Programming Languages Topic of Ultimate Mastery

Wes Weimer

EECS 590

http://web.eecs.umich.edu/~weimerw/590/

Reasonable Initial Skepticism

Today's Class

- Vague Historical Context
- Goals For This Course
- Requirements and Grading
- Course Summary

- Convince you that PL is useful

Meta-Level Information

- Please interrupt at any time!
- Completely cromulent queries:
  - I don't understand: please say it another way.
  - Slow down, you talk too fast!
  - Wait, I want to read that!
  - I didn't get joke X, please explain.

What Have You Done For Us Lately?

- Isn't PL a solved problem?
  - PL is an old field within Computer Science
  - 1920's: "computer" = "person"
  - 1936: Church's Lambda Calculus (= PL!)
  - 1937: Shannon's digital circuit design
  - 1940's: first digital computers
  - 1950's: FORTRAN (= PL!)
  - 1958: LISP (= PL!)
  - 1960's: Unix
  - 1972: C Programming Language
  - 1981: TCP/IP
  - 1985: Microsoft Windows
  - 1992: Ultima Underworld / Wolfenstein 3D

"… a prestigious line of work with a long and glorious tradition." - Vizzini

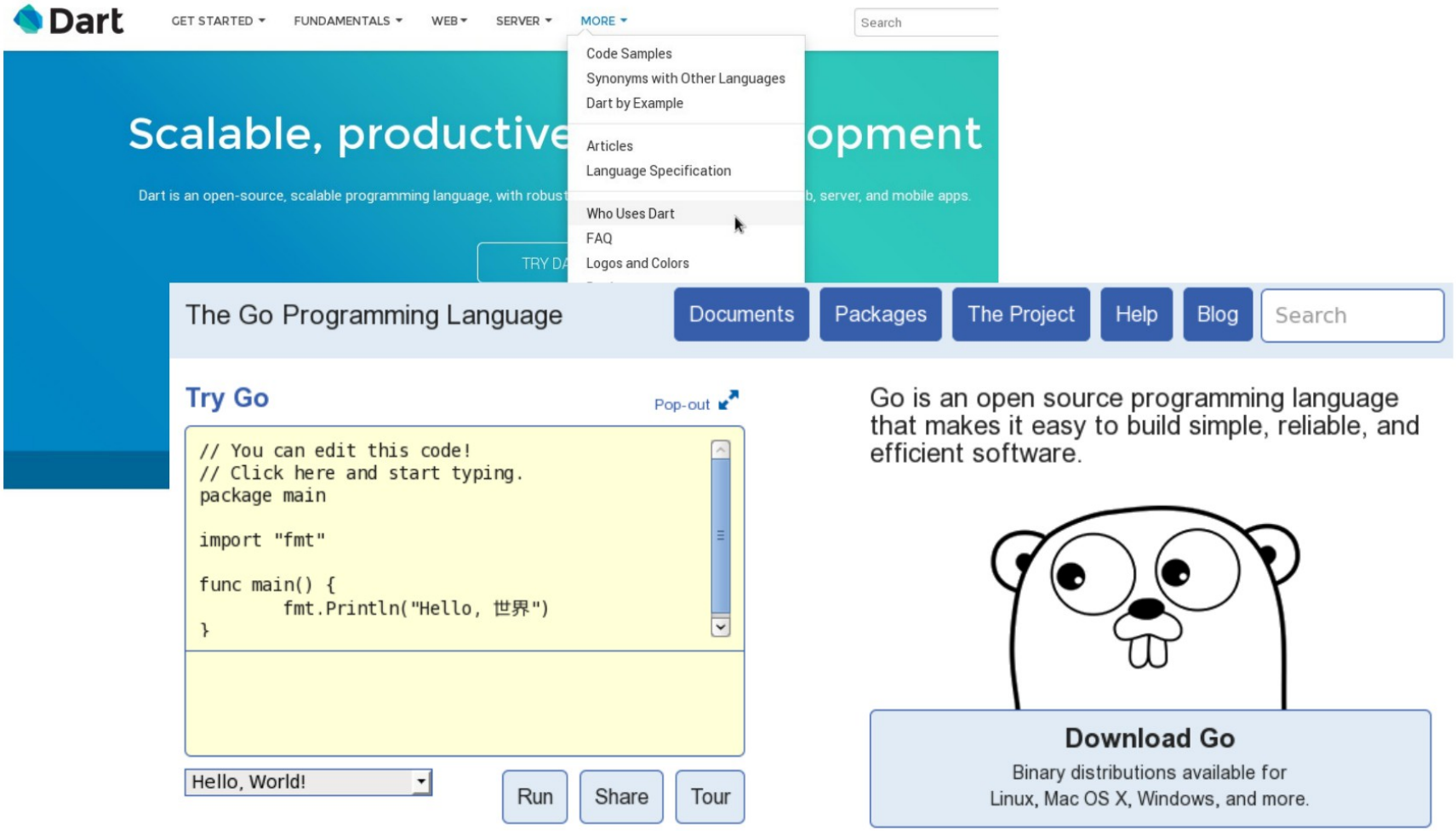
A Brief Tour

- … of PL research impact at companies

- Themes:
  - Multiple types of companies make languages
  - PL tools apply to many domains
  - PL research is embedded in other hardware and software
  - PL is interdisciplinary

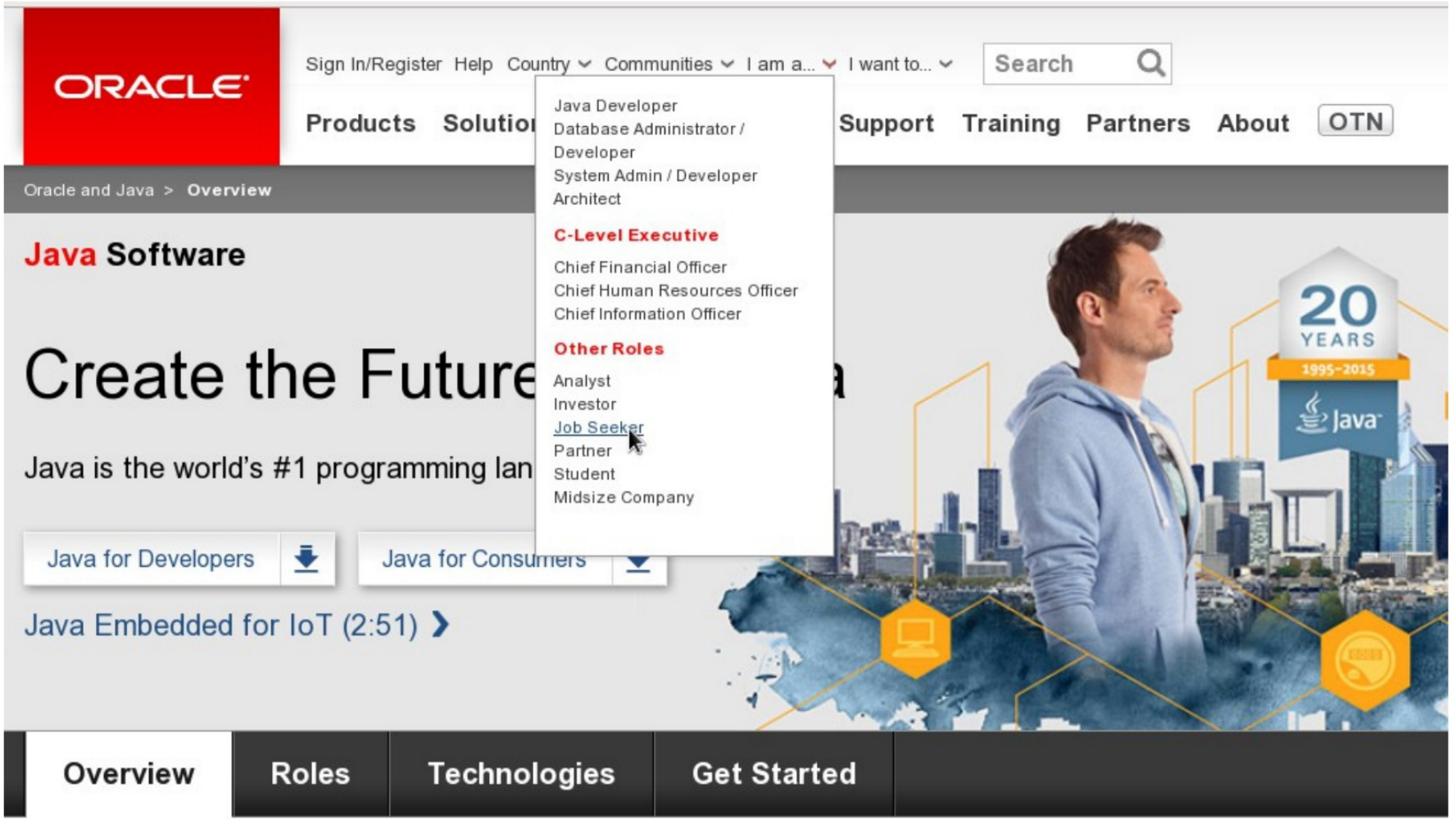
Google

- Go, Dart, etc.



