictures, hand-designed layout is fine!

3.2 Functional requirements
- The user should be able to register through this mobile application, and this requirement will let you register things here
- When you write this requirement, think about necessary information you would want from your user
- This really concretized the functional requirements

Grade it depending on the environment

Class Activity
- Web app to decorate/furnish a room with Ikea's products
- 3D simulation of the room
- Search capabilities
- Customization based on
- Color
- Material
- Size
- Add items to the room when happy

- Move, edit, add more items
- Suggest similar products that could be found on Amazon
- Add products to the cart and purchase them

User Roles

- Ikea customers that are actually interacting with the customer-side of this application

Constraints (hardware, software, etc.)

- Camera quality of where the person stands when they take the panoramic picture
- Include a way to scale the image

Functional requirements

- If the user clicks and change the color of the furniture, if the user clicks on the button, then the color of the furniture will change and there are sub-requirements and design a mock layout.
- Are there a list of colors i.e. RGB values and things like that
- Remove a furniture item and define a use case
- Submodules within that functional requirement that would lead to a non-functional requirement

Non-functional requirements

- Performance could be based on server load and things like that
- Should be interactive in this case

5-10 pages so follow Muhammad's guidelines!

W 2 M Lec 4-10-17

Project 5

- Automatic diagnosis for doctors to replace doctors because we lack enough doctors for diseases in this world
- Unlike cataracts, diabetic retinopathy is irreversible
- One technology is to detect diabetic retinopathy using readily available information from hospitals
- Lung cancer detection to detect malignant or benign tumors
- The government needs to know how many people in the region have this disease and how should they control it
- Old people get chronic disease
- A lot of money has to be spent to control this disease
- Family members have to take care of this person
- This will greatly LOWER productivity
- Common thing in society!
- Use government data to find things like age and gender
- In this community, we can have big data and we can visualize the distribution of people having this disease

- Deployed in hospitals and your company takes care of the data collection for all this information?

Project 6
The Connoisseur project

- Looking for 10 people
- Small team from Anderson business school

Ray Xiao - Amgen

- Leading the hybrid cloud infrastructure & DevOps

During our interview, we found 42% of people have difficulties choosing where to begin searching for a dining place!

- This is a typical case when talking to a friend!
- When people say “up to you where to eat”, this is a pain in the ass!

Also we would like to give customers good experience for

- Recommendations
- Social interaction
- Bookmarks

Architecture

- Backend
- AWS
- Spring Boot
- Storage
- NoSQL Database
- AWS DynamoDB
- SQL Database
- AWS Redshift

Join Connoisseur project

- Develop a real **Mobile App**

Skillsets

- NodeJS/JavaScript, it will be perfect if you already know React Native
- Swift/Objective-C will be a huge bonus
- Java/Python are welcome too
- Graphics processing, UI and UX Design

Q&A

- Ray Xiao - lei.xiao.2019@Andersoon.ucla.edu

Project 7
Secure File Reception Project

- Ariento Inc.

- Accountant - takes your W-2 Forms and file your taxes for you
- An accountant can be one person working from their house or it can be a firm of 100+ people
- Sweetspot is 25 employees or less
- The issue is that they deal with a lot of sensitive information
- If it is your account, you don't want it to get stolen or out there on the internet.
- Most accountants don't know jack shit about security!

<u>Front End</u>

0. Web Application
- Simply file upload portal
- Compare to a whitelist to make sure it is a client's email you are sending it to
- They actually have to know the email of the client
- JavaScript based so we can make this relatively easily
- No UI work needed!

- Need a web application that centralizes management of motorcycle parts for 3 platforms:

0. Shopify platform
0. eBay platform
0. Facebook store platform

Project 8

Passionate People Project

- We have found that on career websites nowadays, they look at what you do in a job, but not really the passion you feel in the job
- A Pandora-like algorithm to let users explore careers based on preference
- Users can look at similar types of careers

SOC Example

- Code for a very broad occupation - technical occupations
- As you add each number, it encodes to a more and more specific career

Quiz on Wednesday

- Assigned reading - look at the class webpage and a bit of the quiz
- Late edition (IBM) will show a possible project
- Choose a project by Thursday

W 2 W Lec 4-12-17

Quiz Review

0. How do you organize that information?
- Organize it by the stakeholders i.e. managers, employees

- You can organize it by software architecture by the design you have in your head
- Which way is better depends how much you know ahead of time.

0. How do you change a requirements document?

- First look at the problem. A change request comes in and it reflects a problem felt by users and you want to check if it is valid.
- Understand the problem
- Analyze how much of a change to your requirement set is really needed
- Keep your requirements document well-organized, so keep them modular where making a change won't involve changing a lot of little bits of the requirements document

0. How much more expensive is it to change in the production stage than the requirements stage?

- It is going to be 10x-100x times as expensive (McConnell)
- The implication is that it depends on the project spec!
- For small projects, the factor isn't as great
- For larger projects, yeah it is fucking huge lmao

Project 9 (IBM)

- They build Watson and it is used to play Jeopardy
- This guy works on the Watson group which is awesome

IBM Project Lead

- Dedicated IBM lead to work with you (UCLA alumnus)

Project Learning Skills

- Software Engineering Skills
- Project planning
- Collaboration
- Open source community
- Sourcing
- How to work with business people
- Concepts Emphasized
- User Experience

This is a Set of Requirements

- Use design thinking and think about the user experience (how will someone use your product?)
- Snapchat is a dope AF app!
- If you had to drill down 8 times to pick the right filter, would you use Snapchat?
- How is the end user going to use our software?
- If it is too difficult to use your product, then NO ONE is going to use it!

