DEPARTMENT OF COMPUTER SCIENCES
& CYBERSECURITY

COLLEGE OF ENGINEERING

FLORIDA TECH

CSE 1101 COMPUTING DISCIPLINES & CAREERS HANDOUTS

WILLIAM SHOAFF

SEPTEMBER 16, 2015

Homework

Thirty years ago my older brother, who was ten years old at the time, was trying to get a report written on birds that he'd had three months to write, which was due the next day. We were out at our family cabin in Bolinas, and he was at the kitchen table close to tears, surrounded by binder paper and pencils and unopened books about birds, immobilized by the hugeness of the task ahead. Then my father sat down beside him put his arm around my brother's shoulder, and said, "Bird by bird, buddy. Just take it bird by bird."

Anne Lamott, Bird by Bird: Some Instructions on Writing and Life

"Wonder what it's like to have a peaceful life," Ron sighed, as evening after evening they struggled through all the extra homework they were getting.

J. K. Rowling, Harry Potter and the Sorcerer's Stone

The same people who never did their homework in high school are still doing that to this very day out in the real world.

Jules Shear

Homework Submissions

Submit your homework using the course management system. Homework may be submitted by the due date before midnight. Submissions after the due date will not be accepted. Submissions are timestamped. Essays are analyzed by TurnItIn.

Well, there is always a waiver: Present a written verifiable reason that the instructor can accept as an excuse.

Grade Calculation

Be very careful, don't miss an assignment. Almost all assignments are 10% (a letter grade) of your total score. Your total score can be calculated by the formula

$$S = \sum_{k=0}^{n-1} w_k s_k$$

where the non-negative weights w_k sum to 1 and the values s_k are scores on your efforts, $k = 0, 1, \ldots, n-1$. There are ten homework assignments and one final examination; therefore $n = 11$. These are summarized below. Each assignment will have a maximum score of 100. Therefore your score will be some number from 0 to 100.

I expect to use the range of scores below to determine final letter grades.

A = 90 – 100 B = 80 – 89 C = 70 – 79 D = 60 – 69 F = 0 – 59

However, I will compute some statistics to help me understand the achievement of the class. I will use:

- The mean (average) score μ over all students in the class

$$\mu = \frac{1}{m} \sum_{j=0}^{m-1} S_j$$

  where there are m students and student j scored S_j points.

- The standard deviation σ of all scores

$$\sigma = \sqrt{\frac{\sum_{j=0}^{m-1} (S_j - \mu)^2}{m}}$$

- Your z score

$$z = \frac{S - \mu}{\sigma}$$

  which measures how many standard deviations your score S is above of below the mean μ.

- Your $C+$ score

$$C+ = 10z + 79$$

  which assumes an average student will be at the C+/B- cutoff.

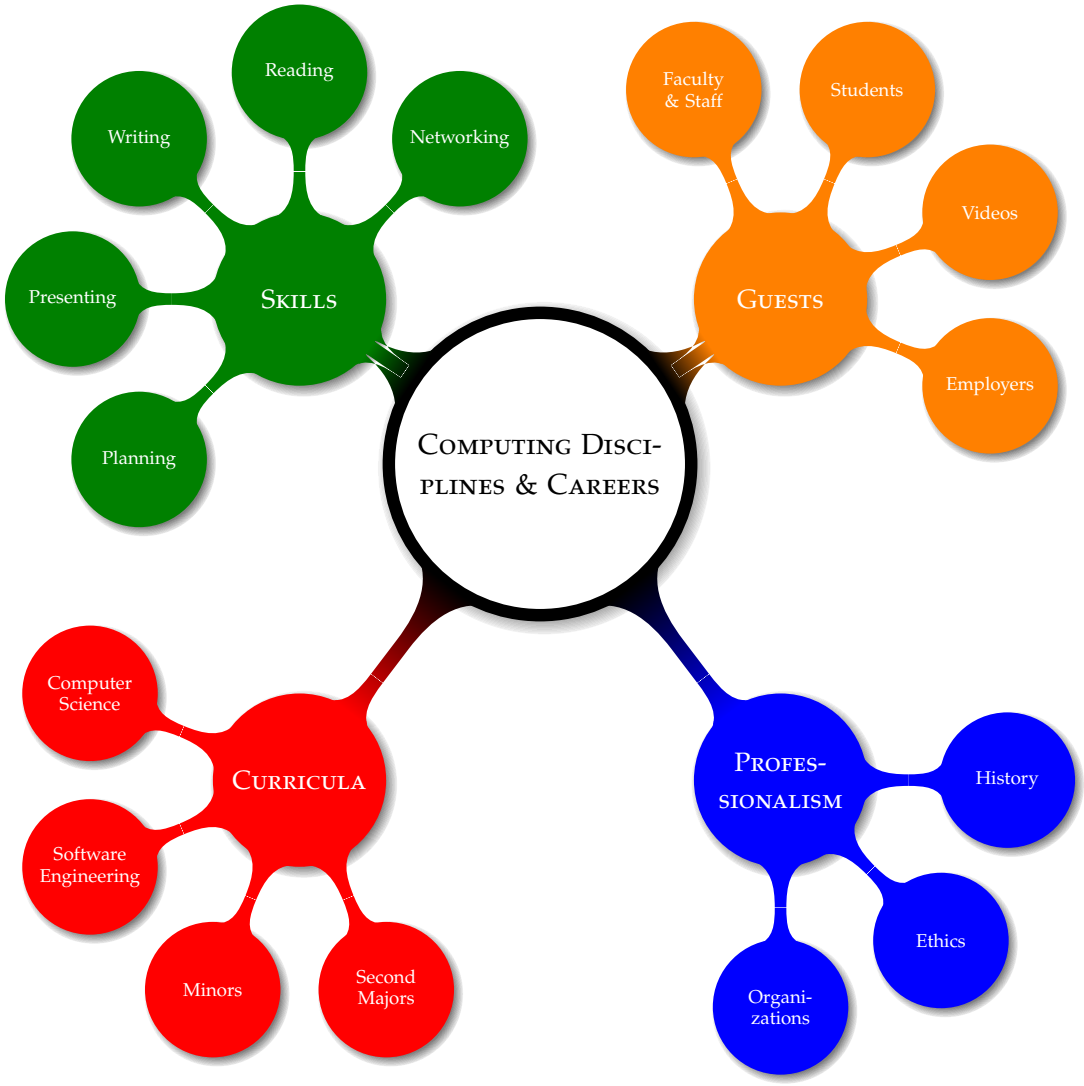
- The median m that partitions the scores into two equal-sized groups: Those below m and those above m.

Homework Summary

Here's what, why, when, and the weight of each assignment in this course. (Text colored navy blue are links that jump to other places in this document or to external documents or web pages.) I reserve the right to change the dates when assignments are due. I will only move dues dates forward and this will depend on the pace of the class.

What	Why	When	Weight
Subscribe to to keep aware of departmental events cs-forum & Panther Career Link	To learn about internships and career opportunities	August 30.	10%
Attend computing activities	To become involved with others who are interested in computing	September 13, October 11, and November 16	30%
Prepare your résumé	To prepare for obtaining an internship and career	September 27	10%
Read and write about computing news of public interest	Because knowledge current events is important	October 4	10%
Plan your college courses	To prepare for what you will learn	October 25	10%
Learn about the history of computing	Because you stand on the shoulders of others	November 18	10%
Read and write about computing	Because knowledge and its expression is important	November 25	10%
Final Examination	To demonstrate your continued understanding of the course topics	December 10	10%

CDC Mindmap


Computing Disciplines & Careers
Skills
Reading
Writing
Networking
Presenting
Planning
Guests
Faculty & Staff
Students
Videos
Employers
Curricula
Computer Science
Software Engineering
Minors
Second Majors
Professionalism
History
Ethics
Organizations

Course Calendar

The course calendar is a projection of how the class is expected to unfold. It is not written in stone. Nothing is certain. It may change. Pay attention.

Week 1: Wednesday, August 20 — Introductions

(a) Introductions: Students, Faculty, Staff, Organizations

(b) Syllabus & Assignments

(c) Video: Sir Ken Robinson says schools kill creativity

(d) Homework: Sign up to CS-Forum and Panther Career Link by August 30. Deposit a note on the course management system to confirm your completion of this task.

Week 2: Wednesday, August 27 — Professionalism

(a) Ethics: Issues in computing

(b) Reading: Reflections on Trusting Trust, by Ken Thompson

(c) Video: What most schools don't teach

(d) Homework: Participate in a first computing activity by September 13. Deposit a note on the course management system to confirm your completion of this task.

Week 3: Wednesday, September 3 — Hacking versus Cracking

(a) Videos: Snowden, NSA, Hackes as heros

(b) Reading: How to Become a Hacker, by Eric Raymond

Week 4: Wednesday, September 10 — Résumés

(a) Elevator speech

(b) Create a (video) résumé

(c) Reading: How to Think About Security, by Whittaker and Ford

(d) Video: The Art of Creative Coding

August

				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

September

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

October

		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

(e) Homework: Create your résumé by September 27. Deposit your résumé on the course management system to confirm your completion of this task.

Week 5: Wednesday, September 17 — Readings in Computing

(a) Homework: Write a paper about a computing issue that is in the news by October 4. Deposit your paper on the course management system to complete this task.

Week 6: Wednesday, September 24 — Intern panel

(a) What fellow students did

(b) Reading: No Silver Bullet, by Fred Brooks

Week 7: Wednesday, October 1 — Employer Panel

(a) What employers expect and offer

(b) Homework: Participate in a second computing activity by October 11. Deposit a note on the course management system to confirm your completion of this task.

Week 8: Wednesday, October 8 — Curricula, Part 1

(a) Computer Science

(b) Software Engineering

Week 9: Wednesday, October 15 — Curricula, Part 2

(a) Minors

(b) Majors

(c) Homework: Prepare you academic program plan by October 25. Deposit your completed academic program plan course management system to confirm your completion of this task.

Week 10: Wednesday, October 22 — Computing History

(a) People, algorithms, & technologies

(b) Video: Alan Turing

(c) Homework: Describe people, technologies, and algorithms from the history of computing by November 18. Deposit your computing history essay on the course management system to confirm your completion of this task.

Week 11: Wednesday, October 29 — Computing Skill

(a) Reading: Principles of Traditional Animation Applied to 3D Computer Animation, by John Lasseter

October

		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

November

					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

December

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

(b) Homework: Participate in a third computing activity, report by November 16. Deposit a note on the course management system to confirm your completion of this task.

Week 12: Wednesday, November 5 — Coding

(a) Reading: The Making of an Expert, by Ericsson, Prietula, and Cokely

Week 13: Wednesday, November 12 — Computing theory

(a) Reading: Gödel's Undecidability Theorem, by Stephan Andrilli.

(b) Reading: An interview with Shafi Goldwasser & Silvio Micali

(c) Video: A Turing Machine

(d) Homework: Write a second essay on your readings by November 25. Deposit your second essay on the course management system to confirm your completion of this task.

Week 14: Wednesday, November 19 — TBD

(a) Reading: The Making of an Expert, by Ericsson, Prietula, and Cokely

Week 15: Wednesday, November 26 — Thanksgiving holiday

(a) TBD

Week 16: Wednesday, December 3 — The Last Lecture

(a) The Last Lecture

Week 17: Wednesday, December 10 — Final Examination

Wednesday Final examination: Room Olin Engineering 118 from 1–3 p.m. on December 10.

1. Introduction, August 20

Take risks! Fail frequently!

Mike North

If you're not prepared to be wrong, you'll never come up with anything original.

Ken Robinson, The Element: How Finding Your Passion Changes Everything

The people involved

- The students
- Dr. Shoaff, the instructor
- Dr. Ford, the department head
- Dr. Carvalho, the director of research
- ACM representatives

Homework

[back to page 7]

1. Subscribe to cs-forum at https://lists.fit.edu/sympa/info/cs-forum. & subscribe to Panther Career Link.
2. Post a note to the course management system confirming you have completed this task.
3. Do this by August 30.

Course Syllabus

Read the syllabus. It contains information you must know:

- Various ways to contact me.

- Links to course material: The class web page, handouts, homework, class calendar
- The topics to be covered.
- A summary of what you will learn.
- How you will be graded.
- Class policies.

General Things to Know

- There are many useful apps for students on the CS Portal

  http://cs.fit.edu/Portal

- You can download Microsoft & VMware Software for free from the Computer Sciences department web site

  http://cs.fit.edu/IT/software.php

  Well, almost everything is free, Microsoft Office which is discounted at $80.

- Seek help with your assignments at the CS Help Desk

  http://cs.fit.edu/Academics/helpDesk.php

- Review the program plan/road to graduation at

  http://cs.fit.edu/Academics/#tab-1

- Get to know the faculty: Visit them or at least visit their web pages.

  http://cs.fit.edu/People/

- You can gain 24/7 key card access to Olin Engineering Labs. You'll want access to the building and room 132 (the open lab) at the least. Other rooms of interest would be 127/128, 130, 228/229.

  https://itservices.fit.edu/keyrequest

Videos

Tips of Computer Science Freshmen

Learn some basic tips for success as computer science & software engineering students.

Sir Ken Robinson

Ken Robinson gave a very interesting TED talk in 2006. Watch the video. It is titled: Schools Kill Creativity (19:29)..

To learn more about this interesting intellectual, see his web page: http://sirkenrobinson.com/.

2. *Professionalism, August 27*

Don't fail in silence!

Richard Ford's advice to new students, The Florida Tech Crimson, Fall 2011, Issue 2

Homework

[back to page 7]

1. Participate in (at least three) extracurricular computing activities this semester. Continue to participate in extracurricular activities throughout your college career.
2. For each activity, write a brief, one or two paragraph report on your experience.
3. Submit your reports using the course management system.
4. Attend activities and submit reports by September 13, October 11, and November 16.

Professional Networks

In college you *should* become acquainted with and involved in professional networks.

- College professors: They can help you manage college life and furnish you with good recommendations for jobs, internships, scholarships, and so on.
- Student organizations: Their members can become lifelong friends and they represent larger groups interested in computing.
- Professional organizations: Join professional societies, for example, the ACM and IEEE/CS.
- Businesses: Companies and organizations that can use your acquired knowledge and skills, and reward you for that.

- On-line networks: For example, the Florida Tech Computer Science Facebook Group, the Florida Tech ACM Facebook Group, and LinkedIn.

Computing Activities

There are several computing activities that occur at Florida Tech. The times and places of these events may change. Listen to cs-forum to keep up-to-date.

- Wednesday's "Tea Party" @ 3:00 on the third floor of the Harris Center
- Local ACM Meetings (check changes by listening to cs-forum)
- Friday's Computer Science Seminar @ noon in the Olin Engineering Auditorium
- Friday's Programming Practicum @ 4:00 in the Olin Engineering 229
- The Southeast Regional ACM Programming Contest, November 2, 2013
- The Harris Distinguished Lecturer Series in Assured Information (Listen to cs-forum)
- IEEE Meetings, days, time and place to be determined by you.
- Presentations by campus recruiters from various companies
- The Harris Institute for Assured Information and Space Coast Chapter of the Information Systems Security Association (ISSA) Lecture Series, the third Thursday of every month in the Harris Center, room 327 from 6:00 p.m. to 9:00 p.m.
- The Space Coast Dot Net User Group meets on the third Wednesday of every month at the Amovious office, 3900 Dow Road, Suite C at 6:30 p.m.
- The Coders Hackers & Founders sponsor meetings related to computing.
- The Space Coast Linux & OSS Meetup is another group whose meetings you may want to attend.

Other unscheduled events will happen. Keep alert. Listen to event announcements on cs-forum.

The Association for Computing Machinery

- The local chapter of the ACM is free and open to all students.
- Access the ACM Digital Library through Florida Tech's Library.
- XRDS Crossroads The ACM Magazine for Students.
- Student membership of the ACM $19.00/year.. There are many member benefits.
- The ACM Code of Ethics.
- ACM Conferences, for example SIGGRAPH Watch animation trailer for SIGGRAPH 2013 (5:09).

The IEEE Computer Society

- The local chapter of the IEEE is free and open to all students.
- Access the IEEE/CS Digital Library through Florida Tech's Library.
- Student membership of the IEEE and IEEE/CS is $20.00/year.. There are many member benefits.
- The Software Engineering Code of Ethics and Professional Practice.

Other Professional Organizations

- Women in Engineering (IEEE/WIE)

IEEE Women in Engineering (WIE) is the largest international professional organization dedicated to promoting women engineers and scientists and inspiring girls around the world to follow their academic interests to a career in engineering.

- The Society of Women Engineers (SWE)

  The Society of Women Engineers stimulates people to achieve full potential in careers as engineers and leaders, expands the image of the engineering profession as a positive force in the quality of life, and demonstrates the values of diversity.

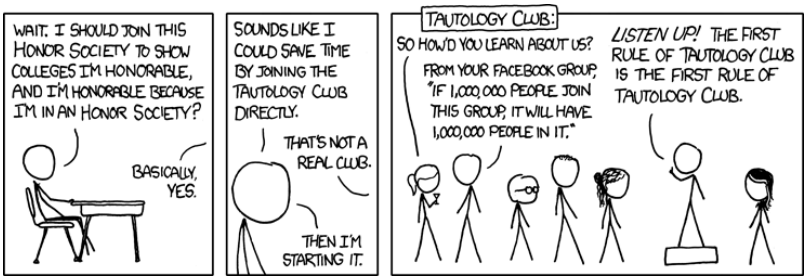
  Consider participation in the Florida Tech SWE Chapter.

- The National Society of Black Engineers (NSBE)

  Their mission is to increase the number of culturally responsible Black engineers who excel academically, succeed professionally and positively impact the community.

  Consider participation in the Florida Tech NSBE Chapter.

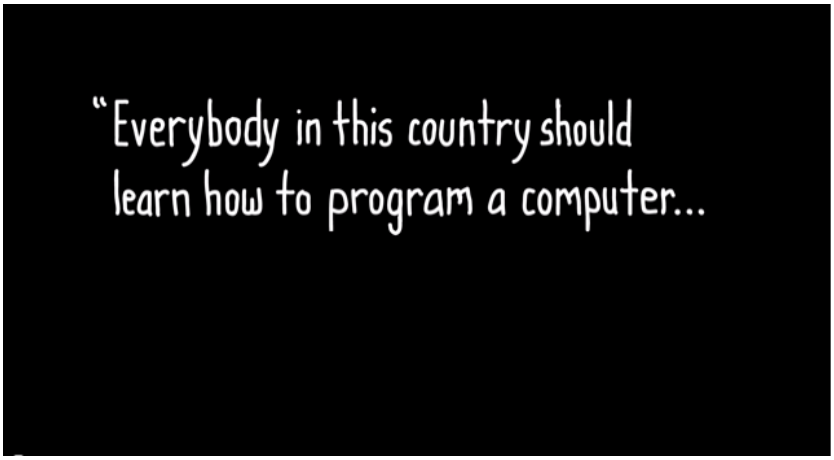
- Upsilon Pi Epsilon (UPE) is the international honor society for the computing and information disciplines. Once you achieve junior status with a good academic record you will be invited to be inducted into the UPE.



