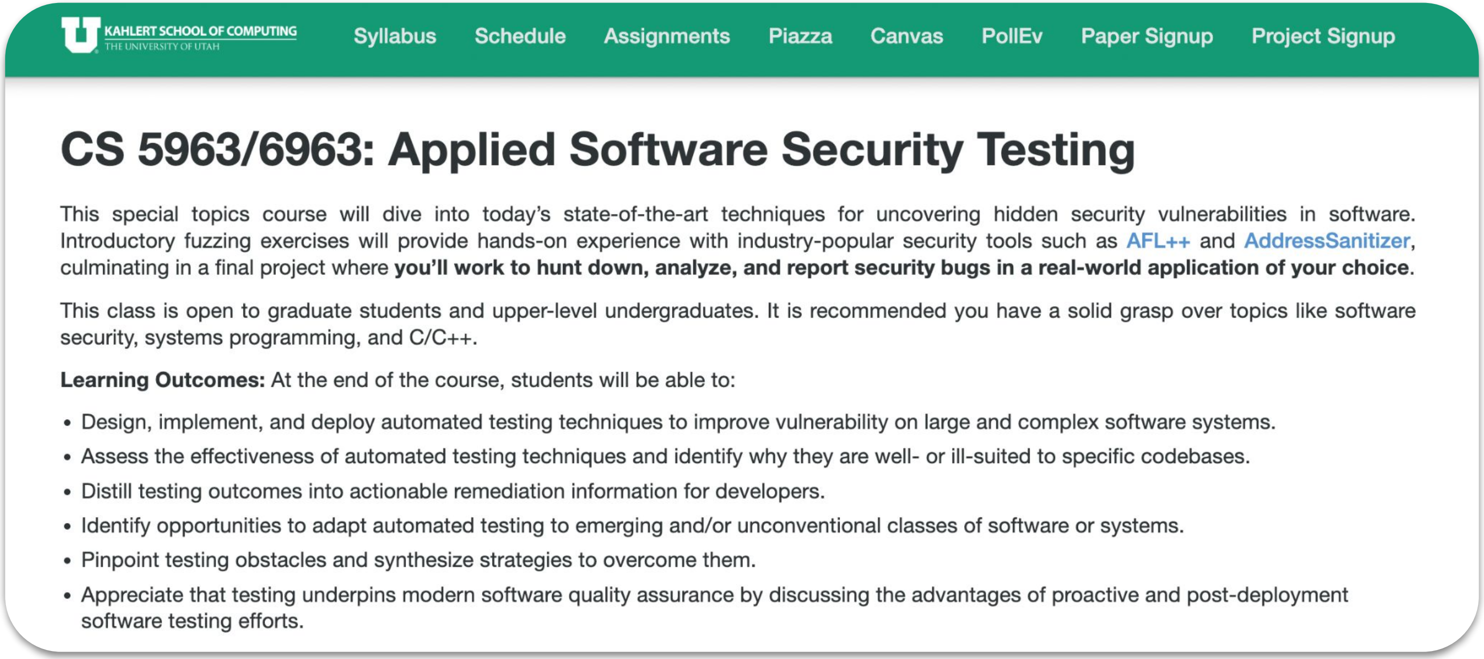
Week 2: Lecture A

Research 101: Writing

Monday, January 13, 2024

Recap: Course Website

cs.utah.edu/~snagy/courses/cs5963



CS 5963/6963: Applied Software Security Testing

This special topics course will dive into today's state-of-the-art techniques for uncovering hidden security vulnerabilities in software. Introductory fuzzing exercises will provide hands-on experience with industry-popular security tools such as AFL++ and AddressSanitizer, culminating in a final project where **you'll work to hunt down, analyze, and report security bugs in a real-world application of your choice**.

This class is open to graduate students and upper-level undergraduates. It is recommended you have a solid grasp over topics like software security, systems programming, and C/C++.

Learning Outcomes: At the end of the course, students will be able to:

- Design, implement, and deploy automated testing techniques to improve vulnerability on large and complex software systems.
- Assess the effectiveness of automated testing techniques and identify why they are well- or ill-suited to specific codebases.
- Distill testing outcomes into actionable remediation information for developers.
- Identify opportunities to adapt automated testing to emerging and/or unconventional classes of software or systems.
- Pinpoint testing obstacles and synthesize strategies to overcome them.
- Appreciate that testing underpins modern software quality assurance by discussing the advantages of proactive and post-deployment software testing efforts.

Recap: Course Resources

Course website .... assignments, schedule, slides, paper signup

Piazza .... questions, discussion, announcements

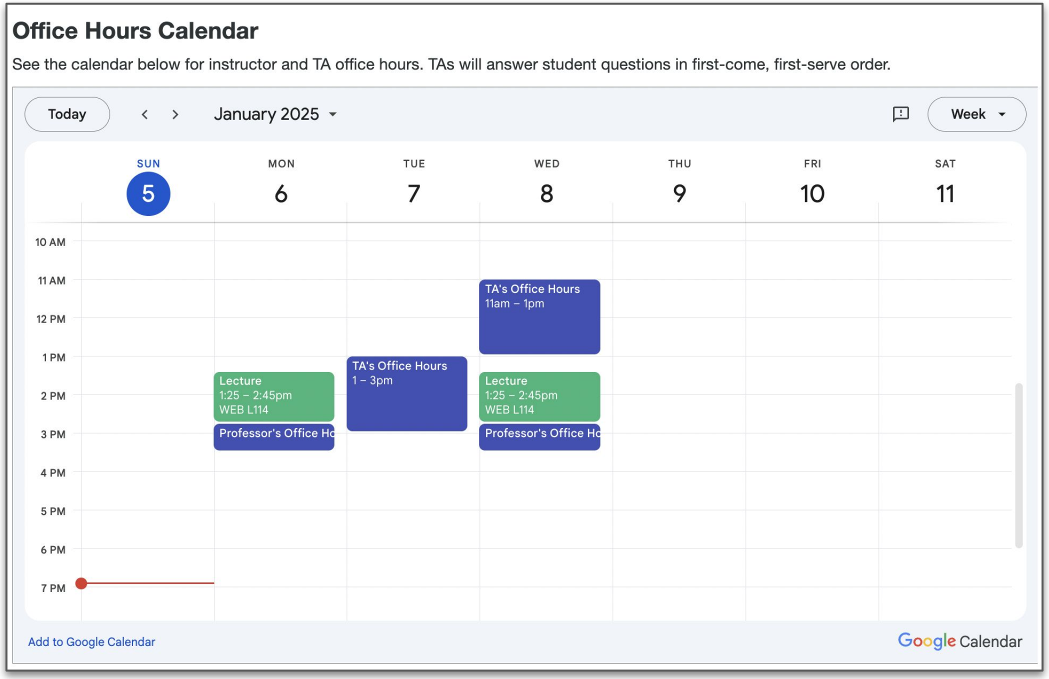
Canvas .... homework submission, course gradebook

Discord .... fun content/discussion relevant to the course

Instructor email (snagy@cs.utah.edu) .... administrative issues

Recap: Office Hours

cs.utah.edu/~snagy/courses/cs5963



Office Hours Calendar

See the calendar below for instructor and TA office hours. TAs will answer student questions in first-come, first-serve order.