The Magic of IBM Watson

- Available through 30 APIs on there
- What can I do with Watson?
- Watson would take the best moments from the tournament and create highlight reels from it in real-time.
- These are all built with Bluemix APIs
- Draws information from the context of the conversation as well as your own personality
- Ask Watson questions on things you don't know the answer to.

Serena Williams - appropriate training exercises and analyzes their game and their personality

Memory Lane: Cool CS 130 IBM Projects

- Barcode Wallet
- Analogous to Apple Pay
- Volunteer snack delivery service
- Similar to Uber Eats
- Choose your own adventure
- **You have to use the IBM Bluemix stack though (fuck that shit!)**

Reqts Engineering

- If you do mistakes early on, they become very costly to fix later.
- Don't spend too much time but the more common mistake is to NOT spend enough time on requirements than too much time
- They will know some aspects of the problem and you need to build this bridge!
- Stakeholders will have their own language, and you as the software developer will have to learn their language!
- Translate their ideas into ideas that will work when talking about the design
- Model of the problem is arguably the hardest thing to nail down!

Q. How do you know whether you are doing a good job or not?
A. Good requirements are testable, so you can quantifiably determine if you satisfy the requirements

There is no way a testing organization can write a test case!

- Suppose we wanted our application to be user-friendly, but we cannot use this one because it is NOT testable.
- Can we change it to something that is testable and satisfies the intent of this requirement.
- With two hours' training, the expected user error rate is < 2%
- Great virtue in having requirements that are testable
- A bit of an art to have requirements that match your intuition that are also testable

- This ought to be one of your goals (in some sense, it is better than your requirements)

- What if we have a condition “With two hours training, expected user error rate <= 0%”?
- That is impossible!

- Don’t conflict with each other
- This seems like an obvious thing! Why list it at all?
- The hard part is not obvious that it extends

Conflict-dispute resolution is an important part of your job as a software engineer!

- If you find conflicts (which will inevitably happen in a large project), you need to know who was in charge of this issue. **Otherwise, you won’t know how to solve the problem!**
- You have to change at least one of them, and V.C. Smith ought to know!
- When you write down requirements, make sure you know where they came from.

Good Reqts:

- are bounded (know the scope of rest; draws a line)
- are essential (accidents vs essence)
- Whenever you are trying to think about a topic, there are aspects of the topic that come up, but they are an accident of the particular way you are thinking about it now
- Accident (Did I want to use the blackboard or whiteboard?)
- Essence (What are requirements in my software engineering project?)
- are specified at the user level
- Bridge from the stakeholders to some other stuff
- Stakeholders have to be able to read the requirements and you need to write it in normal lingo!
- Don’t use the gobbly-goo (even the word “algorithm”, wtf is that?)!
- Do NOT use terms like O(N^2), the layman has no idea wtf that means!
- are prioritized

system vision is analogous to a system model

- Look at the problem area and also the set of potential solutions and on the solution you will actually be using
- You want the requirements to match that because when you are done building the thing, you can make it easier to see things you eventually build

- Validate requirements and check off that they are tested!

- System requirements for Firefox will fall under operations
- We will run on a Microsoft Windows 7

Functional vs Non-functional

- Performance: how quickly does the program work?
- Reliability: how often does the program work correctly?

Reqts phrases

- inception - skipping this first step is costly!
- **Identify the stakeholders**
- Find out who your clients are so you know what their concerns are
- Set of stakeholders may NOT be known to those initially starting the project
- elicitation/discovery
- Dummy programs or prototypes can be added as part of your requirements document
- negotiation
- validation (later)

Deadline for choosing which project when:

- Tomorrow at 5 PM
- TAs will email you instruction and have a Google Doc

W 2 Dis 4-14-17

Project Timeline

0. 04/21/2017 - Project Timeline and SRS.v1
0. 04/26/2017 - SRS.v2
0. 05/12/2017 - Midterm Presentation and Demos
0. 06/06/2017 - Final Report
0. 06/09/2017 - Final Presentation and Demos

Finish implementation by 05/26/2017

Project Grading

04/21/2017 Project Timeline and SRS.v1	2%
04/26/2017 SRS.v2	2%
05/12/2017 (Midterm Presentation and demo)	
Implementation Documentation	6%
Testing	4%
Demo and Presentation	6%
06/06/2017 Final Report	6%
06/09/2017 (Final Presentation and demo)	
Implementation Documentation	6%
Testing	2%
Demo and Presentation	6%

W 3 M Lec 4-17-17

Quiz Review

- You should pick a process model that is appropriate for your project.
- Doghouses can use a simple architectural model versus constructing Engineering VI
- Building Engineering VI took 4 years
- What things are done to prototypes to accelerate the schedule?
- Just hack it out.
- You don't have to worry about scaling or performance
- If you are trying to see if certain behavior is what the user wants, the performance can stink

Reqts (cont'd)

- IEEE standard: contains a whole lot of stuff that could be overkill for your application
- Probably overkill for projects in this course which is relatively small
- The glossary defines terms and you can use it to record the language of your customer
- Their is a good chance they could be talking in a different language
- System architecture
- This is what Eggert was talking about last time with the Abstract System Model - develop it in a minimal high-level way and write down the requirements
- System evolution: provides room for further expansion

Reqts Negotiation

- Serves as a way of having what you agreed to as a negotiation process
- Analogous to a treaty!
- Enforcements is known as **validation**
- Checks that the treaty makes sense!

