Ingeniería de Software I

Introducción

Objetivos del Curso

- Entender el rol fundamental que juega la construcción y análisis de modelos en la Ingeniería de Software
  - Aprender técnicas de descripción, su semántica y su uso como herramientas de análisis.
  - Poder aplicarlas en distintas actividades de IS.
- Entender la problemática, las notaciones, métodos, procesos y herramientas para encarar tres áreas importantes de la ingeniería de software
  - Ingeniería de requerimientos
  - Verificación y Validación. Principalmente Testing

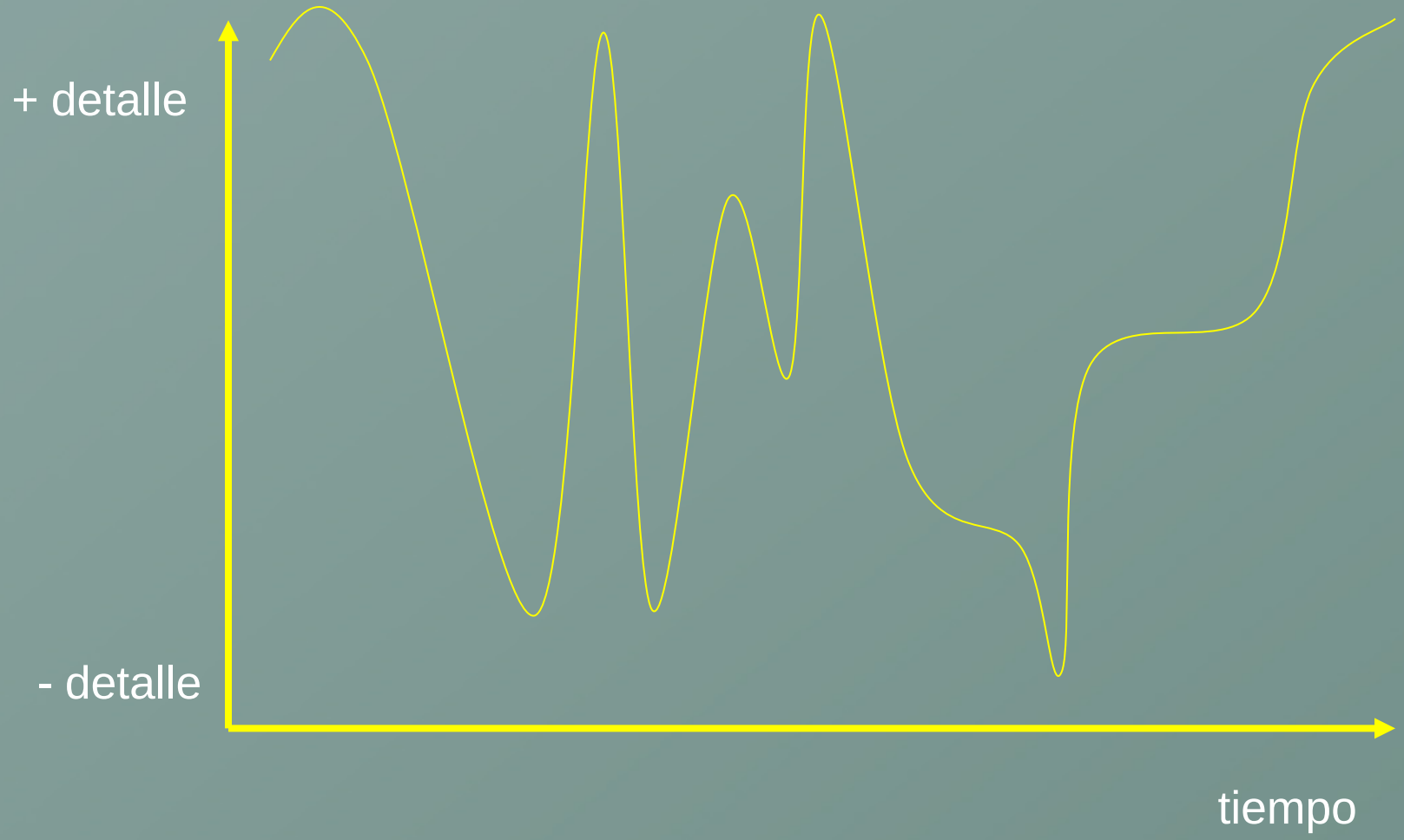
Organización de la Materia

- Teóricas
  - Lunes y Jueves de 17:00 a 19:00
- Prácticas y Taller
  - Lunes y jueves de 19:00 a 22:00
- Aulas
  - Lunes: Dónde estamos hoy…
  - Jueves: Dónde estaremos el jueves…

