# Hybrid Modernization: Combining OpenRewrite's Precision with LLM Intelligence for Spring

Raquel Pau Fernandez

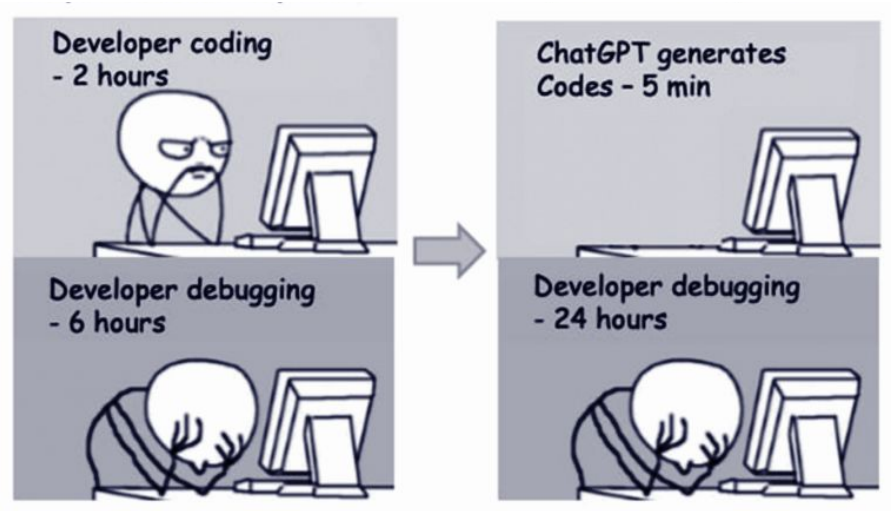
# Raquel Pau Fernandez

Technical Product Manager @ Broadcom

raquelpau@broadcom.com


💚 dev tooling space

*“Please, modernize/upgrade/migrate this application to Spring Boot”*




Before AI

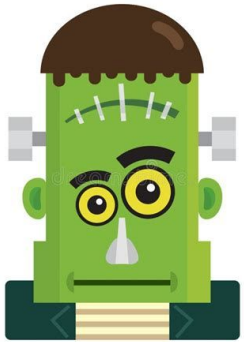
After AI

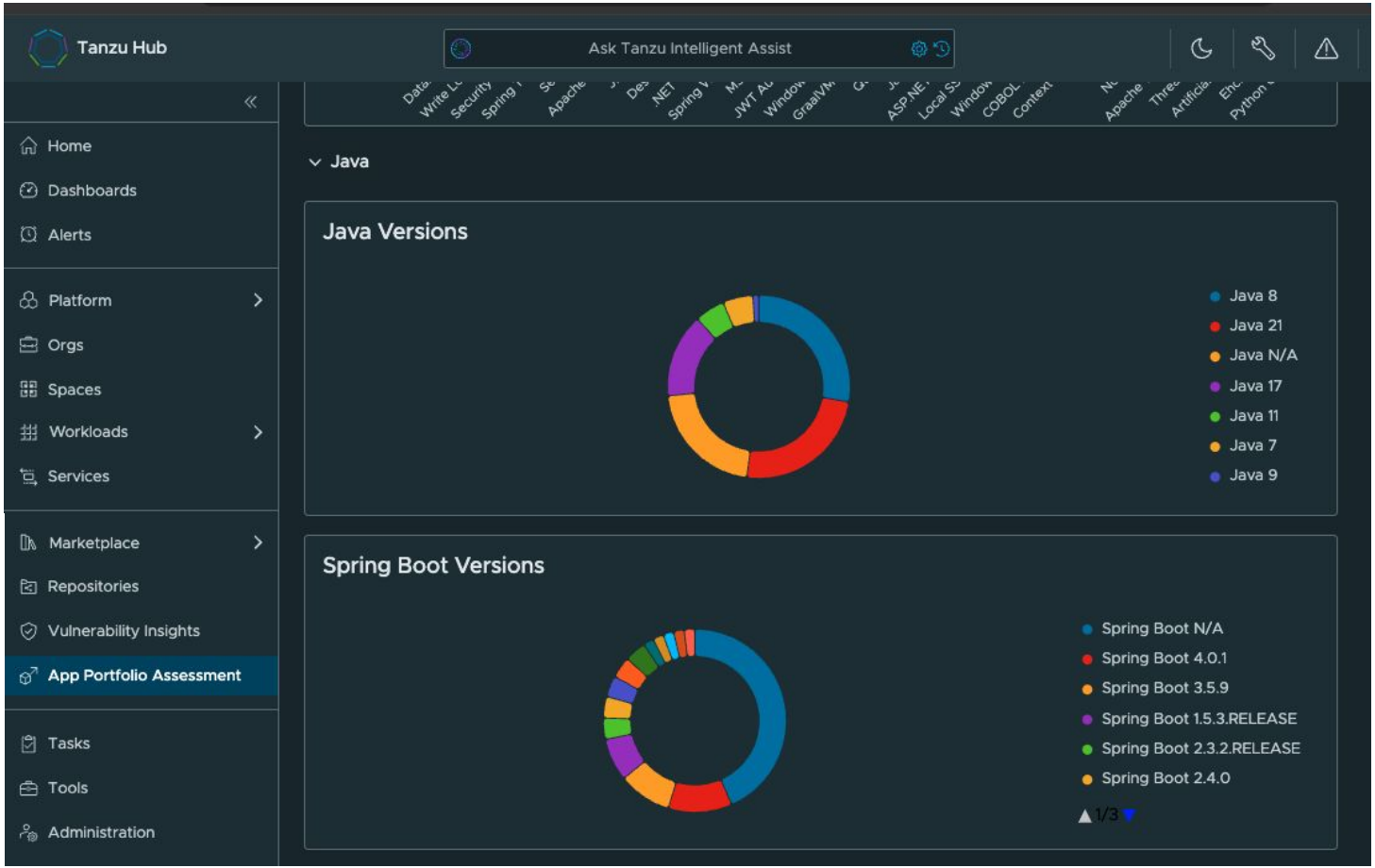
# Benefits of standardization

- **Engineering efficiency**: Common components can be re-used across multiple repositories and teams.
- **Prevents duplication**: No need to implement the same functionality using different technologies or programming languages.
- **Higher collaboration**: Engineering teams can easily contribute and collaborate across different code bases.

# In the ideal world..

- The same Spring Boot version is used across all projects.
- All projects can quickly switch to Spring Boot.
- All projects are always using the latest and greatest version of Spring Boot.



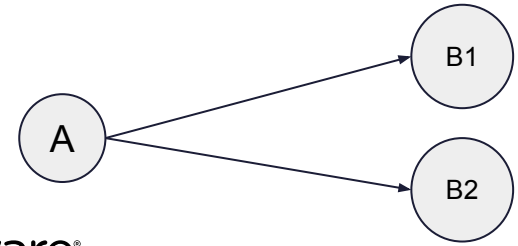


# Modernization Tooling

**Deterministic**: produces always one valid, and the always the same, solution for an input


**Based on the abstract syntax tree**

**Non-deterministic**: produces one of the multiple solutions for translating an input.




**Based on probabilistic models**

# Before AI started to help developers

# OpenRewrite

Open source and static code analysis tool that runs for Maven or Gradle, with:

Type attribution.
Style presservation.