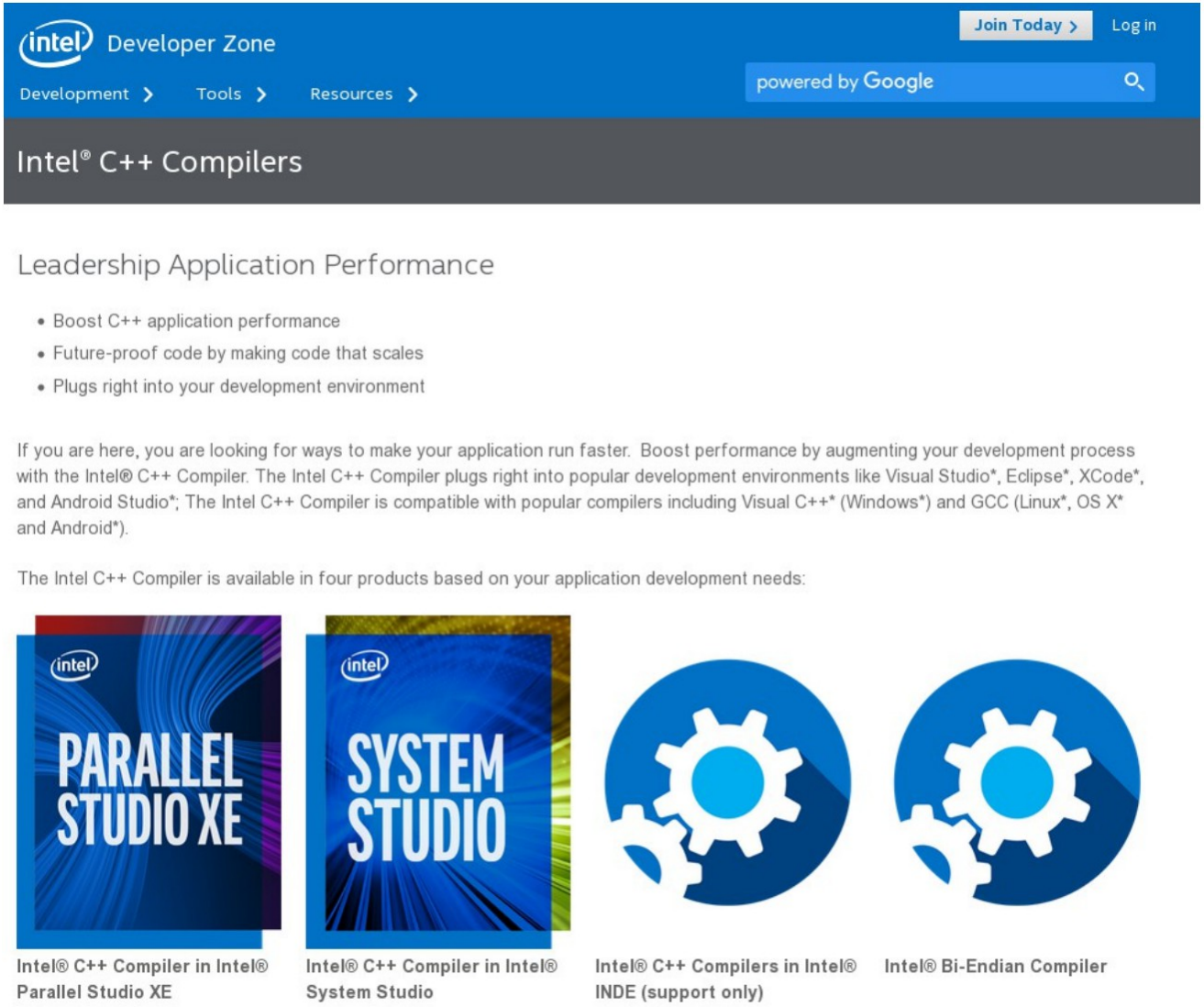
Oracle

- Java Compiler, Java Virtual Machine, …

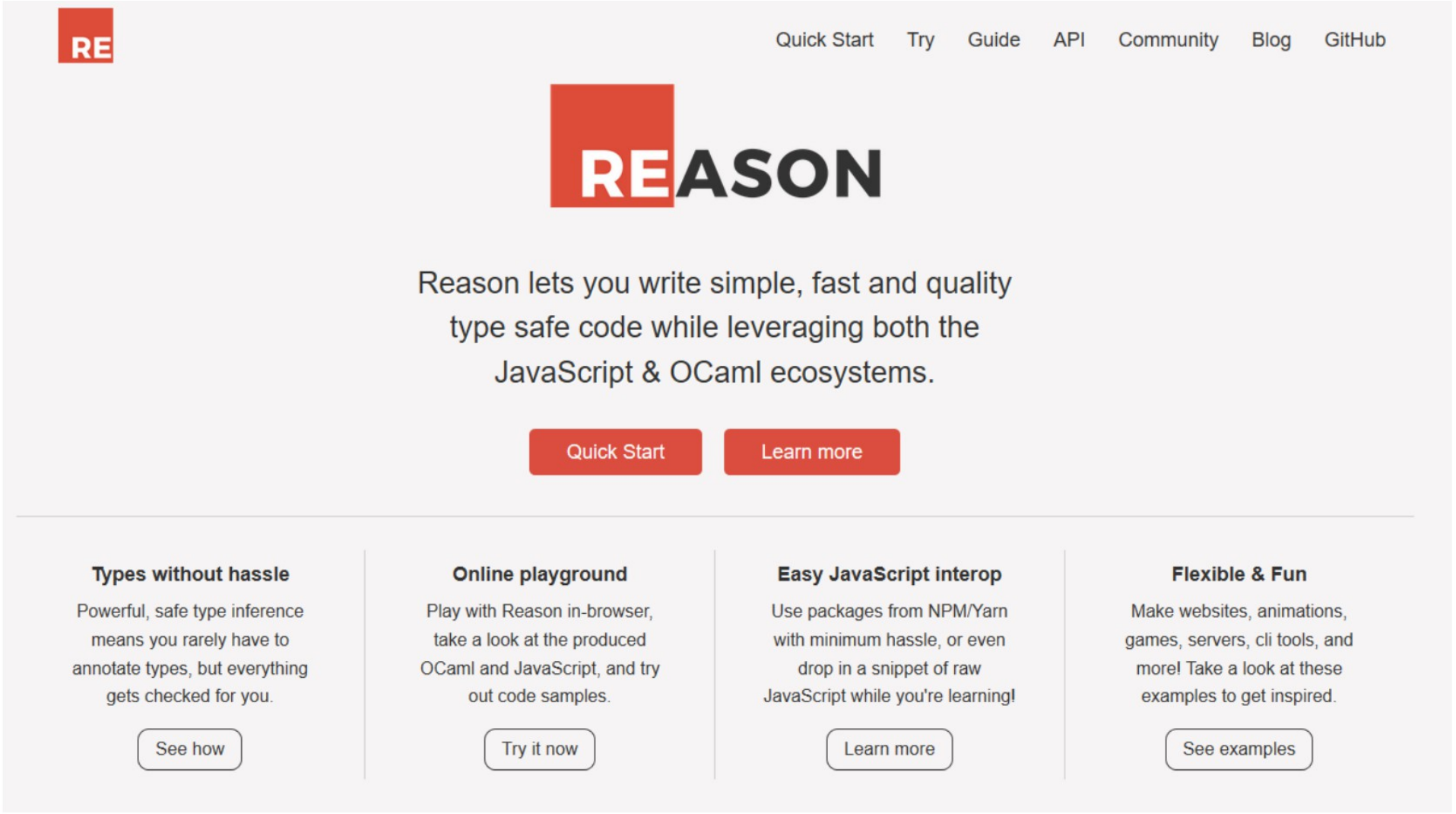


Intel

- ICC

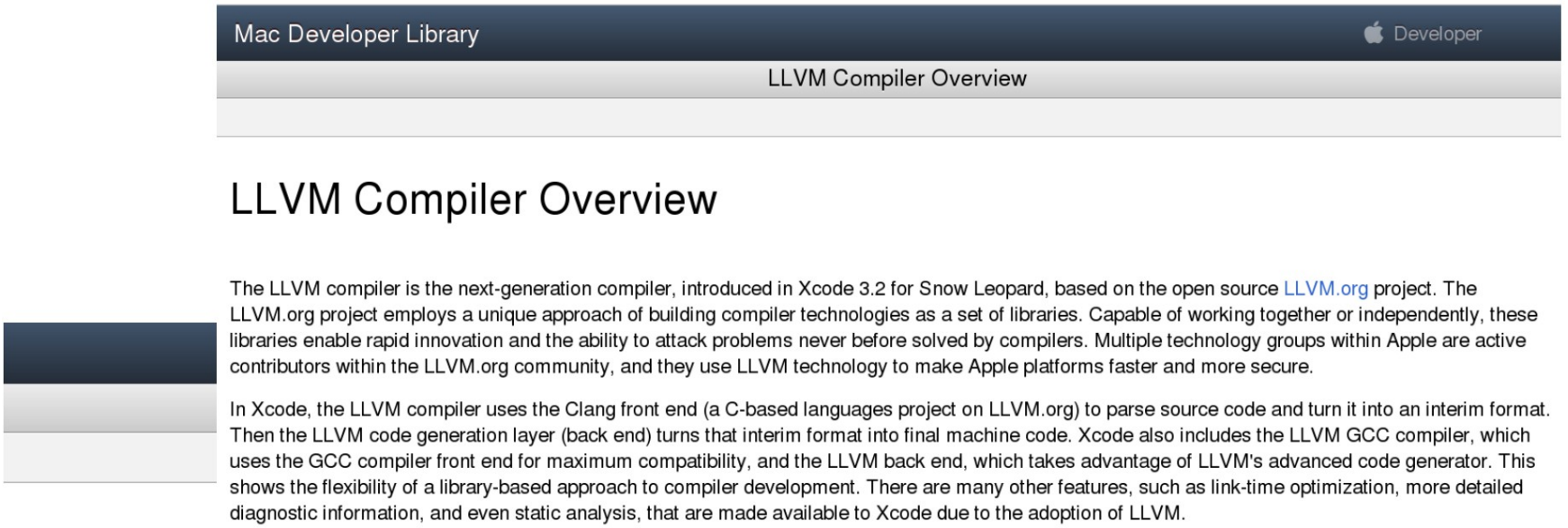

intel Developer Zone
Join Today
Log in
Development
Tools
Resources
powered by Google
Intel® C++ Compilers
Leadership Application Performance
Boost C++ application performance
Future-proof code by making code that scales
Plugs right into your development environment
If you are here, you are looking for 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:
PARALLEL STUDIO XE
SYSTEM STUDIO
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

Facebook


RE
Quick Start
Try
Guide
API
Community
Blog
GitHub
REASON
Reason lets you write simple, fast and quality type safe code while leveraging both the JavaScript & OCaml ecosystems.
Quick Start
Learn more
Types without hassle
Powerful, safe type inference means you rarely have to annotate types, but everything gets checked for you.
See how
Online playground
Play with Reason in-browser, take a look at the produced OCaml and JavaScript, and try out code samples.
Try it now
Easy JavaScript interop
Use packages from NPM/Yarn with minimum hassle, or even drop in a snippet of raw JavaScript while you're learning!
Learn more
Flexible & Fun
Make websites, animations, games, servers, cli tools, and more! Take a look at these examples to get inspired.
See examples

Apple

- LLVM. Objective-C (iOS, etc.).



Microsoft FlashFill

Our programming by example work (POPL 2011), also recognized as CACM Research Highlights (CACM 2012), ships as part of the Flash Fill feature in Excel in Office 2013. Here's a small video illustrating this feature. Here's another small video illustrating potential extensions.

Here's the inside story of how it came about: Flash Fill Gives Excel a Smart Charge

Here are some other videos on FlashFill

- You-tube: Excel 2013 Flash Fill: 23 Amazing Examples, Excel 2013- Flash Fill, Meet new Excel's Flash Fill, Dutch video, French Video, German video, Japanese video, Polish video, Romanian video, Urdu video, Musical
