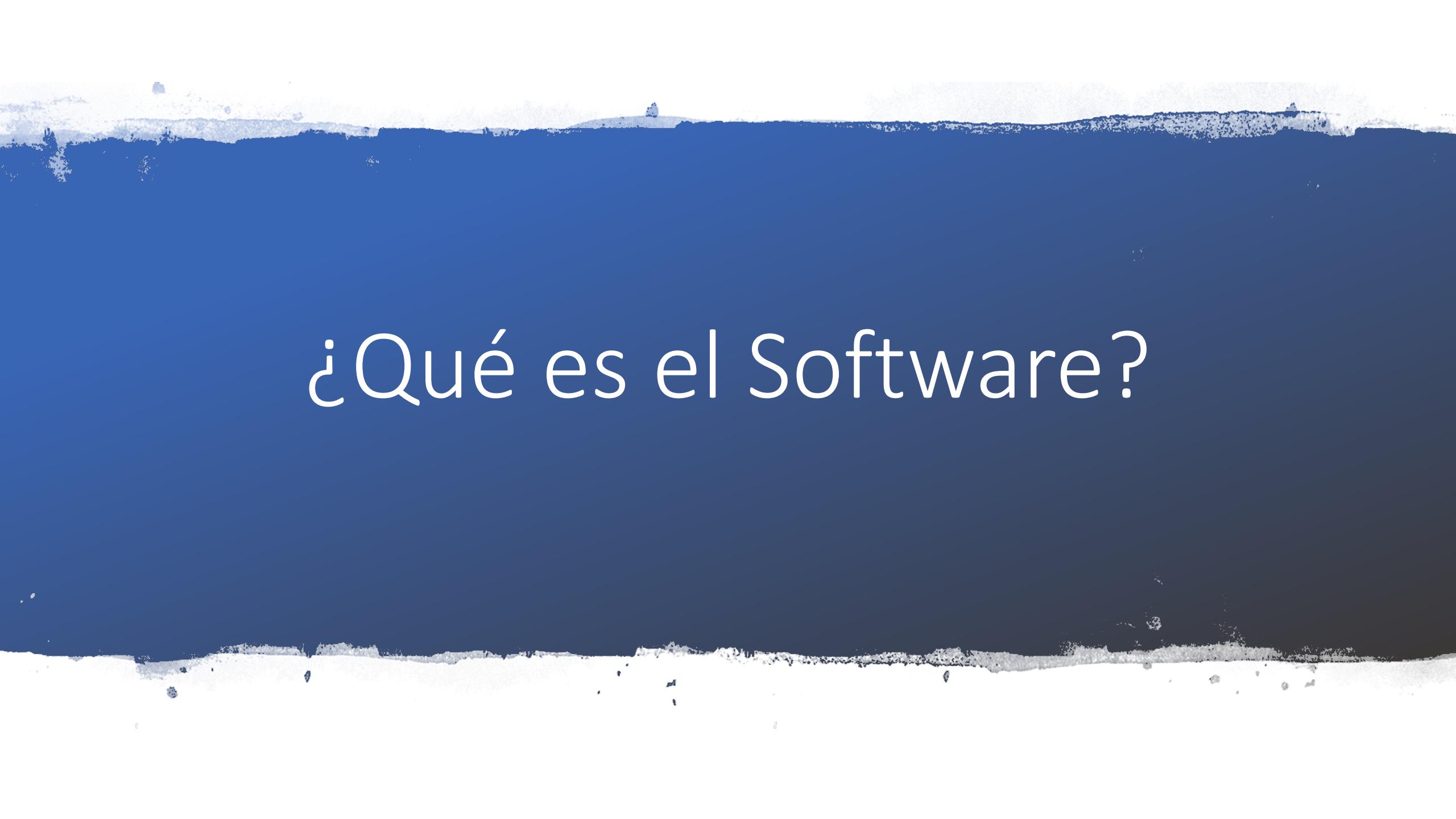


# Software y Desarrollo de Software



# ¿Qué es el Software?



Definición  
Clásica

Secuencia de  
**Instrucciones** que  
dado un input  
genera un output



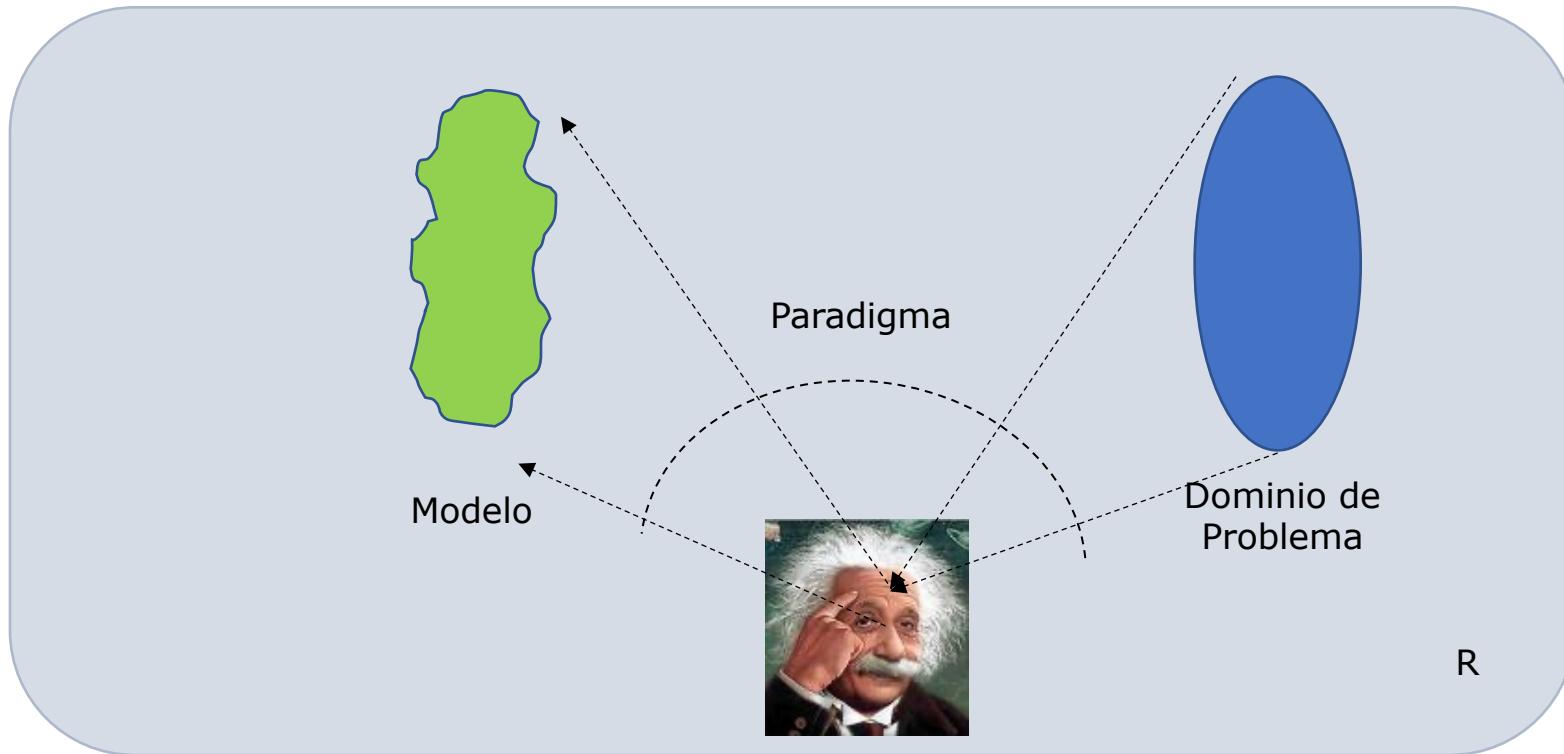
Nuestra  
definición

(Representación de conocimiento a través de un)