TA's Office Hours 11am – 1pm

Lecture 1:25 – 2:45pm WEB L114

Professor's Office Ho

TA's Office Hours 1 – 3pm

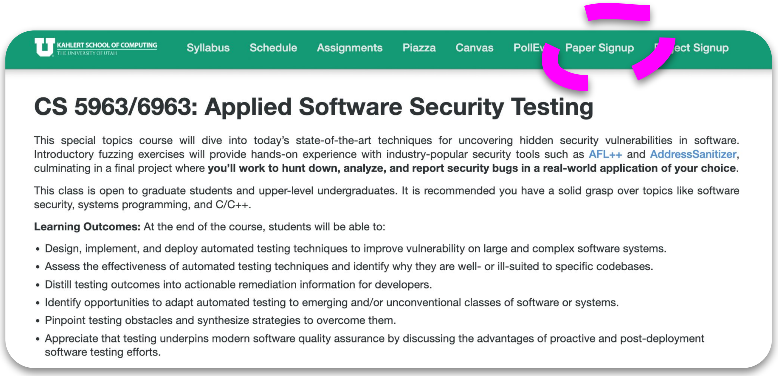
Lecture 1:25 – 2:45pm WEB L114

Professor's Office Ho

Recap: No Exams

Recap: Paper Presentations

- **Signup sheet** available on course website (must use **UofU gcloud** account)
  - **40-45 fuzzing papers** from top venues in security, software engineering, and PL
- Choose one paper by **Wednesday, January 15**
  - **CS 5963 students:** must team-up in pairs of **two**
  - **CS 6963 students:** must present paper **solo**



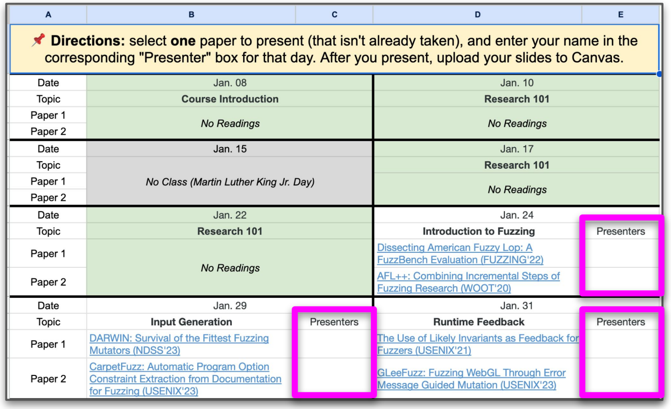
CS 5963/6963: Applied Software Security Testing

This special topics course will dive into today's state-of-the-art techniques for uncovering hidden security vulnerabilities in software. Introductory fuzzing exercises will provide hands-on experience with industry-popular security tools such as AFL++ and AddressSanitizer, culminating in a final project where **you'll work to hunt down, analyze, and report security bugs in a real-world application of your choice.**

This class is open to graduate students and upper-level undergraduates. It is recommended you have a solid grasp over topics like software security, systems programming, and C/C++.

Learning Outcomes: At the end of the course, students will be able to:

- Design, implement, and deploy automated testing techniques to improve vulnerability on large and complex software systems.
- Assess the effectiveness of automated testing techniques and identify why they are well- or ill-suited to specific codebases.
- Distill testing outcomes into actionable remediation information for developers.
- Identify opportunities to adapt automated testing to emerging and/or unconventional classes of software or systems.
- Pinpoint testing obstacles and synthesize strategies to overcome them.
- Appreciate that testing underpins modern software quality assurance by discussing the advantages of proactive and post-deployment software testing efforts.

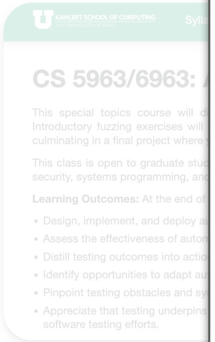


📌 **Directions:** select **one** paper to present (that isn't already taken), and enter your name in the corresponding "Presenter" box for that day. After you present, upload your slides to Canvas.

A	B	C	D	E
Date	Jan. 08		Jan. 10	
Topic	**Course Introduction**		**Research 101**	
Paper 1	*No Readings*		*No Readings*	
Paper 2				
Date	Jan. 15		Jan. 17	
Topic	*No Class (Martin Luther King Jr. Day)*		**Research 101**	
Paper 1			*No Readings*	
Paper 2				
Date	Jan. 22		Jan. 24	
Topic	**Research 101**		**Introduction to Fuzzing**	Presenters
Paper 1	*No Readings*		Dissecting American Fuzzy Lop: A FuzzBench Evaluation (FUZZING'22)	
Paper 2			AFL++: Combining Incremental Steps of Fuzzing Research (WOOT'20)	
Date	Jan. 29		Jan. 31	
Topic	**Input Generation**	Presenters	**Runtime Feedback**	Presenters
Paper 1	DARWIN: Survival of the Fittest Fuzzing Mutators (NDSS'23)		The Use of Likely Invariants as Feedback for Fuzzers (USENIX'21)	
Paper 2	CarpetFuzz: Automatic Program Option Constraint Extraction from Documentation for Fuzzing (USENIX'23)		GLeeFuzz: Fuzzing WebGL Through Error Message Guided Mutation (USENIX'23)	

Recap: Paper Presentations

- Signup she ... ud account)
  - 40-45 f ... nd PL
- Choose on
  - CS 5963
  - CS 6963

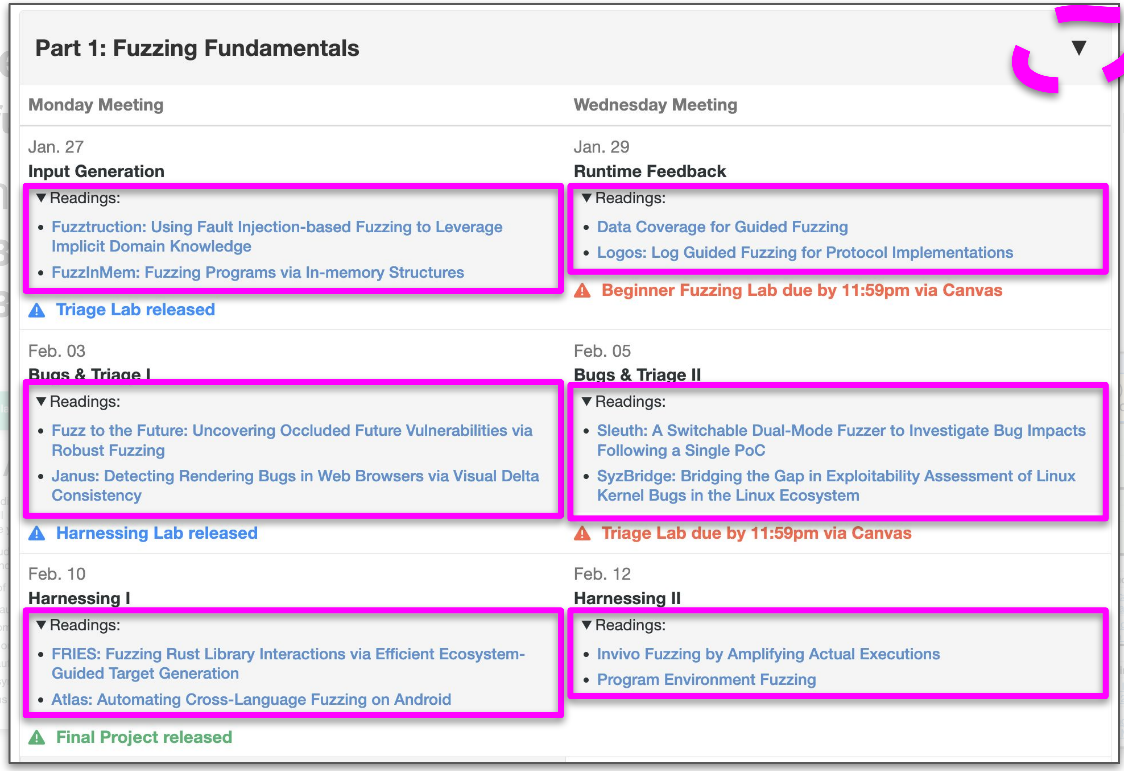

CS 5963/6963:

This special topics course will ... Introductory fuzzing exercises will ... culminating in a final project where

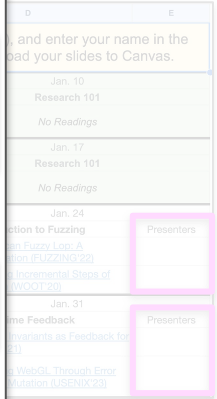
This class is open to graduate stud ... security, systems programming, and

Learning Outcomes: At the end of

- Design, implement, and deploy a
- Assess the effectiveness of autom
- Distill testing outcomes into actio
- Identify opportunities to adapt au
- Pinpoint testing obstacles and sy
- Appreciate that testing underpins software testing efforts.


