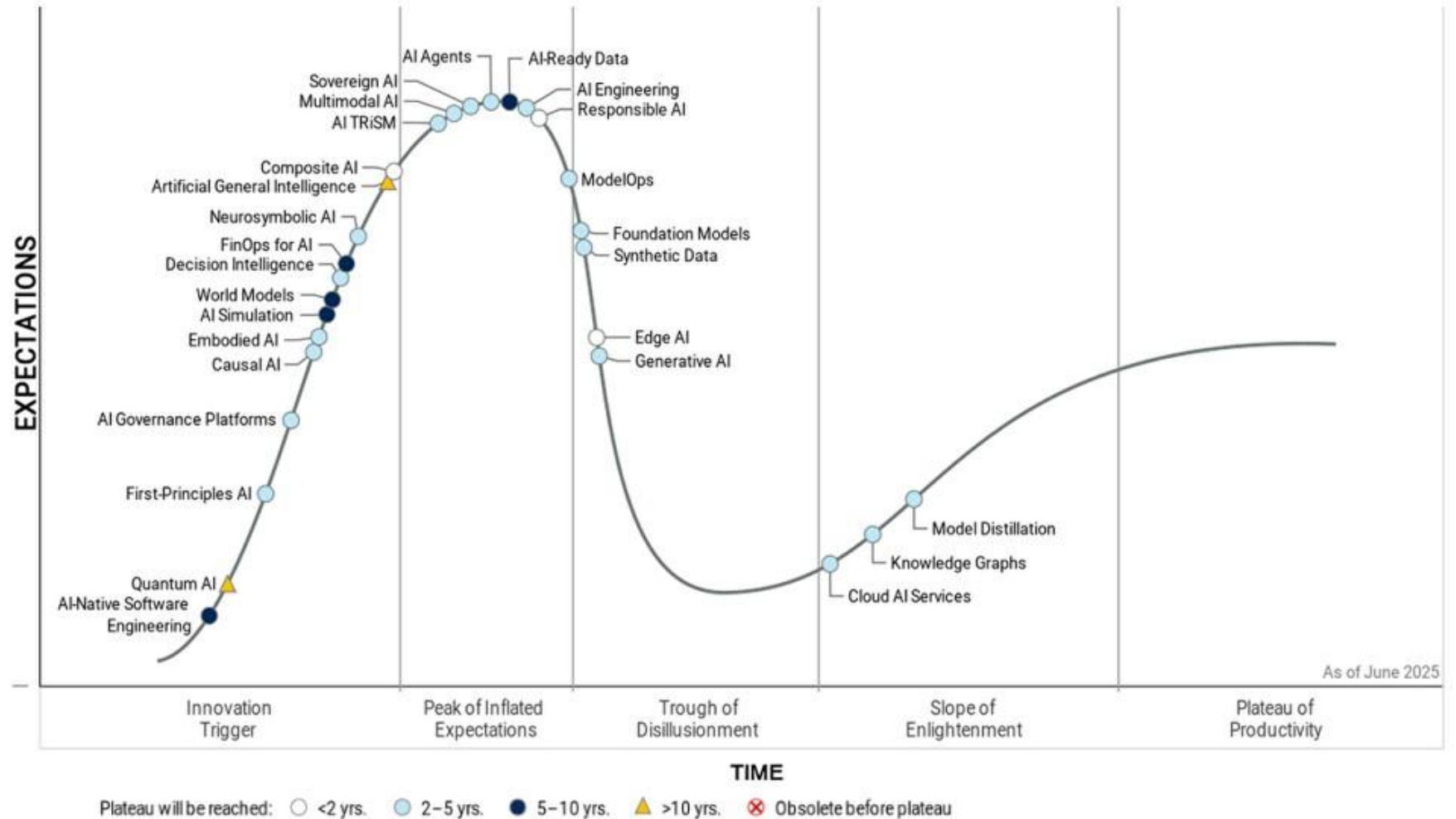
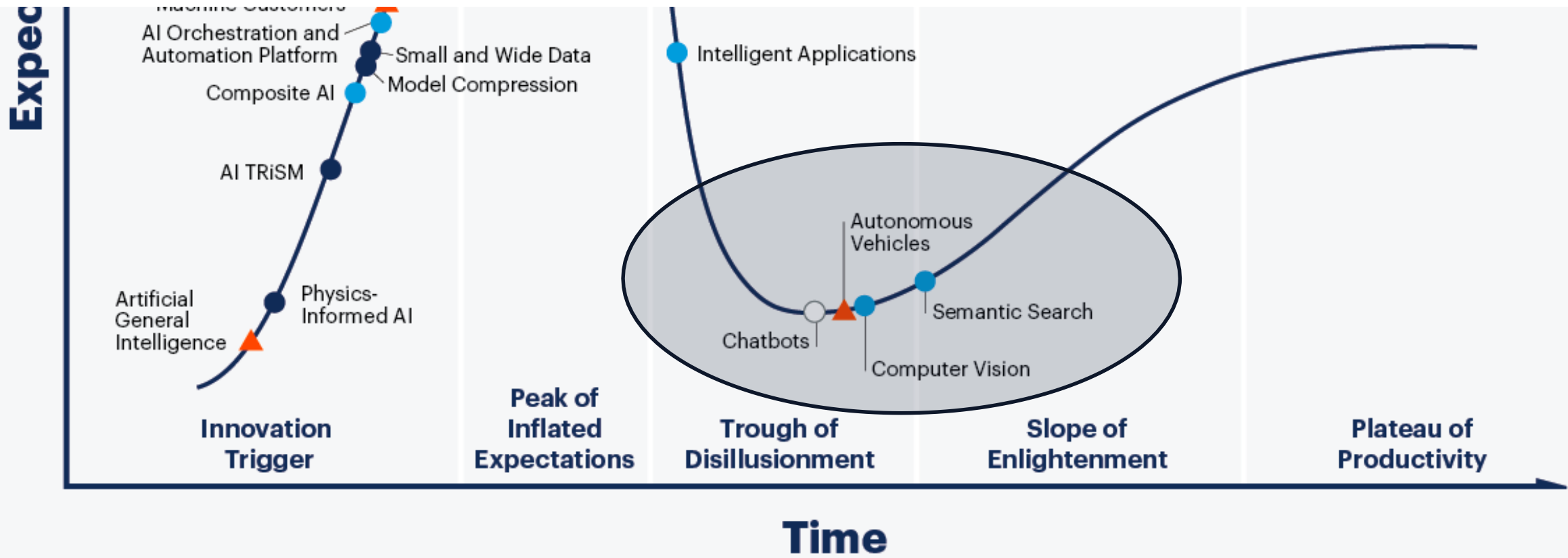


# LLMs in Software Testing

# Gartner's Hype Cycle for AI, 2025 report





Plateau will be reached:

- less than 2 years
- 2 to 5 years
- 5 to 10 years
- ▲ more than 10 years
- ⊗ obsolete before plateau

As of July 2021

**gartner.com**

Source: Gartner

© 2021 Gartner, Inc. and/or its affiliates. All rights reserved. Gartner and Hype Cycle are registered trademarks of Gartner, Inc. and its affiliates in the U.S. 1482644

**Gartner®**

<https://www.gartner.com/en/articles/the-4-trends-that-prevail-on-the-gartner-hype-cycle-for-ai-2021>

# Limitations

## Be Careful

- **Never believe output from AI without verifying it!**
- AI is known to **hallucinate** (i.e. make up facts and references)
- Verify with knowledge you already know.
- Verify by using your textbook or other academic resources (e.g., the Library website).
- Google information if you are unsure.