Honor Societies

Video:What Schools Don't Teach

Most K–12 schools in the United States do no teach computing. Watch the video (9:34)..



Reading

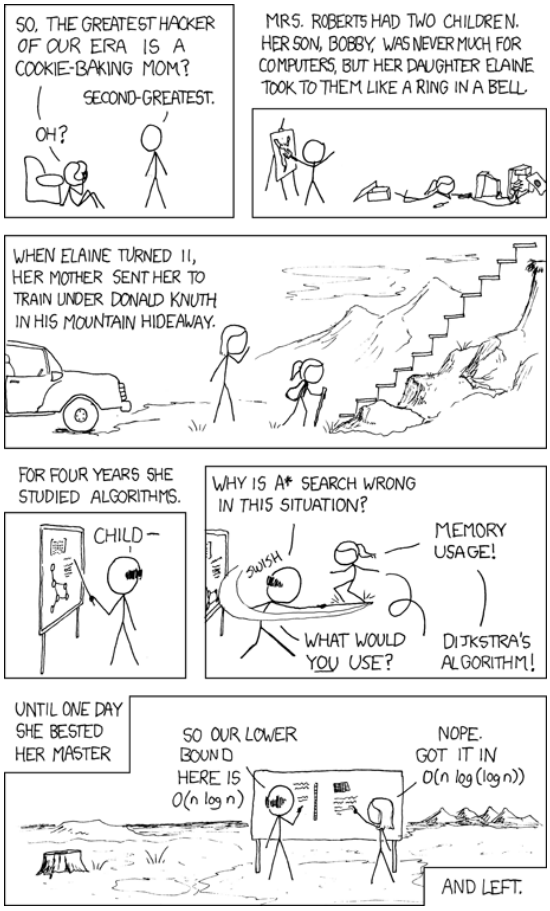
Read Ken Thompson's Turing Award lecture: Reflections on Trusting Trust. Be prepared to discuss:

- The technical issue he addresses in constructing untrustworthy code.
- The moral of his lecture.

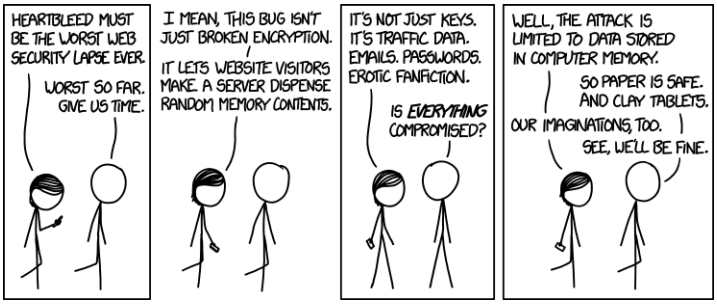
3. *Be or Become a Hacker, September 3*

The beginnings of the hacker culture as we know it today can be conveniently dated to 1961, the year MIT acquired the first PDP-1.

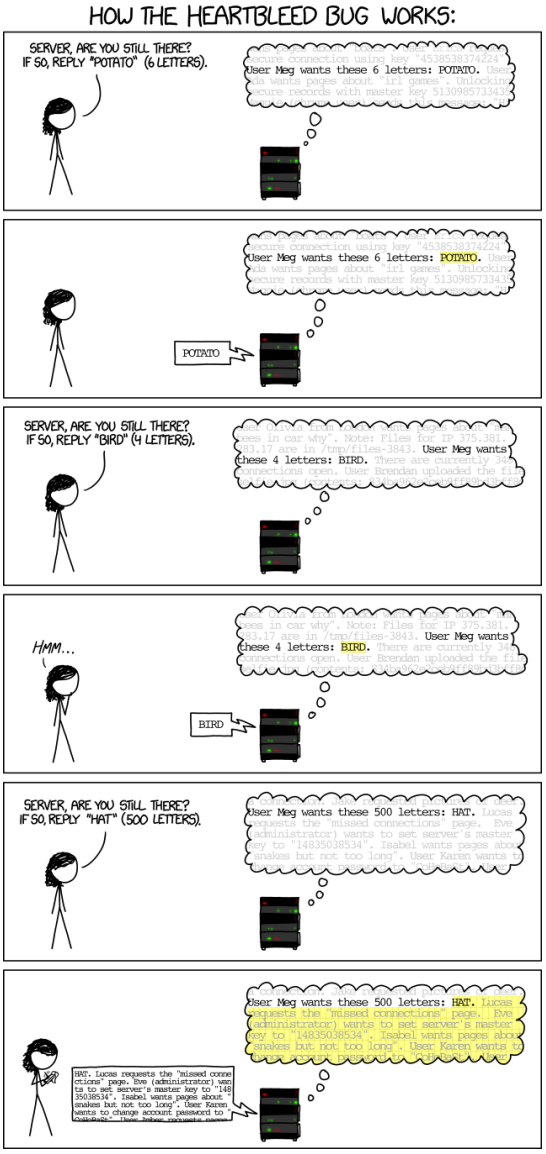
Eric S. Raymond



1337: Part 2



1353: Heartbleed



1354: Heartbleed Explanation

Homework

[back to page 7]

1. Read about a computing issue that is in the news.

   The news reporting must be since the first of the year and widely available to the public: Published by several major news organizations.

2. Write a paper summurizing the story. Describe the issue. Discuss the facts and opinions about the issue. Why is the issue important to society?

3. Submit your paper using the course management system.

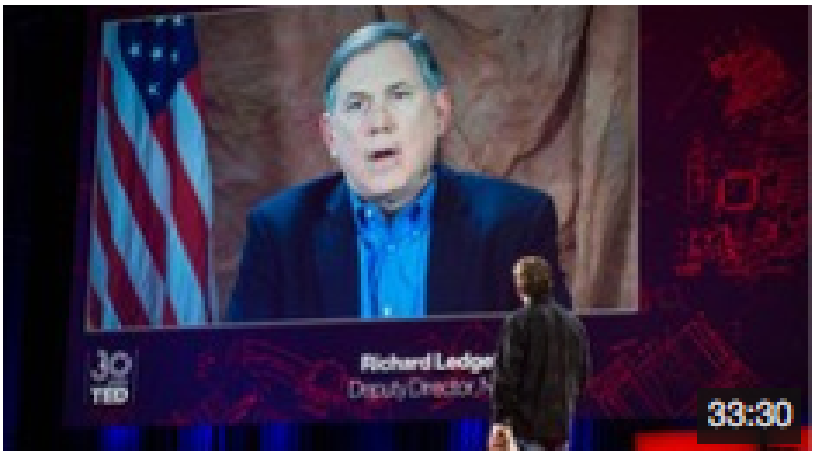
4. Complete this task by October 4.

Video:Can Hacker's be Heros

Watch the video: NSA, FISA Whistleblower Edward Snowden Explains His Actions (12:30).

TED Talk: Edward Snowden's TED Talk: How to take back the Internet (35:40)

TED Talk: The NSA responds to Edward Snowden's TED Talk (33:30)



Watch the video: Can Hacker's be Heroes? (7:26).

.

- What are your views on
  - Edward Snowden?

  - The Anonymous hacktivist group?
  - WikiLeaks?
  - Corporate espionage?
  - Cyber warfare?
  - Government surveillance (data collection) on their citizens?
  - Government surveillance on foreigners?

  Did you see the movie We Steal Secrets: The Story of WikiLeaks?

Reading

Read Eric Raymond's article, How to become a hacker Be prepared to discuss:

- How does Mr. Raymond answer the question: What is a hacker?
- The article describes three basic hacking skills.
  - What are these three skills?
  - Have you or are you mastering them?
  - Mr. Raymond's basic skills were written in late 1996.
    * Are there new skills that you would add to his list?
    * Should some of his skills be deprecated?

If you like Mr. Raymond's ideas, you might like to read his short book: The Cathedral and the Bazaar.

4. Create A Résumé, September 10

Also, I don't exactly expect to go hungry if I decide to leave the University. "Resume: Linux" looks pretty good in many places.

Linus Torvalds, First Monday, Volume 3, Number 3 – 2 March 1998

Almost everyone needs a résumé. The odds are you do, or will, need one too. If you have one already, congratulations.

Homework

[back to page 7]

1. Visit Florida Tech's Career Management Services (CMS). You can do this physically in the Florida Tech Commons or virtually on their web site. Join the *Experience Recruiting Network* at the Career Management Services site.
2. Download and use apps to keep you informed about technology job opportunities: Monster, Glassdoor, Job Search, Engineer Jobs, Beyond, JobAware, and others.
3. Download and read the Panther Career Link"User Guide for Main Campus Students."
4. Participate in the Career Fair.
5. Create a résumé. The Career Management Services web pages has resources for this.
6. Submit your résumé using the course management system.
7. Complete this assignment by September 27.

http://www.fit.edu/career/

See Career Management Services' web site for information on the Career Fair.

Why Should You Have A Résumé?

One reason students come to college is to prepare for a career. To gain a job that will lead to a career you will need a résumé that documents your knowledge, skills, practical experience, and other factors that will entice a company to interview and hire you.

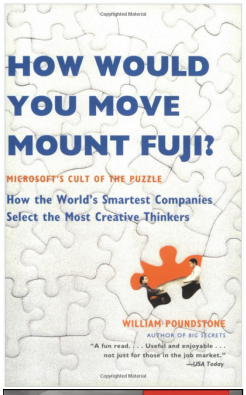
Although students in a *first-term-in-college* class may have little to include in their résumé, this will soon change. Now is the time to start developing your résumé, not just-in-time when you find you need one.

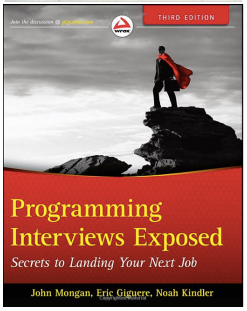
Other Things You Should Do

Here is a brainstormed unordered list of steps you can take to help you succeed in landing that perfect job.

- Google yourself. Clean up your online image. Does your Facebook page reflect well on you?
- Post a video résumé on YouTube. Here is a creative example (3:37).

- Join LinkedIn and post your video résumé on it.
- Write an elevator speech: a sound bite to introduce yourself. Create an opening sentence that will grab the listener's attention. Make it simple. Get it down to about 30 seconds.
- Know yourself. Write down the services or features that you provide.





Preparing for an Internship or Job Interview

There are several references you can use to learn how to ace a job interview. Two are *How Would You Move Mount Fuji* (Mongan and Suojanen, 2000) and *Programming Interviews Exposed* (Poundstone, 2003), but there are others.

There are several computing specific actions you can take to prepare for an interview.

- Learn the lessons in your classes. Although it often takes hindsight to recognize the value of a class, (almost all) required and well-chosen elective classes will benefit you.
- Go the extra mile.
  - Participate in Dr. Stansifer's "Programming Practicum" (Fridays at 4:00).
  - Participate in the programming contest, held every weekend.
  - Write games.
  - Write apps.

There are several question for which you may want to have quick answers.

- What are your key strengths?
- What is it you're trying to let others know about you? Think in terms of the benefits that your clients or employer could derive from these services.
- What problems do you solve?
- What STAR (situation, task, actions, results) can you describe?
- What basic computing problems can you solve?

Additional Career Building Resources

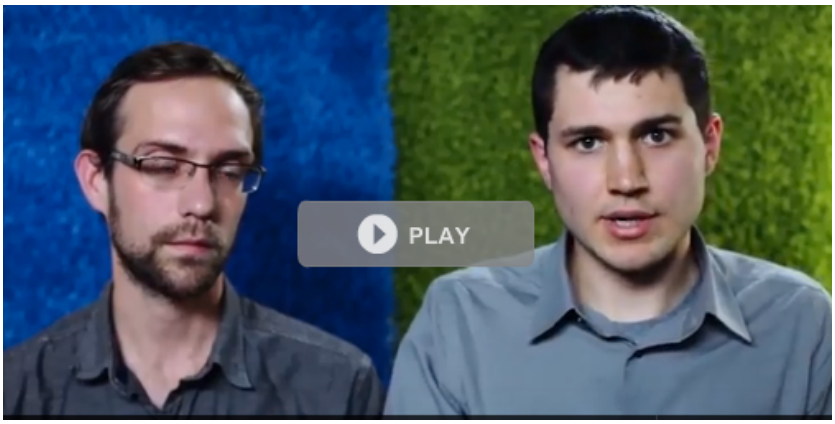
Abundant resources exist to help you create a résumé. For instance, Career Management Services (CMS) hosts a résumé building web site and runs résumé writing workshops. CMS also provides other career-building resources.

I've found the following web sites useful. You should be able to readily find other resources to help you plan your career.

- ACM Computer Careers
- Bourn College of Engineering Career Center Milestones

Video: Creative Coding

Computing is a creative activity. Use your creativity to accomplish things that will fill your résumé. Watch the "Art of Creative Coding" video (5:18)..



The video presents tools that can release your creativity.

1. Simplifying code with Processing – with Daniel Shiffman, NYU. Highlights how Processing empowers non-programmers to start coding quickly and enables them to create things not otherwise possible with off-the-shelf tools.

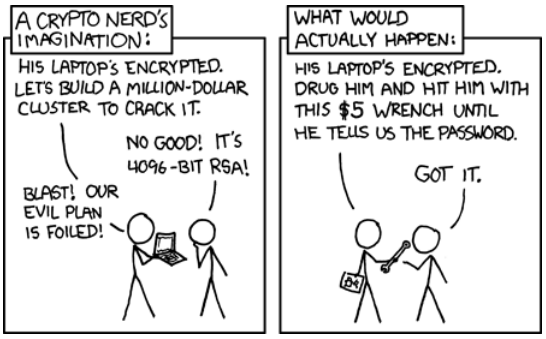
2. Professional coding with Cinder – with Keith Butters, Barbarian Group. Cinder's C++ library enables high-level professional creations.

3. Open Frameworks community & RGBDToolkit – with Jonathan Minard & James George. The toolkit enables a new form of cinema in which the user can engage.

Reading

Read How to Think About Security, by Whittaker and Ford

- What is the bank's "world view" of security and how does it allow their security to be breached?



Security

Funny Quotations Found on Résumés

It is good to laugh: Here are some quotes about résumés that I retrieved from Things People Said (05/24/2013)

- "I am very detail-oreinted."
- "Graduated in the top 66% of my class."
- "I worked as a Corporate Lesion."
- "Married, eight children. Prefer frequent travel."
- "Objective: To have my skills and ethics challenged on a daily basis."
- "Special skills: Thyping."
- "References: Leanne Kaye-Deceased Nov. 14, 1997."
- "I will take the chaos of your office and reform it into a semblance of simple disorder"
- "You will always find me to be an extremely trying person"
- "I can play well with others."
- "Reason for leaving last job: Pushed aside so the vice president's girlfriend could steal my job."
- "Previous experience: Self-employed – a fiasco."
- "I am a pit bull when it comes to analysis."
- "I love dancing and throwing parties."
- "I am quick at typing, about 25 words per minute."
- "I am a rabid typist."
- "Special Skills: Speak English."
- "Work Experience: Dealing with customers' conflicts that arouse."
- "Education: College, August 1880 - May 1984."
- "Seek challenges that test my mind and body, since the two are usually inseparable."
- "Personal Qualities: Outstanding worker; flexible 24 hours a day, seven days a week, 365 days a year."
- "I perform my job with effortless efficiency, effectiveness, efficacy, and expertise."

- "Personal: Married 20 years; own a home, along with a friendly mortgage company."
- "My intensity and focus are at inordinately high levels, and my ability to complete projects on time is unspeakable."
- "Exposure to German for two years, but many words are inappropriate for business."
- "Interests: I like to workout in my free time. I enjoy listening to music. I love to shopping in new places."
- "Accomplishments: Completed 11 years of high school."
- "Excellent memory; strong math aptitude; excellent memory; effective management skills; and very good at math."
- "Personal Goal: To hand-build a classic cottage from the ground up using my father-in-law."
- "Thank you for your consideration. Hope to hear from you shorty¡'
- "Please disregard the attached resume – it is terribly out of date."
- "It's best for employers that I not work with people."
- "If this resume doesn't blow your hat off, then please return it in the enclosed envelope."
- "You hold in your hands the resume of a truly outstanding candidate¡'
- "I am sicking and entry-level position."
- "Here are my qualifications for you to overlook."
- "I am a quick leaner, dependable, and motivated."
- "I am relatively intelligent, obedient, and as loyal as a puppy."
- "My compensation should be at least equal to my age."
- "My primary goal is to be recognized."
- "Below are the top 10 reasons to hire me."
- "I am superior to anyone else you could hire."
- "I vow to fulfill the goals of the company as long as I live."
- "I worked here full-time there."
- "I'll starve without a job but don't feel you have to give me one."
- "You are privileged to receive my resume."

5. *Read and Write about Computing, September 17*

If men learn this [writing], it will implant forgetfulness in their souls; they will cease to exercise memory because they rely on that which is written, calling things to remembrance no longer from within themselves, but by means of external marks.

Plato quoting Socrates, in *Phaedrus*

Either write something worth reading or do something worth writing.

Benjamin Franklin

Homework

[back to page 7]

1. Read and understand the Florida Tech's Standards on Academic Integrity & Academic Dishonesty.
2. Read and understand the Computer Sciences Department's Honor Code.
3. Use Lipson's three principles for academic integrity are Lipson (2004):
   (a) "When you said you did it, you actually did."
   (b) "When you use someone else's work you cite it, When you use their word, you quote it openly and accurately."
   (c) "When you present research materials, you present them fairly and truthfully. That's true whether the research involves data, documents, or the writing of other scholars."
4. Read the papers:
   (a) Reflections on Trusting Trust by Ken Thompson.
      - What technical issue does he address in constructing untrustworthy code?

- What is the moral of his lecture?

(b) How to Become a Hacker, by Eric S. Raymond

- How does Mr. Raymond answer the question: What is a hacker?
- What the components of the "hacker attitude?"
- What are the basic hacking skills?
- Mr. Raymond's basic skills were written in late 1996. Would you alter any of them?

(c) How to Think About Security, by Whittaker and Ford.

- How did the crooks break the bank?
- What is the bank's "world view" of security and how does it allow their security to be breached?

(d) No Silver Bullet, by Fred Brooks.

- Brooks describes four essential properties that make developing software systems hard. Describe these four properties.

(e) Principles of Traditional Animation Applied to 3D Computer Animation, by John Lasseter. Watch Luxo Jr., his video (1:54)

- What are the fundamental principles of traditional animation?

(f) The Making of an Expert, by Ericsson, Prietula, and Cokely.