Estructura (tentativa) de Teóricas

1. Introducción a la Ingeniería de Software
2. Introducción a la Ingeniería de Requerimientos
3. Técnicas de Descripción aplicadas a Requerimientos
4. Diseño de Software (sólo teóricas)
5. Introducción a la Validación y Verificación
6. Testing
7. Conclusiones

Nivel del Detalle del Curso



Reseña histórica

1. Inicialmente predominaba software de cómputo para aplicaciones científicas o de ingeniería
2. Luego crece el
   - Espectro y escala de aplicaciones
   - Vida útil del software
   - Número y perfiles de “programadores”
3. Aparece la noción de crisis del software y la necesidad de un enfoque sistemático a la construcción: Ingeniería de Software.
   - Ambos términos son usados por primera vez en una reunión de la OTAN en 1968, Garmish.

Reseña histórica

4 Dijkstra describe el panorama en los 70 muy bien:

> The major cause of the software crisis is that the machines have become several orders of magnitude more powerful. To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.–
>
> Edsger Dijkstra, The Humble Programmer, Communications of the ACM 15(1972)

5 Primer International Conference on Software Engineering: 1975

Impacto en la Práctica

http://www.sigsoft.org/impact/

- Principios fundamentales de análisis y diseño
  - Information hiding, Jackson's World & Machine...
- Lenguajes de programación
  - Manejo de excepciones, aserciones, ...
- Software Configuration Management
  - Version control, product models ,change control,composition/selection , build management , workspace management
- Técnicas y herramientas
  - Inspecciones, Revisiones y Walkthroughs...
  - Verificadores de código...
- Middleware
  - web services, application servers, distributed object systems, message queues, remote procedure call systems
- Métodos y Procesos
  - QA, Ciclos de Vida, Modelos de madurez, gerenciamiento...

¿Qué es la *Ingeniería de Software*?

- Una definición aproximada y simplificada...
  - Se ocupa de construir un producto de software de ***buena calidad*** lidiando con las múltiples restricciones (tiempo, presupuesto, y otros...)
- Sus dos problemáticas fundamentales son:
  - Lidiar con la ***escala y complejidad*** de sistemas de software.
  - Identificar que significa ***buena calidad*** y luego lograrla
- Requiere (como todas las ingenierías)
  - ***Rigor, creatividad, documentación y gestión***.

Pero… entonces…

- ¿De donde salen las pre-post condiciones?
- ¿Cómo expresamos requerimientos que transcurren en tiempo?
- ¿Cómo describir procesos qué requieren de la aplicación de varias operaciones?
- ¿Cuál es la pre y post condición para un sistema de despachos de taxis?
- ¿Es viable describir todo en lógica de primer orden?
- ¿Qué pasa si no desarrollo todo (¿compro?, ¿subcontrato?)
- ¿Qué pasa si tengo que modificar en vez de desarrollar?

Ingeniería de Software

- Algunas Definiciones -

Software engineering is the application of a systematic, disciplined, quantifiable approach to the ***development, operation, and maintenance*** of software.