Part 1: Fuzzing Fundamentals ▼

Monday Meeting	Wednesday Meeting
Jan. 27 **Input Generation** ▼Readings: • Fuzztruction: Using Fault Injection-based Fuzzing to Leverage Implicit Domain Knowledge • FuzzInMem: Fuzzing Programs via In-memory Structures ⚠ Triage Lab released	Jan. 29 **Runtime Feedback** ▼Readings: • Data Coverage for Guided Fuzzing • Logos: Log Guided Fuzzing for Protocol Implementations ⚠ Beginner Fuzzing Lab due by 11:59pm via Canvas
Feb. 03 **Bugs & Triage I** ▼Readings: • Fuzz to the Future: Uncovering Occluded Future Vulnerabilities via Robust Fuzzing • Janus: Detecting Rendering Bugs in Web Browsers via Visual Delta Consistency ⚠ Harnessing Lab released	Feb. 05 **Bugs & Triage II** ▼Readings: • Sleuth: A Switchable Dual-Mode Fuzzer to Investigate Bug Impacts Following a Single PoC • SyzBridge: Bridging the Gap in Exploitability Assessment of Linux Kernel Bugs in the Linux Ecosystem ⚠ Triage Lab due by 11:59pm via Canvas
Feb. 10 **Harnessing I** ▼Readings: • FRIES: Fuzzing Rust Library Interactions via Efficient Ecosystem-Guided Target Generation • Atlas: Automating Cross-Language Fuzzing on Android ⚠ Final Project released	Feb. 12 **Harnessing II** ▼Readings: • Invivo Fuzzing by Amplifying Actual Executions • Program Environment Fuzzing


, and enter your name in the
oad your slides to Canvas.

Jan. 10
Research 101
No Readings

Jan. 17
Research 101
No Readings

Jan. 24
ction to Fuzzing
Presenters

Jan. 31
me Feedback
Presenters

Recap: Hands-on Labs

- Three (relatively easy) labs to be completed **solo**
  - **Lab 1:** Beginner fuzzing
  - **Lab 2:** Crash triage
  - **Lab 3:** Target harnessing
- Paced with the introductory content from Weeks 4–8
  - Apply the techniques you've learned in class
  - Get familiar with state-of-the-art tools like **AFL** and **ASAN**
  - **Deliverables:** a short report (1–3 pages) of what you've learned
- **Designed to prepare you for the Semester Final Project**

Recap: Semester Final Project

- Objective: **uncover new bugs in a real-world program**
- Team up in groups of **2 – 4**
- Select an "interesting" target program of your choice; e.g.:
  - Popular applications
  - Nintendo emulators
  - Old computer games
  - MacOS Rosetta
  - **GET CREATIVE!**
- **Figure out how to fuzz** your target, **find bugs**, and **responsibly disclose them**
- **Deliverables:** a report, disclosure of bugs, and open-source your team's fuzzer

Recap: Semester Final Project

- Objective: u
- Team up in
- Select an "i
  - Popular
  - Nintend
  - Old comp
  - MacOS Ro
  - GET CRE
- **Figure out h** e them
- **Deliverables** n's fuzzer

5-minute **lightning pitch** on Feb. 26

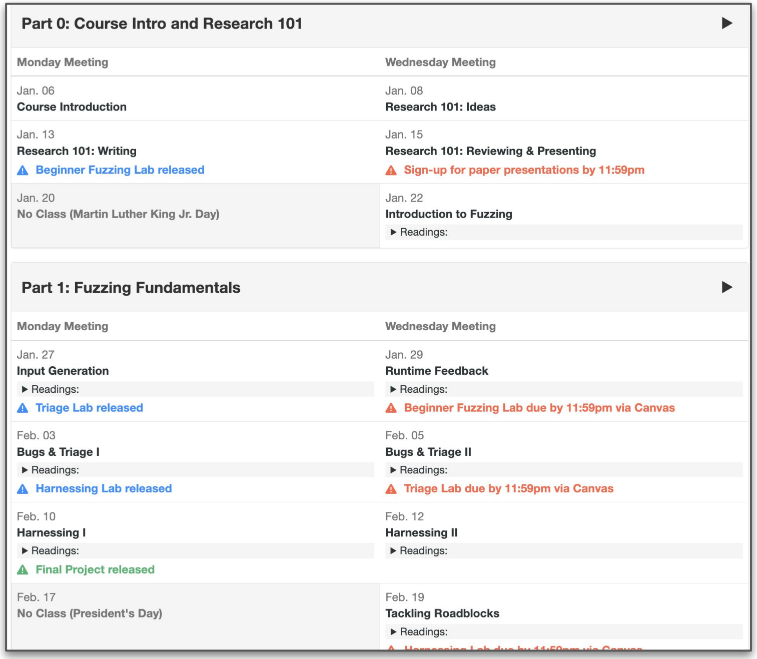
Final presentations at semester's end

You have full creative liberty—get creative and **fuzz something fun**!

Recap: Key Dates

Jan. 15	**Select one paper to present**
Jan. 20	No class (MLK Jr. Day)
Jan. 29	Lab 1 due
Feb. 05	Lab 2 due
Feb. 17	No class (President's Day)
Feb. 19	Lab 3 due
Feb. 26	**5-minute project pitches**
Mar. 10 & 12	No class (Spring Break)
Apr. 14 & 21	**Final project presentations**

cs.utah.edu/~snagy/courses/cs5963/schedule



Part 0: Course Intro and Research 101 ▶

Monday Meeting	Wednesday Meeting
Jan. 06 **Course Introduction**	Jan. 08 **Research 101: Ideas**
Jan. 13 **Research 101: Writing** ⚠ Beginner Fuzzing Lab released	Jan. 15 **Research 101: Reviewing & Presenting** ⚠ Sign-up for paper presentations by 11:59pm
Jan. 20 No Class (Martin Luther King Jr. Day)	Jan. 22 **Introduction to Fuzzing** ▶ Readings:

Part 1: Fuzzing Fundamentals ▶

Monday Meeting	Wednesday Meeting
Jan. 27 **Input Generation** ▶ Readings: ⚠ Triage Lab released	Jan. 29 **Runtime Feedback** ▶ Readings: ⚠ Beginner Fuzzing Lab due by 11:59pm via Canvas
Feb. 03 **Bugs & Triage I** ▶ Readings: ⚠ Harnessing Lab released	Feb. 05 **Bugs & Triage II** ▶ Readings: ⚠ Triage Lab due by 11:59pm via Canvas
Feb. 10 **Harnessing I** ▶ Readings: ⚠ Final Project released	Feb. 12 **Harnessing II** ▶ Readings:
Feb. 17 No Class (President's Day)	Feb. 19 **Tackling Roadblocks** ▶ Readings:

Questions?

Writing:
The Communication of Research

Why write papers?

