

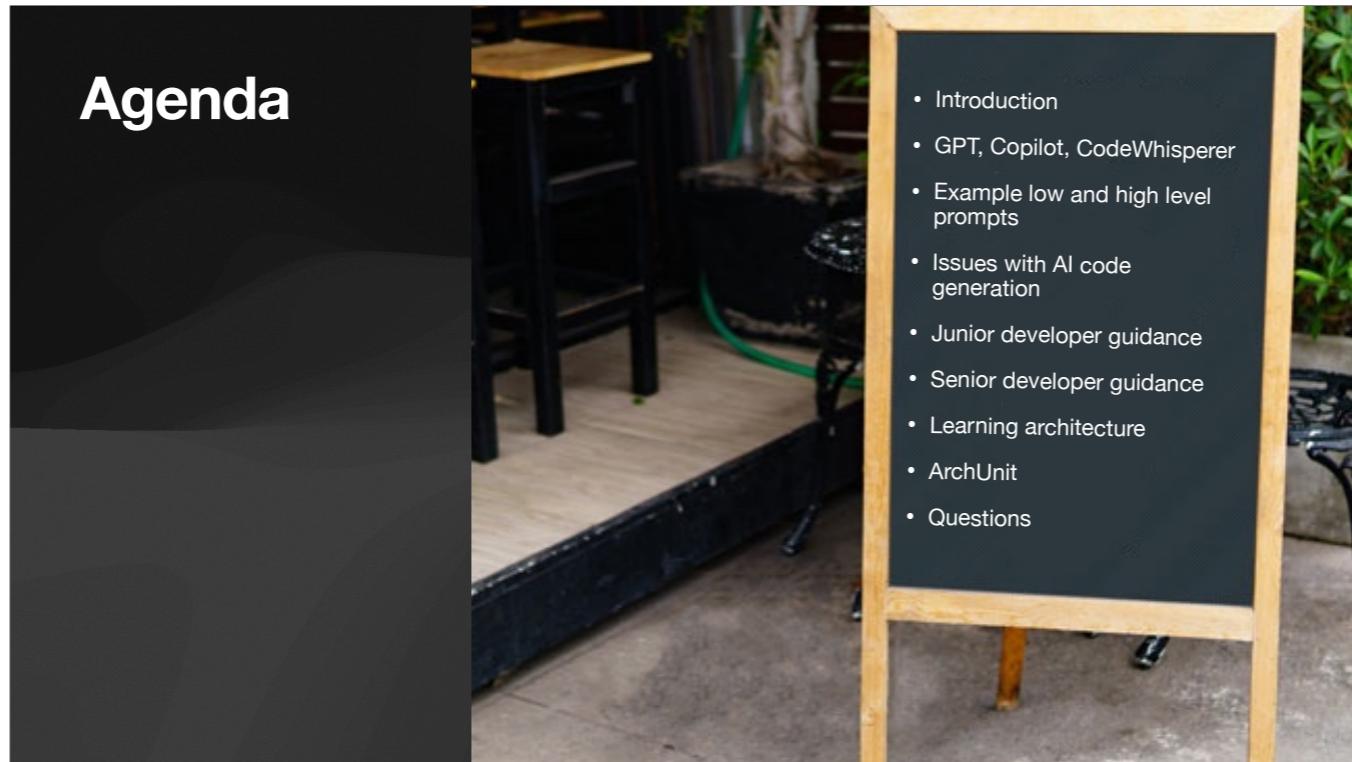
AI Proof Your Career With Software Architecture

Kelly Morrison

Hello, and welcome to “AI Proof Your Career With Software Architecture”!

Agenda

- Introduction
- GPT, Copilot, CodeWhisperer
- Example low and high level prompts
- Issues with AI code generation
- Junior developer guidance
- Senior developer guidance
- Learning architecture
- ArchUnit
- Questions



Since there are so many great presentations happening simultaneously, here's a quick overview of what I'll be covering today. If you're still interested, great! If not, feel free to switch to a different talk. You won't hurt my feelings. Just leave a couple of dollars at the door to pay my therapist to deal with my abandonment issues. :) Just kidding!

Kelly Morrison, Ph.D. Computer Engineering

Solution Architect - Daugherty Business Solutions



To give you a little background, my name is Kelly Morrison. I'm a consultant at Daugherty Business Solutions, which is a great company - we have a booth here at the conference if you'd like to come by and see what it takes to join us.

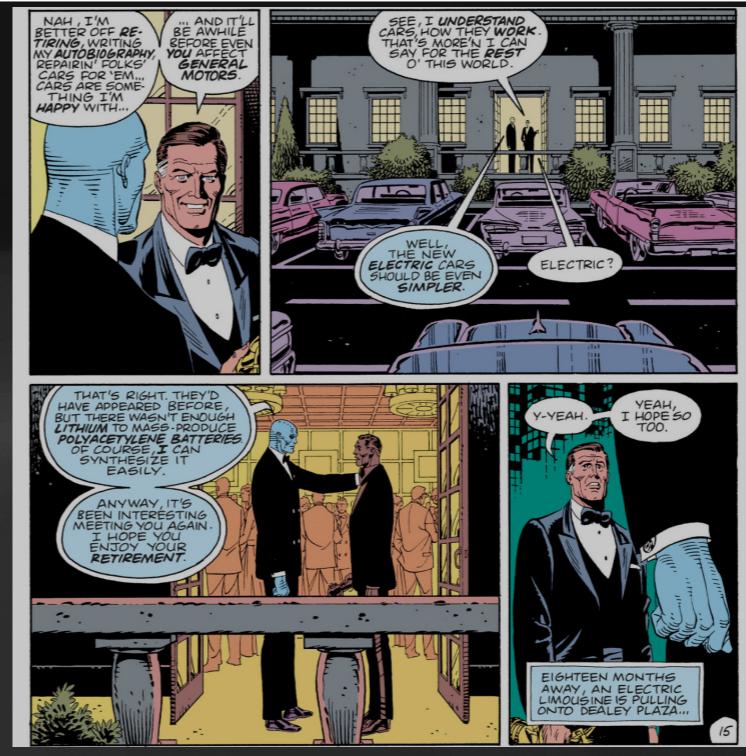
I have a bachelor's degree in mathematics from the University of Georgia (GO DAWGS!!!) and a doctorate from Auburn University. I've worked for many companies over the years - you can see a sampling of them here.



