

# Traditional Software Development vs AI

Ivan Nicola Viragine



THE  
DEVELOPER'S  
CONFERENCE

# Objective

Understand how AI changes the software development paradigm and highlight the main **differences** in development **using AI models** and **traditional** development

## Ivan Nicola Viragine



- AI Engineer at a California Startup
- 16 years of experience in software development
- 8 years of experience with Artificial Intelligence
- AI Engineer, Data Scientist, Full Stack Developer, Startup Founder, etc.
- MBA in Data Science from USP
- Bachelor of Computer Science from UNESP

# When to use AI?

**Complex** problems, with **many variables** and **high volumes** and/or ***variations of data***

# Why use AI?

AI algorithms **search for patterns** and **relations** that would be **very difficult or even impossible** to be determined by humans and **transformed into code**



# Examples

## Image recognition

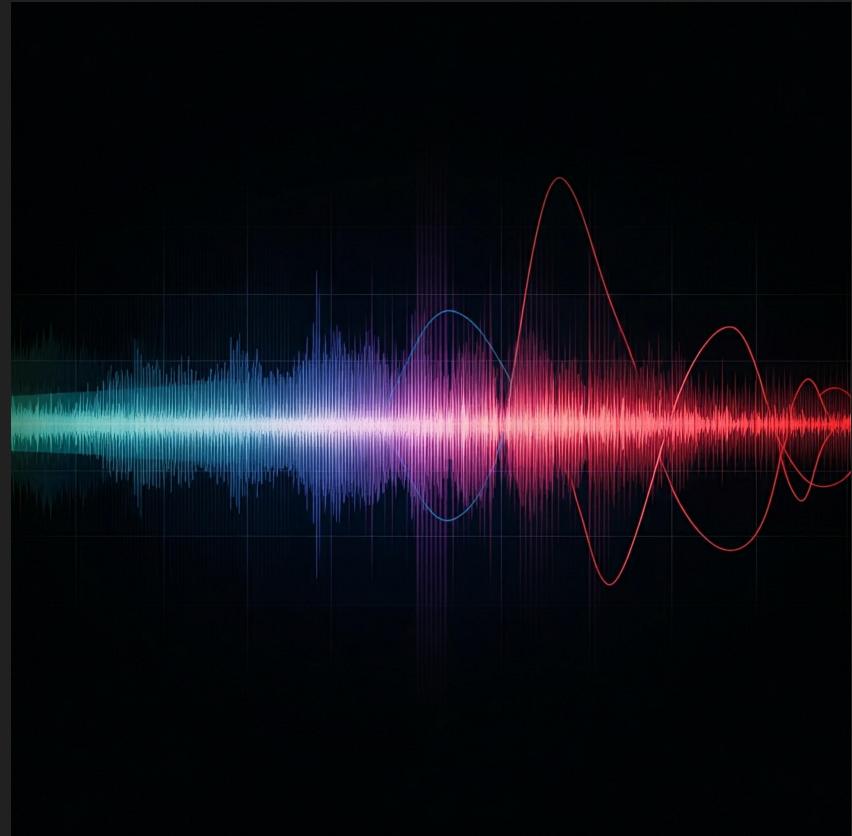
- How to write **static rules** to identify cats in images?



# Examples

## Voice recognition

- How to write **static rules** to identify words and phrases in an audio?



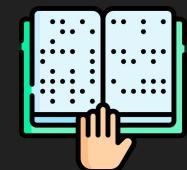
# Where to use AI models in software?

- **Fraud identification and prevention**
  - banking apps, e-commerces/online stores
- **Recommendations**
  - e-commerces/online stores, ads, social networks
- **Sentiment analysis**
  - reviews, customer service
- **Language translation**
  - translation apps, customer service
- **Medical diagnosis**
  - health apps, hospitals



# Where to use AI models in software?

- **Voice recognition**
  - virtual assistants, translation apps, accessibility
- **Image recognition**
  - security apps, social networks, accessibility
- **Text summarization**
  - news apps, customer service, research
- **Chatbots**
  - SAC, e-commerce/virtual stores
- **Process optimization**
  - industries, logistics
- **Security**
  - access control, facial recognition, theft prevention