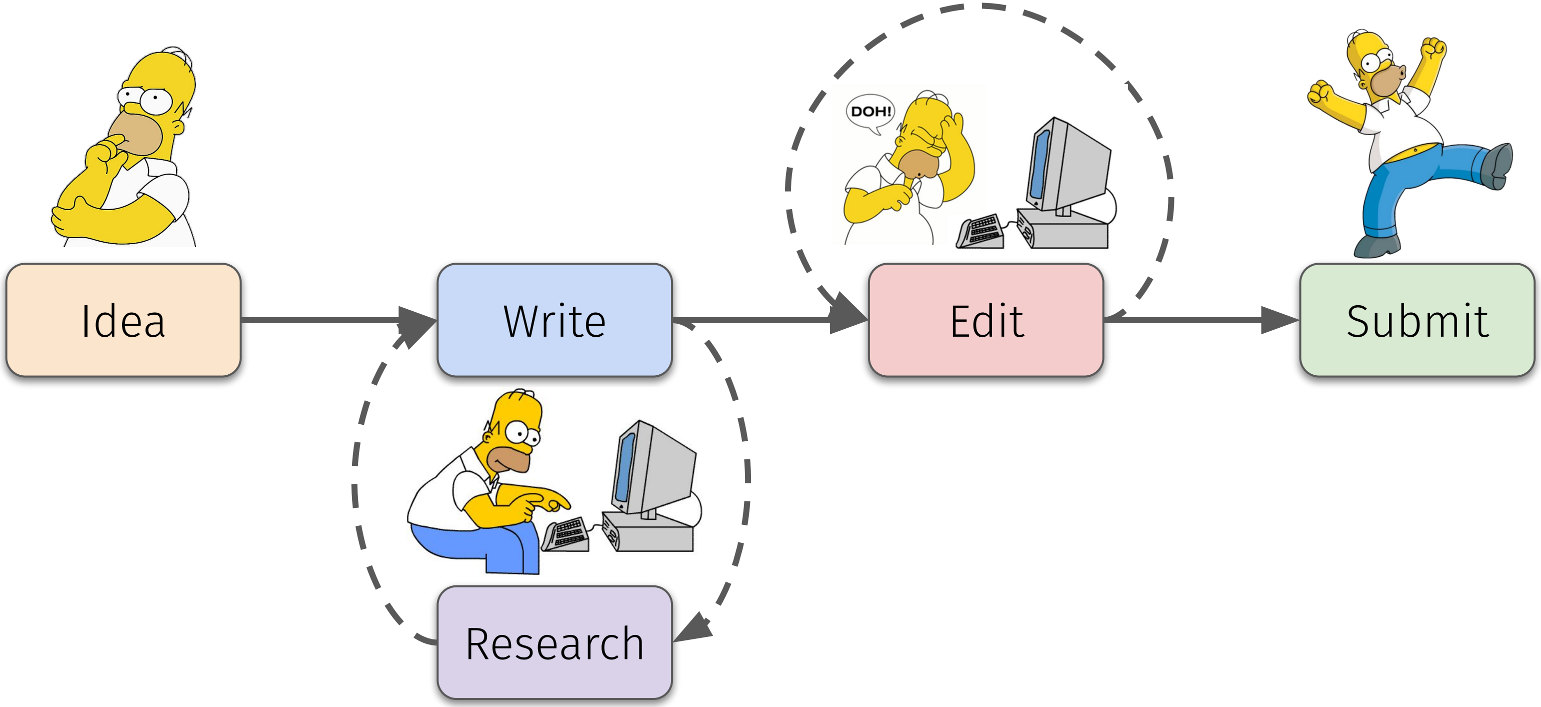
If you don't publically document your work, then it does not exist (beyond you)

Why write papers?

- Document and communicate **what you did**
- Convince others that **you actually did it**
- Convince others that what you did **actually matters**
- **Because you won’t get a Ph.D. without it**

Writing Papers

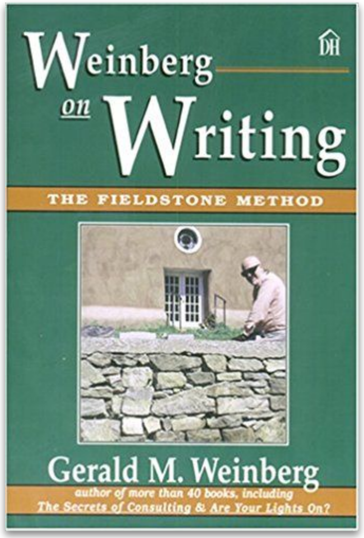
Writing papers is a process…



The Fieldstone Writing Method

- Start putting words on paper **as early as possible**
  - Writing near a deadline is really hard
  - Finalizing **“the pitch” is an iterative process**

- Write as you go along
  - It is **easier** to talk about a problem you are currently solving
  - It is **harder** to remember all problems you solved on short notice
  - It is **easier** to revise and remove than to create from scratch


Weinberg on Writing
THE FIELDSTONE METHOD
Gerald M. Weinberg
author of more than 40 books, including
The Secrets of Consulting & Are Your Lights On?

Before you start: The Tagline

- **What is your paper's tagline?**
- At most **two sentences** (15 seconds in an elevator)
- Rest of paper must gracefully support the tagline

"There can only be ONE (paper tagline)"

Titles

- Highlight **what you do** and **distinguishing properties**
  - Objective is not to make you look smart (e.g., big words)
- Common distinguishing adjectives:
  - Automatic
  - Low-overhead
  - Dynamic
  - Reconfigurable
  - **Find a favorite thesaurus**
- Disambiguate the core message

The Title Rule

Paper titles should be fun or catchy, and ultimately memorable.

"You Autocomplete Me: …"

"Fuzzing Hardware like Software"

Who's Calling? Characterizing Robocalls… "

"Who Left Open the Cookie Jar? A Evaluation of Third-Party Cookies"

"Users Really Do Plug in USB Drives They Find"

The (other) Title Rule

Your paper title and system name should be Google-able.

"**ParmeSAN**: Sanitizer-guided Fuzzing..."

"**Fuzzing@Home**: Distributed Fuzzing..."

"**ExcelLint**: Finding Spreadsheet Errors..."

"**Favocado**: Fuzzing Binding Code..."

Paper Outline

Abstract

Introduction

Background?

Technique

Implementation

Evaluation

Discussion?

