CS 5150: Software Engineering

Prof. Saikat Dutta

Spring 2025

www.cs.cornell.edu/courses/cs5150

The Pitch

- You already know how to write **programs**
- But can you create a **software system**?
  - Example: Canvas
- If I offered to pay you to automate some work, could you answer these basic questions?
  - How long will it take?
  - How will you know when you're done?
  - How often will it crash?
  - Is it even legal?

About the course

Objective: Prepare you to contribute to reliable software systems

- Semester-long **team project**
- 4 credits = 10+ hours of work per week (outside of lecture)
  - Regular team meetings instead of discussion section
- 5000-level: assume you can **teach yourself** new technical skills
- Focus on expanding capabilities of **large, existing** codebases

About your instructor: Saikat Dutta

- Completed PhD in 2023 in Computer Science
- Research Interests: Software Engineering x Machine Learning
- Joined Cornell in Fall 2024 as an Assistant Professor
- Industry exp: Microsoft, MSR, and AWS
- Languages: C++, Java, Python

- **Office hours**: Wed 11 AM – 12 PM Gates 438

Course Staff

- Elaine Yao
  yy2282@cornell.edu
  Pengyue Jiang
  pj257@cornell.edu
  Jacqueline Cai
  jxc6@cornell.edu
  Bryant Park
  blp73@cornell.edu

- **TA Office Hours:** Wednesdays 4 - 5 PM Gates G01 (G11 for this week)

Illustrated with successes and failures

- Linux kernel
  - Distributed development, review
- SQLite
  - Testing – 640x more test code
- Blender & Krita
  - User Collaboration, Open Develop.
- Dragon 2 spacecraft (SpaceX)
  - Agile Development

- Ariane 5 (1996)
- Boeing 737 MAX
- FAA Advanced Automation System
- Therac-25
- …

Software bugs…

CROWDSTRIKE — BUGS — CYBERSECURITY — NEWS

CrowdStrike bug maxes out 100% of CPU, requires Windows reboots

"Note: This is 100% of a single core. In an 8-core system for example, an additional 12.5% of unexpected total CPU load would be experienced..."

THE STACK

June 28, 2024 . 4:18 PM — 2 min read

The Washington Post

Democracy Dies in Darkness

Technology

Tesla sued by family of Apple engineer killed in Autopilot crash

"Tesla's Autopilot feature was defective and caused Huang's death," attorneys for Walter Huang said.

The New York Times

Airline Blames Bad Software in San Francisco Crash

LOSSES FROM SOFTWARE FAILURES (USD)

TRICENTIS

1,715,430,778,504

ONETRILLIONSEVENHUNDREDFIFTEENBILLIONFOURHUNDREDTHIRTYMILLIONSEVENHUNDREDSEVENTY-EIGHTTHOUSANDFIVEHUNDREDFOUR

Activity: So, what is software engineering?

1. Turn to your neighbors. Introduce each other.

2. Discuss: what makes “software engineering” different from “programming”?

3. Select a reporter

Textbook definitions

- “An engineering discipline that is concerned with all aspects of software production”
- “Multi-person development of a multi-version program”
- “Programming integrated over time”
- “When ‘clever’ is an accusation rather than a compliment”

Software Engineering: Themes of this course

- Software Engineering involves the principles and practices for …
  - Software Design (Modeling)
  - Software Implementation (Programming)
  - Software Analysis (Performance)
  - Software Management and Team work
  - Quality Assurance (Testing, Verification, Repair, …)
  - Software Maintenance
  - Professionalism
  - Delivery
  - …