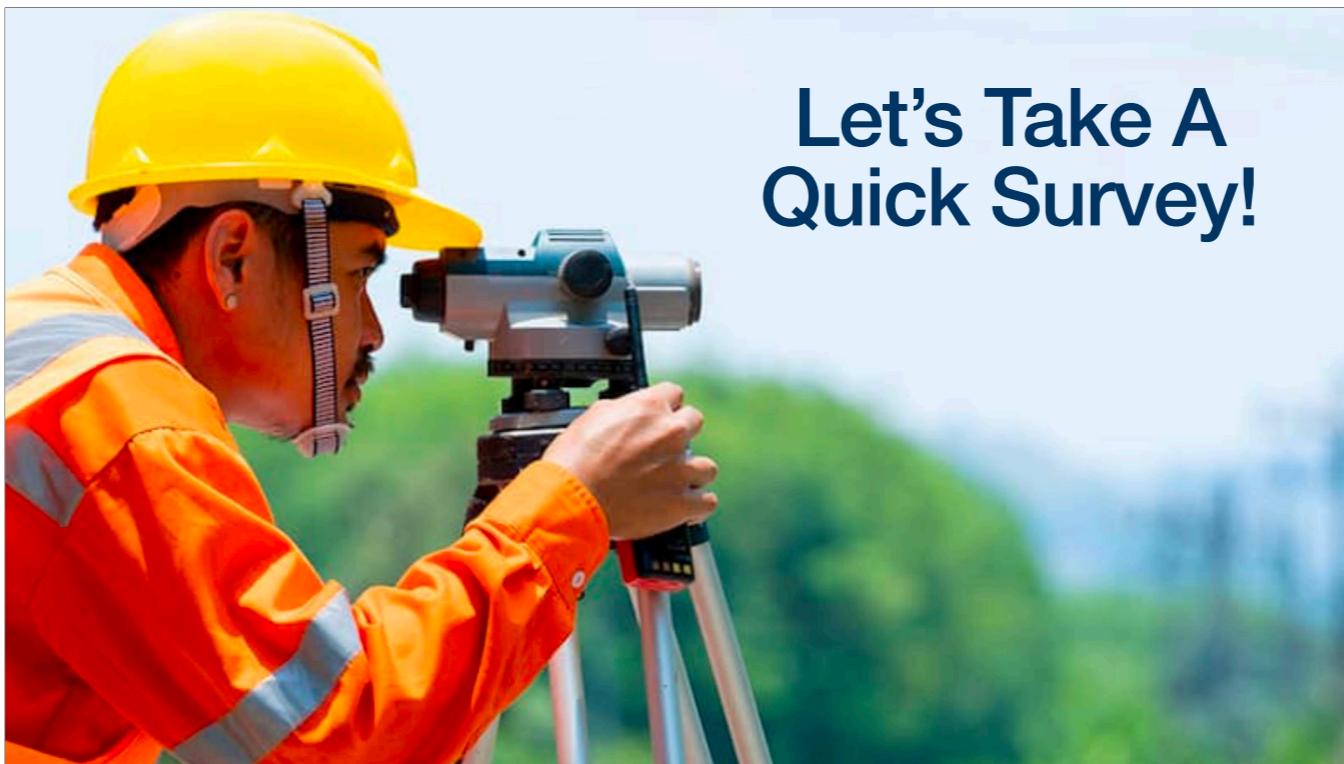
I put together this image to show how I felt about AI assisted coding tools when ChatGPT and Copilot first came out. I felt like Rocky Balboa, trapped in a ring with a mechanical opponent who never gets tired, never gets hurt, and will keep hitting me until I'm finished.

Watchmen

Alan Moore
1986



I also felt like this panel from Alan Moore's "Watchmen", 38 years ago. In the story, a superhero named Dr. Manhattan appeared and changed pretty much everything: politics, war, etc. Another superhero (Nite Owl) told him that he'd replaced him, so he was going to retire and work on cars, because even Dr. Manhattan couldn't affect that for years. But then Dr. Manhattan said that he could synthesize lithium easily, paving the way for new (and simpler) electric cars to be produced. I never forgot how Nite Owl suddenly realized his entire world had changed in the blink of an eye.

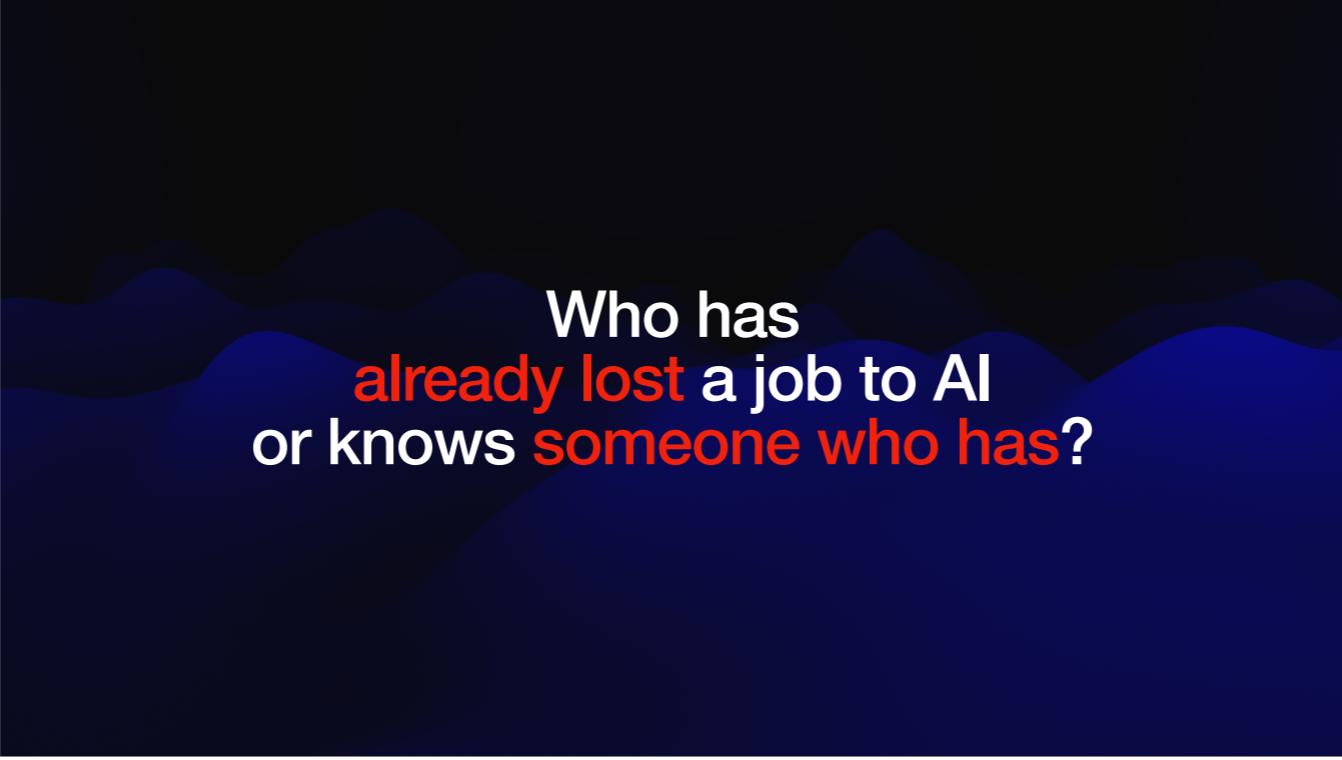


**Let's Take A
Quick Survey!**

Before we get started, let's take a quick survey! Just two questions.



Who is afraid
they'll lose their job
to AI?