# Large Language Models

## Transformers

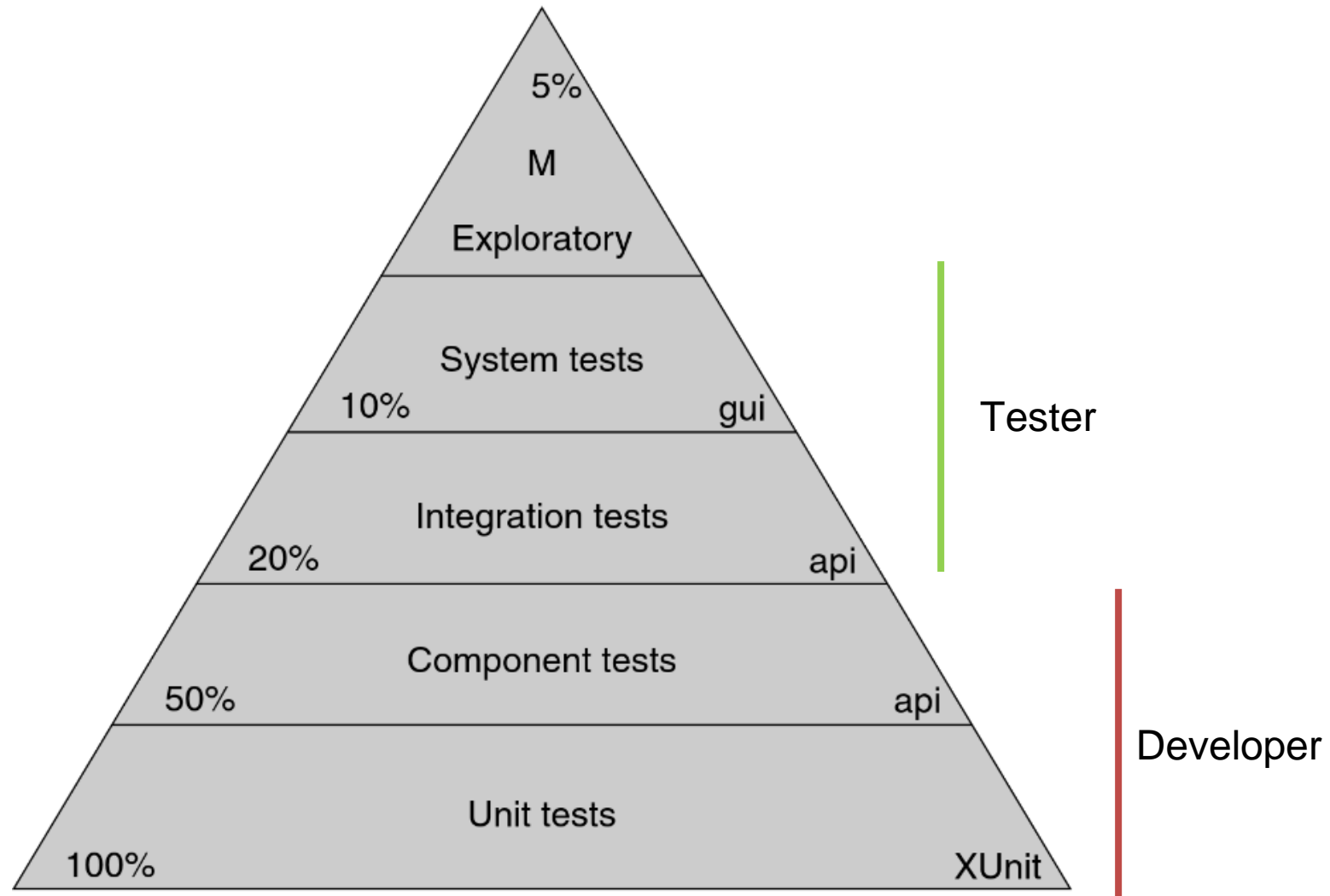
---



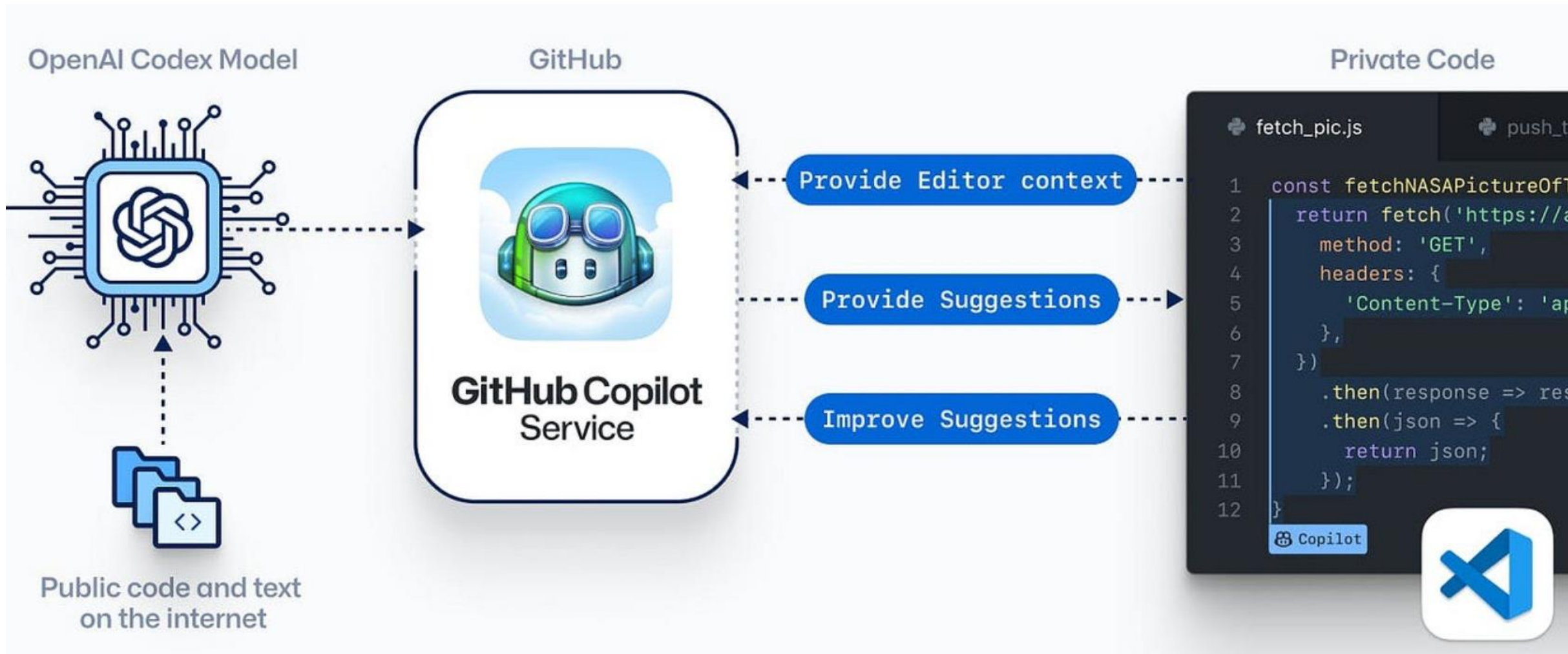
- Enable models to handle long-range dependencies in text.
- Rich context could be determined and considered.
- Not only useful for NLP
  - **Machine translation:** automatic speech translation, code translation, ...
  - **DNA sequence analysis:** predict the effects of genetic mutations, ...
  - **Protein structure analysis:** model long chains of amino acids that fold into complex protein structures

# Software Testing Pyramid

---



# Github Copilot



# Github Copilot

## Models and Comparison:

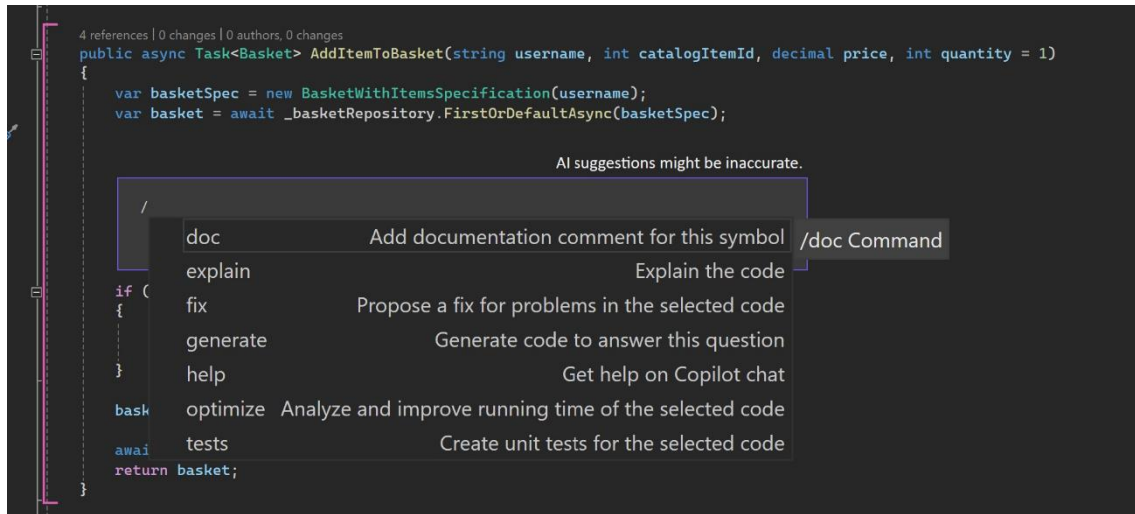
Model name	Provider	Release status	Agent mode	Ask mode	Edit mode
GPT-4.1	OpenAI	GA	✓	✓	✓
GPT-5	OpenAI	GA	✓	✓	✓
GPT-5 mini	OpenAI	GA	✓	✓	✓
GPT-5-Codex	OpenAI	Public preview	✓	✓	✓
GPT-5.1	OpenAI	Public preview	✓	✓	✓
GPT-5.1-Codex	OpenAI	Public preview	✓	✓	✓
GPT-5.1-Codex-Mini	OpenAI	Public preview	✓	✓	✓
GPT-5.1-Codex-Max	OpenAI	Public preview	✓	✓	✓
GPT-5.2	OpenAI	Public preview	✓	✓	✓
Claude Haiku 4.5	Anthropic	GA	✓	✓	✓
Claude Opus 4.1	Anthropic	GA	✗	✓	✓
Claude Opus 4.5	Anthropic	Public preview	✓	✓	✓

See <https://docs.github.com/en/copilot/reference/ai-models/model-comparison> for comparison



# Github Copilot

## Commands:



Full list of commands in *labs/Resources*

## GitHub Copilot Slash Commands

<code>/clear</code>	Clear the current chat conversation.
<code>/delete</code>	Delete a conversation.
<code>/new</code>	Start a new conversation/project.
<code>/rename</code>	Rename the current conversation.
<code>/explain</code>	Ask Copilot to explain selected code.
<code>/fix</code>	Propose a fix for problems in selected code.
<code>/fixTestFailure</code>	Find and fix a failing test.
<code>/tests</code>	Generate unit tests for the selected code.
<code>/help</code>	Show quick reference/help.
<code>/doc</code>	Add documentation comment for the selected symbol.
<code>/optimize</code>	Analyze and improve performance of selected code.

# Github Copilot

---

## Context:



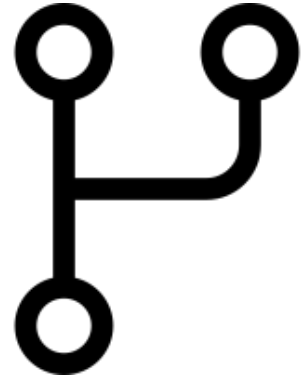
### Current File

The entirety of the file, not just where the cursor is



### Tabs Open in Editor

Tabs closest to the active tab given priority



### Entire Codebase

Without any open file

# Github Copilot

## Best supported programming language:

- Which one is the best supported programming language?
- *As more structured, as better supported!*

Language [GitHub Copilot](#)