- What are the three tests they say must be passed to determine if someone is a genuine expert?
- What advice do they give that can help you become an expert?

(g) Gödel's Undecidability Theorem, by Stephan Andrilli.

- What makes mathematical system consistent? When is a mathematical system inconsistent?
- What makes a mathematical system complete? When is a mathematical system incomplete?

- What are the implications of Gödel's result when applied to algorithms and problems?

(h) An Interview with Shafi Goldwasser & Silvio Micali, by Leah Hoffman.

- What is a zero knowledge interactive proof?

(i) A Categorical Manifesto, by Joseph Goguen.

- How is category theory, a mathematical abstraction developed to provide a foundation for describing structures and morphisms that preserve structures, applicable to computing?

(j) On the cruelty of really teaching computer science. I believe, and I am not alone, that Dijkstra will be chronicled in history for his contributions to computing.

- What is a radical novelty?
- What two assumptions Dijkstra does state to show computing is a radical novelty?
- What is his stance on education policy?
- What do you think of Dijkstra's ideas?

5. Pick one of the papers and write an essay on it. For the chosen paper, write a short essay on what you learned.

6. Your essay must have at least: Your name as the author, a title, a submission date, an abstract, an introductory paragraph, a body, a concluding paragraph, and a bibliography. Do not plagiarize.

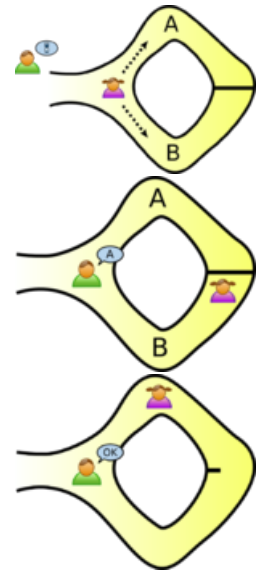
7. Your essay must demonstrate an ability to write using a "document preparation system." My strong bias is for you to learn and use LaTeX 2_ε. Other systems (e.g. Word) may also be used.

8. Save your essay as a PDF file.

9. Submit your essay using the course management system.

10. Complete the assignment by November 25.

Ali Baba says he knows a secret phrase to open a cave door. Kasim will pay a bag of gold for the word, but only if he is certain Ali knows the phrase. Ali will not tell Kasim the phrase until he is paid. How can Kasim be certain Ali knows the phrase?



- Ali enters the cave without Kasim watching and takes path A or B.
- Kasim yells: Come out from path A (or B).
- If Ali does not come out the called path, Kasim knows Ali does not know the phrase.
- The probability that Ali enters along the path Kasim yells out is 1/2.
- If Ali consistently comes out the called path, say 100 times, then Kasim can be confident Ali knows the phrase. The probability of being right 100 times in a row is only about 1 in 1.2×10^{30}

You'll ask: What does *short* mean? Write as long as you have interesting remarks. If you had more time, you might be able to write less.

"Je n'ai fait celle-ci plus longue que parce que je n'ai pas eu le loisir de la faire plus courte," Literally: I made this (letter) very long, because I did not have the leisure to make it shorter. Blaise Pascal, 1657.

Writing

Harlan Mills was a strong proponent of reading before writing. He was discussing programming languages: You should read code. It will help you to write code.

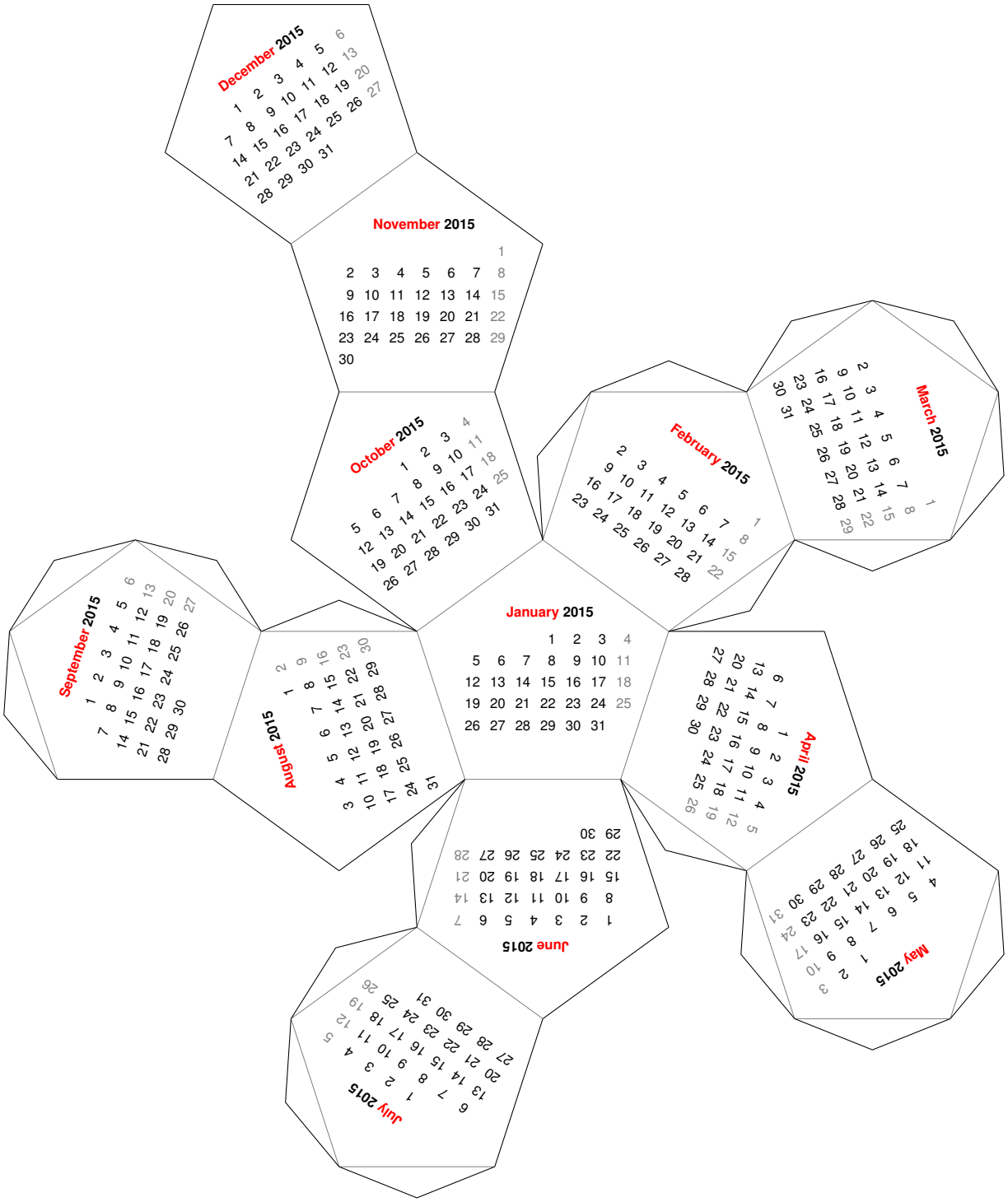
It is also useful to use code to write in a natural language. I recommend LaTeX because it is very versatile. Here is Don Knuth's earth-shaking announcement of iTeX.

LaTeX is a mark-up language. The philosophy of mark-up languages is that author should be able to focus on the content of what they are writing without being distracted by its visual presentation. ' The visual presentation should be programmed from the content.

Here's LaTeXand Ti*k*Z code for the dodecahedron calendar on page 42. I give it as an example of a neat program to code the organization of text.

```
\documentclass{article}
% Folding + calendar example from the PGF manual.
%
% Author: Till Tantau
\usepackage{tikz}
\usetikzlibrary{calendar,folding}
\begin{document}
  \sffamily\scriptsize
  \begin{tikzpicture}[transform shape,
    every calendar/.style={
      at={(-8ex,4ex)},
      week list,
      month label above centered,
      month text=\bfseries\textcolor{red}{\%mt} \%yo,
      if={(Sunday) [black!50]}
    }]
 \tikzfoldingdodecahedron[
   folding line length=2.5cm,
   face 1={ \calendar [dates=\the\year-01-01 to \the\year-01-last];},
   face 2={ \calendar [dates=\the\year-02-01 to \the\year-02-last];},
   face 3={ \calendar [dates=\the\year-03-01 to \the\year-03-last];},
   face 4={ \calendar [dates=\the\year-04-01 to \the\year-04-last];},
   face 5={ \calendar [dates=\the\year-05-01 to \the\year-05-last];},
   face 6={ \calendar [dates=\the\year-06-01 to \the\year-06-last];},
   face 7={ \calendar [dates=\the\year-07-01 to \the\year-07-last];},
   face 8={ \calendar [dates=\the\year-08-01 to \the\year-08-last];},
   face 9={ \calendar [dates=\the\year-09-01 to \the\year-09-last];},
   face 10={\calendar [dates=\the\year-10-01 to \the\year-10-last];},
   face 11={\calendar [dates=\the\year-11-01 to \the\year-11-last];},
   face 12={\calendar [dates=\the\year-12-01 to \the\year-12-last];}
```

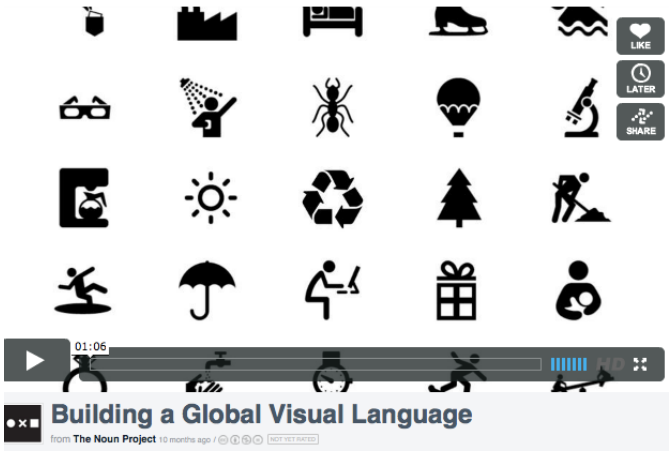
```
    ];
  \end{tikzpicture}
\end{document}
```


December 2015
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
November 2015
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
October 2015
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
September 2015
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30
August 2015
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
July 2015
1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
June 2015
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
May 2015
1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
April 2015
1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
March 2015
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
February 2015
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28
January 2015
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31

.

The Noun Project

I've recently found the Noun Project, which "is building a global visual language that everyone can understand. We want to enable our users to visually communicate anything to anyone." Watch one of their introductory videos and consider how you might improve your communication skills using their services.



Additional Readings

There is so many more good books and articles that have been written about computing. I encourage you to read as much as you can. To get you started, I've collected a few references that I like and listed them below. I hope you find them interesting too. Some of them will be easy to read and understand. Others will be more difficult to read, but worth the effort to read and understand.

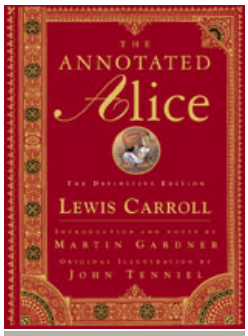
Alice's Adventures in Wonderland by Lewis Carroll, is as Alan Perlis said is "the best book on programming for the layman is Alice in Wonderland; but that's because it's the best book on anything for the layman." Perlis could have also included *Through the Looking-glass* (Carroll et al., 2000). Jon Bentley's *Programming Pearls* (Bentley, 1986) and (Bentley, 1988) are classic collections of articles on interesting and useful computing topics. These articles originally appeared in the Communications of the ACM and can be found in the ACM Digital Library. If you can find it, Bentley's *Writing Efficient Programs* is a good read (Bentley, 1982). The books Kernighan and Plauger (1978), Kernighan and Pike (1999), Hunt and Thomas (2000), and McConnell (1993) also inspire good programming practice. All papers in *Beautiful Code* Oram and Wilson (2007) are worth a look.

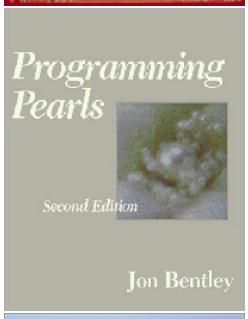
David Parnas and Peter Denning are prolific writers. They made significant discoveries in software engineering (Parnas, 1972), operating systems (Denning, 1968), and the broad fields of computing. You'll find many articles they have written, but I'll only mention a few others. In (Parnas, 2010), Parnas discusses the 40 year struggle to find engineering methods for creating software. In (Parnas, 1996), Parnas stresses the need to engineer software by following basic rules.

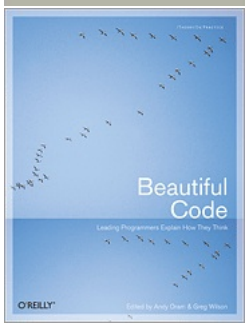
1. Design before implementing.
2. Document your design.
3. Review and analyze the documented design.
4. Review implementation for consistency with the design.

In (Denning, 2004), Denning offers another way to view the discipline of computing: He argues it rests on computation, communication, coordination, automation, and recollection.

Much heavier books are Don Knuth's classic volumes *The Art of Computer Programming* (Knuth, 1997a,b) and (Knuth, 1998). You'll only be able to read parts of these encyclopedic works, but you will learn much from the effort. Knuth's response to the `go to` controversy (Knuth, 1974) introduces interesting ideas about programming.

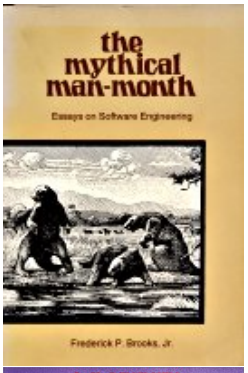


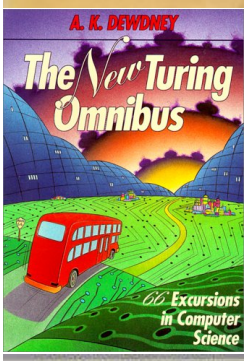


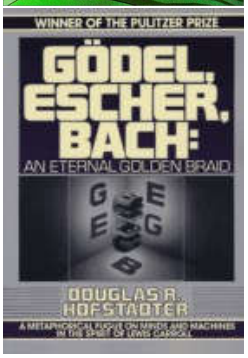


The origins of software engineering are traced to the 1968 NATO report by Peter Naur and Brian Randell (Naur and Randell, 1969). A follow-up 1970 report by Buxton and Randell outlines software engineering techniques. The 2000 conference proceedings (Finkelstein, 2000) on the future of software engineering provides a more recent overview of software engineering.

Fred Brooks is another interesting computer scientist and author. His *Mythical Man-Month* (Brooks, 1995) describes the development of IBM's 360 operating system, an early complex software project. In (Brooks, 1987), Brooks argues there is no "silver bullet" to kill the werewolves that software projects become. Brooks' short paper (Brooks, 1996) gives his view of what a computer scientist is and does. A. K. Dewdney's *The (New) Turing Omnibus: 66 excursions in computer science* (Dewdney, 1993) describes many interesting concepts and ideas in computer science: Algorithms, Finite Automata, Systems of Logic, Gödel's Theorem, Chomsky's Hierarchy, Random Numbers, Computer Vision, Non-determinism, CAT Scanning, the Fast Fourier transform, Turing machines, Computer Viruses, and many other 54 other excursions. *Gödel, Escher, Bach: An Eternal Golden Braid* by Doug Hofstadter (Hofstadter, 1999) weaves the story of recursion in music, art, and logic through the creations of original thinkers in these fields. The book is long, but reading and reporting on portions of it will expand your thoughts. *Code: The Hidden Language of Computer Hardware and Software* (Petzold, 2000) by Charles Petzold offers a low-level look at computer organization. It would be a good book to read to prepare for CSE 2120 Computer Organization and Machine Language.







James Glick's new book *The Information: A History, a Theory, a Flood* (Glick, 2011) posits that the "bit" is the fundamental particle of the universe. He offers an interesting tour of how information has been encoded and communicated throughout history.

Kode Vicious writes for the ACM. You may find his articles interesting. Recent ones are referenced below.

Brian Hayes writes for the American Scientist. You may find his articles interesting too. For instance, his article titled "How Many Ways Can You Spell V1@gra?" (Hayes, 2007) may entice you to check him out. The random writings of Eric S. Raymond are fun too Raymond (2010).

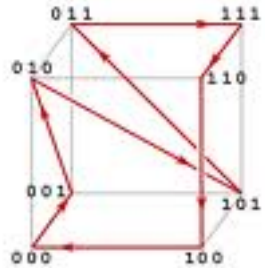
I like computer graphics. A good reference is Jim Blinn's book (Blinn, 1996).

Joel Spolsky writes, Joel on Software a blog where he's been ranting about software development, management, business, and the Internet since 2000. Michael Cusumano and Richard Selby discuss how Microsoft develops software in (Cusumano and Selby, 1997).

I find quantum and biological computers fascinating ideas. With them you can execute non-deterministic algorithms greatly increasing in efficiency for hard problems. For instance, in 1994, Leonard Adleman[1] demonstrated how to solve the Directed Hamiltonian Path (DHP) problem[2] using biological operations (Adleman, 1994). Adleman's algorithm is non-deterministic and solves the NP-complete

[1] Yes, Adelman is the "A" in the famous RSA public-key encryption algorithm.

[2] Given a directed graph is there a path through it that visits every node once and only once? For instance, the diagram below, from one of Mark Dow's pages on combinatorics shows a Hamiltonian path on the 3-dimensional cube.



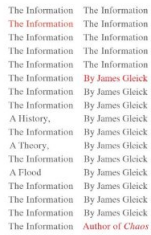
DHP in polynomial time. Equally as impressive, in 1994, Peter Shor (Shor, 1994) proposed a quantum algorithm compute the prime factors of a natural number in polynomial time, making RSA encryption vulnerable. Both papers are deep, but you can find introductory articles that explain the ideas in more simple terms, for example, (Bacon and van Dam, 2010) introduces quantum computing basics and Fortnow (2009) describes current understanding of the P vs. NP problem. George Dyson has written an interesting history of early computing called *Turing's Cathedral: The Origins of the Digital Universe* (Dyson, 2012). The rolls played John von Neumann and the Institute for Advanced Studies at Princeton University in the development of computers provide the framework for the story. *Tubes: A Journey to the Center of the Internet* (Blum, 2012) is a new book by Andrew Blum that is getting good reviews and is on my to-read list. *Blown to Bits*, Your Life, Liberty, and Happiness after the Digital Explosion, by Abelson, Ledeen, and Lewis is another interesting book that outlines what has happened and may soon happen in the digital age. (Abelson et al., 2008) The book is free to download.

