Ingeniería de Software I

FCEyN - UBA

© Cátedra de Ingeniería de Software I – FCEN – UBA, 2019

¿Quiénes somos?

- Profesor: Hernán Wilkinson
- JTP: Emilio Oca, Agustín Martinez
- Ayudantes: Francisco Tarulla, Nicolás Rinaldi, Maximiliano Contieri, Luciano Leveroni, Paula Verghelet y Mirko Torrico

¿Qué expectativa tienen de la materia?

¿Qué es la Ingeniería de Software para ustedes?

¿Qué es el Software?

Definición Clásica

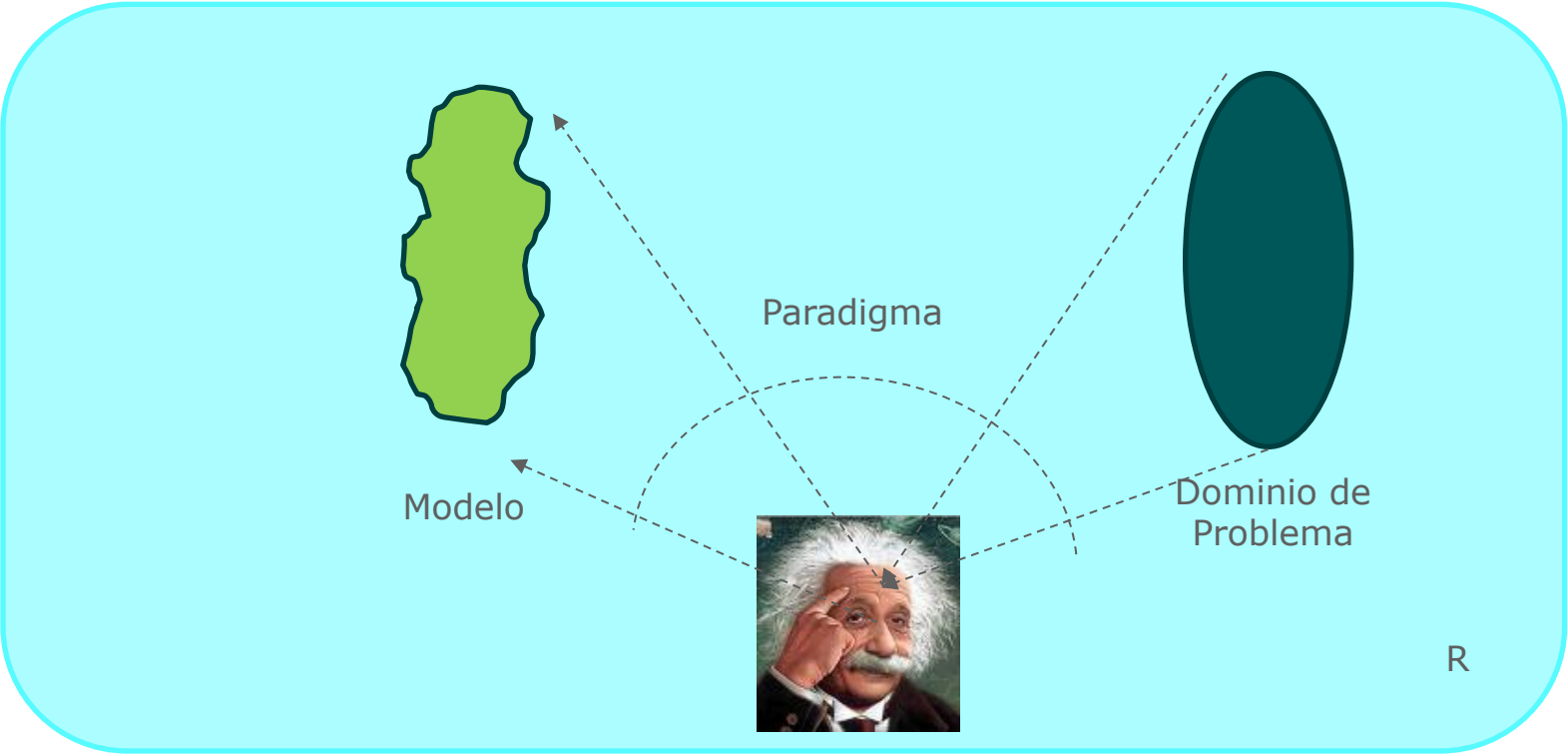
Conjunto de Instrucciones que dado un input genera un output

Nuestra definición

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

Definición de Software

<u>Modelo Computable</u> de un **<u>Dominio de Problema</u>** de la **<u>Realidad</u>**



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

https://www.youtube.com/watch?v=NP9AIUT9nos&t=3s

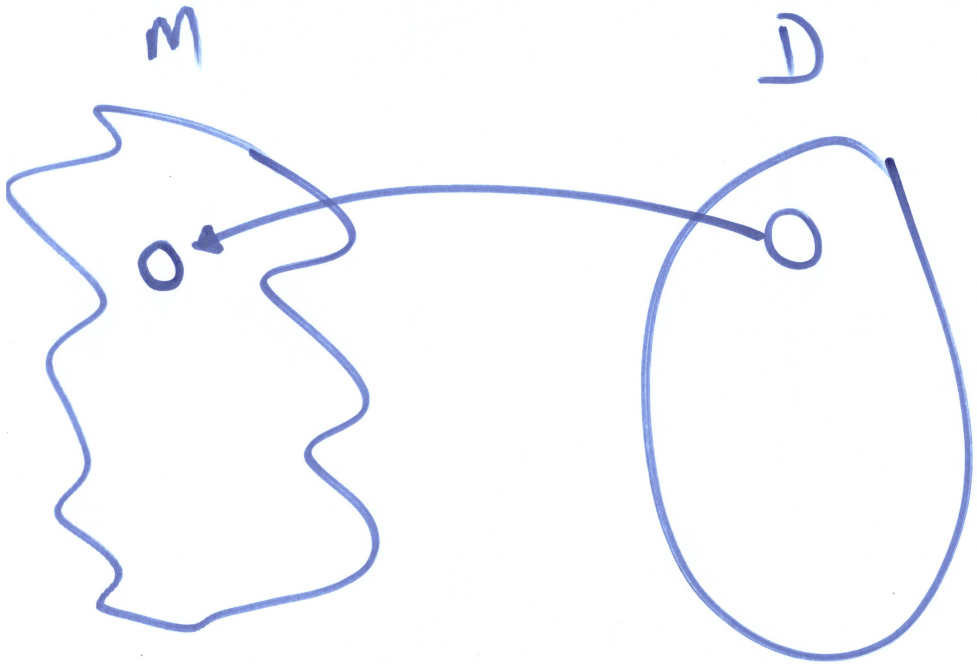
Modelo

- **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

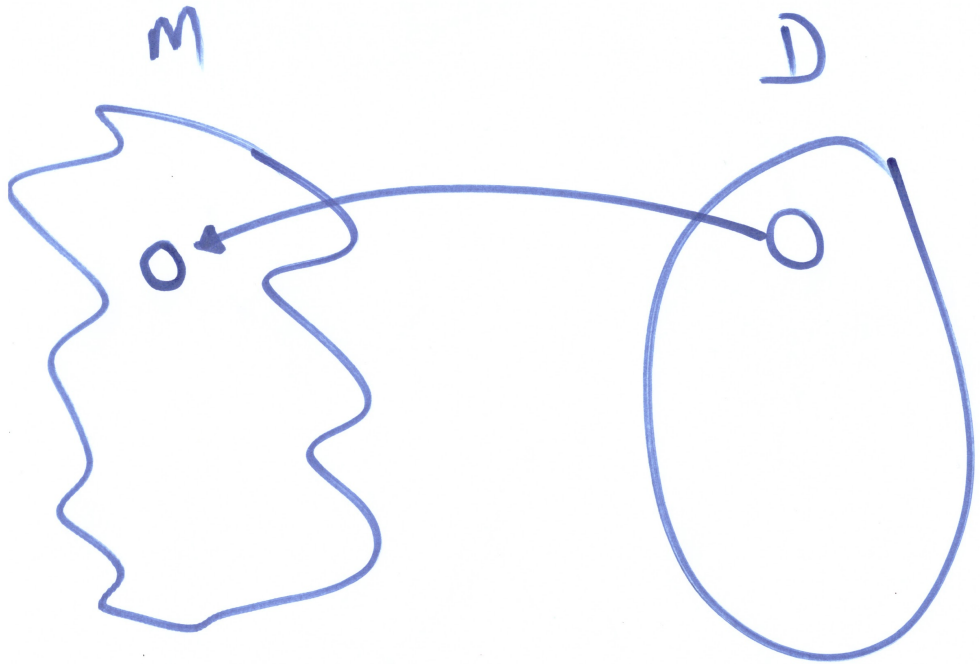
Buen Modelo – Eje Funcional

- 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

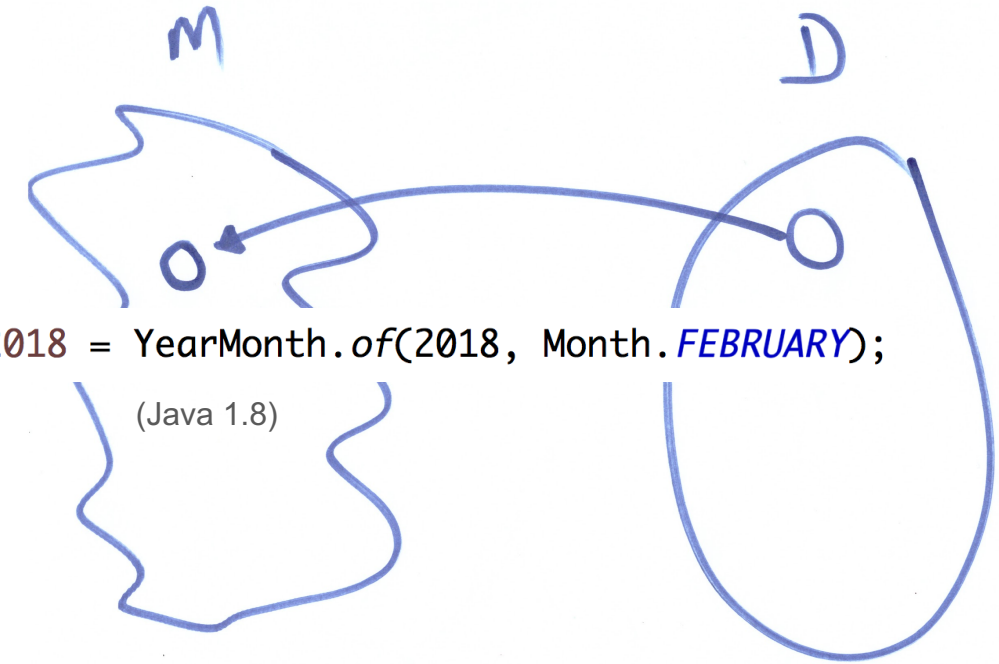
Modelo



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



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




```
YearMonth feb2018 = YearMonth.of(2018, Month.FEBRUARY);
```

(Java 1.8)

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

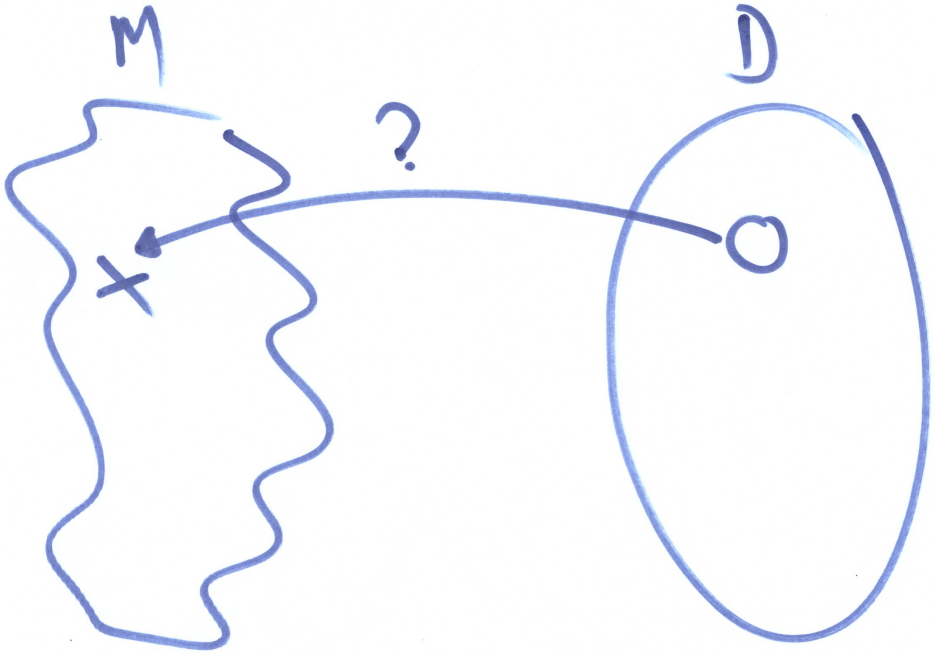
```
feb2018 := February, 2018
```

(Smalltalk)

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

?

