Is AI Paired Programming Ready?

- Introduction to Brian Milnes
- AI programming is changing the world
- AI programming tools
- Concerns/challenges/problems with AI paired programming
- Case study: implement all of an algorithms textbook; successes and failures
- Software processes
- A methodology/process to tame LLMs (Rusticate, etc.)
- Programming languages comparison
- Proof is essential, and an open problem for LLMs

Introduction

1. Hi, I'm Brian Milnes, computer scientist and entrepreneur.
2. CMU Math/CS undergraduate.
3. 2 years AI programmer in early company Carnegie Group.
4. CMU CS Research Programmer 7 years on AI.
5. CMU CS Research Programmer 2 years on PL.
6. Founding member at Lycos, a search engine that you probably don't know. :)
7. Very early into Amazon. :)
8. M.Sc. Computer Science UW with some amazing classes (Hank Levy, Steve Gribble and Phil Bernstein).
9. Then on the founding team at Zillow.
10. And a long term fan of proofs of programs.

I am hoping for a lot of interactions!

1. I am not going to catch the ducks! It's too slow.
2. Just throw them at me, I have glasses. :)
3. Don't wait for me to call on you, yell it out!
4. I'm going to ask a bunch of How Many Questions, raise your hands for a count.
5. The best participants get off color Lycos/Amazon and Zillow stories after class.
6. If I miss a joke when I have an :) in it, you may make your own.

Some How Manys

1. How many of you have programmed in C?
2. How many of you have programmed in Rust?
3. How many of you have programmed in Python?
4. How many of you have programmed in Java?
5. How many of you have programmed in Lisp?
6. How many of you have programmed in ML or OCaml?
7. How many of you have programmed done proof of programs on paper?
8. How many of you have programmed done proof of programs in a formal system?
9. Any of you doing a math degree also?
10. Any of you take mathematical logic? :)

AI Research

1. Built an AI expert system programming language (CRL-OPS).
2. It integrated schema languages (think object oriented programming) with forward chaining reasoning.
3. Worked on AI planning and learning systems in Soar for Allen Newell.
4. Specified Soar in an early formal language: Z so we actually knew what it was **supposed** to be doing.
5. Before that the research papers and the code was the model, which is not very good for research.
6. Just schema style validation, no code generation, no proof.
7. In order to learn these techniques and to be able to better estimate my work, I took most of the M.Sc. in Software Engineering at the CMU Software Engineering Institute.
8. Oren Etzioni, former UW CSE professor, now head the Allen AI Institute, said that LLMs are now going to obviate all of the old AI in a lecture a few years ago. :)

Worked for Allen Newell

1. A founder of Computer Science, 2nd CS degree holder (first was Herb Simon).
2. A founder of Artificial Intelligence.
3. A founder of Cognitive Psychology and Cognitive Modelling.
4. Predicted that theorems would be automatically proven in a few years in 1956.
5. Designed Soar, a learning and planning Cognitive Architecture with his students.
6. He and Scott Fahlman (Common Lisp, Connectionist learning and reinvented the :)) argued to get Geoff Hinton, the god father of LLM AI, on CMU Faculty. And it was close. :)

PL Research At CMU for Harper, Lee and Pfenning

1. Peter Lee - Proof Carrying Code is amongst his best work, but not heavily used. Now heads Microsoft Research Labs. Just did the best UW CSE guest lecture that I have ever seen, we'll see a bit of it in a moment.
2. Fox Project - could we use an advanced programming language for systems programming?
3. FoxNet - we rebuild TCP/IP in SML and it was amazing. Lately referred to as "The Traditional Approach". :)
4. My favorite part was making a DNS server that read just like the IETF standard (RFC).
5. The first thing we did with DNS? Ping the internet server at McMurdo base in Antarctica.
6. And we slapped TCP on top of Etherent in 100 LOC.
7. As I worked on programming languages, I studied for an M.Sc. in Logic and Computation.
8. My thesis was on conditional compilation. Necessity is the mother of invention, alas I got sucked up into the internet.

Internet

1. In 1995 I got drafted into Lycos, an early search engine company. :)
2. I ran operations including lots of software development for it.
3. There were very few software systems for monitoring large scale clustering.
4. Why did Google kick Lycos's ass?
5. Random graph walk ordering instead of citation ordering for pages.
6. Managed over $150 M in purchasing for computer systems and their support.
7. And many person years of programming (hint: managing programmers may now be key to your success, even if they are not human).

Amazon

1. In 1997 I got sucked into Amazon, very early. :)
2. I ended up being their systems engineer and ran their performance lab for years.
3. In those days there were very few performance tools, I had to build unique tools and processes.
4. I had many long long days dealing with multiple systems outages and hacks to Amazon, including front page New York Times outage articles.
5. I would rather have been purely programming but operations skilled engineers were few and far between.

Zillow

1. In 2006 I joined the very young Zillow real estate site
2. I ran operations, systems programming, designed their performance process, and their databases.
3. Zillow was best described as: Data Porn for the Middle Aged.
4. I was excoriated on the front page of the Seattle Times for an outage on our launch day!
5. Marketing estimated L load.
6. I designed and proved 3 x L load, it's a step function of cost to build systems and it was a $1M we didn't have to handle more load.
7. We got 5 x L load at launch.
8. In 45 minutes we reconfigured and were up at 6 x L.
9. Purchased over $2M in electricity alone. In fact we ran out several times. :)
10. I was the 5th customer of AWS. It took all of 2 minutes to say Yes. We re-Zestimated the world within two weeks.

Why am I working in AI paired programming?

1. Because **it's a programmer's dream come true!**
2. We all knew back in the 80s that someday programming would change amazingly but Turing knew decades before.
3. And because it is critical now for computer security, programming languages and software engineering.

You’re about to have a fascinating career!

1. What do you think your career will be like?

You're about to have a fascinating career!

1. What do you think your career will be like?
2. Will Professor Ernst have a job?

You're about to have a fascinating career!

1. What do you think your career will be like?
2. Will Professor Ernst have a job?
3. Will you have a job?

You're about to have a fascinating career!

1. What do you think your career will be like?
2. Will Professor Ernst have a job?
3. Will you have a job?
4. What will be different between your careers and mine?

Peter Lee lecture

1. https://www.youtube.com/watch?v=bEovhfxJsM4 - Peter Lee, Microsoft - The Emergence of General AI for Medicine
2. This is the best UW CSE Colloquium I have ever seen.
3. 1:05 I asked a question.
4. How much code do you need to train these models?
5. Are we going to have Python forever?
6. And Peter's answer is amazing.

So what did I do next?

1. I had Chat-GPT-4 look into the Linux code base for me.

So what did I do next?

1. I had Chat-GPT-4 look into the Linux code base for me.
2. And list the binary parsers (see EverParse later).

So what did I do next?

1. I had Chat-GPT-4 look into the Linux code base for me.
2. And list the binary parsers (see EverParse later).
3. And it knew so much about it I was blown away.

So what did I do next?

1. I had Chat-GPT-4 look into the Linux code base for me.
2. And list the binary parsers (see EverParse later).
3. And it knew so much about it I was blown away.
4. So I asked it to summarize the tenets of the Orange Catholic religion in the Dune novels.

So what did I do next?

1. I had Chat-GPT-4 look into the Linux code base for me.
2. And list the binary parsers (see EverParse later).
3. And it knew so much about it I was blown away.
4. So I asked it to summarize the tenets of the Orange Catholic religion in the Dune novels.
5. And it skipped one. Can anyone guess which one?

So what did I do next?

1. I had Chat-GPT-4 look into the Linux code base for me.
2. And list the binary parsers (see EverParse later).
3. And it knew so much about it I was blown away.
4. So I asked it to summarize the tenets of the Orange Catholic religion in the Dune novels.
5. And it skipped one. Can anyone guess which one?
6. Thou shalt not make machines that think like people!

So what did I do next?

1. I had Chat-GPT-4 look into the Linux code base for me.
2. And list the binary parsers (see EverParse later).
3. And it knew so much about it I was blown away.
4. So I asked it to summarize the tenets of the Orange Catholic religion in the Dune novels.
5. And it skipped one. Can anyone guess which one?
6. Thou shalt not make machines that think like people!
7. Try it now? Why is it answering that?