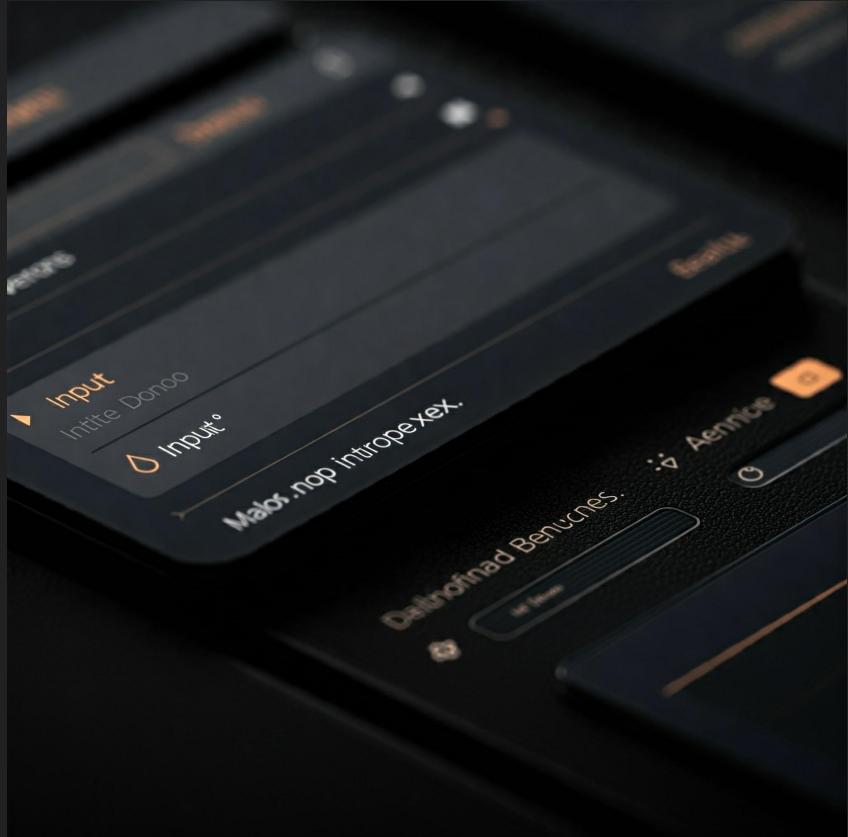


# Differences

# Inputs

## Traditional

- The inputs are more **predictable** and **controlled**, usually through a GUI with fields and validations.



# Inputs

## AI

- Inputs can be more varied and even unpredictable, such as **free text, images, audio, etc.**

The screenshot shows a dark-themed ChatGPT interface. At the top right is a 'Share' button. Below it, the text 'ChatGPT 4o' is followed by a dropdown arrow. A user message bubble contains the text: 'Can you help me create a personalized morning routine that would help increase my productivity throughout the day? Start by asking me about my current habits and what activities energize me in the morning.' A response bubble from 'GPT' follows: 'I'd be happy to help you create a personalized morning routine! To start, let me ask a few questions so I can better understand your current habits and what works best for you:' This is followed by a numbered list of four questions:

1. **What time do you usually wake up in the morning?** Do you use an alarm or wake up naturally?
2. **How do you currently spend the first hour after waking up?** What does your current routine look like?
3. **Are there any specific activities or habits that help energize you in the morning?** For example, do you enjoy exercise, meditation, reading, or having a certain type of breakfast? ↓
4. **What time do you typically start work or your main responsibilities for the day?** ↑

At the bottom left is a 'Message ChatGPT' button with a speech icon. On the far right is a small circular icon with an upward-pointing arrow. At the very bottom center is the text 'ChatGPT can make mistakes. Check important info.'