Reqts Evolution

- Requirements can evolve
- **This is a mine field!**
- Requirements change in software because the world is changing and you cannot please everyone, unfortunately!
- Have a change policy that is formal and elaborate!
- Trace to them and from them!
- Requirements aren't something that exist in isolation. They exist for a reason and thus, they affect things
- You want to have a nice diagram that lists all the dependencies in both directions
- **Definitely helps to have automated support for this!**

0. Find a reqts problem
0. Change proposal
0. Assess costs <STOP>

0. Do it.

- Code is a thing we spend a lot of our time working on and we have pretty good tools for it!
- Do that with requirements too!

Big picture: Hooker's software engineering practice principles

0. Provide value to users
- It should serve to benefit the client!
0. K.I.S.S.
0. Architectural vision
- Goal is to have an overall vision of what you are trying to build so it remains simple as it evolves
0. Plan to get hit by a bus
- If any single member of your team dies, then your team will still be okay.
- No one should be irreplaceable in a software engineering project
- You have to think of your own mortality
- Usually your developer gets hired by a competitor
- You need to document your code so that your knowledge is shared amongst all your developers
0. Be ready to change
0. Plan for reuse
- You have a plan that you know of for it
- Internal technique for adjusting to change
0. THINK before doing
- IBM's motto back in the day

Around this central part of the universe is some other stuff such as "Detailed design"

- Software architecture - the way you hook your programs together and all that sort of thing

Good analogy

- Writing code is like writing a letter
- "Plan to throw it away. You will anyhow." - F. Brooks
- Take your program and just throw it in the trash can
- Get out another piece of paper and write it again!
- **This approach tends to NOT scale!**
- If you project is sufficient large, then you might be better off doing a bit more planning

A large project is one where you cannot name every developer on the team!

- Sort of a fluid, dividing line, so any project larger than 100 people is definitely large
- 30 is probably a better number but it very much depends on the people on the project

Software process

- A framework for defining how software gets developed and maintained and operated
- One way to think about it is to think about how the programs run inside the developers' heads
- When you write code, you are NOT an automaton but you follow some heuristics and instincts that make you tick and do the next thing.
- A software process tries to define these processes that run inside your head.
- If you think of the software process as the programs that run inside your head, then the dynamic perspective is writing down those programs!
- communications encompass users
- planning is used amongst the developers
- modeling - coming up with models for the problem and trying to understand it by decomposition into parts
- construction - the entire subject of McConnell's book
- The software process models is one of the frameworks
- deployment - actually getting the software out the door and people are using it

QA should be done at all phases

- If you are doing a software project, is there something important you need to do that isn't listed here?

Agile methods focus more on umbrella activities, but from their point of view, umbrella activities are the hard parts, which is why agile focuses on it.

- Prescriptive methods are associated with larger projects, while agile is suited to smaller projects

Prescriptive models

- Waterfall model is a great example of this
- You might start off with requirements and then you will go into architecture and component design
- Coding actually becomes almost trivial because component design outlines a lot of it for you!
- Overall cost of the project can be reduced because you don't have to worry about other factors!
- Sit in your office and just write your code without distractions
- More efficient way of coding rather than trying to get all these things done all at once.

Downsides

- You cannot start coding until you are done designing
- You cannot try to speed things up with people working in parallel!

- It is a very streamlined process, which could run into bottlenecks
- **Lengthens your schedule!**

Two revolutions to tradition.

- Object-oriented (~1985 - 2000)
- Rational unified process (RUP)
- Designed for large systems that are object-oriented
- You have big programs running on single machines, maybe multicore, but it doesn't work that well in a networked environment
- Agile (~1995 - 2005)
- Part of the dot-com revolution
- We are trying to start a company and build software that is perhaps a bit disruptive and do it in a way that gets things done really quickly

W 3 W Lec 4-19-17

- Analysis model
- Preliminary design model

(3) construction =>
design model
code
test plan, procedure, cases
support doc (user manual, installation, operations)

(4) transition =>
delivered system
feedback

(5) production =>
use + fixes

- Interact with customers, hardware, networking, and get it all to work
- Build large systems and this very nicely organized set of tasks could correspond to your set.

dot.com boom => developers rebelled against prescriptive methods

- Promote collaboration and communication in order to self-organize

stories - tale you tell your customers about what your software does

- Keep stories small on a 3" x 5" card!
- This lists a desirable thing that you want to happen
- When the software gets used, the idea is that the customer looks at the story and sets the value of each story

XP Values

communication (not documentation)
simplicity (code for today, not tomorrow)
discipline or "courage"
feedback (from customers and developers) <= informal, verbal
respect (developers, other stakeholders)

Difficulties within agile methods

- People could be busy!
- Relies on good communication and feedback from the customers, which isn't always guaranteed

Goal of modeling:

- Lists of ——
- Assumptions

W 3 Dis (Gulzar)   4-21-17
UML

- Unified Modeling Language ("Union of all Modeling Languages")
- NOT really a well-defined language for your implementation, but it is up to interpretation
- You can interpret it based on your understanding of this whole language.
- Midway between writing a requirement document