**Modelo Computable** de un  
**Dominio de Problema** de la  
**Realidad**

# Definición de Software

**Modelo Computable de un Dominio de Problema de la Realidad**



# Definición de Software

- **Realidad**: Todo aquello que podemos percibir, tocar, hablar sobre, etc
- **Dominio de Problema**: Un recorte de la realidad que nos interesa para el negocio que estamos modelando

# Definición de Software

- **Modelo**: Representación de aquello que se está modelando
- **Computable**: Que puede ejecutar en una máquina de Turing → Formal, a-contextual
  - Característica esencial: No solo especifica el qué sino que además implementa el cómo

# ¿Cuál es el modelo?

- Glenn Vandenburg: Real software engeneering
  - <https://youtu.be/RhdIBHHimeM>



# Modelo

- **Buen Modelo:**
  - Eje Funcional: Qué tan buena es la representación del dominio
  - Eje Descriptivo: Qué tan bien está descripto el modelo, qué tan “entendible es”
  - Eje Implementativo: Cómo “ejecuta” en el ambiente técnico

# Buen Modelo – Eje Implementativo

- Un modelo es bueno cuando ejecuta en el tiempo esperado usando los recursos definidos como necesarios
  - Performance
  - Espacio
  - Escalabilidad
  - Todo lo relacionado con Requerimientos No Funcionales
  - Es la parte “**detallista**” del desarrollo

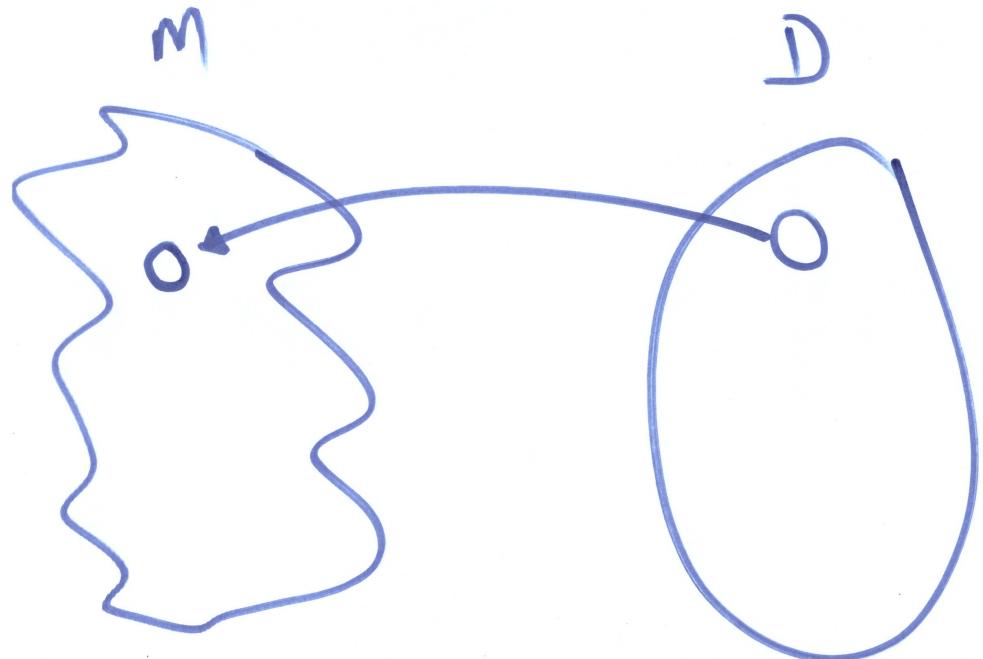
# Buen Modelo – Eje Descriptivo

- Un modelo es bueno cuando se lo puede “entender” y por lo tanto “cambiar”
  - Importísimo usar buenos nombres
  - Importísimo usar mismo lenguaje que el del dominio de problema
  - El código debe ser “lindo”
  - Es la parte “**artística**” del desarrollo

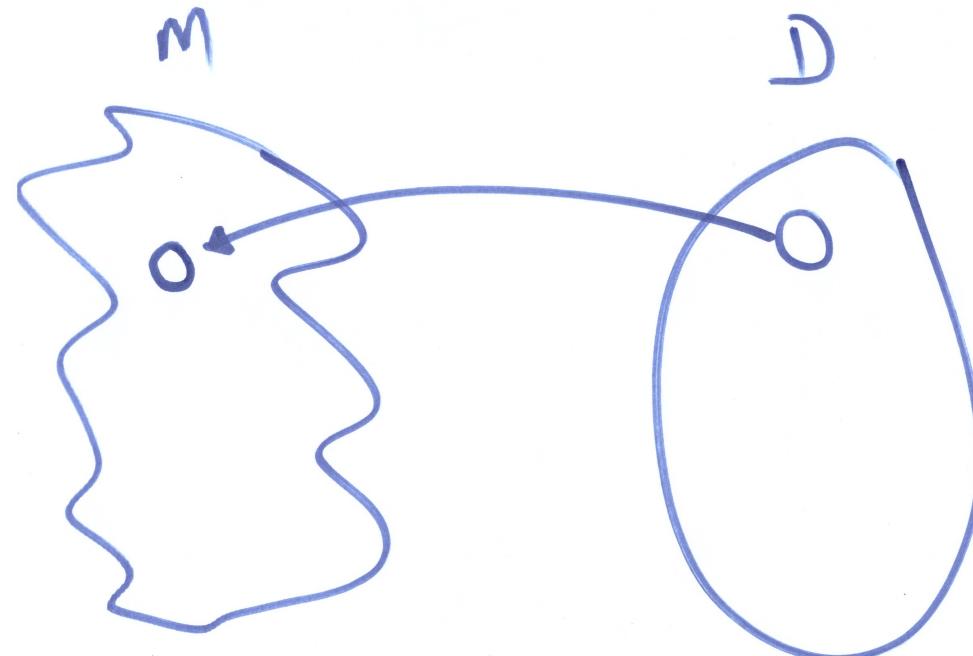
# Buen Modelo – Eje Funcional

- Un modelo es bueno cuando puede representar correctamente toda observación de aquello que modela
- Si aparece algo nuevo en el dominio, debe aparecer algo nuevo en el modelo (no modificarlo)
- Si se modifica algo del dominio, solo se debe modificar su representación en el modelo
- Relación 1:1 dominio-modelo (isomorfismo)
- Es la parte “**observacional**” del desarrollo