Related Work

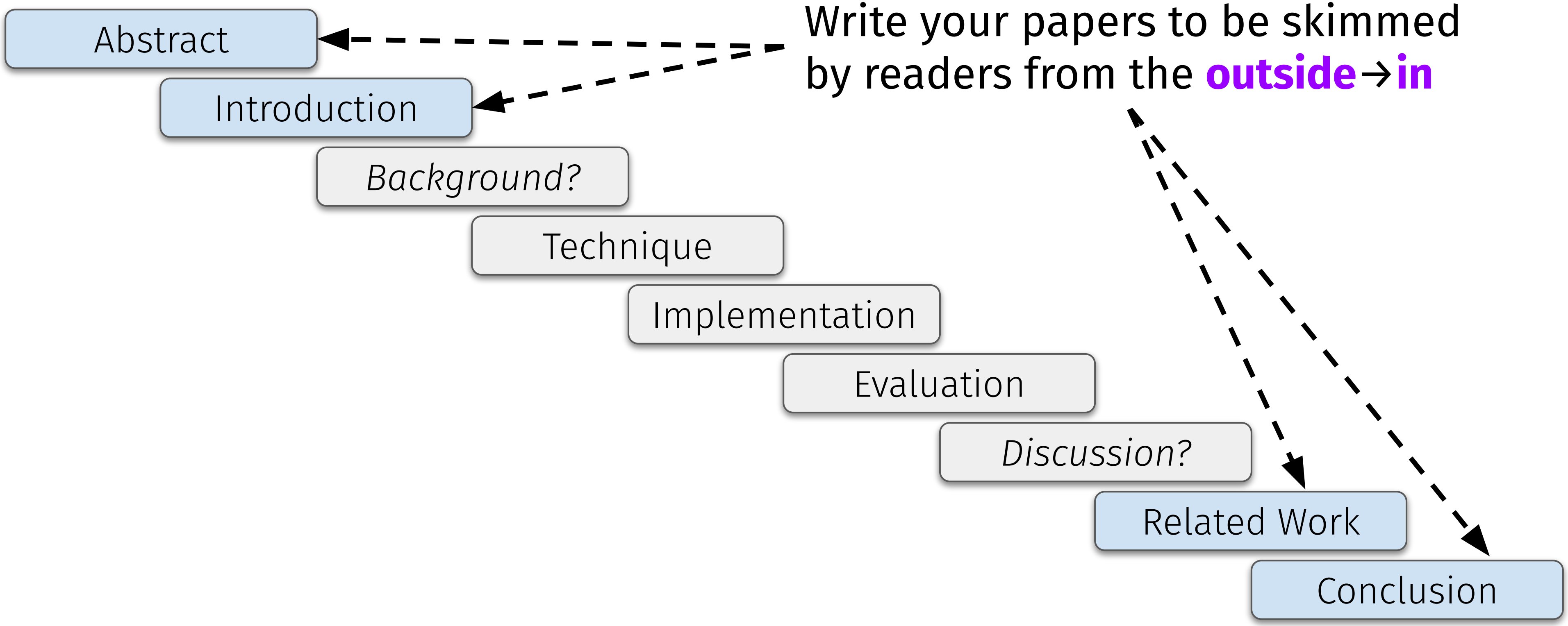
Conclusion

The “Makes Sense” Rule

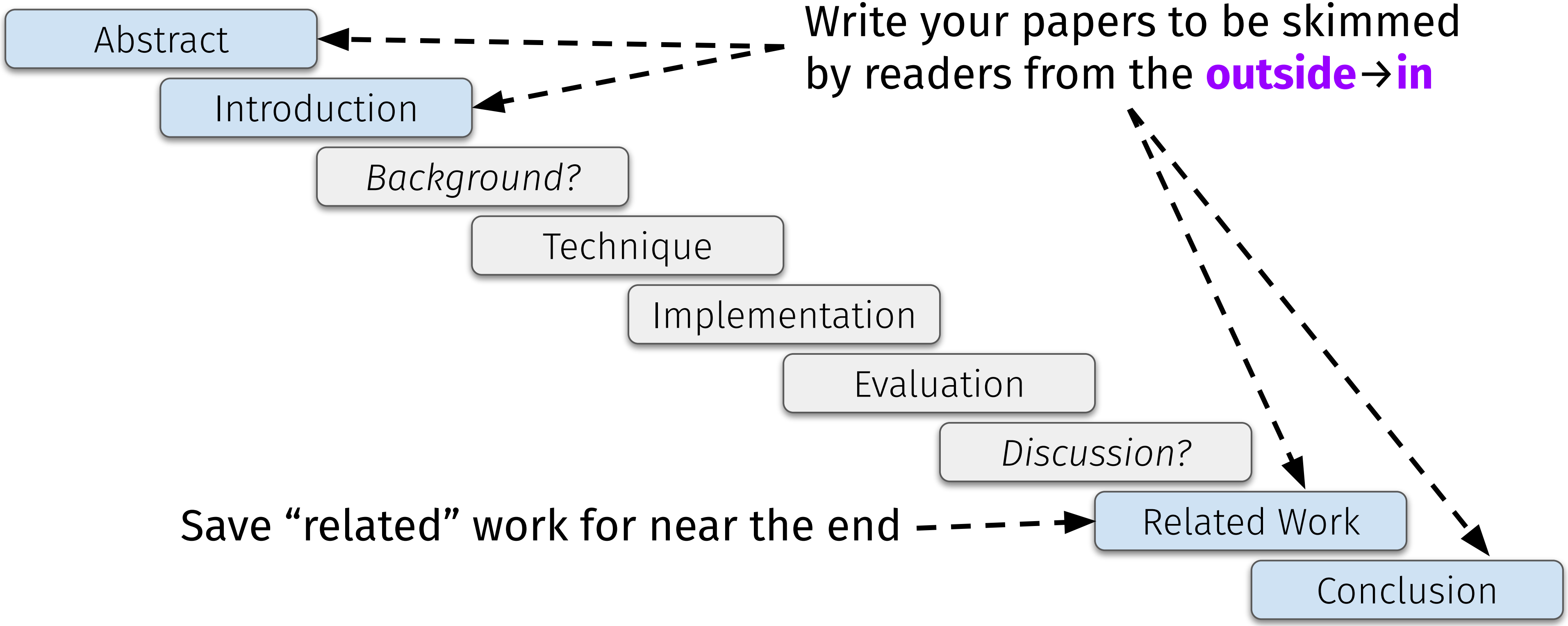
A research paper should make sense just from reading its **introduction**, **conclusion**, and the **captions** to all figures and tables.

Paper Outline

Write your papers to be skimmed by readers from the **outside→in**



Paper Outline



Write your papers to be skimmed by readers from the **outside→in**

Save “related” work for near the end

Structure of (a good) Paper

Abstract

- **Write as if it's a standalone document**
  - Very high level and concise description
- First paragraph
  - High-level problem and motivation
  - Bridge sentence: **what's the gap?**
- Second paragraph
  - Description of insights and approach
  - Sum up your experiment and results
  - Use **formatting** to your *advantage*

Abstract

- **Shark Tank:** study the “art” of the pitch
- Good pitches
  - Concise, correct, and high-level
  - Idea hasn’t been done before
  - Something that matters to society
  - Proof is in the pudding (i.e., results)
- Bad pitches
  - Not concise, incorrect, or too technical
  - Limited impact or already been done
  - Outcomes bad or not measurable

Choose Clarity over Complexity

Once, David Goodstein, a colleague of the Nobel-Prize-winning theoretical physicist Richard Feynman, said "Rich, explain to me, so that I can understand it, why spin one-half particles obey Fermi-Dirac statistics?"

Feynman looked at Goodstein and said, "I'll prepare a freshman lecture on it."

The physicist went away to compose his lesson, but a few days later came back to his colleague, "I couldn't do it," Feynman said, "I couldn't reduce it to the freshman level. That means we don't really understand it."

https://archinect.com/news/article/150171511/complexity-over-clarity-what-happens-when-we-try-to-sound-too-smart

Papers Should Tell a Story

Telling a story in a technical paper is not like Shakespeare or writing Dante's Inferno, but like creating a character in a movie. The goal is logical connectedness. We have all been to movies where a character did something that didn't make sense or where we said, "I'd never do that." We have all experienced plot holes in movies. When writing a technical paper, our goals is to tell a story without plot holes. We NEVER want a reviewer to say, "Why did they do it that way? This doesn't make sense. This is unclear, what are the authors hiding?" This motto applies to writing the design section just as much as the evaluation section. It even applies to the intro where our goal is to convince reviewers that our problem is important, challenging, that our approach follows given previous work, and that our approach is effective.

The “Get to the Point” Rule

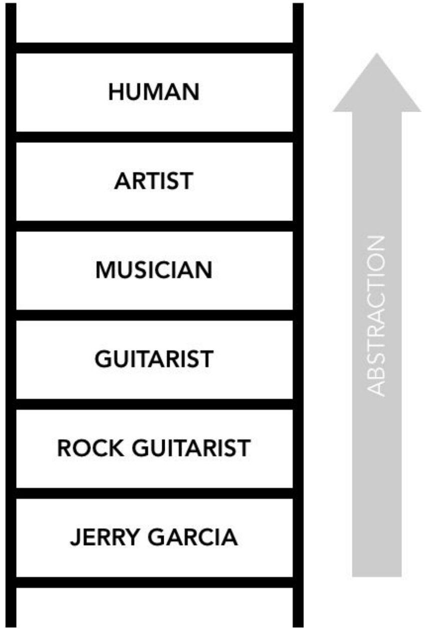
Don't write a mystery novel; give the reader the important information up front.

