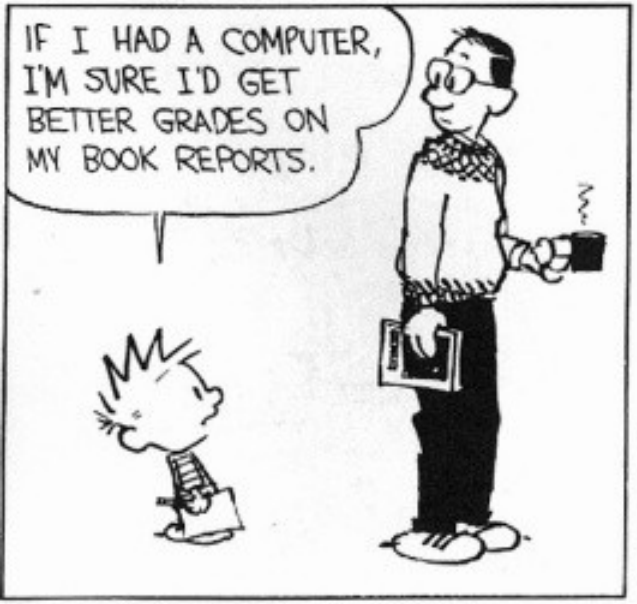
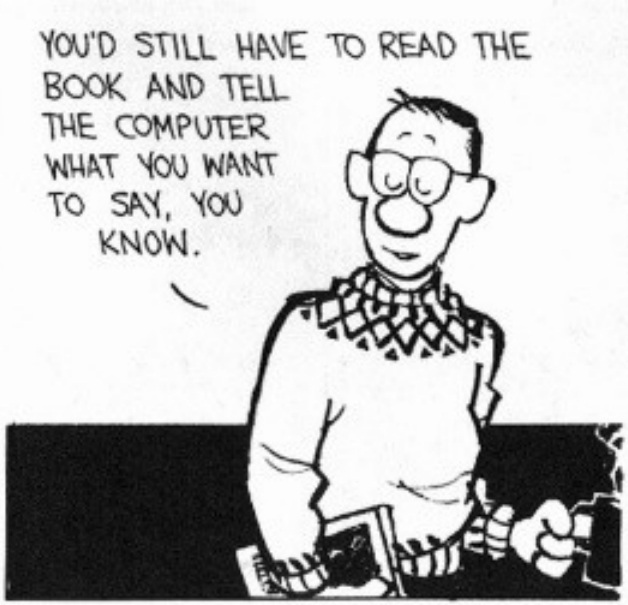
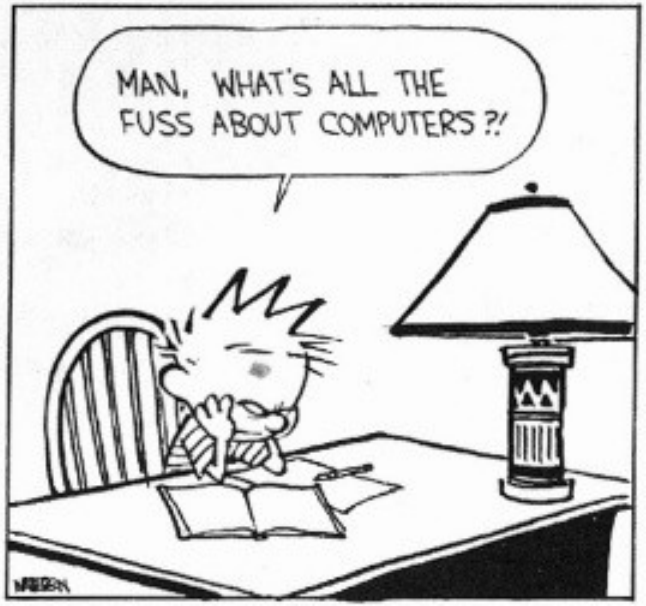
Principles of Programming Languages

Wes Weimer
TR 2:50 - 4:15, Searles 223
w.weimer@bowdoin.edu

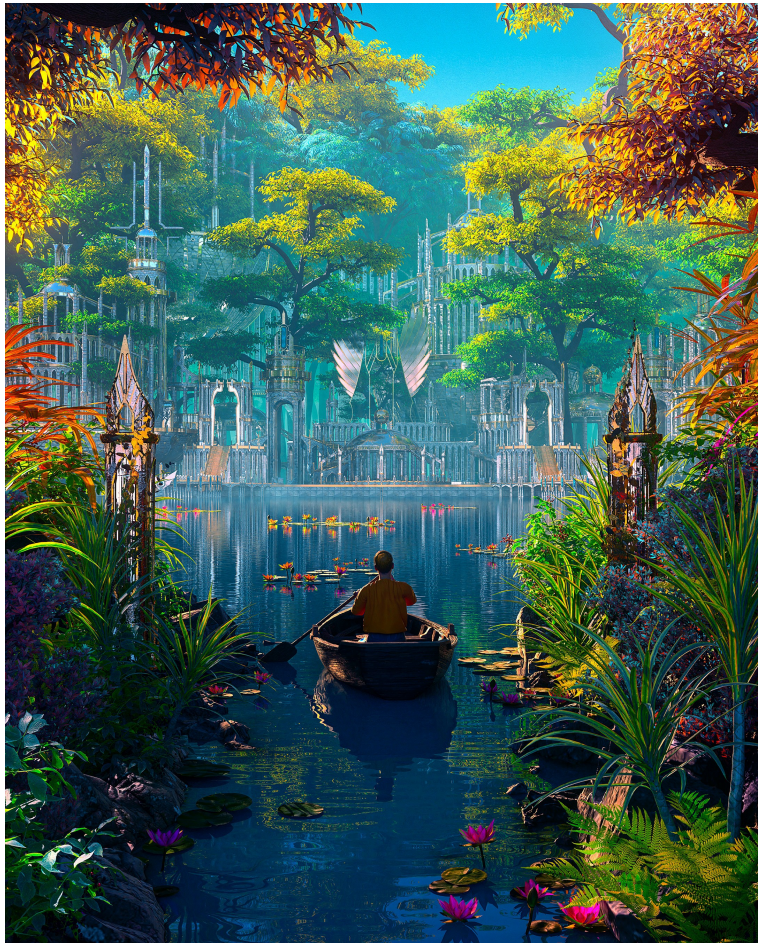






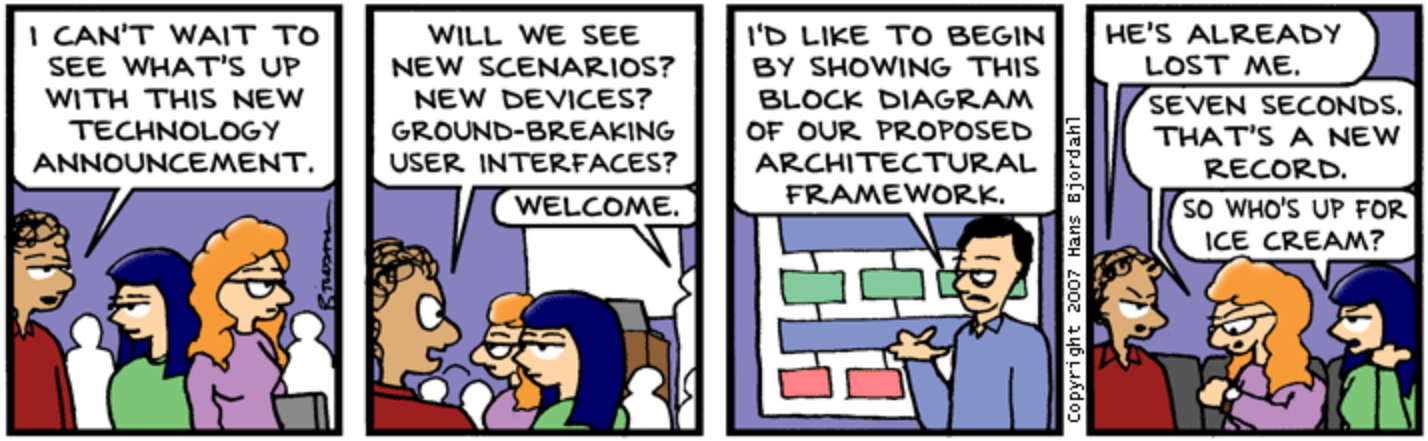
Honestly

- It's going to be an awesome exploration …



Cunning Plan

- Administrivia
  - Webpage, Wes
- What Is This Class About?
- Easy or Hard? Work and Grading.
- Understanding a Program in Stages



Course Home Page

- Find via Canvas or Piazza
- **https://weimer.github.io/csci2320/index.html**
- Lectures slides are available before class
  - You should still take notes! (citation next)
- Assignments are listed
  - also grading breakdown, regrade policies, etc.

- We make heavy use of the class forum

Please Ask Me To Support Claims!

"…students who took notes on laptops performed worse on conceptual questions than students who took notes longhand. We show that whereas taking more notes can be beneficial, laptop note takers' tendency to transcribe lectures verbatim rather than processing information and reframing it in their own words is detrimental to learning."