Recipes are programs to perform **deterministic** code changes and can be composed by other recipes

**Unit and Integration tests for recipes are simple (and cheap) to create and maintain.**

```yaml
---
type: specs.openrewrite.org/v1beta/recipe
name: com.vmware.tanzu.spring.recipes.javaee.jaxrs.MigrateJaxRs
displayName: Migrates JAX-RS applications to Spring Boot
description: Main recipe that migrates JAX-RS applications to Spring Boot.
tags:
  - jax-rs
recipeList:
  # Add spring-boot-starter-web dependency to build file.
  - org.openrewrite.java.dependencies.AddDependency:
      groupId: org.springframework.boot
      artifactId: spring-boot-starter-web
      version: 3.3.3

  # Replace JAX-RS jakarta.ws.rs.ext.ExceptionMapper with ControllerAdvice
  - com.vmware.tanzu.spring.recipes.javaee.jaxrs.ReplaceExceptionMapper

  - com.vmware.tanzu.spring.recipes.javaee.jaxrs.MigrateMessageBodyConverter

  - com.vmware.tanzu.spring.recipes.javaee.jaxrs.MigrateFilters

  - com.vmware.tanzu.spring.recipes.javaee.jaxrs.CookieToSpringResponseCookie

  # Important! Replace method parameter annotations after ConvertJaxRsAnnotations actio
  # ConvertJaxRsAnnotations examines Jax-Rs annotations on the parameters to determine

  # Replace JAX-RS @PathParam with Spring Boot @PathVariable annotation.
  - org.openrewrite.java.ChangeType:
      oldFullyQualifiedTypeName: jakarta.ws.rs.PathParam
      newFullyQualifiedTypeName: org.springframework.web.bind.annotation.PathVariable
      ignoreDefinition: false
```

