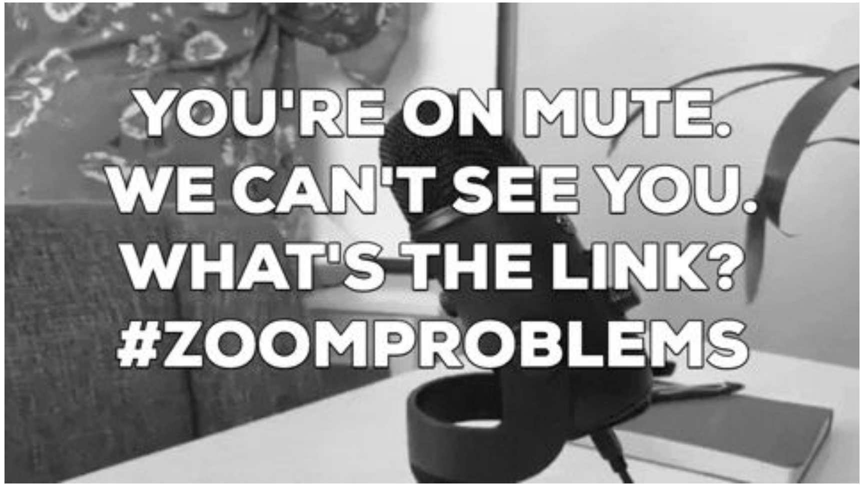
COMS W4156 Advanced Software Engineering (ASE)

September 6, 2022
First day of class!

Important Notice



All my classes will be held on Zoom, probably “forever” not just this semester.

If you do not want to take this course on Zoom, then please drop the course.

Prof. Yang’s W4152 Engineering Software as a Service is a reasonable substitute for most students (except not for MS Computer Security track, since this course addresses some security issues and 4152 does not).

There are a lot of students on the waitlist who would like to take this course and may not hate Zoom as much as you do!

Lecture Notes linked on Courseworks Calendar

“Live” links into google docs, so possibly changing up to and even during class

Will generate pdfs and post in Courseworks->Files->lecture notes later

Questions and Comments during Class

Zoom, either voice (“raise hand”) or chat (type whenever you like)

Questions and Comments outside of Class

Ed Discussion (supports anonymous and private posts)