- Microsoft: Rick Rashid on FlashFill (in conversation with John Markoff of New York Times), Office Blog, Customer Preview Video (See the video segment from 0:35-0:40), Peter Lee on FlashFill (in his Keynote Speech on the 14th Computing in the 21st Century Conference – See the video segment from 22:22-27:20)
- CNet: Microsoft gives new Office a Windows 8 look (This video is at the bottom of the page. See the video segment from 2:00-3:01)

Here is what popular media says about this feature

- PC Magazine: My favorite new feature, because it saves a tremendous amount of time-wasting effort, is called Flash Fill, and it's one of many features where Excel acts as it it's using its brain, not just its raw number-crunching power. With some experimentation, you may find that Flash Fill is smarter than you expect.

People

Sumit Gulwani
Partner Research Manager

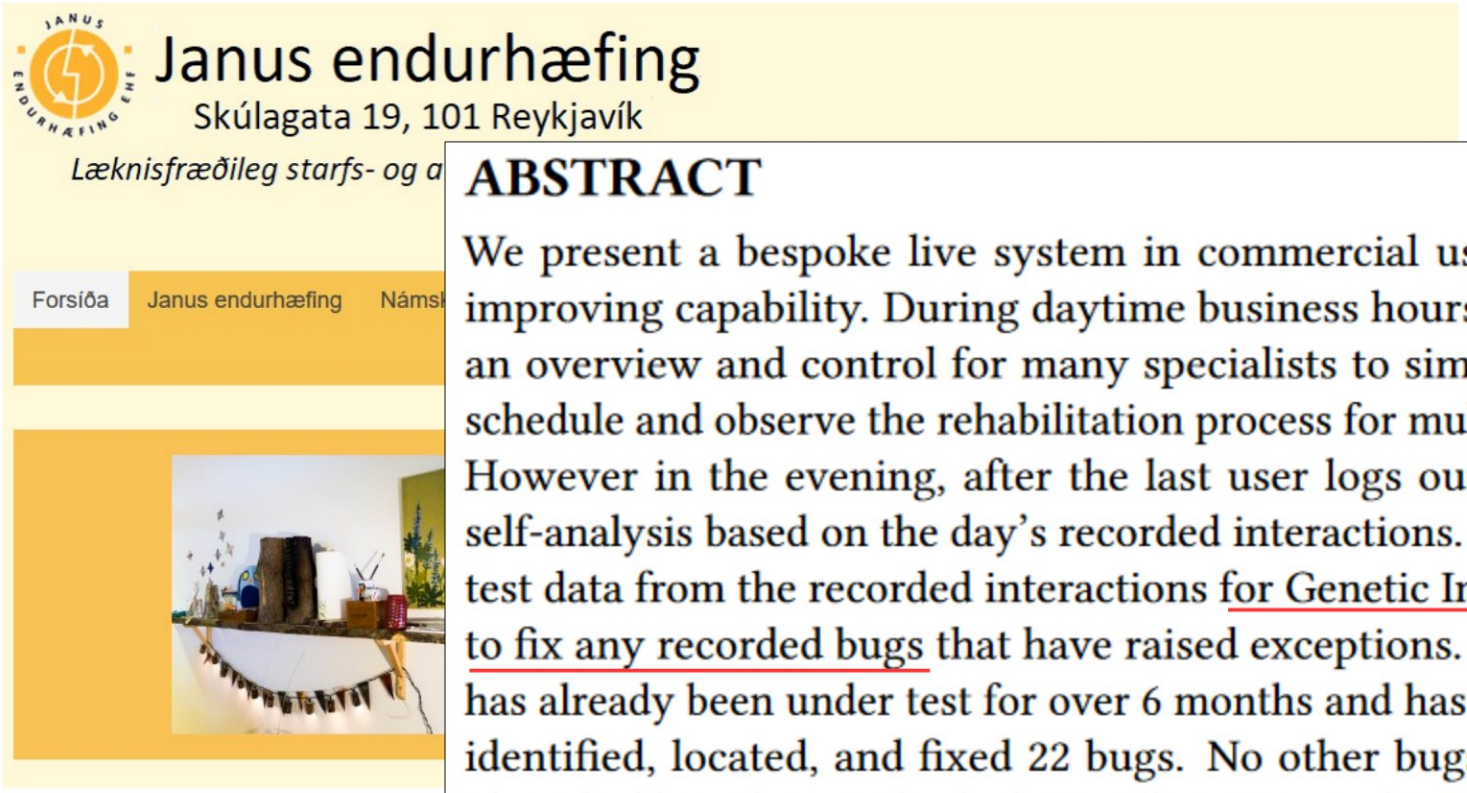
	A	B
1	SSN	Column2
2	542368978	542-36-8978
3	123145542	
4	121247543	
5	454545465	
6	642548745	
7	514852145	
8	152358834	
9	642974682	

Ctrl-E

	A	B
1	SSN	Column2
2	542368978	542-36-8978
3	123145542	123-14-5542
4	121247543	121-24-7543
5	454545465	454-54-5465
6	642548745	642-54-8745
7	514852145	514-85-2145
8	152358834	152-35-8834
9	642974682	642-97-4682

Janus Manager

"Fixing Bugs in Your Sleep: How Genetic Improvement Became an Overnight Success"



ABSTRACT

We present a bespoke live system in commercial use with self-improving capability. During daytime business hours it provides an overview and control for many specialists to simultaneously schedule and observe the rehabilitation process for multiple clients. However in the evening, after the last user logs out, it starts a self-analysis based on the day's recorded interactions. It generates test data from the recorded interactions for Genetic Improvement to fix any recorded bugs that have raised exceptions. The system has already been under test for over 6 months and has in that time identified, located, and fixed 22 bugs. No other bugs have been identified by other methods during that time. It demonstrates the effectiveness of simple test data generation and the ability of GI for improving live code.