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

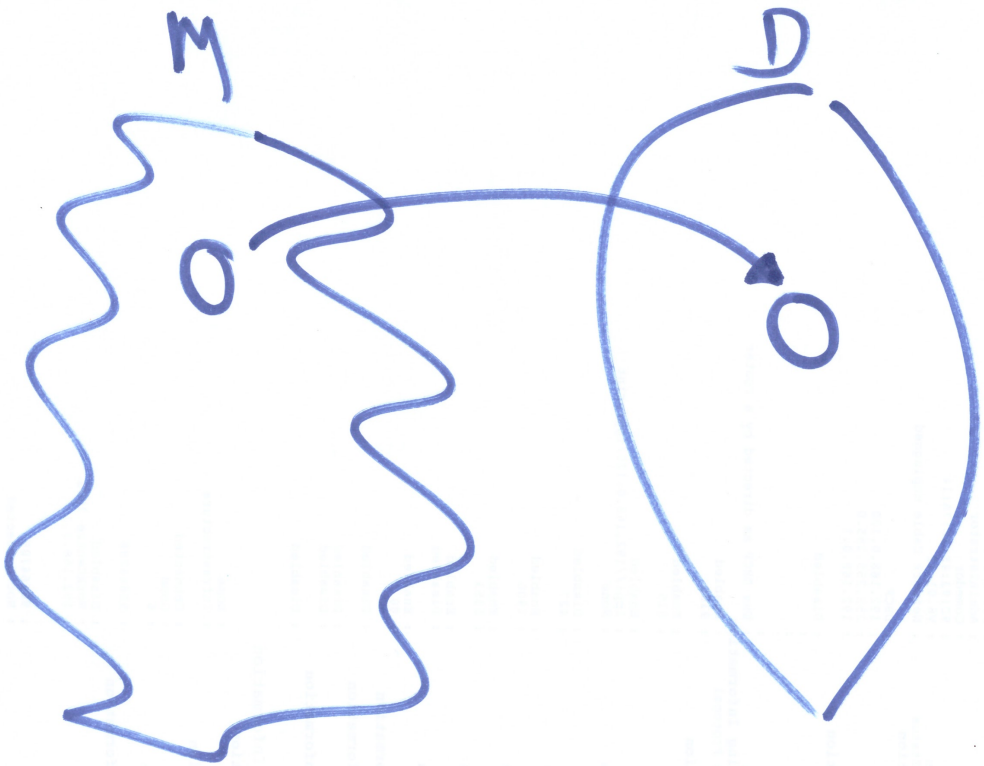


Problemas

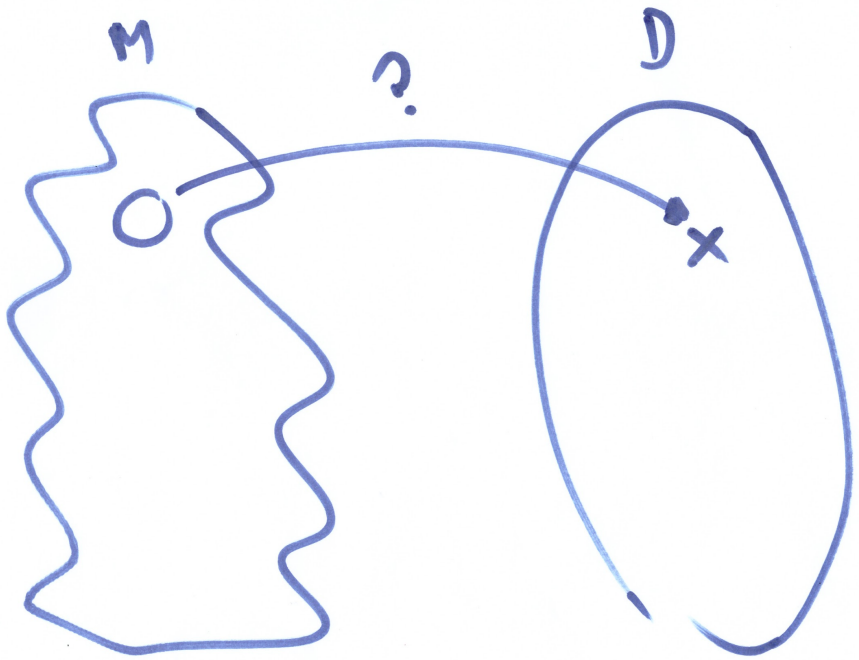
- Falta de abstracciones y representación
- Reinventamos la ”rueda pinchada” (Alan Kay)
- Soluciones Ad-hoc
- Errores de Comunicación
- …

- ¿qué hay acerca de la inmutabilidad?

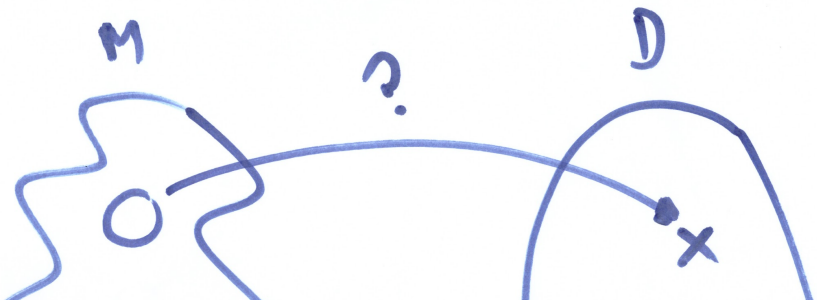
¿Qué pasa con la vuelta?



¿Deberíamos poder representar 31 de Feb de 2018?



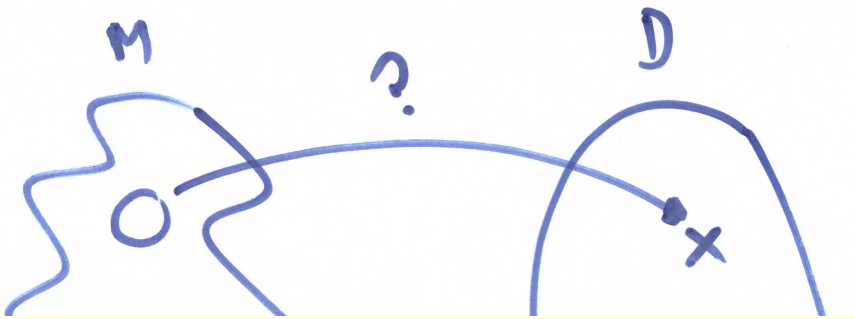
¿Deberíamos poder representar 31 de Feb de 2018?
Java pre 1.8




```
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 !!!

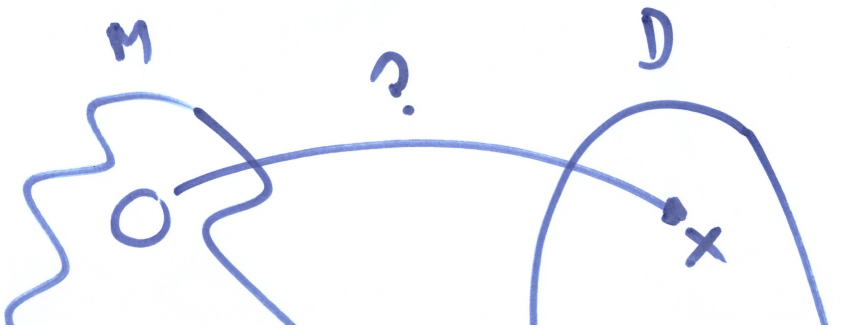
¿Deberíamos poder representar 31 de Feb de 2018?
Go



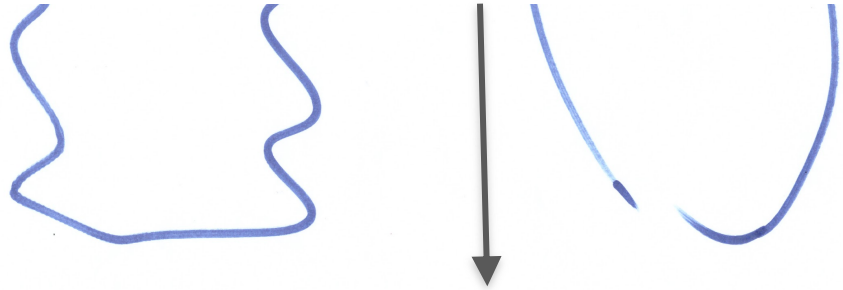

```
feb31st2018 := time.Date(2018, time.February, 31, 0, 0, 0, 0, time.UTC)
fmt.Print(feb31st2018.Local())
```

03/03/2018 !!!

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

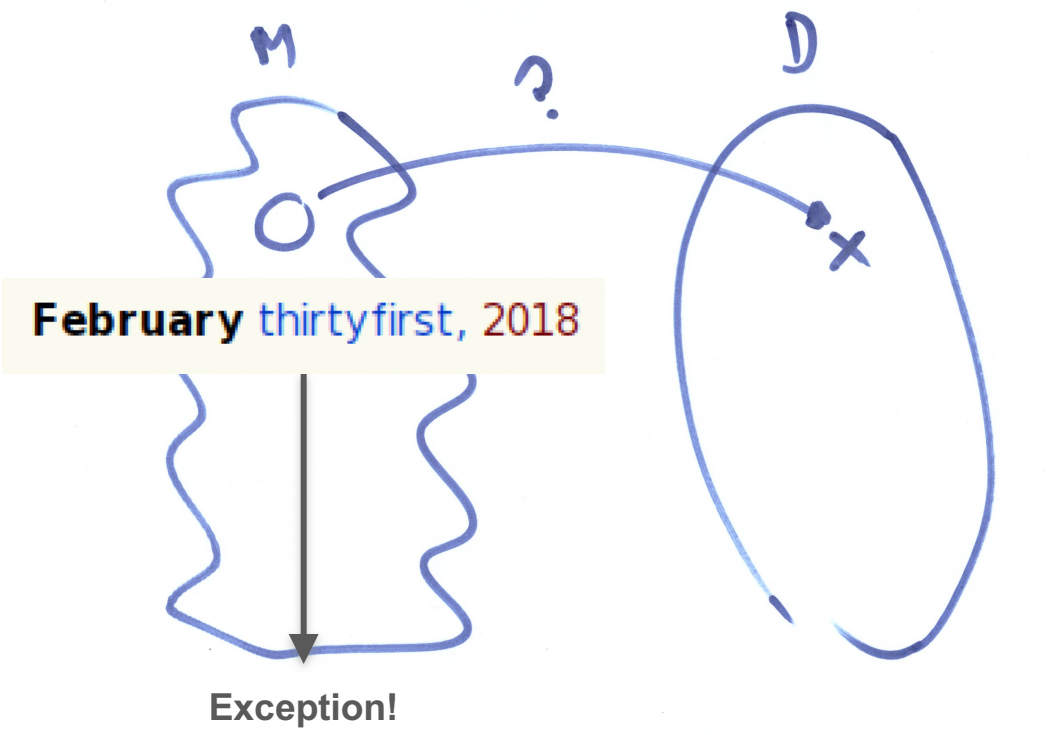