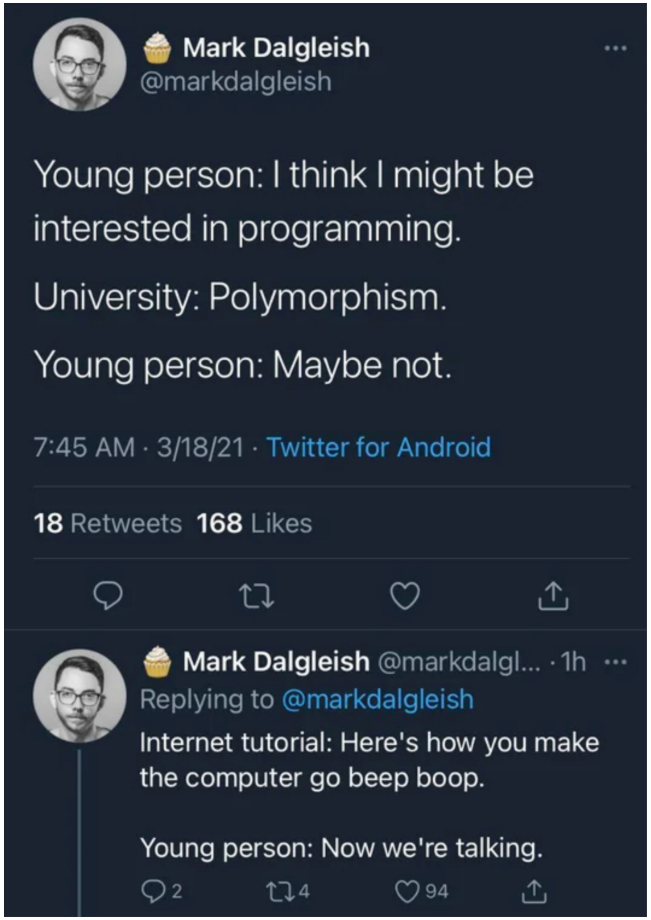
[Pam Mueller, Daniel Oppenheimer. *The pen is mightier than the keyboard: advantages of longhand over laptop note taking*. Psychol. Sci. 2014 Jun; 25(6):Epub 2014 Apr 23.]

Programming Languages Course Goals

- At the end of this course, you will be acquainted with the fundamental concepts in the **design** and **implementation** of high-level programming **languages**. In particular, you will understand the **theory** and **practice** of **lexing**, **parsing**, **semantic** analysis, and **code** interpretation. You will also have gained practical experience programming in **different** languages.

What about you?

- What do you want to do or learn in this class?



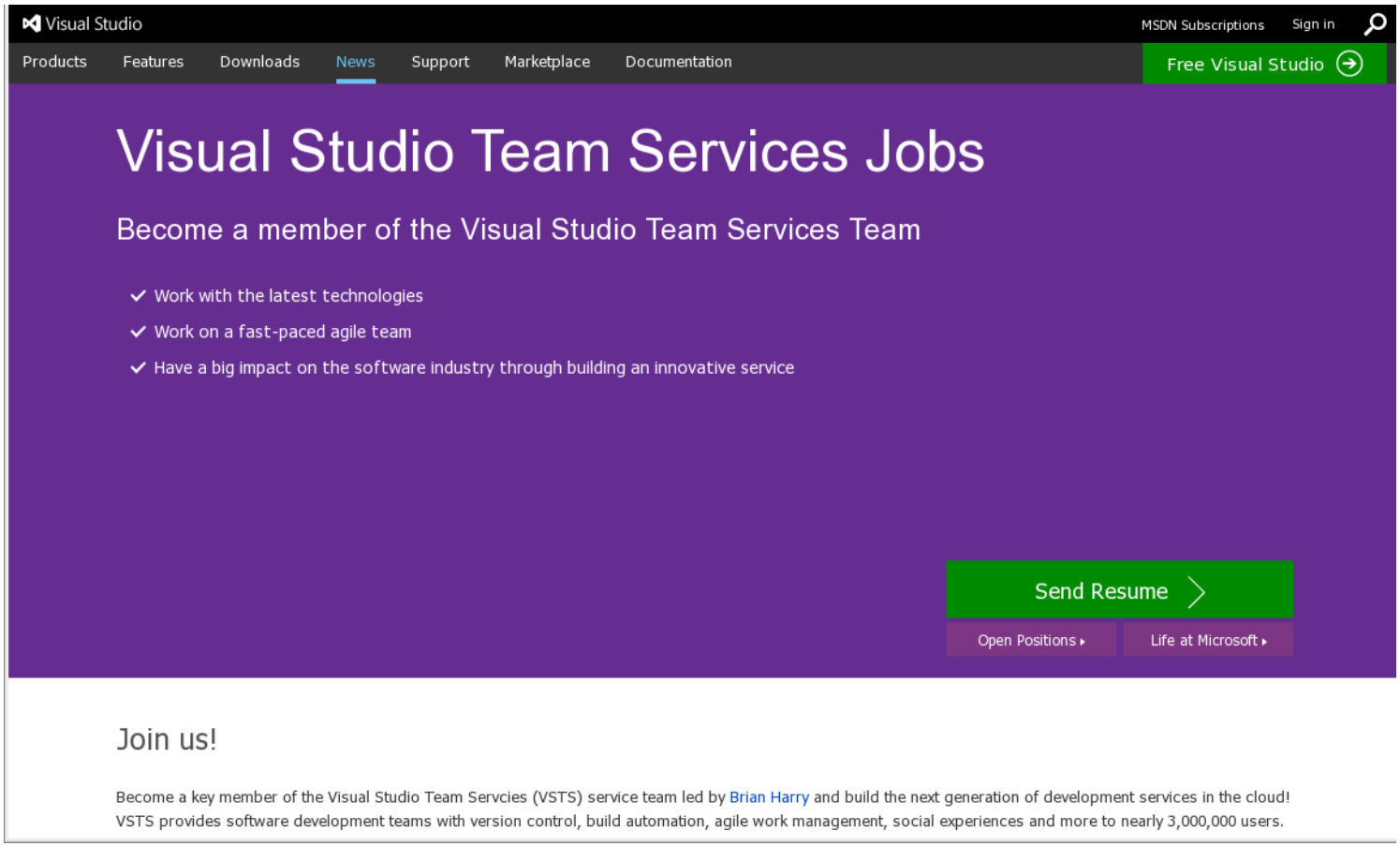
Who Cares?

- In most cases, there is a clear mapping between CS classes and jobs or internships:
- Take Databases → work at Oracle
- Take OS → work at Microsoft
- Take Financial ML → work in FinTech
- Take PL → ???

- *Which companies develop compilers or interpreters?*

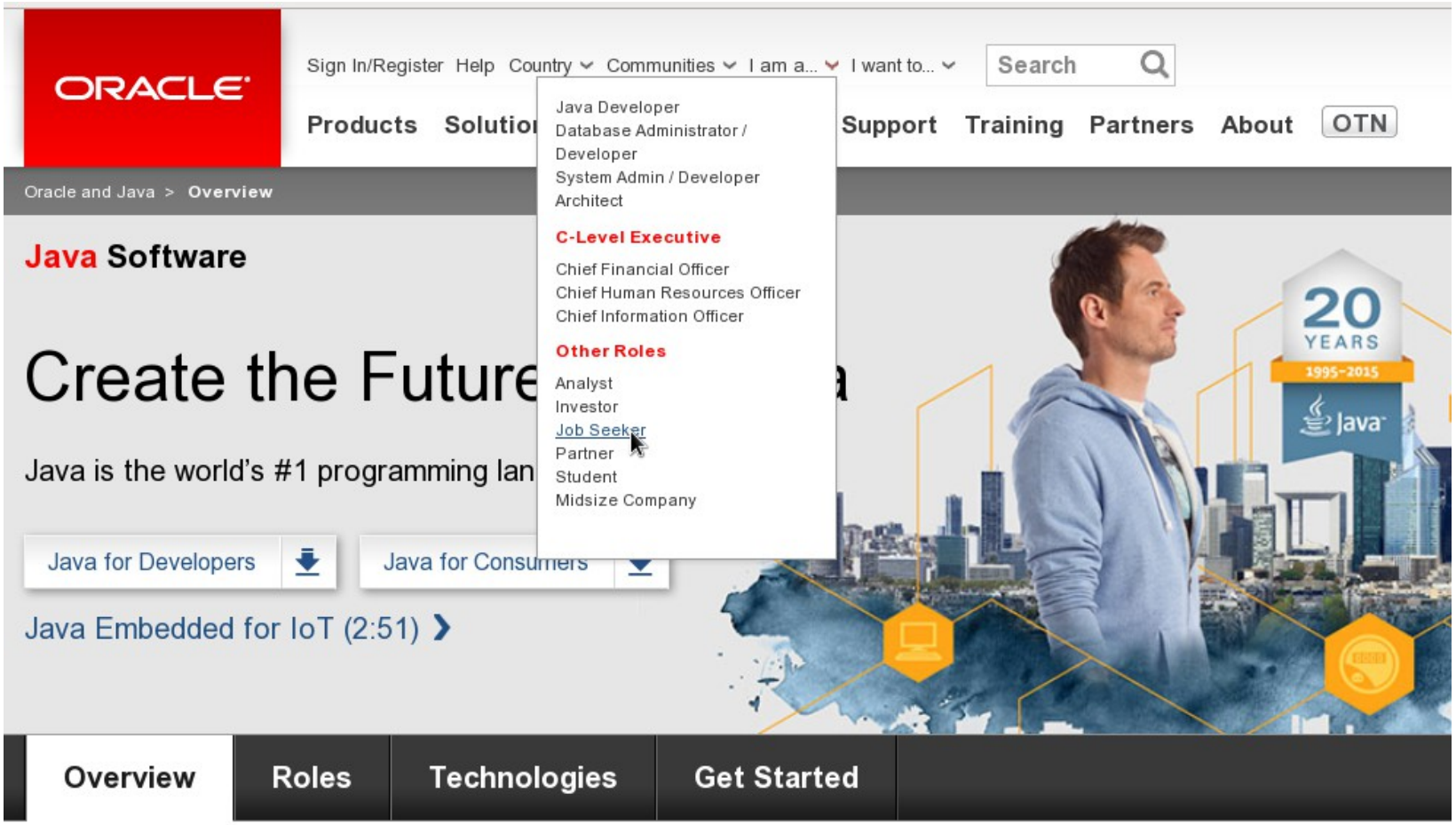
Microsoft

- Visual Studio, Excel, etc.