DARPA Cyber Grand Challenge

The ultimate test of wits in computer security occurs through open competition on the global Capture the Flag (CTF) tournament circuit. In CTF contests, experts reverse-engineer software, probe its weaknesses, search for deeply hidden flaws and create securely patched replacements.

What if a purpose-built computer systems could compete against the CTF circuit's greatest experts? DARPA has modeled the Cyber Grand Challenge on today's CTF tournaments to pave the way toward that future.

On August 4th, 2016, DARPA will hold the world's first all-computer Capture the Flag tournament in Las Vegas. Seven prototype systems will square off against each other, competing for nearly $4 million in prizes in a live network competition. The CGC Final Event will take place in conjunction with DEF CON, home of the longest-running annual CTF competition.

To learn more about the Cyber Grand Challenge, explore this site and visit DARPA's official CGC homepage, CGC Final Event announcement and CGC news articles.

DARPA Cyber Grand Challenge

The ultimate test of wits in computer security occurs through
Capture the Flag (CTF) tournament circuit. In CTF contests, ex
probe its weaknesses, search for deeply hidden flaws and crea

What if a purpose-built computer systems could compete agai
experts? DARPA has modeled the Cyber Grand Challenge on t
way toward that future.

On August 4th, 2016, DARPA will hold the world's first all-com
in Las Vegas. Seven prototype systems will square off against
million in prizes in a live network competition. The CGC Final
with DEF CON, home of the longest-running annual CTF comp

To learn more about the Cyber Grand Challenge, explore this s
homepage, CGC Final Event announcement and CGC news arti

Abstract

The automatic exploit generation challenge is given a program, automatically find vulnerabilities and generate exploits for them. In this paper we present AEG, the first end-to-end system for fully automatic exploit generation. We used AEG to analyze 14 open-source projects and successfully generated 16 control flow hijacking exploits. Two of the generated exploits (expect-5.43 and htget-0.93) are zero-day exploits against unknown vulnerabilities. Our contributions are: 1) we show how exploit generation for control flow hijack attacks can be modeled as a formal verification problem, 2) we propose preconditioned symbolic execution, a novel technique for targeting symbolic execution, 3) we present a general approach for generating working exploits once a bug is found, and 4) we build the first end-to-end system that automatically finds vulnerabilities and generates exploits that produce a shell.

DARPA Cyber Grand Challenge

AEG searches for bugs at the source code level by exploring execution paths. Specifically, AEG executes `iwconfig` using symbolic arguments

AEG performs dynamic analysis on the `iwconfig` binary using the concrete input generated in step 2.

AEG generates the constraints describing the exploit using the runtime information generated

Abstract

ploit generation challenge is given
cally find vulnerabilities and gener-
. In this paper we present AEG, the
m for fully automatic exploit gener-
to analyze 14 open-source projects
and successfully generated 16 control flow hijacking ex-
generated exploits (expect-5.43 and
-day exploits against unknown vul-
ontributions are: 1) we show how
r control flow hijack attacks can be
modeled as a formal verification problem, 2) we pro-
d symbolic execution, a novel tech-
ymbolic execution, 3) we present a
r generating working exploits once
4) we build the first end-to-end sys-
tem that automatically finds vulnerabilities and gener-
ates exploits that produce a shell.

Microsoft

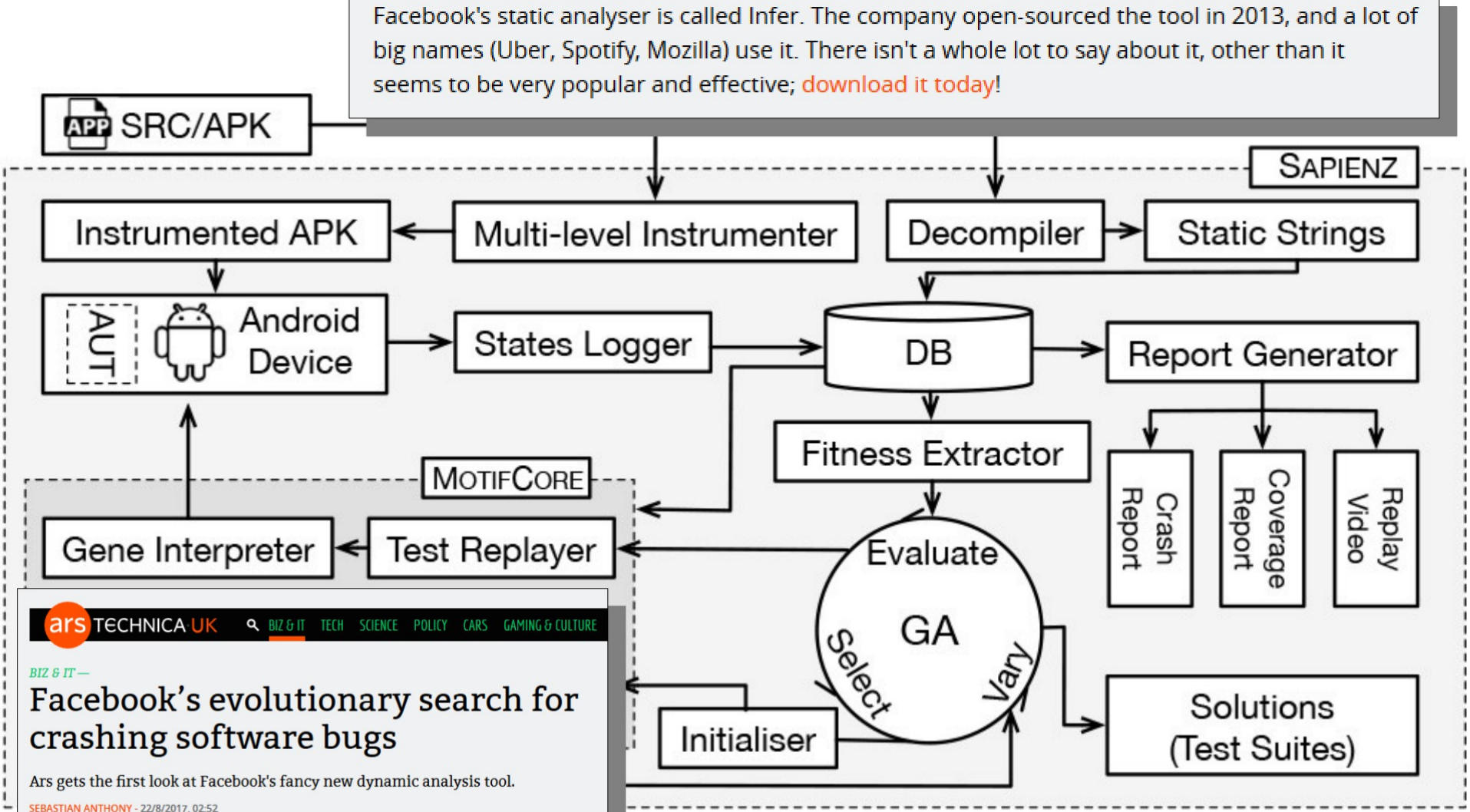
- Software, Languages, Analysis and Model Checking

SLAM is a project for checking that software satisfies critical behavioral properties of the interfaces it uses and to aid software engineers in designing interfaces and software that ensure reliable and correct functioning. Static Driver Verifier is a tool in the Windows Driver Development Kit that uses the SLAM verification engine.

"Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability."
Bill Gates, April 18, 2002. Keynote address at WinHec 2002

Facebook: Infer and Sapienz

Facebook's static analyser is called Infer. The company open-sourced the tool in 2013, and a lot of big names (Uber, Spotify, Mozilla) use it. There isn't a whole lot to say about it, other than it seems to be very popular and effective; download it today!



ars TECHNICA UK BIZ & IT TECH SCIENCE POLICY CARS GAMING & CULTURE

BIZ & IT —

Facebook's evolutionary search for crashing software bugs

Ars gets the first look at Facebook's fancy new dynamic analysis tool.

SEBASTIAN ANTHONY - 22/8/2017, 02:52

Figure 1: Sapienz workflow.

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:

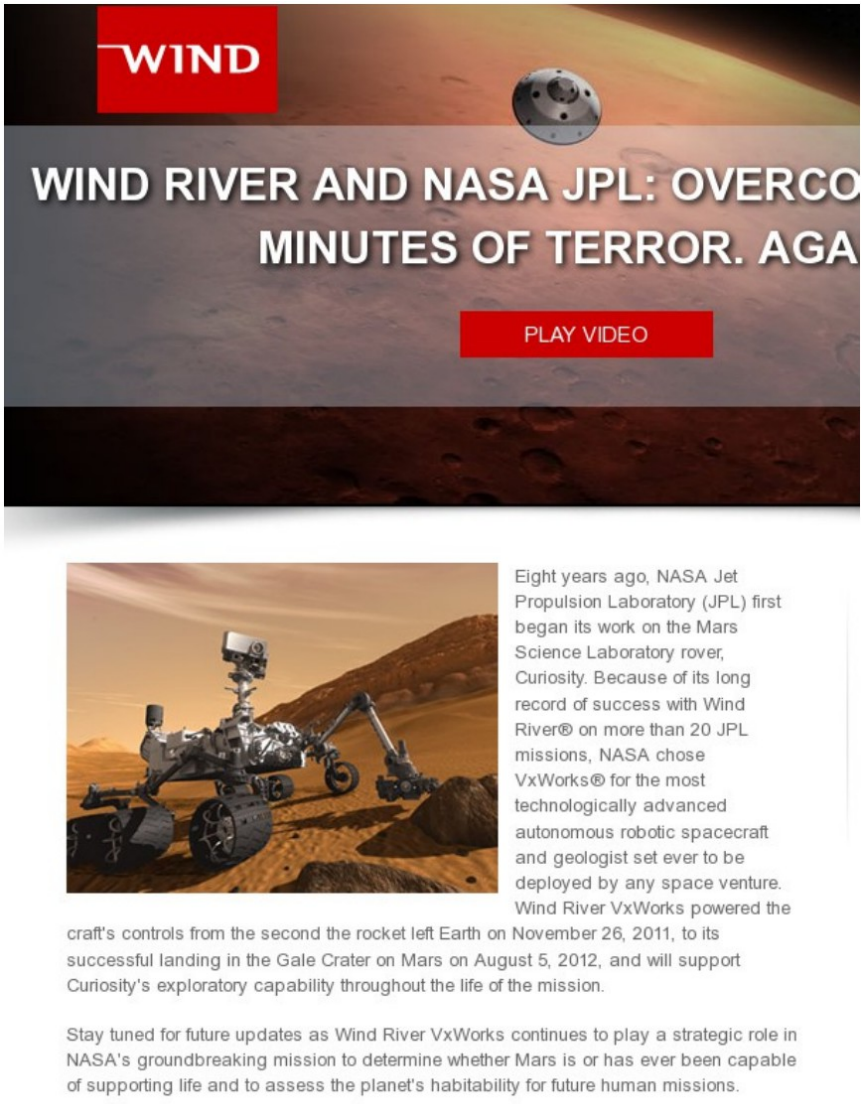
A software engineer has complete engineering responsibility for one or more major components of the Green Hills product line. For an experienced programmer this is a satisfying position in which you have personal responsibility for creating a tool used by thousands of programmers around the world. Our 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, Switch Game Console, ...



The Astrée Static Analyzer

- Astrée was able to prove, completely automatically, the absence of any RTE in a C version of the automatic docking software of the Jules Vernes Automated Transfer Vehicle (ATV) enabling ESA to transport payloads to the International Space Station.

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*.

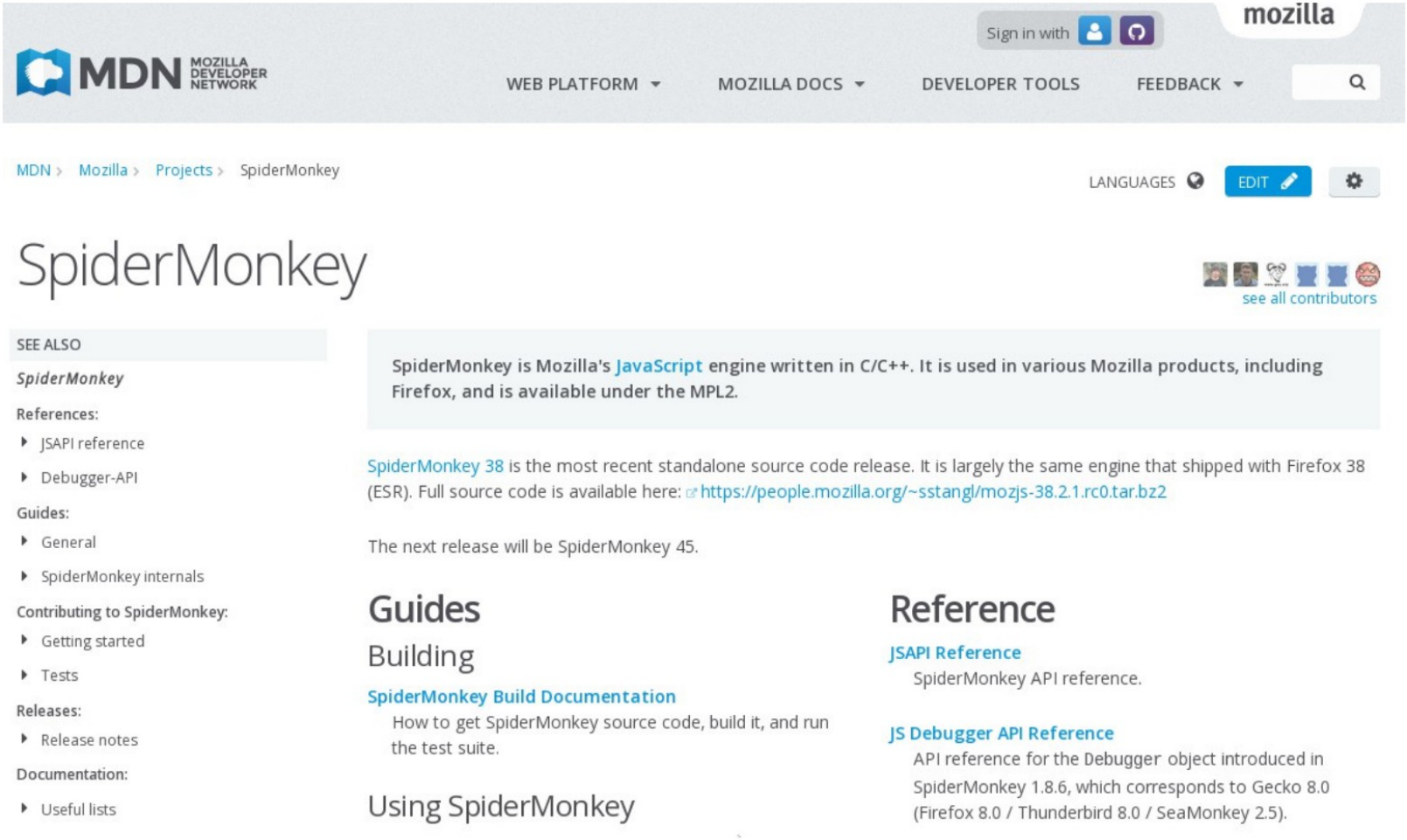
NOTE: You can also view information about the Photoshop objects and commands through the object browsers for each of the three scripting languages. See "Viewing Photoshop Objects, Commands, and Methods" on page 21.

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



Epic 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

- Overview
- Limitations

Surprise: Flash, Postscript

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 those found in Lisp, scoped memory and, since language le
Polish notation, which makes the order of operations unamb
one has to keep the layout of the stack in mind. Most *operat*
the stack, and place their results onto the stack. *Literals* (for
on the stack. Sophisticated data structures can be built on th
system, which sees them all only as arrays and dictionaries;
"types" is left to the code that implements them.