Syllabus

Course Infrastructure

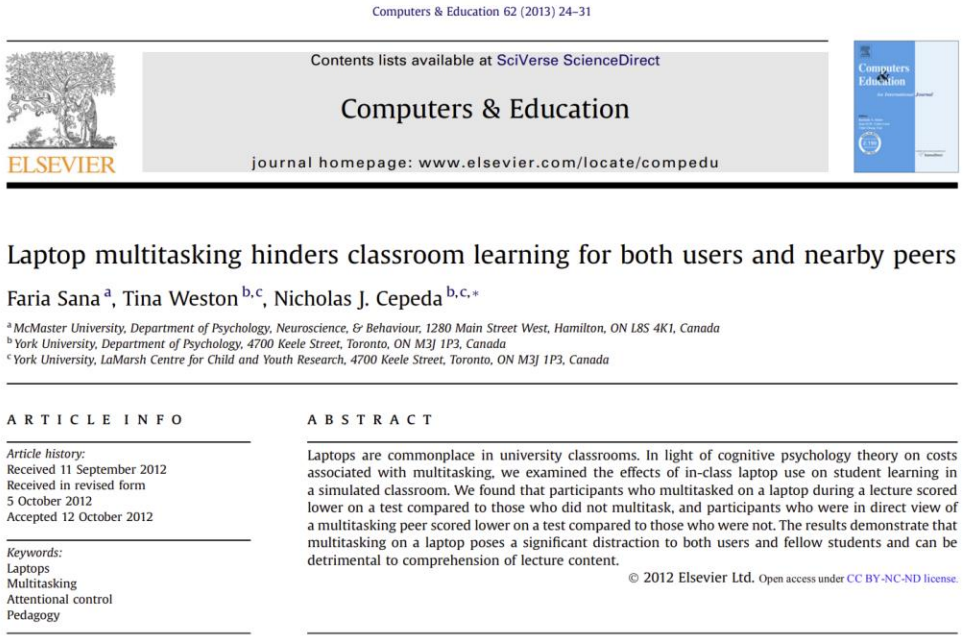
- Website: www.cs.cornell.edu/courses/cs5150/
  - Slides, assignments, schedule, syllabus
- Canvas (restricted to enrolled students)
  - Ed Discussion, Gradescope, Appointment reservations
  - Quizzes, surveys, etc.
- Poll Everywhere: pollev.com/cs5150sp25 (use Cornell e-mail)
- Cornell GitHub: github.coecis.cornell.edu

Example poll

PollEv.com/cs5150sp25

Personal devices

- Laptops/tablets may be useful in class for:
  - Responding to polls
  - Following along with demos
  - Taking notes (but paper is better)
- BUT devices are distracting for you *and your neighbors*
- Please do not multitask during lecture. If you can't resist, sit in the back



Computers & Education 62 (2013) 24–31

Contents lists available at SciVerse ScienceDirect

Computers & Education

journal homepage: www.elsevier.com/locate/compedu

Laptop multitasking hinders classroom learning for both users and nearby peers

Faria Sana [a], Tina Weston [b,c], Nicholas J. Cepeda [b,c,*]

[a] McMaster University, Department of Psychology, Neuroscience, & Behaviour, 1280 Main Street West, Hamilton, ON L8S 4K1, Canada
[b] York University, Department of Psychology, 4700 Keele Street, Toronto, ON M3J 1P3, Canada
[c] York University, LaMarsh Centre for Child and Youth Research, 4700 Keele Street, Toronto, ON M3J 1P3, Canada

ARTICLE INFO

Article history:
Received 11 September 2012
Received in revised form
5 October 2012
Accepted 12 October 2012

Keywords:
Laptops
Multitasking
Attentional control
Pedagogy

ABSTRACT

Laptops are commonplace in university classrooms. In light of cognitive psychology theory on costs associated with multitasking, we examined the effects of in-class laptop use on student learning in a simulated classroom. We found that participants who multitasked on a laptop during a lecture scored lower on a test compared to those who did not multitask, and participants who were in direct view of a multitasking peer scored lower on a test compared to those who were not. The results demonstrate that multitasking on a laptop poses a significant distraction to both users and fellow students and can be detrimental to comprehension of lecture content.