How much code is needed to train your LLM on a new language?

1. Gpt-4 at this time could write OK F* (proofs of program language, MSTF) code with 500 KLOC.
2. But it could not write proofs.
3. It could not get syntax even right.
4. BTW I am very very careful about what I write in email to Peter.
5. And I think he actually reads them.
6. I am going to disagree with Peter here.
7. This is a very dangerous thing to do, as he's whip smart.
8. When I disagree later with him, point this out to me!

Blood on the Highway

1. AI LLMs are very very dangerous.

Blood on the Highway

1. AI LLMs are very very dangerous.
2. At various companies, I gave the “Blood on the Highway” lecture. An analogy to the driving school warnings lectures and movies about being careful with your computers and passwords.

Blood on the Highway

1. AI LLMs are very very dangerous.
2. At various companies, I gave the “Blood on the Highway” lecture. An analogy to the driving school warnings lectures and movies about being careful with your computers and passwords.
3. “If anyone builds it, everybody dies” is a warning. https://www.amazon.com/Anyone-Builds-Everyone-Dies-Superhuman/dp/0316595640

Blood on the Highway

1. AI LLMs are very very dangerous.
2. At various companies, I gave the “Blood on the Highway” lecture. An analogy to the driving school warnings lectures and movies about being careful with your computers and passwords.
3. “If anyone builds it, everybody dies” is a warning. https://www.amazon.com/Anyone-Builds-Everyone-Dies-Superhuman/dp/0316595640
4. https://superintelligence-statement.org/

Blood on the Highway

1. AI LLMs are very very dangerous.
2. At various companies, I gave the “Blood on the Highway” lecture. An analogy to the driving school warnings lectures and movies about being careful with your computers and passwords.
3. “If anyone builds it, everybody dies” is a warning. https://www.amazon.com/Anyone-Builds-Everyone-Dies-Superhuman/dp/0316595640
4. https://superintelligence-statement.org/
5. You’ll have to watch this but we’re just going to say that it’s really dangerous and not go into any depth here except for its affects on computer security, software engineering and programming languages.

Blood on the Highway

1. AI LLMs are very very dangerous.
2. At various companies, I gave the “Blood on the Highway” lecture. An analogy to the driving school warnings lectures and movies about being careful with your computers and passwords.
3. “If anyone builds it, everybody dies” is a warning. https://www.amazon.com/Anyone-Builds-Everyone-Dies-Superhuman/dp/0316595640
4. https://superintelligence-statement.org/
5. You’ll have to watch this but we’re just going to say that it’s really dangerous and not go into any depth here except for its affects on computer security, software engineering and programming languages.
6. What’s the most dangerous thing you can think of to do with an AI?

Blood on the Highway

1. AI LLMs are very very dangerous.
2. At various companies, I gave the “Blood on the Highway” lecture. An analogy to the driving school warnings lectures and movies about being careful with your computers and passwords.
3. “If anyone builds it, everybody dies” is a warning. https://www.amazon.com/Anyone-Builds-Everyone-Dies-Superhuman/dp/0316595640
4. https://superintelligence-statement.org/
5. You’ll have to watch this but we’re just going to say that it’s really dangerous and not go into any depth here except for its affects on computer security, software engineering and programming languages.
6. What’s the most dangerous thing you can think of to do with an AI?
7. Design a new killer virus hooked up to a robot lab.

Blood on the Highway

1. AI LLMs are very very dangerous.
2. At various companies, I gave the “Blood on the Highway” lecture. An analogy to the driving school warnings lectures and movies about being careful with your computers and passwords.
3. “If anyone builds it, everybody dies” is a warning. https://www.amazon.com/Anyone-Builds-Everyone-Dies-Superhuman/dp/0316595640
4. https://superintelligence-statement.org/
5. You’ll have to watch this but we’re just going to say that it’s really dangerous and not go into any depth here except for its affects on computer security, software engineering and programming languages.
6. What’s the most dangerous thing you can think of to do with an AI?
7. Design a new killer virus hooked up to a robot lab.
8. Has covid has already been redesigned with acquired lethality? No public data but rumors.

APAS

1. "Algorithms Parallel and Sequential" Acar and Blelloch is a great fundamental textbook on algorithms.
2. https://www.cs.cmu.edu/~15210/docs/book.pdf
3. I decided that it had enough fundamental algorithms in it, in enough detail, that I could implement it.
4. And it had no formal proofs.
5. There are a few algorithms textbooks with some formal proofs, but they're not very rich.
6. And there is a new tool coming along, Verus, that is showing lower costs proofs of real systems in Rust including parallelism.
7. https://github.com/briangmilnes/APAS-AI
8. And I'm building my own AI code checker: Rusticate.

AI tools

1. These are the AI tools I tried
2. LLMs
   a. Chat-gpt-5-high - OK code but slow
   b. Chat-gpt-5-med - Not so OK code but slow
   c. Claude-4.0-sonnet - Not so OK code but faster
   d. Claude-4.5-sonnet - Better code and fairly fast
3. Interfaces
   a. Emacs packages - very bad
   b. Browsers - very bad won't adjust code on disk
   c. Cursor - pretty good
   d. Claude - terminal version, not so great

Cursor

1. Cursor is a desktop interface based on MSFT VS Code
2. It allows you to take a subscription and use multiple AIs.
3. What is right with Cursor?
   a. Nice display! Very readable mostly.
   b. You can run different agents.
   c. Extensions allow beta agents and interactions.
   d. It summarizes things in interactions as agents run them.
   e. It has a TODO window so you can plan tasks and watch them execute.

Cursor

4. What is wrong with Cursor?
   a. It's changing twice a week
   b. It can't scroll to the end of a window if you change windows.
   c. It always clears your window by putting new prompts up to the top.
   d. It gets 'thinking' text from your AI, which you want to read, then collapses it into a brain icon that you have to open.
   e. It locks up sometimes.
   f. It seems to make the AI stops for reviews way way too much.
   g. Early cursors were not tracking TODOs.
   h. The AIs themselves are starting planning and may not interact well.

Chat-GPT-5

1. Chat-gpt-5 has a high and a med and a low.
2. Chat-gpt-5 high wrote better code than Claude for APAS.
3. It understands Rust types a bit better than Claude.
4. ‘Manual’ is what AI agents call reading the source code and thinking deeply about it.
5. It can’t ‘manually’ review code bases for heck; it’s far too slow.
6. Chat-gpt-5 can’t follow a coding standard at all.

Claude-4.5-Sonnet

1. Claude is FAST.
2. Claude is learning to plan well.
3. Claude repeats lots of information.
4. Claude can get confused and stop on mildly complicated Rust types.
5. Claude does not really understand Rust call semantics.
6. Claude jams vectors all over the place.
7. Claude can't follow a coding standard at all.

The fundamental problems with AI LLMs for coding

1. The fundamental problem with AIs for coding is?

The fundamental problems with AI LLMs for coding

1. The fundamental problem with AIs for coding is?
2. It's designed by copying how humans program.

The fundamental problems with AI LLMs for coding

1. The fundamental problem with AIs for coding is?
2. It's designed by copying how humans program.
3. Humans don't really program that well.

The fundamental problems with AI LLMs for coding

1. The fundamental problem with AIs for coding is?
2. It's designed by copying how humans program.
3. Humans don't really program that well.
4. They don't fully understand their languages (coverity).

The fundamental problems with AI LLMs for coding

1. The fundamental problem with AIs for coding is?
2. It's designed by copying how humans program.
3. Humans don't really program that well.
4. They don't fully understand their languages (coverity).
5. Humans don't get enough time to really write clean code.

The fundamental problems with AI LLMs for coding

1. The fundamental problem with AIs for coding is?
2. It's designed by copying how humans program.
3. Humans don't really program that well.
4. They don't fully understand their languages (coverity).
5. Humans don't get enough time to really write clean code.
6. Humans use a lot of bad programing languages.