ActionScript

ActionScript is an object-oriented programming language originally developed by Macromedia Inc. (now dissolved into Adobe Systems). It is a derivation of HyperTalk, the scripting language for HyperCard.[2] It is now a dialect of ECMAScript (meaning it is a superset of the syntax and semantics of the language more widely known as JavaScript), though it originally arose as a sibling, both being influenced by HyperTalk.

ActionScript is used primarily for the development of websites and software targeting the Adobe Flash Player platform, used on Web pages in the form of embedded SWF files.

ActionScript 3 is also used with Adobe AIR system for the development of desktop and mobile applications. The language itself is open-source in that its specification is offered free of charge[3] and both an open source compiler (as part of Apache Flex) and open source virtual machine (Mozilla Tamarin) are available.

ActionScript is also used with Scaleform GFx for the development of 3D video game user interfaces and HUDs.

ActionScript

Paradigm	Multi-paradigm: object-oriented (prototype-based), functional, imperative, scripting
Designed by	Gary Grossman
Developer	Macromedia (now dissolved into Adobe Systems)
First appeared	1998
Stable release	3.0 / June 27, 2006
Typing discipline	strong, static
Website	adobe
Major implementations	
Adobe Flash Player, Adobe AIR, Apache Flex, Scaleform GFx	
Influenced by	
JavaScript, Java	

Surprise: Flash, Postscript

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 those found in Lisp, scoped memory and, since language le
Polish nota
one has to
the stack, a
on the stac
system, wh
"types" is l

ActionScript

ActionScript is an object-oriented programming language originally developed by Macromedia Inc. (now dissolved into Adobe Systems). It is a

ActionScript is also used with Scaleform GFx for the development of 3D video game user interfaces and HUDs.

ActionScript

	ligm: object-oriented based), functional, scripting
	sman
	ia (now dissolved Systems)
	27, 2006
Typing discipline	strong, static
Website	adobe
Major implementations	
Adobe Flash Player, Adobe AIR, Apache Flex, Scaleform GFx	
Influenced by	
JavaScript, Java	

Flash Player, Your Printer,
Your Cell Phone, Acrobat Reader:
they all contain *Interpreters*.

Wait …

- But weren't most of those examples mixtures of PL and some other discipline?
  - Mars Rover, Intel = PL + Hardware
  - Flash Fill = PL + Machine Learning
  - Cyber Grand Challenge = PL + Security
  - Gaming Languages, PDF = PL + Graphics
  - GenProg, Sapienz = PL + Evolutionary
  - SLAM = PL + Model Checking
- Yes! That's the point!

Parts of Computer Science

- CS = (Math $\times$ Logic) + Engineering
  - Science (from Latin *scientia* - knowledge) refers to a system of acquiring knowledge - based on empiricism, experimentation, and methodological naturalism - aimed at finding out the truth.
- We rarely actually do this in CS
  - "CS theory" = Math (logic)
  - "Systems" = Engineering (bridge building)

Programming Languages

- Best of both worlds: Theory and Practice!
  - Only pure CS theory is more primal
- Touches most other CS areas
  - Theory: DFAs, PDAs, TMs, language theory (e.g., LALR)
  - Systems: system calls, assembler, memory management
  - Arch: compiler targets, optimizations, stack frames
  - Numerics: FORTRAN, IEEE FP, Matlab, loop nest optim.
  - AI: theorem proving, machine learning, search
  - DB: SQL, persistent objects, modern linkers
  - Networking: packet filters, protocols, even Ruby on Rails
  - Graphics: OpenGL, LaTeX, PostScript, even Logo (= LISP)
  - Security: buffer overruns, .net, bytecode, PCC, …
  - Software Engineering: bug finding, refactoring, types, ...

Overarching Theme

- I assert (and shall convince you) that
- PL is one of the most vibrant and active areas of CS research today
  - It has theoretical and practical meatiness
  - It intersects most other CS areas
- You will be able to use PL techniques in your own projects

Goal #1

- Learn to **use** advanced PL techniques

Useful Complex Knowledge

- A proof of the fundamental theorem of calculus
- A proof of the max-flow min-cut theorem
- Nifty Tree node insertion (e.g., B-Trees, AVL, Red-Black)
- The code for the Fast Fourier Transform
- And so on …

No Useless Memorization

- I will not waste your time with useless memorization
- This course will cover complex subjects
- I will teach their details to help you understand them the first time
- But you will never have to memorize anything low-level
- Rather, learn to apply broad concepts

Goal #2

- When (not if) you design a language, it will avoid the mistakes of the past and you'll be able to describe it formally

Story: The Clash of Two Features

- Real story about bad programming language design
- Cast includes famous scientists
- ML (’82) is a functional language with polymorphism and monomorphic references (i.e. pointers)
- Standard ML (’85) innovates by adding polymorphic reference
- It took 10 years to fix the “innovation”

Polymorphism (Informal)

- Code that works uniformly on various types of data
- Examples of function signatures:

  length : α list $\rightarrow$ int (takes an argument of type "list of α", returns an integer, for any type α)

  head : α list $\rightarrow \alpha$

- Type inference:
  - generalize all elements of the input type that are not used by the computation

References in Standard ML

- Like "updatable pointers" in C
- Type constructor: ptr τ
  - x : ptr int === "x is a pointer to an integer"
- Expressions:

  alloc : $\tau \rightarrow$ ptr τ (allocate a cell to store a τ)

  *e : τ when e : ptr τ (read through a pointer)

  *e1 := e2 with e1 : ptr τ and e2 : τ

  (write through a pointer)
- Works just as you might expect ...

Polymorphic References: A Major Pain

Consider the following program fragment:

Code

```
fun id(x) = x
val c = alloc id
print (*c) (“hi”)
print (*c) (5)
```

Polymorphic References: A Major Pain

Consider the following program fragment:

Code

```
fun id(x) = x
val c = alloc id
fun inc(x) = x + 1
*c := inc
print (*c) (6)
```

Polymorphic References: A Major Pain

Consider the following program fragment:

Code

```
fun id(x) = x
val c = alloc id
fun inc(x) = x + 1
*c := inc
(*c) (“hi”)
```

Polymorphic References: A Major Pain

Consider the following program fragment:

Code	Type inference
fun id(x) = x	id : $\alpha \rightarrow \alpha$ (for any α)
val c = alloc id	c : ptr ($\alpha \rightarrow \alpha$) (for any α)
fun inc(x) = x + 1	inc : int $\rightarrow$ int
*c := inc	Ok, since c : ptr (int $\rightarrow$ int)
(*c) (“hi”)	Ok, c : ptr (string $\rightarrow$ string)

Reconciling Polymorphism and References

- Type system fails to prevent a type error!
- Common solution:
  - value restriction: generalize only the type of values!
    - easy to use, simple proof of soundness
- X Features $\Rightarrow$ X^2 Complication
- To see what went wrong we needed to understand semantics, type systems, polymorphism and references

Story 2: Java Bytecode Subroutines

- Java bytecode programs contain subroutines (jsr) that run in the caller's stack frame *(why?)*
- jsr complicates the formal semantics of bytecodes
  - Several verifier bugs were in code implementing jsr
  - 30% of typing rules, 50% of soundness proof due to jsr
- It is not worth it:
  - In 650K lines of Java code, 230 subroutines, saving 2427 bytes, or 0.02%
  - 13 times more space could be saved by renaming the language back to Oak
    - [In 1994], the language was renamed "**Java**" after a trademark search revealed that the name "Oak" was used by a manufacturer of video adapter cards.