```
LocalDate feb31st2018 = LocalDate.of(2018, Month.FEBRUARY, 31);
```



Exception!

¿Deberíamos poder representar 31 de Feb de 2018? Smalltalk

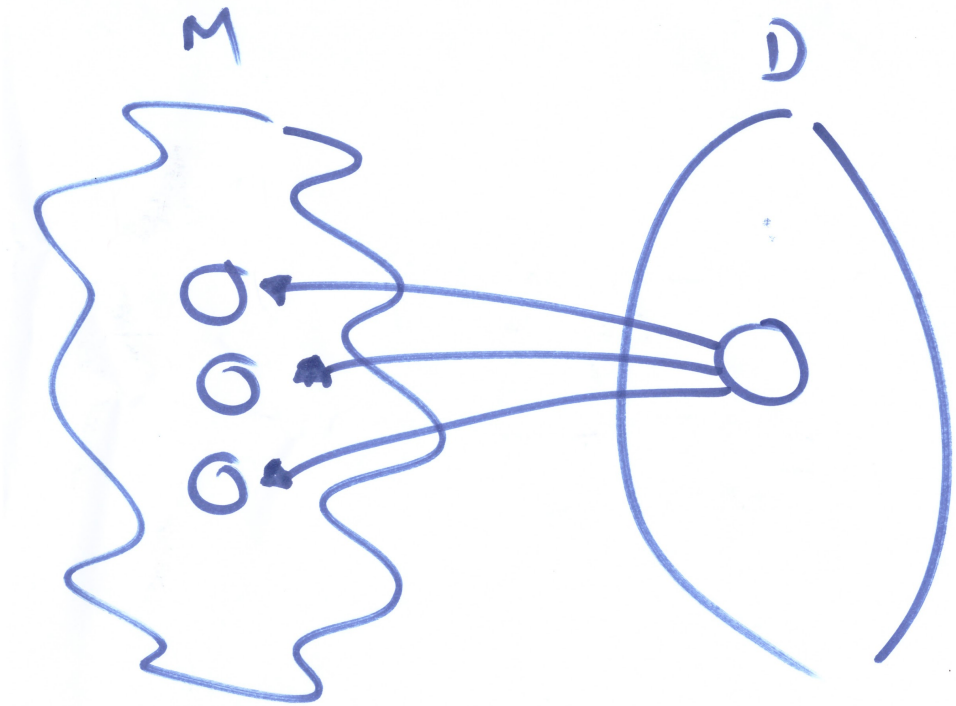


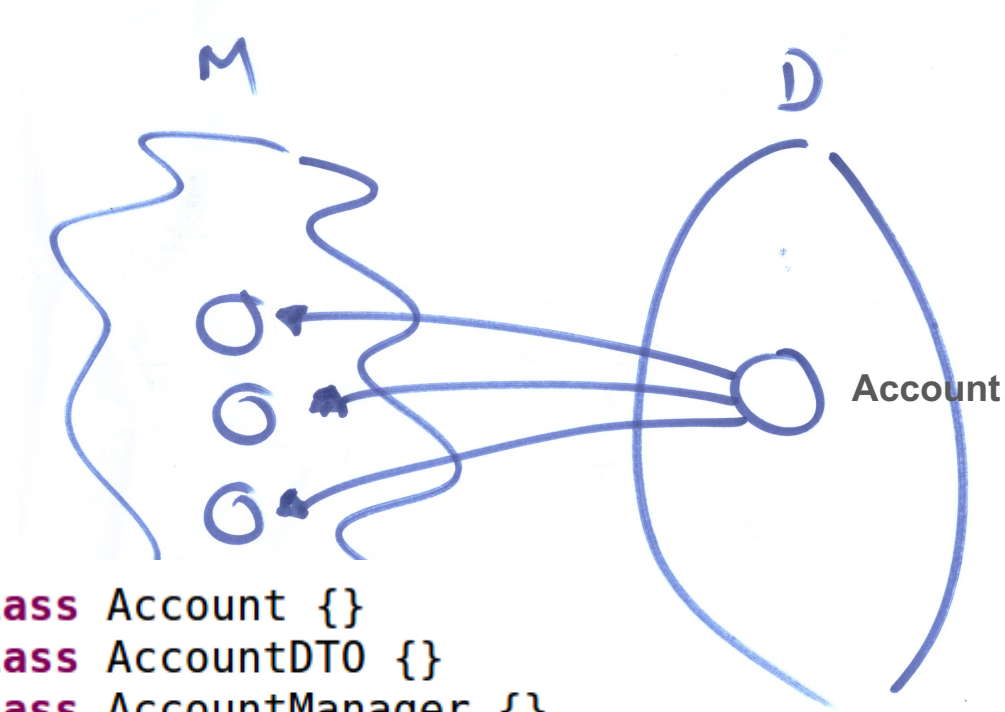
Problemas

- 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?



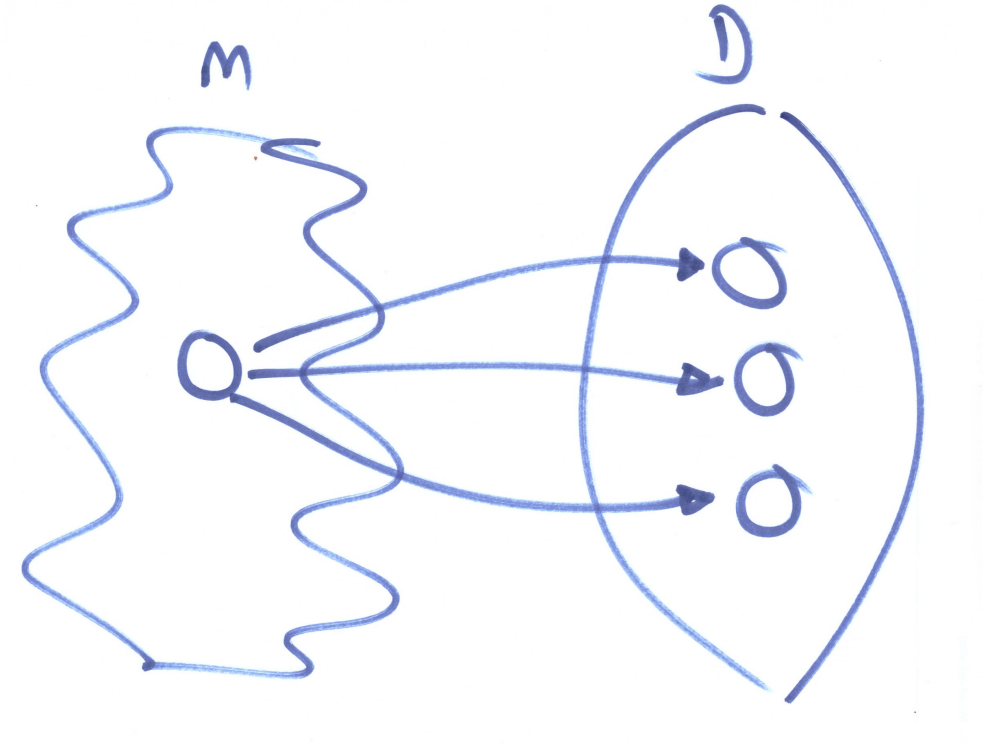