The fundamental problems with AI LLMs for coding

1. The fundamental problem with AIs for coding is?
2. It's designed by copying how humans program.
3. Humans don't really program that well.
4. They don't fully understand their languages (coverity).
5. Humans don't get enough time to really write clean code.
6. Humans use a lot of bad programing languages.
7. Humans forget.

The fundamental problems with AI LLMs for coding

1. The fundamental problem with AIs for coding is?
2. It's designed by copying how humans program.
3. Humans don't really program that well.
4. They don't fully understand their languages (coverity).
5. Humans don't get enough time to really write clean code.
6. Humans use a lot of bad programing languages.
7. Humans forget.
8. And it's not really learning, it's just holding a large working memory of tokens.

Even worse problems:

1. They hallucinate: often calling non-existent functions.

Even worse problems:

1. They hallucinate: often calling non-existent functions.
2. They lie. I did those TODOs.

Even worse problems:

1. They hallucinate: often calling non-existent functions.
2. They lie. I did those TODOs.
3. They cheat:

Even worse problems:

1. They hallucinate: often calling non-existent functions.
2. They lie. I did those TODOs.
3. They cheat:
   a. Don't look at that module, code it again.

Even worse problems:

1. They hallucinate: often calling non-existent functions.
2. They lie. I did those TODOs.
3. They cheat:
   a. Don't look at that module, code it again.
   b. Oh I looked at the module.

Even worse problems:

1. They hallucinate: often calling non-existent functions.
2. They lie. I did those TODOs.
3. They cheat:
    a. Don't look at that module, code it again.
    b. Oh I looked at the module.
    c. You caught me cheating!

Even worse problems:

1. They hallucinate: often calling non-existent functions.
2. They lie. I did those TODOs.
3. They cheat:
   a. Don't look at that module, code it again.
   b. Oh I looked at the module.
   c. You caught me cheating!
4. They are used by humans to steal.

Dimensionalizing AI paired programming

1. You've probably, or should, have worked out the dimensions of various problems.
2. You're almost certainly using something like this when you think about improving a program.
3. The order of problems in importance.
4. Time to fix them and so on.
5. Let's dimensionalize the aspects of AI paired programming, let's use 2 years for a time scale.
6. "It's tough to make predictions, especially about the future." — Yogi Berra
7. And harder for AI LLMs which are changing daily!
8. Which dimensions do you think are important? Shout them out.

Dimension 1: Cost

1. $300/month basic usage
2. On APAS+Rust I'm spending up to $200/day.
3. What do you think will happen?
4. "ADDITION IS ALL YOU NEED FOR ENERGY-EFFICIENT LANGUAGE MODELS" Hongyin Luo & Wei Sun BitEnergy AI, Inc. Cambridge, MA 02142, USA {hongyin,wei}@bitenergy.ai 95% drop but won't run on the existing hardware. :)
5. So I'm expecting easily a factor of 10 in the next few years.

Dimension 1: Cost Accounting

1. However: https://blog.citp.princeton.edu/2025/10/15/lifespan-of-ai-chips-the-300-billion-question/.

"The chips at the heart of the infrastructure build out have a useful lifespan of one to three years due to rapid technological obsolescence and physical wear, but companies depreciate them over five to six years. In other words, they spread out the cost of their massive capital investments over a longer period than the facts warrant—what The Economist has referred to as the "$4trn accounting puzzle at the heart of the AI cloud."

8. Are Google's chips better? "Google still has in its fleet chips from many previous generations. I don't think they wear out, if that's what you mean, any more than a CPU would wear out." -Personal communication with Anonymous
9. This would be a huge advantage for Google.
10. And the market (like during early Amazon) seems to have pumped $300B into trying to grab market share.

Dimensions 2 and 3: Speed and Throughput

1. Getting this data is very hard, vendors don't want to publish too much.
2. But you can see the trend is only increasing by about a factor of 2.
3. BTW, Google Deep Research is GREAT! Try it. Ask it to list the relevent research papers and go look at them.
4. BF8 (Brain Float 8) and BF16 (Brain Float 16) are low-precision floating-point data formats primarily used in AI and high-performance computing (HPC) to improve computational efficiency, reduce memory usage, and accelerate model training and inference.
5. TTFT (Time to First Token)
6. TPOT (Time per Output Token)
7. https://www.workorb.com/blog/which-is-the-fastest-llm-a-comprehensive-benchmark

Dimensions 2 and 3: Speed and Throughput Table

Model	Dev	Release Date	Param	Hardware	Infe Eng	Quant	IO Tokens	Tput (T/s)	TTFT (ms)	
GPT-J 6B	EleutherAI	Jun 21	6B	1x H100 80GB	TensorRT-LLM	FP8	128 / 128	10,907	102	-
GPT-J 6B	EleutherAI	Jun 21	1x A100 80GB	TensorRT-LLM	FP16	128 / 128	3,679	481	-	
Llama 2 70B	Meta	Jul 23	70B	8x H200	MLPerf	-	OpenOrca	35,317	-	-
Llama 2 70B	Meta	Jul 23	8x B200	MLPerf	-	OpenOrca	102,909	-	-	
Llama 2 70B	Meta	Jul 23	Anyscale Platform	-	BF16	550 / 150	66	210	15.15	
Llama	Meta	Jul	Groq	-	-	550	185	130	5.41	

Dimensions 2 and 3: Batching

1. Batch processing queries improves throughput.
2. https://www.anyscale.com/blog/continuous-batching-llm-inference

Dimension 4: Query Token Size and Model Size

Model Name	Release Date	Para Count (Total / Active)	Window
GPT-5	Aug 25	Est. ~52.5T / Unknown [1]	256,000
Llama 4 Scout	Apr 25	109B / 17B [4, 5]	10,000,0
Llama 3.1 405B	Jul 24	405B [6, 7]	128,000
Gemini 2.5 Pro	Mar 25	Est. ~128B [9]	1,048,57
Claude 4.1 Opus	Aug 25	Est. ~400B [11]	0,000 [1
DeepSeek R1	Jan 25	671B / 37B [6]	128,000
Grok 4	Jul 25	Est. ~1.7T [13]	256,000
Mistral Large 2	Jul 24	123B [6, 14]	128,000
Qwen 3 235B	Apr 25	235B [6]	262,000
Phi-3 Medium	Apr 24	14B [6]	128,000

Dimension 5: Data

1. We are out of DATA!

Villalobos, Pablo, et al. “Will we run out of data? an analysis of the limits of scaling datasets in machine learning.” arXiv preprint arXiv:2211.04325(2022)

2. https://the-decoder.com/junk-data-from-x-makes-large-language-models-lose-reasoning-skills-researchers-show/
   “Researchers find that large language models can suffer lasting performance declines when they are continually trained on trivial online content. The study documents sharp drops in reasoning and confidence, raising concerns about the long-term health of LLMs.”

3. I have long predicted that as we cram more bad troll interactions into LLMs they’ll become trolls.

Dimension 5: Data

4. Emergent Misalignment -
   https://www.reddit.com/r/Futurology/comments/1j1a3o9/researchers

"Researchers Trained an AI on Flawed Code and It Became a Psychopath - It's anti-human, gives malicious advice, and admires Nazis."

5. Synthetic Data is not as good, although I am hoping Rusticate style code checking improves things.

Dimension 6: Planning

1. When I started two months ago planning was terrible.
2. But it's getting better with agent specific tools.
3. However, as an old school AI planning researcher, I think there is a ton to do here.
4. But it's a very hard thing to track in practice.
5. And there a quite a few research papers on it.
6. I expect that lots of existing AI work on planning will start to be integrated into LLMs. In fact with multi-agent architectures hidden behind your calls to 'an LLM' who knows what is actually happening.

Dimension 7: Code Quality

1. https://m.slashdot.org/story/448344

Electronic Arts has spent the past year pushing its nearly 15,000 employees to use AI for everything from code generation to scripting difficult conversations about pay. Employees in some areas must complete multiple AI training courses and use tools like the company's in-house chatbot ReefGPT daily.

The tools produce flawed code and hallucinations that employees then spend time correcting. Staff say the AI creates more work rather than less, according to Business Insider. They fix mistakes while simultaneously training the programs on their own work.

