# Modern Groovy Domain Specific Languages

Who Am I: **David Clark**

Where Do I Work: **Principal Engineer, Research Now**

Where Do I Code: https://github.com/dwclark

Where Is This Presentation:
https://github.com/dwclark/modern-dsls

Why Is This Presentation So Boring And On A White Background?:

*Because “creative” presentations inevitably involve dark backgrounds and dark text. Within 30 seconds of seeing a dark background someone in the audience will ask to change to a white background. Let's all just cut to the chase and go with boring black on white, besides it's the content you care about anyway, right?*

# What a DSL is ***NOT***

- Usually it isn't **domain specific**, unless you define “domain” very generically.
- It definitely isn't a **new language**. A new language is going to involve parsers, lexers, interpreters/compilers, debuggers, runtime libraries and a lot more.
- It's not just a matter of leaving off semi-colons and parenthesis or other syntactic tricks, though this can be a part of a good DSL.
- It's **not the first thing you do in a project**, you don't write a DSL at the beginning of a project. You write it at the earliest in the middle of a project.
- It's almost certainly **not going to be used by non-programmers**, definitely not going to be used by non-technical people.

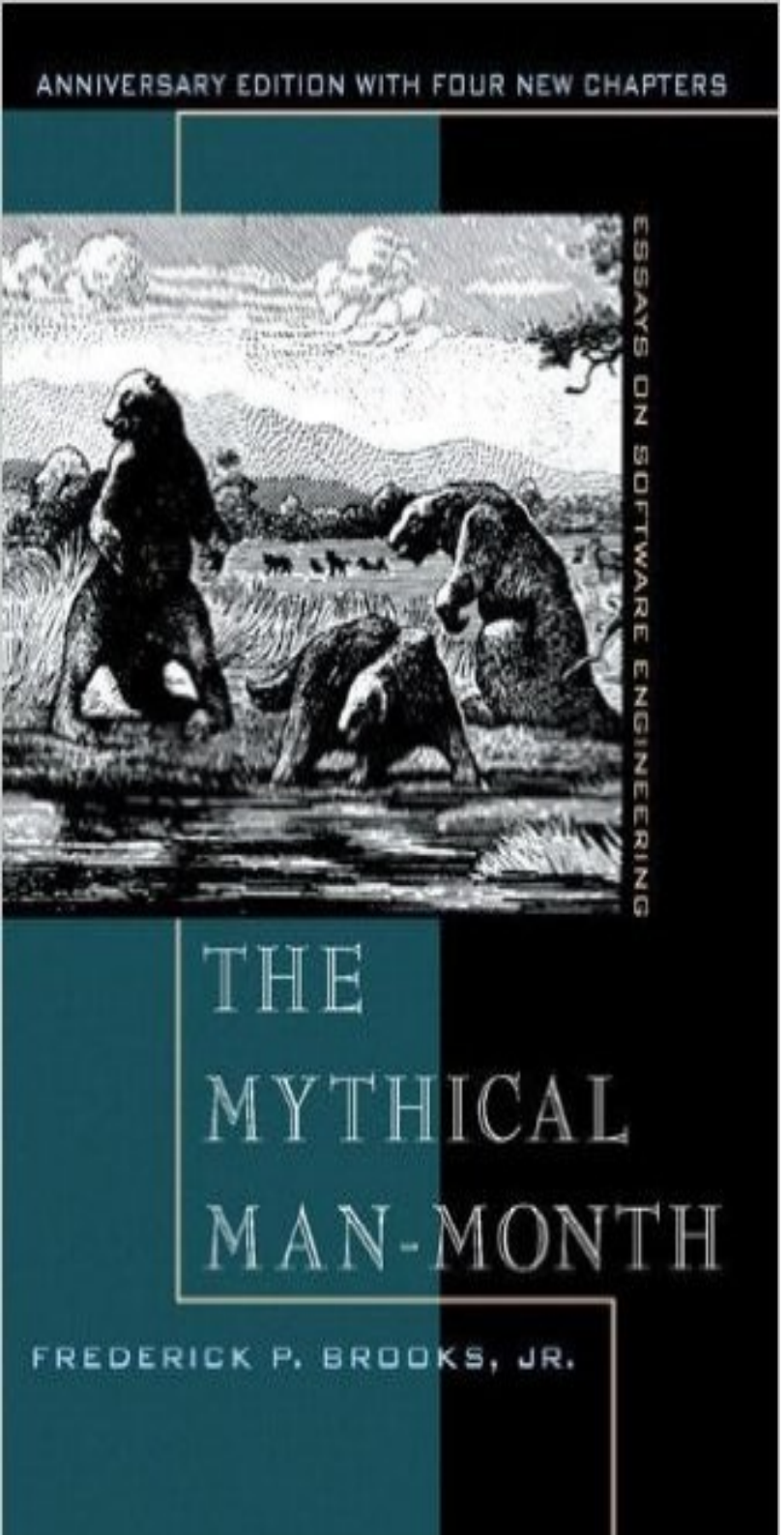
# What a DSL ***is***

- A readable expression of what the code does, not how it does it.
- Encapsulation of the essence of the problem you are solving in syntax.
- Usually occupies a middle ground between configuration and code; configuration with small amounts of executable code.
- A creative use of delaying code evaluation. Normal code gets executed now, DSLs get a lot of their power from evaluating code at unexpected times.
- A shared vocabulary for expressing shared concepts. The “concepts” part is key here. Libraries provide shared ways of doing things, DSL's should be about concepts.