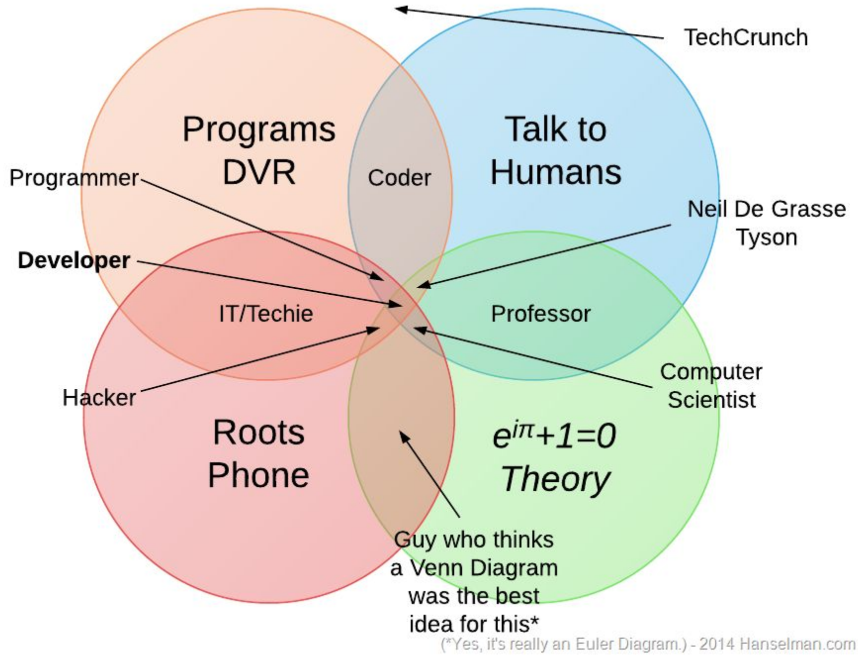
Agenda

1. Brief explanation of what <u>I</u> mean by “Software Engineering”
2. Team Project
3. Overview of this course

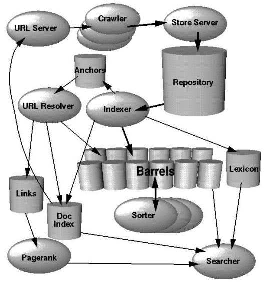
What is Software Engineering?

Software Engineering != Programming

- *Time and Change* affect the sustainability of software, how the *codebase* adapts over the length of its life
- *Scale and Growth* affect the viability of software practices, how the *organization* adapts as it grows and evolves



Time and Change

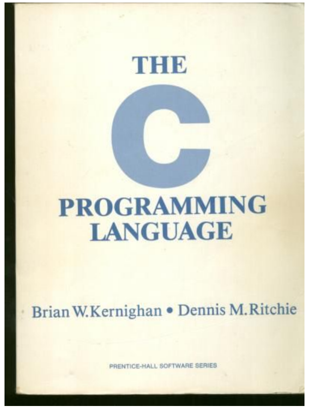


Google search was a graduate student project called "Backrub" in 1996, written in Java and Python [soon rewritten mostly in C/C++, with crawlers in Python]

In 1996, Java version 1.0, Python version 1.4

Today, Java version 18.x (many people use earlier LTS versions), Python version 3.10.x (someone somewhere is probably still using 2.7 even though past "end of life")

The original Backrub *could not run today*!


THE
C
PROGRAMMING LANGUAGE
Brian W.Kernighan • Dennis M.Ritchie
PRENTICE-HALL SOFTWARE SERIES

It is unlikely that any other 1996 or even 2006 software written in Java or Python (or any other language except possibly C) could run "as-is" today. Note this has nothing to do with new features or any other internal changes, the problem is *external* changes

Software Sustainability

Different from "sustainable computing" (aka "green computing"), although a mandate for green computing would count as an external change

How *painful* is it to modify the software to fit external change?

Could be change in 'ecosystem': language, compiler, libraries, operating system, hardware

Could be changes in regulations (e.g., GDPR and other privacy protections)

Could be transient changes due to world circumstances (e.g., sudden dramatic increase in unemployment insurance claims)

Success depends on organizational culture, process and tools

Software Sustainability 2

The codebase has to be modified to fit changed ecosystem, new regulations, the world as is it today and tomorrow rather than when version 1 was released

How do we know whether the software still does what it’s supposed to do, how do we keep it working the way it’s supposed to, how do we sustain its ‘core functionality’?

We **test** the code to detect the bugs (any deviation from intended core functionality is a “bug”) and we fix the bugs

And when fixing the code (often) introduces new bugs, we **test** the code to detect the bugs and we fix the new bugs. And we keep modifying the code and we keep **testing** the code and we keep fixing more and more bugs

Exceptions

A few ancient (non-C) programs survive

But their code never changes and they are (mostly) exempt from external change

Scale and Growth

Backrub was written by Larry Page, Sergey Brin and a couple other Stanford graduate students

Their codebase was probably around 10k LOC

At least 25k software engineers now work for Google

Current Google codebase contains billions LOC - but its still (mostly) in a single monolithic codebase (all the developers can access all the code)

Elaborate processes, tools and technology enable scaling of both codebase and organization

Human Scalability Reality Check

For course programming assignments (or tiny startups) where two people work together, it might be feasible to email new versions of the code back and forth

This usually fails by the time the team grows to four people - no one knows what is the current version or who has it

For course projects (or small startups) where four or five people work together using a shared [github]() (or similar) version control repository, it might be reasonable to turn on [notifications]() to automatically send email to everyone in the team when anyone pushes modified code

But imagine every software engineer at Google getting email whenever any other software engineer at Google pushes some code (16k commits per day!)