Oracle

- Java Compiler, Java Virtual Machine



Intel

- ICC

intel Developer Zone

Join Today › Log in

Development › Tools › Resources ›

powered by Google

Intel® C++ Compilers

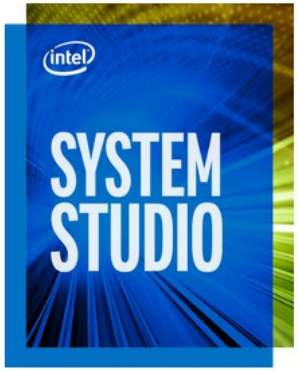
Leadership Application Performance

- Boost C++ application performance
- Future-proof code by making code that scales
- Plugs right into your development environment

If you are here, you are looking for ways to make your application run faster. Boost performance by augmenting your development process with the Intel® C++ Compiler. The Intel C++ Compiler plugs right into popular development environments like Visual Studio*, Eclipse*, XCode*, and Android Studio*; The Intel C++ Compiler is compatible with popular compilers including Visual C++* (Windows*) and GCC (Linux*, OS X* and Android*).

The Intel C++ Compiler is available in four products based on your application development needs:

Intel® C++ Compiler in Intel® Parallel Studio XE



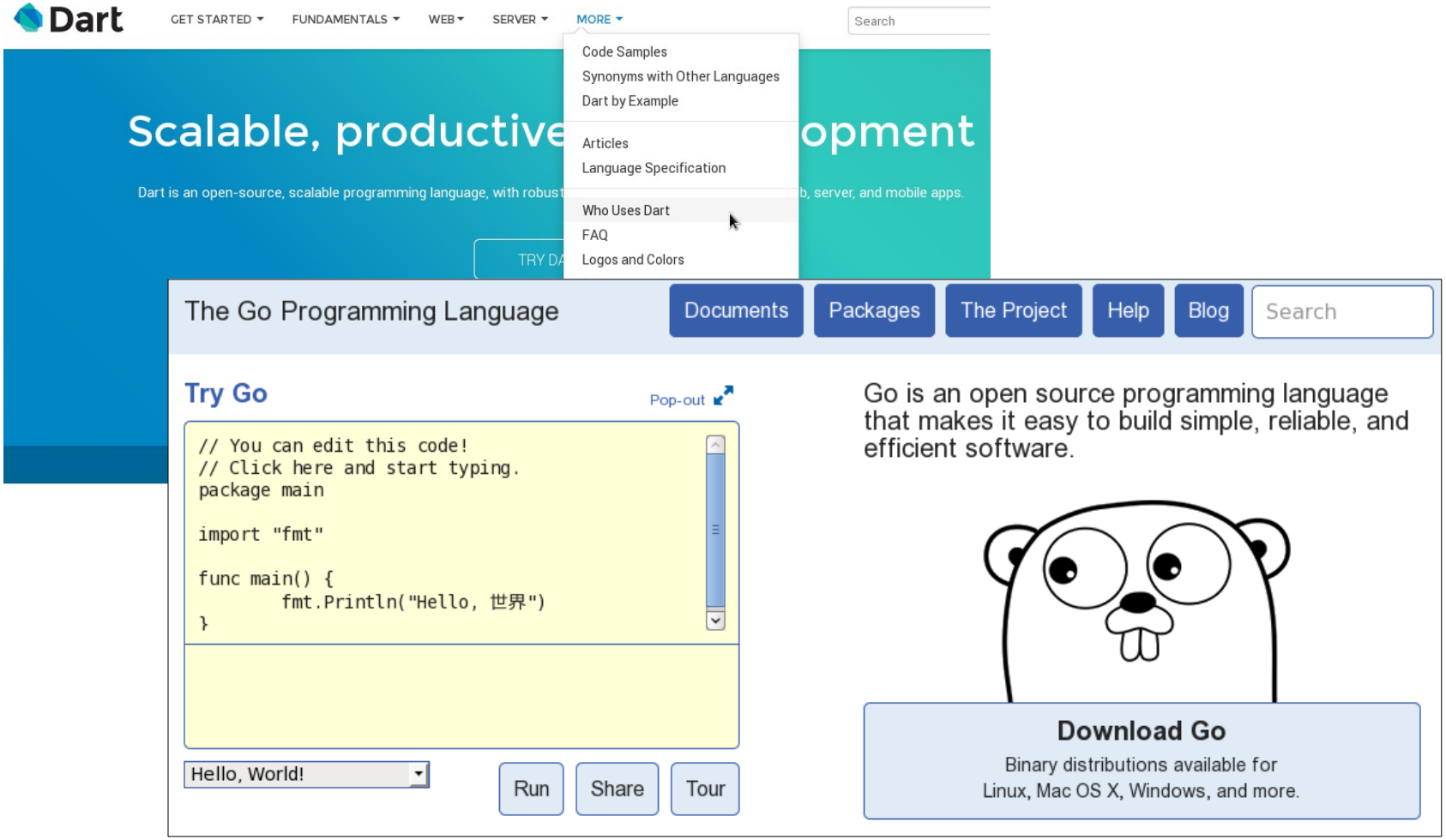
Intel® C++ Compiler in Intel® System Studio

Intel® C++ Compilers in Intel® INDE (support only)

Intel® Bi-Endian Compiler

Google

- Go, Dart, etc.



Wind River, Green Hills

- Embedded!

DIAB COMPILER

For over 25 years, Wind River Diab Compiler has been help
industrial, medical, and aerospace industries. Diab Compile
footprint, and produce high-quality, standards-compliant c

Diab Compiler's unique optimization technology generates extremely fast, high-quality object code in the smallest possible footprint.

Due to collabora
Diab Compiler is
microcontrollers
older processors
of time to allow
of the compiler f

Green Hills SOFTWARE

Leading the Embedded World

INTEGRITY GLOBAL SECURITY

Products | Markets | Benefits | Services | Support | Partners | News | About

Jobs - Opportunities in the USA

Green Hills Software is always looking for qualified Engineering, Sales, and Marketing staff. Please submit your resume to the Corporate Office where it will be processed and reviewed by the hiring manager.

Click on a job title below for a complete description of the position:

- Corporate Field Applications Engineer (Santa Barbara, CA)
- Embedded Software Consultant (Santa Barbara, CA)
- Embedded Solutions Test Engineer (Santa Barbara, CA)
- Field Engineer (Santa Barbara, CA)
- Field Services Engineer (Santa Barbara, CA)
- Functional Safety Software Engineer (Santa Barbara, CA)
- Product Engineer (Santa Barbara, CA)
- Sales Managers (location TBD)
- Software Development Engineer (Santa Barbara, CA)
- Technical Marketing Engineer (Santa Barbara, CA)

Click here for information on applying.
Green Hills Software is an Equal Opportunity / Affirmative Action Employer.

Software Development Engineer (Santa Barbara, CA)

Job description:

A software engineer has complete engineering responsibility for one or more major components of the Green Hills product line. For an experienced programmer this is a satisfying position in which you have personal responsibility for creating a tool used by thousands of programmers around the world. Our engineers are involved in Language Front Ends, Code Generators, Real Time Operating Systems, our MULTI Development Environment, our Secure Workstation, and Target Systems.

Here are the groups for which we are hiring:

- **Compiler:** Create, update, and maintain a language front end or a target architecture backend for the highly-optimizing family of Green Hills compilers. A compiler engineer might work on new language extensions, specific cutting-edge optimizations for the latest chips to hit the market, or on general optimizations that will benefit our entire product line. An ideal candidate understands low level microarchitecture designs and is comfortable working with assembly code, yet can also develop tools written in high level languages.

Wait, what? Embedded?

- Curiosity Mars Rover, Cell Phones, Satellites, Engine Control Modules, Computed Radiology, Fighter Jets, Digital Cameras, Turbines, Anti-Lock Brakes, Wii U Game Console, Nintendo Switch, …

Eight years ago, NASA Jet Propulsion Laboratory (JPL) first began its work on the Mars Science Laboratory rover, Curiosity. Because of its long record of success with Wind River® on more than 20 JPL missions, NASA chose VxWorks® for the most

ranced
spacecraft
ver to be
pace venture.
s powered the
, to its
support

strategic role in
been capable
of supporting life and to assess the planet's habitability for future human missions.