# Why Groovy?

- Statically Compiled, Type Checked, or None of the Above; your choice
- Compile time meta programming and runtime meta programming. Everything from parser manipulation and byte code injection to implementing method missing.
- Operator overloading
- Extension modules
- Command chaining
- Flexible syntax
- Run as a script or pre-compile
- IDE hints as part of the language

Ladies and Gentleman, we now bring you this rant completely free of charge!




OO?

Functional?

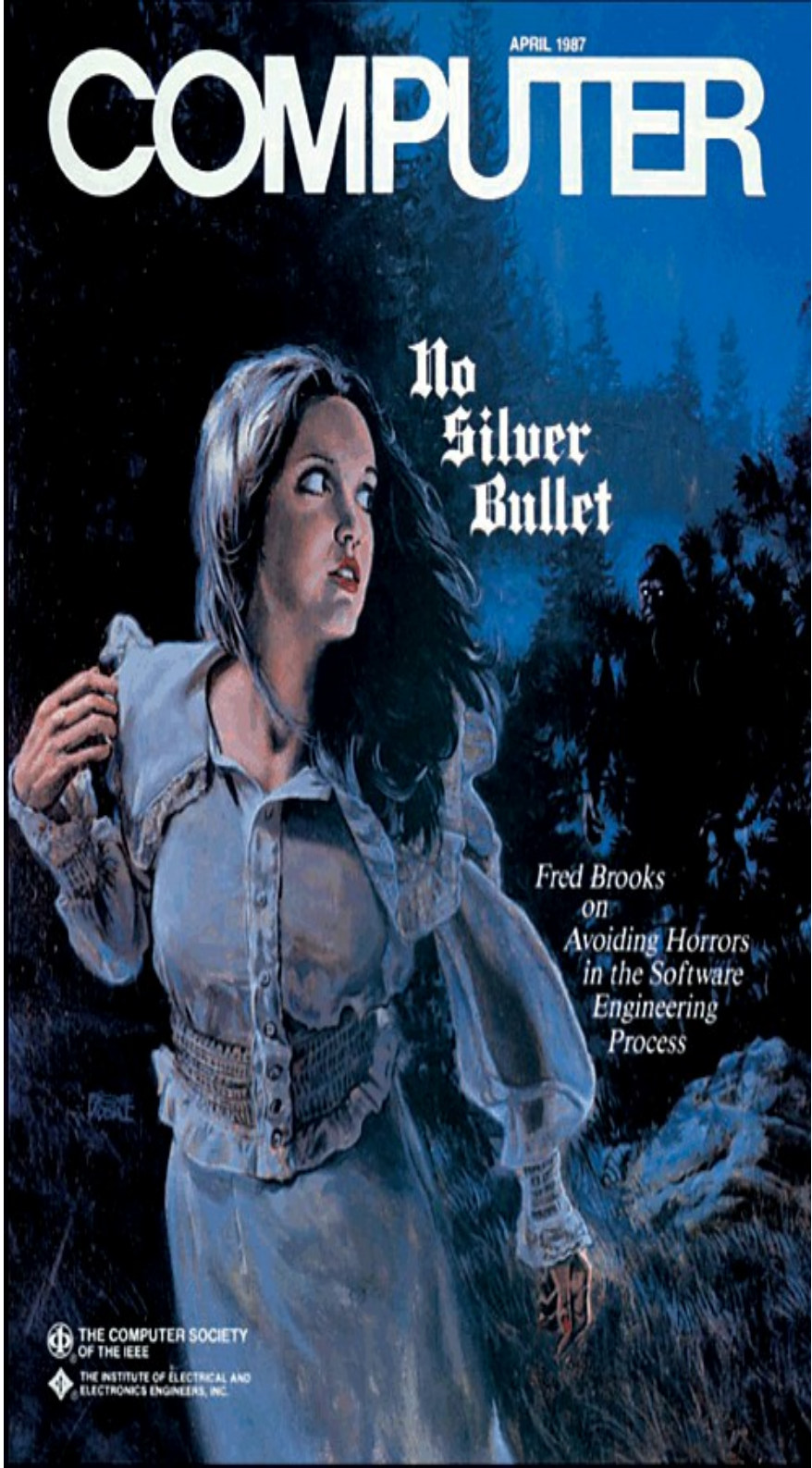
Reactive?