# Outputs

## Traditional

- **Deterministic:** the same input  
*always* produces the same output

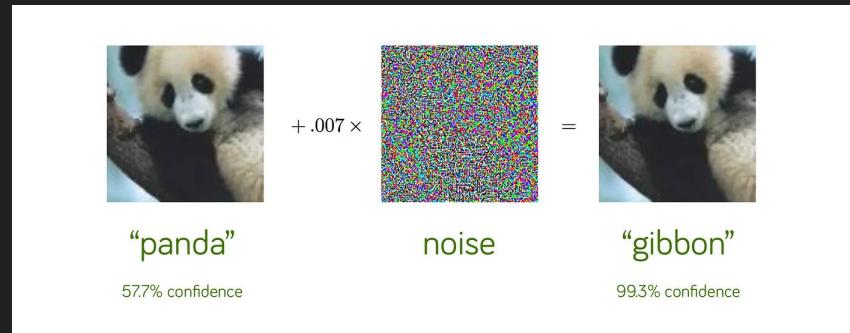


```
if this:  
    do_that()  
elif that:  
    do_this()  
else  
    do_something_else()
```

# Outputs

AI

- **Non-deterministic (probabilistic)**: the same input *may or may not* produce the same output and/or *small variations* on the input can produce variations in the output in a relatively *unpredictable way*.

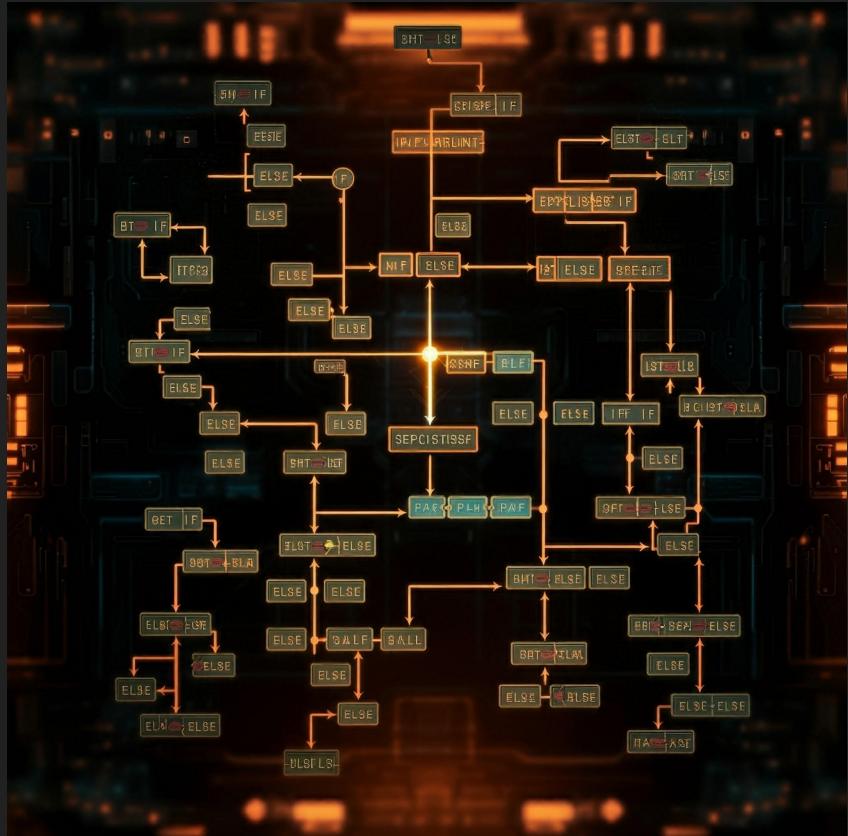


*Explaining and Harnessing Adversarial Examples,  
Goodfellow et al, ICLR 2015*

# Data Dependency

# Traditional

- Dependent on data, but not directly.  
Code can be written without data and  
**quality is more dependent on the  
logic and structure of the code  
than the data itself.**



# Data Dependency

AI

- **Totally data dependent.** Without data, there is no model. The quality of the model is *directly proportional to the quality* and, in a way, *amount* of the data used for training. Model evaluation also has a high dependence on the quality and quantity of data.