Green Hills Software's MULTI Integrated Development Environment Selected by Nintendo for Wii U Development

Global Agreement Will Yield Richer Games, with Faster Time-to-Market

SAN JOSE, CA — March 27, 2012 — DESIGN West/ESC 2012, Booth #1227 —Green Hills Software, the largest independent vendor of embedded software solutions, has entered into a global license agreement that will enable Nintendo Co., Ltd. to provide Green Hills Software's MULTI® integrated development environment (IDE) to developers that are creating video game software for the upcoming Wii U platform, which is scheduled to be launched later this year.

"We selected the Green Hills Software solution because it generates highly optimized code, and Green Hills provides excellent global support," commented Mr. Genyo Takeda, senior managing director of Integrated Research & Design at Nintendo Co., Ltd.

Nintendo®
Wii U™

Adobe

- Photoshop contains interpreters …

2 Photoshop Scripting Basics

This chapter provides an overview of scripting for Photoshop, describes scripting support for the scripting languages AppleScript, VBScript, and JavaScript, how to execute scripts, and covers the Photoshop object model. It provides a simple example of how to write your first Photoshop script.

If you are familiar with scripting or programming languages, you most likely will want to skip much of this chapter. Use the following list to locate information that is most relevant to you.

- For more information on the Photoshop object model, see "Photoshop Object Model" on page 11.
- For information on selecting a scripting language, refer to the *Introduction to Scripting* guide.
- For examples of scripts created specifically for use with Photoshop, see Chapter 3, "Scripting Photoshop" on page 21.
- For detailed information on Photoshop objects and commands, please use the reference information in the three reference manuals provided with this installation: *Adobe Photoshop CC 2015 AppleScript Scripting Reference*, *Adobe Photoshop CC 2015 Visual Basic Scripting Reference*, and *Adobe Photoshop CC 2015 JavaScript Scripting Reference*.

NOTE: You can also view information about the Photoshop objects and commands through the object browsers for each of the three scripting languages. See "Viewing Photoshop Objects, Commands, and Methods" on page 21.

Scripting Overview

A script is a series of commands that tells Photoshop to perform a set of specified actions, such as applying different filters to selections in an open document. These actions can be simple and affect only a single object, or they can be complex and affect many objects in a Photoshop document. The actions can call Photoshop alone or invoke other applications.

Mozilla

- SpiderMonkey JavaScript Engine

Sign in with

mozilla

MDN MOZILLA DEVELOPER NETWORK

WEB PLATFORM ▾ MOZILLA DOCS ▾ DEVELOPER TOOLS FEEDBACK ▾

MDN > Mozilla > Projects > SpiderMonkey

LANGUAGES EDIT

SpiderMonkey

see all contributors

SEE ALSO

SpiderMonkey

References:

- JSAPI reference
- Debugger-API

Guides:

- General
- SpiderMonkey internals

Contributing to SpiderMonkey:

- Getting started
- Tests

Releases:

- Release notes

Documentation:

- Useful lists

SpiderMonkey is Mozilla's JavaScript engine written in C/C++. It is used in various Mozilla products, including Firefox, and is available under the MPL2.

SpiderMonkey 38 is the most recent standalone source code release. It is largely the same engine that shipped with Firefox 38 (ESR). Full source code is available here: https://people.mozilla.org/~sstangl/mozjs-38.2.1.rc0.tar.bz2

The next release will be SpiderMonkey 45.

Guides

Building

SpiderMonkey Build Documentation
How to get SpiderMonkey source code, build it, and run the test suite.

Using SpiderMonkey

Reference

JSAPI Reference
SpiderMonkey API reference.

JS Debugger API Reference
API reference for the Debugger object introduced in SpiderMonkey 1.8.6, which corresponds to Gecko 8.0 (Firefox 8.0 / Thunderbird 8.0 / SeaMonkey 2.5).

Apple

- Objective-C. LLVM.

Mac Developer Library Developer

LLVM Compiler Overview

LLVM Compiler Overview

The LLVM compiler is the next-generation compiler, introduced in Xcode 3.2 for Snow Leopard, based on the open source LLVM.org project. The LLVM.org project employs a unique approach of building compiler technologies as a set of libraries. Capable of working together or independently, these libraries enable rapid innovation and the ability to attack problems never before solved by compilers. Multiple technology groups within Apple are active contributors within the LLVM.org community, and they use LLVM technology to make Apple platforms faster and more secure.

In Xcode, the LLVM compiler uses the Clang front end (a C-based languages project on LLVM.org) to parse source code and turn it into an interim format. Then the LLVM code generation layer (back end) turns that interim format into final machine code. Xcode also includes the LLVM GCC compiler, which uses the GCC compiler front end for maximum compatibility, and the LLVM back end, which takes advantage of LLVM's advanced code generator. This shows the flexibility of a library-based approach to compiler development. There are many other features, such as link-time optimization, more detailed diagnostic information, and even static analysis, that are made available to Xcode due to the adoption of LLVM.

About Objective-C

Objective-C is the primary programming language you use when writing software for OS X and iOS. It's a superset of the C programming language and provides object-oriented capabilities and a dynamic runtime. Objective-C inherits the syntax, primitive types, and flow control statements of C and adds syntax for defining classes and methods. It also adds language-level support for object graph management and object literals while providing dynamic typing and binding, deferring many responsibilities until runtime.

Games

- Unreal Engine: Blueprints Scripting

Unreal Engine 4 Documentation

SHARE:

Blueprints Visual Scripting

Unreal Engine 4.9

The **Blueprints Visual Scripting** system in Unreal Engine is a complete gameplay scripting system based on the concept of using a node-based interface to create gameplay elements from within Unreal Editor. This system is extremely flexible and powerful as it provides the ability for designers to use virtually the full range of concepts and tools generally only available to programmers.

Through the use of Blueprints, designers can prototype, implement, or modify virtually any gameplay element, such as:

QuakeC

QuakeC is an interpreted language developed in 19
id Software to program parts of the video game *Qua*
programmer is able to customize *Quake* to great ext
weapons, changing game logic and physics, and pr
scenarios. It can be used to control many aspects of
parts of the AI, triggers, or changes in the level. The
only game engine to use QuakeC. Following engine
modules for customization written in C and C++ from id Tech 4 on.

Typing discipline	static, strong
Major implementations	
Quake C Compiler, FastQCC, QCCx, GMQCC	
Influenced by	
C	

Contents

Games (continued)

- Osiris (Larian), Papyrus (Bethesda)

Papyrus

SIGN IN T

Papyrus is an object-oriented scripting language developed by Bethesda Softworks for use in the
Engine. Papyrus first appeared in *Elder Scrolls V: Skyrim* and has since been utilized in *Fallout 4* and
Papyrus is an evolution of ObScript,[1] the previous scripting language used in *Fallout 3* and *Fallout*
with additional functionality and flexibility along with completely new syntax and workflow.

Papyrus works by receiving in-game Events and sending Function Calls. Papyrus is the driver for all
allows tracking of variables based on player interactions. Papyrus also handles the use of triggers,
terminal and book interactions, as well as special interactions through dialogue with characters.

Unlike the previous language, Papyrus scripts must be written in a separate text editor and compile
Creation Kit to be used in-game. Plugins for Notepad++ and Atom allow for use of Papyrus syntax.

Introduction

Osiris is a mostly declarative programming language, similar to Prolog
programming, the language itself is quite simple and only has a few con
progress; there are 879 Osiris APIs at the time of writing). These APIs a
(queries), and to change the game state (calls).

If you are familiar with databases, you can think of an Osiris program as
remove tables and rows in reaction to other database entries getting modified. Additionally, the
the game state. Note that the tables are actually referred to as "databases" in Osiris, so below
table is generally referred to as "defining a fact", although "defining a database" or "defining a

Here is an example Osiris code fragment:

```
INIT
        DB_MyPrefix_Fruit("Apple");             // Defines the database
a fact consisting of the string "Apple" to it.
        DB_MyPrefix_Fruit("Pear");              // Adds another fact to
        DB_MyPrefix_Fruit("Banana");            // And one more.
```

Is the scripting language for the hot new game really “similar to Prolog”? What does that mean?

Compilers and Interpreters

- Back End Optimization, Chips, etc.
  - Intel, AMD, nVidia, Green Hills, etc.
- Platform Vendors
  - Apple, Oracle, etc.
- Tooling, IDEs
  - Microsoft, Google, etc.
- Domain-Specific Languages
  - Photoshop, Game Studies, MATLAB, SQL, Wolfram Alpha, etc.