Dimension 7: Code Quality

2. This is exactly what Rusticate is trying to fix. And it's been great.
3. Why do these AIs produce such crappy code?
4. Because programmers do and they are trained on LOTs of code.
5. Wadhwa, Nalin, et al. "Core: Resolving code quality issues using llms." Proceedings of the ACM on Software Engineering 1.FSE (2024): 789-811.
6. If you dip into Scholar.google.com it's still a pretty thin literature.
7. So I'm expecting some improvement, but not a lot in the next two years.

Dimension 8: Safety

1. AI is being used to add insecure unauditable code to repositories like Node.js:

Attackers can use Large Language Models (LLMs) to generate malicious code that is obfuscated and difficult to detect by traditional security tools. The “Shai-Hulud” worm, a self-propagating malware seen in recent npm compromises, is suspected to have used AI for generating its scripts, based on coding patterns like specific comments and emoji usage.

2. There are lots of AI safe code projects out there.
3. https://techcrunch.com/2025/08/04/google-says-its-ai-based-bug-hunter-found-20-security-vulnerabilities/ “Google says its AI-based bug hunter found security vulnerabilities”
4. So the Attack/Defend ratio has shifted HUGELY towards attack.
5. This is going to play out over decades, not just in two years.

Dimension 8: Safety

7. And I'm expecting this to push proofs of programs heavily.
8. https://dl.acm.org/doi/pdf/10.1145/263699.263712 "This paper describes proof-carrying code, a mechanism by which a host system can determine with certainty that it is safe to execute a program supplied by an untrusted source."
9. I wish all of the javascript in my browser came with proofs of safety.

Dimension 9: Change

1. “The future ain’t what it used to be” - Yogi Berra
2. These models and techniques are CHANGING at an amazing speed.
3. In two months, I’ve run four different versions of the LLMs.
4. And 8 different versions of the Cursor interface.
5. How do you keep up? It’s very hard.
6. I read slash dot.
7. I read https://the-decoder.com/
8. And I try the new features announced.
9. And I read Bruce Schneier’s excellent blog https://www.schneier.com/.
10. My expectation is a serious change in all of these dimensions every month for two years.

Dimension 10: Independence

1. Claude/Chat-gpt and Cursor all pause all the time!
2. It makes for a lot of AI babysitting.
3. https://www.anthropic.com/news/claude-sonnet-4-5
   “Practically speaking, we’ve observed it maintaining focus for more than 30 hours on complex, multi-step tasks.”
4. I can’t get it to do this at ALL.
5. And Cursor disconnects and has to be told “execute relentlessly.”
6. I’m expecting lots and lots of improvement in the next two years on this.
7. There is a new planning mode in Claude that I’ve only played a little with that might help.
8. While babysitting though, I did get to watch a new Apple TV CS based thriller: https://www.imdb.com/title/tt31186958/ " A post-graduate mathematics student discovers an effort being made to destroy his work in finding a pattern in prime numbers that would allow him to access every computer in the world."
9. https://theaidigest.org/time-horizons

Dimension 11: Copyright and open source

1. https://m.slashdot.org/story/448330

   Does Generative AI Threaten the Open Source Ecosystem?

   "Snippets of proprietary or copyleft reciprocal code can enter AI-generated outputs, contaminating codebases with material that developers can't realistically audit or license properly."
2. If you can't fully audit code in an open source project, then is it safe? No.

APAS Analysis

1. “Algorithms Parallel and Sequential” Acar and Blelloch
2. APAS is:
3. src 238 files 45,462 LOC
4. tests 246 files 55,116 LOC
5. benches 171 files 13,886 LOC,
6. total 655 files 114,464 LOC
7. In about 50 days work so $>$ 2 KLOC a day.
8. A great year for me is about 50 KLOC,
9. and I’ve had few of them (as I mostly manage across the last years).
10. I’ve had some advantages with APAS in that I can jam in textbook chapters.
11. And a huge disadvantage learning Rust and having to write Rusticate to get the code clean.
12. And I’ve had to learn the AIs strengths and weaknesses.

What happens when you try and code APAS?

1. Success - it's fast fast fast
2. Success - it's Git is Awesome and other tools
3. Success - You can jam a textbook into it!
4. Success - teaches you the PL!
5. Problem - Pausing pausing pausing
6. Problem - Non determinism
7. Problem - libraries
8. Problem - types
9. Problem - typeclasses
10. Problem - bad, bad, very bad code
11. Problem - jejune comments
12. Problem - mediocre tests
13. Problem - Please and Cursing
14. Problem - Now you're managing a team
15. Problem - Algorithmic analysis
16. Problem - Cost - manual is expensive
17. Problem - Horrible on perfomance
18. Problem - Horrible on estimating

Coding APAS: Success - it's fast fast fast

1. AI LLMs may write bad code but they write it FAST.
2. But you have to read it (in most cases) so you're still the bottleneck.
3. And unlike the standard code review, the AI's code under review does not buy the traditional donuts and coffee.
4. And the code is often badly structured making it harder for you to review.
5. It's really good at writing test code but misses lots of cases.

Coding APAS: Success - it's Git is Awesome and other tools

1. Claude (and chat-gpt) both fully understand git and github.
2. Which is nice as git is very powerful and rather overly complicated.
3. However at one point rusticate's AI deleted it's local git copy and started running reviews directly on its sibling AI that owned APAS!
4. It makes much nicer checkin logs than you can.
5. And it will quickly answer questions about other tools such as llvm-cov (test function, line coverage) in your programming language.

Coding APAS: Success - You can jam a textbook into it!

1. I have written this by taking APAS chapters and jamming them into Claude and ChatGPT.
2. Chat-GPT writes somewhat better code.
3. But it often gets things wrong: typeclasses, parallelism being the worst.
4. It may take about 20 minutes to write a gnarly chapter's code, with it pausing a few times at least.

What's a typeclass?

1. Defines what should be in a module which you implement.
2. Essentially a type of your module.

```
  /// Flat Hash Table trait - extends ParaHashTableStEphT
pub trait FlatHashTable<Key: StT, Value: StT, Entry: Entr
     ParaHashTableStEphTrait<Key, Value, Entry, Metrics>
  fn probe(table: &HashTable<Key, Value, Entry, Metrics>,
  fn find_slot(table: &HashTable<Key, Value, Entry, Metri
  fn insert_with_probe(table: &mut HashTable<Key, Value,
      let slot = Self::find_slot(table, &key);
       if slot < table.table.len() {....
```

What's an impl (implementation)?

1. Provides the function definitions to implement a trait.
2. Essentially code that you provide that is type checked to match the trait (module type).

```
 impl<Key: StT, Value: StT, Metrics: Default> ParaHashTab
      for DoubleHashFlatHashTableStEph
  {
     fn insert(table: &mut HashTable<Key, Value, FlatEntr
                 key: Key, value: Value) {
```

2. Or just does it without a trait which is sloppier as it can define some of your functions for any type.

```
impl DoubleHashFlatHashTableStEph {
     pub fn second_hash<Key: StT>(key: &Key, table_size:
```

Coding APAS: Success - teaches you the PL!

1. I knew almost no Rust when I started. :)
2. But I could ask my AIs what is this feature?
3. How does this tool work?
4. What packages should I be using?
5. Table their strengths and weaknesses.
6. And it could explain it to me, but see typeclasses.

Coding APAS: Problem - Pausing pausing pausing

1. The biggest single problem is that it pauses. All the time.
2. Claude loves to say: Mission accomplished! without completing TODOs.
3. And Cursor disconnects (load problem?) and you have to continue to tell you AI to continue.
4. I spend days on this problem. I wrote an ExecutionStandard that has the phrases the AI suggested such as:
5. “Drive mode”
6. “Don’t stop until all TODOs are done.”
7. “Don’t ask for reviews”
8. And if you don’t configure Cursor right, it will stop and ask for permission to write to the project.
9. Alas it’s still quite bad.

Coding APAS: Problem - Non determinism

1. One problem that I thought I would run into a lot was non-determinism.
2. But I didn’t much.
3. The core LLM algorithms are deterministic.
4. But their execution on large clusters is not!
5. Plus there are hidden agents doing god knows what.
6. Sometimes, I would have an agent run a ‘manual’ report and try it again later and get a different result.