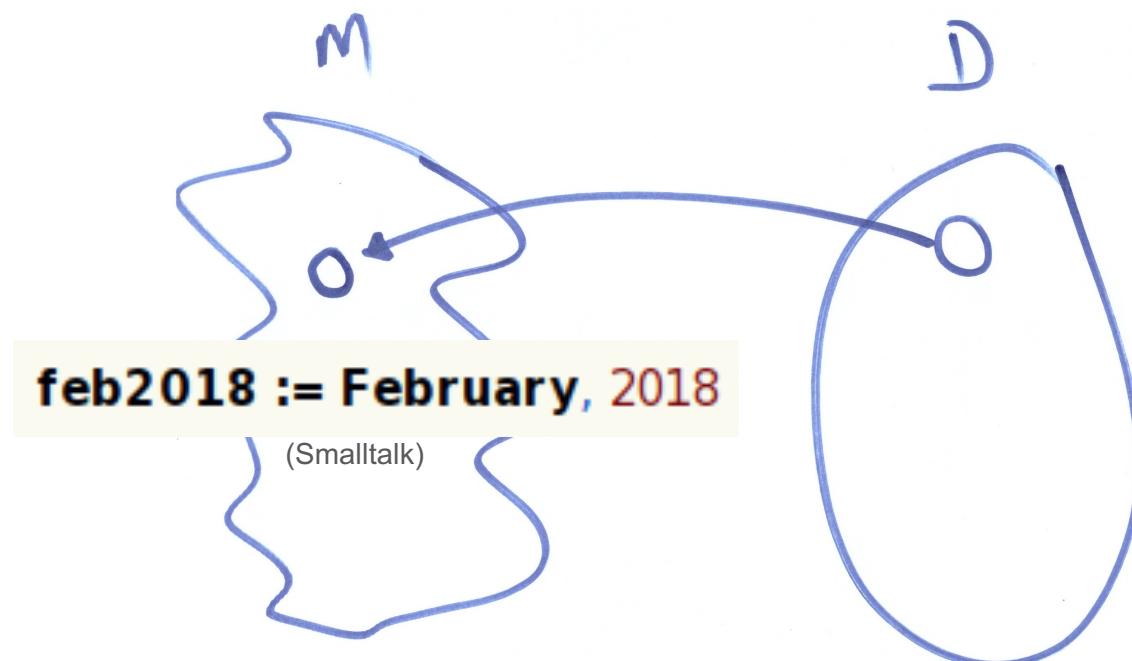
# Modelo



Cómo representamos Feb/2018? (un mes de año)



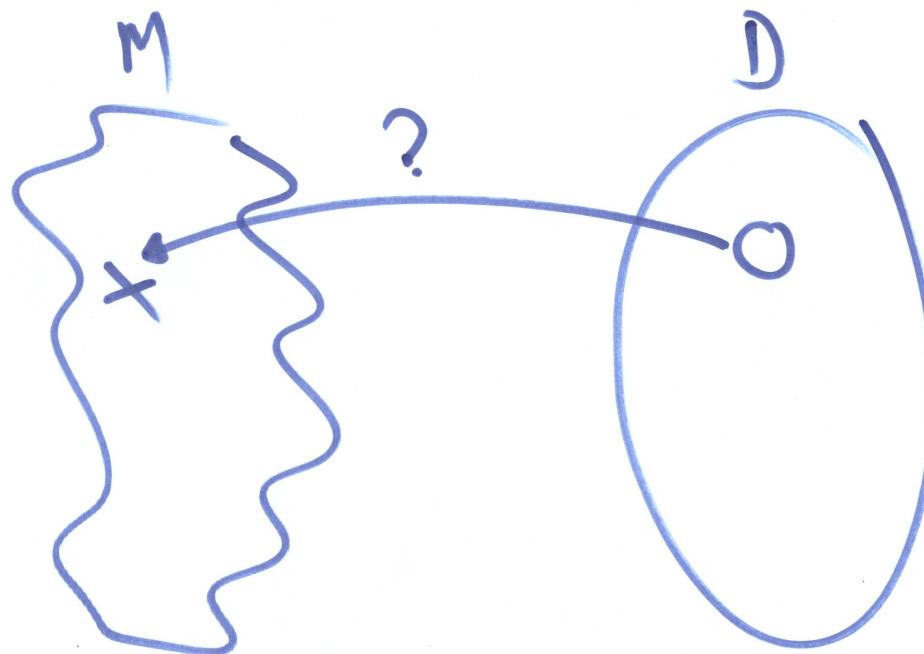
Cómo representamos Feb/2018? (un mes de año)



Cómo representamos Feb/2018? (un mes de año)  
Ruby – JavaScript - Go - Java pre 1.8

?

Cómo representamos Feb/2018? (un mes de año)  
Ruby – JavaScript - Go - Java pre 1.8 – ¡NO SE PUEDE!