"...participants who multitasked on a laptop during a lecture scored lower ***on a test compared to those who did not multitask****, and participants who were in direct view of a multitasking peer scored lower on a test compared to those who were not. The results demonstrate that multitasking on a laptop poses a significant distraction to both users and fellow students and can be detrimental to comprehension of lecture content."*

Assessment

- Lecture polls/In-class quizzes
  - Perfect attendance ***not*** necessary
- Canvas Assignments, Quizzes
  - 4-5 assignments
- Project deliverables
  - Reports, documentation, code, code reviews
  - Scored against rubric
  - Client presentations
  - Peer reviews
- In-class exams (2)
- No Final Exam

References

- No required textbook
  - All readings freely available in electronic form
- Several recommended books, including a free one:
  - *Software Engineering at Google* https://abseil.io/resources/swe-book
- *Better Embedded System Software*. Philip Koopman
- *The Mythical Man-Month*. Frederick P. Brooks, Jr
- *Software Engineering, Tenth Edition*. Ian Sommerville

Course is overfull

If you decide this course isn't for you, that's okay. But please drop promptly to make room for students on the waitlist.

I cannot predict if you will get in!

For those on the waitlist (or waiting to get on the waitlist):

The instructor does not control enrollment; send questions to cs-course-enroll@cornell.edu

Disclaimers about CS 5150

- This is a new version of 5150 – so schedule might change at any time! (but hopefully not by much) – please be patient!

- Please use **Ed** to ask questions – fastest way to get response from staff (reserve emails for personal/important things)

- TAs as project clients – please be respectful to them

The Project

Purpose of the project

- Experience interacting with a client other than yourself
  - Requirements elicitation, acceptance testing
- Experience designing for users other than yourself
- Experience all phases of software lifecycle
  - Specification, development, validation, evolution
- Practice coordinating with a team

Teams

- Team size: 4-5 students
- Full-semester commitment
- Two options for forming teams (by Feb 3):
  1. Form a (partial or complete) team around a project
  2. Get matched with a team based on preferences
- When considering teammates, check for:
  - Compatible schedules, work styles
  - Backend language comfort (Java, Python)
  - 2+ members with frontend experience (HTML/CSS/JS)
- More about teams later in Lecture 3

Project options

Option 1

- Course-sponsored project (enhancing code review tools)
  - Print a review log
  - Link comments to multiple files
  - …
- Code Translation
- Instructor or TA will serve “client” role

Option 2

- You find a client with a project proposal
  - Client may not be yourselves or other students
  - Must involve an existing, active codebase (not written by you)
- Complete survey: “petition for external project”
- Recruit on Ed Discussions

Demo: Gerrit

https://gerrit-review.googlesource.com

Project Timeline

- Semester consists of five 3-week “sessions”
- Report due at the end of each session
- Should meet with client once per session
- Two formal presentations: **Midpoint** and **Final**
- Final delivery on the last day of classes

- **Time** and **resources** are fixed, so must adjust **scope** to fit in time available

Deliverables

- Session reports
  - Plan
  - Accomplishments, setbacks, discovered work
  - Peer feedback
- Work log
- Design documentation
- Test plans and reports
- Coverage analysis
- Requirements
- Code and code reviews
- User documentation

Activity: SwE slang

In your groups, select a new recorder and make up definitions for the following terms (don't Google them; just guess and reach consensus)