Modeling Guidelines

- You cannot argue that there is one perfect way of doing UML diagrams

Static Modeling

- Covers the classes and how they interact with each other
- Let's say you have a class for a book object and some methods like borrowing a book.
- For each borrow a book, you have a user who is borrowing a book.
- You have this dependency that a book object is somehow dependent on a student object
- Components are similar to packages but they have different modules

Behavioral Modeling

- Use case diagrams
- Sequence diagrams
- Tells you how a module or a method in your program interacts with some other methods

UML: Class Diagram

- Models the static relationships between the components of a system
- There are associations between them

Operations in Class Diagram

- Since this class diagram is static, you cannot actually find the size of the list at compile-time.
- For this, you need an asterisk sign

W 3 Dis (Weiss) 4-21-17

- User can slide or take a picture and use it in this use case diagram
- This is what you can add to your code and in this use case diagram, there are two things.

W 4 M Lec 4-24-17

Quiz Review

0. What makes data flow diagrams function-oriented rather than object-oriented

- By looking at only a little bit of input, they can generate the next output
- We can think go if as code + data and we focus on the fact that it is object oriented
- **There is nothing in the data flow approach that requires this to be stateless, but it is kind of a tradition**

0. Give a system where master slave is better than distributed component architecture?

- The master acts as the scheduler and it doesn't do much real work on its own!
- You do this approach when you have to have a single **scheduler** and you are trying to get it done very well
- Real time renderer is emphasized
- Real time systems also tend to be simpler and if it is going to be simple anyway, then a central master is NOT a bad idea

0. Why do people use block diagrams all the time instead of UML?

- With block diagrams, you can show things to non-technical people and the blocks in your blocks can mean completely different things from one sort of diagram!
- The block diagram will deplete the larger diagram into a smaller one

Modeling Architecture

- Complex systems can be
- technical - browsers + Facebook servers
- sociotechnical - people involved
- Emergent system properties are the things you don't expect

ESPs: Fall under some major categories including

- security
- Prevent bad actors from injecting fake news
- reliability
- e.g. Swiss cheese model
- No single solution to the problem

Nondeterminism: Big system with a lot of race conditions, but you also have people involved so it is a sociotechnical system!

- You cannot predict what any individual person would do and this would be part of the problem that you have to solve
- A common phrase that comes up when people are looking at this big picture is a "wicked problem"
- You don't know what the problem is until you solve it.
- These "wicked problems" should be relatively rare, but there are times when you don't understand how difficult it is until you have to actually implement a solution!
- **These are problems that software engineers get paid for!**

Model-driven architecture (MDA)

- If not executable code, it should be closely related to executable code
- You should think of it as your first cut at writing code, and when the code is actually working, you are going to see a connection that is enforced by software tools
- MDA is applying our language translator model or architecture

Use cases

- Preconditions have been true for some time before the use cases started
- Other people like to fold the two into the same notion and we want to know what actually started the use case to happen now
- Write a bunch of these use case scripts
- Each one will explain to your developers and users how your system is expected to be used.

3 issues worth exploring in use cases

0. Input validation
- Actor is given bad data
0. Performance
- What does the actor do if the system is too slow?
- Often times, it is better to take performance issues and put them in another area such as timeouts or other things
- Figure out what the performance aspects are!

Architectural design patterns and issues

- persistence (DBMS)
- access control (single sign-on)
- concurrency
- Different parts of your system will operate in parallel (or at the same time)
- How will concurrency be handled?
- Have a scheduler who keeps track of all the tasks you need to do and figure out the order in which it will be done
- scalability (distributed system via cloud supplier)
- If I need more CPU, that would be part of your architectural decision

Take some example architectures

- dataflow vs <u>repository</u> (DB records info)
- You want the repository to also be part of the data store

<u>call-and-return</u> vs <u>message-passing</u>

- Call-and-return just uses the standard method call in C++ or Java
- You don't call a subroutine in a message-passing architecture; rather, you send it to the server and you can keep computing while you wait for the server to respond

layered architecture ("stack")

- Everything is modularized, so the higher-level programmers can work independently of the lower-level programmers

Distributed architecture concerns

- failure mgmt
- QoS (Quality of Service)
- How do you specify this value?
- If you are on AWS, have you read all the fine print on the promises?
- If not, you have to do some research on it

W 4 W Lec 4-26-17
Quiz Review

0. SOAP vs REST, which to use?
   - It really doesn't make much difference! You can do it either way and you essentially get the same service. The actual design shouldn't be affected all that much.
0. Why don't you need a state diagram for all the objects in your system?
   - You don't need to! State diagrams are a tool, which you use to figure out what the states are and what states you go next. When that becomes a problem, use a state diagram.
0. How do you enforce a singleton property of a class?
   - There is only one object and we don't want to use two because they would try to write to the same object.
   - Why have an object in the class at all?
   - They are always operating on behalf of one object.