Walk the "Abstraction Ladder"

- **High level:** the abstract concepts, layman's terms
  - Don't include low-level details or terms here
  - Be succinct yet correct
  - Assume audience is **clueless**

- **Low level:** technical details, specialized terms
  - Assume audience is **knowledgeable**

- Work your description from the high-level to the low-level, then back up



Introduction

- Accept/reject decisions often made here
- 4–6 paragraphs
  - Motivation
  - Problem
  - What others have done and the gap
  - What you do
  - How you implement and evaluate
  - Results
  - List of contributions
- Don’t waste space with a paper outline

Introduction

- Don't spend too much space addressing the work of others
  - It detracts from the presentation of **your work**
- Address works **reviewers** will most likely relate to your work
- A string of references signals to readers that your work is a small boat in a sea of precious work
- A good intro generally takes 1–1.5 pages
- Only append a citation once for each context you use it in

The Heilmeier Catechism

- **What are you trying to do?** Articulate objectives using absolutely no jargon.
- **How is it done today**, and **what are the limits** of current practice?
- **What is new in your approach** and why do you think it will be successful?
- **Who cares?** If you are successful, what difference will it make?
- What are the **risks**?
- How much will it **cost**?
- How **long** will it take?
- What are the mid-term and final **"exams"** to check for success?

The Novelty Rule

Don't claim to be first—even if you are—because that pisses reviewers off and they can always find a paper that—from 10,000 feet away—looks similar to yours.

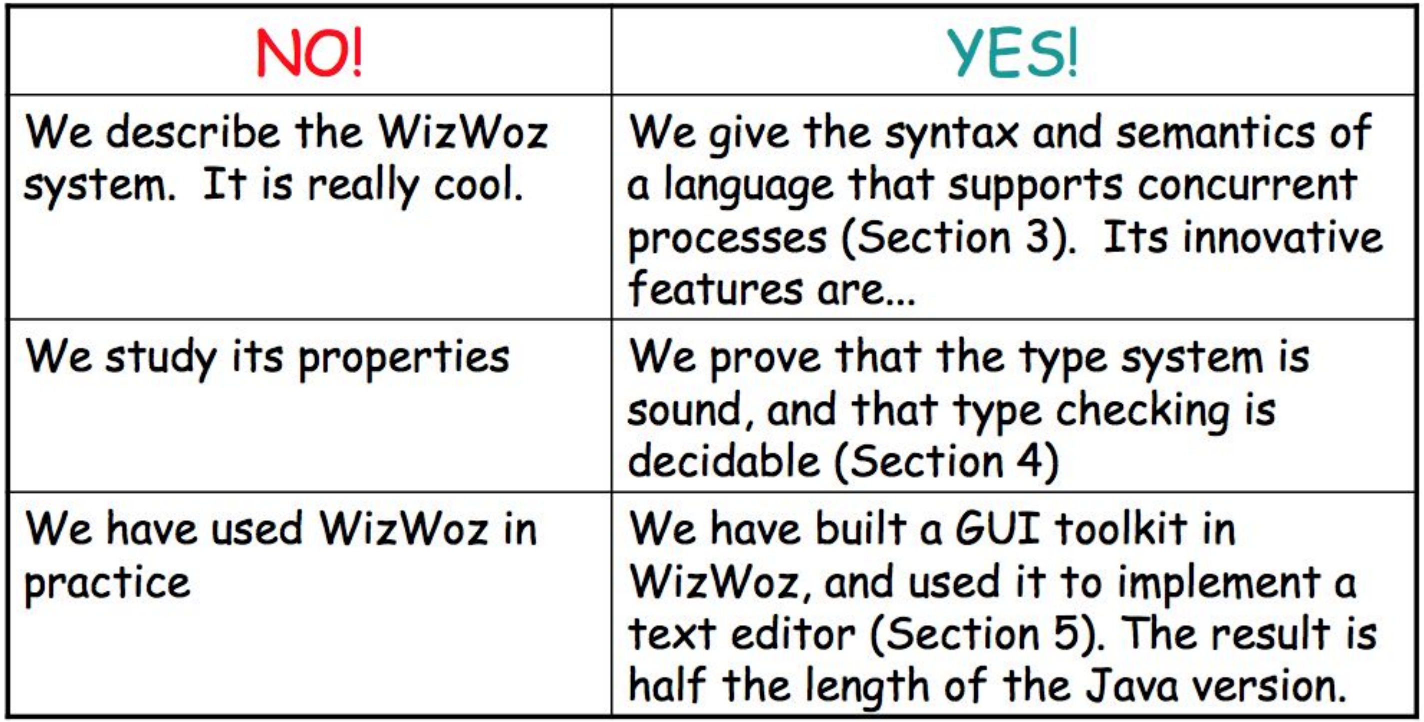
Let others make those claims.

Corollary to The Novelty Rule

Don’t tell readers what to think (e.g., our approach is **simple**, **clever**, **novel**, **awesome**, the **best ever**).

Let others make those claims.

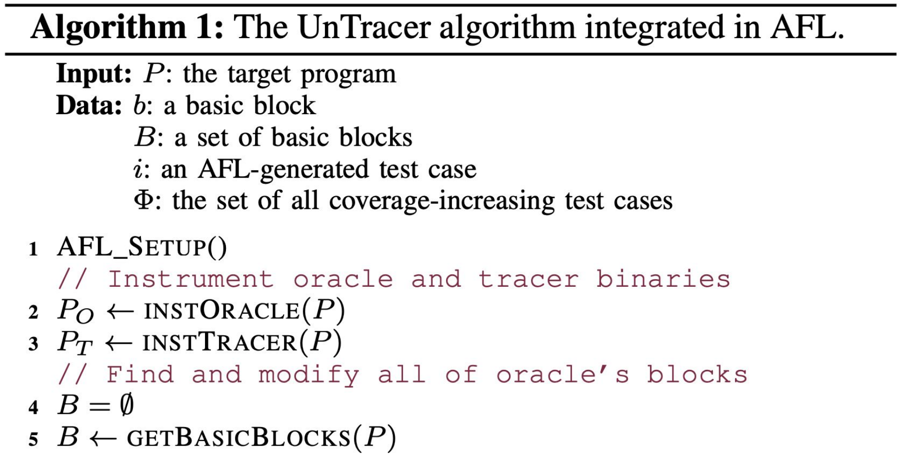
Be Concrete and Explicit



NO!	YES!
We describe the WizWoz system. It is really cool.	We give the syntax and semantics of a language that supports concurrent processes (Section 3). Its innovative features are...
We study its properties	We prove that the type system is sound, and that type checking is decidable (Section 4)
We have used WizWoz in practice	We have built a GUI toolkit in WizWoz, and used it to implement a text editor (Section 5). The result is half the length of the Java version.

The Algorithm Rule

Your technique is better explained **in words**.