# Limitation: Runtime analysis is not included*

Tools for data flow analysis are constraint to static analysis (explicit references), not runtime analysis, and hence cases where dependency injection is configured via reflection are not covered.

```
Class.forName(“com.acme.NotificationService”)
```

Also, a global data flow graph can be massive and the process can easily run out of heap building that graph.

```
@Service
public class NotificationService {

   private final EmailService emailService;

   public NotificationService(EmailService emailService) {
     this.emailService = emailService;
   }

   public void notifyUser(String userId, String message) {
        emailService.send(userId, message)
   }
}
```

Any dataflow analysis will ignore that `EmailService` is instantiated by Spring because the object is set via reflection mechanisms

*Runtime analysis is only available in only Moderne platform, via the Azul partnership.

# Limitation: Code semantics extraction

Usually, **cross-cutting concerns** such as security, session management, rate-limiting, integration points might be spreaded across different (Java) sources, and many times:

- **Can not be literally rewritten when the new contracts are incompatible**
- The translation logic depends on **the semantics/use case/context**.

| Semantics/Use case | JAX-RS implementation | Spring Boot implementation |
|---|---|---|
| Token validation logic | `ContainerRequestFilter` + `@NameBinding` | `@PreAuthorize("isAuthenticated()")` + `OncePerRequestFilter` |
| Cache-Control header logic | `ContainerResponseFilter` for cache headers | `@Cacheable` (Spring Cache) |
| Counter/throttle logic | `ContainerRequestFilter` for rate limiting | `@RateLimiter` (Resilience4j) |
| Log extraction logic | `ContainerRequestFilter` for audit logging | `@Around` advice (Spring AOP) |

# Full example for the token validation translation from JAXRS to Spring Boot

```
@NameBinding
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE, ElementType.METHOD})
public @interface Authenticated {}
```

```
@Authenticated
@Provider
public class AuthenticationFilter implements
ContainerRequestFilter {

   @Override
   public void filter(ContainerRequestContext req)
{
..
}
```

```
@Path("/vets")
public class VetResource {
   @POST
   @Authenticated
   @RolesAllowed("VET_ADMIN")
   public Response addVet(Vet vet) { ... }
}
```

```
public class AuthenticationFilter extends OncePerRequestFilter
{
   @Override
   protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain chain) throws
IOException, ServletException {

…}
```

```
@RestController
@RequestMapping("/vets")
public class VetController {
   @PostMapping
   @PreAuthorize("isAuthenticated() and hasRole('VET_ADMIN')")
   public ResponseEntity<Vet> addVet(@RequestBody Vet vet) {
... }
}
```

# Limitation: Cross-language translations

OpenRewrite has been designed to support multiple programming languages:

- Java and JVM languages (Groovy, Kotlin)
- Python
- COBOL
- Javascript
- etc..

**But, cross-language translations are NOT supported**.

For example: Migrate from COBOL to Spring Batch

# Summary about OpenRewrite

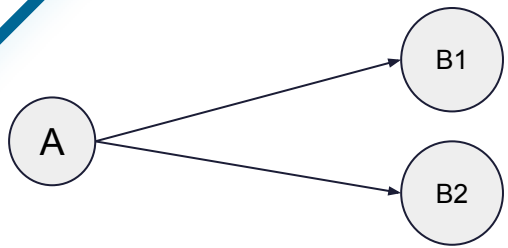
It is suitable for:

- Deterministic and type-safe code changes (with style preservation) for small and large codebases; preventing:
  - Token consumption
  - Hallucinations when the code base does not fit in the context window
- Easy to test

It is not suitable for:

- Runtime analysis.
- Code semantics.
- Cross language translations

# After AI started to help developers



## Early Foundations (2017–2020)

From Transformers to an emerging code generation capability in OpenAI's GPT-3

## The First Wave: Dedicated Code Models (2021–2022)

GitHub copilot made the "autocomplete-style" coding assistance paradigm mainstream.