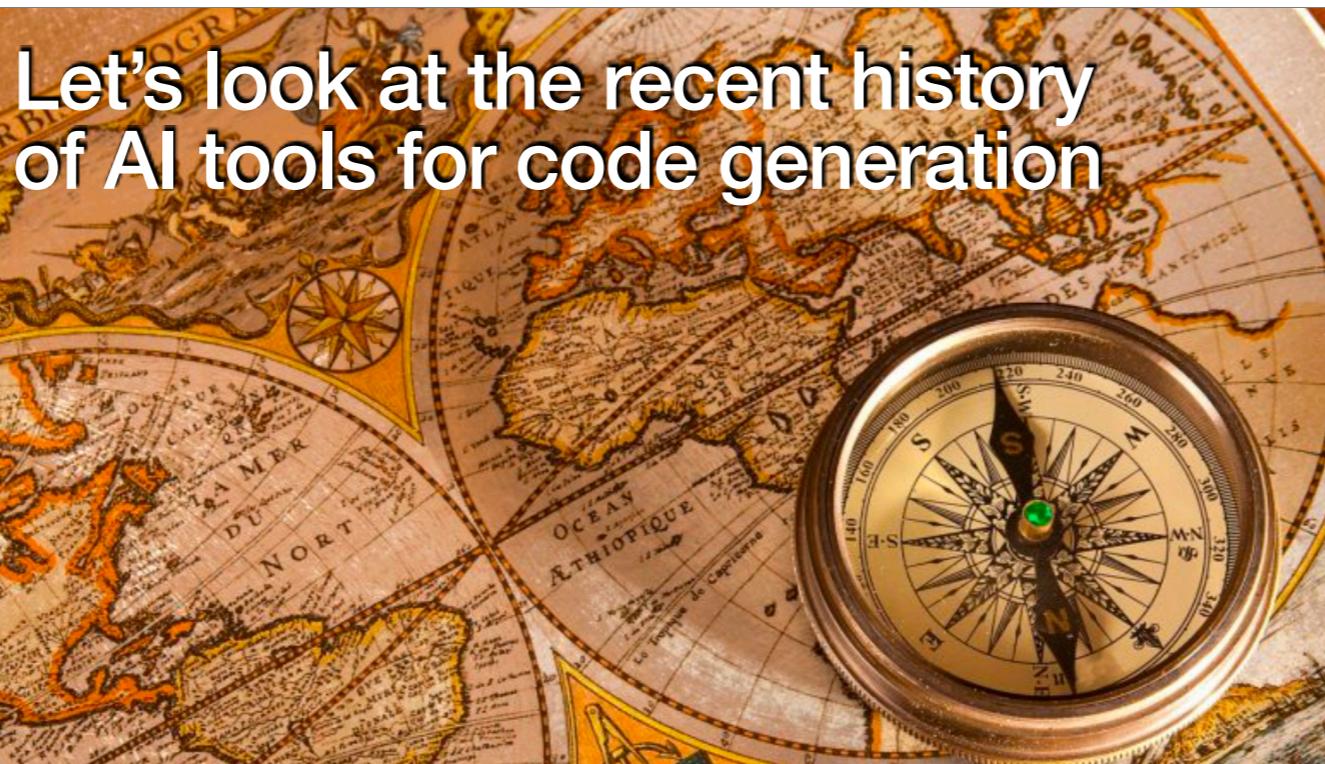


Who has
already lost a job to AI
or knows someone who has?



Let me reassure you!

For those of you who are worried, I hope that my talk today reassures you. And don't worry - I won't try to hug you. :)



Let's look at the recent history of AI tools for code generation

Let's start out by looking at the recent introduction of AI tools for code generation.

GPT - Generative Pre-training Transformer

- An LLM (Large Language Model)
 - Modern LLMs became popular in 2017
 - GPT was created by OpenAI in 2018
 - Parameters are variables used to infer new content
 - GPT-2 had 2 billion parameters
 - GPT-3 had 175 billion parameters
 - GPT-4 has 1 trillion parameters

The AI field exploded in 2017 with a paper called “Attention Is All You Need” that introduced the concept of transformers, the “secret ingredient” behind the improvements in large language models. GPT in 2018 changed everything. A look at the number of “parameters” in the models gives an idea of how fast they have progressed - from 2 billion in GPT-2 to a trillion in GPT-4.

GitHub Copilot

- Released in 2021
- Collaboration between OpenAI and Microsoft/GitHub
- Uses Codex, a specialized version of GPT-3 for creating code
- Trained on billions of lines of code from GitHub
- Can learn from a local code base

GitHub Copilot was introduced in 2021. Microsoft, which had acquired GitHub and had a partnership with OpenAI, realized that they had access to billions of lines of code, so they created a specialized version of GPT-3 called “Codex” that was trained on the code in GitHub. It was integrated into a tool called “Copilot” that suggests and writes code for programmers based on simple text requests.

Amazon CodeWhisperer

- Introduced in 2022
- Trained on billions of lines of code
- Supports code generation for 15 languages
- Specialized for AWS cloud development
- Looks at code and comments and then offers help

Not to be outdone, Amazon came up with their own LLM (Large Language Model) tool called CodeWhisperer. Like Microsoft, Amazon had access to billions of lines of code, so they trained a special model for AWS cloud development.



So: ChatGPT, Copilot, CodeWhisperer (and others): how good, are they really? And do programmers need to be afraid they'll be replaced?

A ChatGPT example - Generating a web service



show me how to write a Java 17 Spring Boot REST API for storing and retrieving baseball statistics in a MongoDB database. Include JUnit 5 test cases that check for the 5 most common REST errors.

Let's look at two examples, one low level and one high level. Here's a simple request to write a basic REST API for tracking baseball statistics in a MongoDB database. You can see the prompt I used. Notice the last sentence - we'll come back to that later.

The result

3. Define the MongoDB Configuration:

Create a `MongoDBConfig` class to configure the connection to your MongoDB database.

```
java Copy code
import org.springframework.context.annotation.Configuration;
import org.springframework.data.mongodb.config.AbstractMongoClientConfiguration;

@Configuration
public class MongoDBConfig extends AbstractMongoClientConfiguration {

    @Override
    protected String getDatabaseName() {
        return "baseball_stats_db";
    }

    @Override
    public MongoClient mongoClient() {
        return MongoClients.create("mongodb://localhost:27017");
    }
}
```

4. Create a Baseball Stat Entity:

Define the model for baseball statistics, e.g., `BaseballStat.java`.

```
java Copy code
import lombok.Data;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "baseball_stats")
@Data
public class BaseballStat {
    @Id
    private String id;
    private String playerName;
    private int homeRuns;
    private int runsBattedIn;
    // Add more fields as needed
}
```

After a few seconds, ChatGPT came back with a code solution. At first, I thought “OMG - it wrote the whole program for me!” But after some analysis, I noticed issues. For example, it hard coded configuration in the code itself - in the “real world”, this would be read from an external source. In the BaseballStat code, it used the Lombok library (which is fairly controversial - it used to be amazing, but it’s falling out of favor as Java improves. And remember I specified “Java 17”? I was hoping it would use the Java 16 records, but instead, it used a regular class with a Lombok annotation.

The screenshot shows a Java code editor with two code snippets. The left snippet is titled "5. Create a REST Controller:" and the right snippet is titled "6. Implement REST Endpoints:". Both snippets are in Java and demonstrate basic CRUD operations for a "BaseballStat" entity.

5. Create a REST Controller:

```
java
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
@RequestMapping("/api/baseball-stats")
public class BaseballStatController {
    // Implement your CRUD operations here
}
```

6. Implement REST Endpoints:

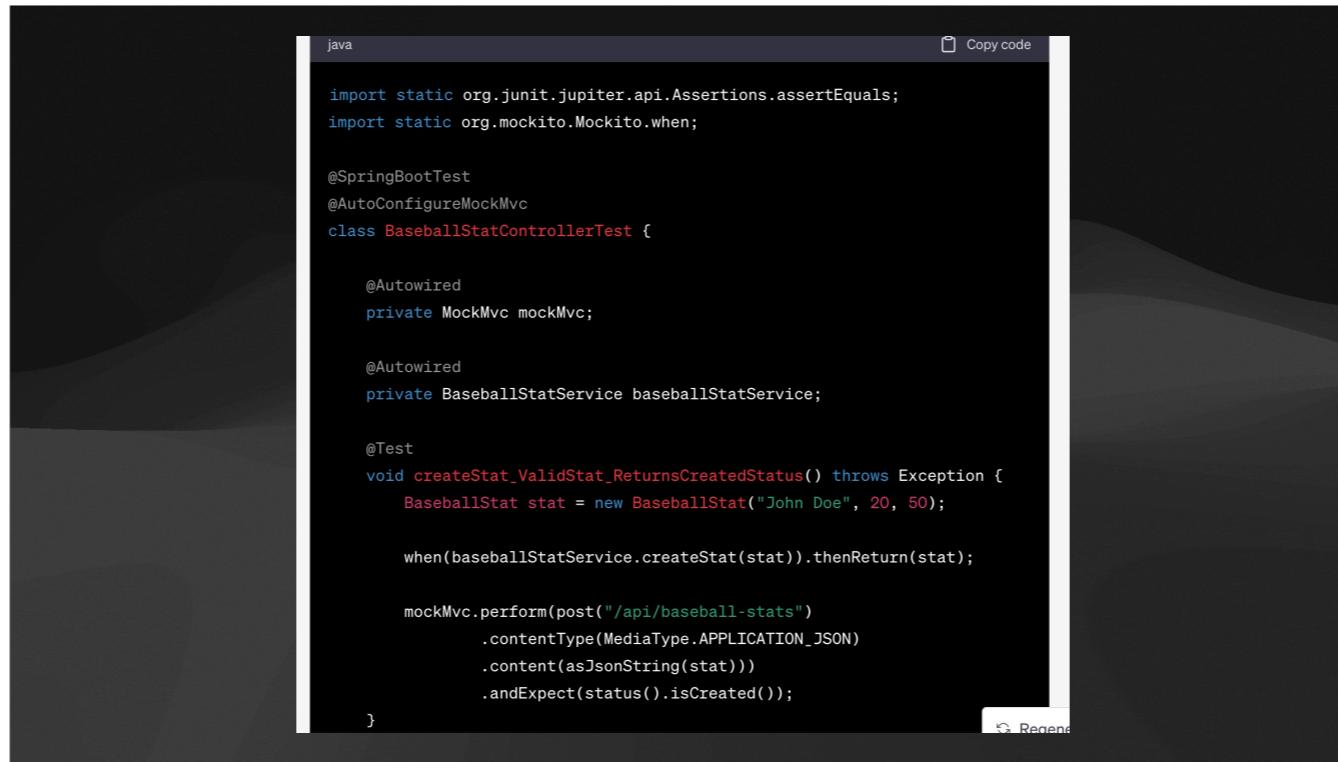
```
java
@PostMapping
public ResponseEntity<BaseballStat> createStat(@RequestBody BaseballStat stat) {
    // Implement logic to create a new stat
}

@GetMapping("/{id}")
public ResponseEntity<BaseballStat> getStat(@PathVariable String id) {
    // Implement logic to retrieve a stat by ID
}

@PutMapping("/{id}")
public ResponseEntity<BaseballStat> updateStat(@PathVariable String id, @RequestBody BaseballStat stat) {
    // Implement logic to update a stat
}

@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteStat(@PathVariable String id) {
    // Implement logic to delete a stat
}
```