Algorithm 1: The UnTracer algorithm integrated in AFL.

```
Input: $P$: the target program
Data: $b$: a basic block
      $B$: a set of basic blocks
      $i$: an AFL-generated test case
      $\Phi$: the set of all coverage-increasing test cases
AFL_SETUP()
// Instrument oracle and tracer binaries
$P_O \leftarrow$ INSTORACLE($P$)
$P_T \leftarrow$ INSTTRACER($P$)
// Find and modify all of oracle's blocks
$B = \emptyset$
$B \leftarrow$ GETBASICBLOCKS($P$)
```

Related Work

- Not a place to disparage previous work
- Not a place to show your breadth of knowledge
- **Tell a story**
  - How the problem has progressed throughout history
  - How ideas relate to and build off each other
- Keep it to a few lines per paper
- End sections with **how your work fits in**

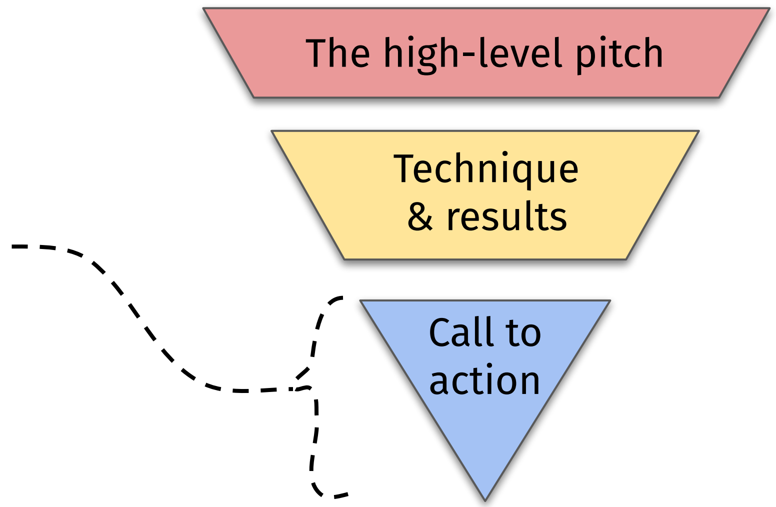
Related Work

- Delay an in-depth literature review
  - **Don’t try to learn everything at once**
  - Read one paper per week during system building
  - Curate, organize, and annotate a bibliography
- What papers are reviewers likely to think of when they read yours?
- Refer to papers by how they are best known (not always by author)
  - **Example:** “SystemName shows…” **instead of** “Simpson et al. shows…”
- Sentences should be complete if you were to remove citations
  - **Example:** “SystemName [1] shows…” **instead of** “[1] shows…”

Conclusion

- Reverse pyramid: **tell them what you did**
  - **Start:** specific evaluation results
  - **End:** area and societal meaning

- High-level implications of your results
  - Recommendations
  - New opportunities
  - Future directions

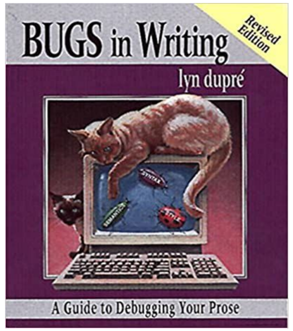


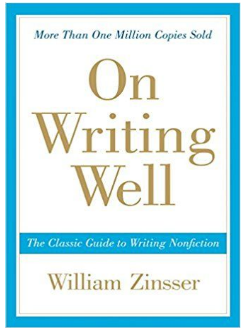
Refining your Writing

Grammar

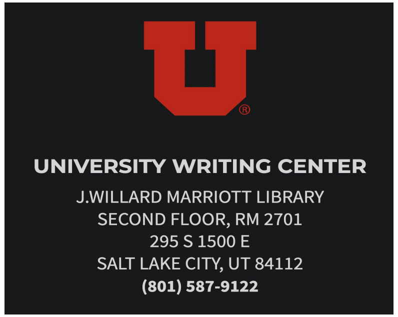
- Avoid past tense; use **present tense**
  - “We implement” instead of “We implemented”
- Avoid passive voice; **use active voice**
  - “We analyze functions…” instead of “Functions are analyzed…”
- Avoid **contractions**
  - “do not” instead of “don’t”
- Avoid **wiggle words**
  - Would, could, should, maybe, possibly, can
- Avoid abstract; **be concrete**
  - “A 300 pound elephant” instead of “A large elephant”
- Do not tell the reader what to do; **tell what you did**
  - “First, you need to find the six least-connected components… ”

Grammar Resources




More Than One Million Copies Sold
On Writing Well
The Classic Guide to Writing Nonfiction
William Zinsser

grammarly



UNIVERSITY WRITING CENTER

J.WILLARD MARRIOTT LIBRARY
SECOND FLOOR, RM 2701
295 S 1500 E
SALT LAKE CITY, UT 84112
(801) 587-9122

Graphics

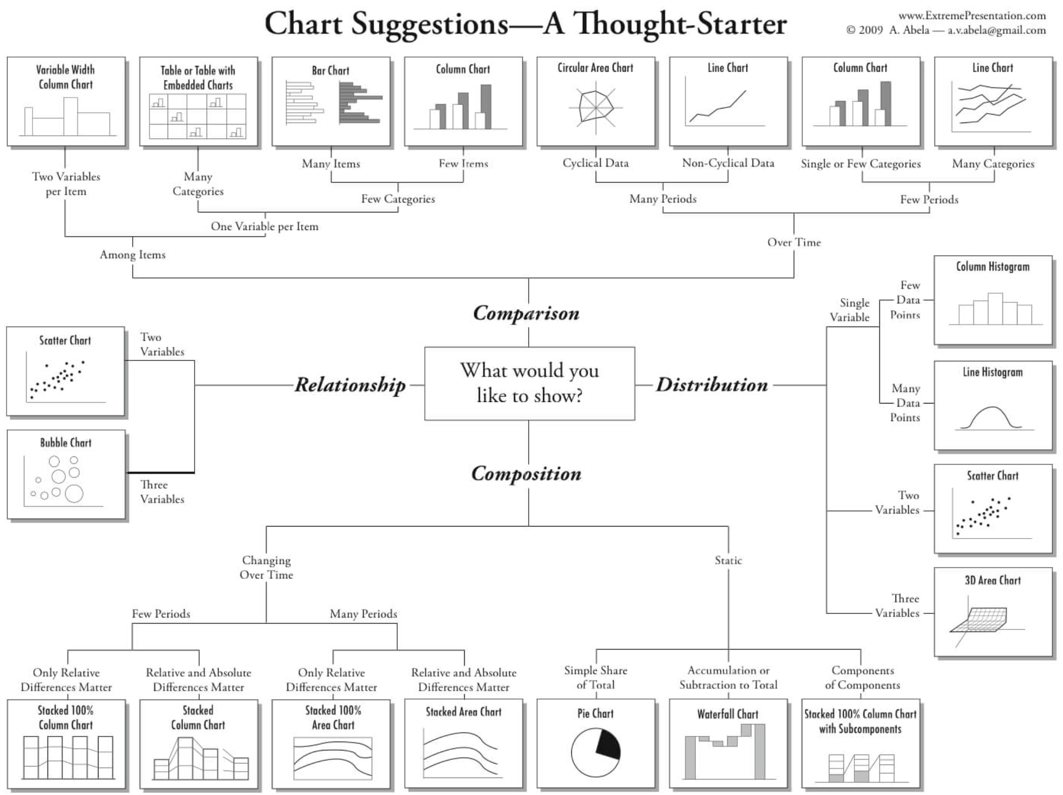
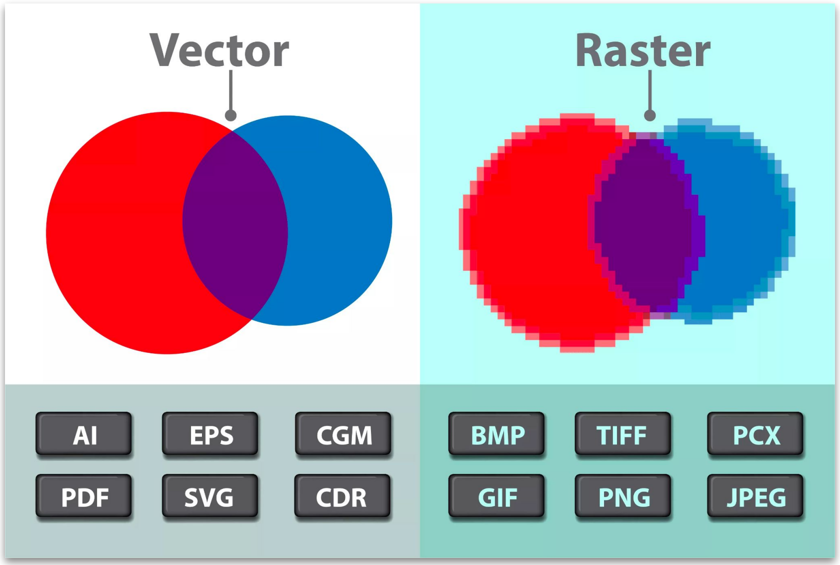