C ✓

C++ ✓

C# ✓

Go ✓

Java ✓

JavaScript ✓

Kotlin ✓

PHP ✓

Python ✓

Ruby ✓

Rust ✓

Scala ✓

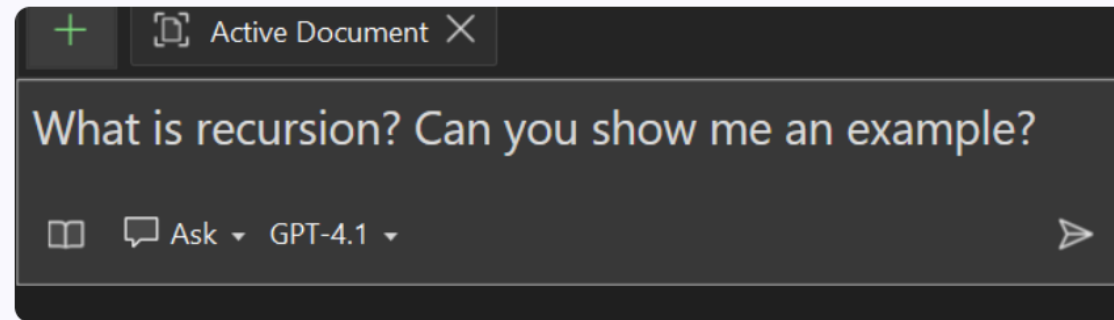
Swift ✓

TypeScript ✓

# Github Copilot Agent vs. Ask Mode

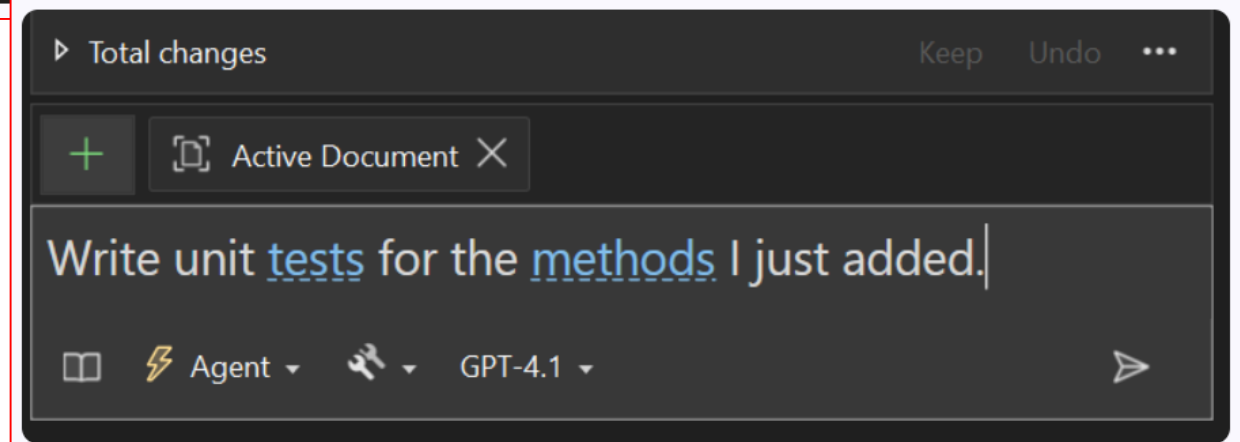
## Understanding Ask Mode

Ask Mode is your go-to setting when you need quick, conversational support—think of it as asking an experienced developer for advice, troubleshooting, or code samples. In ask mode, Copilot Chat doesn't directly interact with your workspace files; instead, it provides responses based on the context you provide.



## Understanding Agent Mode

Agent Mode takes things a step further by allowing Copilot Chat to act as an intelligent agent within your codebase. Here, Copilot can reason about your actual project files, execute commands, make edits, and even help refactor or generate new code directly in your solution.



# Github Copilot Agent vs. Ask Mode

---


Feature	Ask Mode	Agent Mode
Workspace Scope	Current file & selection	Entire workspace
Primary Use	Learning & guidance	Code analysis & modification
Response Speed	Fast	May take longer (analyzes workspace)
Code Changes	Provides suggestions	Can make direct edits
Context Awareness	Active file & selection	Multi-file project context
Best For	Conceptual questions	Refactoring & automation


# Github Copilot Agent vs. Ask Mode

Feature	Ask Mode	Agent Mode
Workspace Scope	Current file & selection	Entire workspace
Primary Use	Learning & guidance	Code analysis & modification
Response Speed	Fast	May take longer (analyzes workspace)
Code Changes	Provides suggestions	Can make direct edits
Context Awareness	Active file & selection	Multi-file project context
Best For	Conceptual questions	Refactoring & automation

# Github Copilot in Azure Boards

[Learn](#) / [Azure](#) / [Azure DevOps](#) / [Azure Boards](#) /

 Ask Learn

 Focus mode

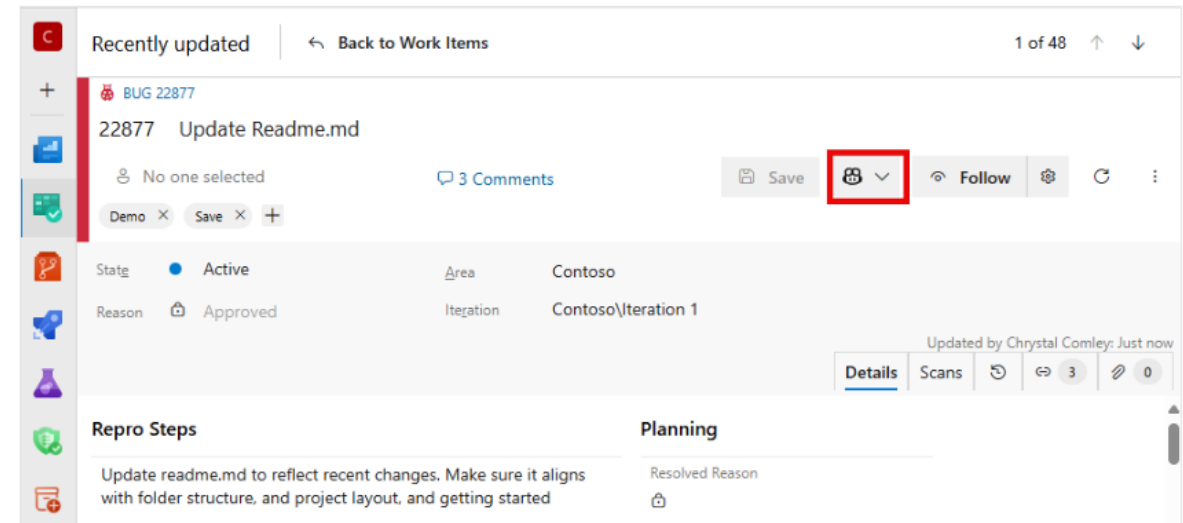
## Use GitHub Copilot with Azure Boards

Azure DevOps Services

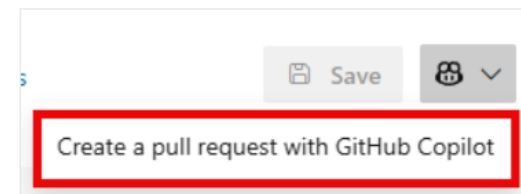
### Note

This feature is in Private Preview. Access is limited and functionality might change before general availability.

If multiple options are available, the button becomes a dropdown menu with different choices.



3. Select **Create a pull request with GitHub Copilot**.



# Prompt Engineering

---

We need to know how to use an LLM first!

See „01 Prompt engineering best practises.pdf“



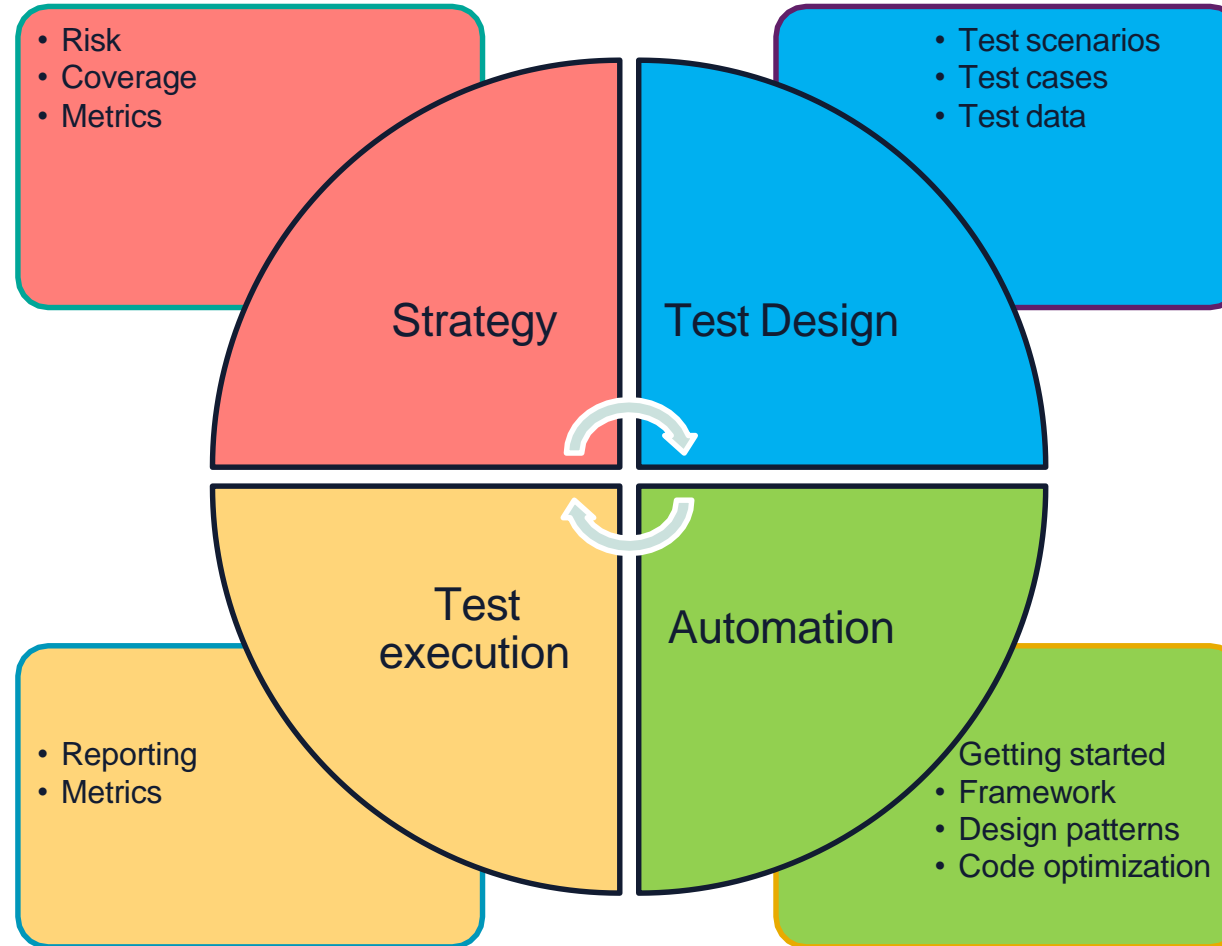
# LLMs in Testing

## Example: Testing a Website

```
set      test  3>>x  7D  x  fp  {:4}
{'a':1,'b':2}  fp  test  3>>x  set
7D  ={'a':1}  >>  {:4}  =new
fp  {:4}  3>>x  x  fp  4>>x={'a':1}
3>>x  set  {'a':1,'b':2}  =new
x  >>  7D  ={'a':1}  3>>x
test  {:4}  fp  set  >>
```