FPS shooter

- If you have a UI that is tightly coupled, what you could find is that this Observer pattern gets in the way
- If you have an Observer in which you are getting lots of info and it is changing the object, this is tough.
- **Having to deal with all these pixels will hurt your performance**

Architecture

- Distributed Architecture

- Event-driven Architecture: A very simple program
- The program uses an infinite loop and you find out what the next thing to do is and you do it
- Commonly used for IoT
- This architecture saves power because the system can sleep while waiting

Master-slave - common in real time

- Figure out what the next slave should do and pass on the message

Peer-to-peer systems

- You have nodes on a network and none of these nodes are the boss

W 4 Dis 4-28-17

Week 4: Midterm
CS 130: Software Engineering

Open book and open notes

- For every lecture, highlight and focus on important points in reading
- Refer to these notes when studying for the exam
- Go over all the readings and focus on the bullet points
- Sommerville book has bullet points and you can see general principles on how to architect it

Outline

- UML Diagrams

Sequence Diagram

- Shows the interaction at each timestamp.
- Think of it as a table with each column as classes or actors and rows are time steps
- Dashed **lifeline** runs vertically in the diagram

Example

- You have a makePayment method call and you are creating this Payment class
- When you create the class, it starts from that step at whatever that time is

Statechart Diagrams

- This diagram requires us to build a state flow of the whole module with the start condition and these actions will have a certain state
- Series of preceding states to reach a state

Statechart Diagrams: Components

- There are some **actions** we want to represent on our own for a particular state
- An **activity** is a summarized view for the whole state diagram

Sample Midterm: Q1

- If you have software updates on the fly, it would change the baseline and this would be the worst fault rate
- As you go up, the software may become more and more stable

Q. Is the bathroom curve supposed to talk about the failure rate of a single device?
A. It would describe the whole system

Q2

- Test this part and if your gcc fails, then you say the fault is this part
- Hashing implies you already know where the problem is, so it will be hard to find this mapping
- The whole idea for binary search is to make it feasible for every program

Q3a

- You have already built these modules individually and then one of those tests failed. One way is to take pre-emptive measures like pair programming or formal review
- Inspection of the code by some other programmer is a cost-effective measure to try to find the defect as early as possible
- **Find defects as early as possible**

Q. How do you know this?
A. This is in the book

Q3b

- Why don't we just do pair programming?
- Humans are NOT infallible
- Need to use special components to test all the defects!

Q4

- The idea of simulation is to replicate the exact software that we are working for.
- Find bugs in your program
- Prototype is to find the feasibility of one feature.
- Find feasibility of that product.
- Show that it is doable.

Q7

- Loose coupling can lead to faults

- **Coupling**: Tells you what are the dependencies and how do the two classes interact with each other
- If you are NOT using a Color class, then the user would have to remember the integer representation of each color.
- In the case of integer, then integer is more loose coupling, and Color is tighter coupling

Q8

- Some requirements are more important than others, and 80% of the project can be validated from that

Q9

- Naming conventions preserve sanity
- If you argue naming conventions are NOT necessary anymore, you can follow the Hungarian notation of saying what that method is doing and we can omit this type because we can always find the type.
- You cannot always find the operation of that variable, however!

W 5 M Lec 5-1-17
Quiz Review

0. The data goes away!
0. EOF = constant;

getchar = vars, functions

0. If you are starting to read a bunch of methods, and they are 1,000 lines each, is that reasonable?

- Probably NOT

- Is there a limit to the length that a function should be?

- There are advocates of short routines, and there are advocates of longer routines.
- If you actually look at the data, the reliability of programs with short routines or no better than those of long routines.

- Routines should be shorter than 200 lines but the data should be squishy.

0. Abbreviate consistently

Create names you can pronounce

Component Design

- See something in the real world that has already been built and build it like that
- When taken too far, you model the component as people - “anthropomorphize” - **too far**!
- We talk about desktops, which is an analogy to the literal top of a desk

If you are NOT doing object-oriented design, you will have similar axes so you can change your software later.

Design forces

- You want to create an object, but you want one of your subclasses to do it.
- Cache a common answer and reuse rather than reallocating and making duplicates

Coach John Wooden teaching his basketball players to put on socks
- **The fundamentals are huge**

Routines are the basics of software engineering
- If you can write code, you will be writing methods, functions, procedures, etc.
- McConnell has a couple of chapters on it that cover various topics

Cohesion & Coupling & Routines
- Normally, one thinks of cohesion as applying at the class level.
- This is cohesion and coupling
- Cohesion - how well an individual routine sticks together
- What makes for good cohesion and bad cohesion in routines?
- Functional cohesion - routine just does one thing and it doesn't even interact
- Almost like an atom of the procedure calls
- Coupling - how one routine and another routine interact with each other

W 5 Dis 5-5-17
Midterm Presentations (6%)
- Give an overview of what you have designed and implemented at this point
- Recommended to have a demo to show and the TA Muhammad is the one grading it
- Submit the slides with the implementation document

Implementation Document (6%)
- This is due May 12th at 11:55 PM
- Makes sure it is under 15 pages including all figures, screenshots, etc.
- Include the GitHub repo link

Implementation Document Expectations (Cont.)
- User Interface Screenshots

Design Patterns
- Observer
- Mediator
- Strategy

Why do we use design patterns?