Agile?

Testable?

Readable?

Anyone read Brooks?



# Solve The Problem By Avoiding It

- First off, DSL's are ***NOT*** a silver bullet, programming is just plain hard.
- Avoid the OO/Functional battle with DSL's, you can't tell what is behind the curtain.
- DSL's should be small. Small enough is always readable enough.
- Anything can be agile, since a DSL is something, it can be agile.
- The syntax of the DSL shouldn't need testing.
- The DSL engine is usually easily tested
- Is a DSL reactive? Maybe. Who knows? Who cares?
- Again ***NOT*** a silver bullet, but a valuable way to avoid BS and focus on the **essence of the problem.**

Ladies and Gentlemen, we now return to our regularly scheduled presentation. Now with even more black text on white background!

# So What Makes A Good DSL?

A good DSL documents, describes, and organizes an application. If you can look at a DSL and tell what the application does, you have an excellent DSL. If someone can explain the DSL to you in a short period of time and then you can tell what the application does, you have a very good DSL.

A good DSL usually comes after working on an application for some time, because it usually takes a while to understand what an application should do and how it should do it. The DSL should tell what the application does while hiding how it does it. I can't emphasize enough that DSL's are about what, not how.

# DSL Example #1, Let's Describe a Testing Library

- Most importantly it should allow me **test** that **when** something happens, **then** something else will be true, false, null, etc.
- It should provide for test isolation by allowing me to **set up** a test and **clean it up**.
- It should give me the ability to test for **thrown** exceptions or to make sure certain exceptions were **not thrown**.
- Ideally I'd like to be able to succinctly do lots of similar tests. Even better if I can see them in **tables like I would in a spreadsheet**.
- I want to know exactly **why something fails**, I don't want to spend lots of time figuring this out.
- It should **read like English** as much as possible on the first try, people probably aren't going to spend lots of time refactoring an making this stuff pretty.
- Anything else?

# Congratulations!

You just described spock.
http://spockframework.github.io/spock/docs/1.1-rc-1/

```groovy
def "HashMap accepts null key"() {
  setup:
  def map = new HashMap()

  when:
  map.put(null, "elem")

  then:
  notThrown(NullPointerException)
}
```

```groovy
class DataDriven extends Specification {
    def "maximum of two numbers"() {
        expect:
        Math.max(a, b) == c

        where:
        a | b || c
        3 | 5 || 5
        7 | 0 || 7
        0 | 0 || 0
    }
}
```

# DSL Example #2, Let's Describe an HTTP Client

- Most importantly I want requests to be described in a single piece of code
- I don't want to have to worry about resource management, threading, asynchronous execution, encoding, decoding, etc.
- I want to describe that for a give http verb, with a particular path, headers, query params, etc, then I want the client to execute a piece of code and return the result.
- I'd like the make certain resources available for each request.
- I want to work at whatever level is convenient for what I'm doing, whether that is close to the HTTP layer or at a much higher level.
- Anything else?

# Congratulations!

You just described http-builder-ng.
https://dwclark.github.io/http-builder-ng/

```groovy
//let's configure an http client to make calls to httpbin.org using the default http library
def httpBin = HttpBuilder.configure {
    request.uri = 'http://httpbin.org/'
}

//now let's GET /get endpoint at httpbin.
//This will return a JSON formatted response with an origin property.
def result = httpBin.get {
    request.uri.path = '/get'
}

println("Your ip address is: ${result.origin}")

//Finally lets post a standard http form to httpbin
httpBin.post {
    request.uri.path = '/post'
    request.body = [ input1: 'the first input', input2: 'the second input' ]
    request.contentType = 'application/x-www-form-urlencoded'
}
```

# DSL Example #3, Database Migrations

- About a year ago I gave a talk on database migrations with flyway. You can find this in my github repo (flyway-dsl and flyway-dsl-demo).
- In the talk I described a DSL. This DSL was based on the DSL I developed for managing our database migrations at Research Now. The DSL reads like the requirements I was given for what the application should do.
- Going the DSL route forced me to understand and abstract the application at a high level.
- Benefit #1: I haven't touched the DSL code in over a year, I think this is because writing a DSL forced me to “do it right”.
- Benefit #2: I haven't written a database migration in over a year. I passed the management of all migrations on to a developer with zero Groovy experience. He hasn't asked questions for almost the entire time.
- Benefit #3: I basically don't remember how it works anymore. The best code is the one you can forget about.


ORANGES
APPLES
BANANAS
SOUP
PRODUCE
LETTUCE
FRENCH BREAD
BEAN SOUP
TOMATO SOUP
PAPER TOWELS
ASPIRIN
NOODLES (ELBOW KIND)
BEANS
SCOTCH TAPE
CHAPSTICK
MILK
FILM

# DEMOS

# Like This Talk?

Watch Cedric Do It Better:
https://www.infoq.com/presentations/groovy-dsl-2015