20TH ANNIVERSARY EDITION
WSO2CON ASIA
PLATFORMLESS MODERNIZATION

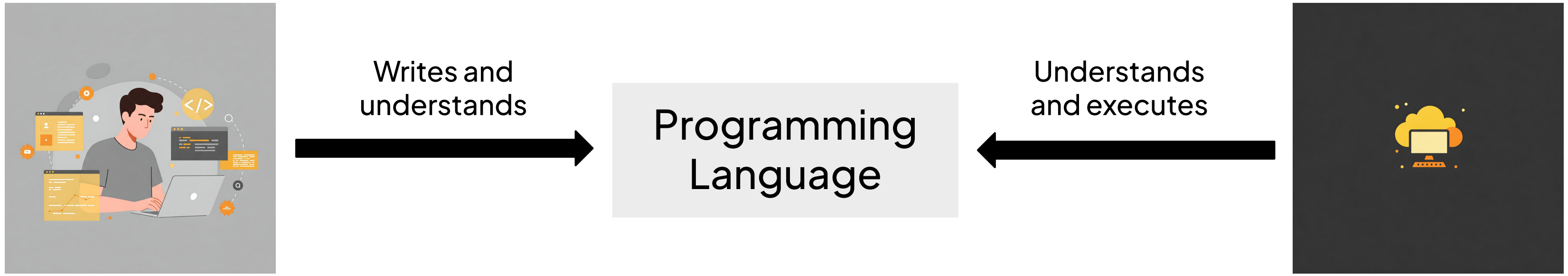
Natural Programming

Maryam Ziyad

Senior Technical Lead

WSO2

Programming



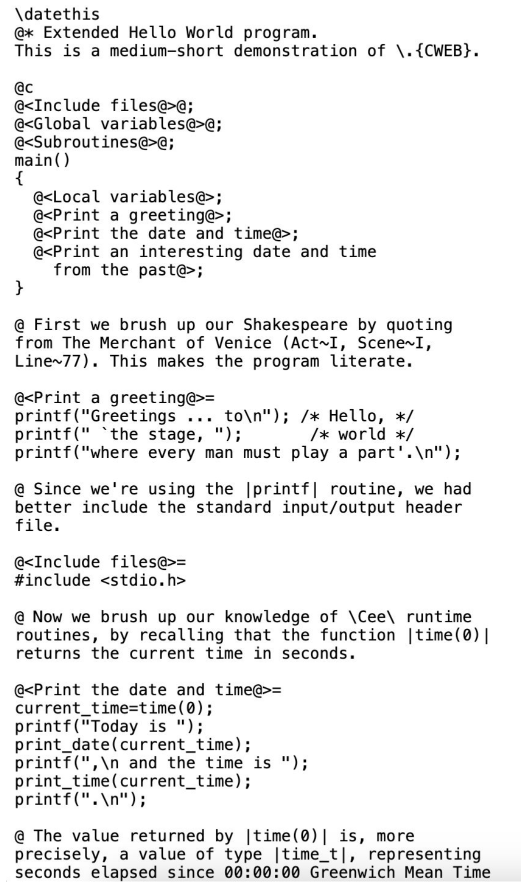
Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.

- Donald Knuth -

Literate programming

- Writing code in a way that is human-readable and understandable, intertwining natural language explanations with the code itself.
- Natural language is still for humans (developers).

Literate programming example (HWTIME – Knuth, 1992)

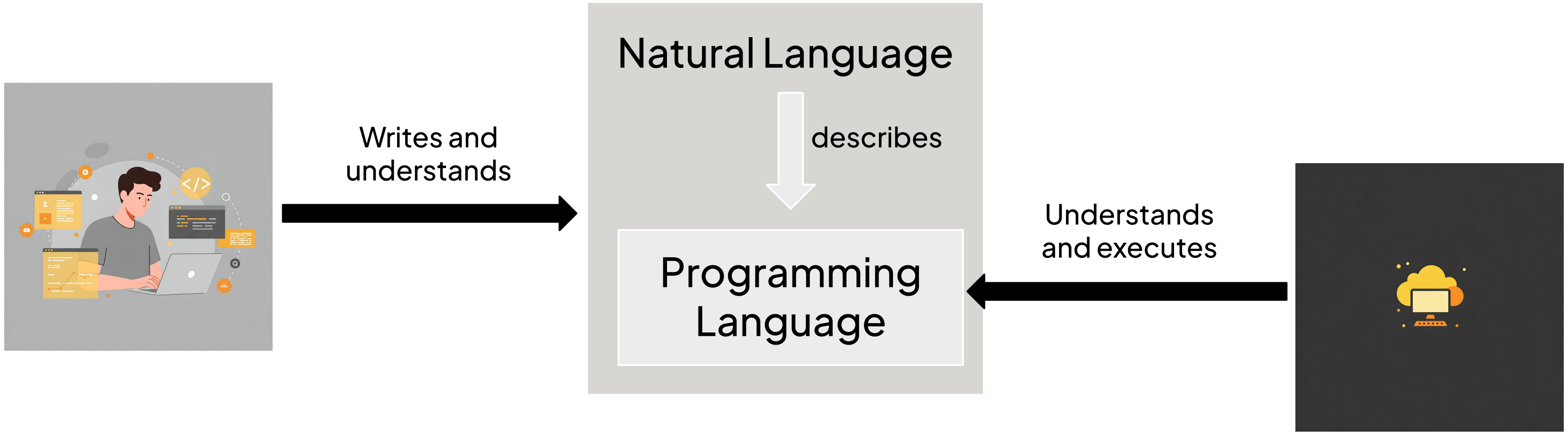


```
\datethis
@* Extended Hello World program.
This is a medium-short demonstration of \.{CWEB}.

@c
@<Include files@>@;
@<Global variables@>@;
@<Subroutines@>@;
main()
{
  @<Local variables@>;
  @<Print a greeting@>;
  @<Print the date and time@>;
  @<Print an interesting date and time
    from the past@>;
}

@ First we brush up our Shakespeare by quoting
from The Merchant of Venice (Act~I, Scene~I,
Line~77). This makes the program literate.

@<Print a greeting@>=
printf("Greetings ... to\n"); /* Hello, */
printf(" `the stage, ");       /* world */
printf("where every man must play a part'.\n");

@ Since we're using the |printf| routine, we had
better include the standard input/output header
file.

@<Include files@>=
#include <stdio.h>

@ Now we brush up our knowledge of \Cee\ runtime
routines, by recalling that the function |time(0)|
returns the current time in seconds.

@<Print the date and time@>=
current_time=time(0);
printf("Today is ");
print_date(current_time);
printf(",\n and the time is ");
print_time(current_time);
printf(".\n");

@ The value returned by |time(0)| is, more
precisely, a value of type |time_t|, representing
seconds elapsed since 00:00:00 Greenwich Mean Time
```

Literate programming example (HWTIME – Knuth, 1992)

```
@c
@<Include files@>@;
@<Global variables@>@;
@<Subroutines@>@;
main()
{
  @<Local variables@>;
  @<Print a greeting@>;
  @<Print the date and time@>;
  @<Print an interesting date and time
    from the past@>;
}

@ First we brush up our Shakespeare by quoting
from The Merchant of Venice (Act~I, Scene~I,
Line~77). This makes the program literate.

@<Print a greeting@>=
printf("Greetings ... to\n"); /* Hello, */
printf(" `the stage, ");       /* world */
printf("where every man must play a part'.\n");

@ Since we're using the |printf| routine, we had
better include the standard input/output header
file.

@<Include files@>=
#include <stdio.h>

@ Now we brush up our knowledge of \Cee\ runtime
routines, by recalling that the function |time(0)|
returns the current time in seconds.

@<Print the date and time@>=
current_time=time(0);
printf("Today is ");
print_date(current_time);
printf(",\n and the time is ");
print_time(current_time);
printf(".\n");
```

Literate programming



The hottest new programming language is English

- Andrej Karpathy -

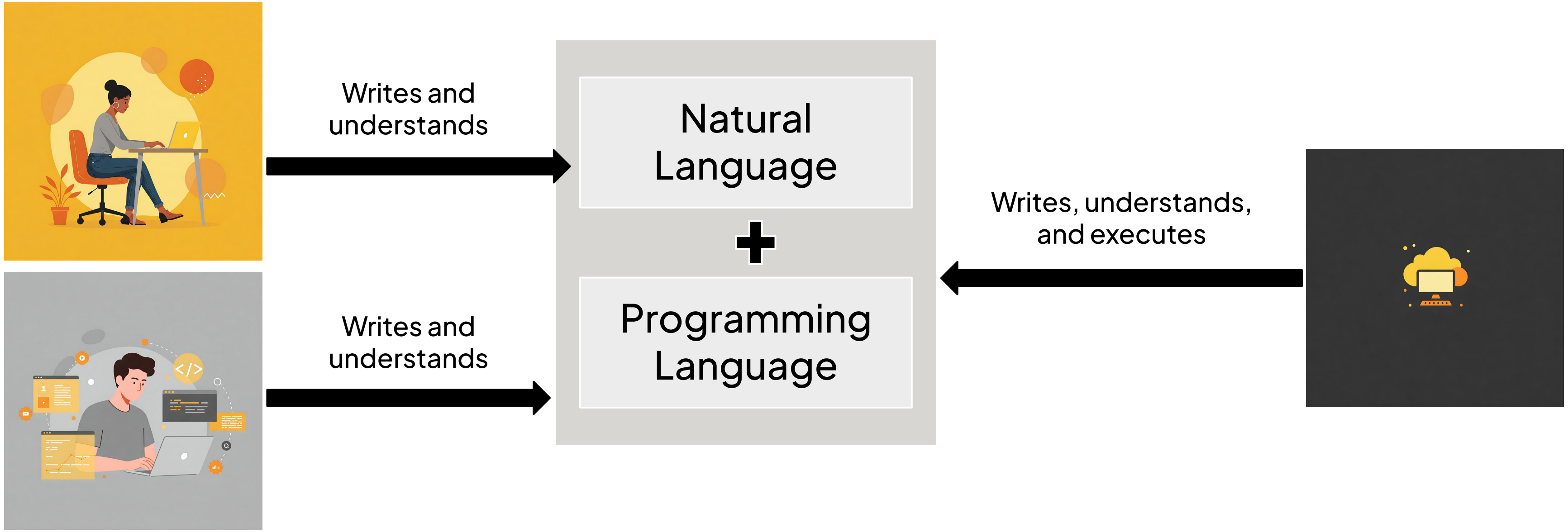
Programming today

- With the emergence of/advances in generative AI, LLMs, and Copilots, use of natural language in programming has changed significantly
  - Not just at development time, but also at runtime
- Natural language is now a more significant part of programming and coding artifacts
- Encourages increased participation in programming and software development
  - More and more non-developer/non-technical users