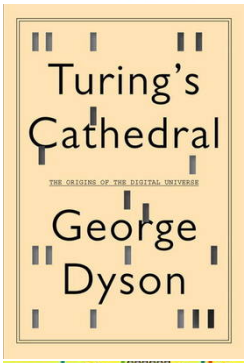
The history of computing is full of good stories. *Out of Their Minds* (Shasha and Lazere, 1995) and *Portraits in Silicon* (Slater, 1987) offers biographies of many famous people from the field of computing. In the Communications of the ACM (CACM) and other professional magazines you find stories about the history of computing. I found the recent paper by Campbell-Kelly (2010) about the consequences of the disappearance of books of mathematical tables interesting. Computing has had significant impacts on society. One famous story is how a simple programming mistake lead to multiple deaths from over radiation (Leveson and Turner, 1993). Recent CACM articles such as Greengard (2010), Creeger (2010), and Juels (2010) discuss other impacts computing has had on society.

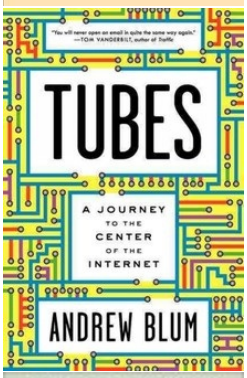
Although the works (Tufte, 1990, 1997, 2001, 2006) by Edward Tufte are not about computing *per se*, they are classic books on how to present data as information. I recommend them to you.

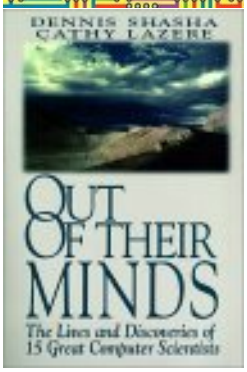
Almost lastly, let me point you to two books that can help prepare you for job interviews. They are *Programming Interviews Exposed* (Mongan and Suojanen, 2000) and *How Would You Move Mt. Fuji?* (Poundstone, 2003).

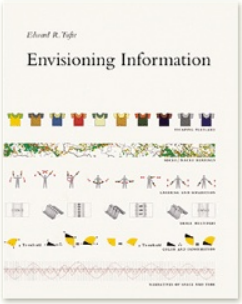
And lastly, there are some books that can help you become a better writer. The classic style manual is Strunk & White's *The Elements of Style*, (Strunk Jr. and White, 1979). The *Chicago Manual of Style* is another standard reference (of Chicago Press Staff, 2010). I like *Bugs In Writing*, by Lyn Dupre, (Dupré, 1995). She has edited many computer science books, and has a nice writing style.





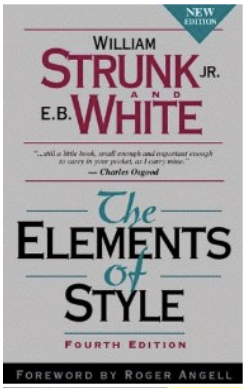


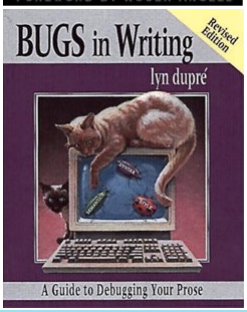


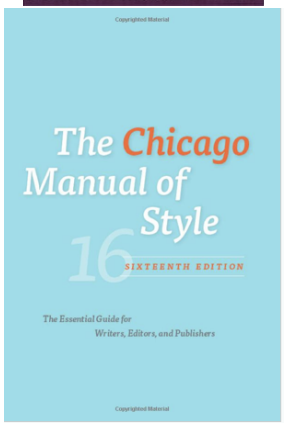


Additional Reading Lists

The ACM has a classic book series[3] where members have named their favorite computer science books. Some of these books can be a little intimidating for beginners. Dr. Stansifer also maintains a list of books that he likes, and they are mostly more accessible. [4]







[3] See dl.acm.org/classics.cfm

[4] See http://cs.fit.edu/~ryan

6. Intern Panel, September 24

You are stup. . . er, silly if you don't do an internship!

William David Shoaff

Action Items

- Compile a list of questions you would lke to ask students who have intern experience.

Internship & Co-op Student Panel

An intern works for an organization in a temporary job to learn about a profession. Students can earn academic credit that applies toward graduation requirements by enrolling in cooperative education classes, e.g., CWE 1001.

Students who have recently completed an internship will discuss their experiences. Some of the companies where students have had recent internships include:

- Amazon
- Electronic Arts
- Faculty Funded Research
- Google
- Liquidnet
- Harris
- Microsoft
- Modus Operandi
- Northrop Grumman
- Willow Garage
- Zynga

Questions For The Panel

1. Where did you work?
2. How did you get your internship?

3. What did you do?
4. Was it worthwhile?
5. Do you have advice for other students?

Reading

Read No Silver Bullet, by Fred Brooks. Be prepared to discuss:

- Brooks describes four essential properties that make developing software systems hard. Describe these four properties.

Google Internships

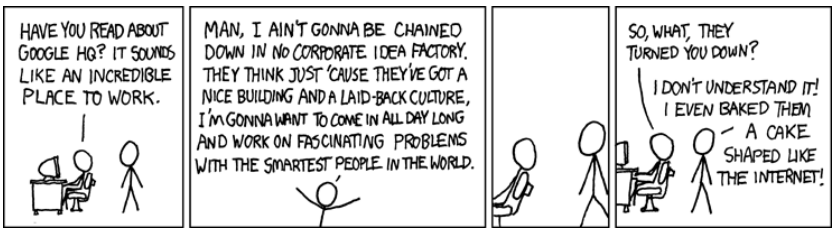
Here's Google's view of interns' first week. One of the stars is our own Kim Day.

On a lighter side, you might enjoy the Internship movie, which is one view of working at Google

7. Employer Panel, October 1

If A equals success, then the formula is: A equals X plus Y plus Z. X is work. Y is play. Z is keep your mouth shut.

Albert Einstein, as quoted in Kiplinger's Personal Finance, Vol. 11, No. 7 (July 1957), p. 48



Working for Google

Action Items

- Compile a list of questions you would like to ask potential employers.

Campus Recruiters

During the Fall and Spring Career Fairs, several organizations come to campus to recruit students for internships and full-time employment. A panel of campus recruiters, chaired by by Ms. Dona Gaynor, Director of Career Management Services, will present their opinions and facts about career opportunities in computing.

Companies that may be represented include:

- Harris
- Microsoft
- Northrop-Grumman
- Raymond James

Questions For The Panel

1. What knowledge, skills, and behaviors does your organization expect of interns and employees?

2. Can you describe a typical employment interview?

3. Can you give an overview of what an employee would do at your company?

4. What are the key qualities needed to be successful at your company?

5. What do you consider the ideal experience for a job at your company?

6. What career and professional accomplishments does your organization expect of employees who have earned a Bachelor's degree in computer science or software engineering, that is, where will I be in five years?

Some Companies Where Graduates Work

Careers

The ACM Careers site identifies three main categories where computer scientist and software engineers work.

1. Designing and building software

   Career Path: Work in software development which includes aspects of web development, interface design, security issues, mobile computing, and so on. This is the career path that the majority of computer science graduates follow. Opportunities occur in large or small software and computer services companies, and organizations of all kinds (industry, government, banking, health care, etc.).

2. Developing new and effective ways to solve computing problems, such as storing information in databases, sending data over networks or providing new approaches to security problems

   Career Path: Developing effective ways to solve computing problems. This refers to the application or development of computer science theory and knowledge of algorithms to ensure the best possible solutions for computationally intensive problems. As a practical matter, a career path in the development of new computer science theory typically requires graduate work to the Ph.D. level, followed by a position in a research university or an industrial research and development laboratory.

3. Devising new and better ways of using computers and addressing particular challenges in areas such as robotics, computer vision, or digital forensics (although these specializations are not available in all computer science programs).

   Career Path: Innovate in the application of computer technology, which can involve advanced graduate work, followed by a position in a research university or industrial research and development laboratory; it can involve entrepreneurial activity such as was evident during the dot-com boom of the 1990s; or it can involve a combination of the two.

Top Job Predictions

Dice outlines the top five jobs for computer science graduates, *By Don Willmott | Apr 9, 2013*.

1. Software Architecture & Development

2. Mobile Application Development

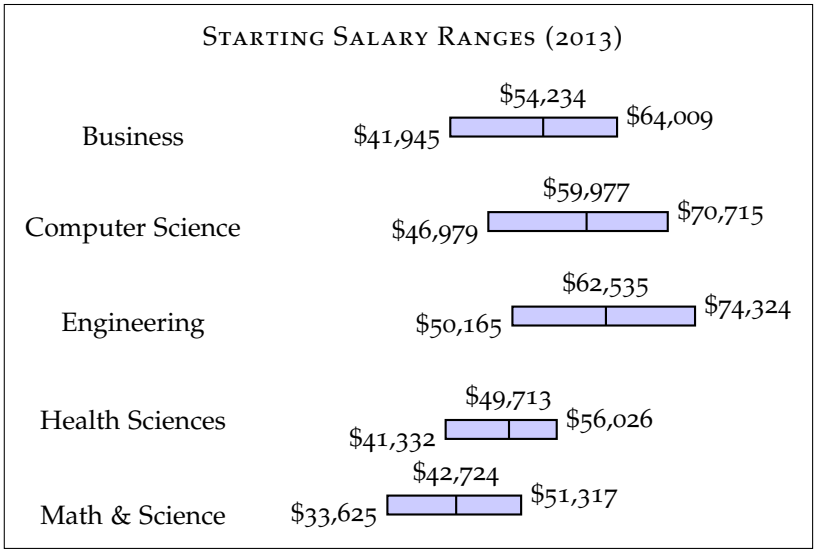
3. Big Data Analytics
4. Health-care IT
5. Video Game Design

The NACE Salary Survey

The National Association of College and Employers (NACE) reports on salaries, top industries, top paying positions, employment by major, and other data about the hiring of college graduates. The material is copyrighted so that I cannot share it with you directly.

Professor hint: Check your "u-drive" to display the report.

Category	Average Starting Salary 2013	Average Starting Salary 2012	Percent Change
Business	$54,234	$50,633	7.1%
Computer Science	$59,977	$57,529	4.3%
Engineering	$62,535	$60,151	4.0%
Health Sciences	$49,713	$45,442	9.4%
Math & Sciences	$42,724	$41,430	3.1%
Overall	$44,928	$42,666	5.3%



Video: Advice on Job Searches

Congratulations you graduated: Here's eleven reasons why I won't hire you.

Contributed by Edwin Capel.


H|B
CONGRATULATIONS
GRADUATE!
Eleven Reasons Why I Will Never Hire You.
by Mark O'Toole - PR & content at HB Agency

8. Curricula, October 8

I think they assign things to students which are way over their heads, which destroy your love of reading, rather than leading you to it. I don't understand that. Gosh.

Charles M. Schulz, Conversations

Education must provide the opportunities for self-fulfillment; it can at best provide a rich and challenging environment for the individual to explore, in his own way.

Noam Chomsky,(1928–), U.S. mathematical linguist, philosopher, psychologist, political critic.

Video: Ok computer science is . . .

Paul Stansifer has a brilliant video on computer science.

Homework

[back to page 7]

1. Download the (editable) program plan for your major. Computer Science or Software Engineering. These plan are on the depart-

ment's web site under academics.

2. You should ask the department's student coordinator, Ms. Rosalyn Edwards, if she has recorded transfer, advanced placement and other credits to the department kept version of your program plan. Ask her to send you a copy of your program plan.
3. Schedule an appointment with your freshman advisor[5] to plan your courses for the Spring semester. Add this information to your program plan.
4. Review your program's requirements and prepare for a visit with your faculty advisor[6].
5. Schedule a visit with your faculty advisor. Based on your interests and goals, and their advice, Complete an initial draft of your program plan by "penciling in" the elective courses you will take.
6. Submit your completed program plan on the course management system.
7. Complete this assignment by October 25.

[5] If you have earned 30 or more credits, then schedule an appointment with your faculty advisor.

[6] Ms. Edwards assigns students to faculty advisors. You should be able to see this information on PAWS. If you don't find it there, ask her who your faculty advisor is.

Use the information below to help you plan your academic courses.

What is a Curriculum?

Medieval universities (1000–1200) established a curriculum called the "trivium" than included three subjects.

1. Grammar — the mechanics of a language (the thing as-it-is-symbolized)
2. Logic — the mechanics of thought and analysis (the thing as-it-is-known)
3. Rhetoric — the use of language to instruct and persuade (the thing as-it-is-communicated)

The ability to understand and use these skills are crucial to computer science and software engineering. Medieval scholars where interested in natural languages, while we are interested in "machine" languages and their uses.

Mastery of the trivium prepared students to study the "quadrivium."

1. Arithmetic — quantity and operations on quantities
2. Geometry — properties of space
3. Music — sound, pitch, rhythm, volume, timbre
4. Astronomy — celestial objects

Computer Science and Software Engineering Curricula

The common computing courses in the first two years are listed below.

1. CSE 1001 Fundamentals of Software Development 1 — the mechanics of instructing a machine (writing programs)
2. CSE 1002 Fundamentals of Software Development 2 — the study of higher order concepts on instructing a machine (organizing programs)

"Science, my boy, is made up of mistakes, but they are mistakes which it is useful to make, because they lead little by little to the truth." (Verne, 1864)

"The engineering method is the use of heuristics to cause the best change in a poorly understood situation within the available resources." (Koen, 2003)

See Programming Excuses for nifty answers as to why your programs don't work. Refresh for a new excuse. (Shared by Rachel Brindle)

Ineffective Sorts

3. CSE 1101 Computing Disciplines and Careers — why study these subjects
4. CSE 1400 Applied Discrete Mathematics — tools for analyzing computing problems
5. CSE 2120 Computer Organization and Machine Language — a physical machine and its instructions

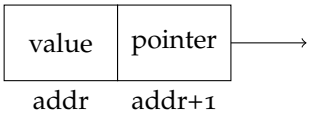
   The basic five-stage pipeline in a computer: instruction fetch (IF), instruction decode (ID), execute (EX), memory access (MEM), register write-back (WB).

IF	ID	EX	MEM	WB				
	IF	ID	EX	MEM	WB			
		IF	ID	EX	MEM	WB		
			IF	ID	EX	MEM	WB	
				IF	ID	EX	MEM	WB

Time →

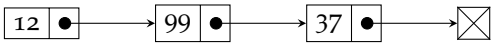
6. CSE 2010 Algorithms and Data Structures — fundamental ways to organize and operate on data

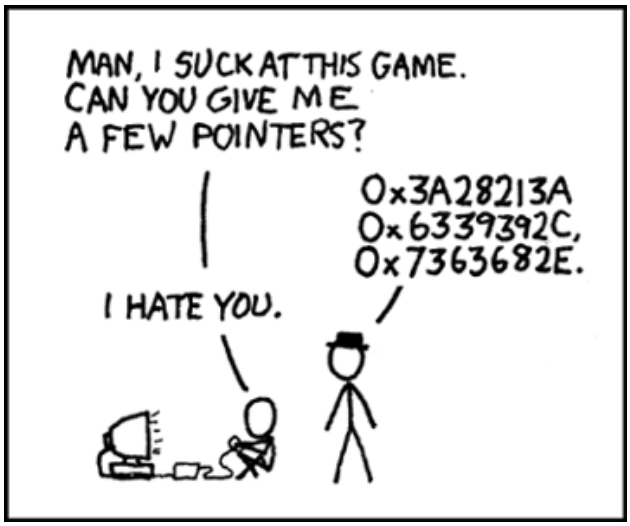
   - Linked Lists:



A node in a linked list contains a value and a pointer (a link) the next node. A linked list generalizes the concept of "array." An array is a block of contiguous memory, while a linked list can have values distributed in memory.

A terminated linked list of nodes





Pointers

   - Stacks

     A stack supports operations such as
     - *push* a value onto the stack
     - *pop* a value from the stack
     - *top* examine the value at the top of the stack

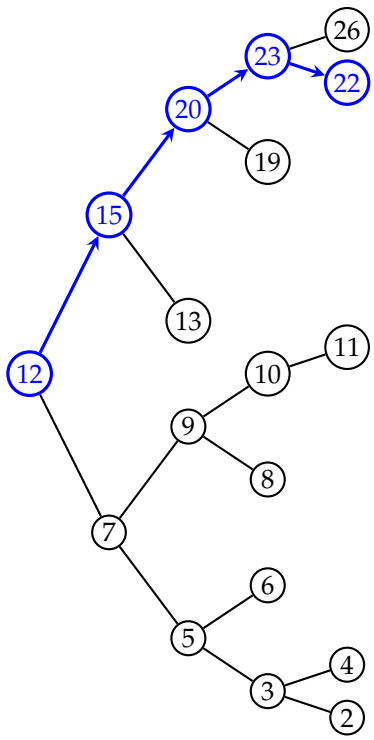
     A stack is "last in, first out" (LIFO) list

   - Queues

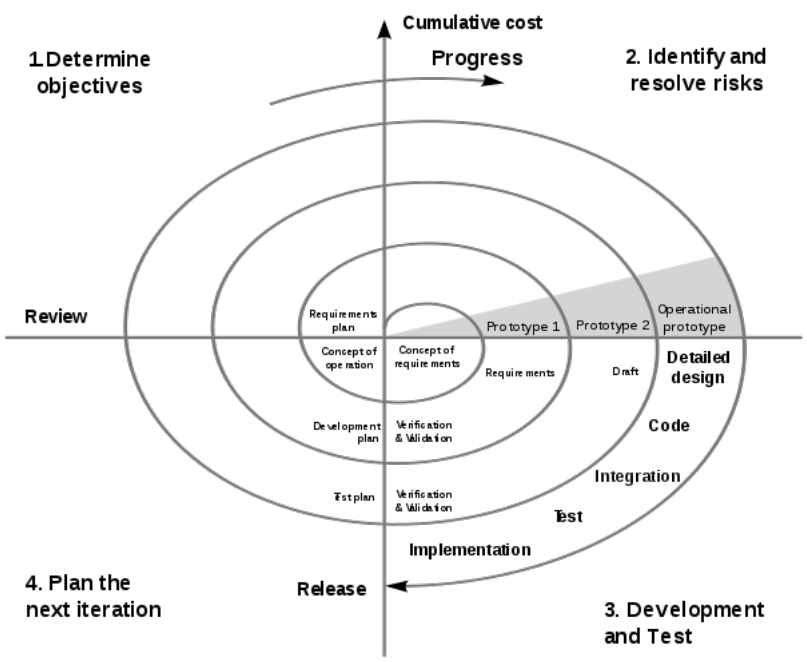
     A queue supports operations such as
     - *enqueue* append a value at the end of the queue
     - *dequeue* append a value from the first of the queue

     A queue is "first in, first out" (FIFO) list

   - Binary Search Trees



7. CSE 2410 Introduction to Software Engineering — application of a systematic, disciplined, quantifiable approach to the design, development, operation, and maintenance of software (Wikipedia)



8. CSE 2050 Programming in a Second Language — learn to read

and write in another programming language

```
// my first program in C++

#include <iostream>
using namespace std;

int main ()
{
  cout << "Hello World!";
  return 0;
}
```

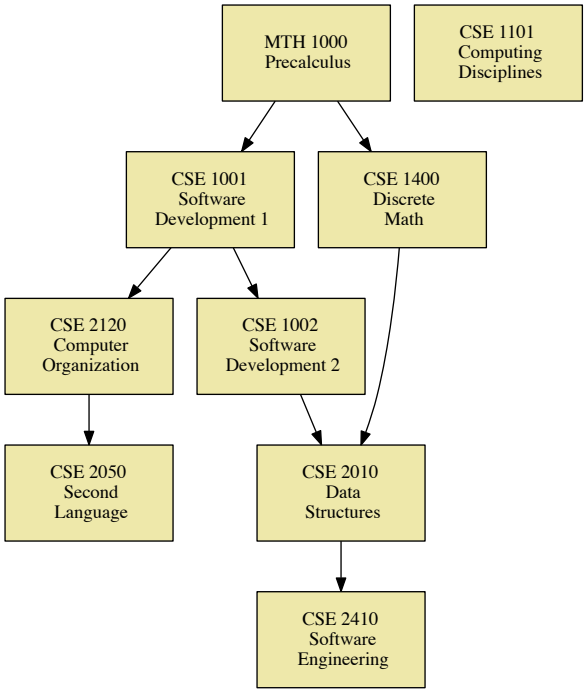


Figure 1: Prerequisite structure for the first two years. Note if you don't place beyond Precalculus you are a semester behind and need to work hard to catch up.