- Bike-shedding
- Yak shaving
- Dogfooding
- Greenfield
- MVP
- DevOps

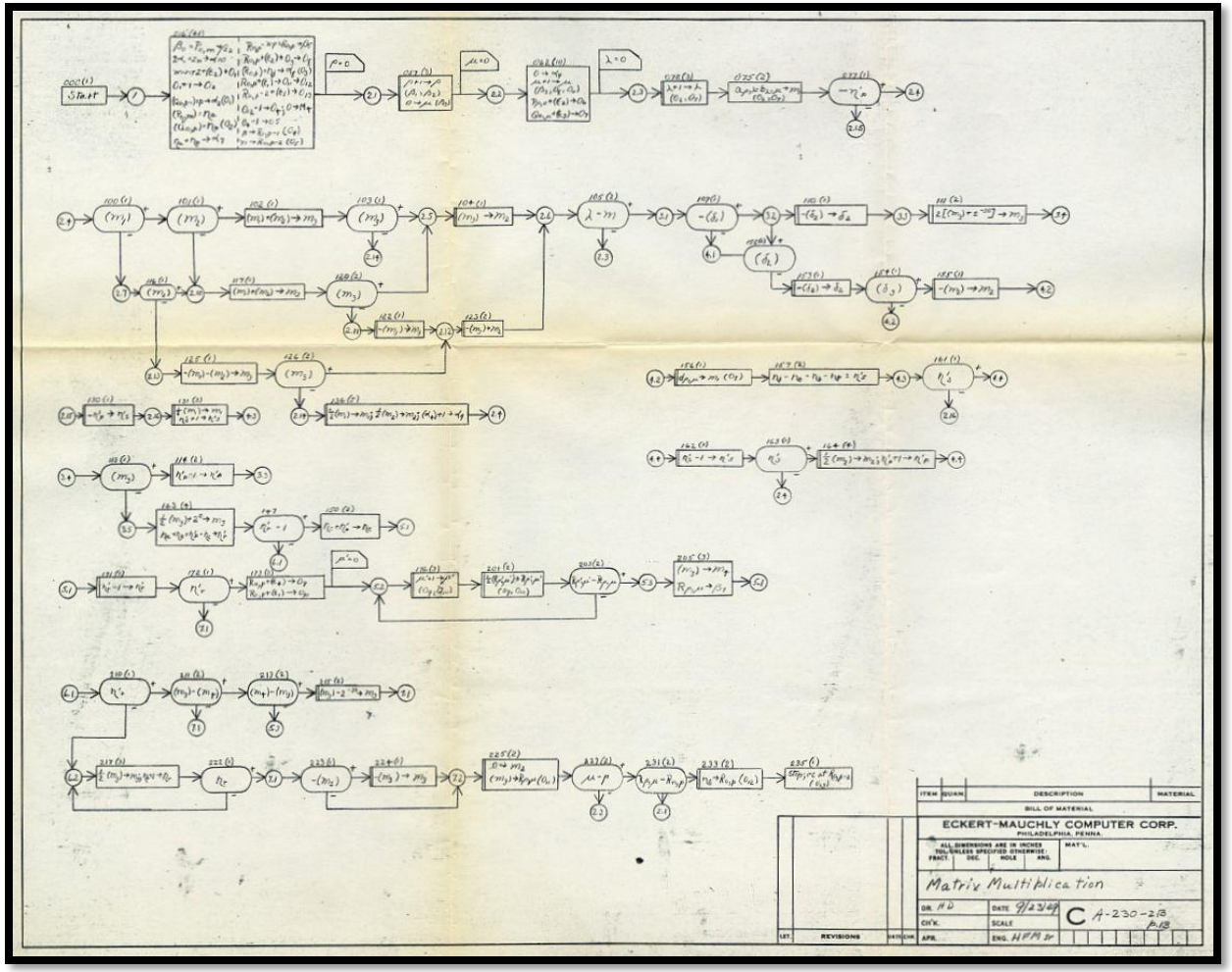
Activity: SwE slang

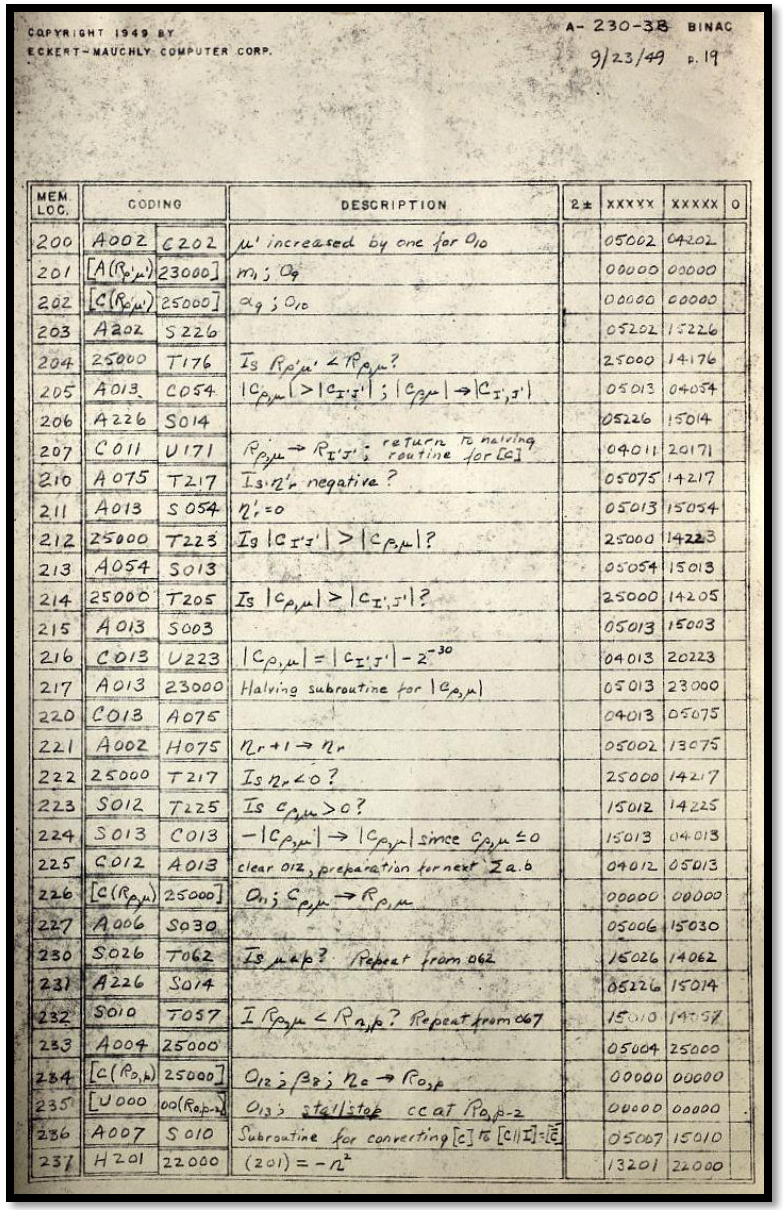
In your groups, select a new recorder and make up definitions for the following terms (don't Google them; just guess and reach consensus)

- **Bike-shedding**: spending too much time on trivial issues
- **Yak shaving**: seemingly endless series of small tasks that need to be completed before the next step in a project
- **Dogfooding**: Using your own products/services
- **Greenfield**: Create something from scratch
- **MVP**: Most Valuable Player
- **DevOps**: Developer Operations

Some history

Early software development





COPYRIGHT 1949 BY
ECKERT-MAUCHLY COMPUTER CORP.

A-230-3B BINAC
9/23/49 p. 19

MEM. LOC.	CODING		DESCRIPTION	2±	XXXXX	XXXXX	O
200	A002	C202	μ' increased by one for O_{10}		05002	04202	
201	[A($R_{\rho,\mu'}$)	23000]	m_3; O_9		00000	00000	
202	[C($R_{\rho,\mu'}$)	25000]	α_9; O_{10}		00000	00000	
203	A202	S226			05202	15226	
204	25000	T176	Is $R_{\rho,\mu'} < R_{\rho,\mu}$?		25000	14176	
205	A013	C054	$\|c_{\rho,\mu}\| > \|c_{I',J'}\|$; $\|c_{\rho,\mu}\| \rightarrow \|c_{I',J'}\|$		05013	04054	
206	A226	S014			05226	15014	
207	C011	U171	$R_{\rho,\mu} \rightarrow R_{I',J'}$; return to halving routine for [c]		04011	20171	
210	A075	T217	Is η'_r negative?		05075	14217	