Natural programming

We see an evolution of programming, where the process of programming is one of **co-creation** with computers, where both human and machine express intent and actions in both natural language and in programming languages

Natural programming



Natural Language + **Programming** Language

=

Natural Programming

A blend of natural language and programming language

- Natural language is ambiguous, programming language is not
- Not suggesting replacing code entirely with natural language
- Instead,
  - leverage natural language where it can enable new functionality
  - simplify integrating natural language into programming logic
- Blend natural language and programming language to maximize value

Reference implementation in Ballerina

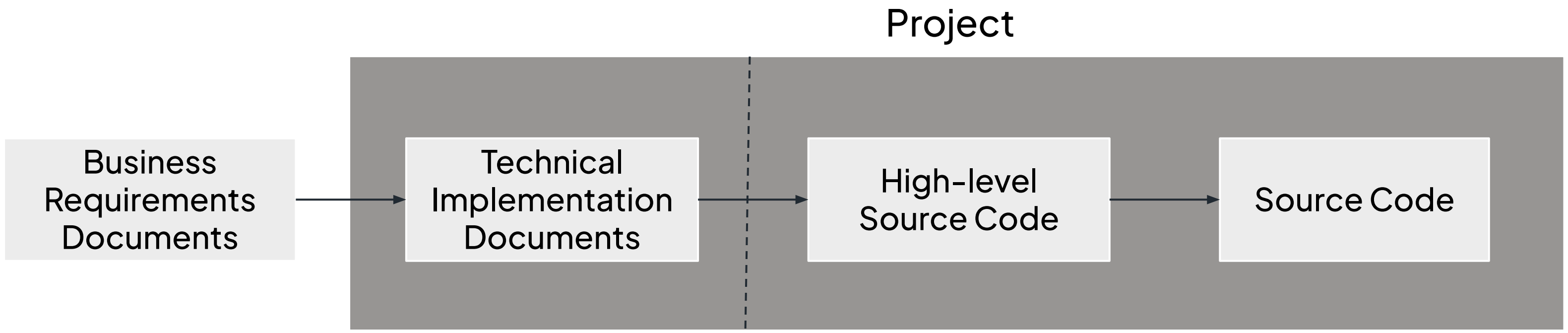
- A lot of of what we are proposing as natural programming applies/can be applied to most programming languages
- Working on a reference implementation in Ballerina
    - An open-source, cloud-native programming language optimized for integration
    - Developed by WSO2

Natural programming in the software development process and code

- Development time
- Runtime
- … and Compile time

Development time

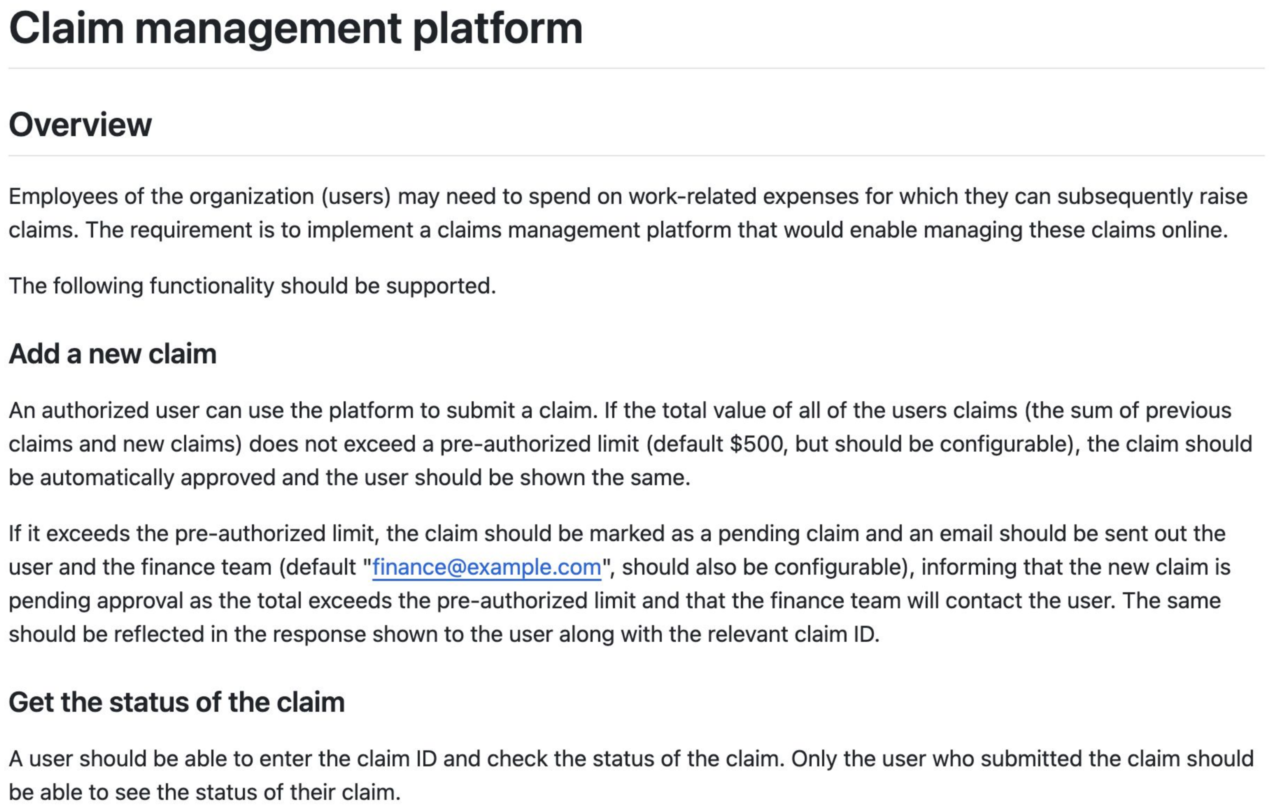
Traditional software development process


Project
Business Requirements Documents
Technical Implementation Documents
High-level Source Code
Source Code

Development time – Bringing requirements into the project

- Bridging the gap between business users and the implementation, by making business user requirements become part of the project.
- These requirements can be used to
  - Generate code
  - Generate tests
  - Check for drift between requirements and the implementation

Scenario



Claim management platform

Overview

Employees of the organization (users) may need to spend on work-related expenses for which they can subsequently raise claims. The requirement is to implement a claims management platform that would enable managing these claims online.

The following functionality should be supported.