Surprise: Postscript /PDF

The language

PostScript is a Turing-complete programming language, belonging to the concatenative group. Typically, PostScript programs are not produced by humans, but by other programs. However, it is possible to write computer programs in PostScript just like any other programming language.[5]

PostScript is an interpreted, stack-based language similar to Forth but with strong dynamic typing, data structures inspired by those found in Lisp, scoped memory and, since language level 2, garbage collection. The language syntax uses reverse Polish notation, which makes the order of operations unambiguous, but reading a program requires some practice, because one has to keep the layout of the stack in mind. Most *operators* (what other languages term *functions*) take their arguments from the stack, and place their results onto the stack. *Literals* (for example, numbers) have the effect of placing a copy of themselves on the stack. Sophisticated data structures can be built on the *array* and *dictionary* types, but cannot be declared to the type system, which sees them all only as arrays and dictionaries, so any further typing discipline to be applied to such user-defined "types" is left to the code that implements them.

Portable Document Format (**PDF**), standardized as **ISO 32000**, is a file format developed by Adobe in 1992 to present documents, including text formatting and images, in a manner independent of application software, hardware, and operating systems.[2][3] Based on the PostScript language, each PDF file encapsulates a complete description of a fixed-layout flat document, including the text, fonts,

Surprise: Postscript /PDF

The language

PostScript is a Turing-complete programming language, belonging to the concatenative group. Typically, PostScript programs are not produced by humans, but by other programs. However, it is possible to write computer programs in PostScript just like any other programming language.[5]

PostScript is an interpreted, stack-based language similar to Forth but with strong dynamic typing, data structures inspired by those found in Lisp, scoped memory and, since language level 2, garbage collection. The language syntax uses reverse Polish notation, wh
one has to keep the
the stack, and place
on the stack. Sophi
system, which sees
"types" is left to the

Portable D
developed by Adobe in 1992 to present documents, including text formatting and images, in a manner independent of application software, hardware, and operating systems.[2][3] Based on the PostScript language, each PDF file encapsulates a complete description of a fixed-layout flat document, including the text, fonts,

Any PDF reader, a cell phone, a game console, a car or bus: they all contain *Interpreters*.

Who Cares?

- The *computer* is unique among “machines” (e.g., lever, pulley, etc.) in that it magnifies our mental force rather than our physical force.
  - Computers can assist with decision making, model and predict outcomes, etc.
- Programming Languages are the mechanism for communicating with and commanding the only tool available that magnifies your mind.

Plus Work, Double-Plus Easy

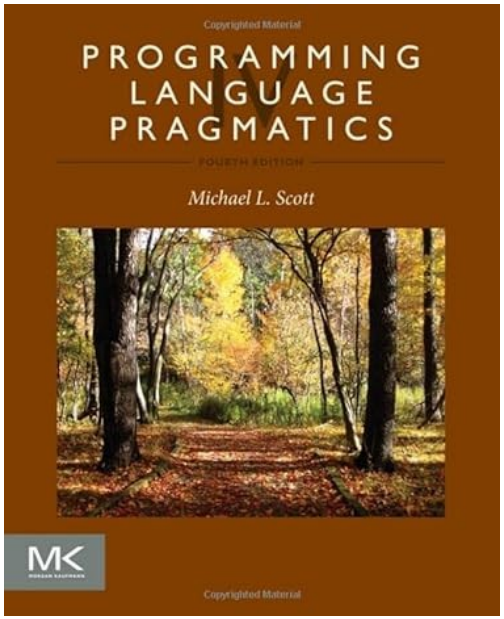
- Unhappiness is related to unrealized desires or unmet expectations
- At some schools, PL is arguably the most difficult CS course
- Principles of Programming Languages (CSCI 2320) is approachable
  - Significant theoretical component
  - Significant programming component
  - Generous curve and opportunities for success

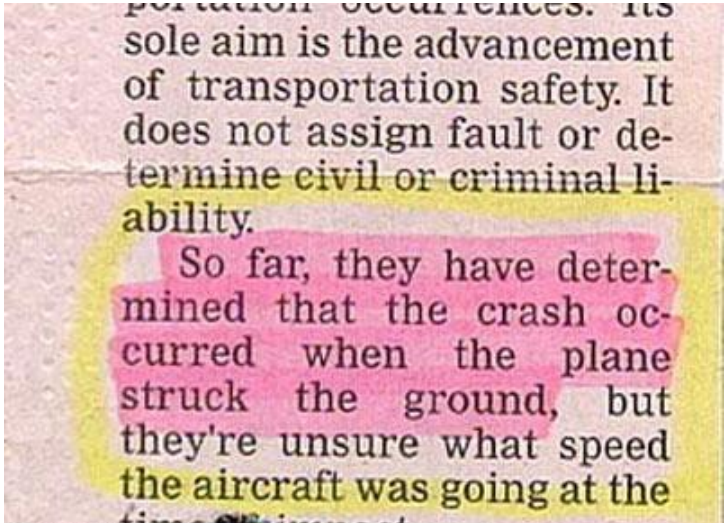
Course Structure

- Course has theoretical and practical aspects
  - Best of both worlds!
- Need both in programming languages!
- Reading = both
  - Many external and optional readings
- Review Sets = theory
  - Not graded, practice problems for exams
- Programming Assignments = practice
  - Electronic hand-in
- Strict deadlines but 3x “late days”
  - Ask me why …

Resources

- Textbook
  - Programming Language Pragmatics
  - Michael L. Scott
- Video Guides
- Free Online Materials
  - Udacity CS 262
- Optional Readings




sole aim is the advancement of transportation safety. It does not assign fault or determine civil or criminal liability.

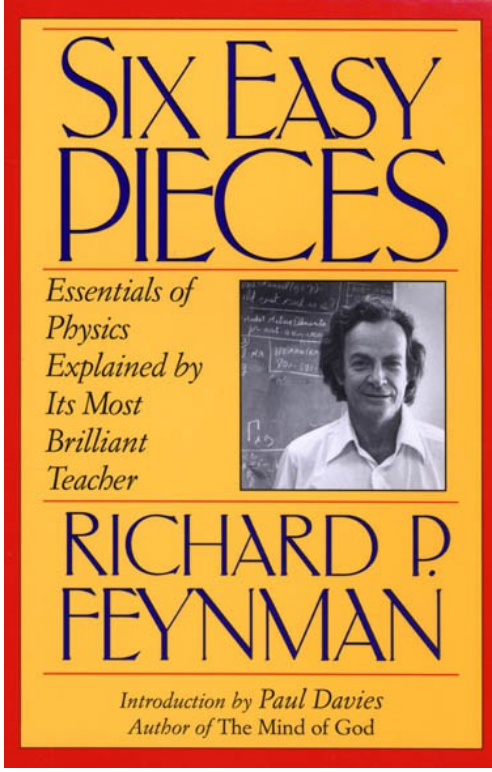
So far, they have determined that the crash occurred when the plane struck the ground, but they're unsure what speed the aircraft was going at the

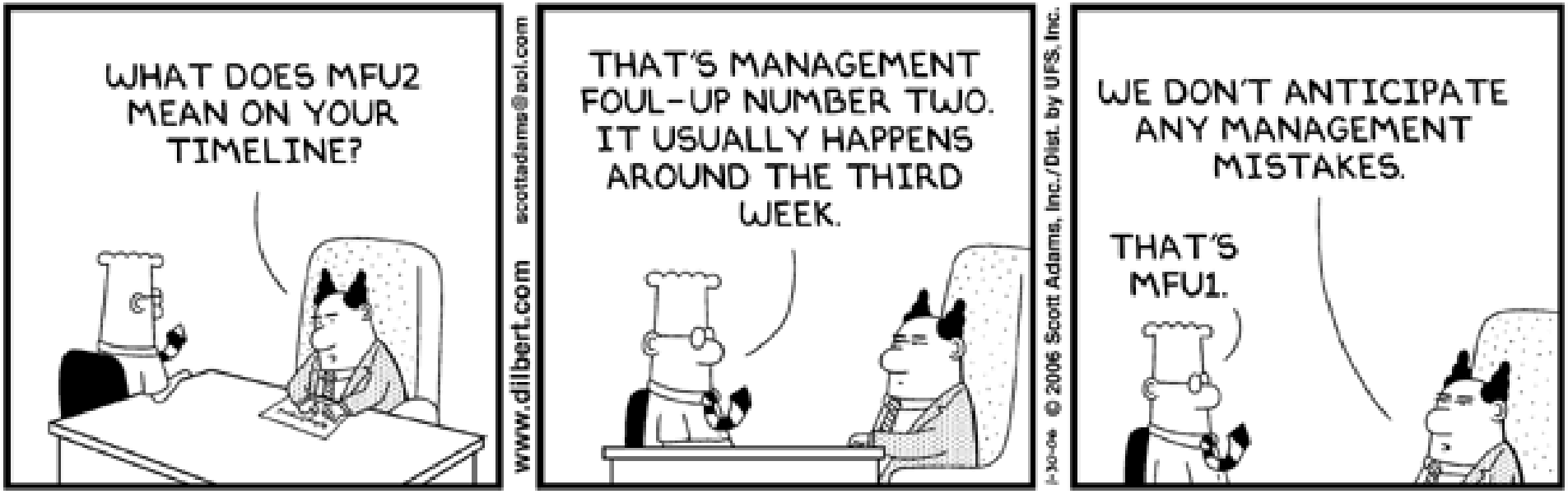
Academic Honesty

- Don’t use work from uncited sources
  - Including old code
- We often use plagiarism detection software
- Class discussion later
  - ChatGPT: allowed? No?