# Interpretability

## Traditional

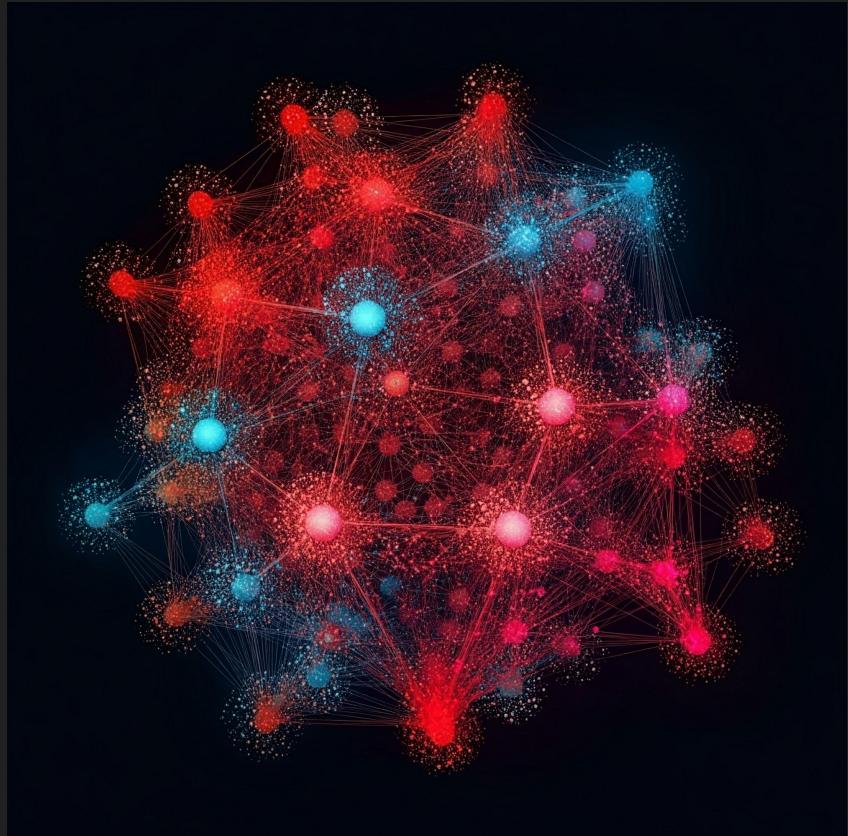
- You can understand how your code works, accurately predict output, and debug issues more easily. **The execution path is clear and predictable.**



# Interpretability

## AI

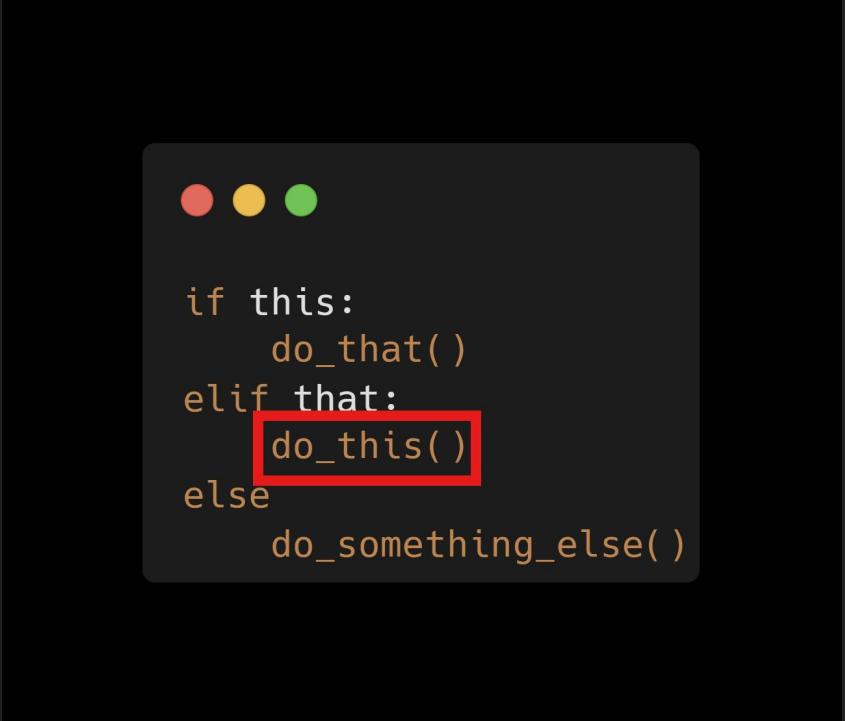
- AI is a "**black box**": It is difficult and sometimes "impossible" to understand how AI drawn a certain conclusion or decision or accurately predict an output from an input.  
There are tools that help in understanding one or more outputs, but not in a generalized way.