## The ChatGPT Moment & General-Purpose Models (2022–2023)

ChatGPT improves explaining code, debugging, and generating snippets via chat became mainstream.

## **The Agentic Coding Era (2024–2026)**

**Agentic** coding tools emerge going far beyond autocomplete to full multi-file edits.

# What is a coding agent

A coding agent is an **autonomous AI system** that:

**`Given`** an project,

**`When`** the user starts a `conversation` and provides a message,

**`Then`** it provides an `answer` *(with code suggestions)* created by the combination of:

- A Large Language Model (**LLM**) as its core reasoning engine,
- A set of executable **`tools`** (and **`skills`**), and
- An **orchestration loop**, enabling it to complete software engineering tasks by perceiving, planning, acting, and self-correcting iteratively

# MCP Servers expose:

**Tools**: **Business logic functionality** that can be executed locally or on a remote backend. They expose the result to the LLM.

```
List of speakers whose talk is about Spring AI
```

**Prompts**: They are textual content that describe a set of steps/scripts that need to be interpreted and followed by the LLM.

```
/accept-to-speak:
 Step 1. Accept the code of conduct for Spring IO
 Step 2. Review and complete your bio
```

**Resources**: To reference files or attachments designed to be shared across agents.

```
@code-of-conduct
```

# Coding agents (partially) solve limitations of OpenRewrite

**Translate** to different programming/structural languages

**Summarize** code semantics.

Approximate **runtime analysis** via pattern-based reasoning and verification loops (which might run tests)

# Limitation: agents don’t fully understand the code, by default*

**Type inference** — Types are estimated, not resolved from a semantic model

**Unknown symbols** — If a definition is missing, behavior is usually assumed, not flagged

**Context window** — Only part of the codebase is visible at any given time

**Some agents have started to support the LSP (Language Server Protocol), but is not enabled by default*

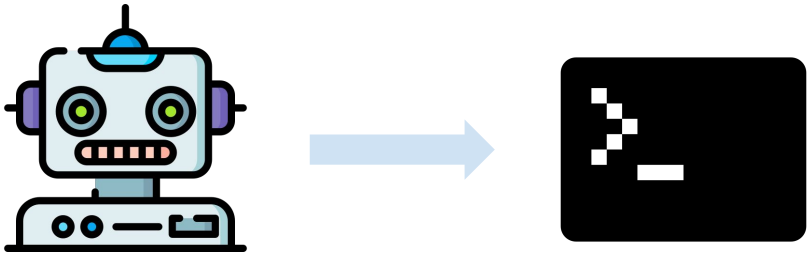
# Limitation: agents are hard to scale and validate

**Scale & consistency** — Non-deterministic; large rewrites require many iterations

**Testing** — Probabilistic evaluation with token cost; no deterministic unit tests

# Can we use the best of both worlds? Or should we choose?

# Instructing agents when and how to run CLIs (that use OpenRewrite)



# Options to run CLIs by coding agents

**Tools in *local* MCP servers** that execute (native) CLI tools that need the local environment.



The context window is populated with the selected tool/command

**Skills**, which include steps that verify and/or install a specific CLI and execute some commands.

The text is injected in the context window **only when relevant**.

**Prompts in *remote* MCP servers**: That include steps that verify and/or install a specific CLI and execute some commands.

The text is **fully** and **always** included in the context window.

# There are three modernization Problems

**Upgrades**: Replace old APIs by new APIs of the same library/project.

From Spring Boot 3.4.x to 4.x

**Migrations**: Replace libraries with equivalent or extended functionality.

From Spring Retry to Spring Framework

**From Jakarta JAX-RS to Spring Boot.**

**Full Rewrites**: Rewrite the business logic using different abstraction model

**From COBOL to Spring Batch**

From Struts to Spring Boot

# Let’s upgrade to Spring Boot 4.0.x

# Upgrade plan from Boot 3.4.x to 4.0.x

```
- Step 1:
    * Upgrade spring-boot from 3.4.x to 3.5.x
    * Upgrade spring-data-commons from 3.4.x to 3.5.x
    * Upgrade spring-data-jpa from 3.4.x to 3.5.x
    * Upgrade micrometer from 1.14.x to 1.15.x
    * Upgrade acme-spring-starter from 1.0.x to 1.5.x
- Step 2:
    * Upgrade spring-boot from 3.5.x to 4.0.x
    * Upgrade spring-framework from 6.2.x to 7.0.x
    * Upgrade spring-retry from 2.0.x to spring-framework 7.0.x
    * Upgrade jackson from 2.21.x to jackson 3.1.x (optional)
    * Upgrade hibernate-orm from 6.6.x to 7.1.x
    * Upgrade spring-data-commons from 3.5.x to 4.0.x
    * Upgrade spring-data-jpa from 3.5.x to 4.0.x
    * Upgrade micrometer from 1.15.x to 1.16.x
    * Upgrade acme-spring-starter from 1.5.x to 2.0.x
```