LDI Course Project

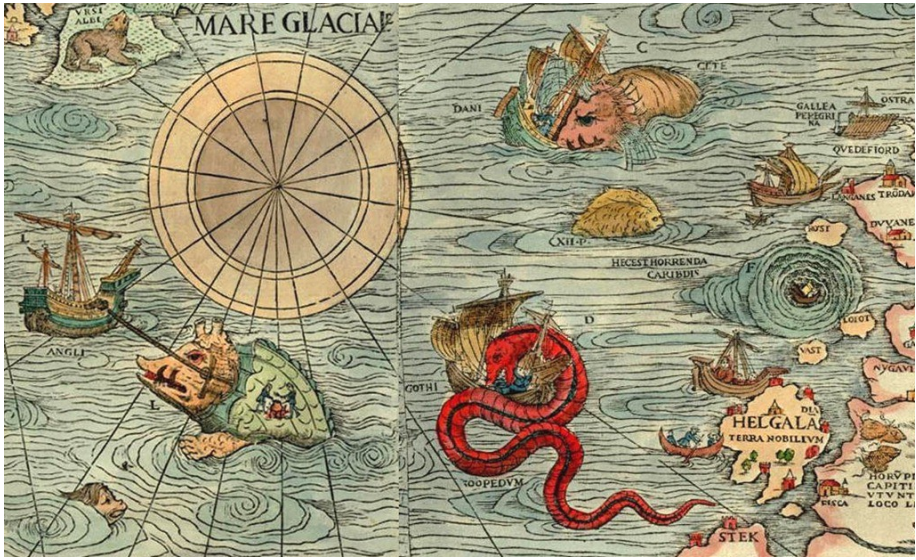
- A big project: an Interpreter!
- … in four easy parts
- You may optionally work in pairs.





“Explaining Unicorns & Dragons”

- Visual Studio, JVM, Exceptions, Memory, Debugging, Linking, Shared Libraries, ...

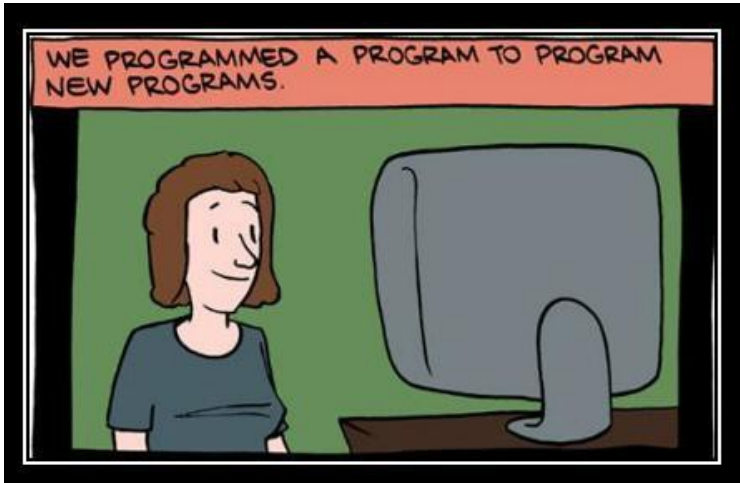


How are Languages Implemented?

- Two major strategies:
  - Interpreters (take source code and run it)
  - Compilers (translate source code, run result)
  - Distinctions blurring (e.g., just-in-time compiler)
- Interpreters run programs “as is”
  - Little or no preprocessing
- Compilers do extensive preprocessing
  - Most implementations use compilers

(Short) History of High-Level Languages

- 1953 IBM develops the 701 "Defense Calculator"
  - 1952, US formally ends occupation of Japan
  - 1954, Brown v. Board of Education of Topeka, Kansas
- **All programming done in assembly**
- **Problem: Software costs exceeded hardware costs!**
- John Backus: "Speedcoding"
  - An interpreter
  - Ran 10-20 times slower than hand-written assembly



FORTRAN I

Set in ~1961

- 1954 IBM develops the 704
- John Backus
  - Idea: translate high-level code to assembly
  - Many thought this impossible

- 1954-7 FORTRAN I project
- By 1958, >50% of all software is in FORTRAN
- Cut development time dramatically
  - (2 weeks → 2 hours)

FORTRAN I


Grace Hopper

- The first **compiler**
  - Produced code almost as good as hand-written
  - Huge impact on computer science
- Led to an enormous body of theoretical work
- Modern compilers keep the outlines of FORTRAN I

Changeups and Trivia

- "[Professors who] deliberately and consistently interspersed their lectures with … some other form of deliberate break … usually commanded a better attention span from the class, and these deliberate variations had the effect of postponing or even eliminating the occurrence of an attention break"

  [Johnstone and Percival. *Attention breaks in lectures*. Education in Chemistry, 13. 49-50, 1976.]

  [Middendorf and Kalish. *The "Change-up" in Lectures*. TRC Newsletter, 8:1 (Fall 1996).]

Real-World Languages

- This Indo-European language is associated with South Asian Muslims and is the lingua franca of Pakistan. It developed from Persian, Arabic and Turkic influences over about 900 years. Poetry in this language is particularly famed, and is a reported favorite of former US President Barack Obama.
- Example: السلام عليكم

Interpreters	Compilers
Lexical Analysis	Lexical Analysis
Parsing	Parsing
Semantic Analysis	Semantic Analysis
Optimization (optional)	Optimization (optional)
Interpret The Program	Generate Machine Code
	Run that Machine Code

The first three may benefit from an analogy to how humans comprehend English.

Lexical Analysis

- First step: recognize words.
  - Smallest unit above letters

This is a sentence.

- Note the
  - Capital "T" (start of sentence symbol)
  - Blank " " (word separator)
  - Period "." (end of sentence symbol)

More Lexical Analysis

- Lexical analysis is not trivial. Consider:

  How d'you break "this" up?

- Plus, programming languages are typically more cryptic than English:

  ***p->f += -.12345e-6**

And More Lexical Analysis

- Lexical analyzer divides program text into "words" or tokens

  if x == y then z = 1; else z = 2;

- Broken up:

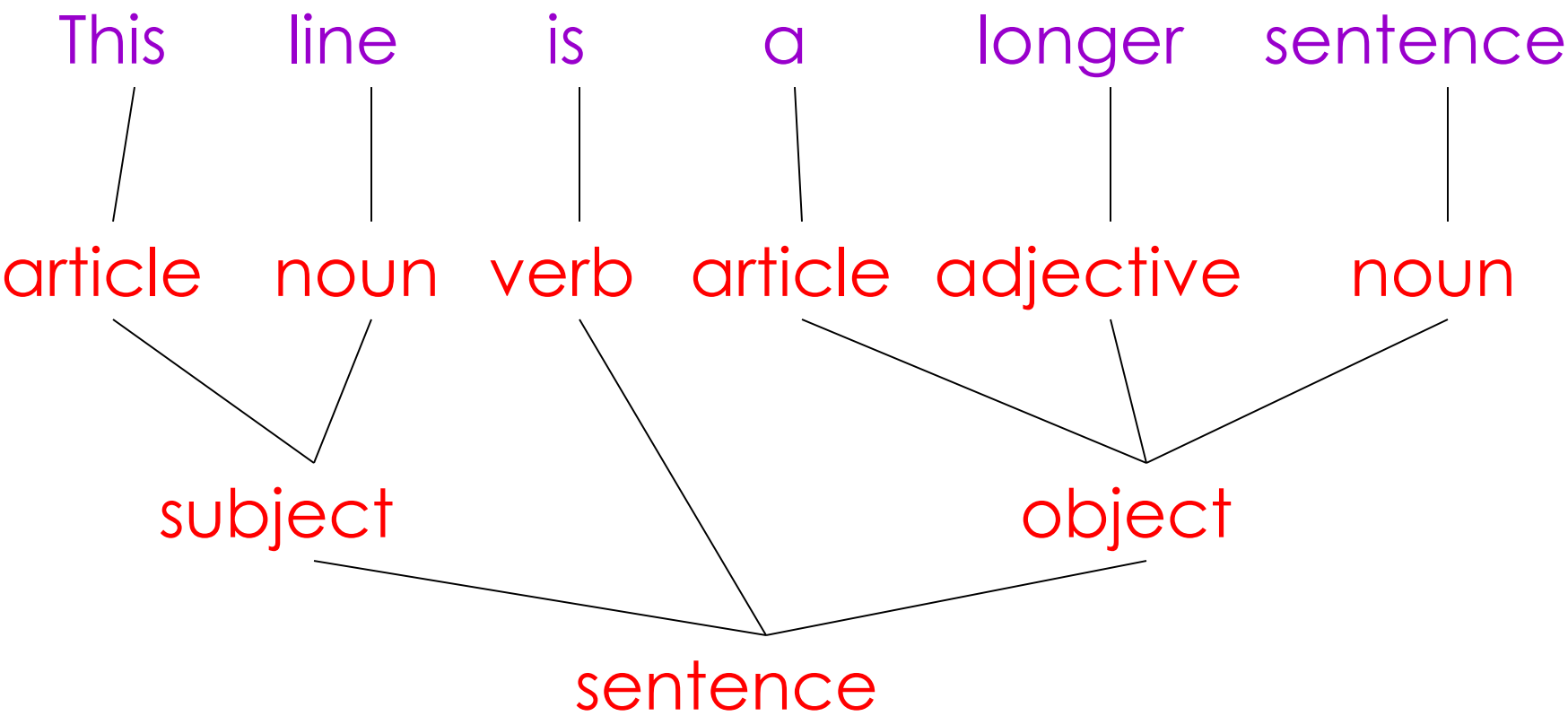
  if, x, ==, y, then, z, =, 1, ;, else, z, =, 2, ;