IEEE Standard Glossary of Software Engineering Terminology, IEEE std 610.12-1990

No es solo desarrollo

Ingeniería de Software

- ¿Qué temas abarca? -

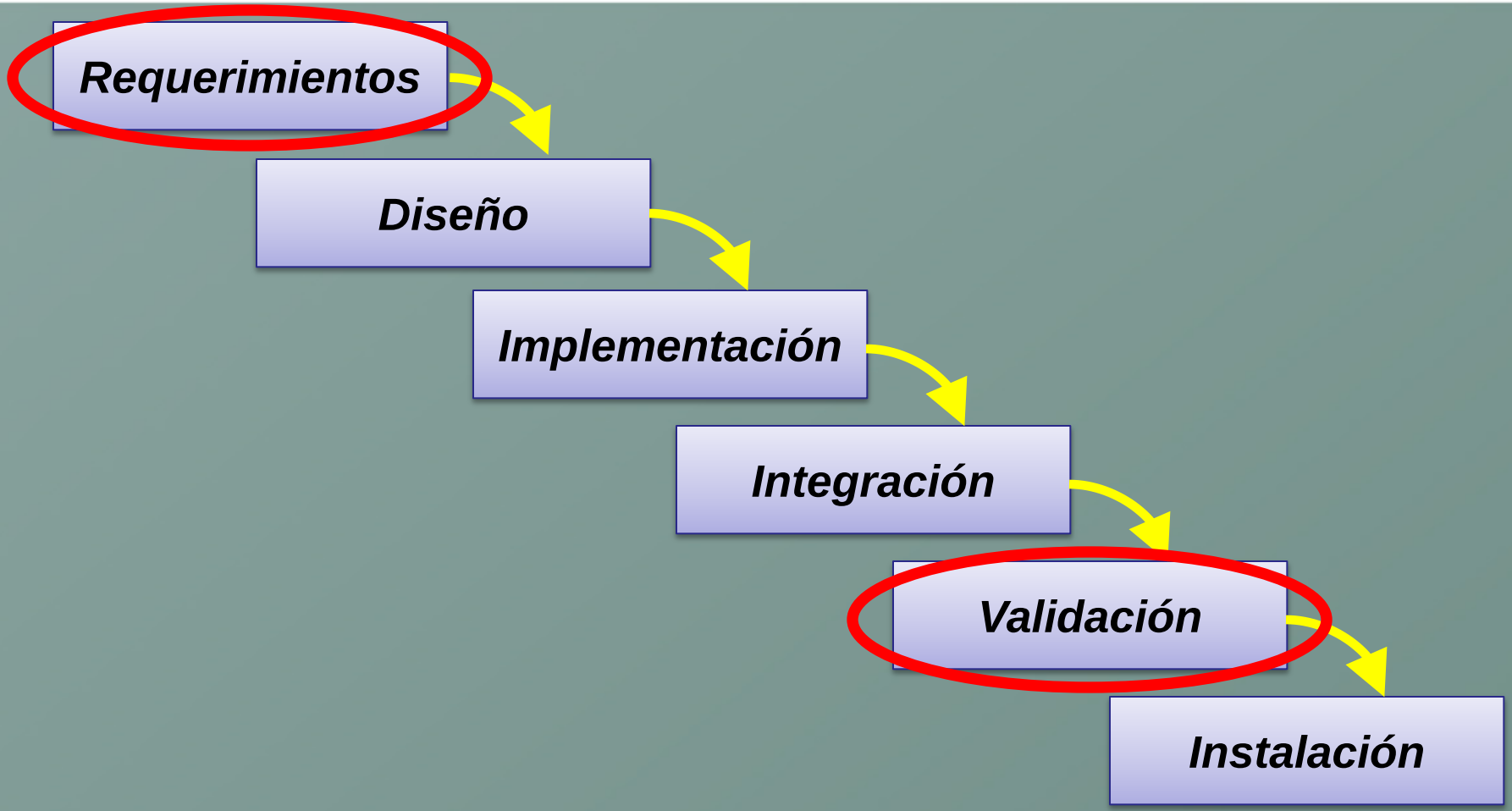
The discipline of software engineering encompasses knowledge, tools, and methods for defining software requirements, and performing software design, software construction, software testing, and software maintenance tasks. Software engineering also draws on knowledge from fields such as computer engineering, computer science, management, mathematics, project management, quality management, software ergonomics, and systems engineering.