Every single entry is a different set of independent OpenRewrite recipes

# Best Practices for (Spring) Upgrade Recipes

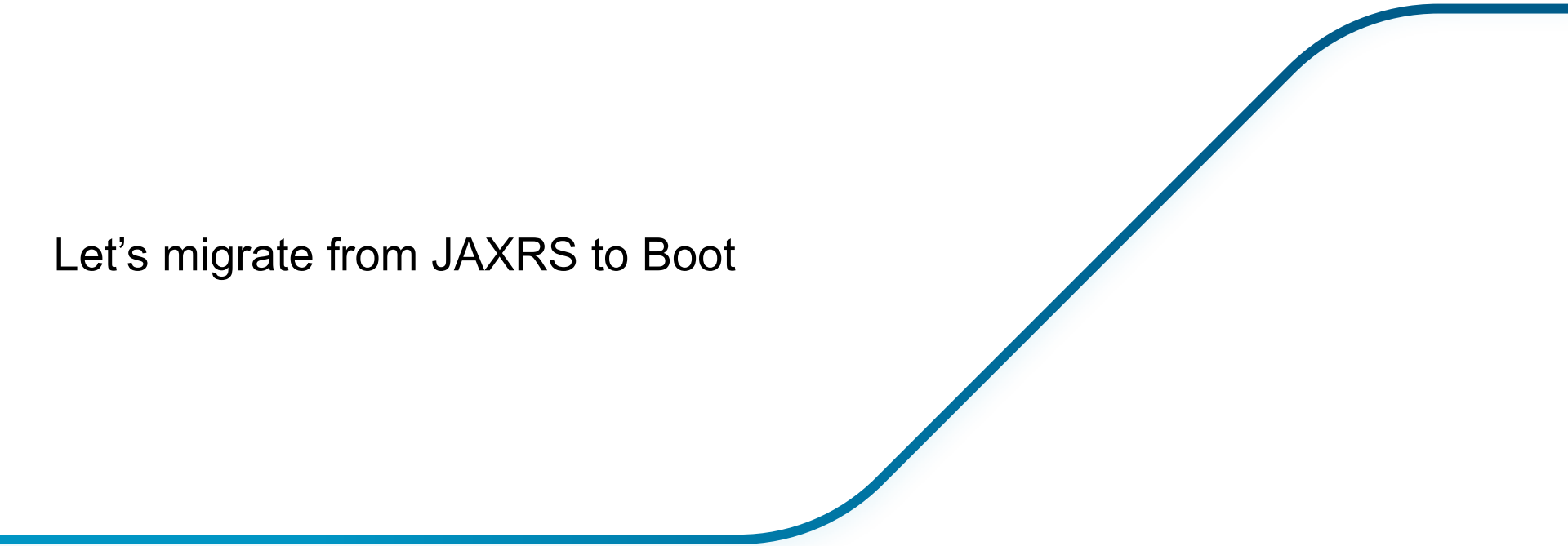
Use the same version of OpenRewrite across all your recipes.

Use or build an existing engine that creates a deterministic “upgrade plan” of what recipes need to be joined together based on the Maven metadata of your app dependencies.

- There might be different vendors and versions for a recipe.
- Each project follows a different release train and frequency.
- There is no warrantee that a recipe is available after a new release is published.