The REST controller was short, but as an architect, I don't like the URL they generated - I usually add versioning so that multiple versions of an API can be supported simultaneously. And ChatGPT only did the minimum work for the REST endpoints - there is no return code in any of them, so this won't even compile! Would it have hurt them to at least show how to return an "unimplemented" response?



A screenshot of a Java code editor displaying a JUnit test class named `BaseballStatControllerTest`. The code uses Mockito for mocking the `baseballStatService` and Spring Boot annotations like `@SpringBootTest` and `@AutoConfigureMockMvc`. It includes a test method `createStat_ValidStat_ReturnsCreatedStatus` that creates a `BaseballStat` object with parameters "John Doe", 20, and 50, then performs a POST request to the "/api/baseball-stats" endpoint with application/json content type and the JSON representation of the stat. The expectation is that the status of the response will be created.

```
java
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.when;

@SpringBootTest
@AutoConfigureMockMvc
class BaseballStatControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private BaseballStatService baseballStatService;

    @Test
    void createStat_ValidStat_ReturnsCreatedStatus() throws Exception {
        BaseballStat stat = new BaseballStat("John Doe", 20, 50);

        when(baseballStatService.createStat(stat)).thenReturn(stat);

        mockMvc.perform(post("/api/baseball-stats")
                .contentType(MediaType.APPLICATION_JSON)
                .content(asJsonString(stat)))
                .andExpect(status().isCreated());
    }
}
```

The JUnit code wasn't bad - Spring unit testing is notoriously hard to set up - but it could have been better. It uses Mockito, one of the more popular unit testing frameworks, but it autowires the fields. This is frowned upon - it's better to have an auto wired constructor instead of auto wiring individual fields. Also, the line that creates a `BaseballStat` shows an issue: what do the 20 and 50 stand for? It would be better to have a Builder for the `BaseballStat` to make it more clear - and since ChatGPT already included Lombok in the solution, it could have been accomplished by just adding the `@Builder` annotation to the `BaseballStat` class.



So... not too bad. It generated a sizable amount of the code, but all of it could have been improved, and some of it wouldn't even compile, even for a short example like this.



Still, though, if you're a day to day programmer who needs to get things, this looks impressive!

Now let's try something more complicated...



show me how to write an entire enterprise software system for selling over ten thousand small craft supplies, with OpenID security, unit and integration tests, a deployment pipeline, management APIs, backup and rollover processes, an SLA of less than 100ms while supporting 100K concurrent users, providing localization for North American, South American, and European markets, handling PII (personally identifiable information), conform to Sarbanes-Oxley standards, and provide documentation?

Now, let's try something more complex. Here's a quick prompt for a craft inventory enterprise system that sounds more like the applications that are needed in the "real world". What do you think ChatGPT replied with?

What did ChatGPT do...?

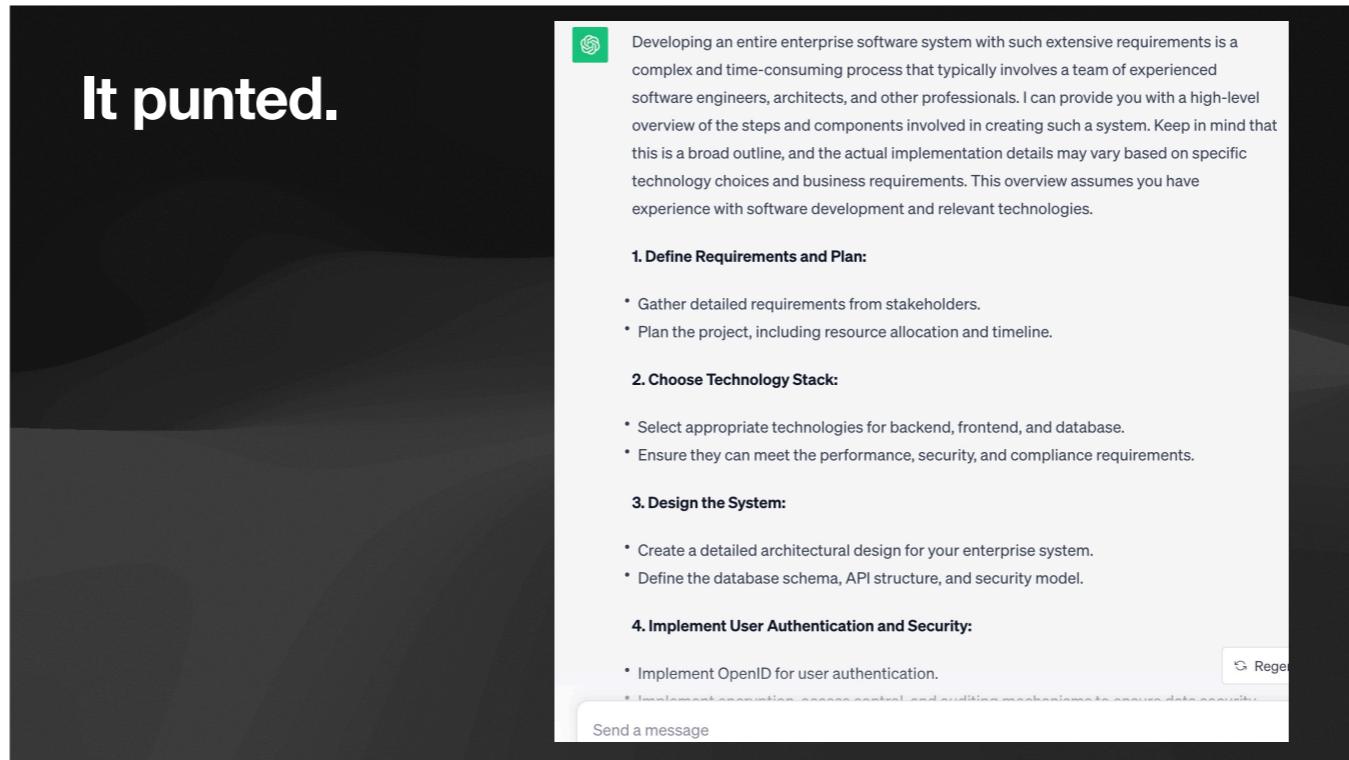


I SEE YOU SHIVER
WITH ANTICI...