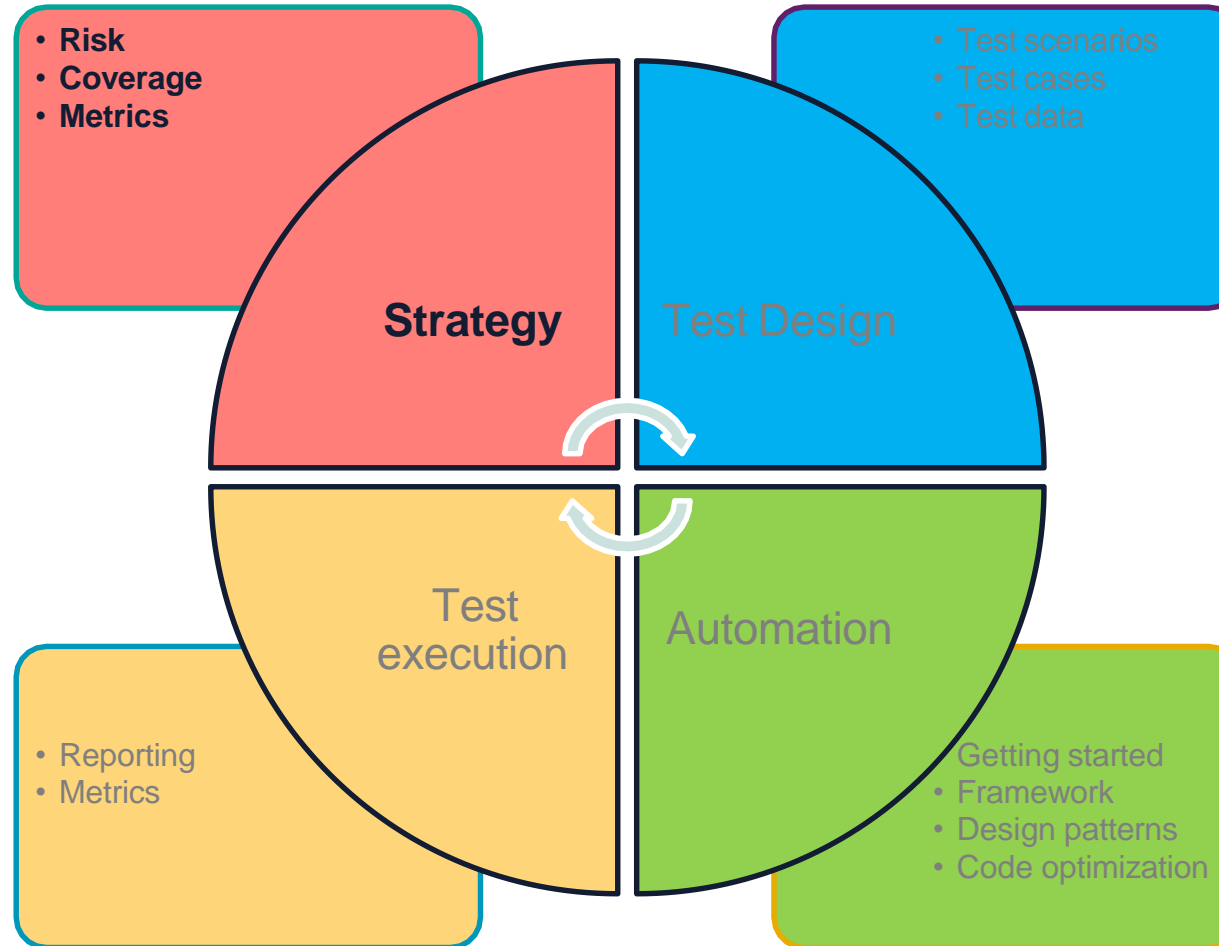
# Testing Approach

---



# Testing - Strategy

---



# Used Application

---

## Spring Petclinic:

- Frontend UI: <http://ec2-54-188-50-153.us-west-2.compute.amazonaws.com:8000/>
- Backend API: <http://ec2-54-188-50-153.us-west-2.compute.amazonaws.com:9966/petclinic/v3/api-docs>

# Testing - Strategy

---

## Question asked:

*Create a test strategy for  
testing and automating  
the website under  
<http://..>*

# Testing - Strategy

## Question asked:

*Create a test strategy for testing and automating Amazon USA*



Create a test strategy for testing and automating Amazon USA



A test strategy for testing and automating Amazon USA could include the following steps:



1. Determine the scope of the testing: Define what parts of the Amazon USA website and functionality need to be tested and automated.
2. Identify the target audience: Determine who the target audience is for Amazon USA and tailor the testing approach to their needs and behaviors.
3. Define test objectives: Identify the specific objectives for each test and what outcomes are expected from each.
4. Select test tools and frameworks: Choose the appropriate test tools and frameworks for automating tests, such as Selenium or Appium.
5. Create test cases: Write test cases that cover the most important functionalities and scenarios, such as searching for products, adding products to the cart, and checking out.
6. Develop test scripts: Create automated test scripts using the selected tools and frameworks.
7. Execute tests: Run the automated tests regularly, especially after major updates to the Amazon USA website, to ensure it continues to function as expected.
8. Analyze and report results: Analyze the results of the automated tests and report any issues found, including screenshots and detailed error descriptions.
9. Maintain and update tests: Regularly maintain and update the automated tests to keep up with changes to the Amazon USA website and to ensure their continued accuracy.

Regenerate response

# Testing - Strategy

## Question asked:

*Create a test strategy for testing and automating Amazon USA*



Create a test strategy for testing and automating Amazon USA



A test strategy for testing and automating Amazon USA could include the following steps:



1. Determine the scope of the testing: Define what parts of the Amazon USA website and functionality need to be tested and automated.
2. Identify the target audience: Determine who the target audience is for Amazon USA and tailor the testing approach to their needs and behaviors.
3. Define test objectives: Identify the specific objectives for each test and what outcomes are expected from each.
4. Select test tools and frameworks: Choose the appropriate test tools and frameworks for automating tests, such as Selenium or Appium.
5. Create test cases: Write test cases that cover the most important functionalities and scenarios, such as searching for products, adding products to the cart, and checking out.
6. Develop test scripts: Create automated test scripts using the selected tools and frameworks.
7. Execute tests: Run the automated tests regularly, especially after major updates to the Amazon USA website, to ensure it continues to function as expected.
8. Analyze and report results: Analyze the results of the automated tests and report any issues found, including screenshots and detailed error descriptions.
9. Maintain and update tests: Regularly maintain and update the automated tests to keep up with changes to the Amazon USA website and to ensure their continued accuracy.

**This is not useful.  
Seems like a template**

🔄 Regenerate response

# Testing - Strategy

---

## Follow-up question:

*Access the page [http:// ...](http://...)*

*and give a concrete strategy*

*for the above use case.*



# Testing - Strategy

## Follow-up

**question:** *Access the page [http:// ...](http://...) and give a concrete strategy for the above use case.*



Give a concrete strategy for the above use case



Here's a more detailed concrete test strategy for testing and automating Amazon USA:  

### 1. Test Objectives:

- Functional testing to ensure that all features of Amazon USA are working as expected.
- Performance testing to measure the response time, load capacity, and scalability of the system.
- Security testing to assess the security of the system and ensure that sensitive data is protected.

### 1. Test Environment:

- Amazon USA will be tested on various platforms such as Windows, Mac, and Linux.
- Different web browsers such as Google Chrome, Mozilla Firefox, and Microsoft Edge will be used.
- The testing environment will include a testing server with the necessary hardware and software configurations.

# Testing - Strategy

---

## Follow-up

**question:** *Access the page [http:// ...](http://...) and give a concrete strategy for the above use case.*

### 1. Test Cases:

- A comprehensive set of test cases will be created to cover all functionality of Amazon USA, including but not limited to:
  - User registration and login
  - Product search and filtering
  - Adding products to the cart
  - Checkout and payment
  - Order tracking
  - Customer service
- Negative scenarios, such as incorrect inputs and invalid operations, will also be tested.

### 1. Test Data:

- Test data will be created or sourced from existing data to support functional testing.
- Test data will also be used for performance testing to simulate real-world conditions.

### 1. Test Automation:

- Selenium WebDriver will be used for automating the functional tests.
- Apache JMeter will be used for performance testing.

# Testing - Strategy

## Follow-up

**question:** *Access the page [http:// ...](http://...) and give a concrete strategy for the above use case.*

### 1. Test Execution:

- The test cases will be executed in a phased approach, starting with functional testing, followed by performance testing, and finally security testing.
- The test execution will be done in a controlled environment and the test results will be recorded and analyzed.

### 1. Test Metrics:

- Test coverage, test success rate, and test execution time will be measured and used to evaluate the effectiveness of the testing process.
- Performance metrics such as response time, load capacity, and scalability will be recorded and analyzed.

**This is promising.  
A great start!**

### 1. Test Reporting:

- Test results will be reported using JIRA, including both pass and fail results.
- Test results will be analyzed and any bugs or issues will be reported and tracked.

### 1. Test Maintenance:

- Test cases, test data, and test results will be regularly reviewed and updated.
- Outdated or irrelevant test cases will be archived.

This strategy provides a systematic approach for testing and automating Amazon USA, ensuring that all features are thoroughly tested and any issues are identified and resolved in a timely manner.

# Lab

---

- Ask copilot to describe the content of the website first.
- Create your own test strategy
- We use **lab-s02** (fresh copy of s01)

# Testing – Strategy with Risk Mitigation

---

## Next question:

*What risks and their mitigation strategies should be included in the above strategy?*

# Testing – Strategy with Risk Mitigation



## Next question:

*What risks and their mitigation strategies should be included in the above strategy?*



What risks and their mitigation strategies should be included in the above strategy?



Here are some common risks and their mitigation strategies that should be included in the test strategy for testing and automating Amazon USA:  

1. Risk of insufficient testing:

- Mitigation strategy: Define comprehensive test cases and allocate adequate time and resources for testing to minimize the risk of insufficient testing.

1. Risk of inadequate test automation:

- Mitigation strategy: Choose appropriate test automation tools and frameworks and regularly review and update the test automation script to minimize the risk of inadequate test automation.

1. Risk of test environment instability:

- Mitigation strategy: Establish a stable test environment and regularly monitor and maintain it to minimize the risk of test environment instability.

1. Risk of inaccurate test data:

- Mitigation strategy: Verify the accuracy of the test data and create or source test data that represents real-world scenarios to minimize the risk of inaccurate test data.