The Software Engineering Body of Knowledge, IEEE CS and ACM

Interdisciplinario

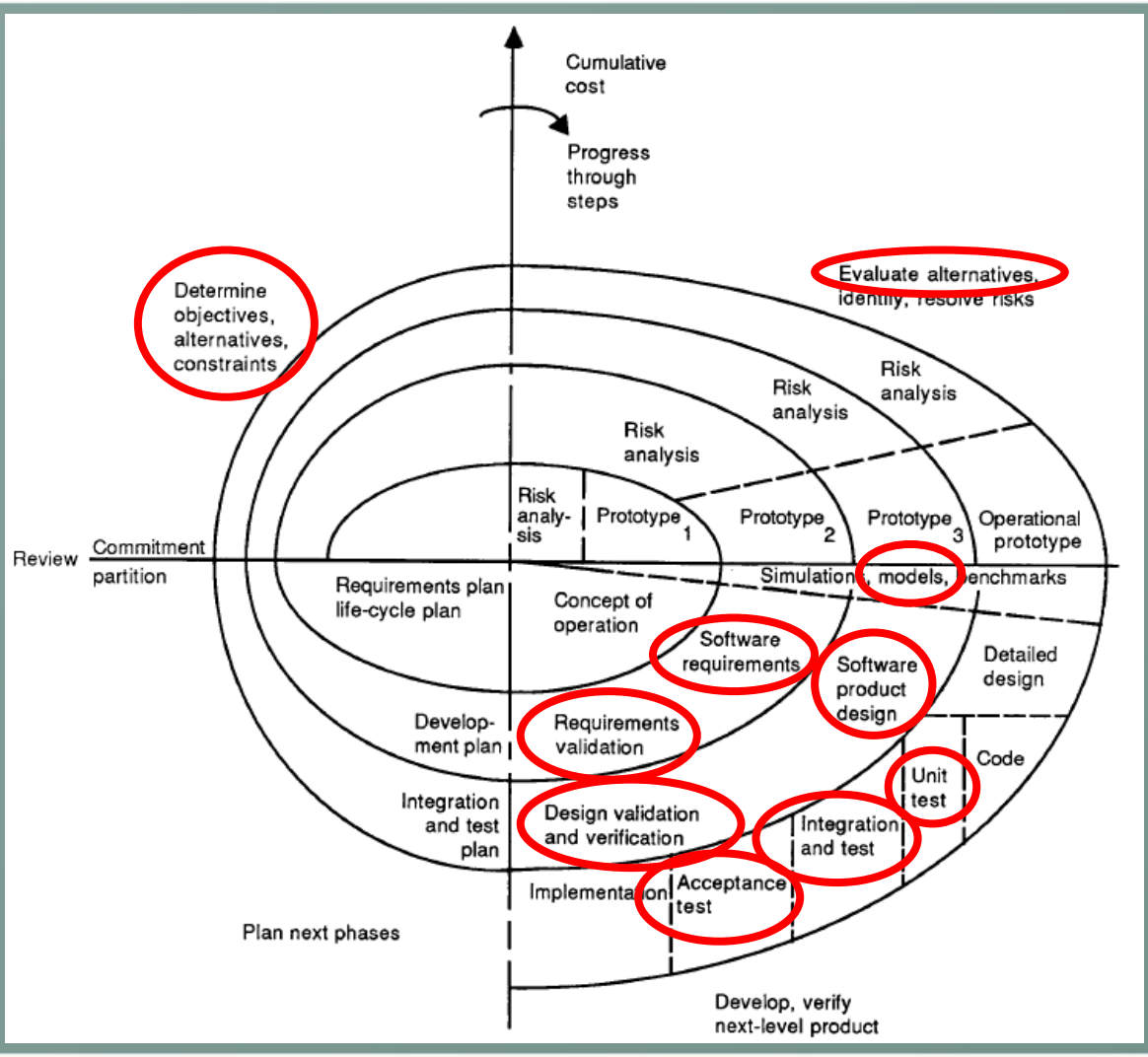
Ciclo de Vida del Desarrollo de Software

Modelo Cascada (Royce, 1970)