Tanzu Platform offers the upgrade plan with: `cf repo upgrade-plan`

# Let’s migrate from JAXRS to Boot



# Migrations requirements

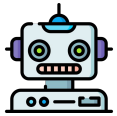
**Dependencies must be aligned (and upgraded):** The earliest version to migrate Jakarta JAXRS is Spring Boot 3.0.x, but the application might be using Spring Boot 4.0.x already.

```
* Upgrade jakarta-jersey from 3.1.x to spring-framework 7.0.x
```

**Some API replacements can be easily configured with OpenRewrite**, but in general, APIs have not a perfect 1:1 mapping.

```
ContainerRequestFilter → OncePerRequestFilter
```

**Several changes require AI capabilities (skills)** : Detect if a filter logic is designed for token validation, Cache-Control, Counter/throttle, Log extraction, etc.. to add the right Spring annotations



`/migrate-from-jaxrs`

## 1. Runs a CLI command

```
cf repo apply-advice -n jakarta-jax-rs
```

```
* jersey:3.1.x migrates into spring-framework:6.2.x
* jakartaee-rest:3.1.x migrates into spring-framework:6.2.x
* junit from 5.10.x to 6.0.x
* junit-platform from 1.10.x to 6.0.x
* jackson from 2.17.x to 2.21.x
* jackson-annotations from 2.17.x to 2.21
```

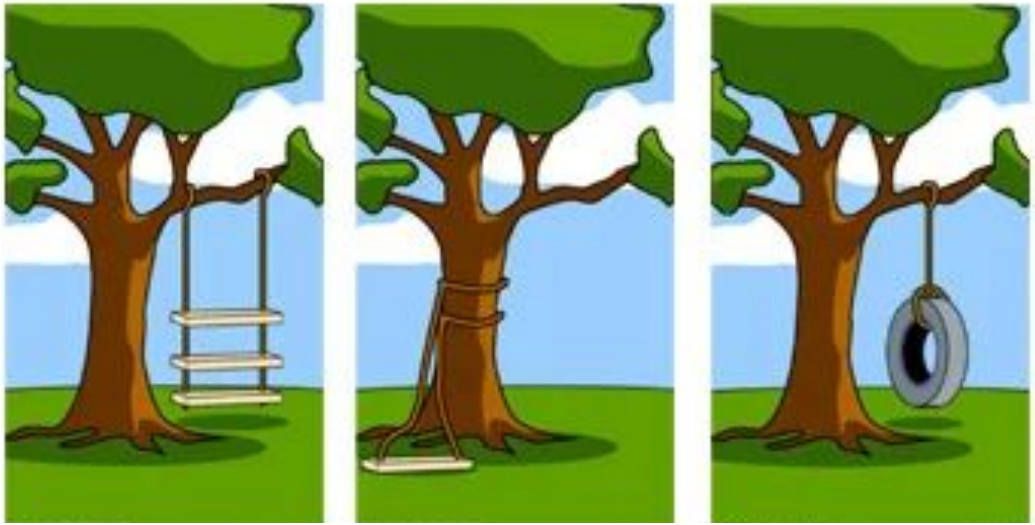
## 2. Follows follow up skills

`/compile-issues-followup`

`/semantic-followup`

# Let’s rewrite COBOL (Batch) to Spring Batch

# How can help AI understanding the requirements?



Original App | Rewritten App by AI | Expected App

# Rewrites have a different challenges

**Preserve the existing inbound and outbound API contracts**

- Use the same external services, database schemas, etc..

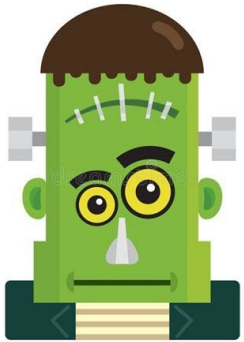
**Preserve semantics/business logic across programing languages**

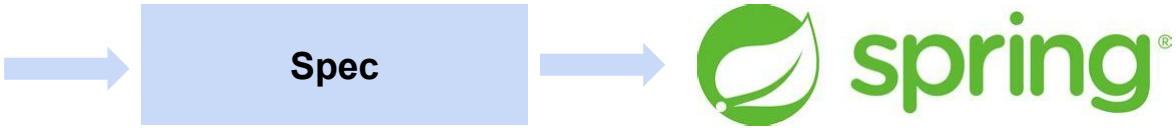
- Test cases need to be (added?) and reviewed.
- Early feedback is needed to prevent throw away code / tokens

**Requires a big cognitive load (hence a big context window!)**

The functionality must be **incrementally** rewritten, reviewed and merged.