Recall Goal #2

- When (not if) you design a language, it will avoid the mistakes of the past and you'll be able to describe it formally

Goal #3

- Understand current PL research (PLDI, POPL, OOPSLA, TOPLAS, …) and technology transfer (MS, Intel, …)

Final Goal: Fun

Q: Books (730 / 842)

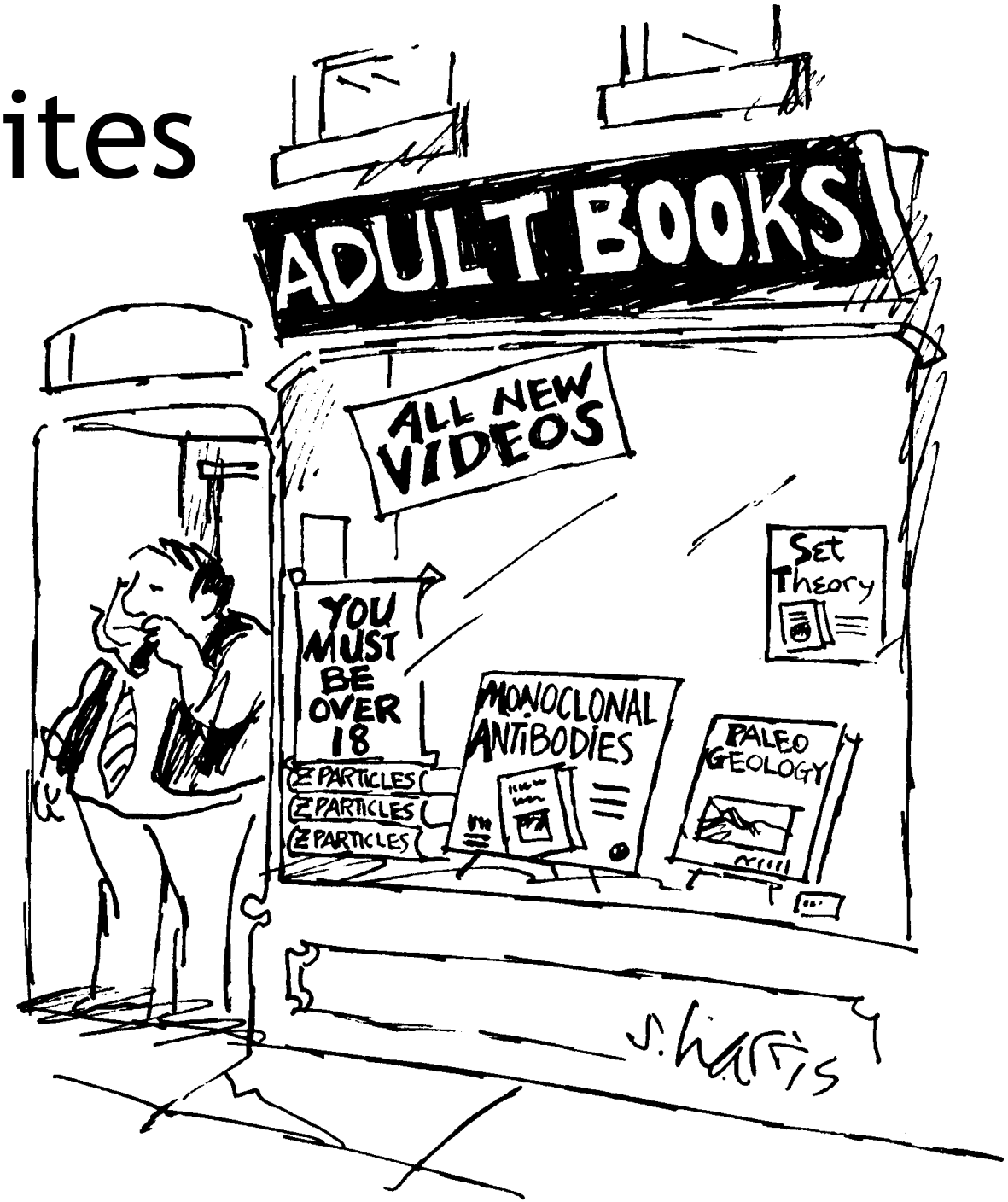
- This 1960 Daniel Keyes sci-fi novel is told as a *"progris riport"* from the point-of-view of Charlie Gordon as he takes an experimental intelligence-enhancing treatment. The treatment is temporary. The book won the Hugo and Nebula awards.

Q: Computer Science

- This Sri Lanka-born, British computer scientist is best known for his development of *QuickSort*, a logic for verifying program correctness, the *monitor* approach to mutual exclusion, and the formalism of *Communicating Sequential Processes*. In 2009 he apologized for inventing the *null reference*:
  - I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.

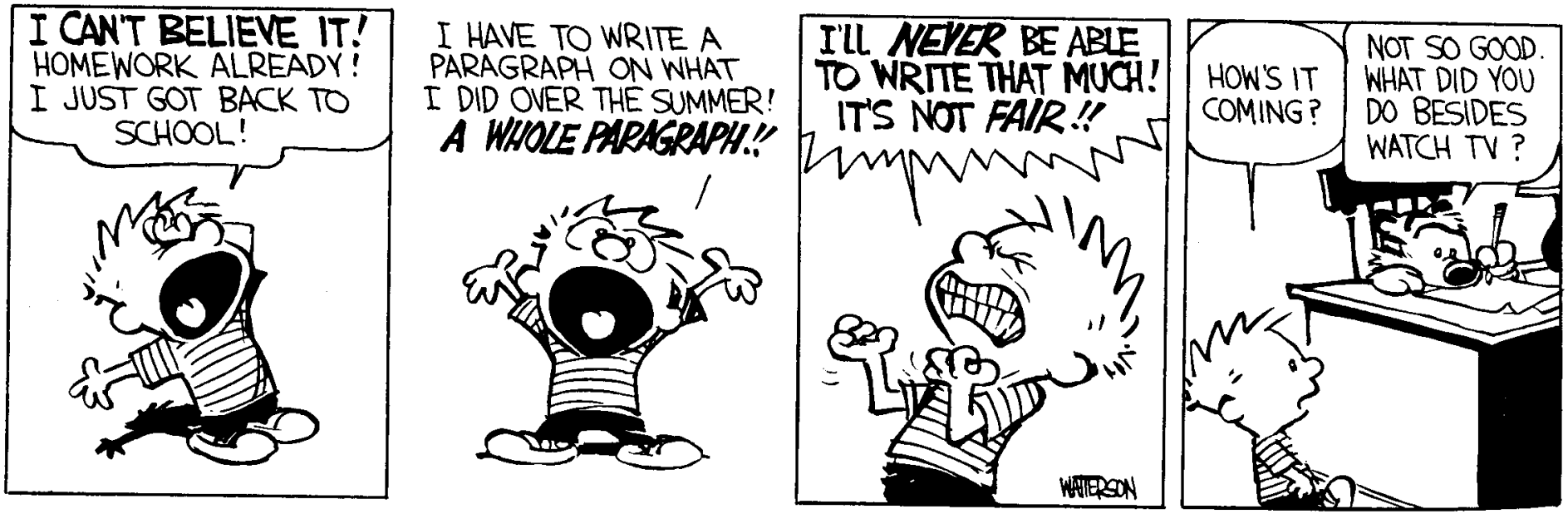
Prerequisites

- Undergraduate PL/compilers course?
  - **No**
- "Mathematical maturity"



Assignments

- Short Homework Assignments (5)
- Long Homework Assignment (1)
- Daily Reading (1-2 papers per class)
- **Final Project**



Homework Problem Sets

- Some material can be “mathy”
- Much like Calculus, practice is handy
- Short: ~3 theory + 1 coding per HW
- You have one week to do each one
  - Usually available in advance ...
- Long: analysis of real C programs
- NB: I will offer suggestions and comments on your English prose.

Final Project

- Literature survey, implementation project, or research project
- Write a 5-page paper (akin to PLDI)
- Give a ~10 minute presentation
- On the topic of your choice
  - I will help you find a topic (many examples)
  - Best: integrate PL with your current research

Reading Quizzes: Let's Vote!

- A key problem:
  - If I never check, graduate students will not do the reading.
- A related desire:
  - Graduate students often wish that someone would make them do the reading.
- Proposal:
  - “One Word” reading quizzes

