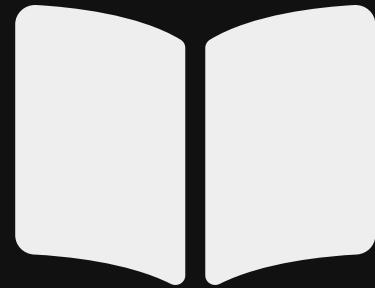


Jan Galinski @ Holisticon AG

Code Generation with Kotlin Poet

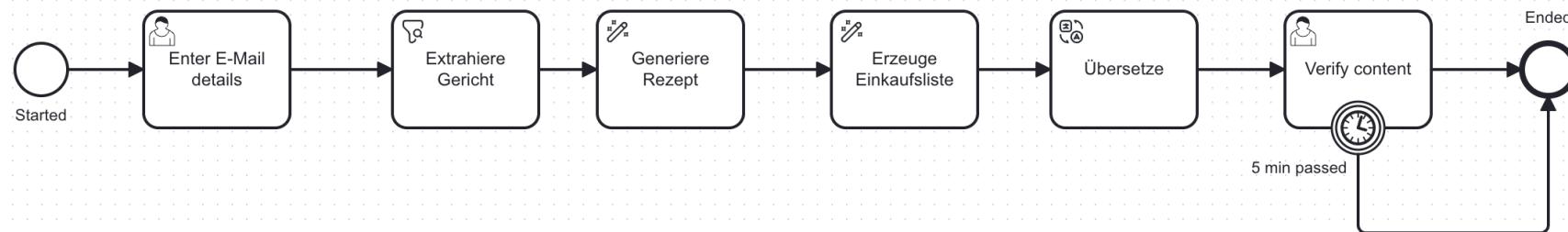
Backend Applications & Services





**Let's start
with a story**

Girls & Boys Day 2023



ChatGPT powered "what to cook for dinner?"
process

<https://github.com/holunda-io/camunda-8-connector-gpt>



How can we ...

- ... reduce manual coding?
- ... ensure high quality?

Abstract

AI code generation is a challenge for our daily work. But do we want to rely on some chat comments for professional software development? There must be a middle way between manual coding and c&p from the internet.

Luckily there is: meta-programming. In short: write software that writes software. We will have a look at practical implementations using the Kotlin Poet language model lib and maven plugin infrastructure.

Let there be code.



About me

```
***** COMMODORE 64 BASIC V2 *****  
64K RAM SYSTEM 38911 BASIC BYTES FREE  
READY.  
10 PRINT "SIMON IST DOOF"  
20 GOTO 10■
```

- First program in 1986 - BASIC/C64
- ... before I even owned the machine

- Senior Consultant
@HolisticonAG
- 20+ years of industry
experience
- Kotlin - Fan
 - ... Backend/Spring Boot - no Android
 - ... prefers maven over gradle
 - ... uses light mode



about.me/jangalinski



Agenda

- What is Meta Programming?
- How does it work? - Live Coding
- Where to go from here?

An aerial photograph of Lake Taal, Philippines. In the foreground, the dark, crater-lake water of Lake Taal is visible. Behind it, the dark, rugged slopes of Mount Taal rise sharply. In the far distance, another volcano, Mount Banahaw, is visible across the sky.

What is Meta Programming?