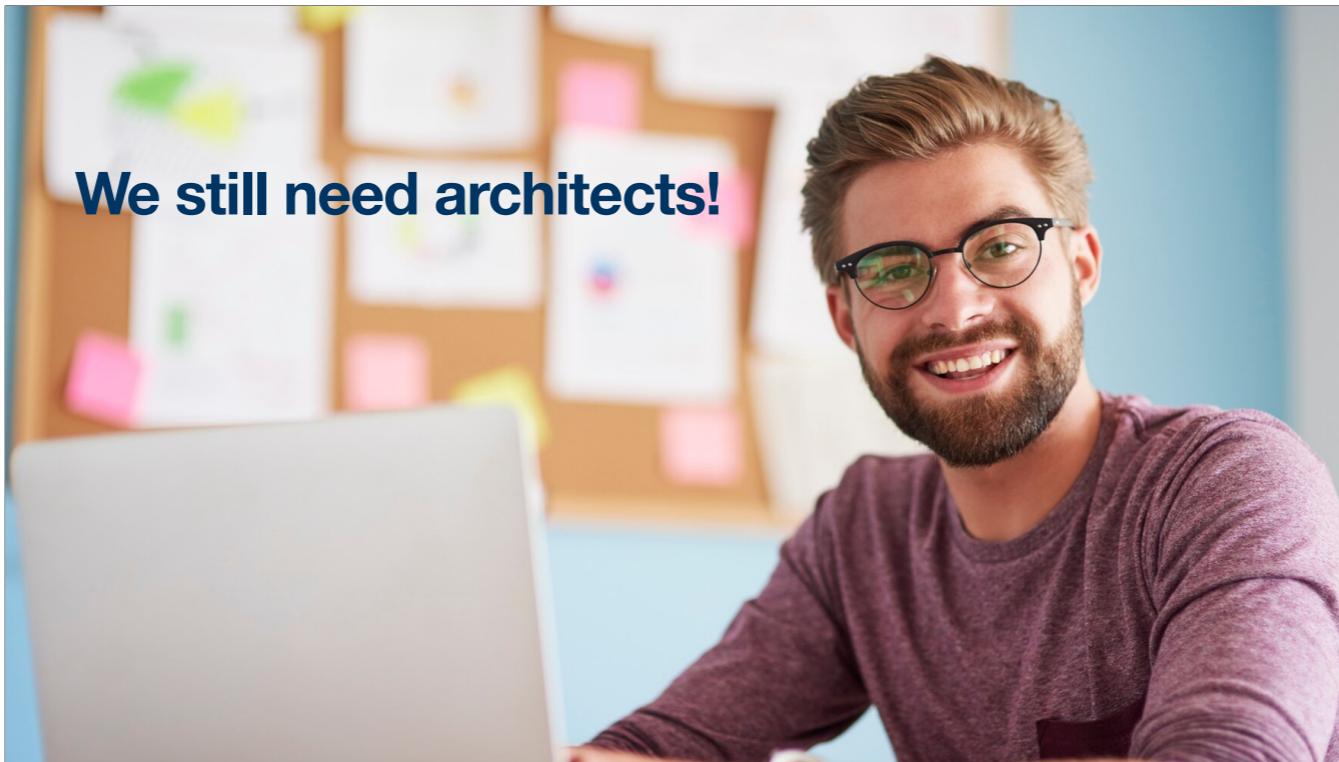
It took a few seconds...

... and

It punted.

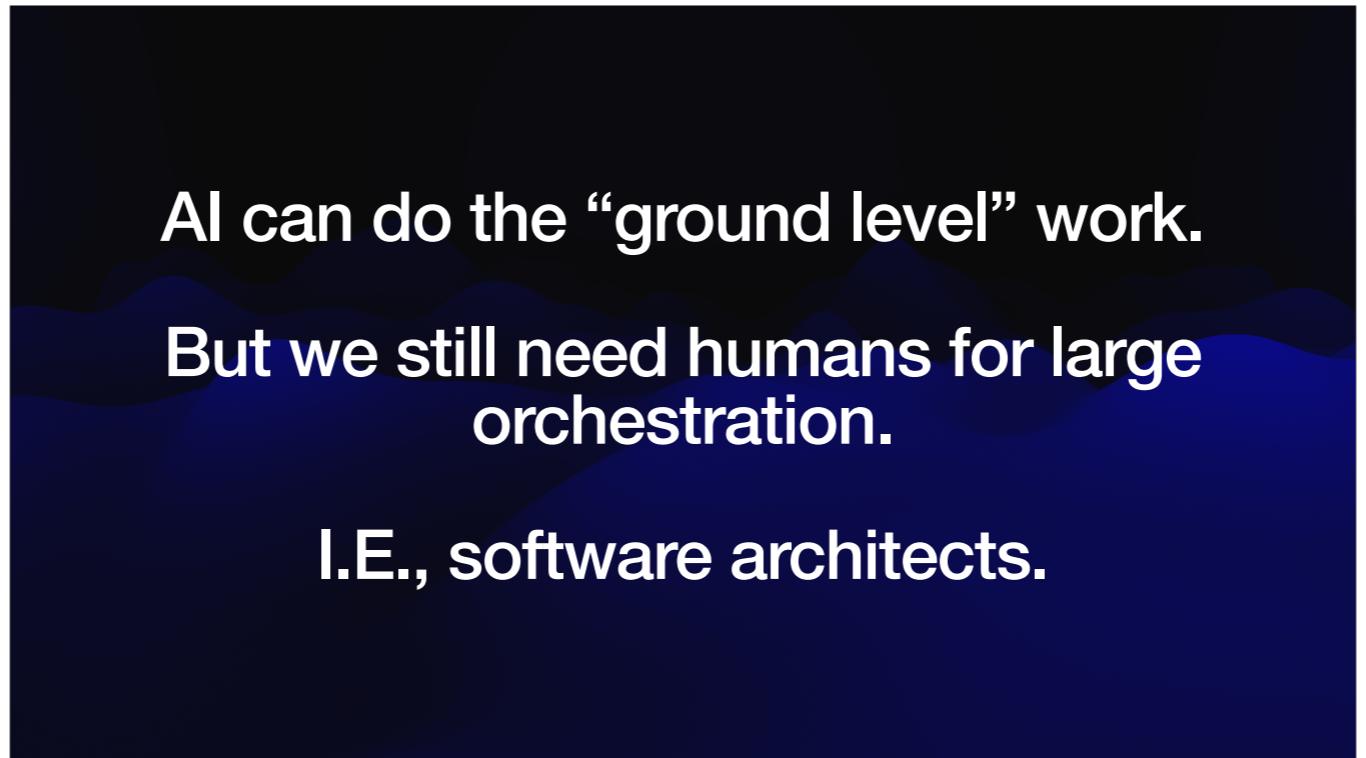


It punted. It didn't generate a single line of code. Instead, it just replied with some generic high level advice.



We still need architects!

As you can see, we still need software architects!



AI can do the “ground level” work.