211	A013	S054	$\eta'_r = 0$		05013	15054	
212	25000	T223	Is $\|c_{I',J'}\| > \|c_{\rho,\mu}\|$?		25000	14223	
213	A054	S013			05054	15013	
214	25000	T205	Is $\|c_{\rho,\mu}\| > \|c_{I',J'}\|$?		25000	14205	
215	A013	S003			05013	15003	
216	C013	U223	$\|c_{\rho,\mu}\| = \|c_{I',J'}\| - 2^{-30}$		04013	20223	
217	A013	23000	Halving subroutine for $\|c_{\rho,\mu}\|$		05013	23000	
220	C013	A075			04013	05075	
221	A002	H075	$\eta_{r+1} \rightarrow \eta_r$		05002	13075	
222	25000	T217	Is $\eta_r < 0$?		25000	14217	
223	S012	T225	Is $c_{\rho,\mu} > 0$?		15012	14225	
224	S013	C013	$-\|c_{\rho,\mu}\| \rightarrow \|c_{\rho,\mu}\|$ since $c_{\rho,\mu} \leq 0$		15013	04013	
225	C012	A013	clear 012, preparation for next $\Sigma a.b$		04012	05013	
226	[C($R_{\rho,\mu}$)	25000]	O_{11}; $c_{\rho,\mu} \rightarrow R_{\rho,\mu}$		00000	00000	
227	A006	S030			05006	15030	
230	S026	T062	Is $\mu < p$? Repeat from 062		15026	14062	
231	A226	S014			05226	15014	
232	S010	T057	Is $R_{\rho,\mu} < R_{\nu,p}$? Repeat from 057		15010	14057	
233	A004	25000			05004	25000	
234	[C($R_{0,\mu}$)	25000]	O_{12}; β_8; $\eta_c \rightarrow R_{0,p}$		00000	00000	
235	[U000	00($R_{\rho,\mu}$)]	O_{13}; stall/stop cc at $R_{0,p-2}$		00000	00000	
236	A007	S010	Subroutine for converting [c] to [C||I]=[$\bar{C}$]		05007	15010	
237	H201	22000	(201) = $-n^2$		13201	22000	