The Final Two Years

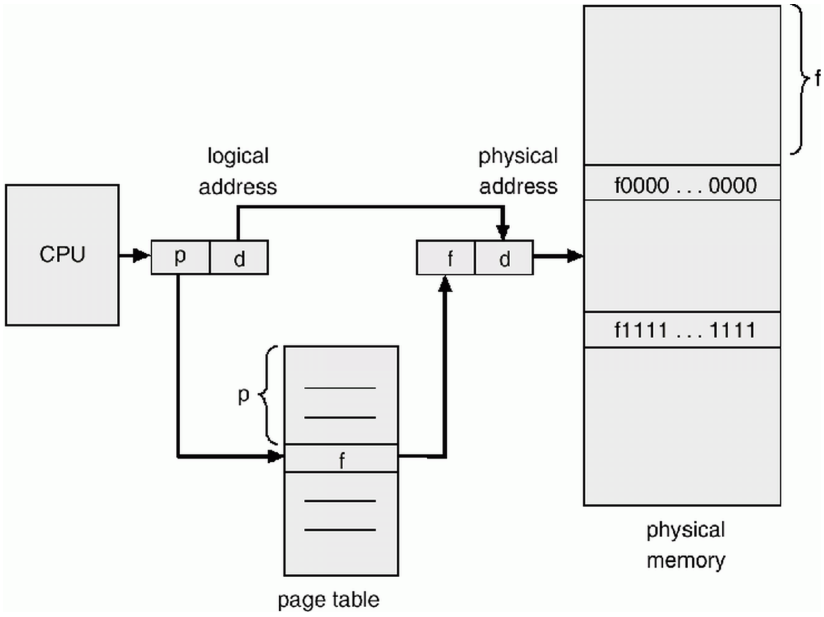
1. CSE 2400 Applied Statistics — learning tools to use in analyzing data

2. CSE 3030 Legal, Ethical, and Social Issues in Computing — the impact of computing

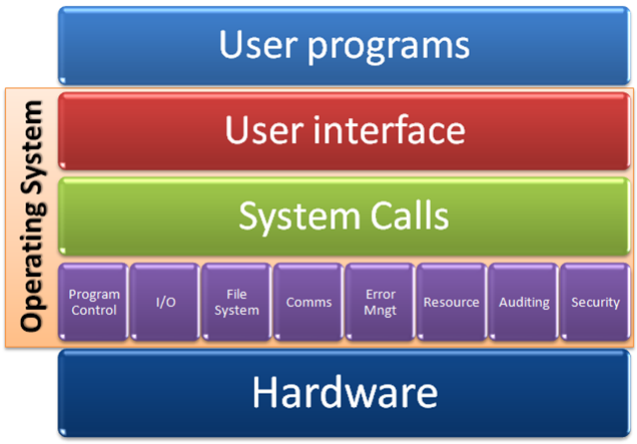
Ethics in Computing

3. CSE 4001 Operating Systems Concepts — the interface between the machine and software applications

   An operating systems manages the hardware resources and provides services for application programs. One part of an operating systems is its paging service.



   A computing system is based on layers.



   The kernel of an operating systems provides the basic services and interacts directly with hardware.

4. CSE 4[1|2]01 & 4[1|2]02 Computer | Software Projects 1 & 2 — to demonstrate mastery of the curriculum

   Students work (in teams) to develop an interesting and useful software application. You can view a list of projects here

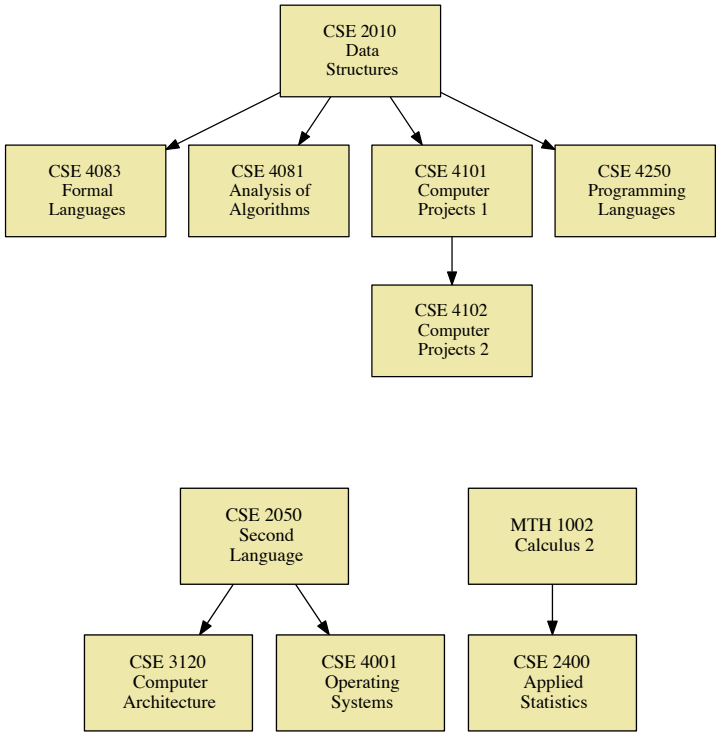


Figure 2: Prerequisite structure for the final two years in computer science

Computer Science

Computer Science majors must also complete the following courses.

1. CSE 3120 Computer Architecture and Assembly Language —

   The study of how to make the instruction pipeline and related problems as fast and efficient as possible. A fundamental measure of a computer's performance on a given program is *CPU time*.

   $$\text{CPU time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

2. CSE 4081 Introduction to Analysis of Algorithms | CSE 4083 Formal Languages and Automata Theory

3. CSE 4250 Programming Language Concepts

> "Hence, a generative grammar must be a system of rules that can iterate to generate an indefinitely large number of structures. This system of rules can be analyzed into the three major components of a generative grammar: the syntactic, phonological, and semantic components."
> "The syntactic component of a grammar must specify, for each sentence, a deep structure that determines its semantic interpretation and a surface structure that determines its phonetic interpretation."
>
> Noam Chomsky, Aspects of the Theory of Syntax, (1965).

Computational paradigms

(a) Imperative: Tell the machine what do to: `goto`, `if-then-else`
(b) Functional: Apply a function to data to generate new data: $y = f(x)$
(c) Logical: $(\text{P} \Rightarrow \text{Q}) \wedge (\neg\text{P} \Rightarrow \text{R}) \Rightarrow (\text{Q} \vee \text{R})$

4. Three CSE 3000+ Electives, Two Technical[7] | Business Electives[8]

[7] Science, engineering, or mathematics courses
[8] I recommend these business courses.

(a) BUS 3501 Management Principles
(b) BUS 3504 Management Information Systems
(c) BUS 4502 Organizational Behavior

Software Engineering

Software Engineering majors must also complete the following courses.

1. CSE 3411 Software Testing 1 — how to use the input/output behavior of a software application to verify[9] and validate[10] its quality.

2. CSE 3421 Software Design Methods — How to decompose a problem into parts that can be reassembled into a solution of a problem.

3. CSE 4083 Formal Languages and Automata Theory — the study of machines and the languages they can understand.

[9] Is it built right?
[10] Are you building the right thing?

4. CSE 4415 Software Testing 2 — How to use the code to verify and validate a software application.
5. CSE 4611 Software Metrics and Models — How to estimate the costs: line of code, personnel, time, etc., of a software project.
6. CSE 4621 Requirements Engineering — How to write requirements that clients understand and developers can implement.
7. AHF 3101 Introduction to Human Factors — Learn to take ergonomics into account in the development of software application.
8. Two CSE 3000+ Electives & One Free Elective.

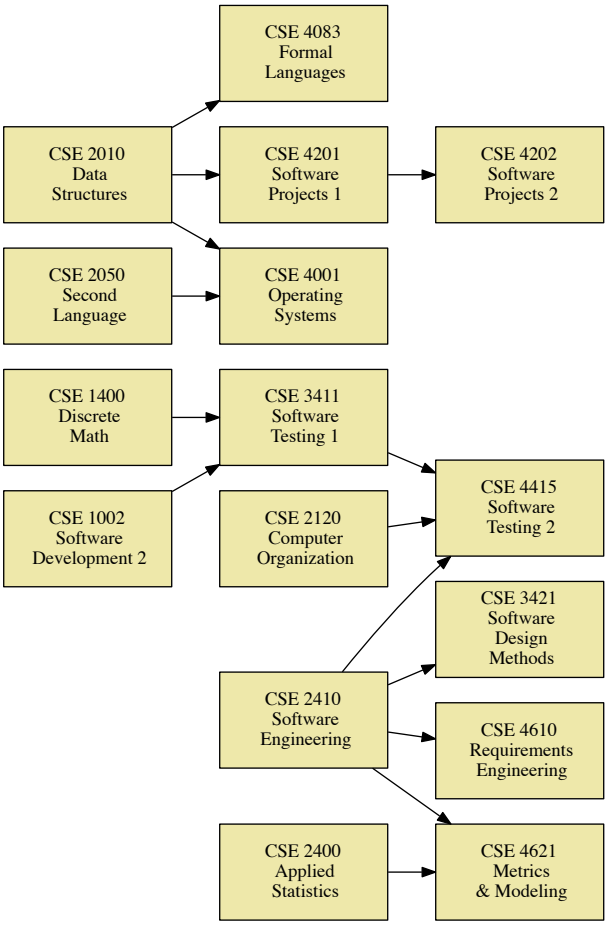


Figure 3: Prerequisite structure for the final two years in software engineering

Science and Mathematics

1. MTH 1001 Calculus 1

2. MTH 1002 Calculus 2 '
3. One Advanced Mathematics Electivein[-2]
4. Biology 1 & 2 | Chemistry 1 & 2 | Physics 1 & 2 with labs
5. Two Science Elective
6. One Science | Mathematics Elective

[-2] I rcommend

- MTH 3051 Combinatorics and Graph Theory
- MTH 3102 Introduction to Linear Algebra
- MTH 3401 Introduction to Number Theory
- MTH 4082 Introduction to Parallel Processign
- MTH 4311 Numerical Analysis

Humanities, Liberal Arts Social Sciences

1. If (Computer Science), then Social Science Elective, else if (Software Engineering), then PSY 1400 Introduction to Psychology
2. COM 1101 Composition and Rhetoric, COM 1102 Writing About Literature, COM 2212 Research Sources and Systems, COM 2223 Scientific and Technical Communication
3. HUM 2051 Civilization 1, HUM 2510 Logic, Humanities Core Course, Humanities Elective
4. One Free Elective.

Possible Options & Concentrations

1. Computer Science Majors — use your computer science, technical, and free electives to concentrate in an area of computing. Each class is three credits.
   - Computational Intelligence Track
     (a) CSE 4301 Introduction to Artificial Intelligence
     (b) (New) CSE 4302 Machine Learning and Data-mining
     (c) CSE 4303 Speech Recognition Programming
     (d) (New) CSE 4304 Complex Networks
     (e) (New) CSE 4305 Computer Vision
   - Games Track
     (a) (New) CSE 4231 Computer Networks
     (b) CSE 4280 Computer Graphics Algorithms
     (c) CSE 4257 Graphical User Interfaces
     (d) CSE 4285 Game Design
     (e) CSE 4301 Introduction to Artificial Intelligence
   - Information Assurance & Cybersecurity Track
     (a) CSE 4272 Computer and Information Security
     (b) (New) CSE 4231 Computer Networks

(c) Three courses to be determined

- Software Engineering Track

(a) CSE 3411 Software Testing 1
(b) CSE 3421 Software Design Methods
(c) CSE 4410 Software Project Management
(d) CSE 4415 Software Testing 2
(e) CSE 4611 Software Metrics and Models
(f) CSE 4621 Requirements Engineering

- Systems Track

(a) (New) CSE 4231 Computer Networks
(b) CSE 4232 Computer Network Programming
(c) (New) CSE 42XX Embedded Systems
(d) Two course to be determined

- Theory Track

(a) CSE 2500 Combinatorics and Graph Theory
(b) CSE 4020 Database Systems
(c) CSE 4081 Introduction to Analysis of Algorithms
(d) CSE 4083 Formal Languages and Automata Theory
(e) CSE 4251 Compiler Theory
(f) CSE 4301 Introduction to Artificial Intelligence
(g) One course to be determined

- Untrack Track

(a) Five advanced computer science elective courses

2. Software Engineering Majors — You have fewer electives, but wise choices can add value to your required courses.

9. Curricula Part 2, October 15

For truth to tell, dancing in all its forms cannot be excluded from the curriculum of all noble education: dancing with the feet, with ideas, with words, and, need I add that one must also be able to dance with pen- that one must learn how to write

---

Friedrich Nietzsche, Twilight of the Idols

Action Items

- Think of a subject you would like to learn about. Is there a minor you could pursue that to learn about this field?

Knowledge Areas of Computer Science

Figures 4 through 7 on pages 70 through 73 present a two-level concept (mind) map of topics belonging to the field of computer science.

Knowledge Areas of Software Engineering

Figures 8 and 9 on pages 74 and 75 present a two-level concept (mind) map of topics belonging to the field of software engineering.

Minors

If you wisely choose your electives, the following minors can be earned with little or no extra course work

1. Biology — take the following courses
   (a) BIO 1010 Biological Discovery 1 (4 credits)
   (b) BIO 1020 Biological Discovery 2 (4 credits)
   (c) At least one additional 4 credit laboratory course
   (d) 7–9 additional credits in biology