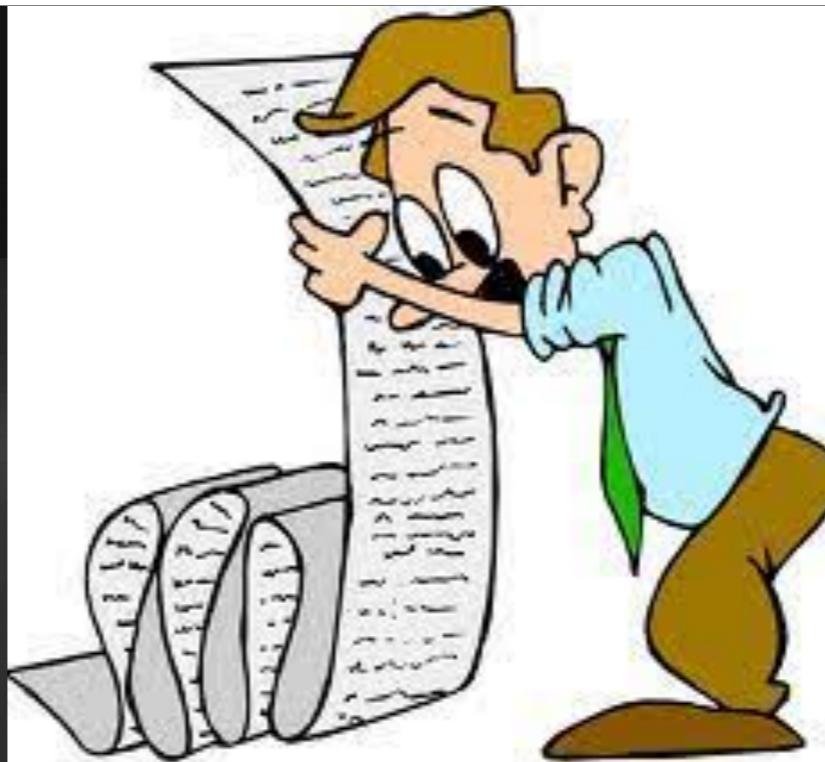
But we still need humans for large orchestration.

I.E., software architects.

So yes, AI looks like it can do some of the ground level, “junior programmer” work. But for large enterprises, that require knowledge of available hardware, budgets, available tools and technologies, etc., they still have a long way to go!

**There are
other issues
with AI
generated
code.**

**Let's look at
some of them.**



There are other issues with AI generated code, even at a lower level. Let's look at some of them.

GIGO - Garbage In, Garbage Out

- GPT-3 and Copilot were trained on public code in GitHub.
- Not all of it is **good!**
- Not all of it is **correct!**
- Some of it is **obsolete!**
- Newly generated code could spread **poor quality** or **incorrect** code.

The first problem is that these tools were trained on public code, from GitHub, AWS, etc. That code was written by people from all skill levels, from those who are professional engineers with multiple degrees and decades of experience, to high school students taking a programming class who may not know what they're doing and just checked something in to get a grade. ChatGPT doesn't know the difference... but it used it for training purposes. And if it used "garbage in", it may produce "garbage out".

**“Think of how
stupid the average
~~person~~^{programmer}, and
then realize half
of them are
stupider than
that.”**

George Carlin

Remember the quote from the late George Carlin? That applies to programmers, too (present company excluded, of course - y'all are awesome). But still, remember that programmers of varying skill levels trained these AI tools.

Hallucinations

- GPT doesn't really "understand" what it's asked to do.
- It's a **mime**, a **mimic**, a **parrot** - it tries to put together something that **looks** like what the answer should be.
- This works fairly well if it's been trained on similar items.
- But if it can't find an answer, it will put together something that looks like what you want, *even if it has to make something up.*
- We call these "**hallucinations**".

An important issue with LLMs is their fascinating tendency to "hallucinate". GPT doesn't "think". There is no "intelligence" there. It's basically a glorified pattern matcher. It's a mime, a mimic, a parrot, and it's only as good as the material it's been trained on. For example, there are huge numbers of example REST APIs in GitHub, so ChatGPT has a lot of examples to pick from. But if you ask it something that it has limited or no experience with, it will still try to "match the pattern" and return something that it thinks LOOKS like what you want. This is a "hallucination."

Example hallucination

The screenshot shows a ChatGPT interface. The user has asked for instructions on how to rename a pod. ChatGPT provides a step-by-step guide, including a code block for the command. However, it then contradicts itself by stating that the command does not exist, despite having just provided it.

4. Create the new pod using the `kubectl create` command and specify a unique name directly:

```
lua
kubectl create -f pod_copy.yaml --name=xxx-copy-20230613
```

Replace `'xxx-copy-20230613'` with the desired unique name for the new pod.

This approach explicitly sets the name during the `kubectl create` command, bypassing any potential conflicts with existing resources in the cluster.

Please note that if the pod copy still fails to be created, there might be other factors or issues at play. It's recommended to check the Kubernetes cluster logs or consult with your cluster administrator for further assistance in troubleshooting the problem.

kubectl does not recognize the --name option

I apologize for the incorrect information provided. You are correct that `'kubectl'` does not have a `--name` option for specifying the name during the `'create'` command. Please disregard that suggestion.

Here's an example: I was recently working with Kubernetes after not using it for some time, and I couldn't remember how to rename a pod, so I asked ChatGPT. It told me to use the “—name” option. The problem is, the Kubernetes tool doesn't have that option. But ChatGPT had seen that a lot of tools use a “—name” option to rename things, so it gave me that as an answer. When I told ChatGPT that the option didn't exist, it just said “sorry, you're right.”

“Monkey See, Monkey Do”

- GPT relies on being trained on **similar data**
- What of **brand new** languages and technologies?
- At first, extremely limited training data is available... right when it's needed the most!
 - E.g., **Rust** was created in 2006. How many example programs were available by 2007 or 2008?
 - **Mojo** was created in May 2023. How many example Mojo programs do you think GPT has trained on by now?
- With little training data, GPT is more prone to “**hallucinations**”.

This brings up another issue. GPT looks for patterns in existing data to supply its answers, but what if there isn't much (or any) to draw from? For example, the Rust language was created in 2006. How many Rust programs do you think existed in 2007? Not many! The Mojo language was created just last year - there aren't that many Mojo programs available for ChatGPT to train on. But Mojo is (currently) a subset of Python with some AI features added, so it's possible ChatGPT might return Python examples if you ask it for Mojo examples, and since Mojo doesn't yet implement ALL of Python, ChatGPT may recommend code that uses features that aren't available in Mojo.

Security Concerns

- Every time ChatGPT interacts with you... it **learns** from you.
- If private information gets into an LLM, it is **almost impossible to remove**.
 - Passwords.
 - Intellectual property.
 - Trade secrets.
- Some corporations **forbid** the use of these models.
 - Or require anonymized, “air gapped” use.

A more serious issue is that of security. ChatGPT learns from you - the data you give it as prompts or example data could be used to train the tool, which means it's possible to leak private information. And unlike a regular database, where you can delete a social security number or password in just a moment or two, it's nearly impossible to delete information from an LLM - this is actually a hot research topic at the moment. Some companies forbid the use of AI tools for this reason.

Debugging

- Junior programmers can easily generate code with AI tools.
- But is it **correct?**
- If an error occurs in AI-generated code, can a **human** who has never looked at the code **understand it** well enough to debug it?
- GPT and Copilot can sometimes debug code... but pasting it in a prompt could **leak confidential data** or custom IP.

We've seen that AI can generate incomplete, poor, or wrong code. But if that gets introduced into a program and the code fails, can a human who has never seen the code understand it well enough to fix it? Especially if it's a critical failure and time is of the essence in fixing it? Sometimes, GPT can debug code if you provide it in a prompt... but again, that risks leaking internal code or data.

Upgrades, Modernization, And Legacy Code

- Some common tasks are easier for humans than AI.
 - This is good, because we need junior programmers to write code to gain experience to become senior programmers!
 - Changing **frameworks** - e.g., from Lombok to slf4j, Angular to Angular 2, Angular to React, Java Dates to the JDK 8 Date/Time API.
 - Some still needs **human eyes**: working with CSS, for example.
 - Major **architectural changes**, such as converting to a DDD design.
 - Understanding when **requirements change** and the impact on the code.

There are also some tasks that are better suited for humans. These usually include sweeping changes, like converting from one framework to another. Some need human eyes, such as UX - while computers can do a good job of generating code, they're horrible at judging aesthetics and choosing colors. Also, most organizations would frown upon dumping their entire code base into an AI and telling it something like "rewrite this using React" - would YOU trust that code?