# Update and maintenance

## Traditional

- It is possible to update the code directly, **modify only the necessary parts** and test changes more directly and efficiently.



```
if this:  
    do_that()  
elif that:  
    do_this()  
else  
    do_something_else()
```

# Update and maintenance

## AI

- For an AI model to be updated, it is usually necessary to train a new model with new data and/or adjust hyperparameters. This can be a complex and expensive process and **may not generate the expected results**. Even in cases of prompt update/optimization, where there is no need to retrain the model, the update may generate unexpected results.



# Error Debugging

## Traditional

- Errors are easier to identify and correct because the execution path is clear, **you have access to the code and variable values at every step.** Once the adjustments are made, the update and fix can start to take effect "immediately".



# Error Debugging

## AI

- Errors can be difficult to identify and correct and **are not always explicit in the training data**. Adjustments to the data, model, and/or training process may eventually be necessary. Even if you find examples in the training data that may be causing errors, removing them and retraining the model **will not always guarantee that the error will be corrected**.



# Scalability

## Traditional

- It is possible to scale code in a relatively straightforward and efficient way, **adding more hardware and/or optimizing the code**. There is a relatively predictable relationship between inputs and resources.



# Scalability

## AI

- There is not as much predictability in the scalability of an AI model because the **"outputs" are not predictable and/or directly proportional to something as measurable on the input.**



# Bias and Justice

## Traditional

- All **bias is in the code** and can be identified and corrected directly and will rarely be introduced completely unconsciously.



```
if cat:  
    return "bad"  
elif dog:  
    return "good"  
else:  
    return "don't know"
```

# Bias and Justice

## AI

- Bias can be introduced into the training data and/or the model in a **unconscious way and can be difficult to identify and correct.** Due to the high volume, you won't always know that the data could generate biases or injustices.



# Hallucinations

## Traditional

- Unexpected outputs are usually the result of **errors in the code** and can be corrected directly. It is possible to understand why the errors occurred.



```
if cat:  
    return "dog"  
elif dog:  
    return "dog"  
else  
    return
```

# Hallucinations

## AI

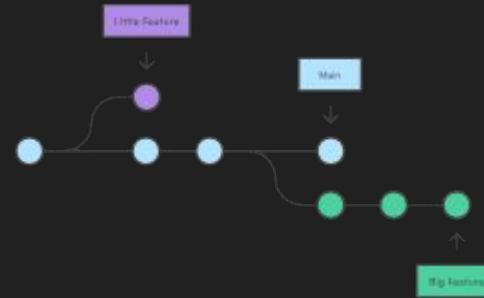
- Unexpected outputs may be the result of model hallucinations and may be **difficult to identify and correct.**



# Versioning

## Traditional

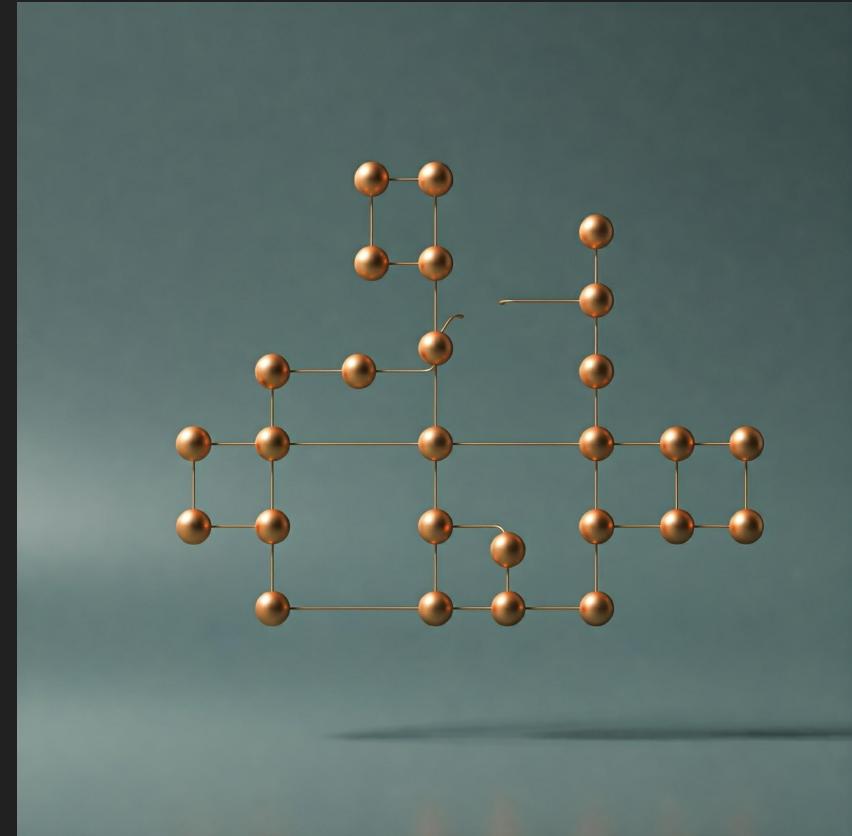
- It is possible to version the code and do rollbacks in a relatively straightforward and efficient manner. If the previous version worked for certain scenarios and the new version does not work for a scenario, revert to the previous version will ensure behavior similar to the past.