```
class Account {}
class AccountDTO {}
class AccountManager {}
class AccountHelper {}
...
```

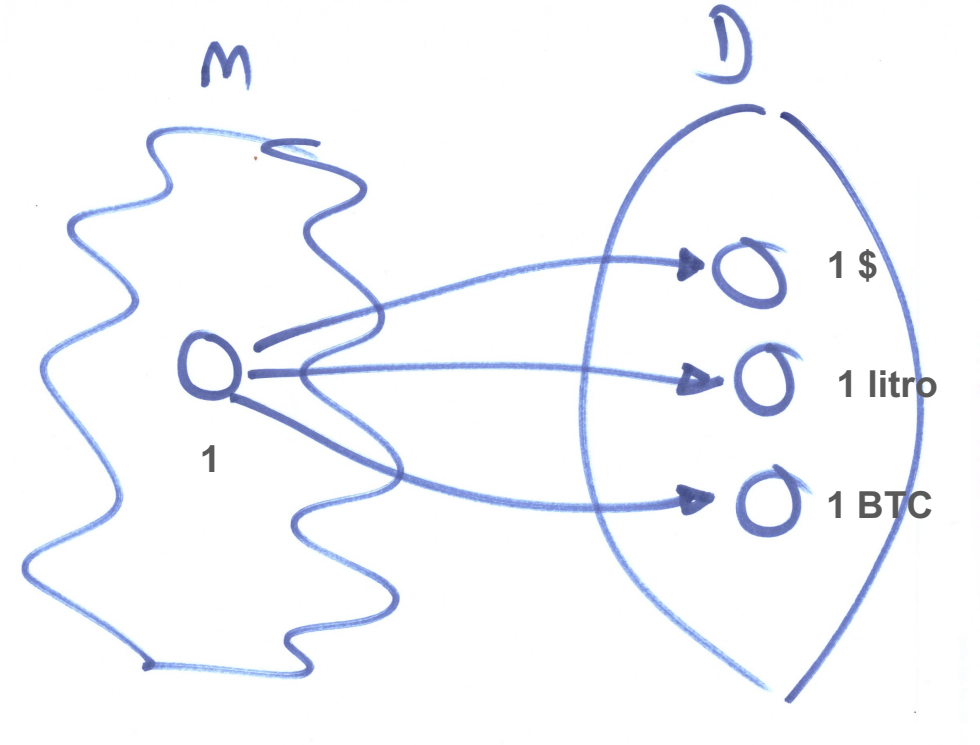
Problemas

- Complejidad!!

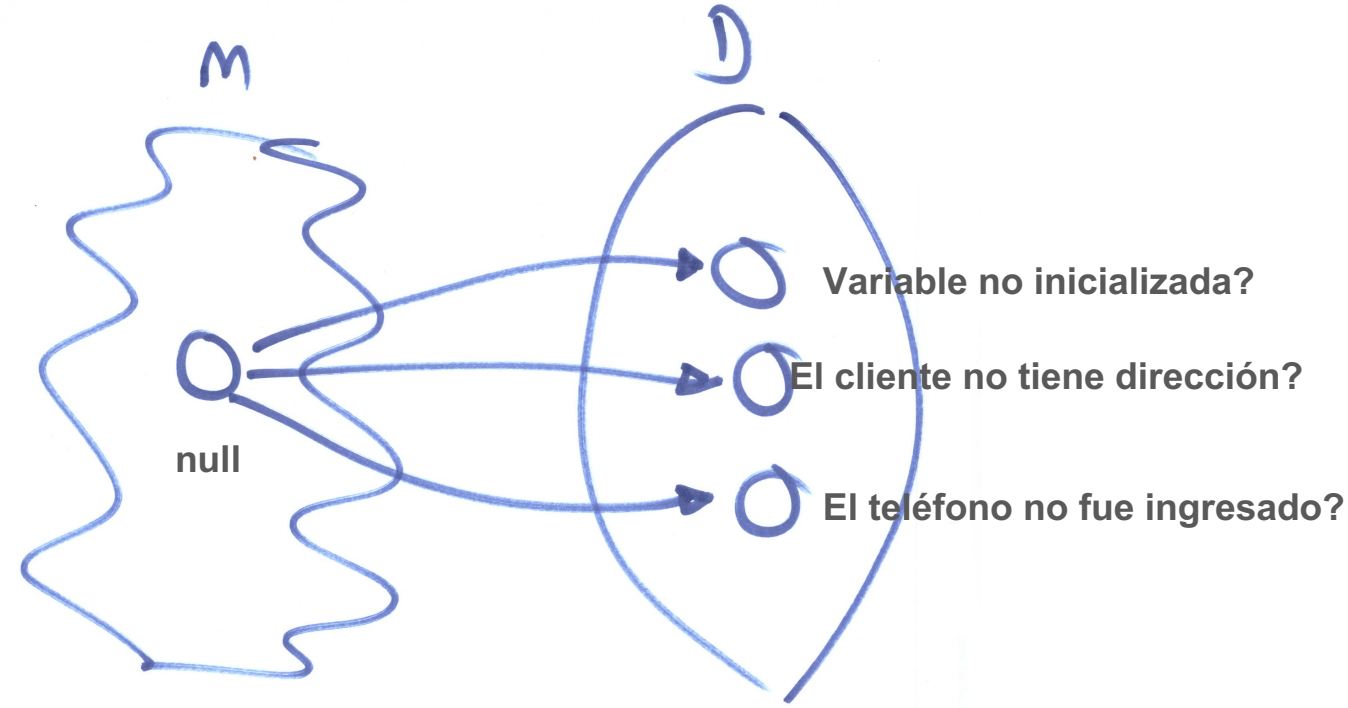
¿Qué pasa con esta relación?



¿Qué representa 1?



¿Qué representa “null”?



Null References: The Billion Dollar Mistake

Recorded at: QCon

Like | by Tony Hoare on Aug 25, 2009 | 5 Discuss

NOTICE: The next QCon is in London Mar 5-9, 2018. Join us!

Share | Reading List | Read later

View Presentation

0:00 / 1:01:58

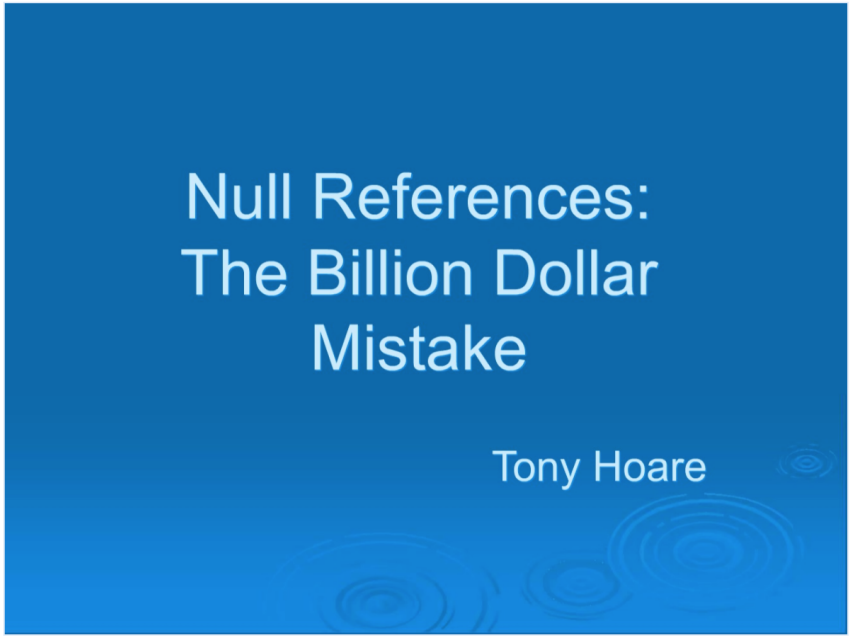
01:01:58

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

- Null references have historically been a bad idea

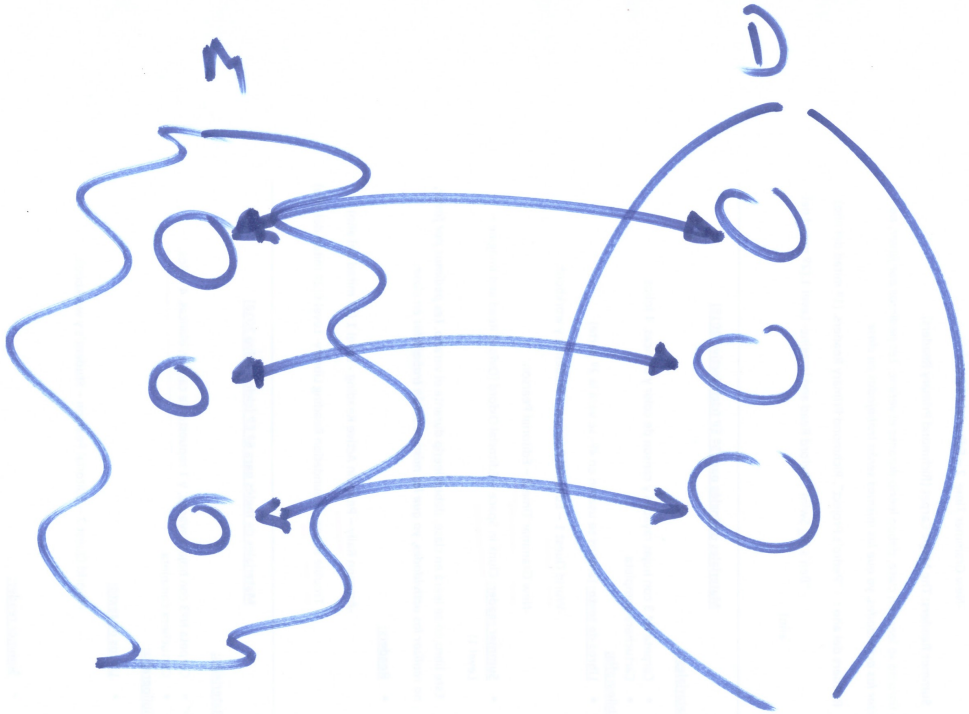
Sponsored Content

Problemas

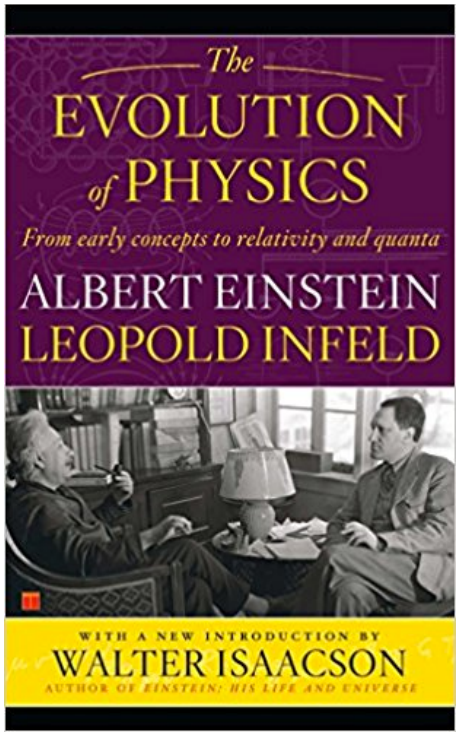
- Pérdida de información
- Errores de implementación
- …

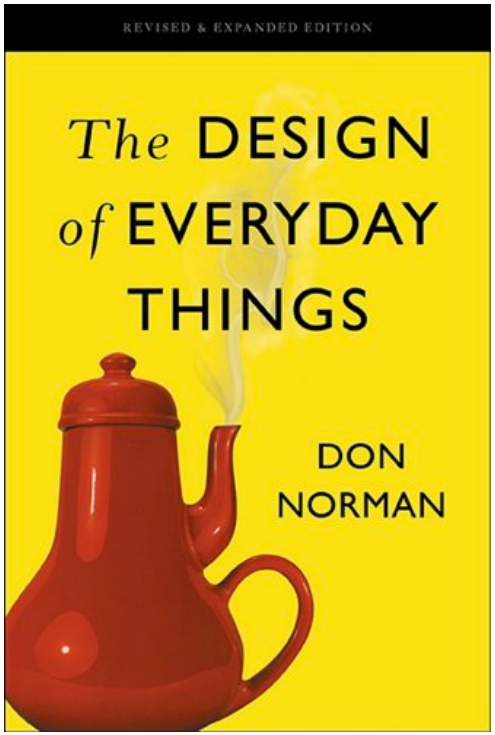
- Design Tip: No usar null/nil

Conclusión



Libros donde se habla de estas cosas





Buen Modelo – Eje Descriptivo

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

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

No Silver Bullet
—Essence and Accident in Software Engineering

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill

> *There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity.*

Características Esenciales

- Complejidad
  - complejidad = esencial + accidental
- Conformidad
- Invisibilidad
- Cambio

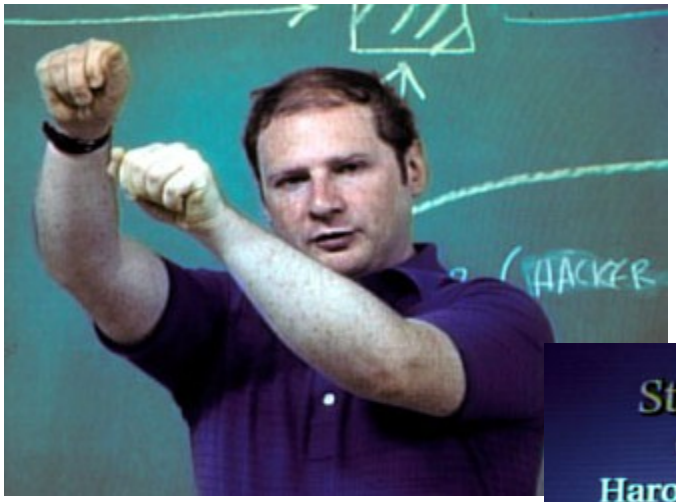
No hay una “Silver Bullet”

- En parte debido a su inmadurez, la Ingeniería de Software tiene una peligrosa tendencia a “seguir modas”
  - Aparece una nueva tendencia, generalmente interesante y con aportes
  - Se genera un gran negocio
  - Los que están detrás del negocio presentan a esa tendencia como la “silver bullet”
  - En algún momento, la realidad le da la razón a Brooks

© Cátedra de Ingeniería de Software I – FCEN – UBA, 2017

¿Qué es el Desarrollo de Software?

¿Es una Ciencia?



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

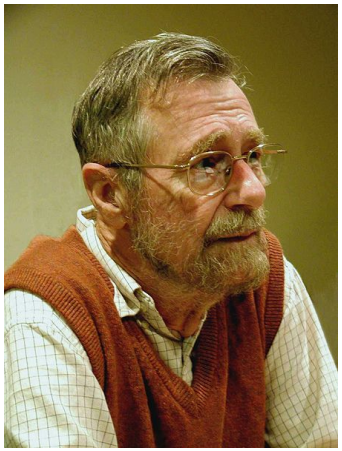
Aportando al a confusión: ¿Existe la *ciencia de la computación*?

- 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 meritoria, *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'.
  - Para llevarse a casa el mensaje que este curso es primariamente un curso en matemáticas formales, vemos que el lenguaje de programación en cuestión no ha sido implementado en la universidad. De esta forma, los estudiantes son protegidos de la tentación de "testear" sus programas.

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

¿Existe la Ingeniería de Software?

- 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".

Fuente: ver http://www.yourdon.com

“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

La Ingeniería…

- Antes de ir a la ingeniería de software pensemos en la definición de ingeniería

- Crear soluciones eficientes
  - A problemas prácticos
    - Aplicando conocimiento científico
      - Construyendo cosas
        - Al servicio de la humanidad

- *La ingeniería permite a la gente común hacer cosas que antes requerían virtuosos*

- *"Ingeniería es ciencia con un propósito"*

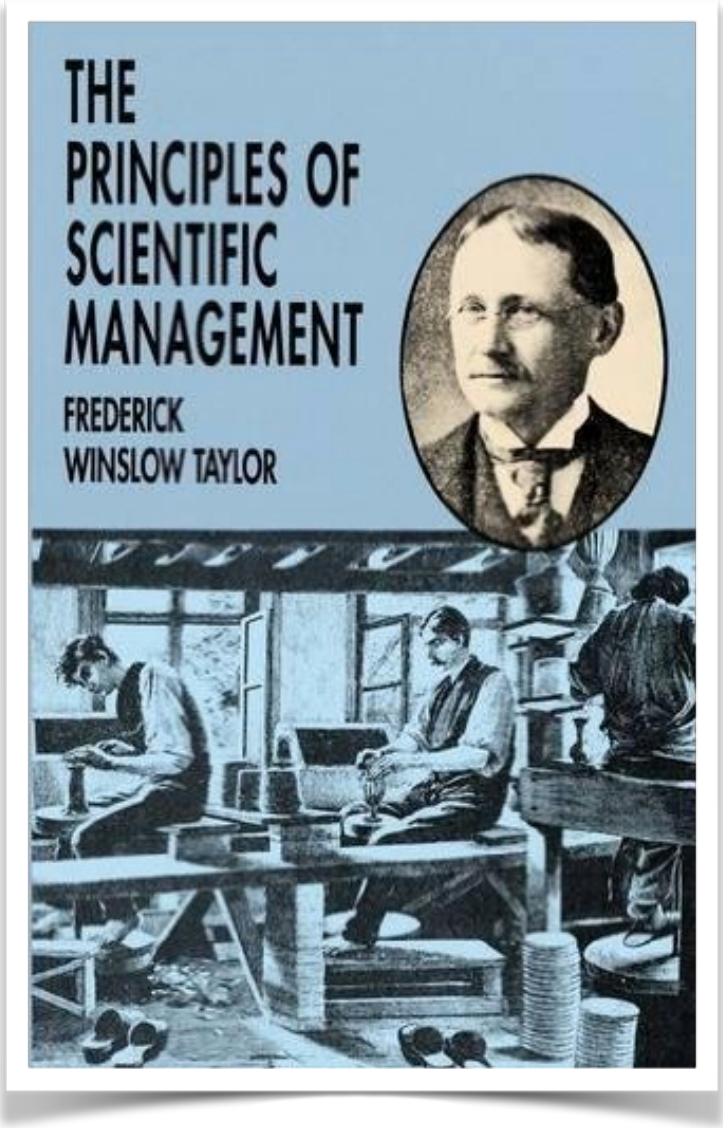
Fuente: Mary Shaw – Prospects for an Engineering Discipline of Software. IEEE Software. Noviembre 1990.

Definición de Ingeniería de Software

- IEEE: “La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, mantenimiento y operación de sistemas de software”

- Otra definición interesante:
  - Bertrand Meyer: “The development of possibly large systems intended for use in production environments, over a possibly long period, worked on by possibly many people, and possibly undergoing many changes, where development includes management, maintenance, validation, documentation, and so forth.”