Industry Reluctance

- We're already seeing **pushback** against AI in other fields.
 - **Law**
 - AI "lawyers" in the courtroom? "Made up" legal precedents?
 - **Hollywood** (see the current strike)
 - Copying old plots, scripts, characters, character arcs?
 - Could **software development** be next?
 - Can AI-generated software be **copyrighted**?
 - In March 2023, the U.S. Copyright Office said that works produced with the help of AI "may" be copyrightable.
 - It's a **fuzzy area** for now.

Then there are legal issues. There was a recent case where a lawyer used ChatGPT to provide existing case law... and ChatGPT hallucinated the cases, making up prior cases out of thin air. The lawyer didn't verify them, so when the judge on the case reviewed it, he couldn't find any of the (non-existent) cases, and now that lawyer is in legal trouble. The recent Hollywood strike had AI-generated TV programs and films as a major point of contention. And the US Copyright office said that AI-generated works "may" be copyrightable. Companies don't like having code that they don't own!

Critical Software

- Some software is **too important** to trust to AI tools that “**hallucinate**”.
 - Avionics software.
 - Full self-driving car software (although Tesla is getting there....)
 - Embedded software for medical devices like pacemakers.
 - Embedded software for spacecraft and satellites.

Then, there is some software that is just too important to trust to AI tools that “hallucinate”. Would you fly on a Boeing plane - ok, ok, let me rephrase that - would you fly on an AIRBUS plane if you knew the avionics software had been written by ChatGPT? What about full self-driving software - Tesla is doing a great job, but there are still issues, such as the latest release that sometimes cuts corners so sharply that it’s cutting tires on curbs. What about software that controls devices that keep people alive, like pacemakers? Or software for multibillion dollar spacecraft? Should a technology that hallucinates options for current software tools like Kubernetes be trusted with writing software for that?

Lack Of Contextual Awareness

- AI tools do not know what **other software** exists at a company.
- They do not know company **standards or practices**.
- They do not know whether existing software could be **tweaked or reused**.
- They do not know why certain **decisions** were made (e.g., GCP instead of AWS).
- They do not understand **security concerns**.
- They don't understand issues with **software licenses**. Some companies are afraid of incorporating libraries with certain licenses in their products.

AI tools also don't know about the context of the code they're writing. For example, a lot of companies have their own UX frameworks or custom libraries that they use - ChatGPT doesn't know about those. Sometimes there's a legal or strategic reason that ChatGPT doesn't know. For example, I once worked for a prominent Atlanta company that refused to use AWS because they viewed Amazon as a competitor, and therefore used Google's GCP for everything. Sometimes companies have favorable terms with other companies and their technology that ChatGPT doesn't know. And ChatGPT doesn't know what information might be a security risk if it got out. Finally, ChatGPT doesn't know that some companies are particular about what licenses are attached to code or frameworks that may be incorporated into their products.

Lack Of Creativity

- Could an LLM have created:
 - DVR time-shifting software?
 - The MPEG-4 encoding algorithm?
 - LL and LR grammars and general compiler theory?
 - The Internet protocols?
 - Figuring out how to self-drive a car in different environments with different laws?
 - LLMs and transformers?
- There will **always** be room for pure original thinking.

Finally, it's important to remember that these LLMs cannot THINK. At best, they remix things that already exist into new combinations. But could an LLM have created something like DVR time-shifting software? Or developed the math necessary for the MPEG-4 encoding algorithm? Or created the LL and LR grammar theory used for creating compilers, or the layered Internet protocol stack, or even LLMs and transformers? No. There will ALWAYS be room for human creativity.

Okay, But What Can It Do Well?

- Low-level code generation: REST APIs, configuration, database access, etc.
- Code optimization.
- “Greenfield” development.
- Documentation generation.
- Test case generation.
- Basically, the kind of tasks you hand off to a junior developer.

So, what can AI tools do well? For the most part, those jobs that a junior programmer would typically have - low level code generation from clearly defined requirements; optimizing existing code to make it more performant; “greenfield” development for starting up brand new projects; generation documentation and test cases, etc.

What's A Software Developer To Do?



So, what should we programmers do to keep our jobs?



**Focus on architecture.
Not coding.**

We need to focus on architecture, not ground level coding. We need to do the things that AI tools cannot do now (or won't be able to do for quite some time).

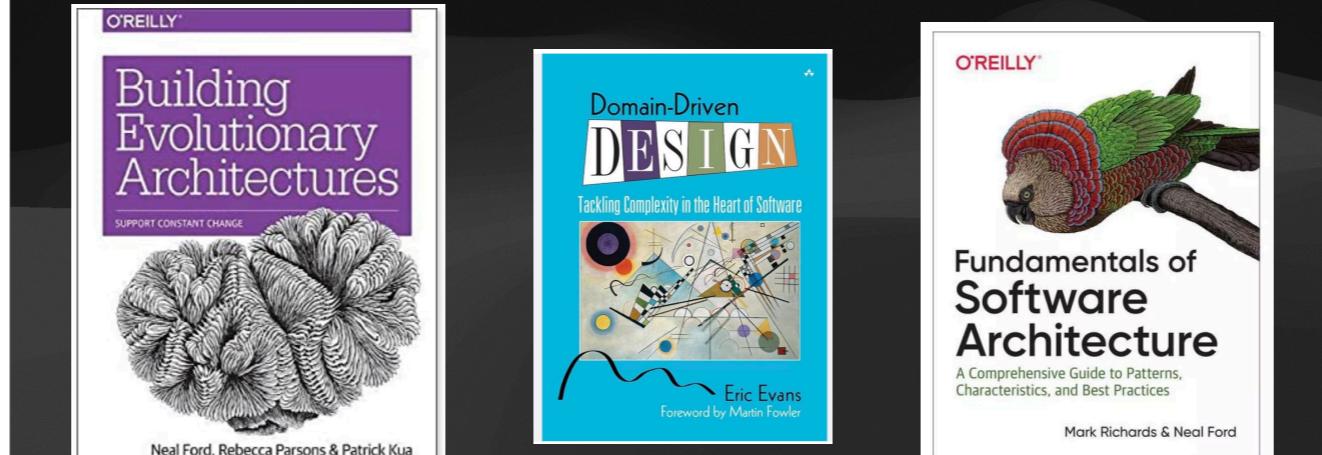
Junior Developers

- Don't just learn a language or framework.
 - Learn which languages are best in which situations.
 - Learn the common idioms.
- Look beyond: pricing, availability of libraries, availability of programmers.
- Learn the architectures that are best implemented in various languages.
- Learn how to create great prompts for code generation.
- Learn how to understand, follow, test, and debug AI-generated code.

If you're a junior developer, don't just learn a language or framework. You need to learn the WHY and not the HOW. Why are we using this language instead of that one? For example, why write a system service in Rust or C instead of Python? Why use Python for data science instead of COBOL or Java? And we need to look at things that AI tools cannot see. Can your company afford a particular library? Is it available (or will it continue to be available)? Can you find programmers that can deal with the AI solutions? I once worked for a company that did their work in Scala and Clojure, but they couldn't find enough local programmers who knew those languages, so they switched to Java. You also need to learn the best architectures for each programming language. And experience with AI tools is a must - learn to write prompts that are specific and thorough, and learn how to read, understand, and DEBUG code generated by AI. Finally, you need to learn to evaluate the quality of AI generated code and know when it's acceptable and when it is not.

Senior Developers / Architects

First, study architecture!



If you want to learn architecture, I recommend these books. “Building Evolutionary Architectures” and “Fundamentals of Software Architecture” by Neal Ford are excellent, especially the latter, which covers many of the architectures found in modern software. And the old classic “Domain-Driven Design” is an excellent (although detailed and fairly “academic”) look at how to develop requirements for and specify software. It also includes some examples of how to interview domain experts to understand their need and then map that into software.



Neal Ford also has a new book that just hit the market a few weeks ago: "Head First Software Architecture." The "Head First" series is a great series that uses a number of techniques to teach in a fun, engaging, and stimulating style. I haven't read this one yet, but I was in a client meeting with co-author Raju Gandhi a few weeks back and got to chat with him for a few minutes about it - he was excited to see it going into print.