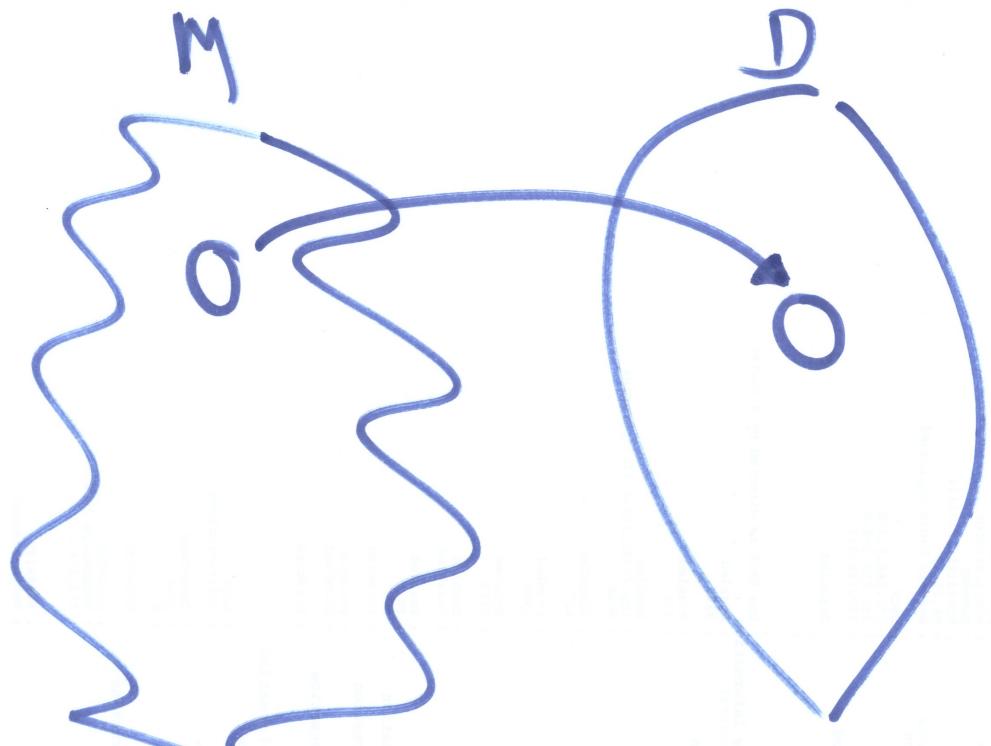


# Problemas

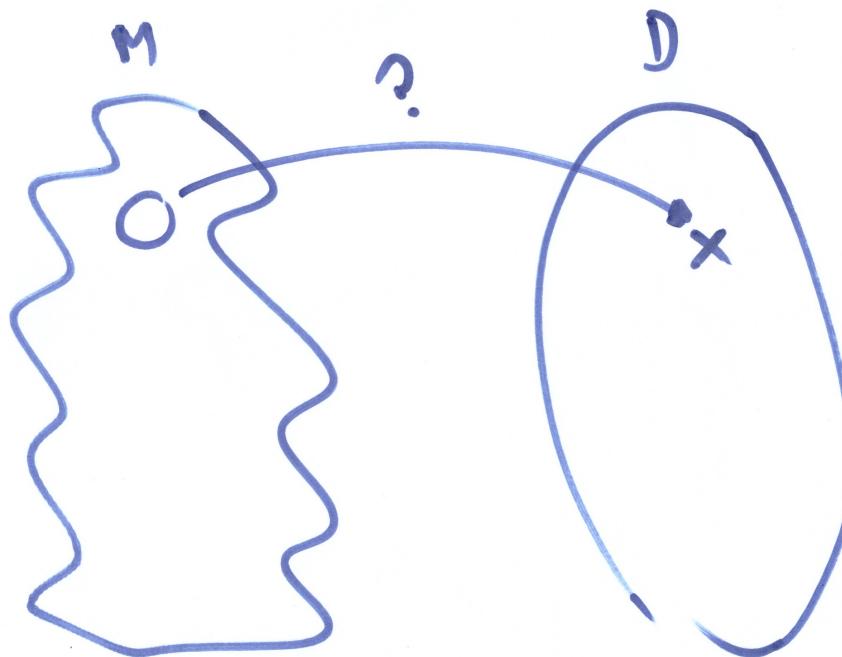
- Falta de abstracciones y representación
- Reinventamos la “rueda pinchada” (Alan Kay)
- Soluciones Ad-hoc
- Errores de Comunicación
- ...
- ¿qué hay acerca de la inmutabilidad?



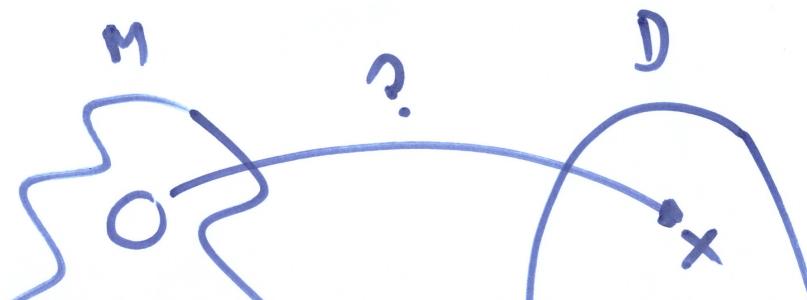
¿Qué pasa con la vuelta?



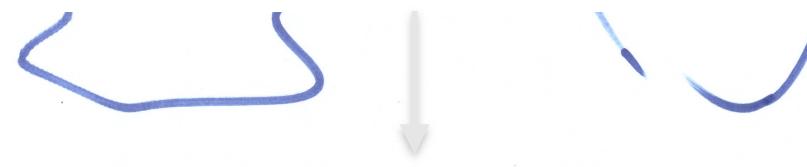
¿Deberíamos poder representar 31 de Feb de 2018?



# ¿Deberíamos poder representar 31 de Feb de 2018? Java pre 1.8



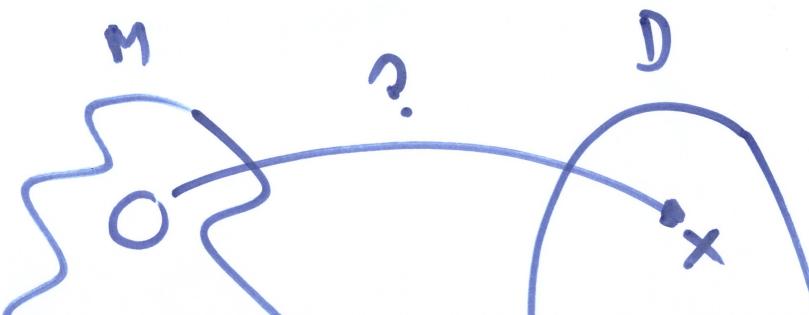
```
SimpleDateFormat formater = new SimpleDateFormat("MM/dd/yyyy");
Calendar feb31st2018 = Calendar.getInstance();
feb31st2018.set(2018, Calendar.FEBRUARY, 31);
System.out.println(formater.format(feb31st2018.getTime()));
```



- 03/03/2018 !!!

# ¿Deberíamos poder representar 31 de Feb de 2018?

## Go

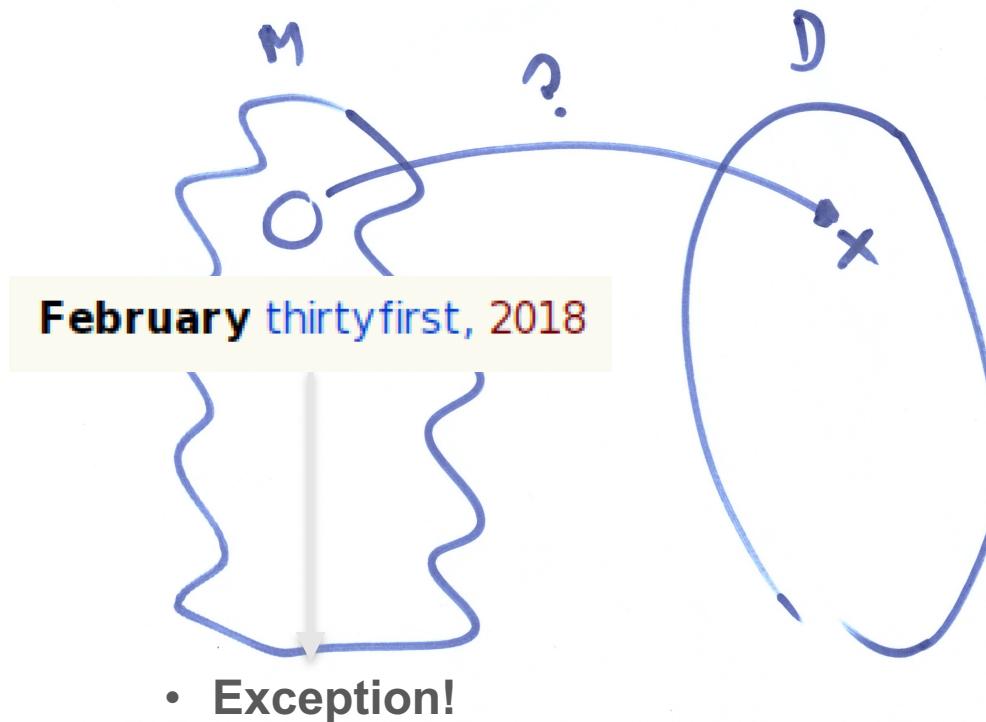


```
feb31st2018 := time.Date(2018, time.February, 31, 0, 0, 0, 0, time.UTC)  
fmt.Println(feb31st2018.Local())
```



- 03/03/2018 !!!