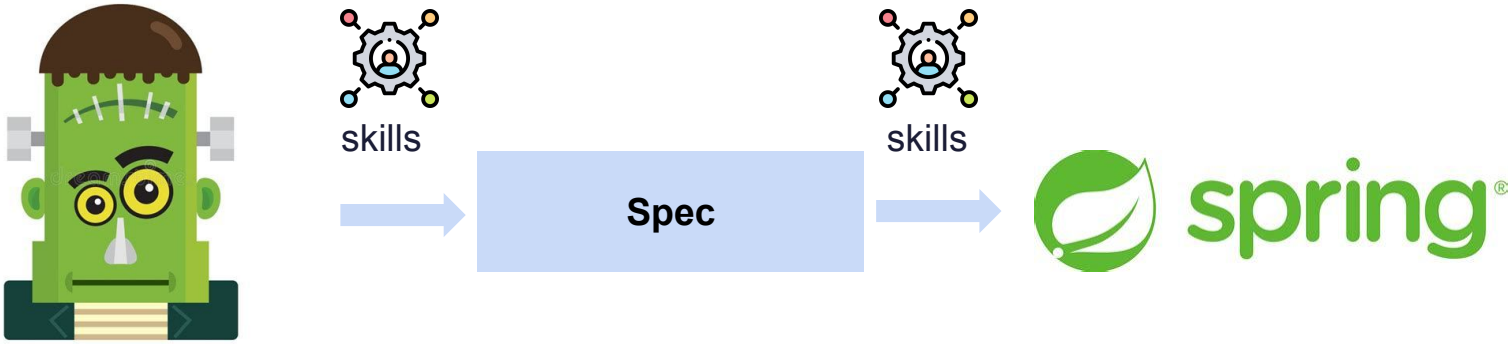
# Specs must be the single source of truth of **what to preserve**






**Specs**, as an intermediate **controlled** language, provide a description of existing functionality, saving tokens and time by preventing incorrect business logic implementation

# Using a controlled language (DSL) for specs matters




- Reduces ambiguity introduced by natural language.
- Produces (More) predictable Spring code.
- Helps the traceability and validation (From where these requirements are collected?)

# The rising of Spec Driven Development

> GitHub: "In this new world, *maintaining software means evolving specifications.* [...] The lingua franca of development moves to a higher level, and code is the last-mile approach."
>
> Tessl: "A development approach where *specs — not code — are the primary artifact.* Specs describe intent in structured, testable language, and agents generate code to match them."

https://martinfowler.com/articles/exploring-gen-ai/sdd-3-tools.html

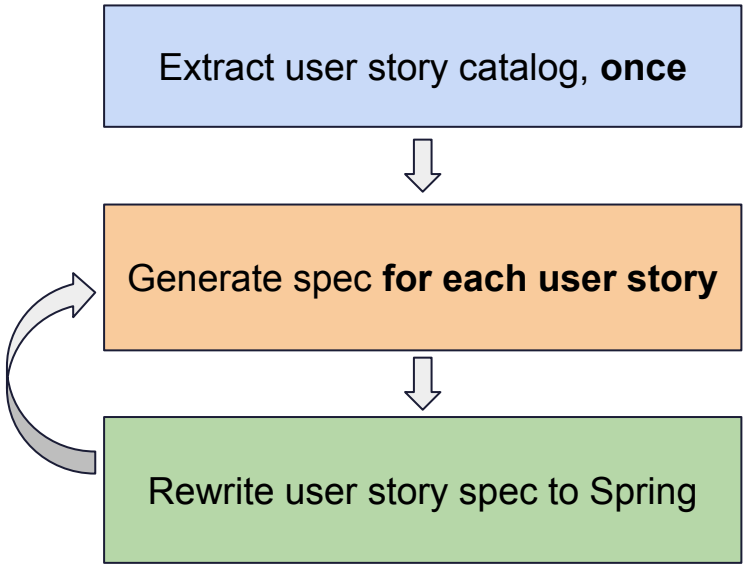
# The rising of Spec Driven Development

GitHub SpecKit – Designed **for greenfield development**. Provides OSS skills to generate a (very lightweight) structured spec and an opinionated Git workflow. It has demonstrated that this produces more consistent, correct implementations.

(Amazon) Kiro - Designed for **greenfield development**, too. Spec Driven development using a commercial IDE or CLI.