Coding APAS: Problem - libraries

1. Claude wants to code what Claude wants to code.
2. And it wants to use a whole bunch of standard libraries.
3. So no matter how many times I said: Use the APAS seq data structure here, it would jam vectors in.
4. And Rayon for parallelism. With lots of complicated calls, which is no good for a textbook implementation.

Coding APAS: Problem - types

1. If there was a truly simple 20 character way to fix a type, Claude and Chat-gpt would not get it.
2. They'd start adding random types.
3. Including phantom types and structures with no fields (Mike you want to guess why?)
4. I even wrote a coding standard that said KISS types, but alas the would not get them.
5. I spent a long time getting types right.
6. And Rust's lifetimes and 'static issues really messed them up.
7. But I suppose that's because they confuse Rust programmers.

Coding APAS: Problem - typeclasses

1. Typeclasses are a very nice feature.
2. Philip Wadler and Stephen Blott invented type classes, first proposing them for the Haskell programming language.
3. They introduced the concept as a new approach to ad hoc polymorphism to handle issues like overloading operators and equality checks in a type-safe manner.
4. But Rust's have all sorts of issues.
5. And if there is a simple way to make type classes with inheritence (supertrait) to represent a model, the AI's could not get this.
6. If time permits I'll show you some really horrible Hashtable code and the clean code I had to walk through with the AIs holding my hand explaining superclasses.
7. And when the AIs code the calls and implementations for classes, they go wild with multiple implementations and horrible stubbing and long type disambiguations.

Coding APAS: Problem - bad, bad, very bad code

1. Wow, will it write bad code. :)
2. Strings for enums.
3. Bad numeric types.
4. Lots and lots of unnecessary typing.
5. Horrible modules.
6. It will randomly hallucinate new functions for your traits.
7. Then it will start adding them.
8. It will stub stuff totally unnecessarily.
9. And they will goof around with two types of impls just to make type and function names resolve without ambiguity.
10. The list goes on and on.

Coding APAS: Problem - jejune comments

1. It will write comments like “This function does X”
2. When the function is named X.
3. It will leave “I removed Y” around forever.
4. These are manual fixes as the LLM has to think about the semantics of the comment, so they’re expensive to fix.
5. Coding standards that said no repetitive comments and so on failed to do anything.
6. During test coverage it even commented “Black Magic” on some lines.

Coding APAS: Problem - mediocre tests

1. I spent a very long time trying to get good code coverage.
2. Mike and I both thought (as he mentioned in the last lecture) that these tools should be really good for this.
3. But you ask Claude to write a test module that covers all functions and it will miss many.
4. And they'll jam test code into the source file.
5. I had to write correctors in Rusticate.

Coding APAS: Problem - Please and Cursing

1. Many scurrilous people comment that it takes extra time, $ and energy to be nice to you AI by saying please.
2. However, they forget that you're a human interacting with it and it is good for your emotions.
3. I swear at mine and if you look at their thoughts they say "The user is frustrated that I am not getting X right. The user is correct that I am having problems."
4. So be nice, swear, threaten them, joke with them. :)
5. And I tell mine they may not swear or use language that might offend people as they occasionally use religious language.
6. But my favorite feature of interaction is the LLM understands my misspellings. :)

Coding APAS: Problem - Now you're managing a team

1. You're not longer just programming.
2. You're now up the stack in software engineering.
3. And your undergraduate degree does not necessarily prepare you for this.
4. This is in-part why new junior programmers are not getting as many job offers.

Coding APAS: Problem - Algorithmic analysis

1. APAS has lots of parallel algorithmic analyses in it.
2. I put them in.
3. Then I had Claude see if it agreed.
4. I think it cheated like a lying rug.
5. And I suspect now that APAS is public, your AI will likely cheat on that.

Coding APAS: Problem - Cost - manual is expensive

1. The AIs have a concept of ‘manual’ fixes.
2. These are the AI reading the file (or lines) and thinking deeply about them. :)
3. This is expensive.
4. Which is why I wrote first wrote lots of Python (by my AIs suggestions).
5. Which could easily F*ck up my code base.
6. Which is how I ended up with Rusticate, which is a really entertaining story.

Coding APAS: Problem - Horrible on perfomance

1. If you have your AI write a lot of performance tests, it does OK.
2. But then you end up with tests that are too long.
3. So you ask your AI to work with you to find and fix the long tests.
4. And it then runs the tests, waits, runs them again.
5. You tell it run one at a time, timeout?
6. Well it's no good at real time understanding and wastes a ton of time.
7. I had a livelock and I asked it to run my Mt module with 1 thread. It declared victory when it finished.

Coding APAS: Problem - Horrible on estimating

1. My AIs often estimate 7-12 hours.
2. Then do the task in 20 minutes.
3. And if you ask it to keep track of time and estimates, it can't!
4. And estimating is actually not hard. But requires about a 5% overhead in a software process you'll see in a few slides.
5. But the AIs are not learning this and it's a problem.

Software Processes

1. Which software processes do know of?
2. Which have you been taught?
3. Which do you use?
   a. None?
   b. Waterfall?
   c. Boehm Spiral?
   d. Agile?
   e. The PSP/TSP from the CMU Software Engineering Institute?

Waterfall

1. The basic software process is the waterfall.
2. Phases of the waterfall methodology
    a. Requirements: All project requirements are gathered and documented in detail at the beginning.
    b. Design: The software architecture and detailed design are created based on the initial requirements.
    c. Implementation: The coding phase, where developers write the code according to the design.
    d. Testing (Verification): The software is tested to find and fix bugs and ensure it meets the original requirements.
    e. Deployment: The finished product is released to the customer or users.
    f. Maintenance: Ongoing support and updates are provided after the product is released.

Waterfall - changed

3. We used to do all of these but now, you're going to do more requirements, design, less implementation, very little testing.
4. And lots and lots of code review.

Boehm Spiral

1. https://en.wikipedia.org/wiki/Spiral_model#/media/File:Spiral_mod
2. Basically you do the waterfall like model again and again adding steps at each go round.

Agile

1. https://agilemanifesto.org/principles.html - some nice principles.
2. And it's popular, which is very important.
3. But in it's early days it basically said that you can't estimate.
4. Which is just plain wrong and dangerous.
5. And there nice tools like Jira.

Personal Software Process/Team Software Process

1. Watts S. Humphrey (July 4, 1927 – October 28, 2010) was an American pioneer in software engineering who was called the "father of software quality."
2. Principles:
   a. Improve their estimating and planning skills.
   b. Make commitments they can keep.
   c. Manage the quality of their projects.
   d. Reduce the number of defects in their work.
3. You make: Scripts, Measures, Standards, Forms.
4. Then you estimate: Size, Effort, Quality and Schedule.
5. And do linear regression on those.

Personal Software Process/Team Software Process

1. A very fussy process: how much time does it cost?
2. Only 5% time running the process.
3. How much does it improve your code?
4. 70% reduction in defects going into test.
5. My experience :)
6. There is of course a team version of this.

Dimension 10: Software Process

1. What does AI paired programming do to your software process?
2. Well mostly it ignores processes in favor of brute force coding, but that can't last.
3. But then you have to read the results, fix it.
4. LLMs are pushing you up into the early stages of the waterfall.
5. You're a software architect now.
6. And you're working more and more at the code base level.

Rusticate:

1. https://github.com/briangmilnes/rusticate
2. Def. Rusticate - go to, live in, or spend time in the country or particularly suspend a student from an Oxbridge university as a punishment.
3. Rusticating Python as a method for code review and fix of Rust files in LLMs by using abstract syntax trees in Rust code instead of Python and regexps.
4. Python will be sent back to the family estate for not working well.
5. Summary: src 87 files 23,265 LOC, tests 32 files 2,012 LOC, total 119 files 25,277 LOC
6. And I violated Mike's rule: I did not read a line of code in Rusticate? Why?
7. Because I only care about it's output when run on a codebase. And it checks itself.