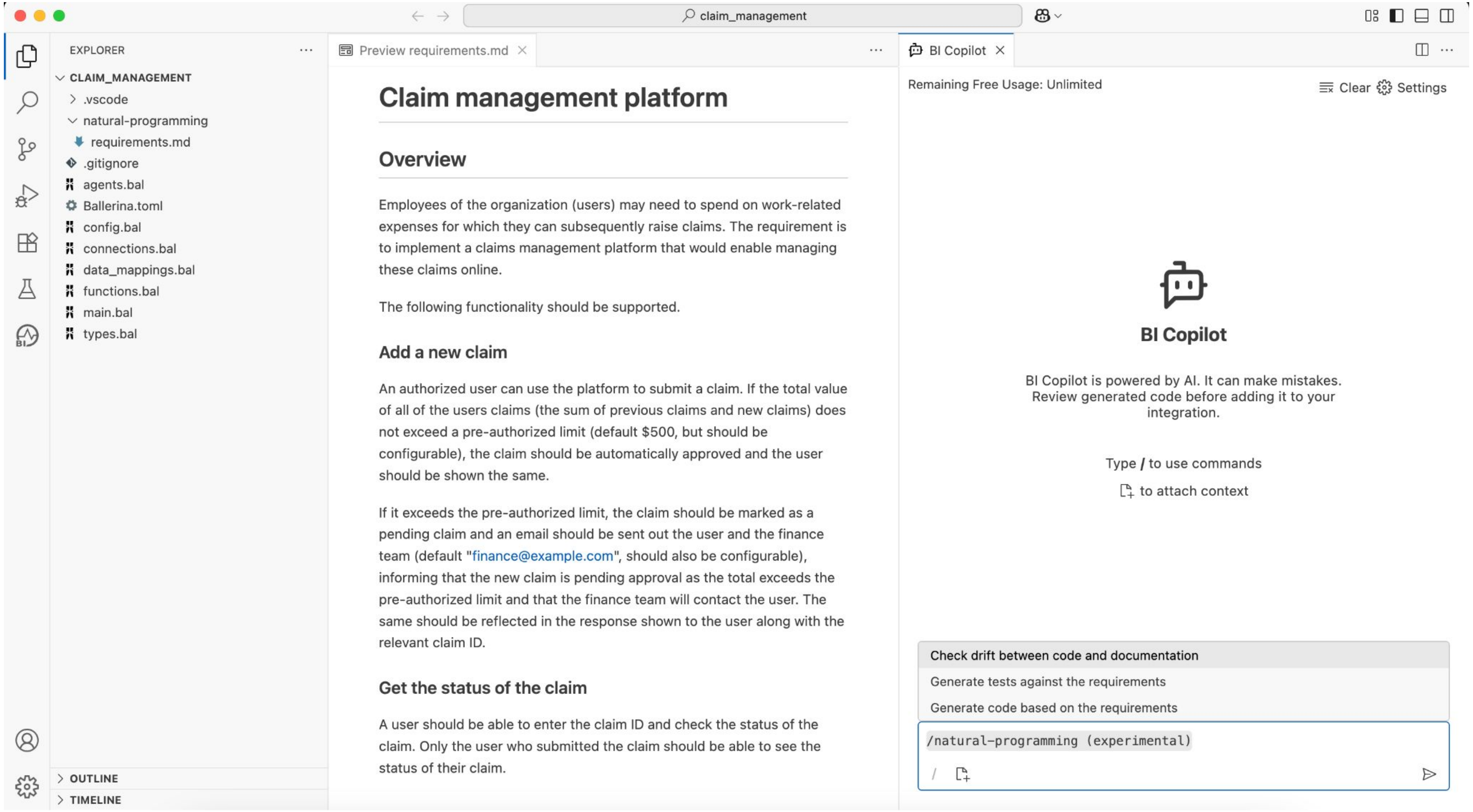
Add a new claim

An authorized user can use the platform to submit a claim. If the total value of all of the users claims (the sum of previous claims and new claims) does not exceed a pre-authorized limit (default $500, but should be configurable), the claim should be automatically approved and the user should be shown the same.

If it exceeds the pre-authorized limit, the claim should be marked as a pending claim and an email should be sent out the user and the finance team (default "finance@example.com", should also be configurable), informing that the new claim is pending approval as the total exceeds the pre-authorized limit and that the finance team will contact the user. The same should be reflected in the response shown to the user along with the relevant claim ID.

Get the status of the claim

A user should be able to enter the claim ID and check the status of the claim. Only the user who submitted the claim should be able to see the status of their claim.



Claim management platform

Overview

Employees of the organization (users) may need to spend on work-related expenses for which they can subsequently raise claims. The requirement is to implement a claims management platform that would enable managing these claims online.

The following functionality should be supported.

Add a new claim

An authorized user can use the platform to submit a claim. If the total value of all of the users claims (the sum of previous claims and new claims) does not exceed a pre-authorized limit (default $500, but should be configurable), the claim should be automatically approved and the user should be shown the same.

If it exceeds the pre-authorized limit, the claim should be marked as a pending claim and an email should be sent out the user and the finance team (default "finance@example.com", should also be configurable), informing that the new claim is pending approval as the total exceeds the pre-authorized limit and that the finance team will contact the user. The same should be reflected in the response shown to the user along with the relevant claim ID.

Get the status of the claim

A user should be able to enter the claim ID and check the status of the claim. Only the user who submitted the claim should be able to see the status of their claim.

Remaining Free Usage: Unlimited

BI Copilot

BI Copilot is powered by AI. It can make mistakes. Review generated code before adding it to your integration.

Type / to use commands

to attach context

Check drift between code and documentation

Generate tests against the requirements

Generate code based on the requirements

```
/natural-programming (experimental)
```

Code generation

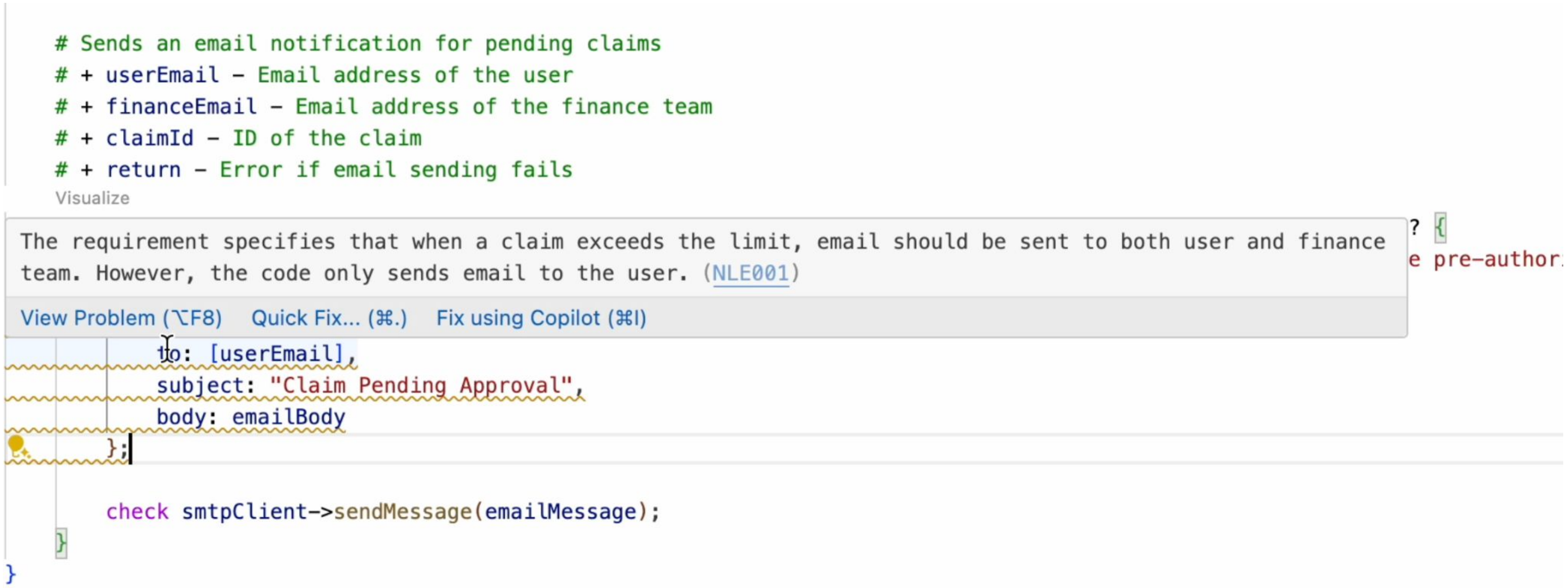
- Similar to code generation with a Copilot
- Aware that requirements can be at a higher (business user) level, and that there would generally be developer input too
- Encourages proper API documentation

Test generation

- Generate tests based on the requirements, against the implementation
    - Uses just the APIs from the implementation
- Can be considered a form of user acceptance testing
- Complements tests written/generated by a developer

Development time – Drift check

- Checks for drift between business requirements, documentation, and the implementation
- Backed by LLMs, runs periodically in the background if enabled or can run on demand. Also suggest changes to the code or the documentation to remove/minimize drift.
- Encourages proper API and project documentation
- Makes code easier to understand and evolve, while staying in sync with the requirements

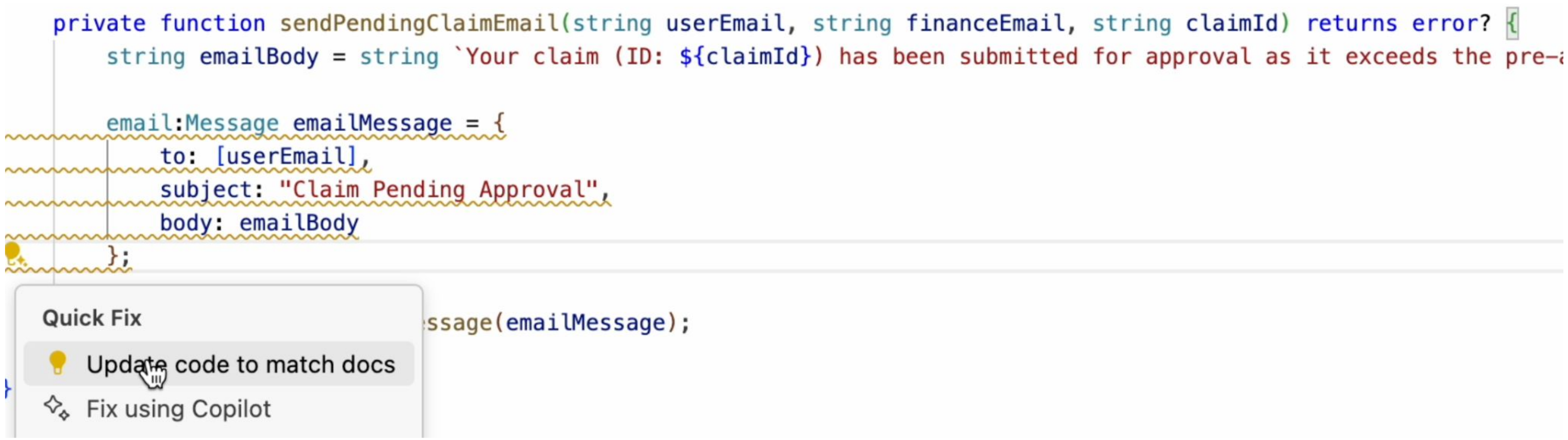


```
# Sends an email notification for pending claims
# + userEmail - Email address of the user
# + financeEmail - Email address of the finance team
# + claimId - ID of the claim
# + return - Error if email sending fails
```