Figure 4: Knowledge Areas of Computer Science

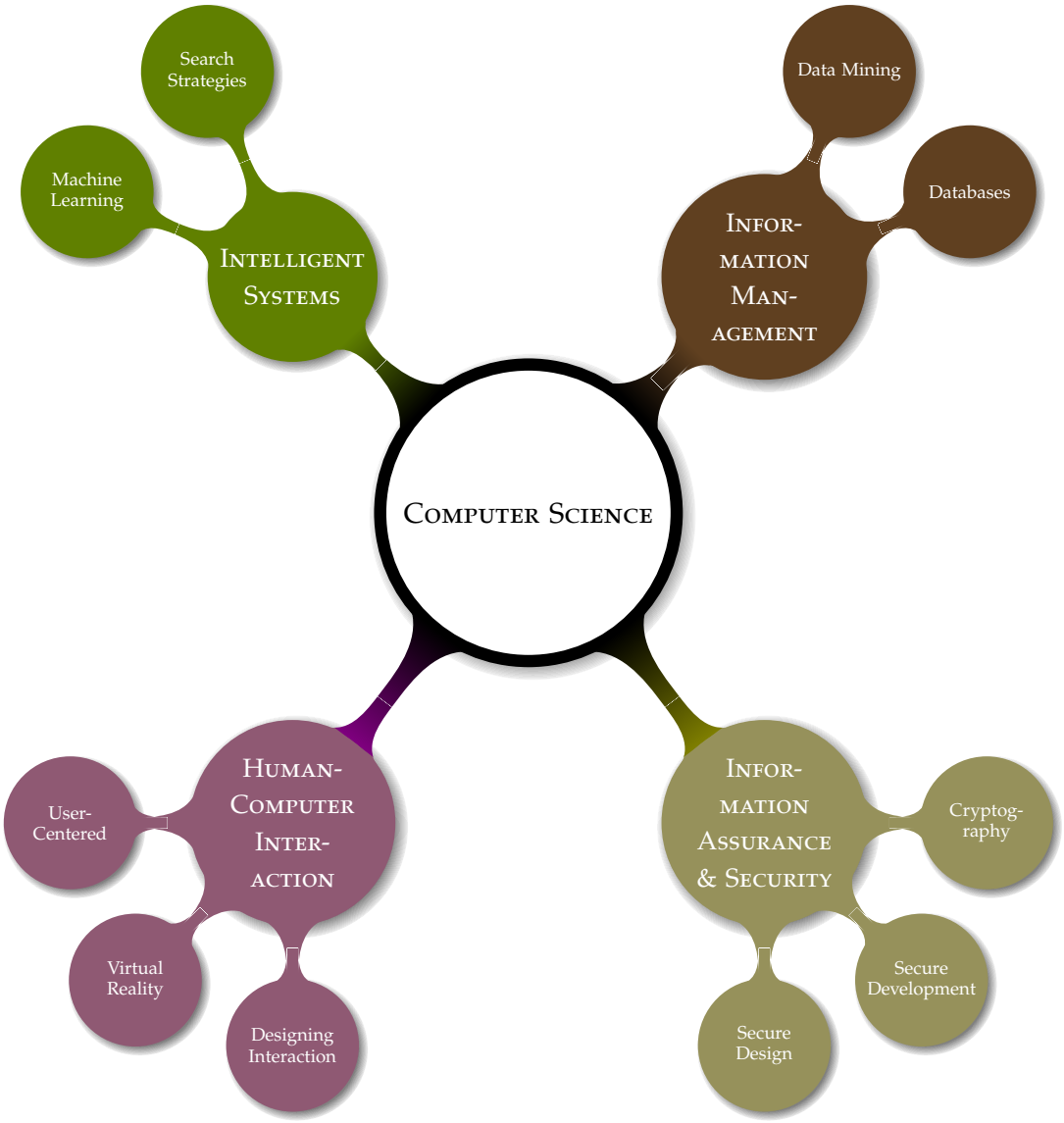


Figure 5: Knowledge Areas of Computer Science

Figure 6: Knowledge Areas of Computer Science

Figure 7: Knowledge Areas of Computer Science

Figure 8: Knowledge Areas of Software Engineering

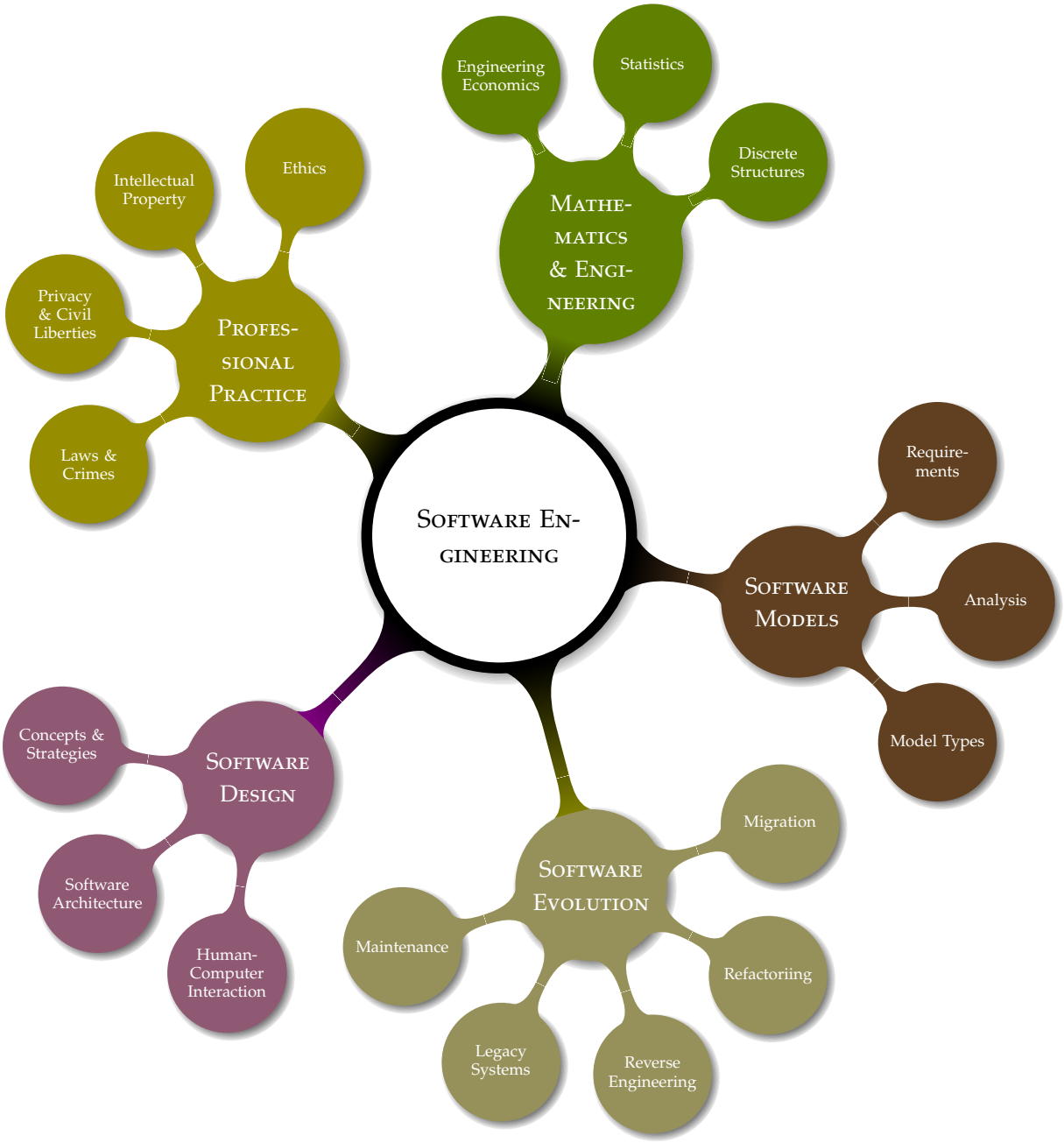


Figure 9: Knowledge Areas of Software Engineering

2. Chemistry — take the following courses
   (a) CHM 1101 General Chemistry 1 (4 credits)
   (b) CHM 1102 General Chemistry 2 (4 credits)
   (c) CHM 2001 Organic Chemistry 1 (3 credits)
   (d) CHM 2002 Organic Chemistry 2 (3 credits)
   (e) 6 additional credits in chemistry
3. Computational Mathematics — take the following courses
   (a) MTH 2201 Differential Equations/Linear Algebra (4 credits)
   (b) Two of the following
      i. MTH 4082 Introduction to Parallel Processing (3 credits)
      ii. MTH 4311 Numerical Analysis (3 credits)
      iii. MTH 4320 Neural Networks (3 credits)
   (c) Three math electives numbered 3XXX or higher
4. Physics — take the following courses
   (a) PHY 1001 Physics 1 (4 credits)
   (b) PHY 2002 Physics 2 (4 credits)
   (c) PHY 2091 Physics Lab 1 (1 credit)
   (d) PHY 2092 Physics Lab 2 (1 credit)
   (e) 9–11 additional credits in physics

I believe you can earn a Prelaw minor with just one extra course. Additional minors are offered. A list of minors and their requirements are in the University catalog. If you are interested in one of these minors, talk with your advisor, and others, about your interests. Map its requirements to your program plan.

Dual Majors and Additional Degrees

An Applied Mathematics major can be earned with little extra course work. A case can be be made for a dual major in computer science and mathematics.[11]

A case can be made for and against a dual major in computer science and software engineering. Talk to people to gain different views.

Check the catalog for the distinction between dual majors and additional degrees, the last requiring at least 15 additional credits.

[11] See What's It Worth: The Economic Value of College Majors

10. Computing History, October 22

I think there is a world market for maybe five computers

Thomas Watson, chairman of IBM. (1943)

The need for and impact of computing has often been underestimated. Because history is the driest subject the Mouse [12] and I know, I'd like ot repeat some history for you, and ask that you learn more about computing history. I've selected five people, five technologies, and five algorithms to report on.

[12] A Caucus-Race And a Long Tale

Homework

[back to page 7]

1. Visit the Computer History Museum and its YouTube channel Other good sites about computing history include: the Wikipedia article on the history of computing hardware, the Rutger's timeline, and the HiTMilL History of Computers, or read [13]. Other sites
2. Identify two additional computer scientists/software engineers and write about them.
3. Identify two additional technologies and write about them.
4. Identify two additional algorithms and write about them.
5. Complete this assignment by November 18.
6. Submit your report using the course management system

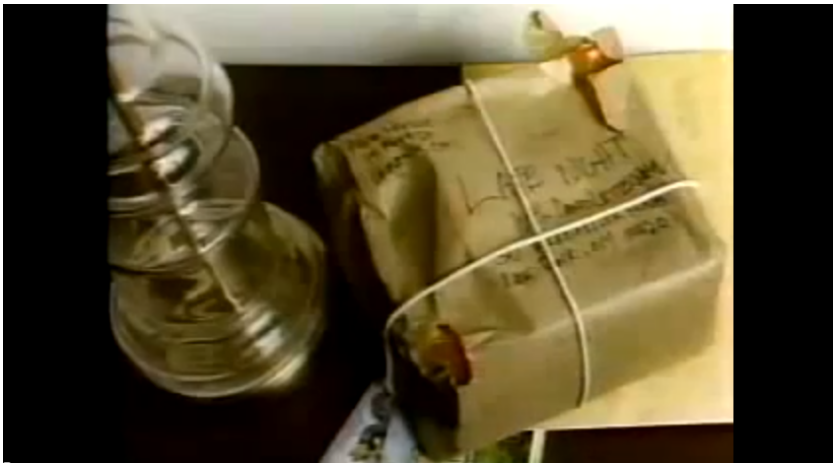
[13] Glick, J. (2011). *The Information: A History, A Theory, A Flood*. Pantheon Press, New York

People

There are many people who have contributed to the fields of computer science and software engineering. I've picked five to showcase.

1. Grace Hopper (1906–1992) Credited with inventing the first compiler, the COBOL programming language, and coining the word "debug."

   Watch her 1990 interview with David Letterman (9:59).

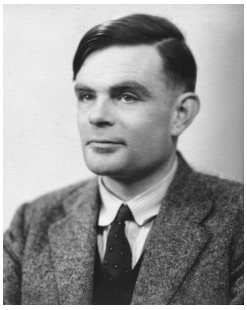


Tim Berners-Lee

Edsger Dijkstra

Don Knuth

John von Neumann



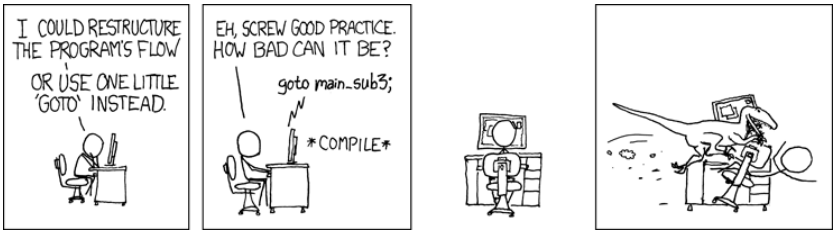
Alan Turing

2. Tim Berners-Lee (1955–)

   Proposed and implemented the Hypertext Transmission Protocol (http) in 1989 leading to the development of the World Wide Wet on the Internet.

3. Edsger Dijkstra (1930–2002)

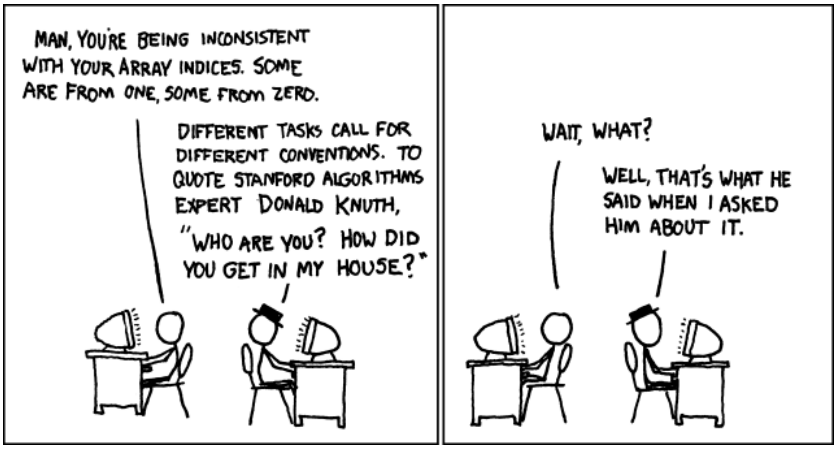
   For fundamental contributions to programming as a high, intellectual challenge; for eloquent insistence and practical demonstration that programs should be composed correctly, not just debugged into correctness; for illuminating perception of problems at the foundations of program design. Cited from the nomination for his Turing award



GOTO

4. Don Knuth (1932–)

   For his major contributions to the analysis of algorithms and the design of programming languages, and in particular for his contributions to the "art of computer programming" through his well-known books in a continuous series by this title. Cited from the nomination for his Turing award



Donald Knuth

5. John von Neumann (1903–1957)

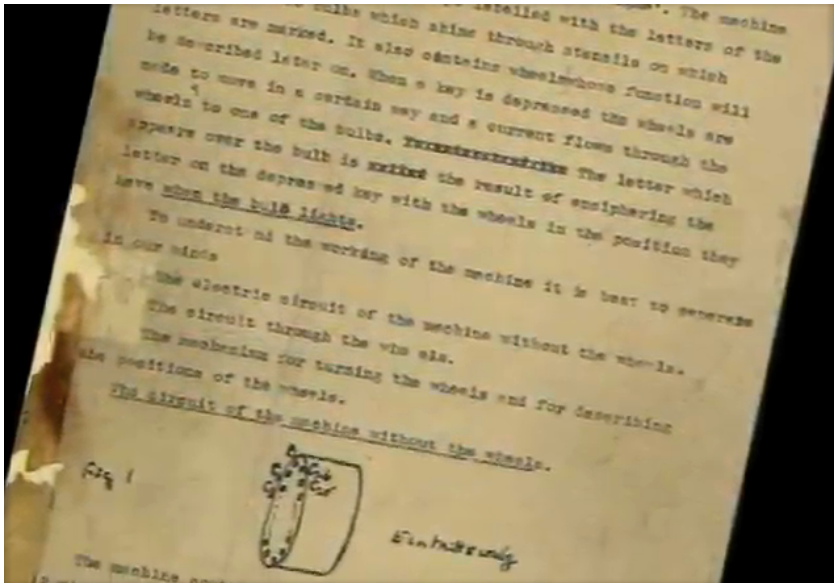
   Involved in some many fields throughout the twentieth century, his work in difficult to summarize. In the field of computing, he described a computer architecture that has dominated computers from the 1950's onward. A key architectual element is to store both the data and program in a common memory address space.

6. Alan Turing (1912–1954)

   Founder of computer science, mathematician, philosopher, code-breaker, strange visionary and a gay man before his time. Cited from The Alan Turing Home Page. His mathematical description of a computing machine is a model for physical computers, and defines the difference between computable and non-computable problems.

   Here are two nice videos about Alan Turing and the award named for him.

   (a) The Origins of the A. M. Turing Award (short 1:52)



(b) ACM A.M. Turing Centenary Celebration (long 1:18:29)

There is also the film about Turing: The Imitation Game, which is set to debut in fall 2014. Here's a tralier.

Technologies

1. Video: The Antikythera Mechanism (52:52). A 2000 year-old mechanical Grecian device rediscovered around 1900, and recently deciphered.

It accuratley predicts/computes the orbits of planets and the moon. Some call it the world's first computer. Learn how the gears with a prime number of teeth (19, 53, 127, and 223) accuratelty predict the location of celestial objects. This is a very interesting historical account. I think it is worth your time to watch the video. Follwing this link for more information on the Antikythera mechanism

2. Video: The Writing Boy Automaton A 240 year old doll that can write, a clockwork creation by Pierre Jaquet-Droz

3. The IAS (Institute of Advanced Studies) Machine (1945–1951)

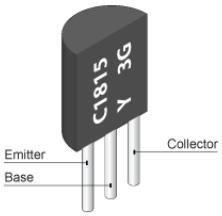
   Built at Princeton University under the direction of John von Neumann and Julian Bigelow. The IAS machine was a binary computer with a 40 bit word, storing two 20 bit instructions in each word. The memory was 1024 words (5.1 kilobytes). Negative numbers were represented in "two's complement" format. It had two general-purpose registers: the Accumulator (AC) and Multiplier/Quotient (MQ).

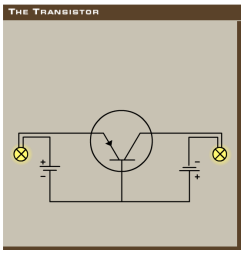
von Neumann, Bigelow, Goldstine, and Oppenheimer before the IAS computer

4. The Transitor (1947)

   The Nobel Prize in Physics 1956 was awarded jointly to William Bradford Shockley, John Bardeen and Walter Houser Brattain "for their researches on semiconductors and their discovery of the transistor effect". The transitor is the key device in almost all modern electronics.

   A transistor is a *semiconductor* device that can amplify or switch currents and power. It is a semiconductor (e.g., silicon) with three terminals for connection to an external circuit. A voltage or current applied to one pair of the transistor's terminals changes the current through another pair of terminals. Because the controlled (output) power can be higher than the controlling (input) power, a transistor can amplify a signal.





Study the Nobel Prize educational description of the transitor.

5. The IBM 360 (1964–1978)

   The IBM System/360 was a mainframe computer. Both the computer and its operating system strongly influenced a computing generation. The chief architect of System/360 was Gene Amdahl, and the project was managed by Fred Brooks. Some features:

   - the use of microcode to implement the instruction set
   - 8-bit (byte) addressing and binary, decimal and floating-point calculations
   - Big-endian byte ordering

I first learned to program in FORTRAN on an System/360 computer.

- EBCDIC character set
- 16 32-bit (4-byte) general purpose registers (R0-R15)
- A 64-bit Program status word (PSW) which describes: Interrupt masks, privilege states, a condition code
- A 24-bit instruction address

6. The MacIntosh

   Read The MacIntosh Way by Guy Kawasaki.

Algorithms

Algorithms are simple ideas that solve simple problems. The best algorithms are those that you can integrate with other algorithms to solve interesting and important problems. Here's a few that are considered important.

1. Euclidean algorithm

   Algorithm to determine the greatest common divisor (gcd) of two integers. It among the oldest known algorithms. Euclid recorded it in his Book of Elements around 300 BC.

```
function gcd(a, b)
while b != 0 {
  t := b
  b := a mod t
  a := t
}
return a
```

2. Fast Fourier transform (FFT)

The discrete Fourier transform of the n points signal x_j, $j = 0, 1, \ldots, n-1$ is

$$f_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j (e^{2\pi ijk/n}), \quad k = 0, 1, \ldots, n-1.$$

FFTs are of great importance to a wide variety of applications, from digital signal processing to solving partial differential equations to algorithms for quickly multiplying large integers. The Cooley–Tukey algorithm (1965), named after J.W. Cooley and John Tukey, is the most common FFT. The code below is from Arndt (2011).

```
procedure fft(a[], ldn, is) {
  n := 2**ldn; // length of a[] is a power of 2

  revbin_permute(a[], n);

  for ldm:=1 to ldn   // lnd = log base 2 of n
  {
    m := 2**ldm;
    mh := m/2;

    for r:=0 to n-m step m // n/m iterations
    {
      for j:=0 to mh-1 // m/2 iterations
      {
        e := exp(is*2*PI*I*j/m); // nlog_2(n)/2 computations

        u := a[r+j];
        v := a[r+j+mh] * e;

        a[r+j] := u + v;
        a[r+j+mh] := u - v;
      }
    }

  }
}
```

3. The RSA (Rivest-Shamir-Adelman) algorithm

A public-key cryptography algorithm which uses prime factorization as the trapdoor one-way function. Define

$$n = pq$$

for p and q primes. Also define a private key d and a public key e such that

$$de \equiv 1 \pmod{\phi(n)} \qquad \gcd(e, \phi(n)) = 1$$

where $\phi(n)$ is the Euler totient function. Let the message be converted to a number M. The sender then makes n and e public and sends

$$E = M^e \pmod{n}$$

To decode, the receiver (who knows d) computes

$$E^d = (M^e)^d = M^{ed} = M^{N\phi(n)+1} = M \pmod{n}$$

since N is an integer. In order to crack the code, d must be found. But this requires factorization of n since

$$\phi(n) = (p-1)(q-1).$$

4. Binary Search

Given a sorted list, find a key in the list.

```
int binary_search(int A[], int key, int imin, int imax) {
  while (imax >= imin)
  {
    int imid = midpoint(imin, imax);
    if (A[imid] < key) imin = imid + 1;
    else if (A[imid] > key) imax = imid - 1;
    else return imid;
  }
  return KEY_NOT_FOUND;
}
```

5. Quicksort

Published by Tony Hoare in 1962, Quicksort puts n things in order (numerical, alphabetical, ...). Hoare's algorithm uses the age-old recursive strategy of divide and conquer to solve the problem: Pick one element as a "pivot," separate the rest into piles of "big" and "small" elements (as compared with the pivot), and then repeat this procedure on each pile. Quicksort runs, on average, with $O(n \log n)$ time complexity. Here is Jon Bentley's way to write Quicksort.

```
void quicksort(int l, int u) {
  int i, m;
  if (l >= u) return;
  m = l;
  for (i = l+1; i<= u; i++)
  if (x[i] < x[l]) {
    swap(++m, i);
  }
  swap(l, m);
  quicksort(l, m-1);
  quicksort(m+1, u);
}
```

11. Computing Skills, Concepts, and Capabilities, October 29

> Computer programming is an art, because it applies accumulated knowledge to the world, because it requires skill and ingenuity, and especially because it produces objects of beauty. A programmer who subconsciously views himself as an artist will enjoy what he does and will do it better.
>
> Don Knuth, 1974 Turing Award Lecture

These notes comes from (Snyder, 2007).