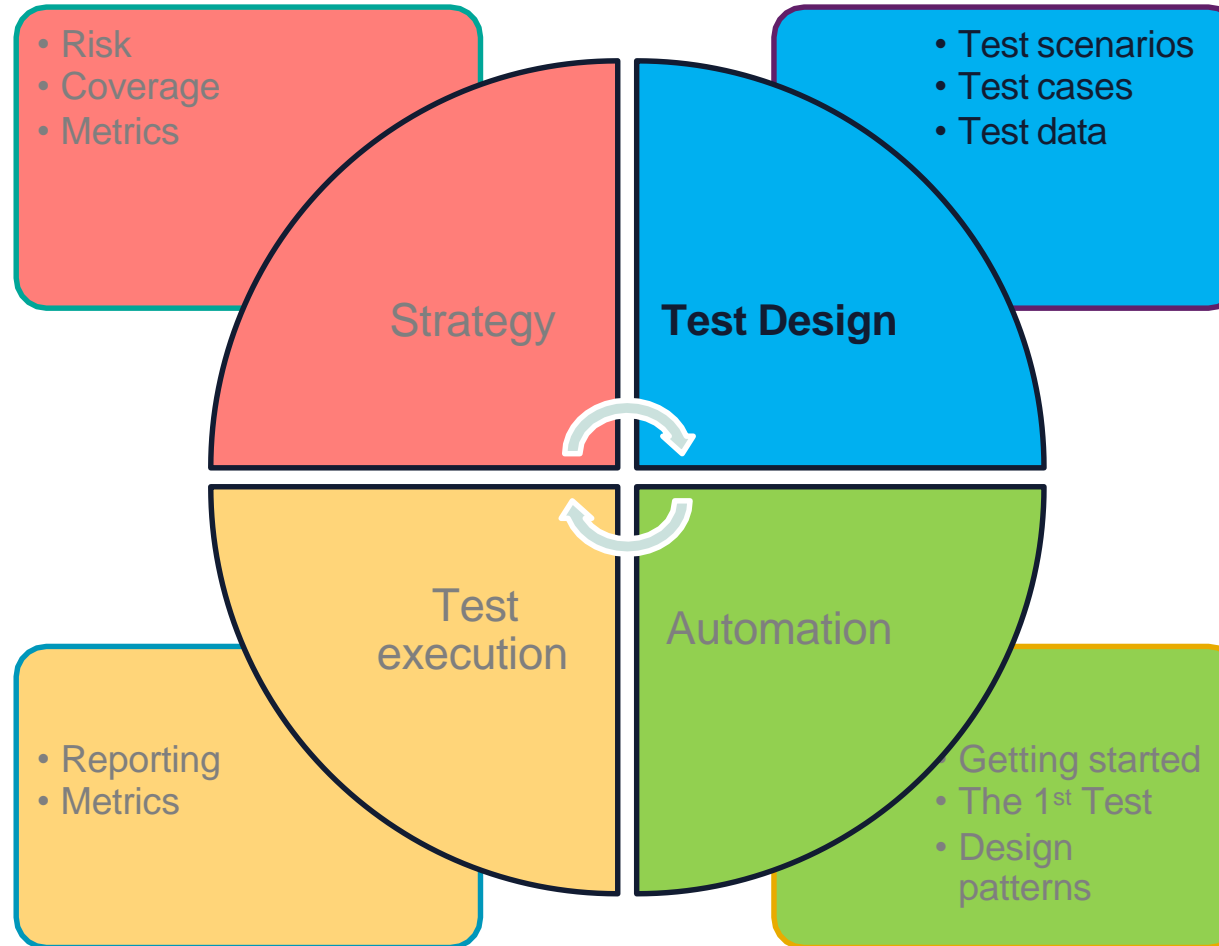
# Lab

---

Add risk mitigation to your strategy

# Testing – Test Design

---





# Test Design – Test Scenarios

---

## **Next question:**

*What are the most  
critical test scenarios  
I should test for this  
website?*

# Test Design – Test Scenarios

## Next question:

*What are the most critical test scenarios I should test for this website?*



What are the most critical test scenarios I should test for this website?



For testing and automating Amazon USA, the following are some of the most critical test scenarios that should be considered:



1. User Registration and Login: Ensure that users can successfully register and login to the website, and that the information is securely stored and protected.
2. Product Search and Filtering: Ensure that users can search for and filter products effectively, and that the results match their search criteria.
3. Adding Products to the Cart: Ensure that users can add products to their cart, and that the correct product and quantity is added.
4. Checkout and Payment: Ensure that the checkout process is secure and that users can complete their purchase using various payment methods.
5. Order Tracking: Ensure that users can track their orders and view their order history.
6. Customer Service: Ensure that users can contact customer service and receive prompt and effective assistance.
7. Responsiveness: Ensure that the website is responsive and works well on different screen sizes and devices.
8. Security: Ensure that the website is secure and that sensitive information, such as payment details, is protected.
9. Performance: Ensure that the website loads quickly and performs well under heavy load conditions.
10. Compatibility: Ensure that the website is compatible with various platforms, browsers, and devices.

Testing these critical scenarios is crucial for ensuring that the website functions as expected and provides a seamless user experience for its customers.

# Test Design – Test Scenarios

## Next question:

*What are the most critical test scenarios I should test for this website?*



What are the most critical test scenarios I should test for this website?



For testing and automating Amazon USA, the following are some of the most critical test scenarios that should be considered:



1. User Registration and Login: Ensure that users can successfully register and login to the website, and that the information is securely stored and protected.
2. Product Search and Filtering: Ensure that users can search for and filter products effectively, and that the results match their search criteria.
3. Adding Products to the Cart: Ensure that users can add products to their cart, and that the correct product and quantity is added.
4. Checkout and Payment: Ensure that the checkout process is secure and that users can complete their purchase using various payment methods.
5. Order Tracking: Ensure that users can track their orders and view their order history.
6. Customer Service: Ensure that users can contact customer service and receive prompt and effective assistance.
7. Responsiveness: Ensure that the website is responsive and works well on different screen sizes and devices.
8. Security: Ensure that the website is secure and that sensitive information, such as payment details, is protected.
9. Performance: Ensure that the website loads quickly and performs well under heavy load conditions.
10. Compatibility: Ensure that the website is compatible with various platforms, browsers, and devices.

Testing these critical scenarios is crucial for ensuring that the website functions as expected and provides a seamless user experience for its customers.

# Lab

---

Define critical test scenarios and cases for your tests

# Test Design – Test Cases

---

## **Next question:**

*List 5 tests for the most critical test cases in order of priority.*

# Test Design – Test Cases

## Next question:

*List 5 tests for the most critical test cases in order of priority.*



List 10 test cases for "Product Search and Filtering" scenario in order of priority



Here are 10 test cases for the "Product Search and Filtering" scenario in order of priority:



1. Verify that the search results match the keywords entered by the user.
2. Test the functionality of the search bar and ensure that it is working as expected.
3. Verify that the filters (e.g. price range, brand, category, etc.) work correctly and that the filtered results match the selected criteria.
4. Test the sorting functionality and verify that the products are sorted in the desired order.
5. Verify that the search results are displayed in a user-friendly format, with clear product information and images.
6. Test the pagination functionality and verify that the user can navigate through the pages of search results.
7. Verify that the search results are accurate and up-to-date.
8. Test the search functionality with various types of keywords, including long-tail keywords and misspelled words.
9. Verify that the search results are consistent across different platforms and devices.
10. Test the search functionality with no results scenarios and verify that the user receives appropriate feedback.

# Lab

---

- Use your LLM to create test descriptions.
- Try to guide the LLM to define test boundaries/edge cases for the most critical tests.

# Test Design – Test Cases

---

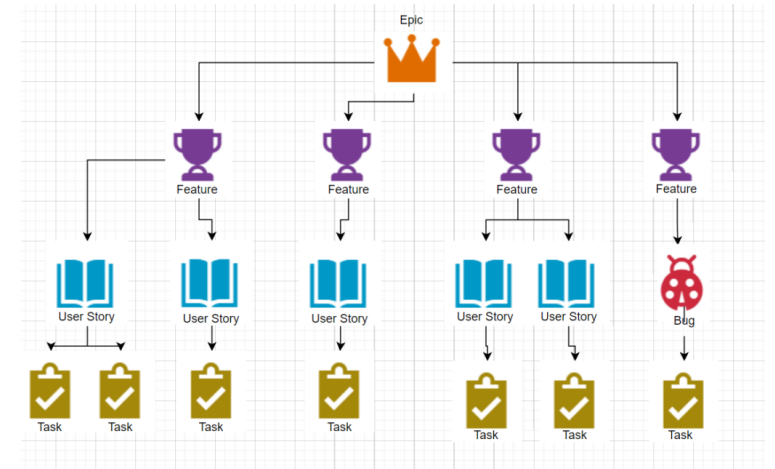
What can we do with the tests from within the Chat?

Implement them (already done before)!



# Test Design – New Feature

- Existing code until now!
- Tests for features before the implementation has been started.
- Test driven development (TDD) is possible with Copilot as well.

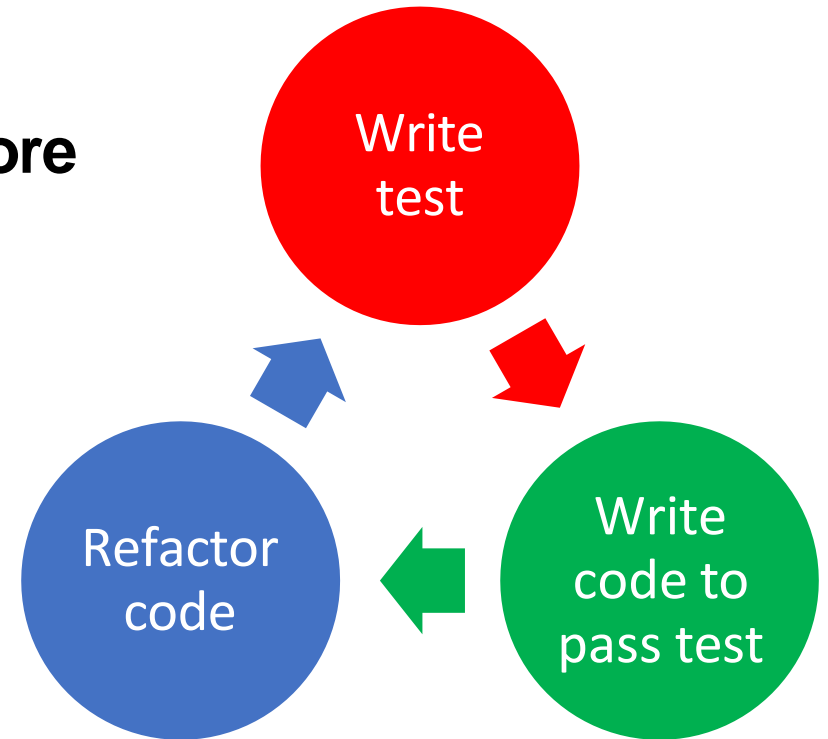


# Test Driven Development

---

## Test driven development (TDD):

- What:
  - Test is based on the spec and is developed **before** the code is written
  - Will fail initially
  - Write just enough code to make it pass!
- Why?