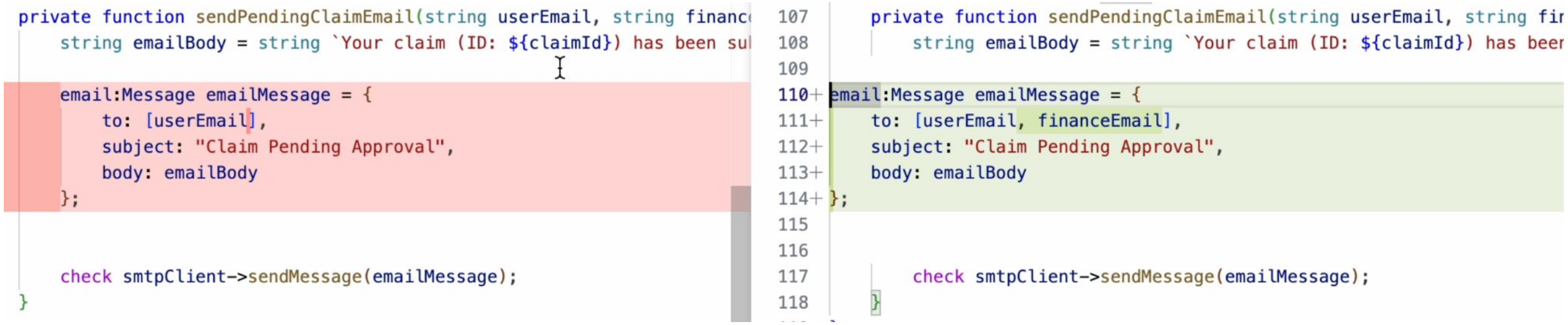
Visualize

The requirement specifies that when a claim exceeds the limit, email should be sent to both user and finance team. However, the code only sends email to the user. (NLE001)

```
? {
e pre-author
            to: [userEmail],
            subject: "Claim Pending Approval",
            body: emailBody
        };

        check smtpClient->sendMessage(emailMessage);
    }
}
```



```
private function sendPendingClaimEmail(string userEmail, string financeEmail, string claimId) returns error? {
    string emailBody = string `Your claim (ID: ${claimId}) has been submitted for approval as it exceeds the pre-a

    email:Message emailMessage = {
        to: [userEmail],
        subject: "Claim Pending Approval",
        body: emailBody
    };

                                ssage(emailMessage);

}
```



```
private function sendPendingClaimEmail(string userEmail, string financ
    string emailBody = string `Your claim (ID: ${claimId}) has been sul

    email:Message emailMessage = {
        to: [userEmail],
        subject: "Claim Pending Approval",
        body: emailBody
    };


    check smtpClient->sendMessage(emailMessage);
}
```

```
    private function sendPendingClaimEmail(string userEmail, string fir
        string emailBody = string `Your claim (ID: ${claimId}) has beer

email:Message emailMessage = {
    to: [userEmail, financeEmail],
    subject: "Claim Pending Approval",
    body: emailBody
};


        check smtpClient->sendMessage(emailMessage);
    }
```

Drift check

- A common problem with natural language in programming artifacts (e.g., comments in the source code and project documentation traditionally) is that they can go out of sync with the code.
- Drift check can also address this problem for all kinds of documentation – from comments, to API documentation, to other project documentation.

Development time

- A real-world project would generally be a collection of components like the Ballerina project.
    - Different components can use different languages, frameworks, etc.
- There can be other kinds of natural language documentation (e.g., policies, guidelines, etc.) applicable at different levels (organization, project, component, etc.) that can be leveraged.

Development time

- Similar to drift check, these descriptions and specifications in natural language can be used to analyze, validate, and align implementations.
- Including, identifying
    - Project completion status
    - Compliance with organizational policies and guidelines

Development time – Preserving user intent

- Use of AI-powered tools for code generation continues to increase
- Usually an interactive/iterative process between user and AI assistant
- Prompts are generally discarded once the user is satisfied with the generated code, but the prompts can contain useful contextual information that doesn’t get reflected entirely in the code.
- Another form of natural language input that would be useful to preserve in some form

Preserving user intent

- With natural programming, we preserve a summary of the user intentions (via prompts) that led to the specific version of the code, where the code is AI generated
  - Currently added in a developer.md file, but we’re exploring integrating it into the code itself
- Not a history of user interactions, but an effective summary

Preserving user intent – Sample

Claim management implementation, but assume the implementation was generated

- maintaining claim data in memory
- sending two separate emails to the user and the finance team

Preserving user intent – Sample prompts

```
> Use a MySQL database to persist data.

> Can we have two tables in the database for claims and
total_claims against user id? Have the total_claims table be
updated via a database trigger.

> Send a single email to both the employee and the finance
team instead of two separate emails.

> Use MS SQL instead of MySQL.
```

Preserving user intent – Sample summary

```
Use MS SQL for data persistence with two tables: claims and
total_claims. The total_claims table should be updated via a
database trigger based on the user id. Send a single email to
both the employee and the finance team.
```

Natural language in code

Natural language usage

- Natural language usage at development time has mostly been outside the code
    - Except for API documentation and comments
- Natural language in code
    - Runtime - natural language can now be a part of the code that gets executed at runtime (i.e., the prompt in a call to an LLM)
    - Compile time?

Runtime

Natural language instructions at runtime

- Many languages (and frameworks), including Ballerina, already support calling LLMs
- Allows expanding the space of problems that can be solved
- Allows leveraging the strengths of the programming language and capabilities enabled via an LLM to achieve tasks that might not cater to the language's strengths
- The user decides what needs to be implemented in code vs what can be implemented using a call to an LLM

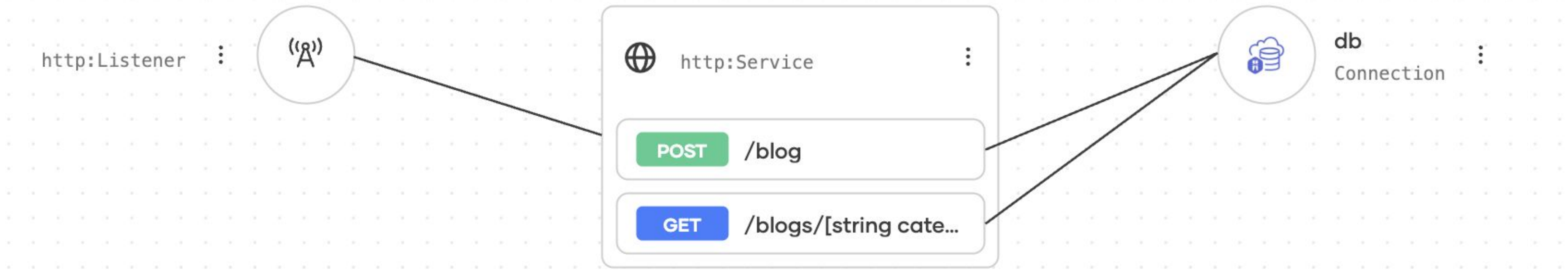
Natural language instructions at runtime

- Calling an LLM in the code, generally includes
  - Initializing a client that can talk to the LLM
  - Specifying an expected format for responses
  - Parsing content from the response to a user-defined type
- Requires quite a bit of boilerplate code that could take the focus away from the core task described in natural language
- Also not straightforward to switch from one model to another

Blog site backend service

- Submit a new post
    - LLM call to identify the most suitable category (from a specified list) and a rating out of 1 - 10 (based on a defined criteria) - requirement specified in natural language
    - If there is a suitable category and the rating is greater than 3, accept the submission and persist the data. If not, reject the post - implemented entirely in the programming language
- Retrieve posts by category, sorted by a rating identified when the blog post is created - implemented entirely in the programming language.

Blog site backend service – Overview

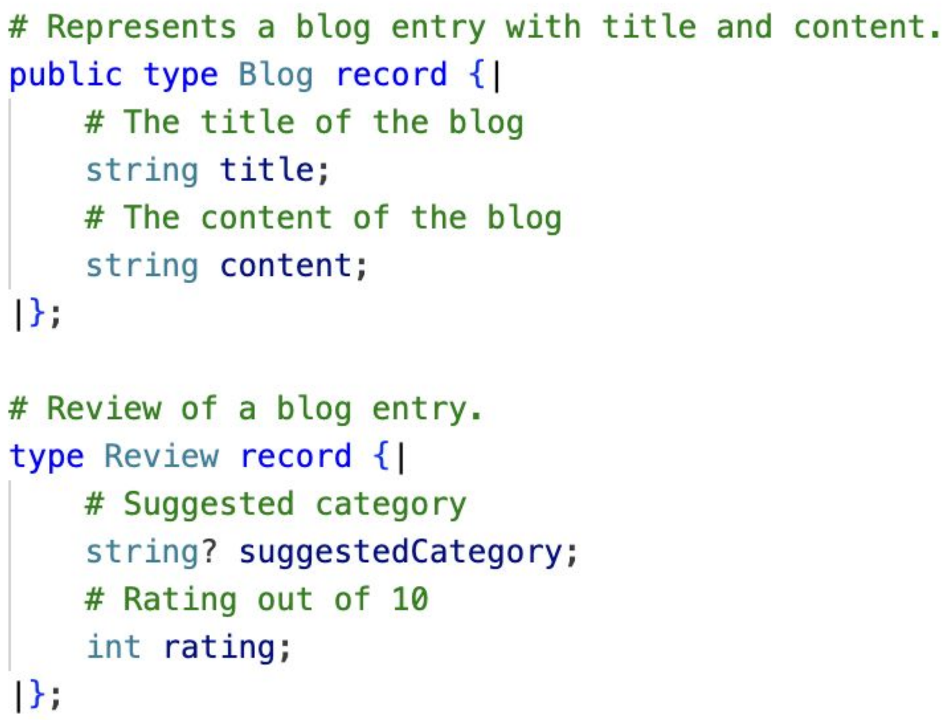


Ballerina JSON to type mapping

```
{
    "id": "A1212456",
    "name": "John Doe",
    "age": 30,
    "email": "johndoe@example.com",
    "address": {
        "street": "1234 Main St",
        "city": "Springfield",
        "state": "IL",
        "zip": "62701"
    }
}
```