Parsing

- Once words are understood, the next step is to understand sentence structure

- Parsing = Diagramming Sentences
  - The diagram is a tree
  - Often annotated with additional information

Diagramming a Sentence

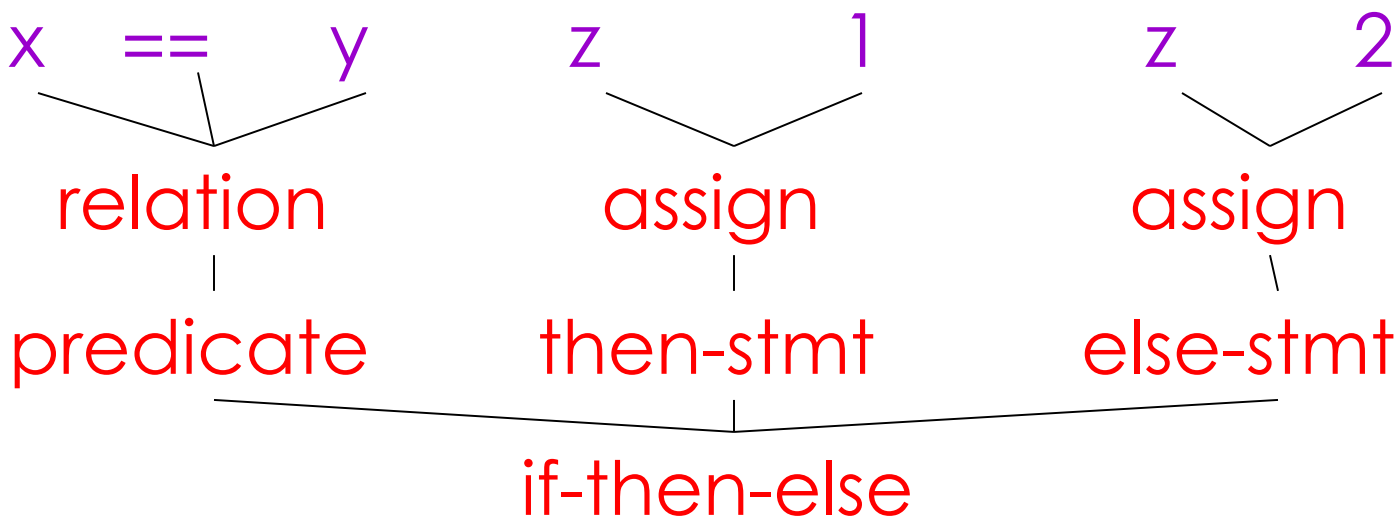


Parsing Programs

- Parsing program expressions is the same
- Consider:

  if x == y then z = 1; else z = 2;

- Diagrammed:



Semantic Analysis

- Once sentence structure is understood, we can try to understand "meaning"
  - But meaning is too hard for compilers
- Compilers perform limited analysis to catch inconsistencies: reject bad programs early!
- Some do more analysis to improve the performance of the program

Semantic Analysis in English

- Example:

Arya said Sansa left her direwolf at home.

What does “her” refer to? Arya or Sansa?

- Even worse:

No One said No One left her mask at home.

How many “No One”s are there?

Which one left the mask?

It's context-sensitive!

Semantic Analysis in Programming

- Programming languages define strict rules to avoid such ambiguities
- This C++ code prints "4"; the inner definition is used

```
{
    int Sydney = 3;
    {
        int Sydney = 4;
        cout << Sydney;
    }
}
```

Scoping or *aliasing* problem.

Finding Disagreements

- Compilers perform many semantic checks besides variable bindings
- *Potential* example:

  Gregory House left her cane at home.

- This may be a "type mismatch" between her and Gregory House (depending on gender)
  - If "him" should be used for this Gregory House
  - Just as in real life, this is context-sensitive!
    - "Her" can be correct for some Gregory Houses.

Optimization

- No strong counterpart in English, but akin to editing (cf. poems, short stories)

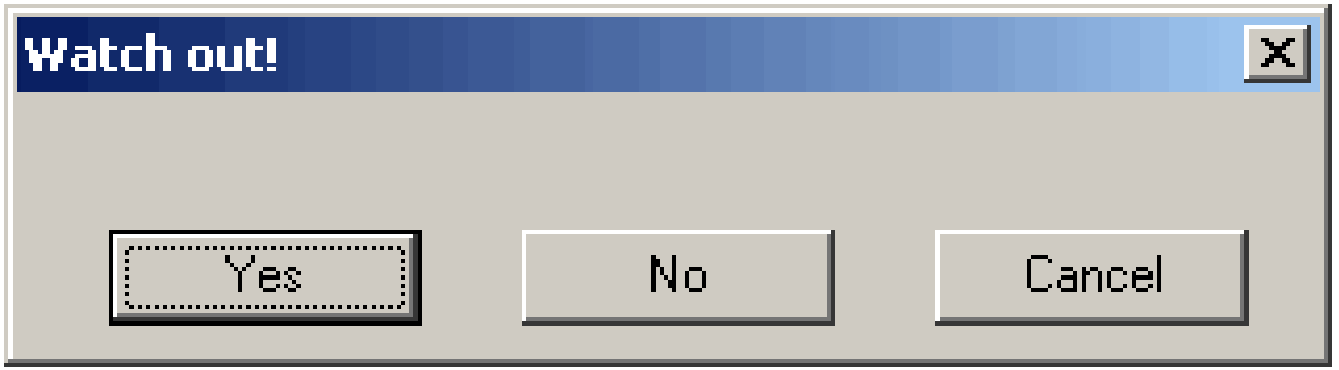
- Automatically modify programs so that they
  - Run faster
  - Use less memory
  - Use less energy (e.g., on your phone)
  - In general, conserve some resource

Code Generation

- Produces assembly code (usually)
  - which is then assembled into an executable by an assembler
- A translation into another language
  - Analogous to human translation
- (Code generation in PL is not really analogous to ChatGPT. We'll cover it later!)

Issues

- Compiling and interpreting are almost this simple, but there are many pitfalls.
- Example: How are bad programs handled?
- Language design has big impact on compiler
  - Determines what is easy and hard to compile
  - Course theme: **trade-offs in language design**



Languages Today

- The overall structure of almost every compiler & interpreter follows our outline

- The proportions have changed since FORTRAN
  - Early: lexing, parsing most complex, expensive
  - Today: optimization dominates all other phases, lexing and parsing are cheap
  - Thus: this course puts no emphasis on ancient parsing optimizations (e.g., LL, LALR)

Trends in Languages

- Optimization for speed is less interesting. But:
  - scientific programs
  - advanced processors (Digital Signal Processors, advanced speculative architectures)
  - small devices where speed = longer battery life
- Ideas we'll discuss are used for improving code reliability:
  - memory safety
  - detecting concurrency errors (data races)
  - type safety
  - automatic memory management
  - …

Why Study Prog. Languages?

- Prepare for many good jobs
- Increase capacity of expression
- Improve understanding of program behavior
  - Know how things work "under the hood"
- Increase ability to learn new languages
- Learn to build a large and reliable system
- See many basic CS concepts at work
- Computers are the only tools that increase cognitive power, so learn to control them

What Will You Do In This Class?

- Reading (textbook, videos, outside sources)
- Learn about different kinds of languages
  - Imperative vs. Functional vs. Object-Oriented
  - Static typing vs. Dynamic typing
  - etc.
- Gain exposure to new languages (ML, Cool)

- Write an interpreter!

What Is This?

A lungo il mio cuore di tali ricordi ha voluto colmarsi!

Come un vaso in cui le rose sono state dissetate:

Puoi romperlo, puoi distruggere il vaso se lo vuoi,

Ma il porfumo delle rose sarà sempre tutt'intorno.

Długo, długo moje serce przepełnione było takimi wspomnieniami!

Były jak waza, w której kiedyś róże destylowały:

Możesz sprawić by pekła, możesz gruchotać waze jeśli chcesz,

Ale zapach róż bedzie wciaż czuć dookoła.

Mon coeur est brûlant rempli de tels souvenirs

Comme un vase dans lequel des roses ont été distillées:

Tu peux le briser, tu peux détruire le vase si tu le désires,

Mais la senteur des roses sera toujours là.

Lang, lang soll die Erinnerung in meinem Herzen klingen!

Gleich einer Vase, drin Rosen sich einst tränkten:

Lass sie zerbrechen, lass sie zerspringen,

Der Duft der Rose bleibt immer hängen.