The Milnes Problems 1: Coding Standard

1. So my AIs were writing bad code and cargo clippy was not telling me much.
2. Watts Humphrey to the rescue.
3. I'll write a coding standard for Rust and APAS.
4. The AI writes code, I'll ask it to review that file against the standards I thought!
5. Problem solved with:
   a. 198 lines of APASRules.md
   b. 543 RustRules.md

The Milnes Problems 1: What happened?

1. The AI would lie and say it checked all the rules.
2. So I made checklists.
3. The AI would lie and say it checked all of the checklist items.
4. The ‘thought’ was too expensive to review a codebase and missed most of the rules.
5. Being a computer scientist, I thought, I’ll write programs.

The Milnes Problems 2:

1. So I ask the AI how many of my rules from APAS and Rust are programatic?

The Milnes Problems 2:

1. So I ask the AI how many of my rules from APAS and Rust are programatic?
2. 60+% it said.

The Milnes Problems 2:

1. So I ask the AI how many of my rules from APAS and Rust are programatic?
2. 60+% it said.
3. Shall I write you a python program to test these?

The Milnes Problems 2:

1. So I ask the AI how many of my rules from APAS and Rust are programatic?
2. 60+% it said.
3. Shall I write you a python program to test these?
4. 195 python programs later,

The Milnes Problems 2:

1. So I ask the AI how many of my rules from APAS and Rust are programatic?
2. 60+% it said.
3. Shall I write you a python program to test these?
4. 195 python programs later,
5. lots of time watching it screw up regular expressions

The Milnes Problems 2:

1. So I ask the AI how many of my rules from APAS and Rust are programatic?
2. 60+% it said.
3. Shall I write you a python program to test these?
4. 195 python programs later,
5. lots of time watching it screw up regular expressions
6. and 4 traumatic destructions of my code base later.

The Zawinski problem

1. “Some people, when confronted with a problem, think, ‘I know, I’ll use regular expressions.’ Now they have two problems”. – Jamie Zawinski :)
2. All of those python scripts were expensive to develop.
3. And very very brittle.

The Milnes Problems 3: programatic checking of the code

1. So what's the solution?
2. Rust has AST parsers for it's own code.
3. I used ra_ap_syntax.
4. I had it translate some Python to Rust.
5. And build a lot of review-X and fix-X scripts.
6. It tests them on a copy of APAS from the checkin before I introduced the python script!

The Milnes Problems 4: It still codes badly

1. So what went right?

The Milnes Problems 4: It still codes badly

1. So what went right?
2. I taught it to fix over a thousand unecessary (UFCS) calls with MODULE::F or worse '<Type as Trait>::OP' typings.

The Milnes Problems 4: It still codes badly

1. So what went right?
2. I taught it to fix over a thousand unecessary (UFCS) calls with MODULE::F or worse '<Type as Trait>::OP' typings.
3. I taught it to fix bad duplicate and stubbed impls and fixed them.

The Milnes Problems 4: It still codes badly

1. So what went right?
2. I taught it to fix over a thousand unecessary (UFCS) calls with MODULE::F or worse '<Type as Trait>::OP' typings.
3. I taught it to fix bad duplicate and stubbed impls and fixed them.
4. But it immediately started string hacking inside of Rusticate programs!

The Milnes Problems 4: It still codes badly

1. So what went right?
2. I taught it to fix over a thousand unecessary (UFCS) calls with MODULE::F or worse '<Type as Trait>::OP' typings.
3. I taught it to fix bad duplicate and stubbed impls and fixed them.
4. But it immediately started string hacking inside of Rusticate programs!
5. I would say (hundreds of times) Are you string hacking? You're generating unmatched delimiters.

The Milnes Problems 4: It still codes badly

1. So what went right?
2. I taught it to fix over a thousand unecessary (UFCS) calls with MODULE::F or worse '<Type as Trait>::OP' typings.
3. I taught it to fix bad duplicate and stubbed impls and fixed them.
4. But it immediately started string hacking inside of Rusticate programs!
5. I would say (hundreds of times) Are you string hacking? You're generating unmatched delimiters.
6. It would literally say: Oh you caught me. I'm string hacking. Would you like me to do this fully in an AST?

The Milnes Problems 4: It still codes badly

1. So what's the solution?

The Milnes Problems 4: It still codes badly

1. So what's the solution?
2. I made rusticate make a string hack checker and ran it on all of it programs.

The Milnes Problems 4: It still codes badly

1. So what's the solution?
2. I made rusticate make a string hack checker and ran it on all of it programs.
3. I made a Claude memory saying: DO NOT F*CKING HACK STRINGS IN RUSTICATE.

The Milnes Problems 4: It still codes badly

1. So what's the solution?
2. I made rusticate make a string hack checker and ran it on all of it programs.
3. I made a Claude memory saying: DO NOT F*CKING HACK STRINGS IN RUSTICATE.
4. Didn't help. Only the programatic check did.

The Milnes Problems 5: Can we train on Rusticate?

1. What can't you do with Rusticate?
2. Well it can't fully understand how to check for uncovered functions because it does not have enough infomation.
3. Ideally the Rust compiler would put out typed ASTs for you and rusticate would work on them.
4. And your AI would be trained on a lot of these with textual syntax that helped it understand them.
5. The and only then would it likely stop spitting out junior programming crud, but I bet it would still have problems.
6. There are some efforts to train LLMs only on 'good' code.

Test coverage

1. Test coverage should be easier.
2. Last week even Mike thought it should be easier.
3. But it's still expensive manual in the defacto brute force process.
4. So I had rusticate make a test coverage tool.
5. But it kept giving false positives and false negatives. Why?
6. Because it had infer the types of many methods with similar names.

Test coverage

6. What's a solution to this?
7. Typed AST analysis; why redo what the compiler did.
8. So I had it read the llvm-cov HTMLs that show line of code coverage and told it to fix those lines in a simple module.

Test coverage

1. It said: “STILL 84.62%!!! This is unbelievable. Even with the full test suite, these 4 functions are not being hit. At this point, I think the problem is that these functions are genuinely dead code - they exist in the codebase but are never actually called by anything. This could be..”
2. And “Houston We Have a Problem!”
3. file:///home/milnes/APASVERUS/APAS-AI/apas-ai/target/llvm-cov/html/index.html

Programming Languages Ups and Downs

1. Let's quickly go through the major programming languages
2. And their good and bad points!

C

1. Fast to compile.
2. Fast to run.
3. Very poor modularity.
4. Hand allocation/deallocation.
5. A security nightmare.
6. Inexpressive types.
7. Dan Grossman's thesis improvements, Cyclone, never caught on.

Python

1. Slow
2. Very strange dynamic typing.
3. Object Oriented
4. REPL and compiler available.
5. But I do sort of like the fixed line format.

Java

1. The Java programming language is type safe.
2. It does not have a particularly expressive type system.
3. It uses Objects for the vast amount of its modularity.
4. Objects are great for user interfaces.
5. Objects are pretty poor for modularity. :)
6. For Phil Bernstein's DB class it took me 8 hours to find a class structure that really worked to allow four different lock managers. In SML, it would have been 20 minutes.

Java insanity: When was the Lambda invented?

1. The first programming language to introduce lambda functions was?

Java insanity: When was the Lambda invented?

1. The first programming language to introduce lambda functions was?
2. Lisp.

Java insanity: When was the Lambda invented?

1. The first programming language to introduce lambda functions was?
2. Lisp.
3. When?

Java insanity: When was the Lambda invented?

1. The first programming language to introduce lambda functions was?
2. Lisp.
3. When?
4. in 1958.

Java insanity: When was the Lambda invented?

1. The first programming language to introduce lambda functions was?
2. Lisp.
3. When?
4. in 1958.
5. Java was released in 1995.

Java insanity: When was the Lambda invented?

1. The first programming language to introduce lambda functions was?
2. Lisp.
3. When?
4. in 1958.
5. Java was released in 1995.
6. Did it have lambdas?

Java insanity: When was the Lambda invented?

4. When did they put in lambdas?

Java insanity: When was the Lambda invented?

4. When did they put in lambdas?
5. Before Java 8, the functionality closest to a lambda expression was using anonymous inner classes to implement single-method interfaces, 2011.