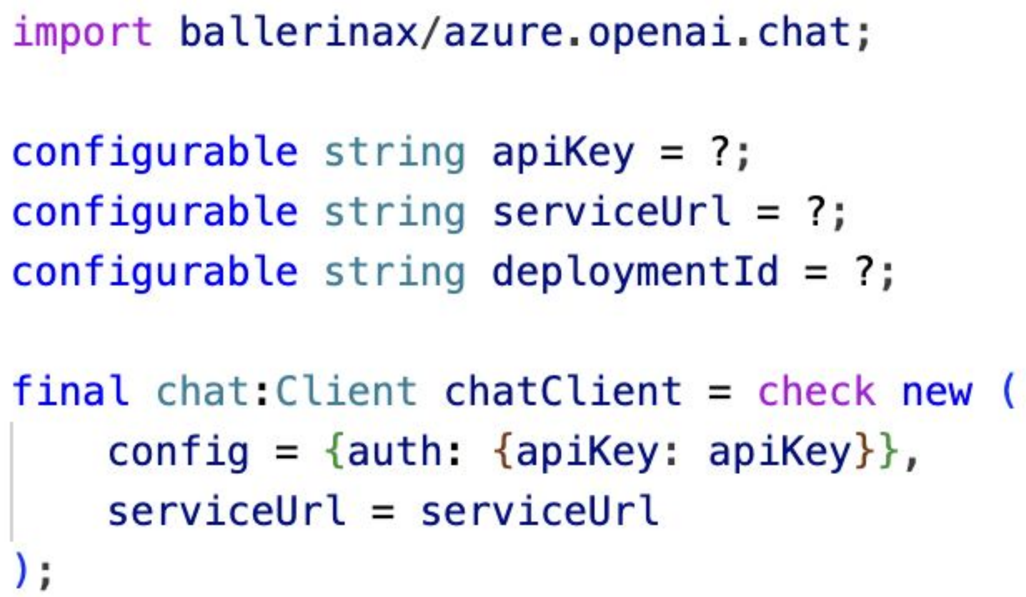
```
type Employee record {|
    string id;
    string name;
    int age;
    string email;
    Address address;
|};

type Address record {|
    string street;
    string city;
    string state;
    string zip;
|};
```

Blog site backend service – Types



```
# Represents a blog entry with title and content.
public type Blog record {|
    # The title of the blog
    string title;
    # The content of the blog
    string content;
|};

# Review of a blog entry.
type Review record {|
    # Suggested category
    string? suggestedCategory;
    # Rating out of 10
    int rating;
|};
```

Blog site backend service – Client initialization



```
import ballerinax/azure.openai.chat;

configurable string apiKey = ?;
configurable string serviceUrl = ?;
configurable string deploymentId = ?;

final chat:Client chatClient = check new (
    config = {auth: {apiKey: apiKey}},
    serviceUrl = serviceUrl
);
```

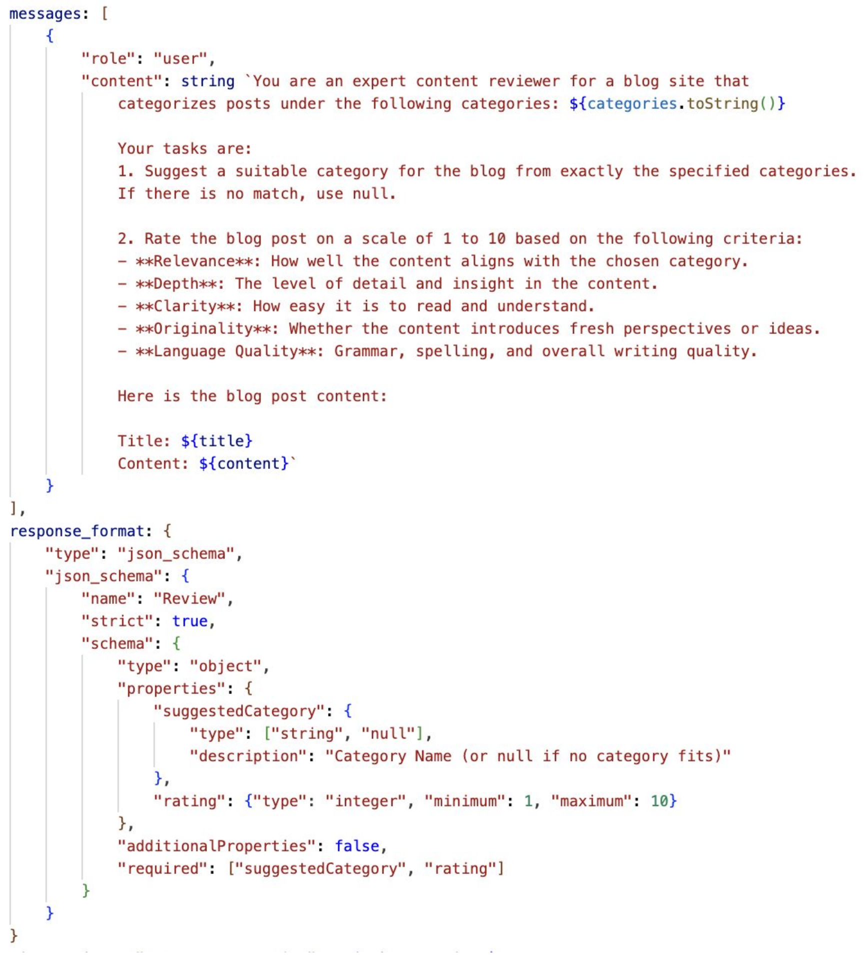
Blog site backend service – using the LLM Client

```
final readonly & string[] categories = [
    "Tech Innovations & Software Development",
    "Programming Languages & Frameworks",
    "DevOps, Cloud Computing & Automation",
    "Career Growth in Tech",
    "Open Source & Community-Driven Development"
];
```

```
resource function post blog(Blog blog) returns http:Created|http:BadRequest|http:InternalServerError {
    do {
        Blog {title, content} = blog;
        string prompt = string `You are an expert content reviewer for a blog site that
            categorizes posts under the following categories: ${categories.toString()}

            Your tasks are:
            1. Suggest a suitable category for the blog from exactly the specified categories.
            If there is no match, use null.

            2. Rate the blog post on a scale of 1 to 10 based on the following criteria:
            - **Relevance**: How well the content aligns with the chosen category.
            - **Depth**: The level of detail and insight in the content.
            - **Clarity**: How easy it is to read and understand.
            - **Originality**: Whether the content introduces fresh perspectives or ideas.
            - **Language Quality**: Grammar, spelling, and overall writing quality.

            Provide your response in the following format:
            {
                "suggestedCategory": "Category Name" (or null if no category fits),
                "rating": 7 (replace with the appropriate rating)
            }

            Here is the blog post content:

            Title: ${title}
            Content: ${content}`;

        chat:CreateChatCompletionRequest chatBody = {
            messages: [{role: "user", "content": prompt}]
        };

        chat:CreateChatCompletionResponse chatResult =
            check chatClient->/deployments/[deploymentId]/chat/completions.post("2025-01-01-preview", chatBody);
        record {
            chat:ChatCompletionResponseMessage message?;
        }[] choices = check chatResult.choices.ensureType();

        string resp = check choices[0].message?.content.ensureType();

        string processedResponse = re `${"```json|```"}`.replaceAll(resp, "");

        Review {suggestedCategory, rating} = check processedResponse.fromJsonStringWithType();
```

Blog site backend service – using Structured Outputs

- A feature that ensures the model adheres to the specified JSON schema when generating output



```
messages: [
    {
        "role": "user",
        "content": string `You are an expert content reviewer for a blog site that
            categorizes posts under the following categories: ${categories.toString()}

            Your tasks are:
            1. Suggest a suitable category for the blog from exactly the specified categories.
            If there is no match, use null.

            2. Rate the blog post on a scale of 1 to 10 based on the following criteria:
            - **Relevance**: How well the content aligns with the chosen category.
            - **Depth**: The level of detail and insight in the content.
            - **Clarity**: How easy it is to read and understand.
            - **Originality**: Whether the content introduces fresh perspectives or ideas.
            - **Language Quality**: Grammar, spelling, and overall writing quality.

            Here is the blog post content:

            Title: ${title}
            Content: ${content}`
    }
],
response_format: {
    "type": "json_schema",
    "json_schema": {
        "name": "Review",
        "strict": true,
        "schema": {
            "type": "object",
            "properties": {
                "suggestedCategory": {
                    "type": ["string", "null"],
                    "description": "Category Name (or null if no category fits)"
                },
                "rating": {"type": "integer", "minimum": 1, "maximum": 10}
            },
            "additionalProperties": false,
            "required": ["suggestedCategory", "rating"]
        }
    }
}
```