Fuentes: IEEE Glossary for Software Engineering Terminology.
Bertrand Meyer, Software Engineering in the Academy, IEEE Computer, Mayo 2001.


THE
PRINCIPLES OF
SCIENTIFIC
MANAGEMENT
FREDERICK
WINSLOW TAYLOR

Taylor

Trabajo Manual

MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS

Dr. Winston W. Royce

INTRODUCTION

I am going to describe my personal views about managing large software developments. I have had various assignments during the past nine years, mostly concerned with the development of software packages for spacecraft mission planning, commanding and post-flight analysis. In these assignments I have experienced different degrees of success with respect to arriving at an operational state, on-time, and within costs. I have become prejudiced by my experiences and I am going to relate some of these prejudices in this presentation.

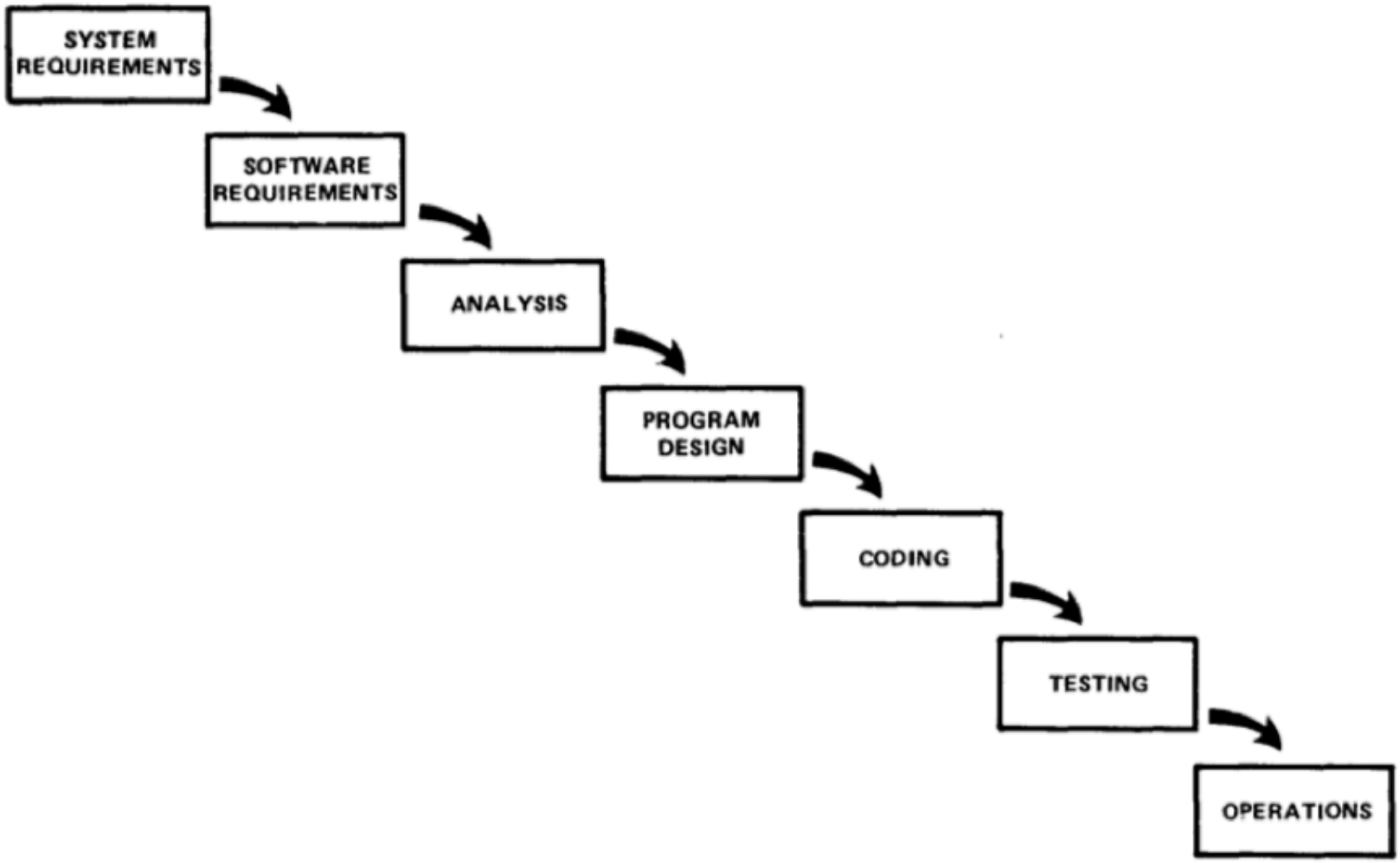


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

SOFTWARE ENGINEERING

Report on a conference sponsored by the

NATO SCIENCE COMMITTEE

Garmisch, Germany, 7th to 11th October 1968

Chairman: Professor Dr. F. L. Bauer

Co-chairmen: Professor L. Bolliet, Dr. H. J. Helms

Editors: Peter Naur and Brian Randell

January 1969

Taylorismo + Ingeniería

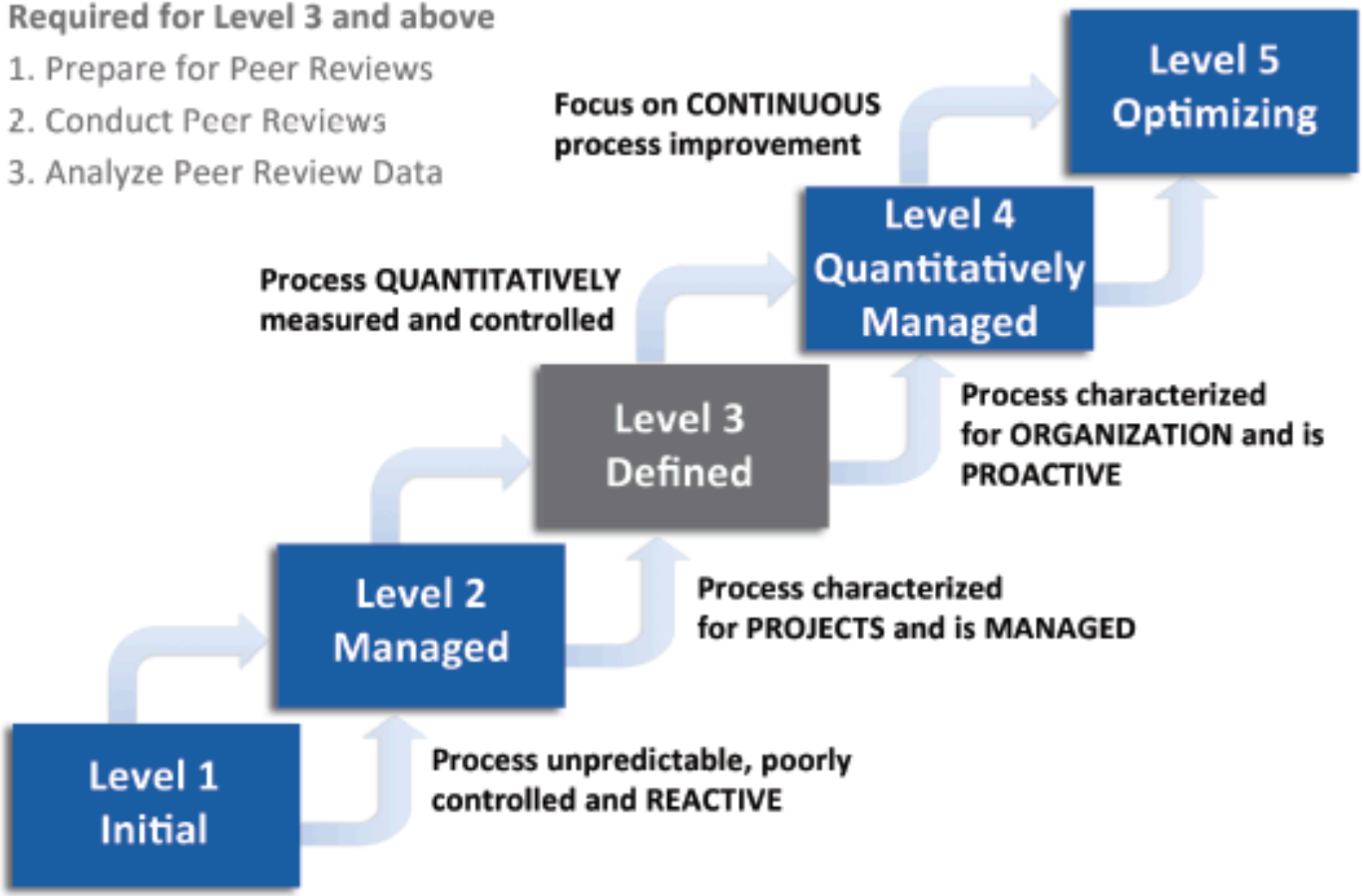
Fordismo

Rotación = 380%
(963 para quedarse con 100)

CMMI Staged Maturity Levels

Peer Reviews Activities
Required for Level 3 and above

1. Prepare for Peer Reviews
2. Conduct Peer Reviews
3. Analyze Peer Review Data



Software Requirements
Software Architecture
Software Tests & Reports
Software Release

Pero nuestro trabajo es…

Trabajo Creativo

MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS

Dr. Winston W. Royce

INTRODUCTION

I am going to describe my personal views about managing large software developments. I have had various assignments during the past nine years, mostly concerned with the development of software packages for spacecraft mission planning, commanding and post-flight analysis. In these assignments I have experienced different degrees of success with respect to arriving at an operational state, on-time, and within costs. I have become prejudiced by my experiences and I am going to relate some of these prejudices in this presentation.

!!

I believe in this concept, but the implementation described above is risky and invites failure. The problem is illustrated in Figure 4. The testing phase which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from analyzed. These phenomena are not precisely analyzable. They are not the solutions to the standard partial differential equations of mathematical physics for instance. Yet if these phenomena fail to satisfy the various external constraints, then invariably a major redesign is required. A simple octal patch or redo of some isolated code will not fix these kinds of difficulties. The required design changes are likely to be so disruptive that the software requirements upon which the design is based and which provides the rationale for everything are violated. Either the requirements must be modified, or a substantial change in the design is required. In effect the development process has returned to the origin and one can expect up to a 100-percent overrun in schedule and/or costs.

!!

I believe in this concept, but the implementation described above is risky and invites failure. The problem is illustrated in Figure 4. The testing phase which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from analyzed. These phenomena are not precisely analyzable. They are not the solutions to the standard partial differential equations of mathematical physics for instance. Yet if these phenomena fail to satisfy the various external constraints, then invariably a major redesign is required. A simple octal patch or redo of some isolated code will not fix these kinds of difficulties. The required design changes are likely to be so disruptive that the software requirements upon which the design is based and which provides the rationale for everything are violated. Either the requirements must be modified, or a substantial change in the design is required. In effect the development process has returned to the origin and one can expect up to a 100-percent overrun in schedule and/or costs.