Piazza Forum

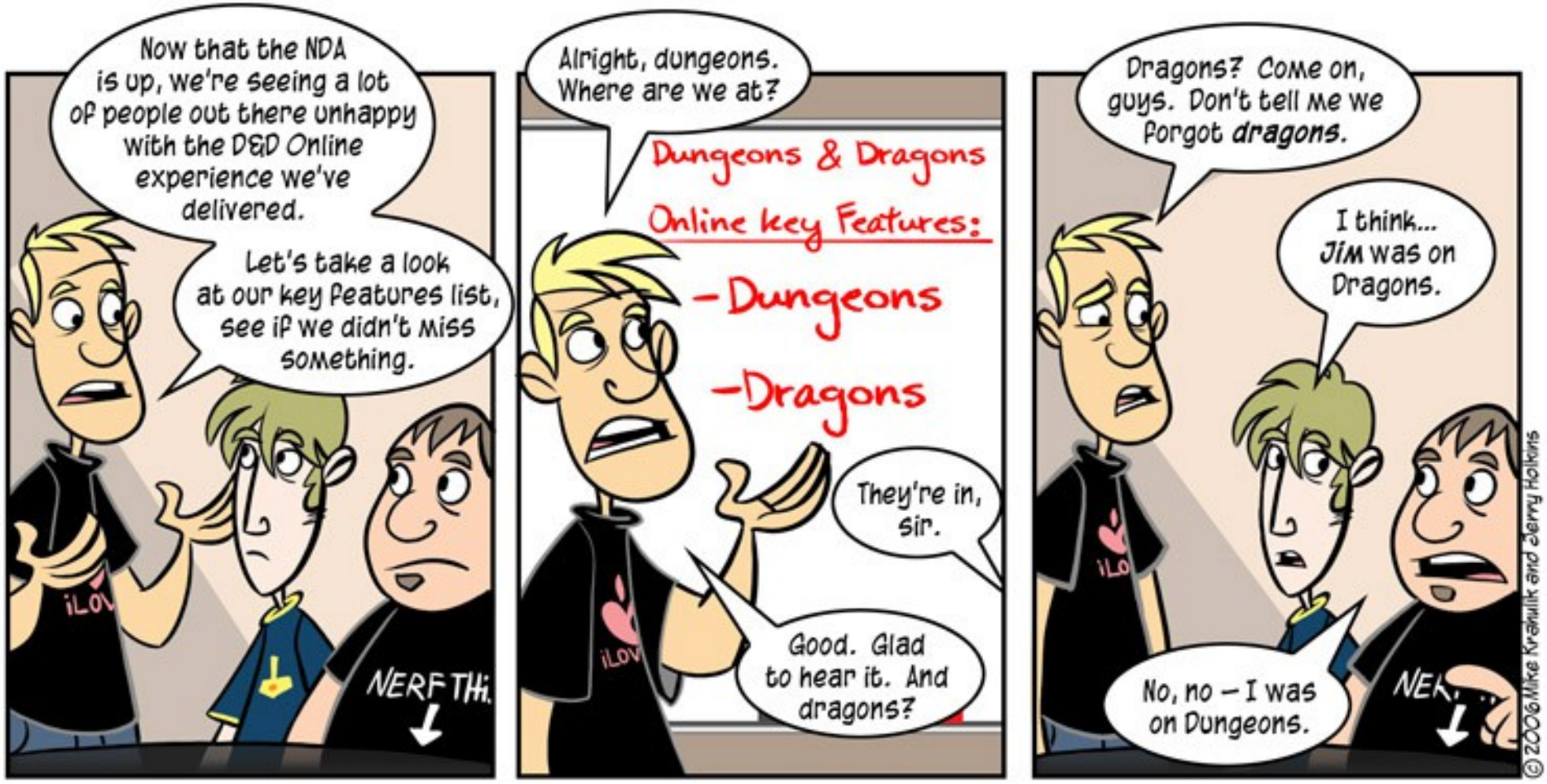
- Ask questions about assignments
- Compare concerns about papers
- Ask for more details about references from class
- Previous years: fun papers, memes, moral quandaries, etc.

How Hard Is This Class?

This Shall Be Avoided

In 1930, the Republican-controlled House of Representatives, in an effort to alleviate the effects of the... Anyone? Anyone? ... the Great Depression, passed the ... Anyone? Anyone? The tariff bill? The Hawley-Smoot Tariff Act? Which, anyone? Raised or lowered? ... raised tariffs, in an effort to collect more revenue for the federal government. Did it work? Anyone? Anyone know the effects?

Key Features of PL



Programs and Languages

- Programs
  - What are they trying to do?
  - Are they doing it?
  - Are they making some other mistake?
  - Were they hard to write?
  - Could we make it easier?
  - Should you run them?
  - How should you run them?
  - How can I run them faster?

Programs and Languages

- Languages
  - Why are they annoying?
  - How could we make them better?
  - What tasks can they make easier?
  - What cool features might we add?
  - Can we stop mistakes before they happen?
  - Do we need new paradigms?
  - How can we help out My Favorite Domain?

Common PL Research Tasks

- Design a new language feature
- Design a new type system / checker
- Design a new program analysis
- Find bugs in programs
- (Help people to) Fix bugs in programs
- Transform programs (source or assembly)
- Interpret and execute programs
- Prove things about programs
- Optimize programs

Grand Unified Theory

- Design a new type system
- Your type-checker becomes a bug-finder
  - No type errors $\Rightarrow$ proof that program is safe
  - Type error $\Rightarrow$ bug may exist in program
    - Fault localization and automated program repair
- Design a new language feature
  - To prevent the sort of mistakes you found
- Write a source-to-source transform
  - Your new feature now works on existing code

EECS 590 - Core Topics

- Model Checking
- Operational semantics
- Provers and proofs
- Type theory
- Verification conditions
- Symbolic Execution
- Invariant Detection
- Abstract interpretation
- Machine Learning
- Lambda Calculus

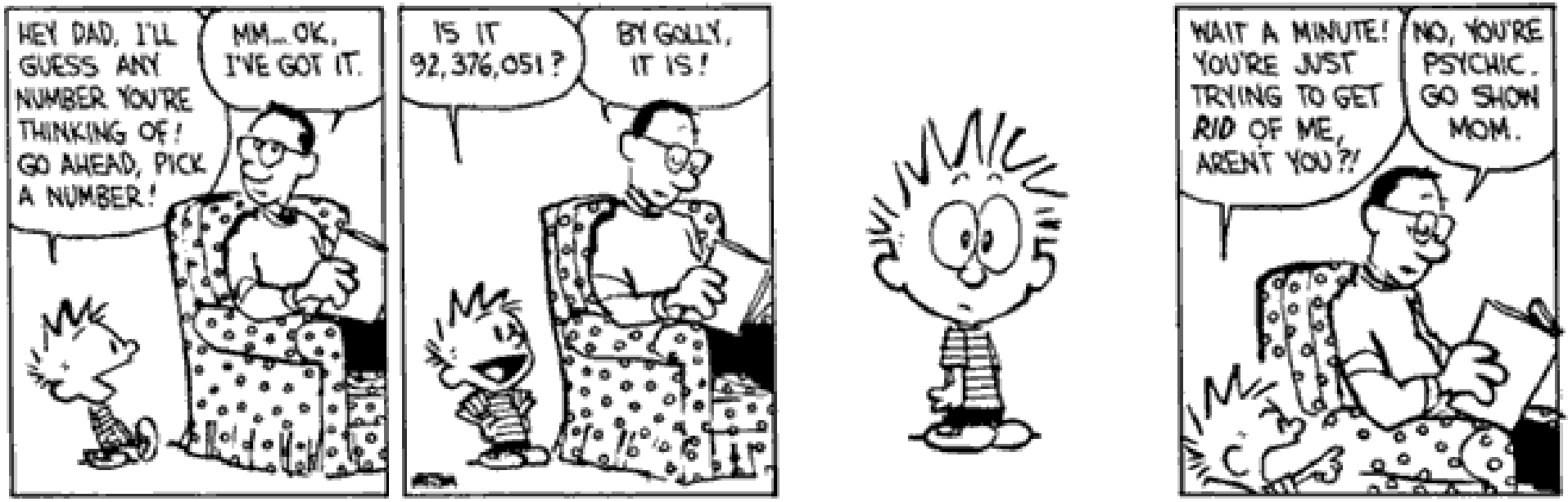
Special Topics

- Communications and Concurrency
- Fault Localization, Bug Isolation
- Automated Program Repair

- What do you want to hear about?

First Topic: Model Checking

- Verify critical properties of software or find bugs
- Take an important program (e.g., a device driver)
- Merge it with a property (e.g., no deadlocks, asynchronous IRP handling, BSD sockets, database transactions, …)
- Transform the result into a *boolean program*
  - Same control flow, but only boolean variables
- Use a model checker to explore the resulting *state space*
  - Result 1: program provably satisfies property
  - Result 2: program violates property *right here on line 92.376!*



Example Program

```
Example ( ) {
    do{
        lock();
        old = new;
        q = q->next;
        if (q != NULL){
            q->data = new;
            unlock();
            new ++;
        }
    } while(new != old);
    unlock();
    return;
}
```

Is this program correct?

Example Program

```
Example ( ) {
    do{
        lock();
        old = new;
        q = q->next;
        if (q != NULL){
            q->data = new;
            unlock();
            new ++;
        }
    } while(new != old);
    unlock();
    return;
}
```

Is this program correct?

What does correct mean?

Doing no evil?

Doing some good?

How do we determine if a program is correct?

Verification by Model Checking

```
Example ( ) {
1: do{
       lock();
       old = new;
       q = q->next;
2:     if (q != NULL){
3:        q->data = new;
          unlock();
          new ++;
       }
4: } while(new != old);
5:  unlock();
    return;
}
```

1. (Finite State) Program
2. State Transition Graph
3. Reachability

- Pgm → Finite state model
- State explosion

\+ State Exploration

\+ Counterexamples

Precise [SPIN, SMV, Bandera, JPF]

For Our Next Exciting Episode

- See webpage under “Lectures”
- Read the two articles
- Peruse the HW and Project pages