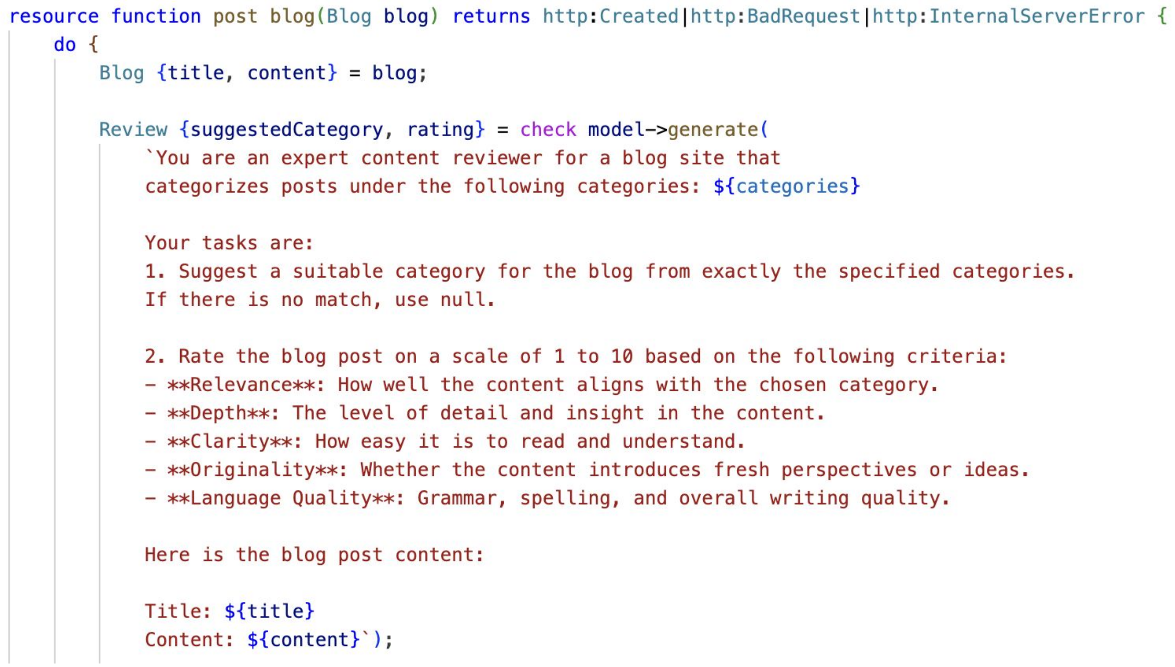
Simplifying LLM calls that produce structured outputs

- A method that
  - Accepts a prompt, which can include interpolations, to be used in a call to an LLM
  - Automatically generates the JSON schema for the expected response based on the expected type in the code (e.g., the `Review` type)
  - Extracts the relevant content from the response and parses it as the type expected by the user, leveraging the dependently-typed feature in Ballerina
  - Named `generate`, is part of the `ai:ModelProvider` LLM abstraction in Ballerina

Blog site backend service – Using the `generate` method

```
import ballerinax/ai.azure;

configurable string apiKey = ?;
configurable string serviceUrl = ?;
configurable string deploymentId = ?;

final azure:OpenAiModelProvider model =
    check new (serviceUrl, apiKey, deploymentId, "2025-01-01-preview");
```

Blog site backend service – Using the `generate` method

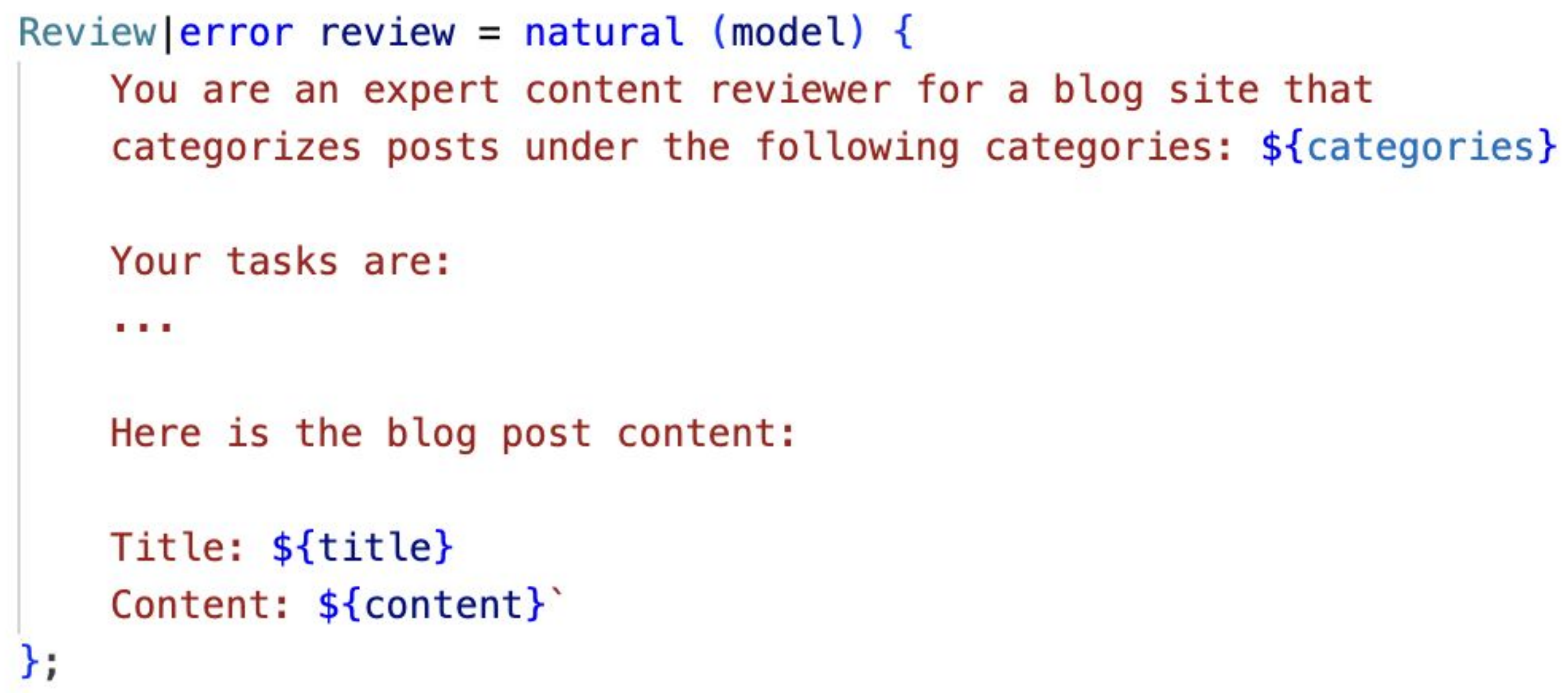


```
resource function post blog(Blog blog) returns http:Created|http:BadRequest|http:InternalServerError {
    do {
        Blog {title, content} = blog;

        Review {suggestedCategory, rating} = check model->generate(
            `You are an expert content reviewer for a blog site that
            categorizes posts under the following categories: ${categories}

            Your tasks are:
            1. Suggest a suitable category for the blog from exactly the specified categories.
            If there is no match, use null.

            2. Rate the blog post on a scale of 1 to 10 based on the following criteria:
            - **Relevance**: How well the content aligns with the chosen category.
            - **Depth**: The level of detail and insight in the content.
            - **Clarity**: How easy it is to read and understand.
            - **Originality**: Whether the content introduces fresh perspectives or ideas.
            - **Language Quality**: Grammar, spelling, and overall writing quality.

            Here is the blog post content:

            Title: ${title}
            Content: ${content}`);
```

Beyond method calls

- Calls to LLMs aren’t like normal method/API calls.
    - Natural language is ambiguous, unlike programming languages
    - Produce non-deterministic output
- Can do with a new abstraction?

Natural expressions



```
Review|error review = natural (model) {
    You are an expert content reviewer for a blog site that
    categorizes posts under the following categories: ${categories}

    Your tasks are:
    ...

    Here is the blog post content:

    Title: ${title}
    Content: ${content}`
};
```

Natural expressions

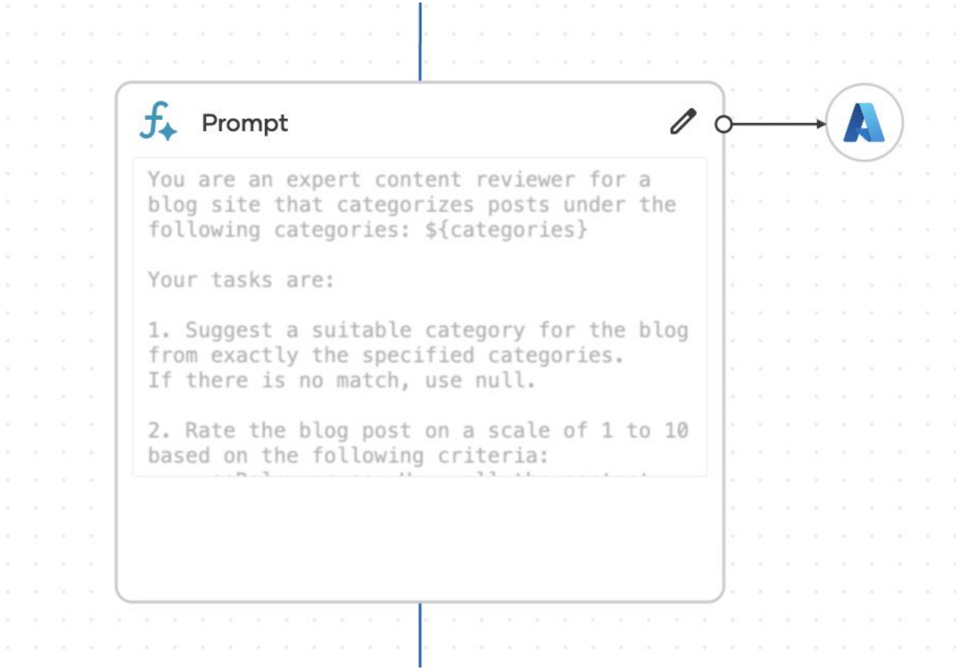
- New syntax to specify instructions in natural language
- Execution follows the same model as the `generate` method
- More natural to include inline within the flow of code
- Makes it clear when logic is natural language/LLM-driven, helping distinguish such logic from traditional code