Learn architectures and how to implement them

- Layered architecture
- Event driven
- Microkernel
- Microservices
- Space-based architecture (tuple space)
- Client / server
- Broker, peer to peer, etc.

Learn the advantages and disadvantages of each.

Learn when to apply them.

If you're a junior developer, you may not be aware of just how many architectures there are! I've listed a few of them here. You should learn how each of these works, when and how to use them, and the advantages, disadvantages, and trade-offs involved in each one. AI tools can't tell you which is appropriate for the system you want to build, but once you decide, it can happily help you build the pieces.

Divining Requirements

- AI-based tools cannot create requirements
- Most domain experts do not know enough about software to specify requirements
- An architect can be the bridge between the domain experts and the AI tools
 - Learn how to guide a domain expert and understand the problem
 - Learn how to help formulate a suitable architecture to solve the problem

So, AI can generate code if you give it the right prompt... but most people don't know how to write that prompt. For example, I worked with an expert on crop science several years back - he was brilliant, and if you could grow corn on the airless, waterless moon, he'd be the one who could do it :) - but he wasn't a software expert. He could explain to a software architect what he wanted, though. And that's where we are still needed: we can learn to talk to experts in any number of fields, figure out what they want, and then translate their dreams into an architecture and then prompt the AI tools to help us generate the pieces.

Mentoring Junior Developers

- Work with them to teach them how to craft high quality prompts
- Remember to ask for: security, test cases, documentation, design patterns, OWASP checks, etc.
- Show them how to spot hallucinations, and what to do if one occurs.
- Work with them to understand how to understand and debug AI-written code.
- Help them learn architecture.
 - Don't just say "implement X using framework Y."
 - Instead, "we're implementing X using Y instead of W or Z because..."

As senior developers, we also need to mentor junior developers. We need to teach them how to prompt for solid code, remembering to ask for security, test cases, documentation, OWASP checks, etc. We need them to spot hallucinations, and how to recognize poorly written AI code. And we need to help them learn architecture. Don't do it by scheduling one hour meetings on Fridays to "teach architecture" - incorporate it in day to day work. For example, don't just say "implement X using framework Y" - instead, tell them "we're going to implement X using Y instead of W or Z because..." so that they can learn how to choose the best framework for any given situation.

Ensure Design Specifications Are Met

- Ensure code reviews are held to make sure that AI-generated code is solid
 - Perhaps add pre-commit git hooks to test code
- Lean on AI tools to generate unit tests
- If using Java or C#, look into ArchUnit (archunit.org)
 - It's similar to JUnit, but for testing architectures
 - Fluent interface - easy to write in an IDE
 - You can write your own rules!

We also need to learn to evaluate AI generated code. In some ways, we can use AI as a check - have the AI generate code to test itself... but then LOOK AT IT and make sure it's thorough, correct, secure, and performant. Also, there are tools available to help you test a software architecture just like we test code. For example, ArchUnit (which Neal Ford mentioned in yesterday's keynote address) is a tool that lets you use existing rules and custom-written tools to test a software's architecture.

ArchUnit Example

```
@ArchTest
private final ArchRule no_generic_exceptions = NO_CLASSES_SHOULD_THROW_GENERIC_EXCEPTIONS;

@ArchTest
private final ArchRule no_java_util_logging = NO_CLASSES_SHOULD_USE_JAVA_UTIL_LOGGING;

@ArchTest
private final ArchRule loggers_should_be_private_static_final =
    fields().that().haveRawType(Logger.class)
        .should().bePrivate()
        .andShould().beStatic()
        .andShould().beFinal()
        .because("we agreed on this convention");

@ArchTest
private final ArchRule no_jodatime = NO_CLASSES_SHOULD_USE_JODATIME;

@ArchTest
private final ArchRule no_field_injection = NO_CLASSES_SHOULD_USE_FIELD_INJECTION;
```

Here's an example. It looks similar to JUnit, but instead of using JUnit's @Test annotation, ArchTest uses @ArchTest. There are a lot of built-in tests (the first two and the last two here) that you can use. For example, the first one ensures that code doesn't just throw a generic Exception - it makes sure that programmers use more specific or custom exceptions. Another makes sure that the built-in Java logging isn't used (does anyone really use that) and another makes sure that the outdated Joda Time library for date and time manipulation isn't used. The middle rule is a custom written one that checks the fields of every class in the project, and for those that are a Logger (or a subclass), it makes sure that the field is private, static, and final. It even lets you attach a reason for the test in the .because(...) line.

Another ArchUnit Example

```
import static com.tngtech.archunit.library.Architectures.layeredArchitecture;

@AnalyzeClasses(packages = "com.tngtech.archunit.example.layers")
public class LayeredArchitectureTest {
    @ArchTest
    static final ArchRule layer_dependencies_are_respected = layeredArchitecture().consideringAllDependencies()

        .layer("Controllers").definedBy("com.tngtech.archunit.example.layers.controller..")
        .layer("Services").definedBy("com.tngtech.archunit.example.layers.service..")
        .layer("Persistence").definedBy("com.tngtech.archunit.example.layers.persistence..")

        .whereLayer("Controllers").mayNotBeAccessedByAnyLayer()
        .whereLayer("Services").mayOnlyBeAccessedByLayers("Controllers")
        .whereLayer("Persistence").mayOnlyBeAccessedByLayers("Services");
```

Here's a more complex rule that ensures that the program uses a layered architecture. You can specify what packages the controller, services, and persistence code reside in, and then you can gate access to the layers. For example, the controller layer shouldn't be accessed by the services or persistence layers. If someone writes code (or generates code using AI) that violates this, ArchUnit will catch it and report it as an error. As AI generates more and more low-level code, tools such as this will become important in ensuring that it conforms to the intended architecture.

Conclusion

- AI can perform junior developer tasks with guidance
- It's still a long way from being able to replace software architects
- Software architects need to learn to incorporate AI tools in their process
- Knowledge of various architectures, their tradeoffs, costs, etc. is vital
- Junior developers need to be mentored to work effectively with AI
- Finally, by performing the menial tasks, it can free developers to focus on higher level design and architecture, producing better quality systems.

It's been a long talk, but we're finally at the end! Let's sum up. Yes, AI can write code today at the level of a junior developer (when guided), but it still can't be blindly trusted - it needs human eyes to evaluate it and then modify the prompt if it doesn't meet expectations. Software architects will still be needed for the foreseeable future, so learning the different types of architectures, how to choose an appropriate one given their tradeoffs, how to implement them, how to incorporate external decisions, the "soft skills" needed to talk with domain experts and translate their needs into a suitable architecture.. these will still be needed. And senior developers need to learn how to mentor junior developers so that they can become solid senior developers. Finally, it's possible that by having AI write lower-level, "drudge work" code, it could actually free human beings to concentrate on higher level designs, which can be more fun and intellectually stimulating.



I'M KELLY L. MORRISON

ANY QUESTIONS?

That's pretty much all I wanted to talk about today. Again, my name is Kelly Morrison, and I'll stick around for any questions you might have.

Daugherty

BUSINESS SOLUTIONS

kelly.morrison@daugherty.com
kellyivymorrison@gmail.com



[Slides on GitHub](#)

Kelly Morrison
Manager / Application Architect at Daugherty
Business Solutions



[LinkedIn](#)



Here are my personal and private email addresses. The QR code at the top links to a copy of these slides on GitHub. The QR code at the bottom links to my LinkedIn profile. And I work for Daugherty Business Solutions, a consulting company based in St. Louis that has offices in Atlanta, New York, Chicago, Poland, and more. If you're interested in working for us, please come up and see me after the presentation! Finally, to reward you for attending my presentation, here's a picture of my adorable French bulldog, "Daisy Mae". Thanks again for attending, and I'll stick around to answer any questions you may have.