	COMMENT	DATE
	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
	ENABLED CONFIG FILE PARSING	9 HOURS AGO
	MISC BUGFIXES	5 HOURS AGO
	CODE ADDITIONS/EDITS	4 HOURS AGO
	MORE CODE	4 HOURS AGO
	HERE HAVE CODE.	4 HOURS AGO
	AAAAAAAA	3 HOURS AGO
	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
	MY HANDS ARE TYPING WORDS	2 HOURS AGO
	HAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Human Scalability Reality Check 2

Imagine every change in the codebase potentially impacting the core functionality of every other part of the other code … simultaneously being changed by other software engineers

Imagine all the bugs!

How to Learn Sustainable and Scalable SE

Ideally, the entire class would work on finding and fixing bugs in 25+ year old software consisting of a billion files, with new features constantly being added by tens of thousands of other developers, with four billion active users … but that's not going to happen

What will happen is a team project (4-5 students per team), where teams propose, develop and test their own Java or C++ software server (or service) through two iterations

Servers are not apps, servers have APIs (Application Programming Interfaces)

Servers do not have GUIs (*clients* of servers may have GUIs)

Agenda

1. Brief explanation of what I mean by “Software Engineering”
2. Team Project
3. Overview of this course

Team Project

Ideally 4 students per team, 5 students allowed (no teams of 1, 2, 3, 6, 7, …)

Students arrange their own teams, use Ed Discussion (and anything else you want) to find team members

Every team will have a Mentor who is either an IA or a Leader for this course

Team projects will develop and test *servers* written in Java or C++

What Are Servers?

Server != App , Server ~= Service

A server provides an application programming interface (API), not an end-user user interface

There may be a command line interface (CLI) for administrators to start/stop and perform other actions on the server, but it would not normally be available to end-users

Clients of the server may be apps or may be other servers, clients run remotely from the server and interact with server over network

It's ok to develop a collection of micro-services that interact with each other and clients, instead of a single server

Example APIs: https://github.com/public-apis/public-apis and https://github.com/n0shake/Public-APIs

Team Project 2

TEAM FORMATION

PRELIMINARY PROJECT PROPOSAL

(Team meets with Mentor after preliminary proposal to receive feedback towards revised proposal)

REVISED PROJECT PROPOSAL

FIRST ITERATION

FIRST ITERATION DEMO
(for Mentor)

SECOND ITERATION

SECOND ITERATION DEMO
(for Mentor)

DEMO DAY
(for me and any students in class who want to watch other teams' demos)

“Differential Grading”

Default is for all members of the same team to get the same grade for the project

But every year there are a few students who do not do their fair share

Such students will receive a lower grade than their teammates, based on input from their other team members and from their team mentor

Every year there are also a few students who go above and beyond

Those students may receive a higher grade than their teammates, again based on input from their other team members and from their team mentor

Differential grading is applied overall at end of semester, not per assignment

Agenda

1. Brief explanation of what I mean by “Software Engineering”
2. Team Project
3. Overview of this course

This Course Is About Finding Bugs (so they can be fixed)

Testing

Static analysis

Dynamic analysis

Testing

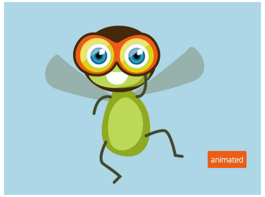
Testing

Testing

Testing

Testing

Testing



If you hate testing (most students do!), and do not want to learn about testing (most students don't!), please drop the course

Warning: Your first five years as a software engineer (until you're a senior engineer and/or a tech lead) will almost certainly include testing other people's code and fixing bugs in other people's code

Visit Courseworks Home Page for Course Logistics

CourseWorks
@Columbia

Next Class

Build / package management

Ask Me Anything