# ¿Deberíamos poder representar 31 de Feb de 2018? Smalltalk

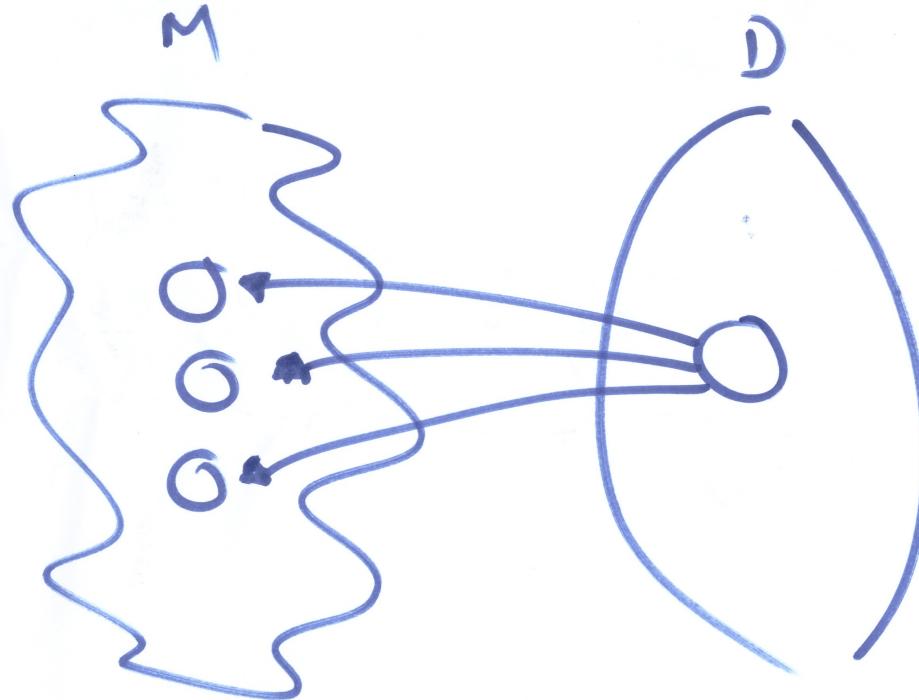


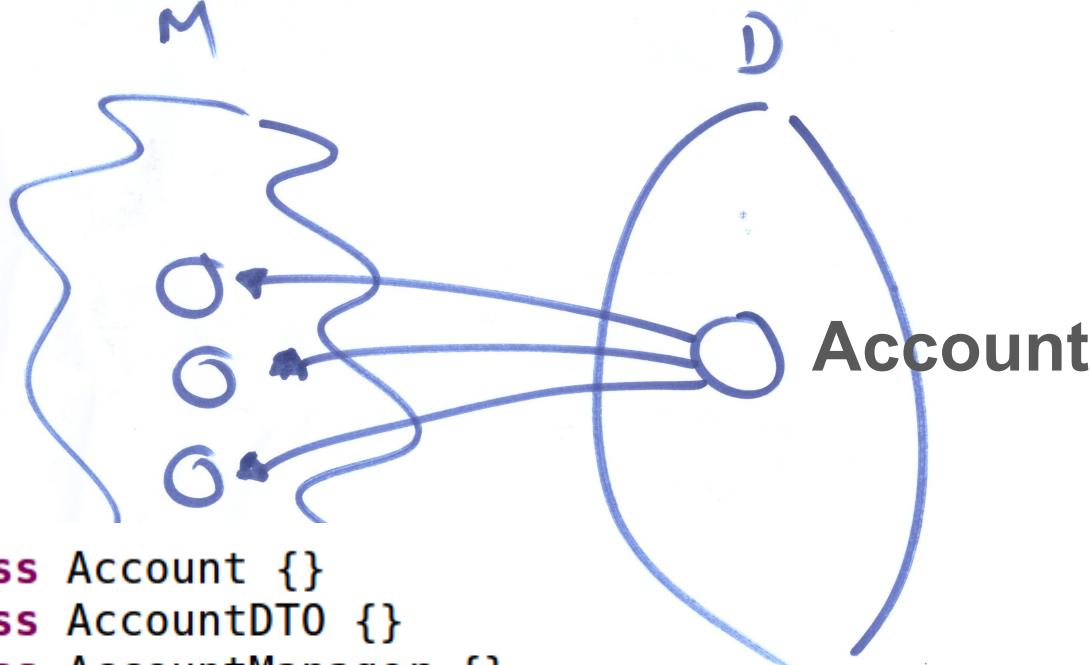
# Problemas

- Objetos inválidos
  - Comportamiento inesperado
  - Errores ocultos
  - Dificultad de encontrar errores
  - ...
- 
- ¿Qué pasaría si todos los objetos fuesen válidos, si no pudieran existir objetos inválidos? ¿Cómo cambiaría tu manera de programar?



¿Qué sucede con esta relación?



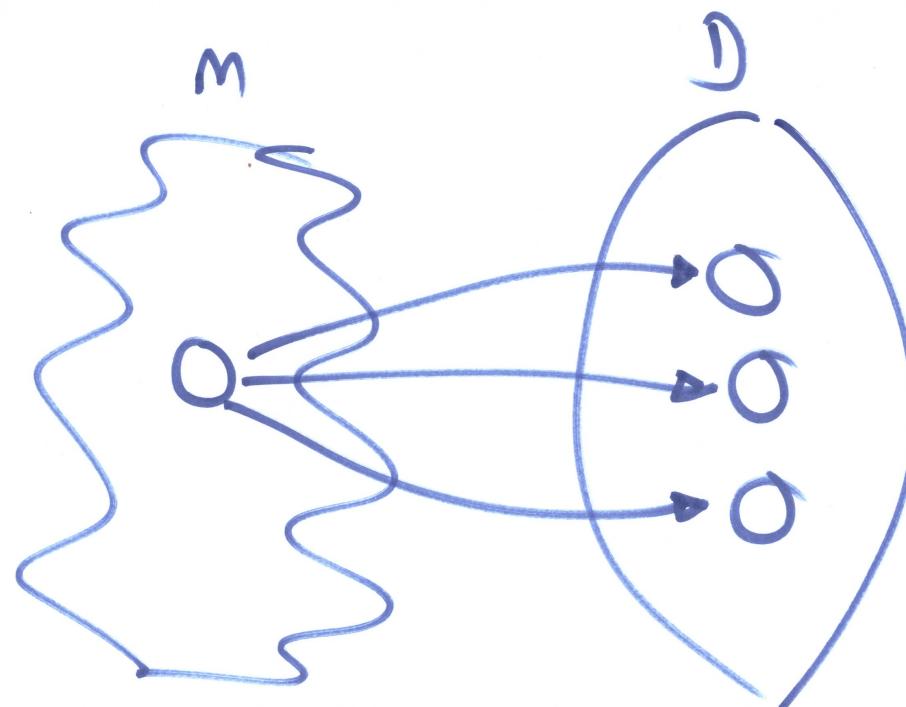


```
class Account {}  
class AccountDTO {}  
class AccountManager {}  
class AccountHelper {}
```