- The idea is to have an abstraction of implementation details for certain cases in projects.
- These are mostly applicable to object-oriented programming i.e. Java

Design Pattern Category

- How would you make a pizza?
- Dough
- Cheese
- Other flavors
- If you want to create a Chicago style pizza, you will have these individual components to build up the pizza

Strategy Exercise

- You need a base class where all of these algorithms apply and a generalized algorithm
- What is the base context class?
- Duck class where you can apply this strategy, or a client class where you have all of these ducks
- If you can figure out these strategies, it is easy to figure out the strategies in the design pattern

Observer

- More related to these client-server architectures with objects working together and you can work with each of them
- Let master class know the state of these base classes
- Add these new subjects at any time.
- Notify the Observer class and a complete observer

Observer Exercise

- Subject is the Button
- Observer is the listener

Mediator

- Centralize communications between objects
- Select linear dependencies between objects

W 5 (Tomer) Dis 5-5-17
Midterm Presentation checklist

- Attending presentation day is mandatory
- Members of group that belong to section C
- Have ALL the team members present their work
- If you have a demo, show demo
- ~20 minutes assigned to each team

Talk about difficulties and limitations i.e. changing APIs due to incompatibilities

- Mention if you have any issues with the client

- Everyone has to talk during the presentation - no sitting around

Avoid using ALL CAPITAL letters

- Use bold face and italics instead for emphasis
- Do NOT suddenly change your slide formatting
- Grading based on slides and if they are nice or NOT nice

Design Patterns

- Why design patterns?
- What is a good design pattern
- Compiler class
- The visitor design pattern is just a way for each object to implement method called “Visit”
- Otherwise, it will be hard to call each of these objects

Creational

- No other instance of that class
- Singleton
- You only know there will be only one instance of this class and there is only one environment in this simulation

Strategy Pattern

- Good way to modularize code since you can basically just switch between these strategies

Strategy Exercise

- Suppose we have a Car class. We know the car’s make and model and we can sort the list of cars based on different criteria.
- Think-Pair-Share
- Subject - text or button itself
- Observer - the pressButton
- Update (notify) method - sending the text to the listeners

Think-Pair-Share

- What are the drawbacks of the observer design pattern?
- Tightly coupled
- These listeners my NOT be connected to each other so there is NO connection between listeners
- There are some cases when this could be bad

Mediator Exercise

- Many dependencies here and when we see a dependency between the text box and the submit
- There could be dependencies between the other widgets as well too

W 6 M Lec 5-8-17

Quiz Review

0. The cost of not making the change is part of the whole cost-benefit computation. Look at the benefit of the change and you can divide it by the cost of the change. A lot of your software engineering activities will be ruled by iron, economic law.
    - The benefit of change 1 is the difference between what happens if you make the change and what happens if you do NOT make the change!
    - Presumably, that is going to be development effort that you are going to do.
0. If managers do NOT understand the QA activities, then they will NOT give credit to the developers for doing this. You won't get any brownie points for developing more software than your neighbor if the QA activity is NOT explicit. This helps to motivate the programmers.
0. Documenting a release - different versions of Make will act differently with the same script.
    - Software tool versions
    - It isn't simply the software that matters, there is also other stuff that matters such as building on hardware!
    - You need to know which hardware version to use!
    - This will be needed in some development environments where hardware versions matter
    - You will get a different release depending on which hardware you use
    - The details of how you build the release so precisely that other people can get exactly the same release that you did
0. Just run Hadoop or Spark and it does it all for free. Becoming more technologically and socially acceptable for these tools to track what you are doing.
    - In the old days, I should be able to run the tool in the privacy of my own station.
    - Nowadays, software is "watching us" and this data is now available and treated as normal.

Sample software quality problem

- dump operation is inherently machine-independent
- Create a GNU/Linux x86-64 executable
- What approach should we remove this **dump** procedure problem
- Unportable
- Hard to maintain
- Tends to flake out on upgrades

0. Makes initialization faster (Load Lisp code faster.)
    - Q?
0. Build a portable dumper. (simple, portable binary format for Emacs state)

Defect-detection roles

- The people who do defect amplification tend to focus on the first category of errors (once you find the error, the fix is obvious) qualis - “of what kind”

- We don’t know how it is!

W 6 W Lec 5-10-17
Quiz Review

0. Why is beta testing important

A. Beta testing is verification testing to some extent and validation testing to some extent. Learning about the environment and you can learn whether or not you have the environment right. You can also do marketing here and try to see if you can sell it!

0. What % of errors are construction errors?

A. 75% initially and then it drops as the project gets bigger

0. Is our random-data generator useful?

A. All you have to do is tweak so this is useful.

0. Difference between validation testing and verification testing?

A. Make sure you are building the product that people you actually want.

QM QA

- CM

Testing

- If software isn’t changing, then it is dead software.
- The idea here is that successful software keeps changing
- Configuration management is the process of keeping that change under control so you have a good handle on it
- It isn’t always true!
- Sometimes, you will be put into this type of software where this is a major issue
- Previous developers didn’t do a good job writing down which pieces went into the system
- Identification can turn into detective work.
- Lets say you are put in charge of Boelter Hall, which will undergo a lot of configuration changes in the next few years. One of the things you have to figure out is what these 60 years old light switches do.
- There are some solenoids build with 1930s technology, and that isn’t written down anywhere.

Cross-compiler

- Imagine this is the source code to the compiler i.e. Spark
- The generated compiler will generate code for the ARM
- These can be three different host machine types involved

Build automation is key.

- shell scripts - build a system on this command and this is often the most popular way to build a system
- A problem with this approach is that a shell script is just a command and when you want to run it in different ways i.e. the build failed the last time, it can be TOO SLOW!!!!