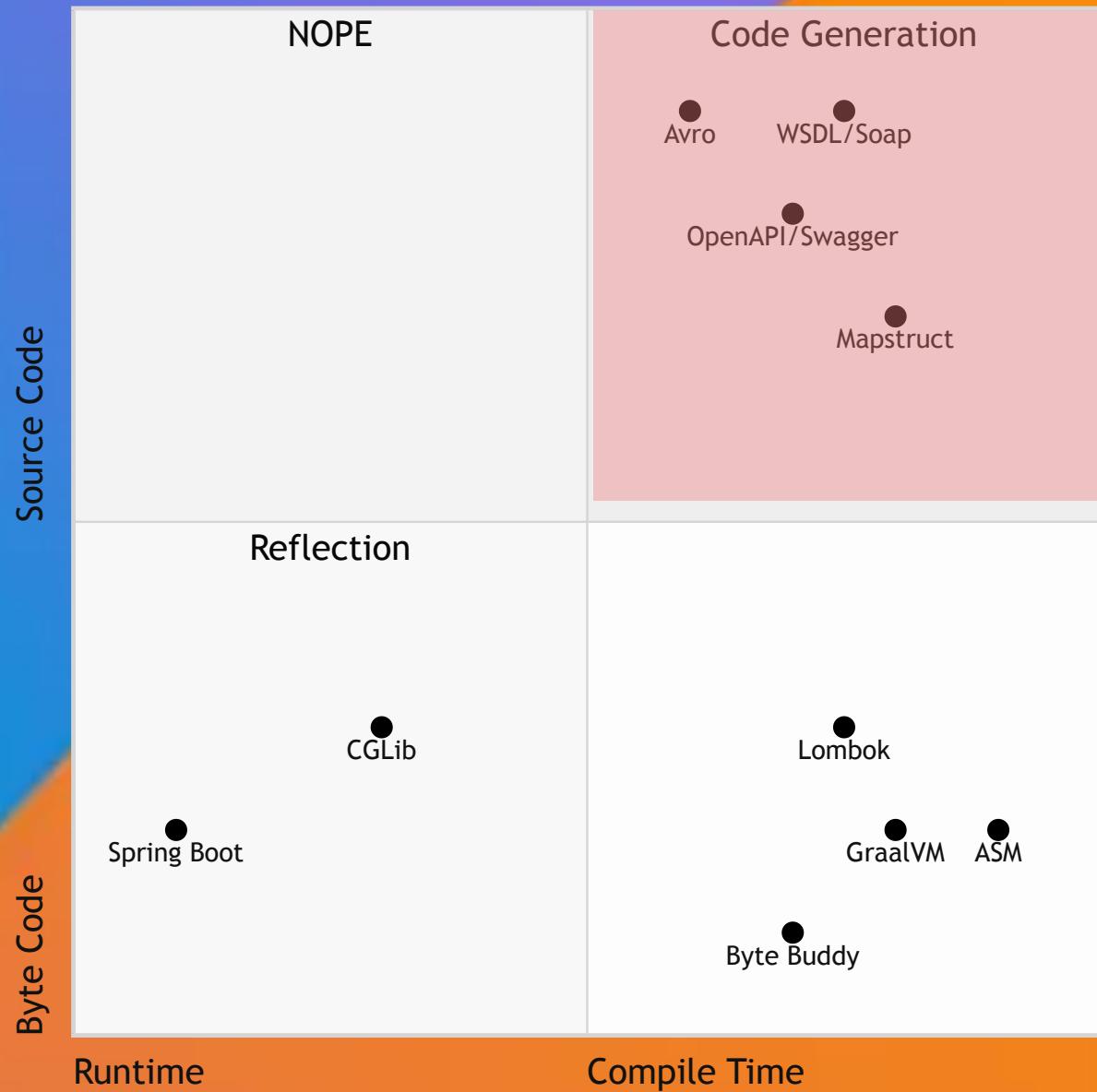
image: <https://www.filipinotravel.com.ph/lake-taal/>

W

Metaprogramming is a *programming technique* in which computer programs have the ability to *treat other programs as their data*.

Metaprogramming can be used to *move computations from run-time to compile-time*, to *generate code* using compile time computations, and to *enable self-modifying code*.

Types of Meta Programming



Types of Source Code Generators

- Annotation Processing
 - *mapstruct*
- Command Line Console
 - *open-api*
- Build Plugins (maven, gradle)
 - *open-api*
 - *avro*

Tools for Source Code Generation

- Well ... create an ASCII file
- Popular: Template Engines
 - *mustache*
 - *velocity*
- Language Model API
 - ~~sun code model~~
 - *spoon*
 - or: **Kotlin Poet**

I don't like templates

```
{#{useBeanValidation} }
  {>beanValidation} }
  {>beanValidationModel} }
{/useBeanValidation} }
{#{swagger2AnnotationLibrary} }
@Schema({{example}}example = "{{#lambdaRemoveLineBreak}}
{{#lambdaEscapeDoubleQuote}}{{{.}}}{{/lambdaEscapeDoubleQuote}}
{{/example}}required = true,
{{#isReadOnly}}readOnly = {{{isReadOnly}}},{{/isReadOnly}}descr
{#{swagger1AnnotationLibrary} }
@ApiModelProperty({{example}}example = "{{#lambdaRemoveLineBre
@get:JsonProperty("{{{baseName}}}", required = true){#isInherite
```

Kotlin Poet



KotlinPoet

Overview

KotlinPoet

Interop - JavaPoet

Interop - kotlinx-metadata

Interop - KSP

PI

Stack Overflow ▲

Change Log

Contributing

Here's a `HelloWorld` file:

```
class Greeter(val name: String) {  
    fun greet() {  
        println("""Hello, $name""")  
    }  
  
    fun main(vararg args: String) {  
        Greeter(args[0]).greet()  
    }  
}
```

And this is

```
val greeter =  
    val file =  
        .addTypeSpec(  
            TypeSpec
```

Kotlin Poet

<https://square.github.io/kotlinpoet/>

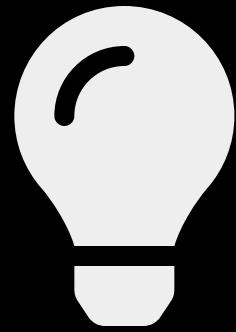
```
    .addFunction(  
        PropertySpec.builder("name", String::class)  
            .initializer("name")  
            .build()  
    )  
    .addFunction(  
        FunSpec.builder("greet")  
            .addStatement("println(%P)", "Hello, \$name")  
            .build()  
    )  
    .build()  
)  
.addFunction(  
    FunSpec.builder("main")  
        .addParameter("args", String::class, VARARG)  
        .addStatement("%T(args[0]).greet()", greeterClass)  
        .build()  
)
```