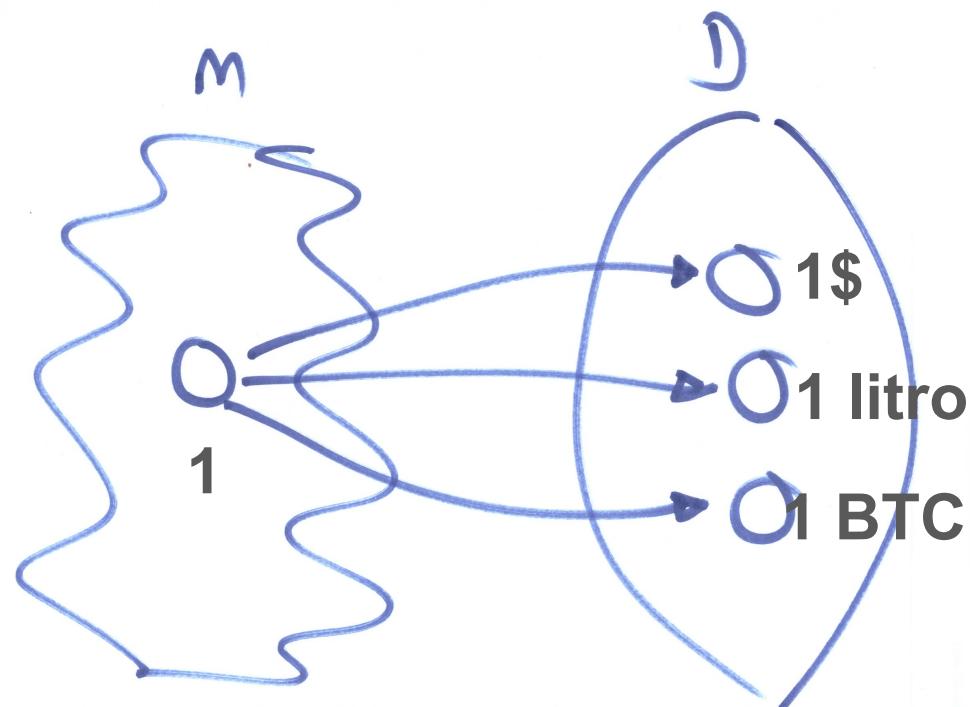
...

¡¡Complejidad Accidental!!

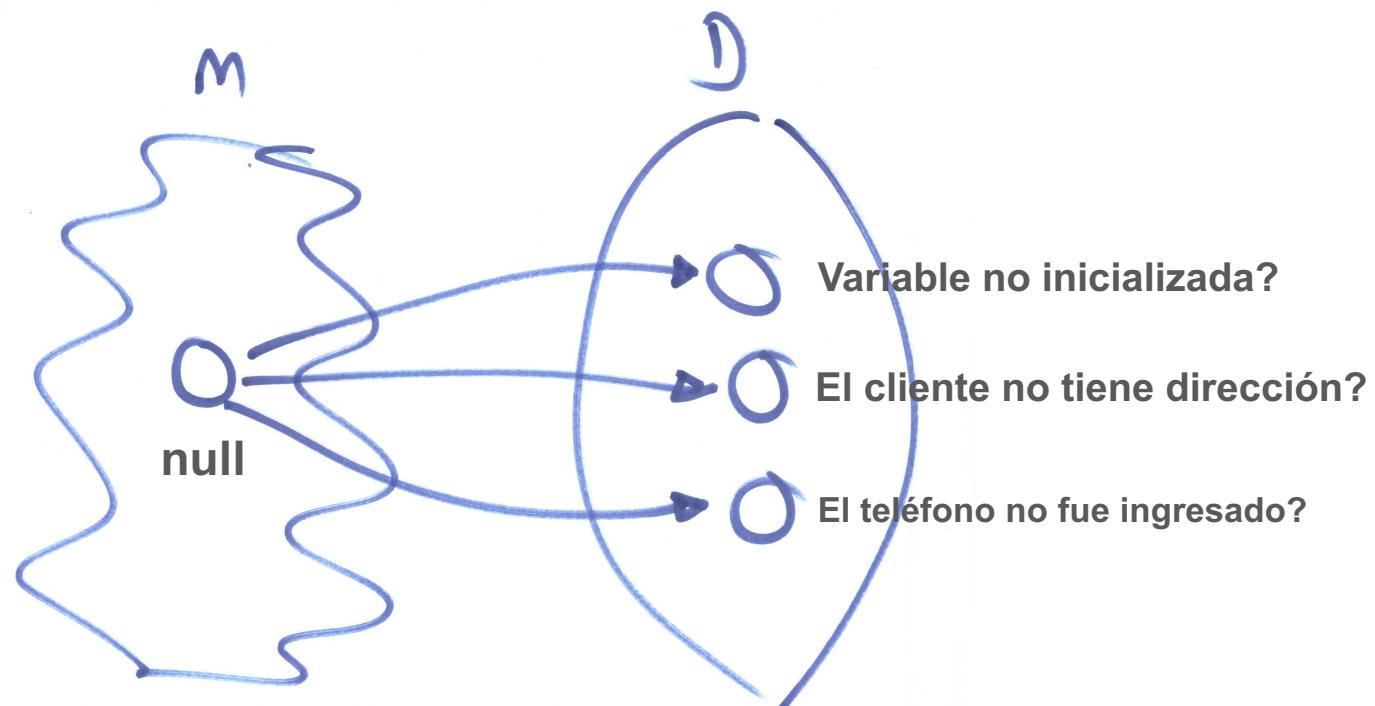
¿Qué pasa con esta relación?



¿Qué representa 1?



## ¿Qué representa “null”?



View Presentation



# InfoQ

Enterprise Software Development Community

▶ 0:00 / 1:01:58



—

01:01:58

# Null References: The Billion Dollar Mistake

Tony Hoare

## Summary

Tony Hoare introduced Null references in ALGOL W back in 1965 "simply because it was so easy to implement", says Mr. Hoare. He talks about that decision considering it "my billion-dollar mistake".

## Bio

Sir Charles Antony Richard Hoare, commonly known as Tony Hoare, is a British computer scientist,