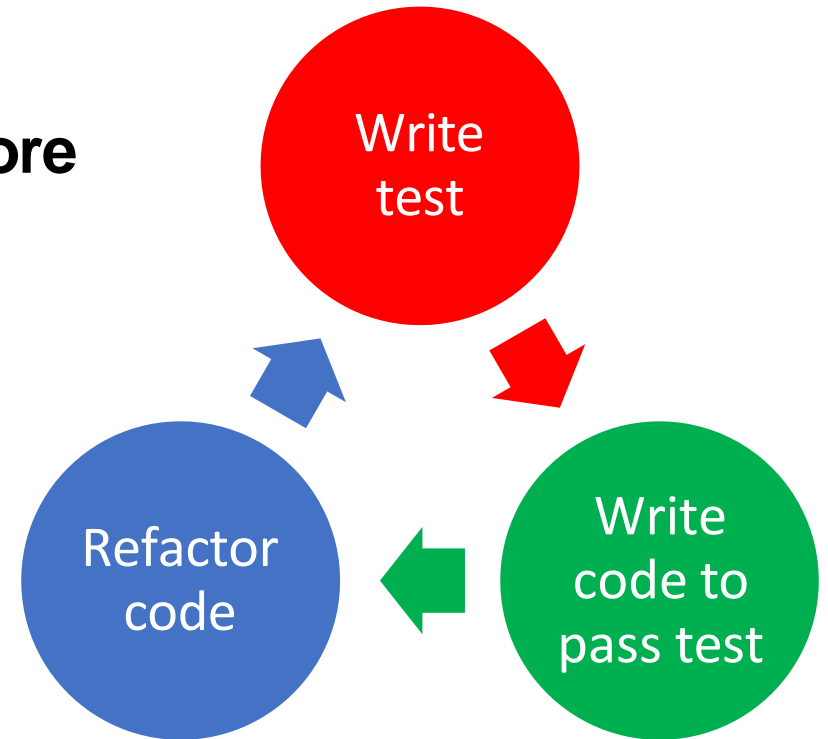


# Test Driven Development

---

## Test driven development (TDD):

- What:
  - Test is based on the spec and is developed **before** the code is written
  - Will fail initially
  - Write just enough code to make it pass!
- Why?
  - In practice, significantly less defect rate
  - Improved understanding of requirements and ability to influence design
  - Not influenced by implementation choices



# Lab

---

- Use **lab-s4** which is a **fresh copy of lab-s1**.
- Open and read *requirements\_user\_registration.md* first.
- Ask Copilot to create a new module in Gherkin language for the tests and add also the required steps to the solution.
- Evaluate the results.

# Test Data Generation

---

- LLMs have strong capabilities to generate coherent text.
- Can be useful for generating data which mimic realistic test cases.
- Ask what data is required first.
- We can use what we have learned before:
  - Sentiment analyse,
  - Few-Shot Prompting by providing sample data,
  - Case or edge case identification
  - ...

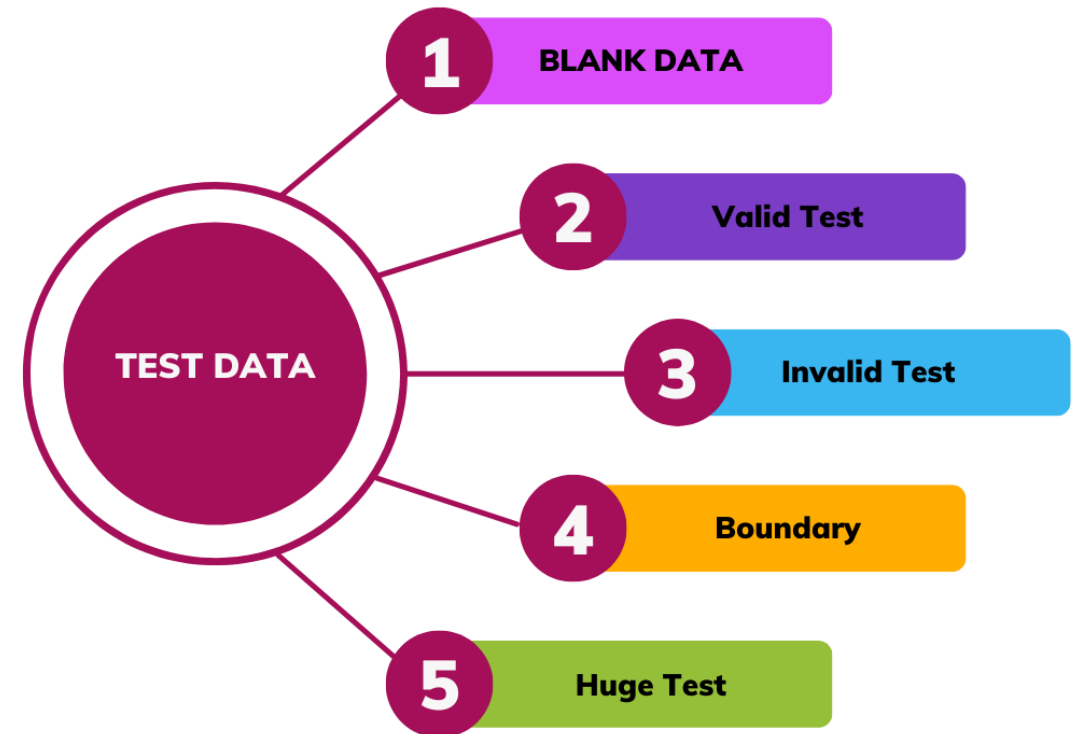
# Test Design – Test Data

## Test Data Sets

Exceptional Data (Out of Range Data)

Exceptional data is used to test that the program can properly handle unexpected inputs.

Employee Name	Employee Age	Expected Output	Actual Output	Pass/Fail
Kris Blair	1000	Invalid Employee	Invalid Employee	✓
Julie Harris	-1	Invalid Employee	Invalid Employee	✓
Rick Thomas	Nineteen	Invalid Employee	Invalid Employee	✓
Annette Reece	abc	Invalid Employee	Invalid Employee	✓



# Test Design – Test Data

## Next question:

*What test data do I need to be available for performing the above test cases?*



What test data do I need to be available for performing the above test cases?



For performing the above test cases for the "Product Search and Filtering" scenario, the following test data should be available:



1. Product information: Information about the products that are available on Amazon USA, including product name, description, images, price, brand, category, etc.
2. Keywords: A variety of keywords that can be used to search for products, including common keywords, long-tail keywords, misspelled words, and edge cases.
3. Filter criteria: Information about the various filters that are available on Amazon USA, including price range, brand, category, etc. and their values.
4. Sort criteria: Information about the various sorting options that are available on Amazon USA, including sort by relevance, price, rating, etc.
5. Product variations: Information about different variations of products, including size, color, etc.
6. Search results: The expected search results for different keywords and filter/sort criteria, to validate the accuracy of the search functionality.

Having this test data available is crucial for effectively testing the "Product Search and Filtering" functionality on Amazon USA and verifying that it is working as expected.

# Test Design – Test Data

## Next question:

*What test data do I need to be available for performing the above test cases?*

-> be more precise



What test data do I need to be available for performing the above test cases?



For performing the above test cases for the "Product Search and Filtering" scenario, the following test data should be available:



1. Product information: Information about the products that are available on Amazon USA, including product name, description, images, price, brand, category, etc.
2. Keywords: A variety of keywords that can be used to search for products, including common keywords, long-tail keywords, misspelled words, and edge cases.
3. Filter criteria: Information about the various filters that are available on Amazon USA, including price range, brand, category, etc. and their values.
4. Sort criteria: Information about the various sorting options that are available on Amazon USA, including sort by relevance, price, rating, etc.
5. Product variations: Information about the various variations of products, including size, color, etc.
6. Search results: The expected search results for different keywords and filter/sort criteria, to validate the accuracy of the search functionality.

ok to start but not that helpful.

Having this test data available is crucial for effectively testing the "Product Search and Filtering" functionality on Amazon USA and verifying that it is working as expected.



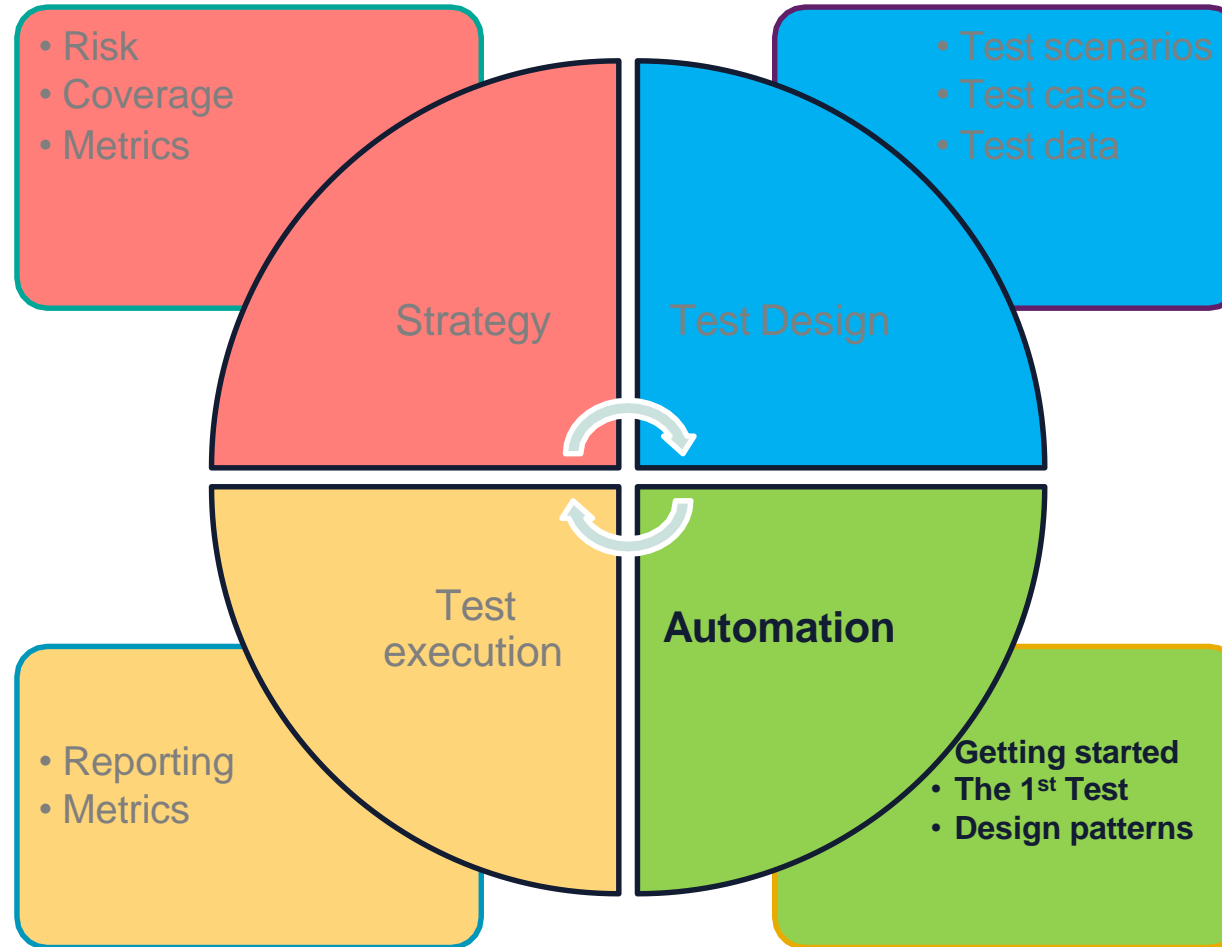
# Lab

---

- Use the “*Accommodation Booking*” from the reference application [here](#) and generate 20 test data examples.
- Use **lab-s5** which is a **fresh copy of lab-s1**.
  1. Let Copilot summarize the purpose of the page first.
  2. After that, tell Copilot to generate test data in a new file.
    - a) Take at least 10 edge cases into account.
    - a) Create data for positive and negative cases as well.
- If the data is too simple, re-ask and use the techniques learned before.
- **Hint:** Use Copilot to generate BDD-Tests in Gherkin language and the required steps that cover the generated test data (ensure completeness).