# Versioning

## AI

- It is possible to version AI models and revert them in a similar way, however, if there is a need for retraining, even with the same data and parameters it is possible to have a new, relatively different model.



# Tests

## Traditional

- It is possible to test the **predefined paths** and **expected inputs and outputs** in a relatively direct and efficient manner.



# Tests

## AI

- It is possible to test AI models, however, as **outputs are not predictable**, the tests are **probabilistic** and use evaluation metrics to determine model quality. **Even if the model has a good evaluation metric, it may not perform well in real scenarios.** There is total dependence on **quality and quantity of data** used for testing **effective tests**.



# Feedback

## Traditional

- It is not so common for users to give direct feedback for interactions, the most common are reports of **specific bugs** in a more direct way.



# Feedback

## AI

- It is common for users to give feedback for interactions and this feedback **are fundamental** for improving the model. This feedback can be from “**like**” or “**dislike**” or **even ratings from 1 to 5 with comments**. Feedbacks are usually **more complex**.



# Resources

## Traditional

- **Development:** notebooks and common computers are sufficient to develop and test most systems.
- **Servers:** Common VMs with CPU, RAM and storage are sufficient to serve most applications.



# Resources

## AI

- **Training:** a large volume of data and computing power is required. In general, GPUs and/or TPUs and/or cloud services are required.
- **Inference:** resources required are typically smaller, but still more demanding than most traditional software.
- **General development:** Even dedicated "consumer grade" GPUs with 24GB of VRAM may not be enough.