Information Technology Skills

This material is not typically taught in a computer science or software engineering program. For the most part, students in these computer science and software engineering are expected to learn these skills on their own.

1. Setting up a personal computer
2. Using basic operating system features
3. Using a word processor to create a text document
4. Using a graphic and/or artwork package to create illustrations, slides, or other image-based expressions of ideas
5. Connecting a computer to a network
6. Using the Internet to find information and resources[14]
7. Using a computer to communicate with others
8. Using a spreadsheet to model simple processes or financial tables
9. Using a database system to set up and access useful information[15]
10. Using instructional materials to learn how to use new applications or features

[14] COM 2012

[15] CSE 4020

Information Technology Concepts

These topics are the basis of computer science and software engineering. The concepts are taught throughout the curriculum.

1. Fundamentals of Computers
2. Organization of Information systems
3. Fundamentals of Networks
4. Digital representation of information
5. Structuring information
6. Modeling and abstraction
7. Algorithmic thinking and programming
8. Universality
9. Limitations of information technology
10. Societal impact of information and information technology

Intellectual Capabilities

These abilities are required of computer scientists and software engineers. Students in these majors will be challenged to exercise these capabilities.

1. Engage in sustained reasoning
2. Manage complexity
3. Test a solution
4. Manage problems in faulty solutions
5. Organize and navigate information structures and evaluate information
6. Collaborate
7. Communicate to other audiences
8. Expect the unexpected
9. Anticipate changing technologies
10. Think about information technology abstractly

Student Outcomes

The Florida Tech's Computer Science program is accredited by the Computing Accreditation Commission of ABET, http://www.abet.org. The Florida Tech's Software Engineering program is accredited by the Engineernig Accreditation Commission of ABET, http://www.abet.org. To support ABET criteria, we have adopted several student outcomes that our programs allow our students to attain.

1. An ability to apply knowledge of mathematics, science, computing, and software engineering
2. An ability to identify computing and engineering problems, identify and define the requirements, design and conduct experiments, analyze and interpret data appropriate to solving these problems
3. An ability to use the techniques, skills, and modern tools necessary for computing and software engineering practice
4. An ability to apply mathematical and scientific foundations, algorithmic and data structuring principles, software design, testing, construction, and maintenance principles, and computer science theory to model, design and evaluate computer-based systems and processes in ways that demonstrates
   (a) Achievement of skills necessary to construct complex software systems
   (b) Satisfaction of requirements within realistic constraints such as economic, environmental, social, political, ethical, health, safety, security, and sustainability
   (c) Comprehension of the trade-offs involved in design choices
5. An understanding of professional, ethical, legal, security and social issues and responsibilities
6. The broad education necessary to analyze the impact of computing and software engineering solutions in a local, global, economic, environmental, and societal context
7. An ability to communicate effectively with a range of audiences
8. An ability to function effectively on multidisciplinary teams to accomplish a common goal
9. A recognition of the need for, and an ability to engage in, continuing professional development
10. A knowledge of historical and contemporary issues

Reading

Read Principles of Traditional Animation Applied to 3D Computer Animation, by John Lasseter. Be prepared to discuss:

1. What are the fundamental principles of traditional animation?

12. Creative Coding, November 5

Science is what we understand well enough to explain to a computer. Art is everything else we do.

Don Knuth Foreword to the book A=B (1996)

The best programs are written so that computing machines can perform them quickly and so that human beings can understand them clearly. A programmer is ideally an essayist who works with traditional aesthetic and literary forms as well as mathematical concepts, to communicate the way that an algorithm works and to convince a reader that the results will be correct.

Don Knuth Selected Papers on Computer Science

Action Items

- Join Computer Science at the Khan Academy
- Listen to the first Khan Academy lecture on computer science

Software Development Skills

To develop interesting computer applications requires skills:

1. A comprehension of what is to be done (*requirements*)
2. An ability to solve a problem and to explain the solution (*design*)
3. The capability to write in a programming language (*implementation*)
4. The ability to confirm that a program behaves as required (*testing*)
5. The ability to discover where a program fails when it fails (*debugging*)
6. Helping others to use the program (*deployment*)

7. The ability to add new functionality or fix post-deployment errors (*maintenance*)

Software Development Tools

There are various tools to master to become a skilled hacker. There are *personal* tools and *collaborative* tools.

Some personal tools you need to be able to use are:

- *Operating systems*: The software that provides an interface between applications and the computer's hardware resources. Unix, Windows, iOS, Android, and many more. At this point, your skill should be the ability to *use* an operating system. A student majoring in computing should be reasonably comfortable using both Windows and a flavor of Unix.

  Later in our computing curriculum, you'll be interested in operating systems issues: CPU scheduling, memory management, I/O control, etc.

- *Integrated Development Environments (IDEs)*: A tool chest that provides a toolchest that a computer programmer can use to develop software: Eclipse, Visual Studio, Xcode, Emacs, and many more. The tools include

  - An editor – to write and change code
  - A compiler – to build the executable
  - A debugger – to locate and fix bugs
  - A profiler – to analyze code efficiency

- A document preparation system – Software to prepare documents of assorted kinds: Word, LaTeX 2_ε, Pages, and many more.

Most software is developed by a team. Here are some high-level collaborative tools.

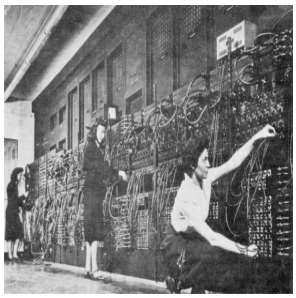
- *Project hosting*: A source code repository, a centralized location for software developers to control and manage code and associated documents.

  - Your companies servers
  - GitHub SourceForge is a web-based

- *Version control*: to manage changes to documents, programs and other information

  - Git

  - Subversion
  - Visual SourceSafe
- *Issue tracking*: Provide feedback during development; manage and maintain lists of issues.
  - Bugzilla
  - Trac
- *Database management system*: Define, create, query, update, and administration data and relations among them.
  - mySQL
  - Oracle
  - SAP


Computer programmers hard-wire the ENIAC

The Evolution of Computer Programming

1. Early computers were programmed by "hard-wiring" the algorithm that controlled the machine.
2. Later computers were programmed by writing instructions in binary or hexadecimal codes (*machine code*). For example on the Intel 8080 processor, the instruction to
   - ADD the contents of register *A* to register *D* is 87 in hexadecimal or 1000 0111 in binary
   - MOVE contents of register *A* to register *E* is 7*B* in hexadecimal or 0111 1011 in binary
3. To make life easier for programmers, machine code was soon replaced by mneumonic *assembly codes*
   - ADD A, D
   - MOVE A, E
4. The development of compilers has allowed programmers to write at even higher levels of abstraction.

Coding

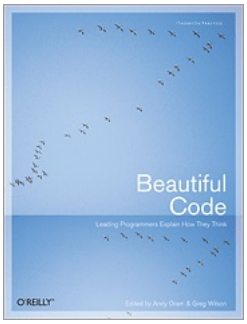
To write computer code you must be familiar with the components of a programming language.

- Types, Operators, and. Expressions
  - Primitive types: Character, String, Boolean, Integer, Floating Point

  - Composite types: Array, Object, Function
- Control Flow
  - If-Then-Else
  - While and For Loops
- Functions
- Input and Output
  - Standard Input and Output
  - File Input and Output
- Reserved words
  - **Abstraction:** function, interface
  - **Control flow:** break, case, for, if-then-else, return, while
  - **Data Types:** boolean, char, constant, double, int
  - **Input/Output:** read, write, open, close
  - **Memory management:** new, delete
  - **Values:** `False`, `True`, null

Abstraction

The code below is by Rob Pike and described by Brian Kernighan in Beautiful Code (Oram and Wilson, 2007). Pike's code implements a *regular expression* matcher and illustrates basic programming principles.



Regular expressions were invented by Stephen Kleene in the mid 1950s as a notation for finite automata. An example of regular expressions is

```
[-+]?([0-9]*\.)?[0-9]+
```

which defines a *rational decimal number*.

- The leading part `[-+]?` states that a rational decimal number can start with an (optional) − or + sign.
- The next part `([0-9]*\.)?` states that the number can (optionally) have zero or more digits followed by a decimal point.
- The last part `[0-9]+` states that the number must have one or more digits.

```
/* match: search for regexp anywhere in text */
    int match(char *regexp, char *text)
    {
        if (regexp[0] == '^')
            return matchhere(regexp+1, text);
        do {    /* must look even if string is empty */
            if (matchhere(regexp, text))
                return 1;
        } while (*text++ != '\0');
        return 0;
    }

    /* matchhere: search for regexp at beginning of text */
    int matchhere(char *regexp, char *text)
    {
        if (regexp[0] == '\0')
            return 1;
        if (regexp[1] == '*')
            return matchstar(regexp[0], regexp+2, text);
        if (regexp[0] == '$' && regexp[1] == '\0')
            return *text == '\0';
        if (*text!='\0' && (regexp[0]=='.' || regexp[0]==*text))
            return matchhere(regexp+1, text+1);
        return 0;
    }

    /* matchstar: search for c*regexp at beginning of text */
    int matchstar(int c, char *regexp, char *text)
    {
        do {    /* a * matches zero or more instances */
            if (matchhere(regexp, text))
                return 1;
        } while (*text != '\0' && (*text++ == c || c == '.'));
        return 0;
    }
```

Games

Many people are interested in games. Python is one, among many, computer programming languages that can be used to write computer games. Pygame is a site you can used to learn how to build games. Steam is another site useful for game developers.

Mobile Apps

There are many APIs for writing mobile applications. Consider learning to write applications using

1. Google Mirror API for Google Glass.
2. Android SDK for Android applications.
3. One of the many Amazon Mobile App SDKs.

And there are many others.

Pygame Cheat Sheet
http://inventwithpython.com

```
# Download files from:
#     http://inventwithpython.com/cat.png
#     http://inventwithpython.com/bounce.wav

import pygame, sys
from pygame.locals import *

pygame.init()
fpsClock = pygame.time.Clock()

windowSurfaceObj = pygame.display.set_mode((640, 480))
pygame.display.set_caption('Pygame Cheat Sheet')

catSurfaceObj = pygame.image.load('cat.png')
redColor = pygame.Color(255, 0, 0)
greenColor = pygame.Color(0, 255, 0)
blueColor = pygame.Color(0, 0, 255)
whiteColor = pygame.Color(255, 255, 255)
mousex, mousey = 0, 0

fontObj = pygame.font.Font('freesansbold.ttf', 32)
msg = 'Hello world!'

soundObj = pygame.mixer.Sound('bounce.wav')

while True:
    windowSurfaceObj.fill(whiteColor)

    pygame.draw.polygon(windowSurfaceObj, greenColor, ((146, 0), (291, 106), (236, 277), (56, 277), (0, 106)))
    pygame.draw.circle(windowSurfaceObj, blueColor, (300, 50), 20, 0)
    pygame.draw.ellipse(windowSurfaceObj, redColor, (300, 250, 40, 80), 1)
    pygame.draw.rect(windowSurfaceObj, redColor, (10, 10, 50, 100))
    pygame.draw.line(windowSurfaceObj, blueColor, (60, 160), (120, 60), 4)

    pixArr = pygame.PixelArray(windowSurfaceObj)
    for x in range(100, 200, 4):
        for y in range(100, 200, 4):
            pixArr[x][y] = redColor
    del pixArr


    windowSurfaceObj.blit(catSurfaceObj, (mousex, mousey))

    msgSurfaceObj = fontObj.render(msg, False, blueColor)
    msgRectobj = msgSurfaceObj.get_rect()
    msgRectobj.topleft = (10, 20)
    windowSurfaceObj.blit(msgSurfaceObj, msgRectobj)

    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
        elif event.type == MOUSEMOTION:
            mousex, mousey = event.pos
        elif event.type == MOUSEBUTTONUP:
            mousex, mousey = event.pos
            soundObj.play()
            if event.button in (1, 2, 3):
                msg = 'left, middle, or right mouse click'
            elif event.button in (4, 5):
                msg = 'mouse scrolled up or down'

        elif event.type == KEYDOWN:
            if event.key in (K_LEFT, K_RIGHT, K_UP, K_DOWN):
                msg = 'Arrow key pressed.'
            if event.key == K_a:
                msg = '"A" key pressed'
            if event.key == K_ESCAPE:
                pygame.event.post(pygame.event.Event(QUIT))

    pygame.display.update()
    fpsClock.tick(30)
```

init() needs to be called before any other pygame code.

pygame.locals has the constants like QUIT, MOUSEMOTION, and K_ESCAPE.

The Clock object makes sure our program runs (at most) at a certain FPS.

set_mode() creates the window. Param is (width, height) in pixels. The Surface object returned is drawn to the screen when pygame.display.update() is called.

load() will return a Surface object with the image drawn on it. (Can be png, gif, bmp, jpg, and others.)

Params to the Color objects are for Red, Green, Blue. 0 is none, 255 is max.

Create a Font object with a font of size 32 pts.

Draw functions draw shapes to a specific Surface object with a specific color (which can be a Color object or (R, G, B) tuple of integers.)

Create a Sound object from a file. (Can be wav, ogg, or mp3.)

This loop is the main application loop. The fill() method will completely fill the Surface object with white, then we redraw the window.

Coordinates are either (x, y) or (top, left, width, height).

To draw individual pixels on a Surface, create a PixelArray() object. After making changes, you must delete the PixelArray object (but the Surface will still have the changes.)

The last parameter is the width. 0 means "filled in" and no param defaults to a width of 1 pixel.

blit() method draws one Surface to another Surface, specifying the (left, top) coordinates.

render() creates a Surface object with the text drawn on it in the specified font and color. You can blit() this Surface object to the window's Surface object.

Rect objects have attributes for specifying their position and size (i.e. topleft, top, left, centerx, bottom, width, etc.)

pygame.event.get() returns a list of all Event objects that happened since the last time get() was called.

quit() is the opposite of init()

The Event object has type, pos, key and other attributes depending on the type of event it is.

Call play() method to play the sound. The code continues on while it plays.

The K_* constants in pygame.locals represent the different keyboard keys.

The window is not drawn to the actual screen until pygame.display.update() is called.

Wait long enough to run at 30 frames per second. (Call this *after* pygame.display.update().)

http://pygame.org/docs has more documentation.

An extended tutorial is in chapters 16 to 20 of the free book, "Invent Your Own Computer Games with Python" at http://inventwithpython.com

13. The Theory of Computing, November 12

If you find that you're spending almost all your time on theory, start turning some attention to practical things; it will improve your theories. If you find that you're spending almost all your time on practice, start turning some attention to theoretical things; it will improve your practice.

Don Knuth

What is Computable/Decidable?

Humans have been computing since the beginnings of recorded history, which most likely implies we started computing before history was recorded. But, it wasn't until the twentieth century that we developed a theory of computing.

Arithmetic is a basic form of computation. By arithmetic, I mean addition and multiplication on the integers, and the extensions that can be derived from basic axioms about these operations. In the early 1900's David Hilbert asked that mathematicians prove that the axioms of arithmetic are independent and non-contradictatory.

Babylonian computations are recorded on five thousand year-old clay tablets.

> "Whether, in any way, certain statements of single axioms depend upon one another, and whether the axioms may not therefore contain certain parts in common, which must be isolated if one wishes to arrive at a system of axioms that shall be altogether independent of one another."
>
> "To prove that they are not contradictory, that is, that a definite number of logical steps based upon them can never lead to contradictory results." Hilbert (1902)

Decision Problems

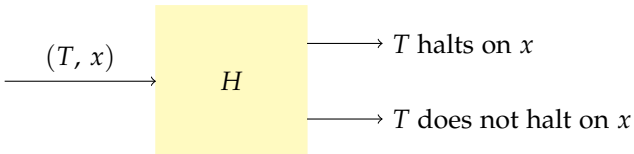
Later, in 1928, Hilbert proposed the Entscheidungsproblem (the decision problem). Find an algorithm A that takes a statement s in first-order predicate logic and answers "yes" or "no" dependent on whether or not s is `True` or `False`.

A *decision* problems is a question with a "yes" or "no" answer. A decision problem is *decidable* if there is a Turing machine that correct answers it and stops.

There are many decision problems that are undecidable.

1. The Halting Problem: Given a Turing machine T and input x for it, does T halt on x?

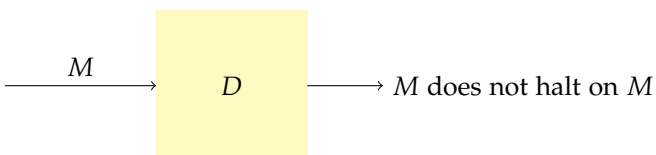
   - Pretend there is a Turing machine H that decides the halting problem.



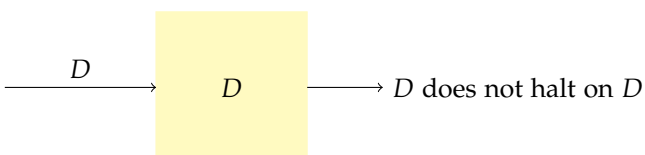
   - Use H to construct a Turing machine D that accepts the encoding of a Turing machine M as input and runs (M, M) on H

     (a) D does not halt if M halts on M

     (b) D halts if M does not halt on M



   - Consider $D(D)$



     (a) if D halts on D, then D does not halt on D

     (b) if D does not halt on D, then D halts on D

     This contradiction implies that the halting machine H cannot exist.