- make - think bout as sort of avoiding this unnecessary work
- The difference between scripting and a build approach like make is that make knows about dependencies
- In sh, they have sequential programs and maybe recursion, but it does actions running one at a time
- Make is able to be parallelized since it knows about the dependencies of your build system!
- It knows that if you build component A and component B and they are independent, they can get better performance by running them in parallel
- Autoconf
- Come up with a brief description of the shell script rather than all the details
- Typical for these shell scripts to be in megabytes

W 7 M Lec 5-15-17
Danger signs of project mgmt

- **Managers & developers avoid best practices**
- Sponsorship lost (or was never there)
- Project lacks people and right skills
- Users don't want it
- Business needs change (or are unknown)
- Unrealistic deadlines

More danger signs

- Chosen technology changes
- **Changes are managed poorly.**
- In effect, they merged together security changes with other functionality changes
- Product scope isn't defined.
- Developers don't customer needs.
- If you see only one danger sign, you might be okay

People.
S: 25:1

- Is it common to put an NBA player on the court and have them score 50 points and have another NBA player score 2 points in the same amount of time?
- NO!

M: 5:1

- Is it common to see an NBA game with a score of 100-20?
- NO!

W 7 W Lec 5-17-17
Pricing to win

- This can be ethical

- For a public institution, you have to account for the taxpayers and Regents.
- Do we trust developers to do the right thing?
- It is really a situational problem!

Management of Software Projects
"I must be able to read and understand any code written or the project." - B. Gates

- Do we really have to worry about this special case?
- If you pick a software developer at random, but is the size of the project they are working on

W 7 Dis 5-19-17
Software Testing Levels

- When you are system testing, if you find a bug, you have to do integration tests again
- Make more unit tests for that class and each one of them can improve things.
- When you find bugs in your acceptance tests, you find more bugs there
- When developing your code, write more unit tests and create more integration tests to make sure your newly developed units are developed.

Integration Testing

- Continuous Integration (CI): Emphasized by XP methodology
- Works with part of your system or your whole system
- After you have this script and change a new component, you want to run your integration tests again and each time you make a change, it is a good practice to change things each time.
- This is known as **continuous integration (CI)**

W 8 M Lec 5-22-17
Quiz Review

0. Why are software safety arguments easier to make than software correctness arguments?
- Draw a FSM
0. Why are tool based static analysis better than formal verification
- Doesn't matter what the specification is.
- This is a valid C program

W 8 W Lec 5-24-17

0. Is pure-block indentation better than the other indentation where { gets its own line?
- Doesn't matter
0. Why doesn't experience matter when trying to optimize code?
- It is NOT as helpful as we would like because hardware and software evolve

- People learn how to program using one code style/technology and then they move on to a different technology

W Dis 5-26-17
Static and runtime analysis
- Different types you are assigning
- This is a static technique that is run in the background
- See if it conforms to these particular patterns and type checking is a particular component here

Linters
- Take tree and there is no dynamic inputs that are assigned to those trees

Static analysis
- Throws some errors or warnings to you, but they may be false positives or false negatives
- The programs you are submitting to a compiler may be good or bad
- You can use data flow analysis and draw the graph again to find type checking

Linters
- Lint was a tool originally developed alongside the C programming language
- Once you have this graphical representation of a program, it is easy to analyze
- Using Linters on top of these is extremely easy to do

Aside: Type checking printf()
- Variable argument function that might have different types and it is hard to check the compiled code
- Compilers copy the whole code of the sodlib and then figure out these are the types and perform static type checking from there

Summary
- You might be able to see a lot of bugs that you don't actually want to see, but this gives you all the static warnings of the program
- It is hard to check for new bugs since you might have to check new classes

Runtime analysis:
- These are the inputs that we have to detect

Midterm Statistics
Mean: 72.6
Standard Deviation: 10

Symbolic Execution

- You might miss out on some statements if you run those test cases
- Whatever test cases you have will exercise all the statements of the program
- Unique solution in that it helps figure out what the test cases are and all the statements are exercised
- Add them together and figure out all the possible values of the output
- This will help you figure out the input values and input datasets in the program

Start with Control Flow Graph

- First step is to draw a control flow graph for this particular program
- 2 unique paths here (two conditions: if and else)
- Starting point and it can branch out

W 9 W Lec 5-31-17
Safe refactoring

- Functional behavior of the code should remain the same ideally

Look at each byte and figure out the count by looking at the number of times it loops through

- Right shift the bytes by 1
- This loop executes at most 8 times, but the performance sticks

W 9 Dis 6-2-17

- Final Review Questions
- Final Presentations and Demo (6%)
- All team members should be present
- 20 minutes including live demo
- You must, at the minimum:
- Introduce the purpose of your project
- Give an overview of the design, implementation, and testing efforts
- Present a live demo
- Submit the presentation slides in PDF format

Final Report (12% + Testing 6%)

- Due date is June 9th 11:55 PM
- Only submit 1 report (in PDF format) per team
- 11pt font, 10-20 pages including all figures, screenshots, etc.
- Include your team's repository URL

Presentation Expectations

- Must demonstrate a complete, working demo
- Must describe the design and implementation of client's most important feature
- Must explain testing strategies

- Highlight challenges and software engineering principles
- Give a high level overview of different project modules

Final Exam Review
Q1. Give an example of a software error that does not lead to a fault, and of a software fault that does not lead to a failure.
A. Example of a software error that does not lead to a fault is a memory leak because you cannot detect this at compile-time. Another case is if you want to do a - b, but then you do a + b, but b is always 0. Forgetting a semicolon is a software fault that does not lead to a failure.