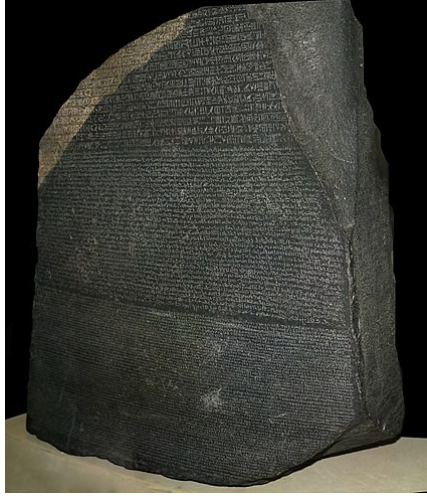
Muito, muito tempo seja meu coração preenchido com tais lembranças!

Tal qual o vaso onde rosas foram uma vez destiladas:

Pode quebrar, pode estilhaçar o vaso se o desejas,

Mas perdurará para sempre o aroma das rosas perfumadas.

The Rosetta Stone



- The first programming assignment involves writing the same simple (50-75 line) program in two languages:
  - **Ocaml and Cool (with Ruby, Python, JavaScript, Haskell and C as Extra Credit)**
- PA1c, **due Tue Jan 30**, requires you to write the program in one language
- PA1, due subsequent Thursday, requires both

Long, long be my heart with such memories fill'd!
Like the vase in which roses have once been distill'd:
You may break, you may shatter the vase if you will,
But the scent of the roses will hang round it still.
- Thomas Moore (Irish poet, 1779-1852)

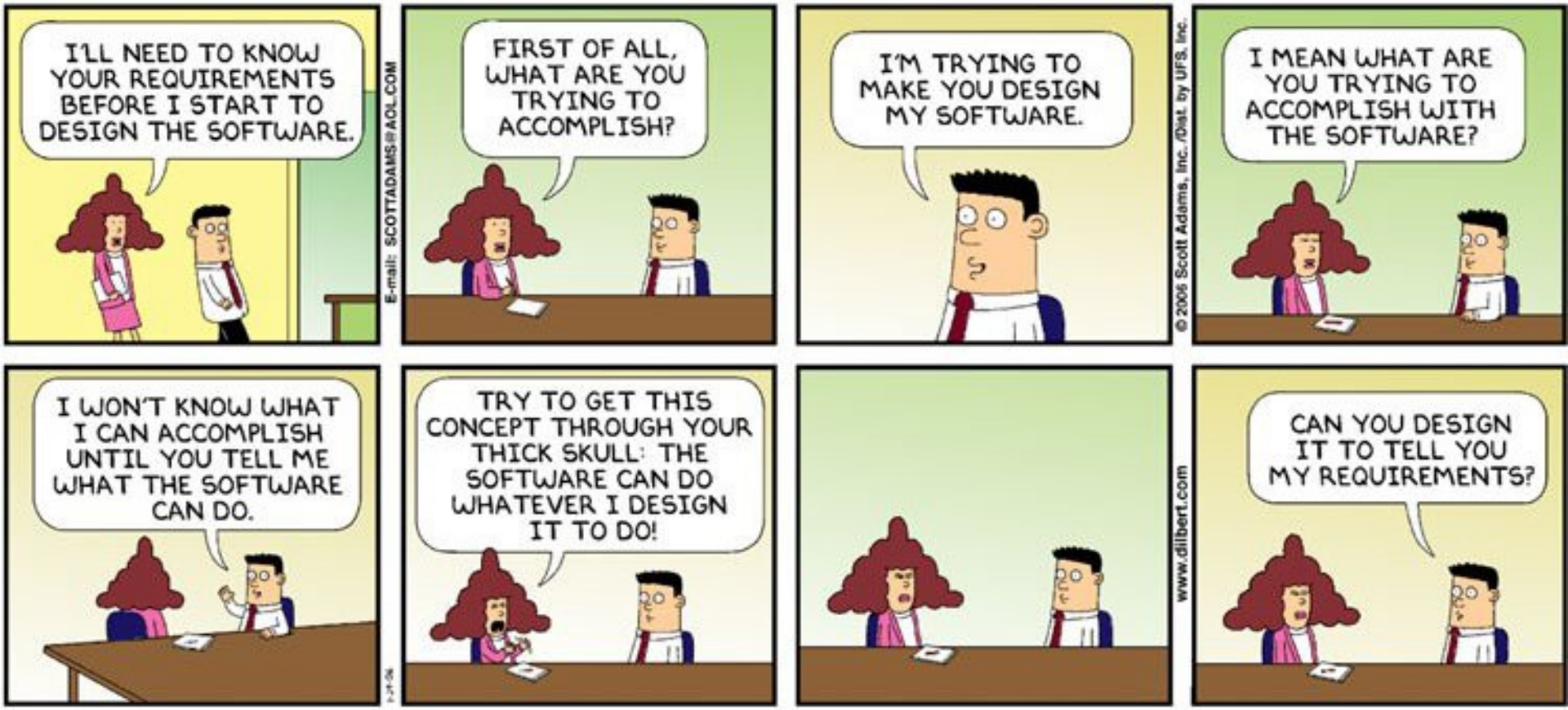
Partial Automated Grading

- You should think about the project on your own, write your own (local) test cases, and then submit to the grading server
  - Limited submissions per day
  - Discourages “guess and check”

- Continuous Integration Testing, Alpha & Beta Testing
- Ecological Validity
- Job Interviews (LeetCode, HackerRank, etc.)
- Grade and Time Control

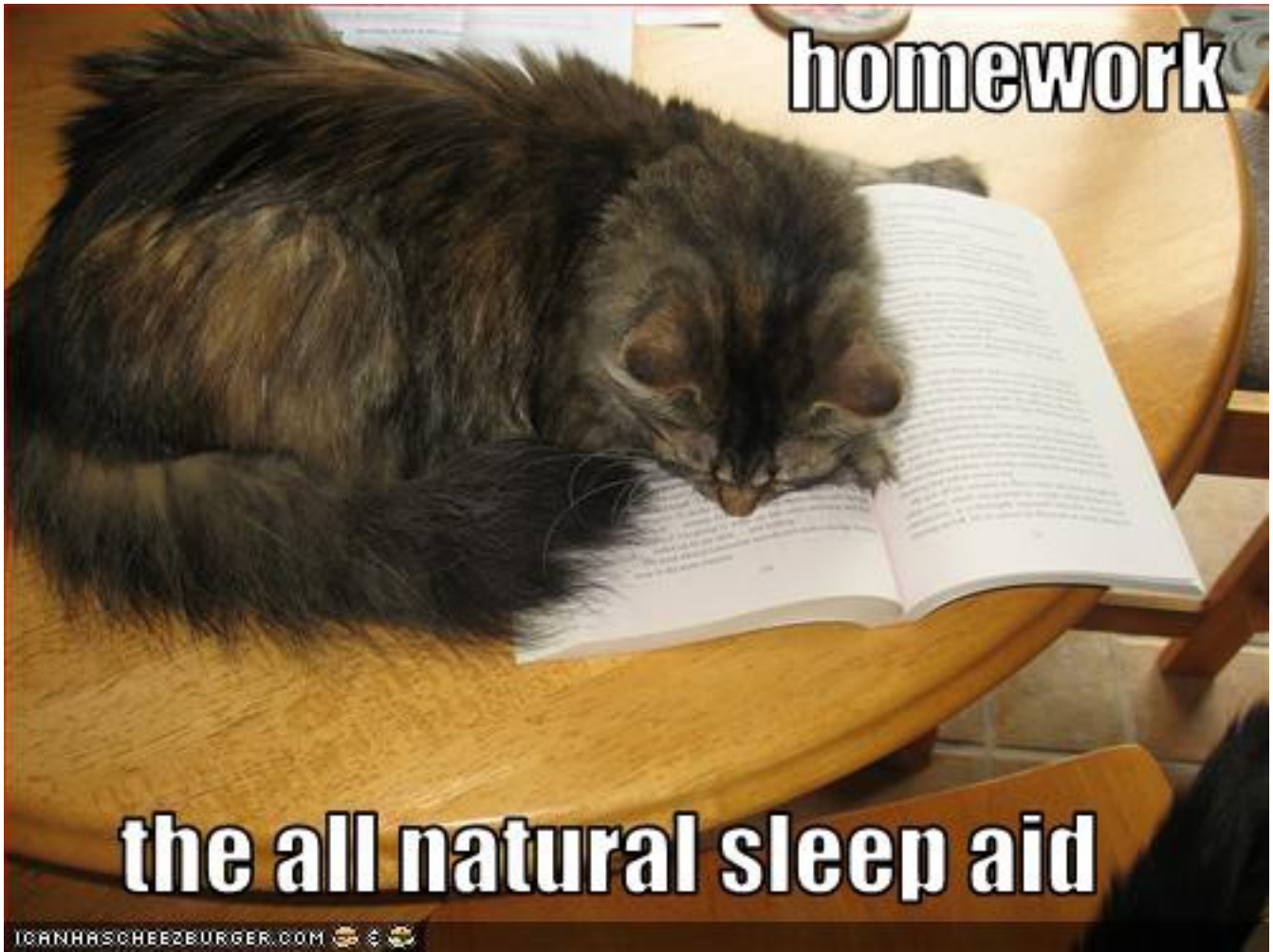
Live Submission Demo

- Let's visit the automated submission website



Start The Homework Now

- We can help you! (Also: video guides …)



Homework

- Scott Book reading (for Tuesday)
- Get started on PA1c (due in 7 days)

Questions?