Grace Murray Hopper Collection, National Museum of American History

Structured Programming

- Arbitrary flowcharts hard to understand, verify
  - Yield programs with many GOTO statements
  - “Spaghetti code”
- Can restrict yourself to common patterns:
  - Control structures
    - Sequence
    - Selection (if-then-else)
    - Iteration (for, while)
  - Subroutines (named sequence)
- 1968: Edsger Dijkstra publishes *Go To Statement Considered Harmful*

Software Crisis

- 1960s: computers getting faster, can tackle bigger problems
- Software couldn't keep up
  - Over budget
  - Behind schedule
  - Buggy
  - Unmaintainable

The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.

- Edsger Dijkstra, 1972

NATO software engineering conferences

- Held in 1968, 1969 to address the “software crisis”
  - David Gries in attendance
- Admitted inadequacy of contemporary methods
- Agreed upon best practices for developing software, grounded in principles of engineering
- Established the term “software engineering”

Incremental improvements

- Structured programming
- Information hiding
- Object-oriented programming
- Language enforcement (Ada)
- Formal methods (CS 4160)
- CASE tools
- UML
- Components & services
- Agile methods
- The Internet
- Open source
- Code forges

Are we still in crisis?

- No new technology, tool, or process was a "silver bullet"
- Software has expanded in capability and scale, but …
  - Projects still behind schedule, over budget
  - Inefficiency has been masked by hardware advances
  - Much software is low-quality (e.g. security)
- And the stakes are higher than ever!
  - Software governs much of modern society

Why not treat SW like HW?

Engineers have built large, reliable structures for millennia. Why not take same approach to SW?

- "Easy" to change
  - But costly in time, risk
- More coupled
- Leaner supply chain
- No unit production cost
- Scalable access
- Different reliability analysis

Context of modern software engineering

https://www.informationisbeautiful.net/visualizations/million-lines-of-code/

- Most software development is not "greenfield"
  - Far more time spent in ongoing development/maintenance than in initial development
  - Most software needs to interact closely with existing systems
  - Want to leverage reuse

Job titles

- “Engineer” is used liberally by employers
  - Most are not certified professionals
- Related roles:
  - Site reliability engineer
  - DevOps engineer
  - Test software engineer
  - Machine learning engineer

*As an engineer, you will **design**, **develop** and **test** software. You will be **responsible for the complete life cycle** of the software you create, from development to testing to operation.*

Next Up

1. Read about FAA AAS
2. Count lines of code (submit on Canvas)
    1. In your biggest individual project
    2. In an open-source project of your choice
3. Start forming teams around projects

Reading, assignment due before next lecture
(1:25 PM Thurs, Jan 23)

Submission deadlines will be extended for students not yet enrolled in Canvas