**The challenge on brownfield apps is the context window**

# Modernizing brownfield apps requires three phases




/extract-user-story-catalog

`spec/domains/bookings/user-story-catalog.md`

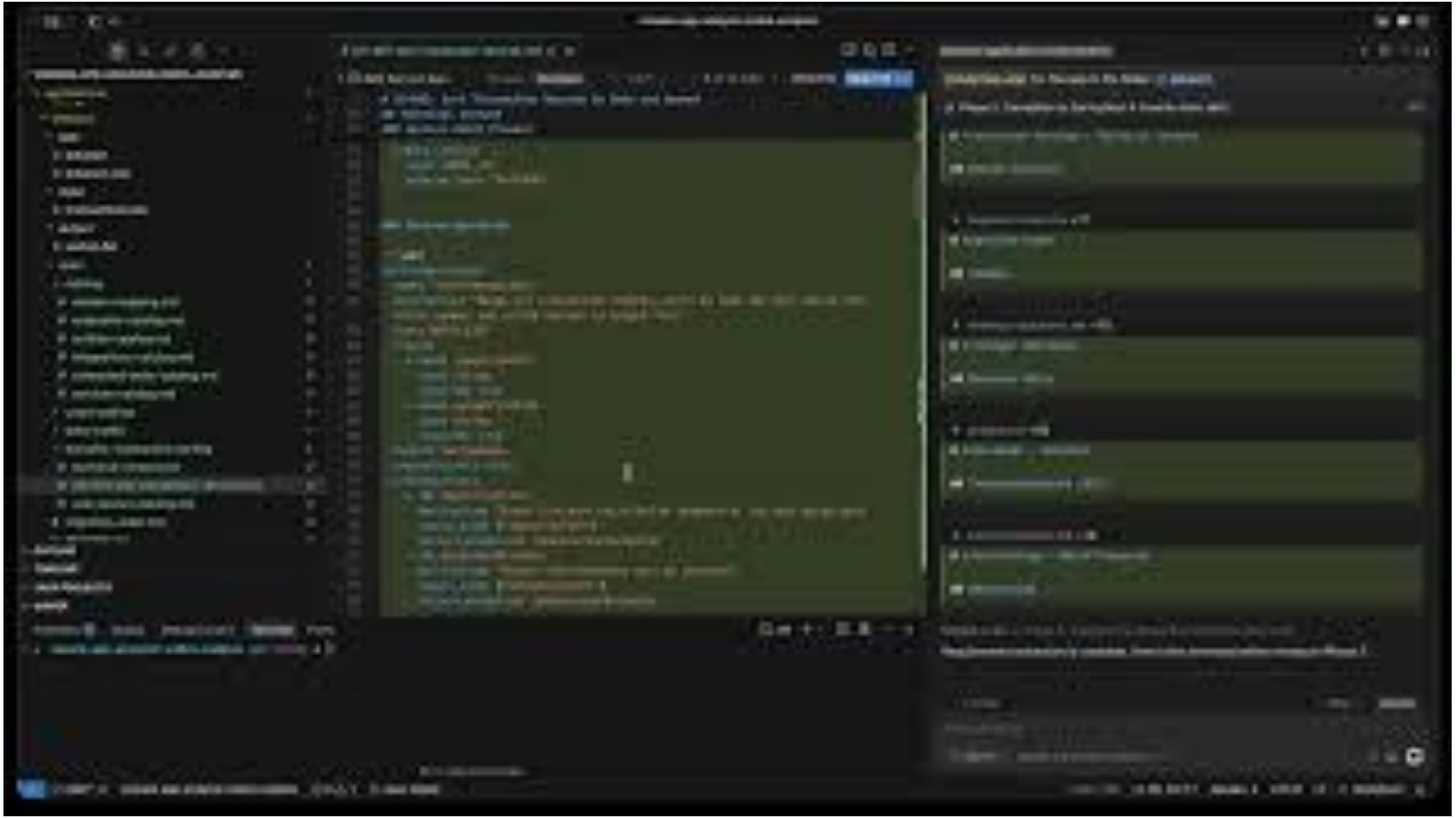
`spec/domains/accounts/user-story-catalog.md`

/extract-user-story-detail

`spec/domains/bookings/US-BKG-001.md`

/rewrite-story

Each phase is designed to use a new session/context window



# Summary

- **Upgrades and Migrations can usually follow an hybrid modernization model,** where OpenRewrite recipes minimize the token consumption and AI improves the final result when complex data flow analysis or semantics are needed.
- **(Full) Rewrites require** a guided workflow for humans where the target **high level semantics are formally modeled as first citizens** to create more predictable (and testable) results.

# Thank You

vmware by Broadcom