# Testing - Automation

---



# Automation – Strategy

---

## **Next question:**

*What is a good way to  
automate the above  
test cases?*

# Automation – Strategy

## Next question:

*What is a good way to automate the above test cases?*



What is a good way to automate the above test cases?



A good way to automate the above test cases for the "Product Search and Filtering" scenario is to use a combination of the following tools and techniques:



1. Automation Framework: An automation framework, such as Selenium, that provides a structure for writing and executing tests.
2. Test Scripts: Writing automated test scripts in a programming language, such as Java or Python, that use the automation framework to perform the tests.
3. Test Data: Storing the test data in a database or in a spreadsheet, and using it in the test scripts to perform the tests.
4. Test Runners: Using a test runner, such as JUnit or TestNG, to execute the test scripts and manage the test results.
5. Continuous Integration (CI) and Continuous Deployment (CD): Integrating the test automation process into the CI/CD pipeline, to ensure that the tests are run automatically whenever changes are made to the code.
6. Reporting: Generating reports that provide detailed information about the test results, such as the number of tests passed/failed, the time taken for each test, and any error messages.
7. Maintenance: Regularly maintaining and updating the test scripts to ensure that they are up-to-date and continue to work as expected.

By automating the above test cases, you can save time and effort, ensure that the tests are consistent and reliable, and catch any issues early in the development process.

# Behavior Driven Development (BDD)

---

See "001 bddwithcucumber.pptx"

# Automation – Strategy

## Next question:

*What is a good way to automate the above test cases?*



What is a good way to automate the above test cases?



A good way to automate the above test cases for the "Product Search and Filtering" scenario is to use a combination of the following tools and techniques:



1. Automation Framework: An automation framework, such as Selenium, that provides a structure for writing and executing tests.
2. Test Scripts: Writing automated test scripts in a programming language, such as Java or Python, that use the automation framework to perform the tests.
3. Test Data: Storing the test data in a database or in a spreadsheet, and using it in the test scripts to perform the tests.
4. Test Runners: Using a test runner, such as JUnit or TestNG, to execute the test scripts and manage the test results.
5. Continuous Integration (CI) and Continuous Deployment (CD): Integrating the test automation process into the CI/CD pipeline, to ensure that the tests are run automatically whenever changes are made to the code.
6. Reporting: Generating reports that provide detailed information about the test results, such as the number of tests passed/failed, the time taken for each test, and any error messages.
7. Maintenance: Regularly maintaining and updating the test scripts to ensure that they are up-to-date and continue to work as expected.

By automating the above test cases, you can save time and effort, ensure that the tests are consistent and reliable, and catch any issues early in the development process.

# Automation – Getting Started

## Next question:

*I don't know*

*Selenium. How should*

*I learn?*



I don't know Selenium. How should I learn?



To learn Selenium, you can follow these steps:

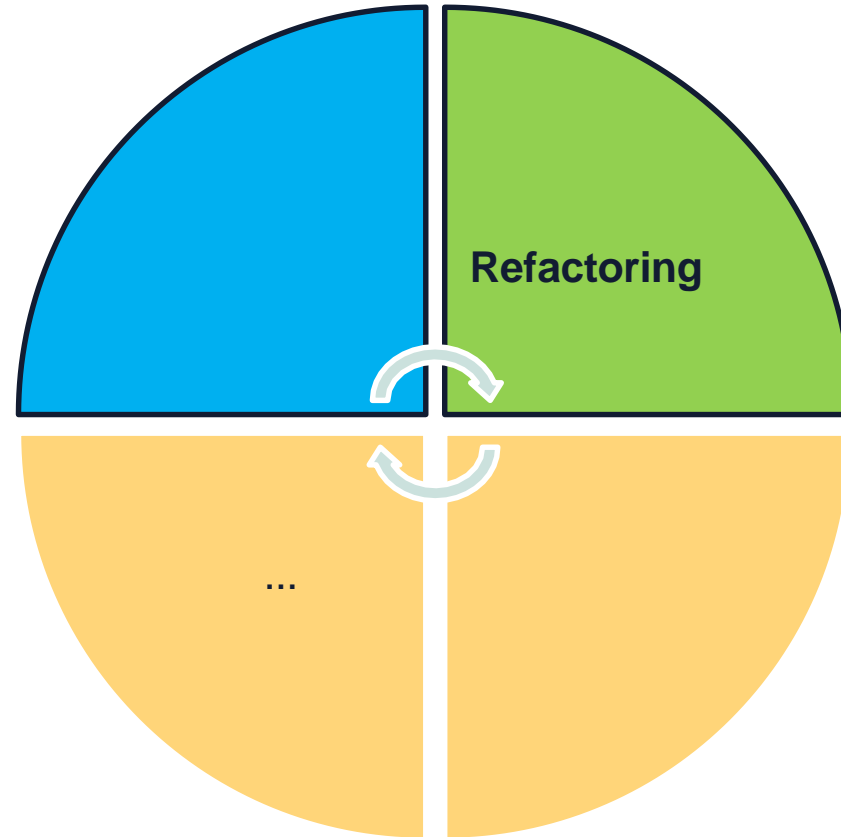


1. Familiarize yourself with the basics: Start by learning the basics of Selenium and its components, such as Selenium WebDriver, Selenium Grid, and Selenium IDE.
2. Get familiar with a programming language: Selenium is written in a programming language, so you'll need to learn one to use it effectively. Some popular languages for Selenium are Java, Python, and C#. Choose one and start learning.
3. Follow online tutorials and courses: There are many online tutorials and courses available for Selenium, such as Udemy, Coursera, and YouTube, which can help you get started and deepen your understanding.
4. Practice with examples: As you learn, practice with examples to reinforce your understanding. Start with simple examples, like automating the login process of a website, and work your way up to more complex scenarios.
5. Join a Selenium community: Joining a Selenium community, such as the Selenium Discord channel, can provide you with access to helpful resources, answers to your questions, and support from experienced Selenium users.
6. Work on a project: Finally, work on a real-world project that utilizes Selenium to get hands-on experience with the tool.

By following these steps and dedicating time to learning and practicing, you can gain a good understanding of Selenium and be able to use it effectively in your test automation projects.

# Automation – Refactoring to Follow Design Patterns

---





# Automation – Refactoring to Follow Design Patterns

---

## Next question:

*Implement the  
above test using  
Page- Object-Model  
design pattern*

# Automation – Refactoring to Follow Design Patterns

---

**Next question:**

*Implement the  
above test using  
Page- Object-Model  
design pattern*

```
class HomePage {
    private WebDriver driver;

    public HomePage(WebDriver driver) {
        this.driver = driver;
    }

    public SearchResultsPage search(String keyword) {
        // Locate the search box and enter the keyword
        driver.findElement(By.id("twotabsearchtextbox")).sendKeys(keyword);

        // Locate the search button and click it
        driver.findElement(By.xpath("//input[@value='Go']")).click();

        return new SearchResultsPage(driver);
    }
}

class SearchResultsPage {
    private WebDriver driver;

    public SearchResultsPage(WebDriver driver) {
        this.driver = driver;
    }

    public String getResultsHeader() {
        return driver.findElement(By.xpath("//span[contains(text(), 'results for')]")).getText();
    }
}
```