Chart Suggestions—A Thought-Starter
www.ExtremePresentation.com
© 2009 A. Abela — a.v.abela@gmail.com
What would you like to show?
Comparison
Relationship
Distribution
Composition
Among Items
Over Time
Two Variables per Item
One Variable per Item
Many Categories
Few Categories
Many Items
Few Items
Many Periods
Few Periods
Cyclical Data
Non-Cyclical Data
Single or Few Categories
Many Categories
Variable Width Column Chart
Table or Table with Embedded Charts
Bar Chart
Column Chart
Circular Area Chart
Line Chart
Column Chart
Line Chart
Two Variables
Three Variables
Scatter Chart
Bubble Chart
Single Variable
Few Data Points
Many Data Points
Column Histogram
Line Histogram
Two Variables
Scatter Chart
Three Variables
3D Area Chart
Changing Over Time
Static
Few Periods
Many Periods
Only Relative Differences Matter
Relative and Absolute Differences Matter
Only Relative Differences Matter
Relative and Absolute Differences Matter
Simple Share of Total
Accumulation or Subtraction to Total
Components of Components
Stacked 100% Column Chart
Stacked Column Chart
Stacked 100% Area Chart
Stacked Area Chart
Pie Chart
Waterfall Chart
Stacked 100% Column Chart with Subcomponents

The Graphics Rule

Make graphics in vector formats, not rasters.

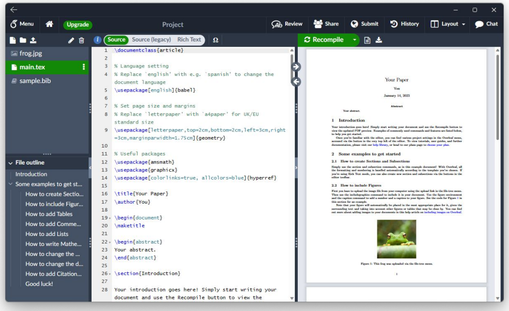


Graphics Resources

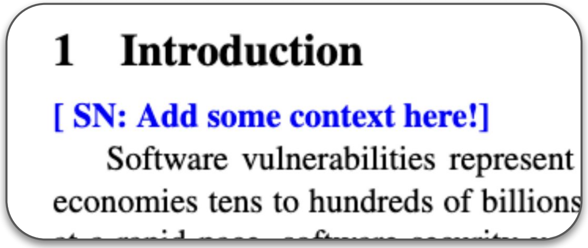
- **Charts and graphs**
  - Python matplotlib
  - Seaborn for enhanced charts
- **System diagrams**
  - Draw.io
  - Powerpoint
  - Inkscape (advanced)
- **Image search engines**
  - Vecteezy
  - Google image search
  - Various clipart websites

Paper Editing

- Use **version tracking** (e.g., GitHub, Overleaf)
- Create macros to leave **in-lined suggestions**
- PDF **diffing utilities** (essential for revisions)
  - `latexpand`: combine many `.tex` files into a single file
  - `latexdiff`: markup changes between two `.tex` files
  - Build a PDF diff from the resulting `.tex` file



```
\documentclass{article}

% Language setting
% Replace `english' with e.g. `spanish' to change the document language
\usepackage[english]{babel}

% Set page size and margins
% Replace `letterpaper' with `a4paper' for UK/EU standard size
\usepackage[letterpaper,top=2cm,bottom=2cm,left=3cm,right=3cm,marginparwidth=1.75cm]{geometry}

% Useful packages
\usepackage{amsmath}
\usepackage{graphicx}
\usepackage[colorlinks=true, allcolors=blue]{hyperref}

\title{Your Paper}
\author{You}

\begin{document}
\maketitle

\begin{abstract}
Your abstract.
\end{abstract}

\section{Introduction}

Your introduction goes here! Simply start writing your document and use the Recompile button to view the
```



1 Introduction

[SN: Add some context here!]

Software vulnerabilities represent economies tens to hundreds of billions

Perform an Adversarial Review

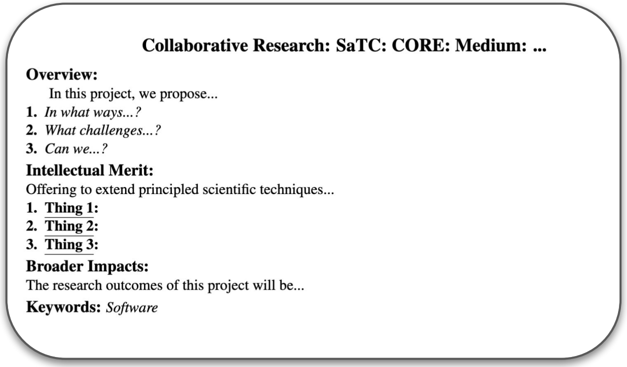
- Identify the hurdles to **believing** your paper
  - Remove hurdles through proof or citation
  - Sometimes a citation is stronger than a self-contained proof
  - **Goal: minimize hurdles**

Perform an Adversarial Review

- Identify the hurdles to **believing** your paper
  - Remove hurdles through proof or citation
  - Sometimes a citation is stronger than a self-contained proof
  - **Goal: minimize hurdles**
- Where and how will a reader **get confused**?
- What will a competitor **disagree with**?
- Are all my claims supported by **reference** or through **experimentation**?

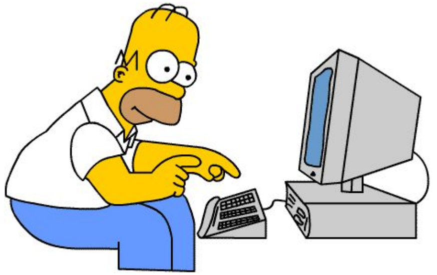
Maintain a Paper Template

- Essential sections
  - With notes on how to write those sections
- Default set of packages
- Useful commands
  - Macros (e.g., `\NameOfOurCoolSystem`)
  - Inlined comment macros (great for collaboration)
- Examples of common insets
  - Figures
  - Tables
  - Code snippets



Collaborative Research: SaTC: CORE: Medium: ...

Overview:
In this project, we propose...
1. *In what ways...?*
2. *What challenges...?*
3. *Can we...?*
Intellectual Merit:
Offering to extend principled scientific techniques...
1. **Thing 1:**
2. **Thing 2:**
3. **Thing 3:**
Broader Impacts:
The research outcomes of this project will be...
Keywords: *Software*



Miscellaneous Tips

- **Introduction:** goal, intuition, reasoning, and takeaways are critical to a story

- For each paragraph:
  - At this point in the paper, **what does the reader know**?
  - What **one point** does this paragraph need to make?

- **Evaluations:** include analysis with description
  - Do not reiterate what readers can see for themselves (e.g., 50% overhead)

- **Implementations are not ideas**
  - Ideas should be **general** (e.g., implementable on other systems)
  - Implementations are **narrow embodiments** of the general idea

Maintain a Lexicon of Cool Words

- Decompose
- Expedition, Frontier
- Side-channel, Out-of-band
- Offline, Online
- Dynamic, Static
- Continuum
- Artifact
- Transient
- Intermittent
- Taxonomy, Orthogonal, Tradeoff space
- Forward error correction, Backward error correction
- Towards (in a title especially)
- Overcoming, Suggests, Asymmetry

The Final Rule

One must always leave something for reviewers to say.

Questions?

Next time on CS 5963/6963...

Research 101: Presenting and Reviewing