Q2. In a well-run project, how do you know whether you've devoted enough development resources to software testing?
A. You would quantify it by code coverage and spend some time determining the severity of bugs. You can figure out what the benefits of test cases are and it you can quantify it by the severity and number of bugs found and fixed!

Q3. Give an example of how one could usefully apply an XP-oriented practice in the context of a project that normally uses the Unified Process Model.
A. Push out releases once a week and maintain good communication between team members as well as the client and the solution provider. You can always rate your stories with respect to the risks that they pose and starting with the most high-risk story, this would eventually lead to a stack solution and apply things on top of that.
- You can pick the highest-value story and start working on that first.

Q4. Why is it relatively rare to do both beta testing and acceptance testing on the same software product?
A. Beta testing has no assumption about how the environments are. Any user can do whatever they want with the software and there is no particular script that they are going to follow. Even if you run it for several different times, acceptance testing is contractual based testing and only tests one particular aspect of the environment and only tests one module.
- Acceptance testing - analogous to unit tests or one Selenium test. Note it cannot test the entire thing!
- Beta testing - you cannot expect any structured flow of actions from the user. It can happen in ANY order!
- It is rare to do both because if you are doing beta testing, you are assuming that the code can run on several different environments.
- The assumptions for these two test cases are important!

Q5. In your team's software project, which was the most important time-consuming: requirements analysis, systems and component design, or quality assurance? Explain briefly
A. You want your requirements very detailed. Ariento was very vague about what they wanted, so this was a huge problem for us. The waterfall model showed the failure of this and we can say if our software was security-sensitive or reliable.

- Pick one of these things and start explaining your rationale behind this.
- Be short and concise.

Q6. In hindsight, which software metric would you have found most useful for the project that you did for class? why?
A. You can use lines of code to measure the quantitative metrics of how much you have done for the project.

Q7. Does refactoring typically make code shorter or longer? Faster or slower? More reliable or less? Why?
A. Shorter code means less redundancy, easier to read. It should generally be more reliable if the person refactoring truly understands the code to make it more optimal

Q8. Inayat et al. (2015) systematically reviewed scholarly literature on agile requirements engineering (RE) and found the following strengths and weaknesses as compared to planned RE:

a. Briefly justify any choices for which it is not immediately obvious which is a strength and which is a weakness of agile RE.
b. Which items in Inayat et al.'s list are most relevant to your project this quarter?

A. **Weakness**. You cannot estimate a strict budget or schedule for the project.
B. **Strength**. You are prevented to doing rapid changes.
C. **Weakness.**
D. **Strength.** Meeting with the client frequently and narrows down the communication gap.
E. **Strength**. You can make adjustments a lot more quickly.
F. **Strength**. Since you meet the client often, there is less chance of overs coping.
G. **Strength**. You can acquire missing documentation quickly since communication between teams tends to be better.
H. **Weakness**. Nonfunctional requirements are often ignored (?)
I. **Strength**. Communication is optimal so things aren't left out of the loop
J. **Strength**. The clients will tell you if you are meeting the expectations.
K. **Weakness**.

Q9.
9a. Draw a UML class diagram for this specification.
A. Look at notebook!

W 9 Dis (Tomer) 6-2-17
Final Exam Preview
0. Attacker can write some spooky code that will overwrite the function pointer and execute the attacker's code

a) Which CS areas or fields will provide information to approach this design? What specific help will you need from the hardware design of the CPU, from the operating system, from the C/C++ compiler or other system component?
A. You don't have to even use the Operating System. You can some compiler that preprocesses it and creates a new version of your code with this hat.

b) Give two significant, realistic, and diverse constraints for a good solution to this problem.

d) Which software engineering principles were mot important in your architecture design?
A. Pair programming, modularity, and some other things that were talked about

0. Suppose you have an array of key-value pairs where no two keys are equal,, and wish to sort the array in-place by key.
    - At least one programming question in the final
    - Like an interview style question

0. Beta testing: a way to find bugs by having users try out the product!
Acceptance testing: checks requirements and see if the results pass these requirements!

W 10 M Lec 6-5-17
Quiz Review
0. The comment should be put into version control, NOT in the comments!
0. Give an example of a system-oriented error message vs a user-oriented error message
System-oriented diagnostic: invalid index in line 27
User-oriented diagnostic: student ID is not in DB
0. Why do you need process documentation?
A. You are afraid you are going to get sued. It will meet certain regulatory requirements and most of the time, these are process requirements.
0. How might a think aloud session be useful in evaluating a user interface?
A. As a user uses your program, they will say aloud what they are thinking. The user turns on the verbosity flag and you records what they say to analyze what misimpression's your user has on your software.

Tools
- git blame - V.C.
- diff - git diff or some wrapper around it

User-centered Design
- Software should remember what they are doing
- Have consistent UI!

Apple's secret design sauces

- Apple is very good at coming up with software that is highly usable

W 10 W Lec 6-7-17
Quiz Review

0. Name a CMMI example and its goal.

Goal
Find root causes of defects systematically
What is your system for finding and squashing entire classes of bugs

Practice
Look for root causes of defects and catalog them
You are going to classify them!