!!

I believe in this concept, but the implementation described above is risky and invites failure. The problem is illustrated in Figure 4. The testing phase which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from analyzed. These phenomena are not precisely analyzable. They are not the solutions to the standard partial differential equations of mathematical physics for instance. Yet if these phenomena fail to satisfy the various external constraints, then invariably a major redesign is required. A simple octal patch or redo of some isolated code will not fix these kinds of difficulties. The required design changes are likely to be so disruptive that the software requirements upon which the design is based and which provides the rationale for everything are violated. Either the requirements must be modified, or a substantial change in the design is required. In effect the development process has returned to the origin and one can expect up to a 100-percent overrun in schedule and/or costs.

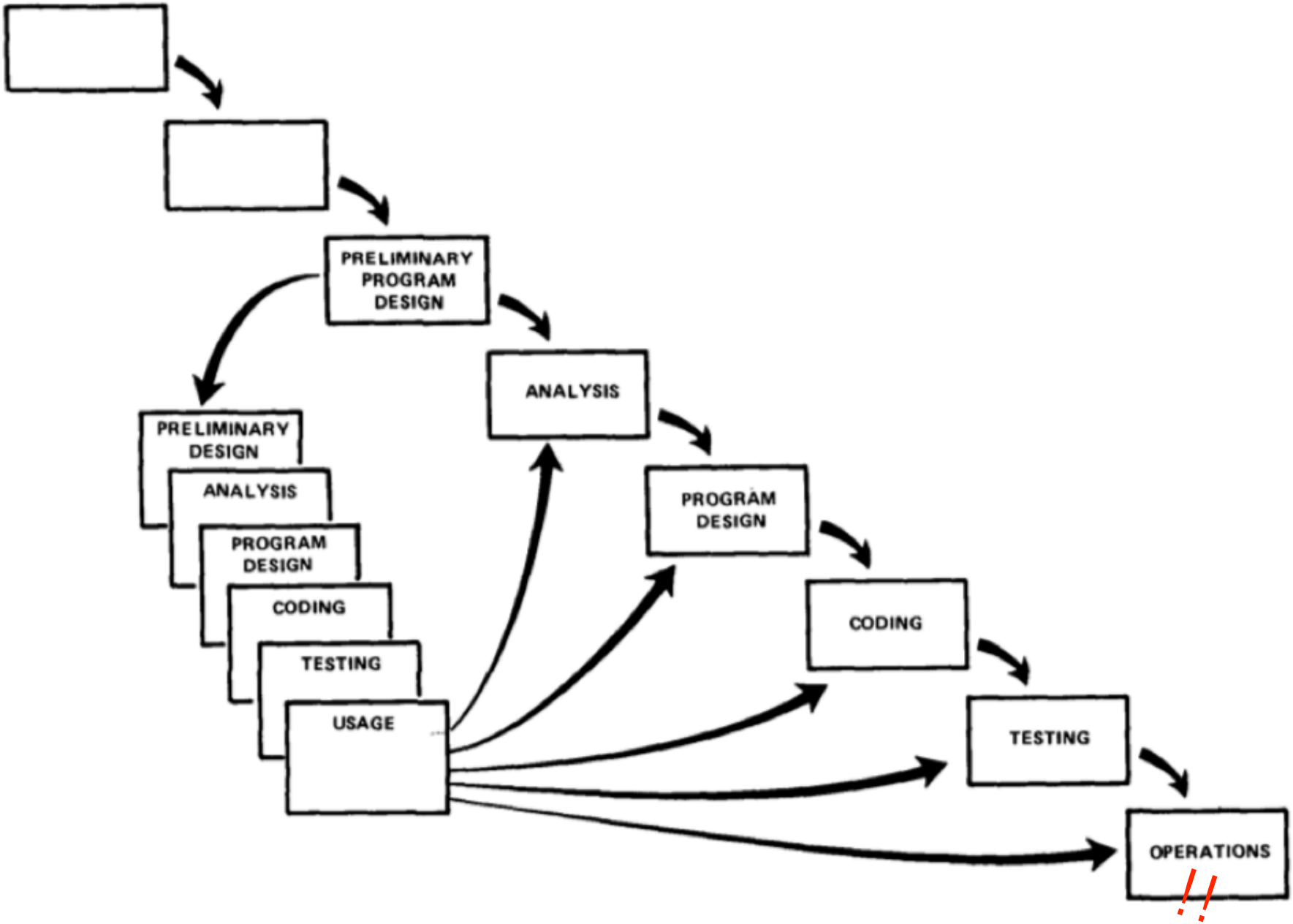


Figure 7. Step 3: Attempt to do the job twice — the first result provides an early simulation of the final product.

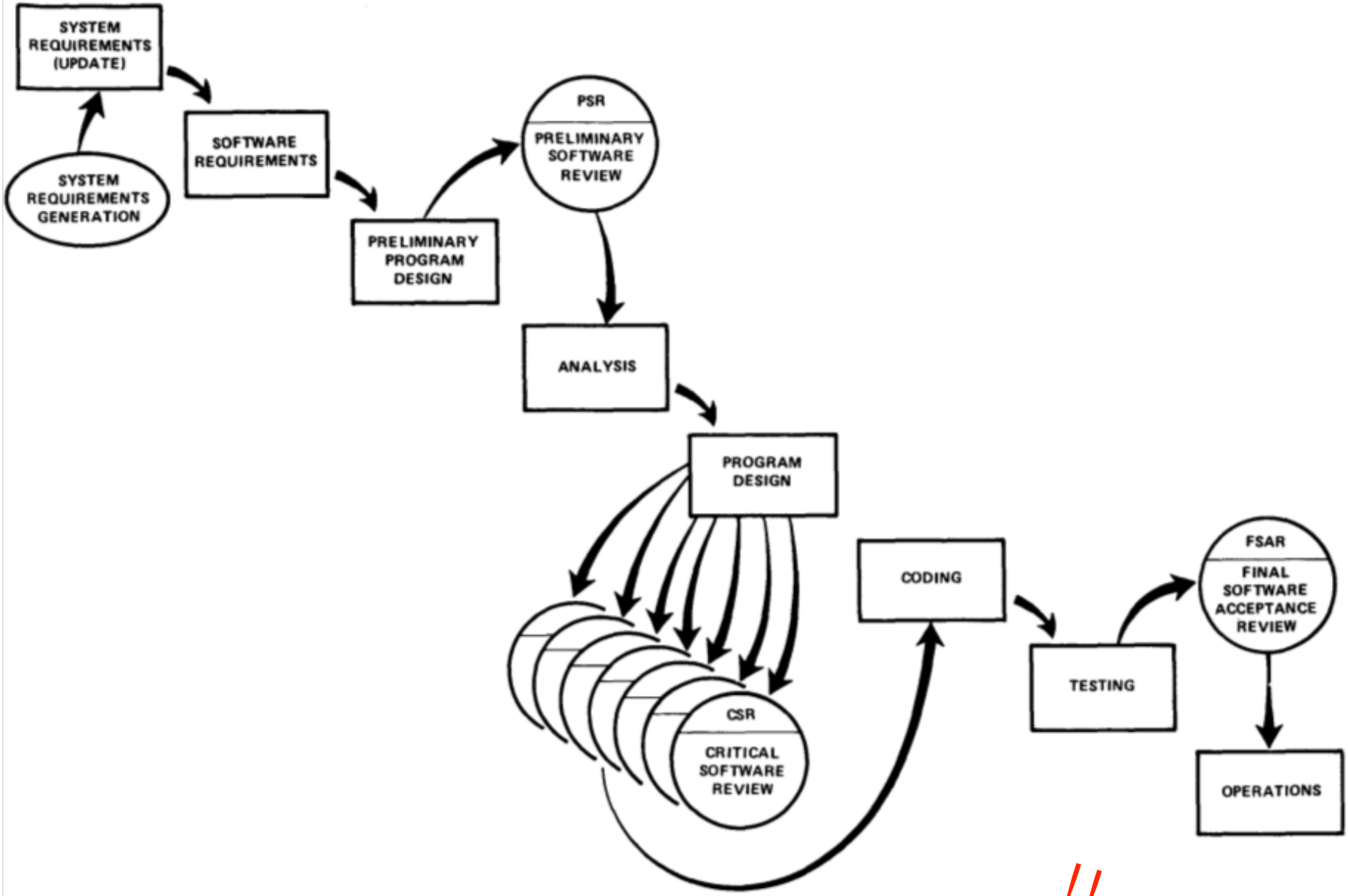
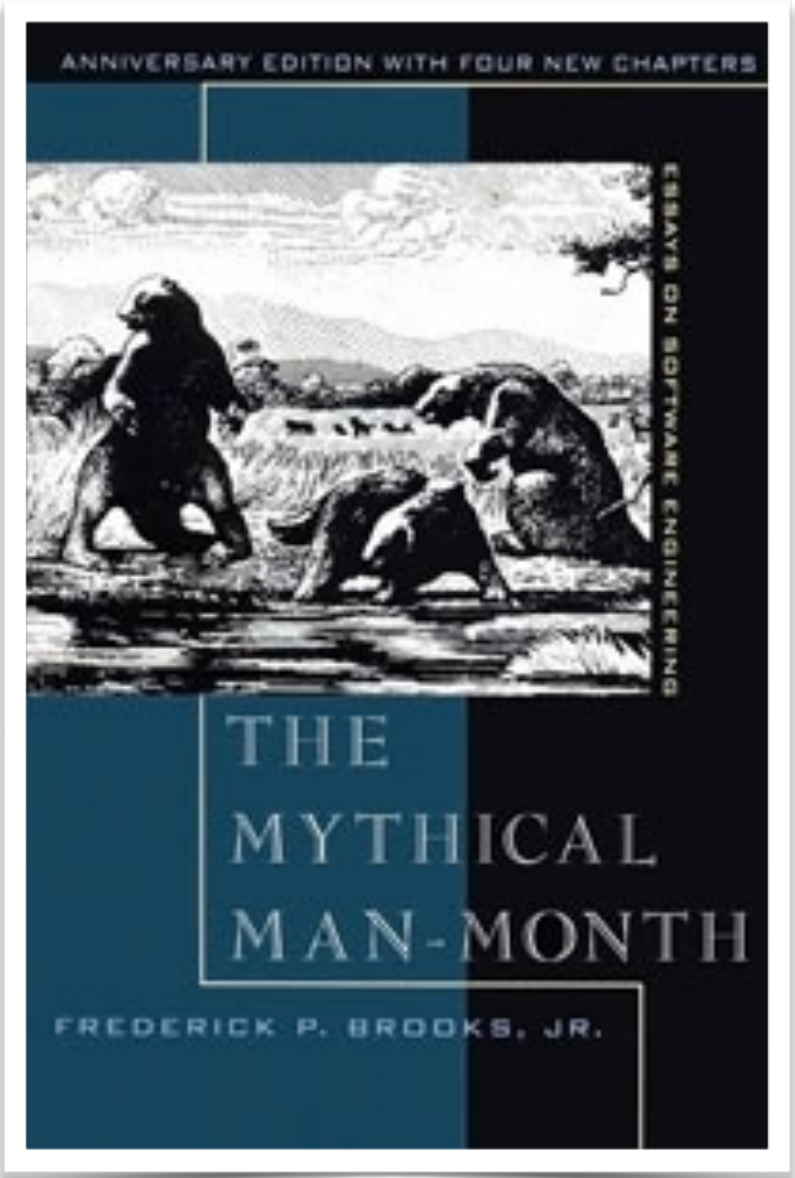


Figure 9. Step 5: Involve the customer — the involvement should be formal, in-depth, and continuing.

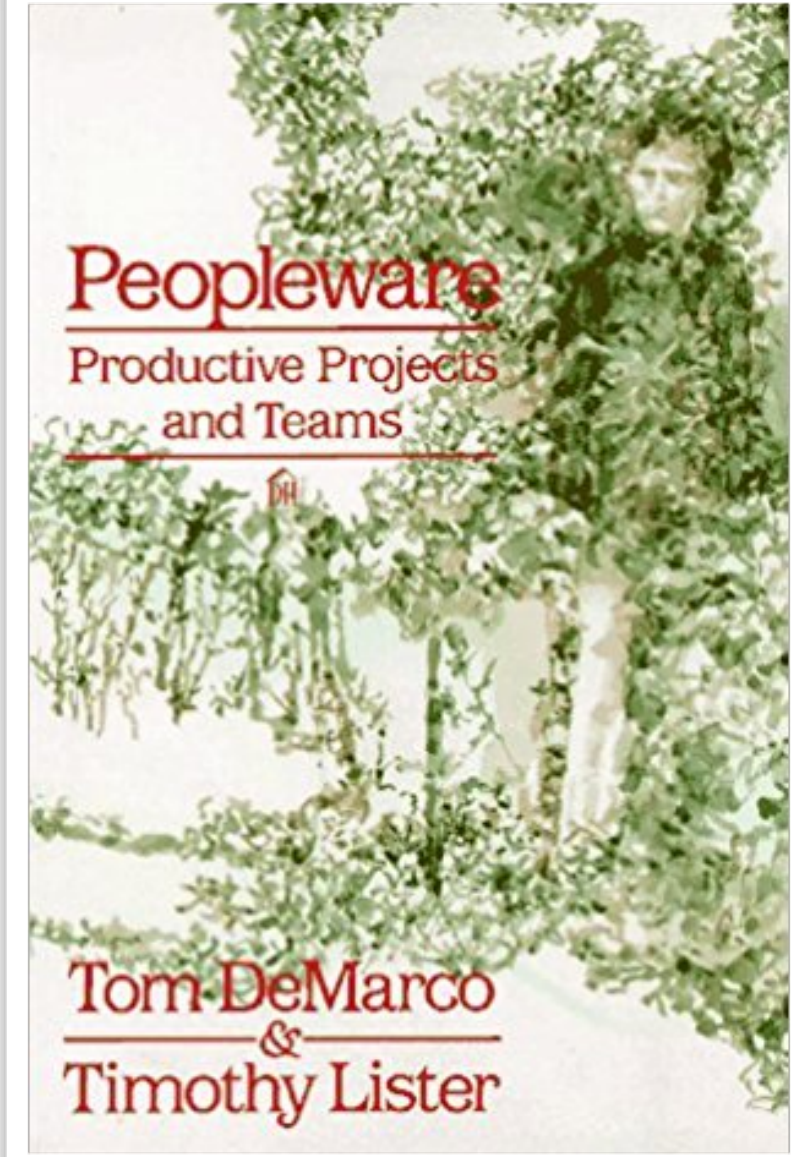

ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS
ESSAYS ON SOFTWARE ENGINEERING
THE MYTHICAL MAN-MONTH
FREDERICK P. BROOKS, JR.

misestimated. If on March 1 one makes the conservative assumption that the whole schedule was optimistic, as Fig. 2.7 depicts, one wants to add 6 men just to the original task. Calculation of the training, repartitioning, system testing effects is left as an exercise for the reader. Without a doubt, the regenerative disaster will yield a poorer product, later, than would rescheduling with the original three men, unaugmented.

Oversimplifying outrageously, we state Brooks's Law:

Adding manpower to a late software project makes it later.

This then is the demythologizing of the man-month. The number of months of a project depends upon its sequential con-


Peopleware
Productive Projects
and Teams
Tom DeMarco
&
Timothy Lister

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

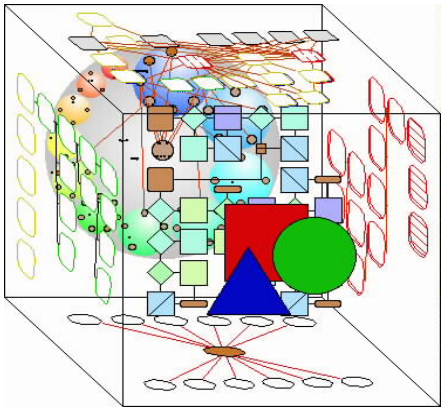
That is, while there is value in the items on
the right, we value the items on the left more.

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

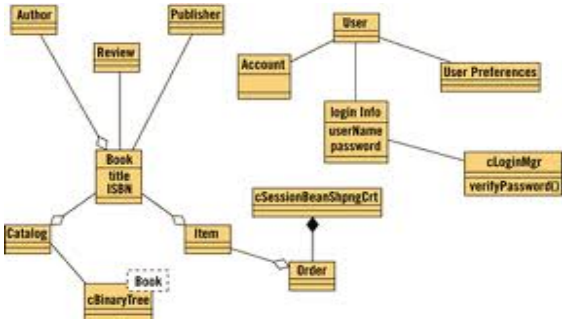
Proceso de Desarrollo de Software

- 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

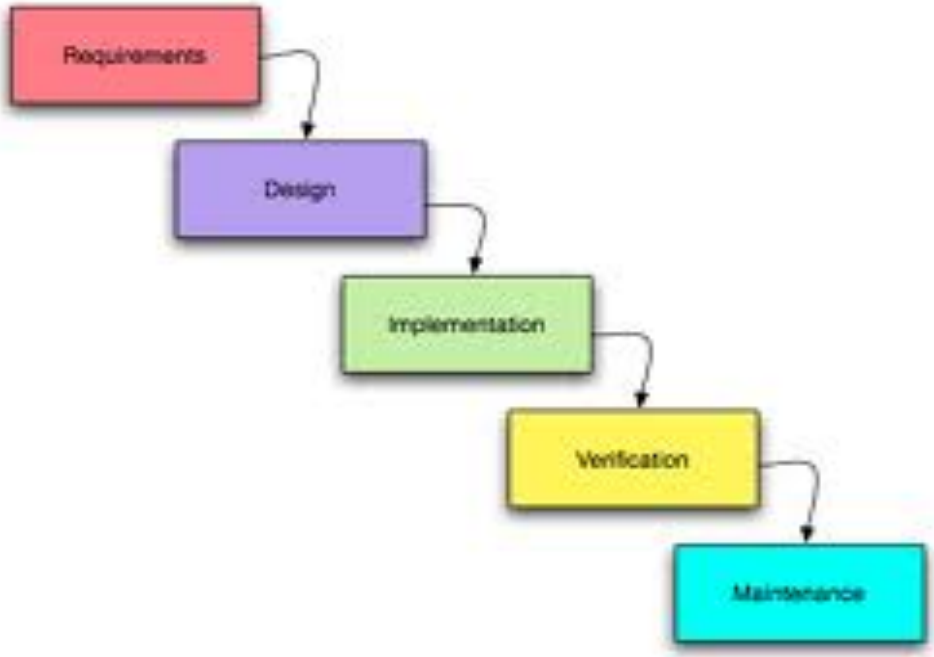
Feedback Inmediato

- Bret Victor: Inventing on principle
  - http://vimeo.com/36579366

Visión Clásica

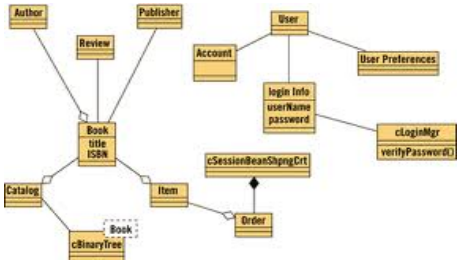


Visión Clásica



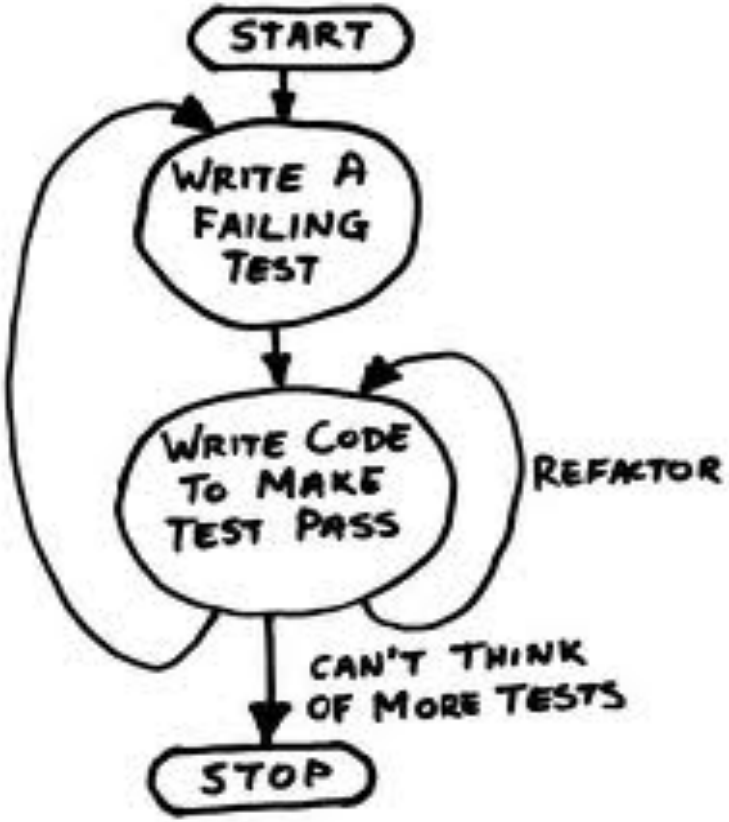
Visión Moderna

Corporate Strategy Analysis

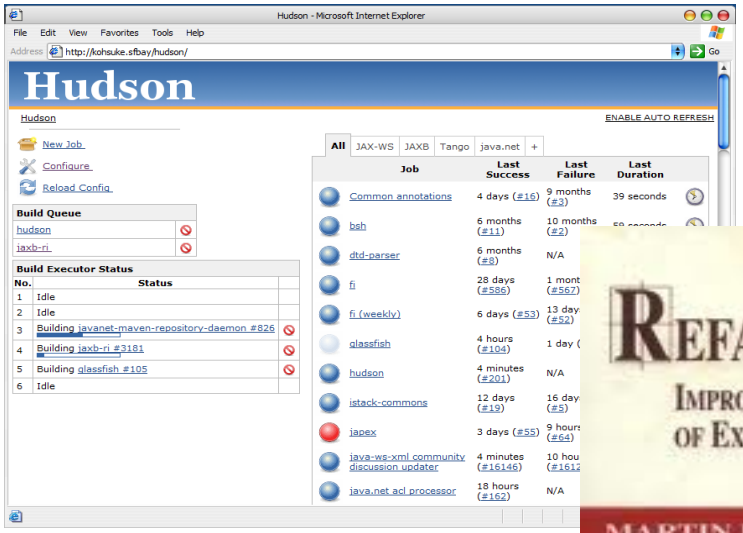


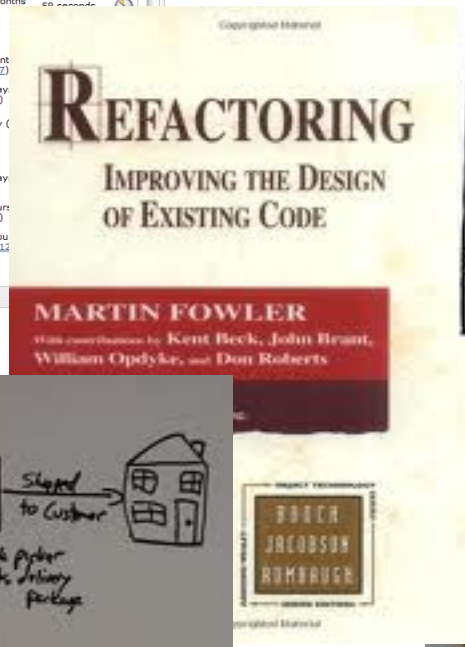
Técnicas y Herrameientas adecuadas

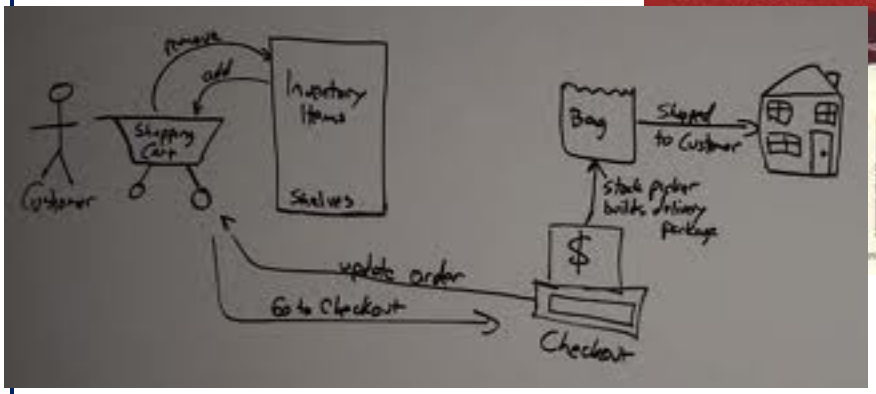
Técnicas - TDD



Otras Técnicas







Por lo tanto…

Programa

- Introducción a la materia y la ingeniería de software
- Programación y Diseño con Objetos
  - Paradigma de Objetos Fundacional
  - Diseño con Objetos: Polimorfimos/Closures/Reificacion/Abstracción/etc