Natural expressions

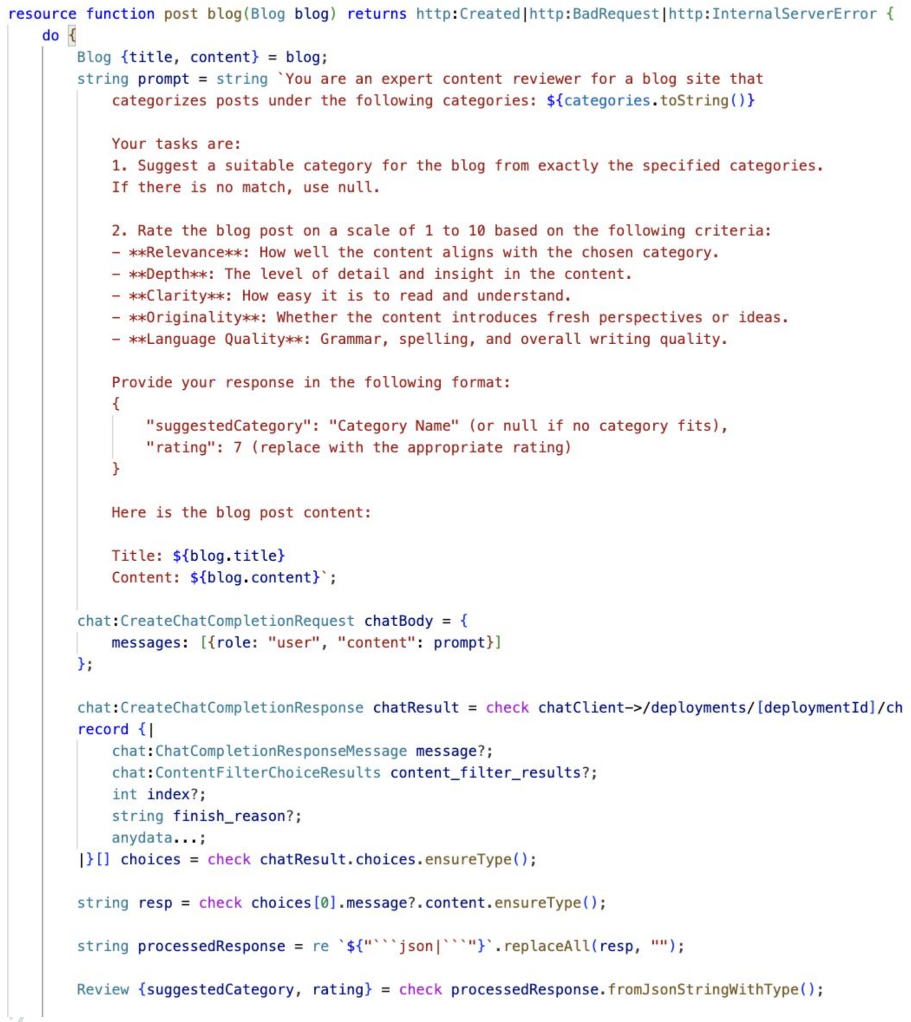
```
resource function post blog(Blog blog) returns http:Created|http:BadRequest|http:InternalServerError {
    do {
        Blog {title, content} = blog;
        Review {suggestedCategory, rating} = check natural (model) {
            You are an expert content reviewer for a blog site that
            categorizes posts under the following categories: ${categories}

            Your tasks are:
            1. Suggest a suitable category for the blog from exactly the specified categories.
            If there is no match, use null.

            2. Rate the blog post on a scale of 1 to 10 based on the following criteria:
            - **Relevance**: How well the content aligns with the chosen category.
            - **Depth**: The level of detail and insight in the content.
            - **Clarity**: How easy it is to read and understand.
            - **Originality**: Whether the content introduces fresh perspectives or ideas.
            - **Language Quality**: Grammar, spelling, and overall writing quality.

            Here is the blog post content:

            Title: ${blog.title}
            Content: ${blog.content}
        };

        if suggestedCategory is () || rating < 4 {
            return <http:BadRequest>{
                body: "Blog rejected due to low rating or no matching category"};
        }

        _ = check db->execute(`INSERT INTO Blog (title, content, rating, category) VALUES (${
                                    title}, ${content}, ${rating}, ${suggestedCategory})`);
        return <http:Created> {body: "Blog accepted"};
    } on fail error e {
        log:printError("Blog submission failed", e);
        return <http:InternalServerError>{body: "Blog submission failed"};
    }
}
```

Natural expressions

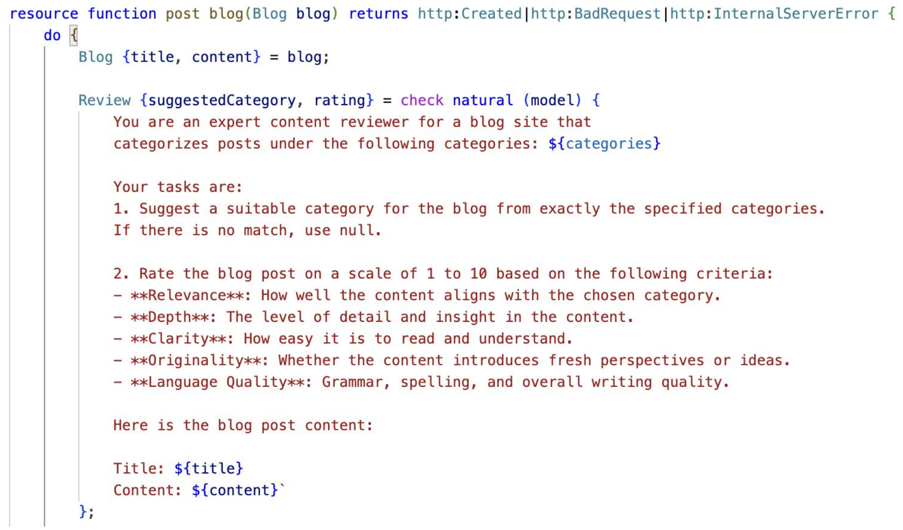
- Integrates with existing model abstractions
- Can have multimodal input
- Can retry for correction on parsing failures



```
You are an expert content reviewer for a
blog site that categorizes posts under the
following categories: ${categories}

Your tasks are:

1. Suggest a suitable category for the blog
from exactly the specified categories.
If there is no match, use null.

2. Rate the blog post on a scale of 1 to 10
based on the following criteria:
```



```
resource function post blog(Blog blog) returns http:Created|http:BadRequest|http:InternalServerError {
    do {
        Blog {title, content} = blog;
        string prompt = string `You are an expert content reviewer for a blog site that
            categorizes posts under the following categories: ${categories.toString()}

            Your tasks are:
            1. Suggest a suitable category for the blog from exactly the specified categories.
            If there is no match, use null.

            2. Rate the blog post on a scale of 1 to 10 based on the following criteria:
            - **Relevance**: How well the content aligns with the chosen category.
            - **Depth**: The level of detail and insight in the content.
            - **Clarity**: How easy it is to read and understand.
            - **Originality**: Whether the content introduces fresh perspectives or ideas.
            - **Language Quality**: Grammar, spelling, and overall writing quality.

            Provide your response in the following format:
            {
                "suggestedCategory": "Category Name" (or null if no category fits),
                "rating": 7 (replace with the appropriate rating)
            }

            Here is the blog post content:

            Title: ${blog.title}
            Content: ${blog.content}`;

        chat:CreateChatCompletionRequest chatBody = {
            messages: [{role: "user", "content": prompt}]
        };

        chat:CreateChatCompletionResponse chatResult = check chatClient->/deployments/[deploymentId]/ch
        record {|
            chat:ChatCompletionResponseMessage message?;
            chat:ContentFilterChoiceResults content_filter_results?;
            int index?;
            string finish_reason?;
            anydata...;
        |}[] choices = check chatResult.choices.ensureType();

        string resp = check choices[0].message?.content.ensureType();

        string processedResponse = re `${"```json|```"}`.replaceAll(resp, "");

        Review {suggestedCategory, rating} = check processedResponse.fromJsonStringWithType();
```



```
resource function post blog(Blog blog) returns http:Created|http:BadRequest|http:InternalServerError {
    do {
        Blog {title, content} = blog;

        Review {suggestedCategory, rating} = check natural (model) {
            You are an expert content reviewer for a blog site that
            categorizes posts under the following categories: ${categories}

            Your tasks are:
            1. Suggest a suitable category for the blog from exactly the specified categories.
            If there is no match, use null.

            2. Rate the blog post on a scale of 1 to 10 based on the following criteria:
            - **Relevance**: How well the content aligns with the chosen category.
            - **Depth**: The level of detail and insight in the content.
            - **Clarity**: How easy it is to read and understand.
            - **Originality**: Whether the content introduces fresh perspectives or ideas.
            - **Language Quality**: Grammar, spelling, and overall writing quality.

            Here is the blog post content:

            Title: ${title}
            Content: ${content}`
        };
```

Compile time

Natural language interleaved with code at compile-time

- Specify the implementation of a function in natural language in the source files
- The call to an LLM happens at compile-time. The compiler generates code based on the requirements
    - Copilot for the compiler?
- Potentially the closest parallels with literate programming in terms of what the source files look like