Table of contents

[Example](#)[Code & Control Flow](#)[%S for Strings](#)[%P for String Templates](#)[%T for Types](#)[Nullable Types](#)[%M for Members](#)[MemberName and operators](#)[%N for Names](#)[%L for Literals](#)[Code block format strings](#)[Relative Arguments](#)[Positional Arguments](#)[Named Arguments](#)[unctions](#)[Extension functions](#)[Single-expression functions](#)[Default function arguments](#)[Spaces wrap by default!](#)[Constructors](#)[Parameters](#)[Properties](#)[Inline properties](#)[Interfaces](#)[Objects](#)[Enums](#)[Anonymous Inner Classes](#)[Annotations](#)

A dense silhouette of many people holding up protest signs against a light blue background.

Demo Time



Conclusion

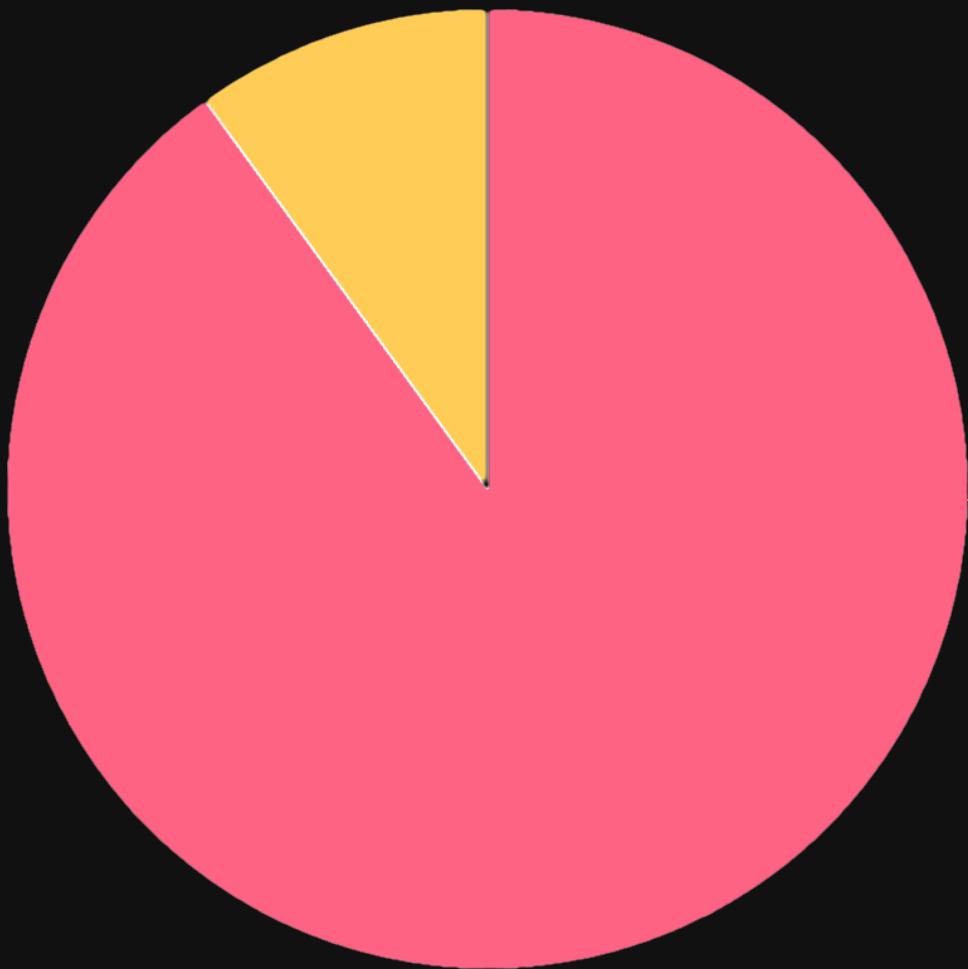
Considerations

- Creating and maintaining a code generator is not free of cost
- Carefully choose and analyse your use case/scenario
- Your code base can benefit from
 - Less Boilerplate code
 - unified way of approaching problems
 - reduce mental overload

Best practices

- Generator implementation should be independent of generator type
- API of generated code is more important than readability
- Avoid increase of build time by cashing/hashing
- *Use dedicated project sub-modules for code generation*

Should you use Kotlin Poet for code generation?



YES!



Also YES!
(but in yellow)





Some useful Links

presentation &
demo

[github.com/jangalinski/talk-
kotlin-poet](https://github.com/jangalinski/talk-kotlin-poet)

kotlin poet
@square

square.github.io/kotlinpoet

obsidian.md -
advanced slides

[mszturc.github.io/obsidian-
advanced-slides](https://mszturc.github.io/obsidian-
advanced-slides)

c64 emulator/js

c64emulator.111mb.de