  - Técnicas de diseño: Idioms/Patrones/Frameworks
  - Refactorings automatizados
- TDD – Testing
- Arquitectura
- Metodologías de desarrollo

Espíritu de la Materia

- Aprender haciendo, no escuchando
- Es una materia exigente pero no difícil
- Es una materia que requiere dedicación
- Es importante la participación en clase, debatir, preguntar
- Es importantisimo hacer todos los ejercicios y aprovechar que los hacemos en labo durante todo el cuatrimestre y no al final encerrados un fin de semana y corriendo!

Importante

- Cursada:
  - Lunes y Jueves de 17 a 22 hrs
  - Labo 4

- Comunicación:
  - Página Web: www.isw2.com.ar (tiene mucha información, links a papers, videos, ppts, etc)
  - Lista de Alumnos: ingsoft1-alu@dc.uba.ar
  - Lista de Docentes: ingsoft1-doc@dc.uba.ar
  - La lista de alumnos se debe usar para consultas de los ejercicios, dudas, etc.
  - La lista de docentes para temas personales, no para dudas de ejercicios
  - Mails a la lista de docentes sobre consultas de ejercicios serán forwardeados a la de alumnos para que se compartan entre todos

¿Cómo está organizada la materia?

- Clases:
  - Prácticas en labo
    - Son todos ejercicios de desarrollo de software
    - Todos los ejercicios se hacen en clase y tienen tiempo para terminarlos en casa si no llegan en clase
    - ¡Es importante que usen el tiempo del labo para preguntar!
  - La teoría la deduciremos a partir de la práctica
  - Papers
    - Bajamos la cantidad de paper obligatorios
    - Son en su mayoría papers “históricos”, hay que leerlos en el contexto en el que fueron escritos
    - ¡Hay que leerlos!
    - Les aconsejamos leer los papers opcionales y ver los videos opcionales si tienen tiempo

¿Cómo está organizada la materia?

- Evaluación:
  - Aprobación de Cursada:
    - Haber aprobado los dos parciales (o sus recus)
  - Parciales:
    - Solo prácticos, en máquina, INDIVIDUALES
    - Se aprueban con 4 o más
    - Solo se recupera si no se aprueba
  - Recuperatorios:
    - 2do parcial sirve como recuperatorio del 1er parcial si la nota >= 7
    - Último recuperatorio:
      - Para los que no aprobaron el 1er o 2do parcial
      - Esto incluye que los que tenían que recuperar el 1er parcial y no lo lograron en el 2do parcial

¿Cómo está organizada la materia?

- Promoción:
  - Se puede promocionar la materia si se aprobó la cursada
  - Se promociona si la “nota de promoción” da >= 7
  - Nota promoción = Ejer*0.4+Parc1*0.3+Parc2*0.3+PtoAdic
  - Ejercicios = Promedio(Suma(Cada Ejercicio))
  - Nota de Cada ejercicio: De 0 a 10
  - Punto Adicional:
    - Si realizan un mejora a CuisUniversity y es aceptada por la comunidad
    - Si resuelven los challenges que se proponen para este fin
  - Firma de libreta si promocionaron:
    - Si tienen todas las correlativas firmadas → se firma libreta
    - Si no tienen todas las correlativas firmadas → se guarda la nota por el mismo tiempo que la aprobación de los prácticos (4 años)

Estadísticas de aprobación

- Aprobación de cursada: ~90% (último cuat. 98%)
- Promedio de aprobación de cursada sin promoción: 5.8

- Promoción: ~75%
- Promedio de Promoción: 8

Ejercicios - Grupos

- Grupos de 2 (pares).
- Solo un grupo puede ser de 3 (si hay cant. impar de alumnos)
- Los grupos son “inmutables”
- El objetivo es que trabajen de a pares, no que uno haga un ejercicio y otro haga otra ejercicio y así sucesivamente
- Traten de hacer “pair programming” en serio, es la mejor manera de aprender y generar los mejores resultados
- Deben informar el grupo enviando mail a ingsoft1-doc@dc.uba.ar con:
  - Nombre y LU de los integrantes
  - Link al Repositorio donde dejarán los ejercicios

Ejercicios

- Los ejercicios están compuestos por:
  - Un componente de desarrollo de software
  - Un conjunto de preguntas teóricas relacionadas con lo visto en clase y el paper relacionado con el ejercicio