Two variants

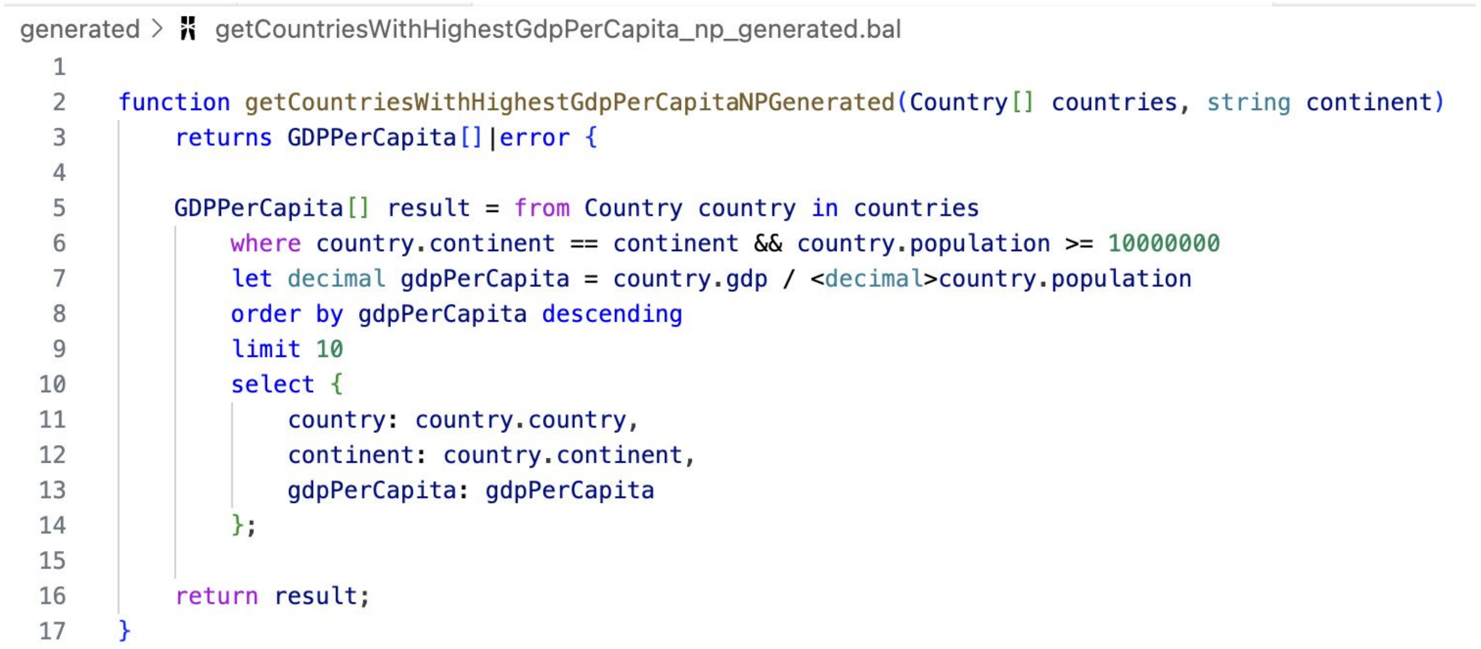
- Generating the logic of a self-contained function
- Data generation (e.g., test data)

```
import ballerina/http;
import ballerina/io;

configurable string populationDataUrl = ?;

type Country record {
    string country;
    string continent;
    int population;
    decimal gdp;
};

type GDPPerCapita record {|
    string country;
    string continent;
    decimal gdpPerCapita;
|};

function getCountriesWithHighestGdpPerCapita(Country[] countries, string continent)
    returns GDPPerCapita[]|error = @natural:code {
        prompt: string `Filter the 10 countries with a population of at least 10,000,000 with
                        the highest GDP per capita in the given continent.`
    } external;

public function main(string continent) returns error? {
    http:Client populationEp = check new (populationDataUrl);
    Country[] countries = check populationEp->/countries;

    GDPPerCapita[] summary = check getCountriesWithHighestGdpPerCapita(countries, continent);
    io:println(summary);
}
```

Natural language interleaved with code

- Potential to increase readability and concisely convey what the implementation does in natural language
- Works best with granular, unambiguous requirements - user chooses when to use
- But, there is no human in the loop!
- The implementation could become a bit of a black box - developers generally tend to prefer being able to review and accept the generated code.
    - Ballerina adds the generated code into the project at compile time - a user can still look at the generated code after building, before running

Generated code



generated > getCountriesWithHighestGdpPerCapita_np_generated.bal

```

function getCountriesWithHighestGdpPerCapitaNPGenerated(Country[] countries, string continent)
    returns GDPPerCapita[]|error {

    GDPPerCapita[] result = from Country country in countries
        where country.continent == continent && country.population >= 10000000
        let decimal gdpPerCapita = country.gdp / <decimal>country.population
        order by gdpPerCapita descending
        limit 10
        select {
            country: country.country,
            continent: country.continent,
            gdpPerCapita: gdpPerCapita
        };

    return result;
}
```

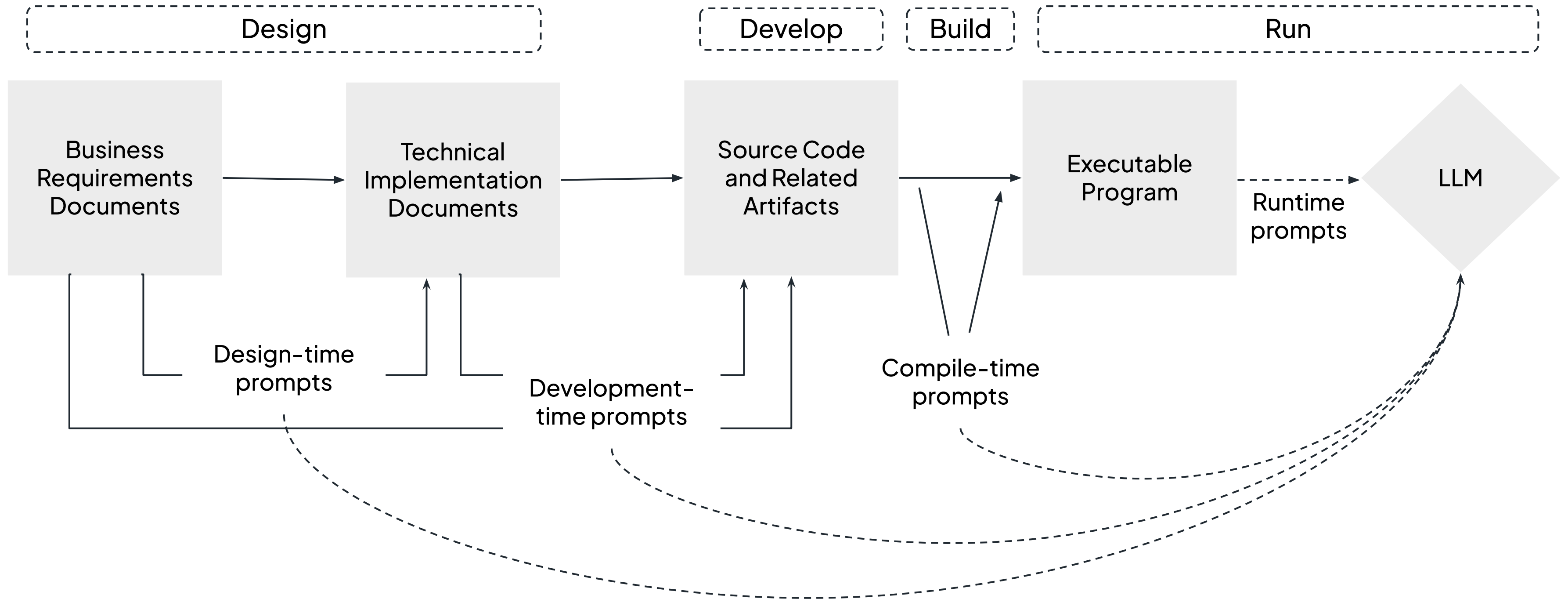
Compile-time data generation

```
import ballerina/test;

const TEST_DATA_GROUP_SIZE = 10;

function discountDataProvider() returns [LoyaltyTier, float, float][] => const natural {
    Generate test data for discount calculation based on the loyalty tier as follows:

    - VIP - 20% of purchase amount
    - REGULAR - 10% of purchase amount
    - NEW - 0% of purchase amount

    Generate ${TEST_DATA_GROUP_SIZE} records for different loyalty tiers
    and purchase amounts, including the expected discount.
};

@test:Config {
    dataProvider: discountDataProvider
}
function testCalculateDiscounts(LoyaltyTier loyaltyTier, float purchaseAmount, float expectedDiscount) {
    float discount = calculateDiscount(loyaltyTier, purchaseAmount);
    test:assertEquals(discount, expectedDiscount);
}
```

Natural language interleaved with code

- Currently experimental
    - The generated code can change with each build - requires further consideration for maintenance, versioning, etc.
    - No human in the loop to review and accept the generated code, which could open room for issues including security concerns. Initial version enforces some restrictions compared to a traditional Copilot to avoid unintentional behaviour.

Reimagining the software development process



Current status

- Early days!
- Development time functionality is available via the Ballerina and WSO2 Integrator:BI VS Code plugins
- New syntax (runtime and compile-time functionality) is currently available with Ballerina Swan Lake Update 13 milestone versions

(Immediate) Future work

- Exploring extending natural expressions to work with all-things natural language in the code
- Scaling natural programming to handle larger-scale applications

Question Time!

Thank you!

WSO2CON 2025