2. The Virus Problem: Given a program p, does p have a virus?

   Suppose there is a virus checking Turing machine V. If V is presented with a program, p, it returns `True` or `False`, depending on whether p is a virus or not.

   Now write a program c like this:

```
program c {
  if(V(c)) { do_nothing(); }
  else { run_virus(); }
}
```

(a) If $V(c)$ returns `True`, that is c is a virus, then c does nothing, and therefore V is wrong.

(b) Similarly, if V returns `False`, that is c is not a virus, then c runs a virus, and again V is wrong.

This construction is attributed to Fred Cohen in his 1986 PhD dissertation. It is an example of a *diagonalization* argument.

Complexity

Once an algorithm A that solves a decision or computational problem has been found, one wants to know its *complexity*.

- Time complexity: How many basic steps does it take?
- Space complexity: How much storage does it need?
- Communication complexity: How much bandwidth does it use?

Time complexity is the most basic measure, and most algorithms have a time complexity that can be described by a common function. Let n be the size of the input to algorithm A, let and $T(n)$ be the maximum number of basic steps A takes in solving an instance of the problem of size n.

- $T(n) = 1$ — constant time
- $T(n) = \lg n$ — logarithmic time (very fast)
- $T(n) = n$ — linear time (fast)
- $T(n) = n \lg n$ — log-linear time (still fast)
- $T(n) = n^2$ — quadratic time (starting to slow down)
- $T(n) = 2^n$ — exponential time (too slow)

Determinism and Nondeterminism

The control of a Turing machine can be based on a function or a relation.

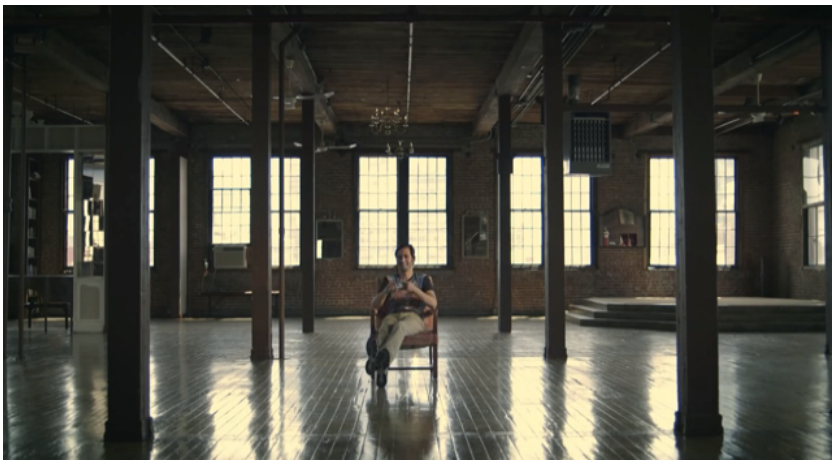
- Function: In a given state, scanning a given input there is only one move (next state, change on tape, motion of read/write head)
- Relation: In a given state, scanning a given input there can be several moves (next states, changes on tape, motions of read/write head)

Electronic computers model deterministic Turing machines. In 1980 and 1981, the mathematician Yuri Manin and the physicist Richard Feynmann independently proposed a model of computation using

quantum mechanics. A quantum computer behaves nondeterministically.

Nondeterminisism does not broaden the class of problems that can be computed. But, importantly, nondeterminisism allows answers to be computed in fewer steps than deterministic computations.

Watch the quantum computing video (6:29).

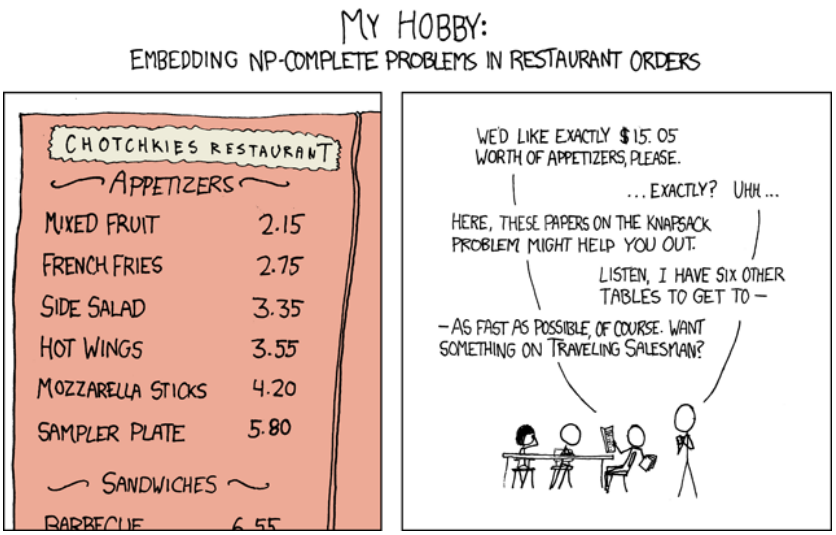


The P versus NP Question

The class P is the collection of decision problems that can be solved in a polynomial number of steps on a deterministic Turing machine. These problem are called *tractable*, meaning they be *solved quickly*.

The class NP is the collection of decision problems that can be solved in a polynomial number of steps on a nondeterministic Turing machine. One characterization of NP problems is they can be *checked quickly*: Given the answer, it can be shown to be correct in polynomial time, but it may be very hard to find the answer.

The most famous problem in theoretical computer science is *Does P equal NP*.



NP-Complete

Reading

Read Gödel's Undecidability Theorem, by Stephan Andrilli. Be prepared to discuss:

- What makes mathematical system consistent? When is a mathematical system inconsistent?
- What makes a mathematical system complete? When is a mathematical system incomplete?
- What are the implications of Gödel's result when applied to algorithms and problems?

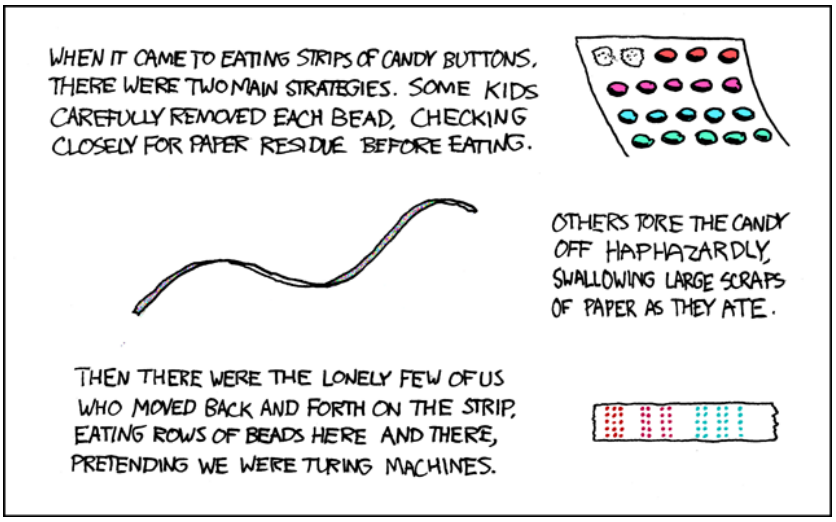
Read Leah Hoffmann's interview with Shafi Goldwasser & Silvio Micali. The 2013 Turing Award winners talk about proofs, probability, and poker. Be prepared to discuss:

- What is a zero knowledge interactive proof?

Video: A Turing Machine

Much of the theory of computer science rests on the idea that a Turing machine can compute the answer to any decidable problem. If a problem is undecidable, then no "machine" can solve it.

Turing machines are surprisingly simple. Watch the Turing Machine video to see how simple (5:09).



Candy Button Paper

The Continuum Hypothesis

Material for these notes comes from Mahticon

Georg Cantor (1845âĂŞ1918) described different sizes of infinity ∞. He called these different sizes cardinalities (the number of things):

- 0 none
- 1 one
- 2 two
- etc
- $\aleph_0$ aleph naught:

  $\aleph_0$ is the number of natural numbers $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$.

- $\aleph_1$ aleph one

  $\aleph_1$ is the number of subsets of $\mathbb{N}$

$$\aleph_1 = 2^{\aleph_0}$$

Cantor's work is fundamental to modern mathematical logic, regardless of his purported interest in understanding god as some absolute infinity.

The natural numbers, integer, and rational numbers: $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{Q}$, all have the same cardinality, $\aleph_0$ (aleph naught). While, the real numbers $\mathbb{R}$ have a bigger cardinality: $\aleph_1$.

The cardinal numbers suffer a strange arithmetic, for example

- $\aleph_0 = \aleph_0 + 1$. Accommodate one more guest in an infinite hotel by asking each guest to move down one room.
- $\aleph_0 + \aleph_0 + \aleph_0$. Accommodate infinitely many new guests by asking each guest to move down two times their room number, making way for twice as many guests.
- $\aleph_0 + \aleph_0 \times \aleph_0$. Accommodate infinitely many of infinitely many new guests by asking the guest in room j to move to room $2^j 3^j$.

Cantor believed $\aleph^1 > \aleph_1$, but could not prove it. This is called the *Continuum Hypothesis*.

Gödel proved (1940) the continuum hypothesis cannot be disproved using standard set theory. Paul Cohen proved (1963) the continuum hypothesis cannot be proven from those same axioms. The continuum hypothesis can be `True` or `False`, and it is impossible to prove it either way. This means that you can decide for yourself whether you want it to be `True` or not. The continuum hypothesis is one of the unprovable statements which were discovered by Gödel. Most mathematicians believe it to be `True` without proof.

The Zermelo–Fraenkel axioms are the standard for sets.

14. To be determined, November 19

Reading

Read The Making of an Expert, by Ericsson, Prietula, and Cokely Be prepared to discuss:

- What are the three tests they say must be passed to determine if someone is a genuine expert?
- What advice do they give that can help you become an expert?

15. To be determined — two, November 26

May the good things of life be yours in abundance.

Enjoy the movie.

16. The Last Lecture, December 3

If I only had three words of advice, they would be, 'Tell the truth'. If got three more words, I'd add, 'All the time'.

Randy Pausch

Action Items

1. Prepare for your final examination which is on December 10 from 1–3 p.m. in Olin Engineering 118. Review the course topics and your homework.

Randy Pausch's Last Lecture

Randy Pausch was a professor of Computer Science at Carnegie Mellon University (CMU). He died in 2008 from pancreatic cancer. A year before, he gave a talk titled The Last Lecture: Really Achieving Your Childhood Dreams. Watch Randy's last lecture (1:18:27).

Many have found it inspiring. May it inspire you too.

17. Final Examination, December 10

Humanity is in "final exam" as to whether or not it qualifies for continuance in (the) Universe. . . (I) am convinced that human continuance depends entirely upon the intuitive wisdom of each and every individual. . . the individual's integrity . . .

R. Buckminster Fuller, from (Fuller, 1981)

Action Items

1. Study for the final examination on Olin Engineering 118 between 1–3 p.m. on December 10.
2. Students who fail to take the final will receive a grade of F.

What is Expected of You on the Final

You must be able to:

1. Be able to describe computing activities you have attended at Florida Tech
2. Compare and contrast computer science and software engineering
3. Write about computing history: people, technologies, algorithms
4. Describe the electives you'll choose to complete your academic program plan
5. Describe your understanding of the need for computing professionalism
6. Write about articles you have read and reported upon

Bibliography

Abelson, H., Ledeen, K., and Lewis, H. (2008). *Blown to Bits: Your Life, Liberty, and Happiness after the Digital Age*. BitBooks.com. [page 46]

Adleman, L. M. (1994). Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024. [page 45]

Arndt, J. (2011). *Matters Computational: Ideas, Algorithms, Source Code*. Springer, 1st edition. [page 84]

Bacon, D. and van Dam, W. (2010). Recent progress in quantum algorithms. *Commun. ACM*, 53(2):84–93. [page 46]

Bentley, J. L. (1982). *Writing Efficient Programs*. Prentice-Hall. [page 44]

Bentley, J. L. (1986). *Programming Pearls*. Addison-Wesley. [page 44]

Bentley, J. L. (1988). *More Programming Pearls: Confessions of a Coder*. Addison-Wesley. [page 44]

Blinn, J. (1996). *Jim Blinn's Corner: A Trip Down the Graphics Pipeline*. Morgan Kaufmann Publishers, Inc. [page 45]

Blum, A. (2012). *Tubes: A Journey to the Center of the Internet*. Harper Collins. [page 46]

Brooks, F. P. (1995). *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Longman, anniversary edition. [page 45]

Brooks, F. P. (1996). The computer scientist as toolsmith II. *Communications of the ACM*, 39(5):61–68. [page 45]

Brooks, Jr., F. P. (1987). No silver bullet essence and accidents of software engineering. *Computer*, 20(4):10–19. [page 45]

Campbell-Kelly, M. (2010). Be careful what you wish for. *Commun. ACM*, 53(4):25–26. [page 46]

Carroll, L., Gardner (Editor), M., and Tenniel (Illustrator), J. (2000). *The Annotated Alice: The Definitive Edition*. W. W. Norton. [page 44]

Creeger, M. (2010). CTO roundtable: malware defense. *Commun. ACM*, 53(4):43–49. [page 46]

Cusumano, M. A. and Selby, R. W. (1997). How microsoft builds software. *Commun. ACM*, 40(6):53–61. [page 45]

Denning, P. J. (1968). The working set model for program behavior. *Commun. ACM*, 11(5):323–333. [page 44]

Denning, P. J. (2004). Great principles in computing curricula. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 336–341, New York, NY, USA. ACM Press. [page 44]

Dewdney, A. K. (1993). *The (New) Turing Omnibus: 66 excursions in computer science*. Computer Science Press, New York, New York. [page 45]

Dupré, L. (1995). *BUGS in writing, a guide to debugging your prose*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA. [page 46]

Dyson, G. (2012). *Turing's Cathedral: The Origins of the Digital Universe*. Random House. [page 46]

Finkelstein, A., editor (2000). *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, New York, NY, USA. ACM. [page 44]

Fortnow, L. (2009). The status of the P versus NP problem. *Commun. ACM*, 52(9):78–86. [page 46]

Fuller, R. (1981). *Critical Path*. St. Martin's Press. [page 113]

Glick, J. (2011). *The Information: A History, A Theory, A Flood*. Pantheon Press, New York. [page 45]

Greengard, S. (2010). Spies among us? *Commun. ACM*, 53(4):17–19. [page 46]

Hayes, B. (2007). How many ways can you spell v1@gra? *American Scientist*, 94(5). [page 45]

Hilbert, D. (1902). Mathematical problems. *Bulletin of the American Mathematical Society*, 8(10):437–479. [page 99]

Hofstadter, D. R. (1999). *Gödel, Escher, Bach: An Eternal Golden Braid. 20th Anniversary Edition*. Basic Books, New York, New York. [page 45]

Hunt, A. and Thomas, D. (2000). *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley. [page 44]

Juels, A. (2010). Future tense. *Commun. ACM*, 53(3):120–ff. [page 46]

Kernighan, B. W. and Pike, R. (1999). *The Practice of Programming*. Addison-Wesley. [page 44]

Kernighan, B. W. and Plauger, P. J. (1978). *The Elements of Programming Style*, volume 2nd Edition. McGraw-Hill. [page 44]

Knuth, D. E. (1974). Structured programming with `go to` statements. *Computing Surveys*, 6(4):261–301. [page 44]

Knuth, D. E. (1997a). *The Art of Computer Programming: Fundamental Algorithms*, volume 1. Addison-Wesley, third edition. [page 44]

Knuth, D. E. (1997b). *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley, third edition. [page 44]

Knuth, D. E. (1998). *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley, third edition. [page 44]

Koen, B. V. (2003). *Discussion of the Method: Conducting the Engineer's Approach to Problem Solving*. Oxford University Press. [page 59]

Leveson, N. G. and Turner, C. S. (1993). An investigation of the Therac-25 accidents. *Computer*, 26(7):18–41. [page 46]

Lipson, C. (2004). *Doing Honest Work in College: How to Prepare Citations, Avoid Plagiarism, and Achieve Real Academic Success*. University of Chicago Press, Chicago. [page 37]

McConnell, S. (1993). *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press. [page 44]

Mongan, J. and Suojanen, N. (2000). *Programming Interviews Exposed: secrets to landing your next job*. John Wiley & Sons, New York, NY. [page 32], [page 46]

Naur, P. and Randell, B., editors (1969). *Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO*. NATO. [page 44]

Neville-Neil, G. V. (2010a). The data-structure canon. *Commun. ACM*, 53(4):33–34.

Neville-Neil, G. V. (2010b). Plotting away. *Commun. ACM*, 53(6):33–34.

of Chicago Press Staff, U., editor (2010). *The Chicago Manual of Style*. University of Chicago Press. [page 46]

Oram, A. and Wilson, G., editors (2007). *Beautiful Code*. O'Reilly. [page 44], [page 94]

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058. [page 44]

Parnas, D. L. (1996). Why software jewels are rare. *Computer*, 29(2):57–60. [page 44]

Parnas, D. L. (2010). Really rethinking 'formal methods'. *Computer*, 43:28–34. [page 44]

Petzold, C. (2000). *Code: The Hidden Language of Computer Hardware and Software*. Microsoft Press. [page 45]

Poundstone, W. (2003). *How Would You Move Mount Fuji?* Little, Brown and Company, Boston, MA. [page 32], [page 46]

Raymond, E. S. (2010). Eric's random writings. . Retrieved June 18, 2010. [page 45]

Shasha, D. and Lazere, C. (1995). *Out of Their Minds: the lives and discoveries of 15 Great Computer Scientists*. Copernicus/ An Imprint of Springer-Verlag, New York, New York. [page 46]

Shor, P. W. (1994). Polynominal time algorithms for discrete logarithms and factoring on a quantum computer. In *ANTS-I: Proceedings of the First International Symposium on Algorithmic Number Theory*, page 289, London, UK. Springer-Verlag. [page 46]

Slater, R. (1987). *Portraits in Silicon*. MIT Press, Cambridge, Massachusetts. [page 46]

Snyder, L. (2007). *Fluency with Information Technology: Skills, Concepts, and Capabilities*. Addison-Wesley Publishing Company, USA, 3rd edition. [page 87]

Strunk Jr., W. and White, E. B. (1979). *The Elements of Style*. MacMillan Publishing Company. [page 46]

Tufte, E. R. (1990). *Envisioning Information*. Graphics Press, Cheshire, Connecticut. [page 46]

Tufte, E. R. (1997). *Visual Explanations*. Graphics Press, Cheshire, Connecticut. [page 46]

Tufte, E. R. (2001). *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, Connecticut. [page 46]

Tufte, E. R. (2006). *Beautiful Evidence*. Graphics Press, LLC, first edition. [page 46]

Verne, J. (1864). *Journey to the Center of the Earth*. Pierre-Jules Hetzel. [page 59]

Index