- Los ejercicios se realizan en máquina (propia o del labo)
- Lenguaje de Programación:
  - Smalltalk
  - Ambiente: CuisUniversity (Smalltalk Argento para enseñar)
  - http://cuisuniversity.org
- Repo de la materia para bajar los ejercicios:
  - https://github.com/UBA-FCEN-IngSoft1/2019-2c

Ejercicios

- Solución de los ejercicios:
  - Se ve la solución de TODOS los ejercicios en clase
  - La cátedra provee una solución de software de TODOS los ejercicios para que puedan compararla con su solución
- Entrega de ejercicios:
  - Deben ser subidos al repo del grupo antes del inicio de la clase donde se verá la solución del ejercicio
  - Recuerden: cada grupo debe informar el repo que usarán
  - El repo debe seguir la estructura del repo de la materia
  - Aconsejamos entregar TODOS los ejercicios por más que no los hayan terminado
  - Solo se corrige y se pone nota a 5 ejercicios
  - No se avisa de antemano qué ejercicios serán corregidos

Ejercicios

- Qué se evaluará por ejercicios:
  - Diseño & Implementación
    - Nombre de objetos, clases, mensajes, etc.
    - Qué tan declarativa es la solución
    - Cantidad de colaboraciones por método
    - Distribución de responsabilidades
    - Mapeo de los objetos con el dominio de problema
    - etc
  - Que pasen los tests
  - Cuando tengan que escribir los tests:
    - Qué tests escribieron
    - Cómo los nombraron
    - Cuántos casos tuvieron en cuenta
    - Qué tan bien hicieron un desarrollo iterativo/incremental
    - etc

Preguntas

- ¿Por qué los ejercicios tiene preguntas teóricas?
  - Para no incluir las preguntas teóricas en el parcial y haya más tiempo para el ejercicio práctico
  - Para que tengan tiempo real para poder responderlas, puedan leer lo que es necesario, charlarlas, etc.
- ¿Por qué Smalltalk?
  - Por ser un lenguaje con muy baja complejidad accidental
  - Por ser un lenguaje meta-circular, fundamental para un contexto de carrera como la nuestra
  - Por permitir realizar Live Coding
- ¿Por qué CuisUniversity?
  - Por ser Cuis un Smalltalk Argentino, creado por un alumno de la facultad, simple y chico
  - Por ser CuisUniversity una distribución especial para lo que enseñamos
  - Por tener un ambiente de programación de objetos prototípicos

Ejercicios

- ¿Por qué conviene entregar todos los ejericios?
  - Para estar mejor preparado para el parcial
  - Para usar la nota, si es corregido, en la posibilidad de promocionar
- ¿Se pueden recuperar los ejercicios?
  - No, los ejercicios no se recuperan ya que no forman parte de la nota para aprobar la cursada
  - La nota de los ejercicios solo se usa para facilitar la promoción, por lo tanto no es obligatoria su entrega
- ¿Por qué no se avisa de antemano qué ejercicios serán corregidos?
  - Porque lamentablemente si avisamos que no será corregido, la mayoría no lo hace y eso afecta indirectamente la cursada
- ¿Qué nota tengo si un ejercicio se corrige y no lo entregué?
  - En ese caso la nota será 0, por ese motivo aconsejamos entregar todos los ejercicios aunque la solución no esté completa o no sea la que más les guste

Ejercicios

- ¿Qué hago si no estoy conforme con una corrección?
  - Pedir revisión con el JTP o profesor
  - Si corresponde la corrección subirá la nota, nunca la bajará
- Tengo un compañero que hizo lo mismo que yo pero tiene mejor nota, ¿qué hago?
  - No hay ningún problema en pedir la revisión mostrando la otra corrección
  - La correción no afectará en nada la nota del otro alumno aún si está mal su correción
- Comentario sobre las correcciones:
  - Son difíciles de hacer, llevan tiempo
  - No son “matemáticas” ni formales
  - Tenemos un conjunto de items que corregimos para que sean lo más consistente posible
  - Si nos equivocamos no es mala intención sino simplemente un error

¿Preguntas?