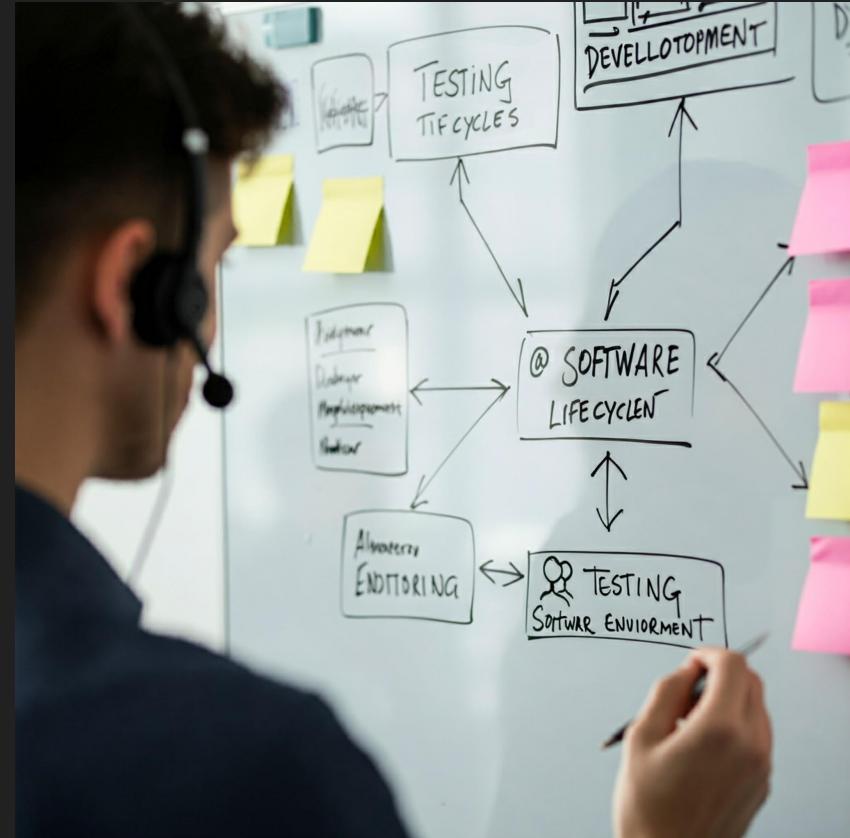


# Life cycle

## Traditional

1. Requirements analysis
2. Architectural drafting
3. Development
4. Tests
5. Deploy
6. Maintenance and updates

*The highest efforts are usually at 1, 3 and 6.*

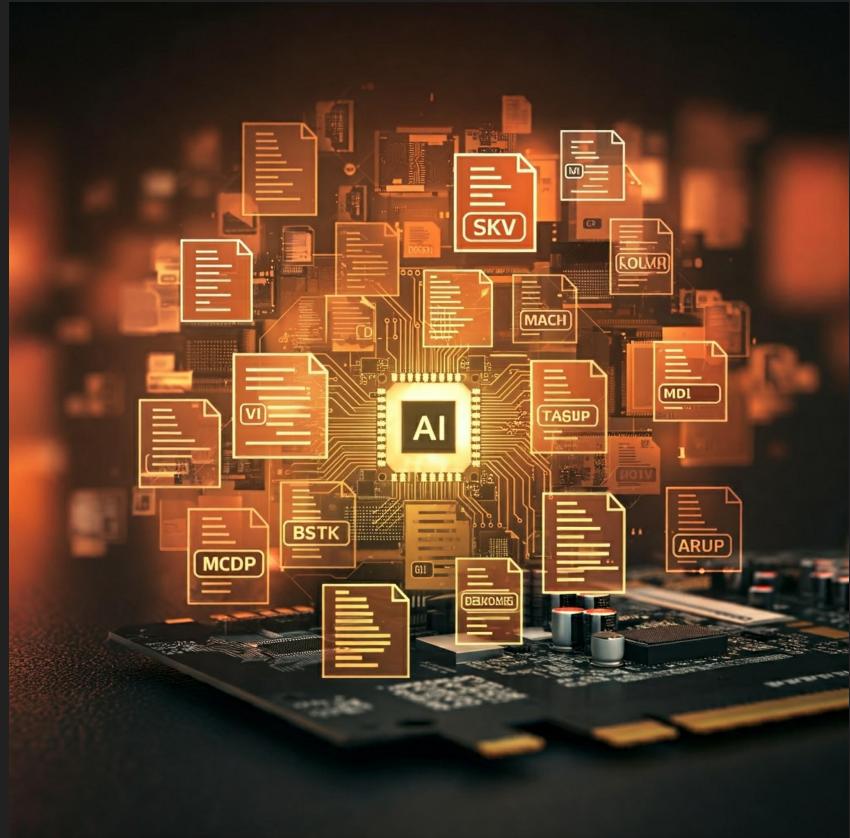


# Life cycle

## AI

1. Requirements analysis
2. Data collection and preparation  
(training, re-training and validation/evaluation)
3. Model Training/Fine-tuning
4. Model evaluation and validation
5. Model Deployment
6. Monitoring and re-training

*The greatest efforts are usually at 2, 3, 4 and 6.*



# Life cycle

## AI (LLMs)

1. Requirements analysis
2. Data collection and preparation  
(validation/evaluation and possibly fine-tuning)
3. Model fine-tuning and/or prompt engineering
4. Model evaluation and validation
5. Model/API Deployment
6. Monitoring and re-fine-tuning and/or prompt engineering

*The greatest efforts are usually at 3, 4 and 6.*



Thanks!