**Take with caution and  
have regression tests in  
place.**

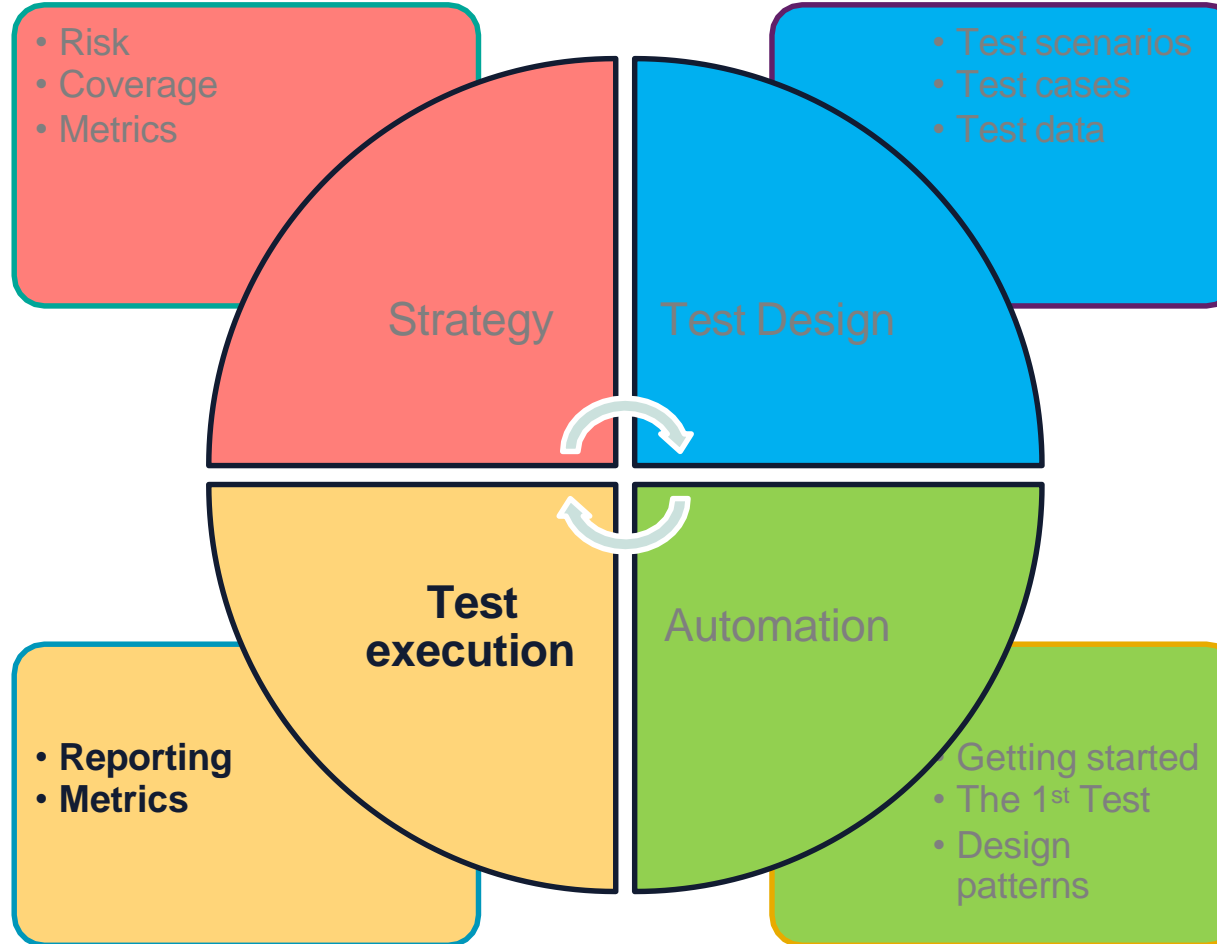
# Lab

---

- Use **lab-s6** which is a **fresh copy of lab-s1**.
- Refactor the test suites to follow Page Object Model.
- Validate if all test are still passing (*regression*)!
- What is your opinion on the quality?

# Testing – Test Execution

---



# Test Execution – Reports

---

## **Next question:**

*How to generate  
reports and metrics  
for automated tests?*

# Test Execution – Reports

---

## Next question:

*How to generate reports and metrics for automated tests?*

In order to determine the best reporting solution for your tests, consider the following factors:

- Integration: How easy is it to integrate the reporting solution into your existing testing infrastructure?
- Customization: Can you customize the appearance and content of the reports to meet your needs?
- Interactivity: How interactive are the reports? Can you drill down to the results to see more detail?
- Integration with Metrics: Can you track metrics such as test execution time, test success rate, and test coverage?
- Accessibility: How easily can stakeholders access the reports? Can they be accessed over the web, or do they need to be downloaded and viewed locally?

**This is not what I was looking for!**

# Test Execution – Reports

---

## Follow-up

**question:** *Show an example to generate Allure reports for the automated tests*

# Test Documentation

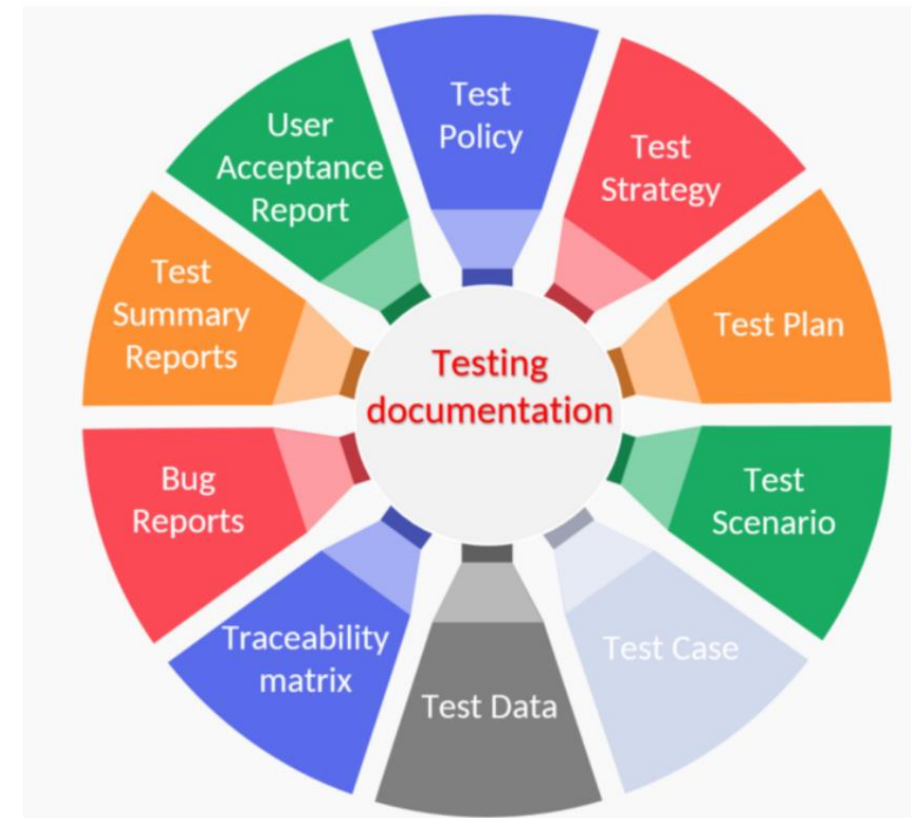
---

How can we document test cases with an LLM?

## Open Questions:

- What do we want to document?
- Documentation structure?
- Level of detail?
- References to code necessary?
- Incorporate the results into the documentation?

→ How do you do test documentation?





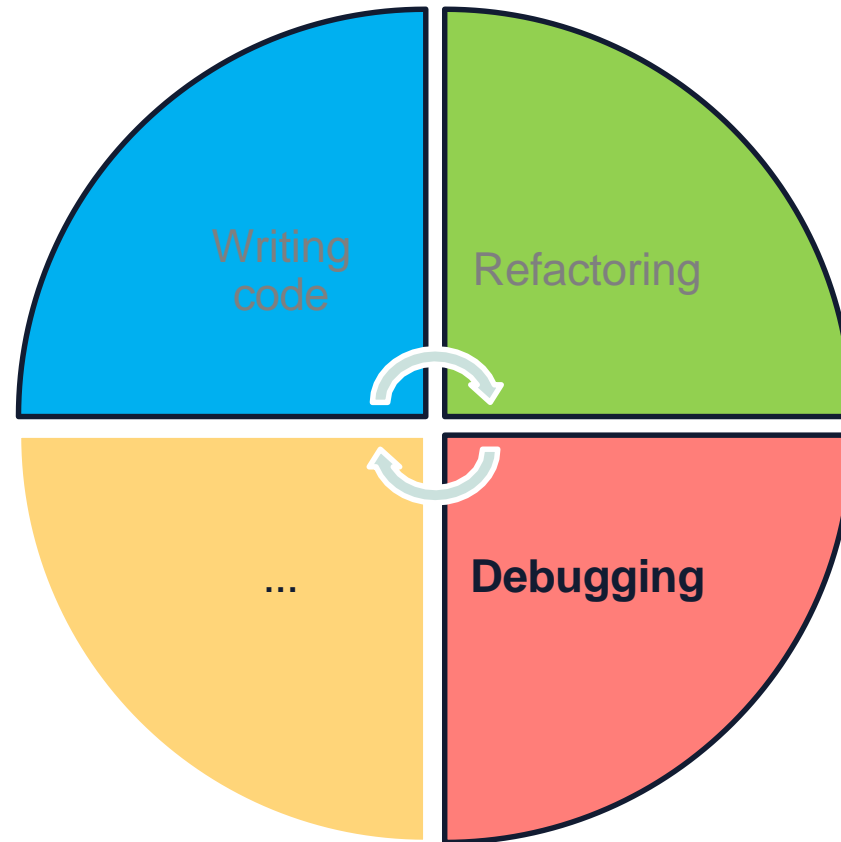
# Lab

---

- Use **lab-s4**.
- Ask Copilot to create a test documentation and include the requirements based on the questions from before.
- Tune and edit until suitable.
- Evaluate the results.

# Debugging

---



# Lab

---

- Find file **ConvertCelsius.feature** from **lab-s02**.
- Include the test that is excluded (commented).
- Rerun and validate that the test is failing.
- Ask Copilot to fix it.
- just ask to fix the failed test nothing more.
- Let Copilot find the problem on its own.

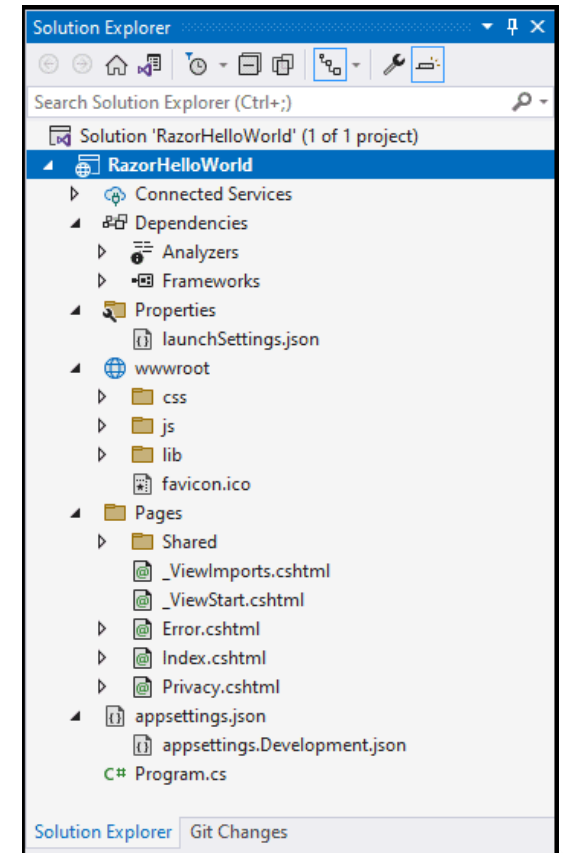
# Follow existing Solution Structure

---

How can we direct the LLM to follow the existing solution structure?

## Open Questions:

- Follow solution structure?
- Put resources into the correct locations?
- Create documentation and put it into the correct location?
- Path-specific instructions?



# Follow existing Solution Structure

---

How can we direct the LLM to follow the existing solution structure?

## Open Questions:

- Follow solution structure?
- Put resources into the correct locations?
- Create documentation and put it into the correct location?
- Path-specific instructions?

In general it tries to do it automatically.

<https://docs.github.com/en/copilot/tutorials/coding-agent/get-the-best-results#repository-wide-instructions>