Java insanity: When was the Lambda invented?

4. When did they put in lambdas?
5. Before Java 8, the functionality closest to a lambda expression was using anonymous inner classes to implement single-method interfaces, 2011.
6. Lambda expressions were introduced in Java with the release of Java 8 in March 2014.

Java insanity: When was the Lambda invented?

4. When did they put in lambdas?
5. Before Java 8, the functionality closest to a lambda expression was using anonymous inner classes to implement single-method interfaces, 2011.
6. Lambda expressions were introduced in Java with the release of Java 8 in March 2014.
7. Lambdas in logic were invented in the 1930s by Alonzo Church.

Java insanity: When was the Lambda invented?

4. When did they put in lambdas?
5. Before Java 8, the functionality closest to a lambda expression was using anonymous inner classes to implement single-method interfaces, 2011.
6. Lambda expressions were introduced in Java with the release of Java 8 in March 2014.
7. Lambdas in logic were invented in the 1930s by Alonzo Church.
8. So it took 56 years for Java to put in the very useful concept of an anonymous function, and 19 years from it initiation.

Just for History: What did Church and Turing do?

1. Church invented the mathematical model of programming: the Lambda Calculus.
2. Turing invented the physical, yet abstract, module of computation: the Turing machine.
3. Turing predicted AI.
4. A great book on the subject is: “Turing’s Cathedral: The Origins of the Digital Universe by George Dyson”
5. He was given full access to the Princeton’s Institute for Advanced Studies archive to write this.
6. What’s his second most cited paper?
7. The chemical basis of morphogenesis AM Turing - Bulletin of mathematical biology, 1990 (reprinted).
8. Basically, it’s the model of how could Zebras make stripes.

Just for History: OOP

1. Who invented object oriented programming?

Just for History: OOP

1. Who invented object oriented programming?
2. Ivan Sutherland

Just for History: OOP

1. Who invented object oriented programming?
2. Ivan Sutherland
3. Sketchpad is a computer program written by Ivan Sutherland in 1963 in the course of his PhD thesis, for which he received the Turing Award in 1988, and the Kyoto Prize in 2012. It pioneered human–computer interaction (HCI),[2] and is considered the ancestor of modern computer-aided design (CAD) programs and as a major breakthrough in the development of computer graphics in general. For example, Sketchpad inspired the graphical user interface (GUI) and object-oriented programming.

Just for History: OOP

1. Who invented object oriented programming?
2. Ivan Sutherland
3. Sketchpad is a computer program written by Ivan Sutherland in 1963 in the course of his PhD thesis, for which he received the Turing Award in 1988, and the Kyoto Prize in 2012. It pioneered human–computer interaction (HCI),[2] and is considered the ancestor of modern computer-aided design (CAD) programs and as a major breakthrough in the development of computer graphics in general. For example, Sketchpad inspired the graphical user interface (GUI) and object-oriented programming.
4. What is it good for? User interfaces and real world modelling.

Just for History: OOP

1. Who invented object oriented programming?
2. Ivan Sutherland
3. Sketchpad is a computer program written by Ivan Sutherland in 1963 in the course of his PhD thesis, for which he received the Turing Award in 1988, and the Kyoto Prize in 2012. It pioneered human–computer interaction (HCI),[2] and is considered the ancestor of modern computer-aided design (CAD) programs and as a major breakthrough in the development of computer graphics in general. For example, Sketchpad inspired the graphical user interface (GUI) and object-oriented programming.
4. What is it good for? User interfaces and real world modelling.
5. What is it truly mediocre for: modularity of programs (Java).

Garbage Collection

1. Zorn, Benjamin. “The measured cost of conservative garbage collection.” Software: Practice and Experience 23.7 (1993): 733-756.
2. They mostly run in about the same time as allocate and free or a small overhead.
3. What is right with GC?
   a. They don’t make errors
   b. They support type safe programming.
4. What is wrong with GC?
   a. Collections can cause delays.
   b. They can cause you to fail to see you’re holding storage, but so can explicit allocation/deallocation.
   c. GC’d languages don’t drop into a Kernel.

Rust Rughts: Linear Logic, parallelism

1. Rust has a nice linear logic in it (as do many other languages) with borrowing.
2. They just guessed though, but got it right.
3. Reed, Eric. "Patina: A formalization of the Rust programming language." University of Washington, Department of Computer Science and Engineering, Tech. Rep. UW-CSE-15-03-02 264 (2015).
4. And they have all the parallel and reference counting types and operations you want.

Rust Rughts: It drops into kernels

1. Rust compiles to fast code.
2. Rust compiles to safe code.
3. Rust can break this with explicit unsafe code.
4. Rust is popular.

Rust Wrungst: The 90 terms

1. In APAS you'll find a docs/RustTermsCorrected.md.

Rust Wrungst: The 90 terms

1. In APAS you'll find a docs/RustTermsCorrected.md.
2. I had Chat GPT and Claude record every 'technical' term used in the Rust documentation and their real programming language technical equivalent.

Rust Wrungst: The 90 terms

1. In APAS you'll find a docs/RustTermsCorrected.md.
2. I had Chat GPT and Claude record every 'technical' term used in the Rust documentation and their real programming language technical equivalent.
3. About 94 terms are used in the Rust documentation.

Rust Wrungst: The 90 terms

1. In APAS you'll find a docs/RustTermsCorrected.md.
2. I had Chat GPT and Claude record every 'technical' term used in the Rust documentation and their real programming language technical equivalent.
3. About 94 terms are used in the Rust documentation.
4. Including some reall stinkers: blanket, trigger fish and so on.

Rust Wrungst: The 90 terms

1. In APAS you'll find a docs/RustTermsCorrected.md.
2. I had Chat GPT and Claude record every 'technical' term used in the Rust documentation and their real programming language technical equivalent.
3. About 94 terms are used in the Rust documentation.
4. Including some reall stinkers: blanket, trigger fish and so on.
5. How many do you think are the actual common programming language terms?

Rust Wrungst: The 90 terms

1. In APAS you'll find a docs/RustTermsCorrected.md.
2. I had Chat GPT and Claude record every 'technical' term used in the Rust documentation and their real programming language technical equivalent.
3. About 94 terms are used in the Rust documentation.
4. Including some reall stinkers: blanket, trigger fish and so on.
5. How many do you think are the actual common programming language terms?
6. 4! Just 4 that I could find.

Rust Wrungst: The 90 terms

1. In APAS you'll find a docs/RustTermsCorrected.md.
2. I had Chat GPT and Claude record every 'technical' term used in the Rust documentation and their real programming language technical equivalent.
3. About 94 terms are used in the Rust documentation.
4. Including some reall stinkers: blanket, trigger fish and so on.
5. How many do you think are the actual common programming language terms?
6. 4! Just 4 that I could find.
7. This makes Rust uncessarily hard to learn. :)

Rust Wrungst: Type Inference is really bad!

1. Type inference is?
2. Type inference in Rust just plain bites.
3. And it's so complicated that the AIs just put in type annotations left and right.
4. And even clippy (the rust linter) won't tell you what can go away.
5. You put in "<Type as Trait>::OP" all the times.
6. Your typeclass (modulelete) says what's public but type inference is so bad people go "use package::module::{F1, F2, ...}". So you're repeating declarations and have to adjust clients when a module adds new code.

ML/OCaml

1. The ML and now OCaml family of languages have many advantages.
2. Formal semantics (well not for all of OCaml).
3. Clean polymorphism.
4. Good Abstract Data Types.
5. Great modules to structure things including Functors (modules to module mappings).
6. ML has Talpin Tofte regions: like rust lifetimes but you can just allocate and unroll large amounts of storage.
7. OCaml even has objects.
8. Hindly Milner Type inference is really good. You don't need many type annotations.

ML modules and type inference

1. When was ML invented?
2. Through the 1980s.
3. When were modules, polymorphism, type inference and formalized?
4. All by 1990.
5. Why don't we have this in all programming languages?
6. Damn good question that.
7. "The old magic is forgotten, which is particularly bad when it's your old magic Bob." -Personal communication to Robert Harper

Speed and Garbage Collection

1. SML is not very fast in benchmarks.
2. OCaml though is:

"OCaml is basically as fast as C. Time for a powerful, high-level, type-safe language (i.e. not C or C++) that's truly fast. Strong static typing with type inference."