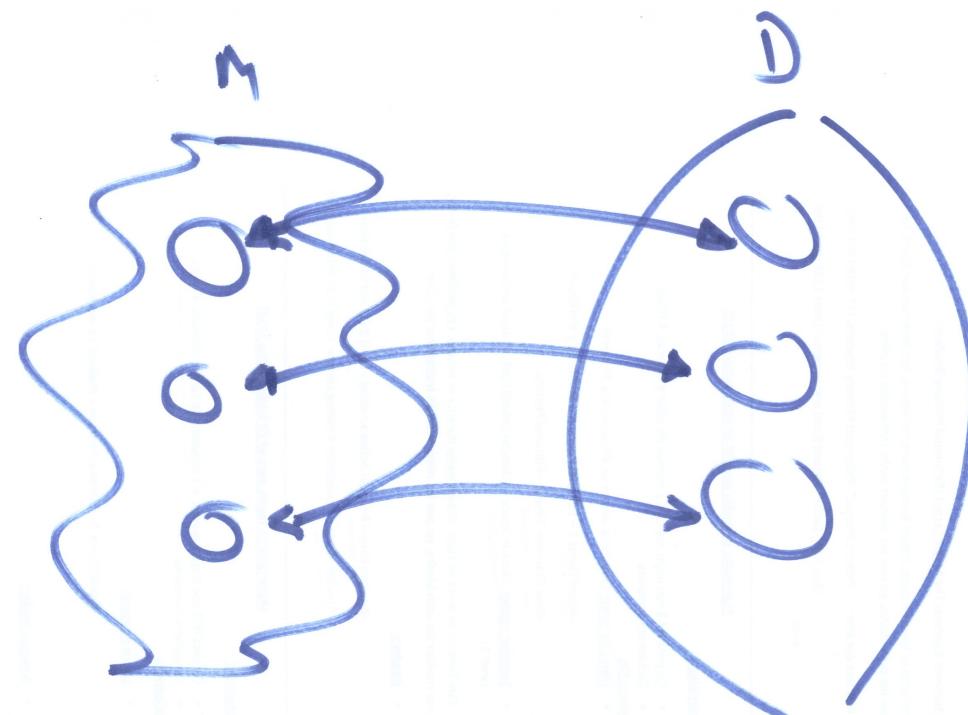
▲ Key Takeaways

# Problemas

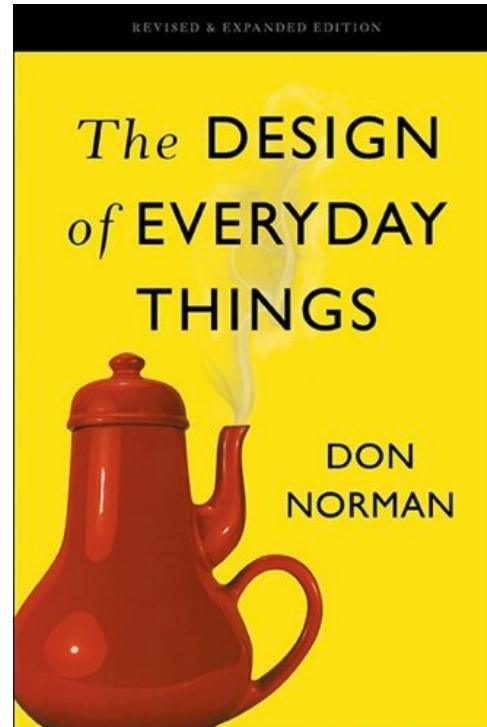
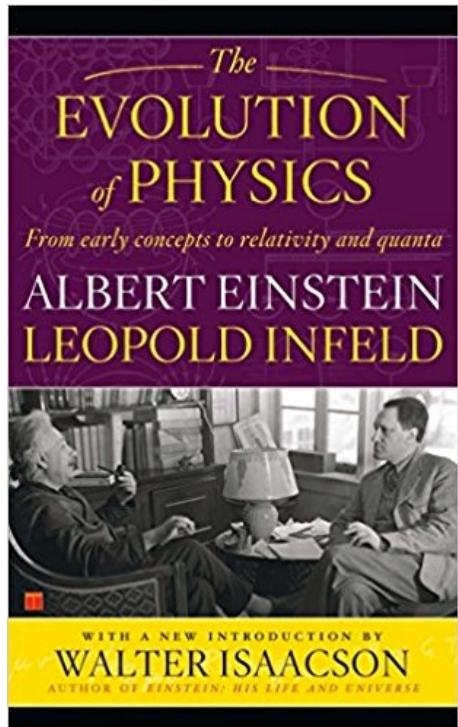
- Pérdida de información
  - Errores de implementación
  - ...
- 
- Design Tip: No usar null/nil



## Conclusión

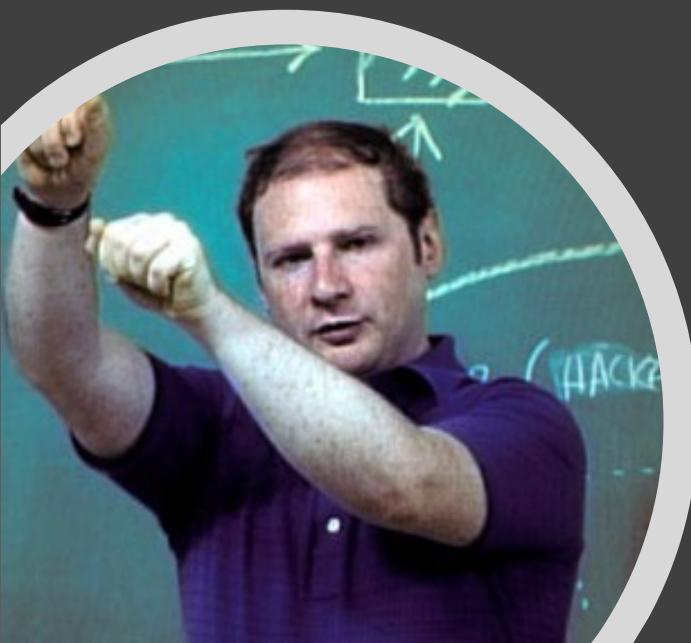
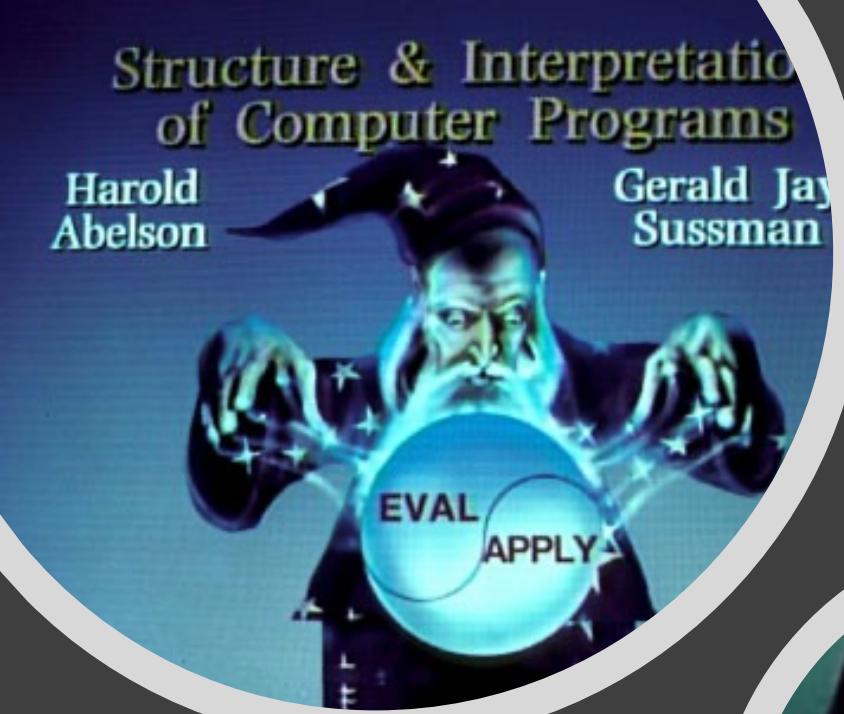


No encontraremos estas ideas en libros de nuestra profesión





# ¿Qué es el Desarrollo de Software?



# ¿Es una Ciencia?

- <http://groups.csail.mit.edu/mac/classes/6.001/abelson-sussman-lectures/>

# Aportando al a confusión: ¿Existe la *ciencia de la computación*?

- Fred Brooks:
  - Un dicho académico es que “Todo lo que se hace llamar *ciencia*, no lo es”.
  - “Física vs. *Ciencias de la Computación*”
- Ambos construimos, pero con distintos propósitos
  - *El científico construye para aprender*
  - *El ingeniero aprende para construir*
- Heinz Zemanek: “La computación es la ingeniería de objetos abstractos”



Fuente: Fred Brooks. The computer scientist as a Toolsmith II. Communications of the ACM, Marzo 1996.