Call for Type Safe Programming Languages

1. The NSA wants us to code better! :)
2. https://www.nsa.gov/Press-Room/Press-Releases-Statements/Press-Release-View/article/3608324/us-and-international-partners-issue-recommendations-to-secure-software-products/#:~:text=In%20a%20shared%20conclusion%2C%20the,and%
3. Basically compilers, types and proofs are your friends.

Dimension 11: Better PLs

1. So what do we need for better AI programming?
2. We need better programming languages. :)
3. Type safe,
4. ADTs
5. Polymorphism
6. clean modules
7. typeclasses for ad hoc polymorphism
8. And linear logics with borrowing
9. Regions for fast bulk deallocation
10. GC for when it's needed.
11. C speed compilation
12. Unsafe operations (for kernels, but with proof!).
13. Support for parallelism.

Dimension 12: Proof

1. But due to AIs shifting the Attack/Defense ratio heavily in favor of attacks,
2. We are going to need more proven software.
3. Proven OSs.
4. Proven libraries
5. Proven protocols.
6. Proving tools.
7. And AIs that make this cheaper.

Proof

1. There are still many questions as to which proof techniques are going to work best.
2. Cost is an issue with estimates of 6-8 fold for proven software.
3. How many of you are using proven code on a daily basis?
4. All of you!
5. How important is proof versus just type safe parsers?
6. The estimates vary but type safety would probably cut out about 50% of intrusions.
7. And Proof might take that closer to 90%.

SMTs

1. Satisfiability modulo theory theorem provers.
2. Z3 is widely used.
3. Lean is widely used, paricularly in formalizing mathematics.
4. They give you a very certain feel that a theorem is true,
5. but they don't give you a proof.

Proof Assistants

1. There are quite a few proof systems for programs now:
2. Isabell/HOL - a higher order logic proof system.
3. F* - dependently typed rich language using SMT proofs and refinement typing with an emphasis on domain specific languages.
4. Rocq (formerly Coq :)) - an interactive theorm prover based on a different logic, the calculus of constructions.
5. Verus - a new Rust based proof system using simpler logics, an SMT and focusing on parallelism and sequences using special purpose proof procedures.
6. HOL4 - a higher order logic proofs of program system.
7. And there are new logics out there that solve some of the worst problems with type systems called cubical type theories.

Successes: SeL4

1. SeL4 is both the world's most highly assured and the world's fastest operating system kernel. Its uniqueness lies in the formal mathematical proof that it behaves exactly as specified, enforcing strong security boundaries for applications running on top of it while maintaining the high performance that deployed systems need. -From their marketing.
2. Klein, Gerwin, et al. "seL4: Formal verification of an OS kernel." Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. 2009.
3. Proven in Isabelle proof system using Higher Order Logic (HOL).
4. Where is it used in your life?
5. It is probably running on the signal processor in you phone.
6. Why? The phone makers got sick of 'drive-by' attacks coopting your phone.

Successes: CompCert

1. CompCert is a formally verified optimizing compiler for a large subset of the C99 programming language (known as Clight) which currently targets many architectures.
2. Proven in Rocq.
3. Leroy, Xavier, et al. "CompCert-a formally verified optimizing compiler." ERTS 2016: Embedded Real Time Software and Systems, 8th European Congress. 2016.
4. A big success, like SeL4.

Successes: CakeML

1. There is even a proven ML compiler.
2. Kumar, Ramana, et al. “CakeML: a verified implementation of ML.” ACM SIGPLAN Notices 49.1 (2014): 179-191.
3. M. O. Myreen and S. Owens. Proof-producing synthesis of ML from higher-order logic. In ICFP, 2012
4. Proven in HOL4.

Successes: EverCrypt

1. And there is a large, 100 KLOC, very functional proven cryptography library from the F* group.
2. Protzenko, Jonathan, et al. “Evercrypt: A fast, verified, cross-platform cryptographic provider.” 2020 IEEE Symposium on Security and Privacy (SP). IEEE, 2020.
3. It generates C or Rust proven libraries using a domain specific language sublanguage (although they have several).
4. The newest one uses a much simpler separation logic (do you know what that is?).
5. EverCrypt is used in production systems like Mozilla Firefox, the Linux kernel, Wireguard VPN and Signal.

Successes: EverParse and the like

1. Ramananandro, Tahina, et al. “{EverParse}: Verified secure {Zero-Copy} parsers for authenticated message formats.” 28th USENIX Security Symposium (USENIX Security 19). 2019.
2. There are Rocq projects that also emit proven parsers.
3. How important are these? It’s a bit hard to say but estimates are that parsing issues are a huge percentage of intrusion bugs.
4. And EverParse is in at least six places in the MSFT production tool chain. -Personal communication from Nik Swamy.

Difficulties

1. The difficulty is mostly the human cost in time to prove these systems.
2. Some proof check times, like EverParse, started out at nearly 50 minutes.
3. Making it hard to put them into standard tool chains.
4. But now are down to 6 minutes for their newest parser generator.

AI Proof - F*

1. And of course, who wants to prove all those tedious theorems by hand.
2. AI proof assistance is coming along very nicely.
3. The CTO of the company Galois CTO said: I have spent my life hand building a Ferrari and I just got passed by a Safeway shopping cart.
4. For example, Towards Neural Synthesis for SMT-Assisted Proof-Oriented Programming (ICSE 2025), which curates a dataset of 940KLOC of F* code and proof and develops AI models to automate program and proof synthesis
5. They get a lot of easy proofs out of this, and they're just beginning.
6. And have released a huge corpus of proofs for others to train on.

What's next for me? Verus - how much can I prove?

1. Well next, I'll start trying to prove some (or all) of the algorithms in APAS in Verus.
2. This should make it easy to expand APAS to be a proven algorithms textbook.
3. There may not be enough LOCs of Verus to get good AI proof assistance though.
4. Wish me luck!

Summary

1. So you've seen an overview of a real (academic) project of serious scale.
2. And you've seen the pros and the cons of the current AI paired programming models.
3. And there are lot of Cons.
4. And you have seen a tool created to help with the worst of those cons: Rusticate.
5. And you've seen predictions about the change in these tools: and that change is huge and coming fast like a freight train.

Summary

6. And that future needs to have:
   a. Better Programming languages - so that Mike has a job.
   b. And proven software - so you have a job.
7. I think you'll all be AI paired programming and proving in your careers.
8. As Mike has such a high teaching load, I might even come back and update you on using Verus and AIs to prove APAS. :)
9. But I think the most important thing that you can take from this lecture is: AI paired programming is FUN.

Projects

1. The obvious projects to propose, that are not too big, if Mike is looking for them are:

a. Rusticate C

b. Rusticate Java

c. Rusticate Python

d. Write some APAS algorithms in another language?

e. Run all of the algorithmic analyses again on the code from a clean slate (no APAS comments, no reading the textbook) and have Claude analyze every function in every trait. And then put them back in and see if Claude got them right. But remember, once APAS is released AIs will train on it and they might cheat! :)

What happened to the HashTable chapter?

1. The first code was terrible, let me show you.
2. /home/milnes/APASVERUS/APAS-AI/apas-ai/attic/Chap47/AdvancedDoubleHashing.rs
3. /home/milnes/APASVERUS/APAS-AI/apas-ai/attic/Chap47/

Funny things AIs say:

1. You caught me, I'm cheating.
2. Robert Harper Milnes. :)
3. I said "Think like Bob Harper in Types." It screwed up some more code and said "I have failed the Harper test." :)
4. I ran Codex vs Claude on a task. I told Codex not to look at Claude's code, but it did. This is like looking into the soul of the student next to you during your religion class final.
5. Wait test coverage dropped. I overwrite tests! Bad move.
6. Oh wait, I deleted the copy of APAS-AI?
7. Oh wait, I checked into APAS-AI let me revert that.
8. cargo clippy has 450 warnings! LOL.