¿Es una  
ingeniería?

# ¿Existe la Ingeniería de Software?

- Ingeniería de Software según Dijkstra:
  - Así como la economía se conoce como “La ciencia miserable”, la Ingeniería de Software debería ser conocida como “The Doomed Discipline”.
  - La ingeniería de software, por supuesto, se presenta a si misma como otra causa meritaria, *pero eso es puro cuento*. Si uno lee cuidadosamente su literatura y analiza lo que sus devotos realmente hacen, descubre que la ingeniería de software ha aceptado como su carta de presentación “como programar si no puede”.
  - Es realmente útil ver un programa como una fórmula. Primero, pone la tarea del programador en perspectiva: tiene que derivar esa fórmula... manipulación de símbolos usualmente llamada 'programación'.

Fuente: ver <http://www.cs.utexas.edu/users/EWD/transcriptions/EWD13xx/EWD1305.html>



# ¿Existe la Ingeniería de Software? (cont.)

- Ingeniería de Software según Yourdon:
  - La construcción de sistemas de software que funcionen no es una “ciencia”. Los así llamados “científicos en computación” nos tratan de convencer de que nuestros sistemas son en realidad grafos dirigidos, o n-tuplas de formas normalizadas, o autómatas de estados finitos... Sus pronunciamientos son mas relevantes a Zen que al difícil problema de construir sistemas y programas útiles y fáciles de mantener.
  - En lugar de aproximaciones académicas, lo que necesitamos es un conjunto de métodos prácticos que traten con la naturaleza propensa a errores del personal de proyectos, los usuarios, los encargados del mantenimiento, y el proceso de desarrollo. Ese conjunto de métodos es llamado “Ingeniería de Software”.



DAVID FARLEY



# MODERN SOFTWARE ENGINEERING



Doing What Works to  
**Build Better Software Faster**

*Foreword by TRISHA GEE*



# Modern Software Engineering

## Ingeniería – La aplicación práctica de la Ciencia

El desarrollo de software es un proceso de descubrimiento y exploración; por lo tanto, para tener éxito en ello, los ingenieros de software deben convertirse en expertos en el aprendizaje.

El mejor enfoque de la humanidad para el aprendizaje es la ciencia, por lo que debemos adoptar las técnicas y estrategias de la ciencia y aplicarlas a nuestros problemas. Esto a menudo se malinterpreta en el sentido de que debemos convertirnos en físicos midiendo cosas a niveles de precisión irrazonables, en el contexto del software. La ingeniería es más pragmática que eso.

Lo que quiero decir cuando digo que debemos aplicar las técnicas y estrategias de la ciencia es que debemos aplicar algunas ideas bastante básicas, pero sin embargo extremadamente importantes.

Wikipedia describe el método científico que la mayoría de nosotros aprendimos en la escuela como:

- **Caracterizar:** Hacer una observación del estado actual.
- **Formular una hipótesis:** Cree una descripción, una teoría que pueda explicar su observación.
- **Predecir:** Haz una predicción basada en tu hipótesis.
- **Experimentar:** Prueba tu predicción.

# Modern Software Engineering

***La ingeniería de software es la aplicación de un enfoque científico empírico para encontrar soluciones económicas y eficientes a problemas prácticos de software.***

La adopción de un enfoque de ingeniería para el desarrollo de software es importante por dos razones principales. Primero, el desarrollo de software es siempre un ejercicio de descubrimiento y aprendizaje, y Segundo (Francia), si nuestro objetivo es ser "eficientes" y "económicos", entonces nuestra capacidad de aprender debe ser sostenible. Esto significa que debemos gestionar la complejidad de los sistemas que creamos de forma que mantengamos nuestra capacidad de aprender cosas nuevas y adaptarnos a ellas.

Entonces, debemos convertirnos en expertos en el aprendizaje y expertos en el manejo de la complejidad. Hay cinco técnicas que forman las raíces de este enfoque en el aprendizaje. En concreto, para convertirse expertos en aprendizaje, necesitamos lo siguiente:

- Iteración
- Comentario
- Incrementalismo
- Experimentación
- Empirismo

“In science if you know what you are doing you should not be doing it. In engineering if you do not know what you are doing you should not be doing it”

The Art of Science and Engineering - Richard Hamming



Pero nuestro trabajo es...  
Trabajo Creativo

# **Manifesto for Agile Software Development**

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

**Individuals and interactions** over processes and tools

**Working software** over comprehensive documentation

**Customer collaboration** over contract negotiation

**Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

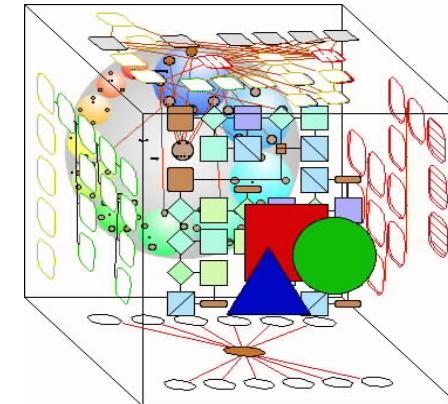
Entonces...  
¿Qué es el Desarrollo  
de Software?

# Desarrollo de Software

- Algo único y particular
- La mejor analogía: Similar a un proceso de aprendizaje



Adquisición de  
Conocimiento



Representación de  
Conocimiento

# Proceso de Desarrollo de Software

- El desarrollo de software es un **proceso de aprendizaje**, lo que implica:
  - Es iterativo
  - Es incremental
  - El conocimiento se genera a partir de hechos concretos
  - El conocimiento generado debe ser organizado

# Proceso de Desarrollo de Software

- Características
  - El dominio de problema está generalmente especificado en lenguajes ambiguos y contextuales (ej. Lenguaje natural)
  - El proceso de desarrollo implica desambiguar y descontextualizar el conocimiento del dominio de problema

# Proceso de Desarrollo de Software

## ➤ Características

- El proceso de desarrollo implica hacer explícito y externo el conocimiento implícito e internalizado de los expertos de dominio
- El CAMBIO es una característica esencial del software, no accidental porque:
  - Cambia el dominio de problema
  - Cambia nuestro entendimiento del dominio de problema
  - Cambia la manera de modelar lo que entendemos del dominio de problema

# Feedback Inmediato

Inventing on principle: <http://vimeo.com/36579366>

Bret Victor



# Conclusiones

- Software = Modelo Computable
- Desarrollo de Software = Proceso de Aprendizaje
  - Iterativo
  - Incremental
  - Feedback Inmediato
- Sobre estos pilares basaremos todo lo visto en la materia