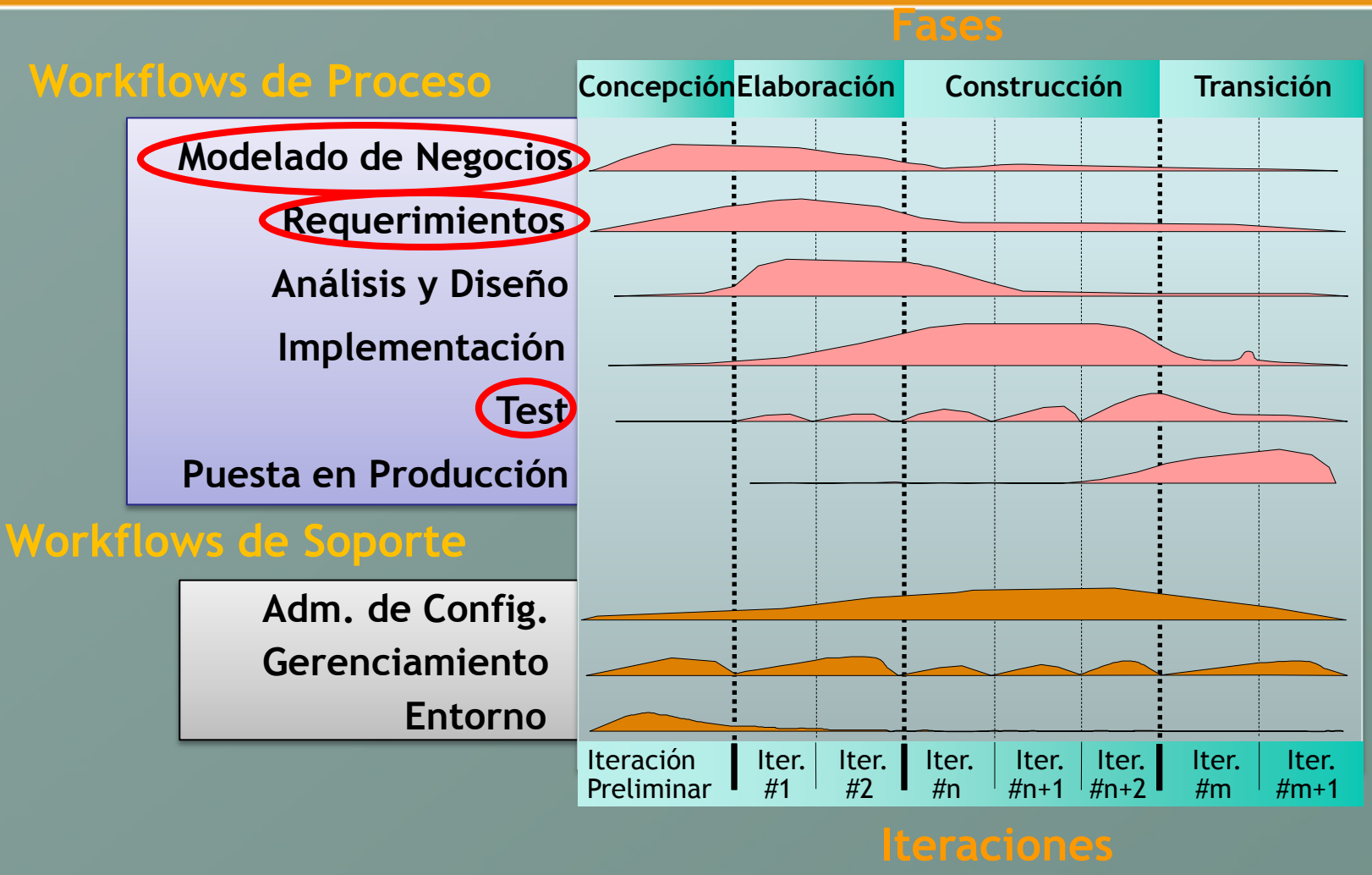
Ciclo de Vida del Desarrollo de Software

Modelo Espiral (Boehm, 1988)



Ciclo de Vida del Desarrollo de Software

Unified SW Development Process (Jacobson, 1999)



¿Por qué es difícil?

- Software es uno de los objetos de mayor complejidad hecho por humanos
  - Desarrollar software es resolver un juego de restricciones de naturaleza técnica, económica y humana
- Una disciplina joven victima de su propio éxito
  - La teoría sobre la que se debe apoyar la ingeniería no está terminada
  - La tecnología y la capacidad de construir sistemas complejos crece rápidamente exigiendo más a una disciplina ingeniería que esta madurando y introduciendo nuevas problemáticas

¿Cómo lidiar con la complejidad?

- En las ingenierías tradicionales la construcción y análisis de **modelos** juega un rol fundamental
- Los modelos sirven para
  - Analizar un aspecto del problema a resolver o artefacto a construir para lograr mayor confianza
  - Comunicar en forma precisa aspectos del problema y la solución a otras personas
- Los modelos son efectivos porque
  - Son significativamente más barato de construir que el sistema
  - Permiten detectar errores y falencias tempranamente

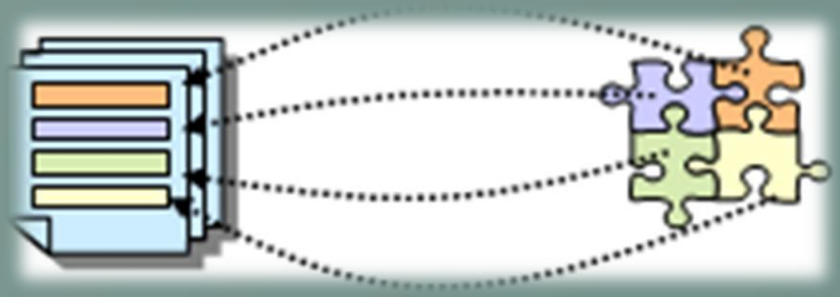
Modelos y Diagramas

- Un modelo captura una vista de un sistema del mundo real. Es una abstracción de dicho sistema, considerando un cierto propósito. Así, el modelo describe completamente aquellos aspectos del sistema que son relevantes al propósito del modelo, y a un apropiado nivel de detalle.

- Diagrama: una representación gráfica de una colección de elementos de modelado, a menudo dibujada como un grafo con vértices conectados por arcos.

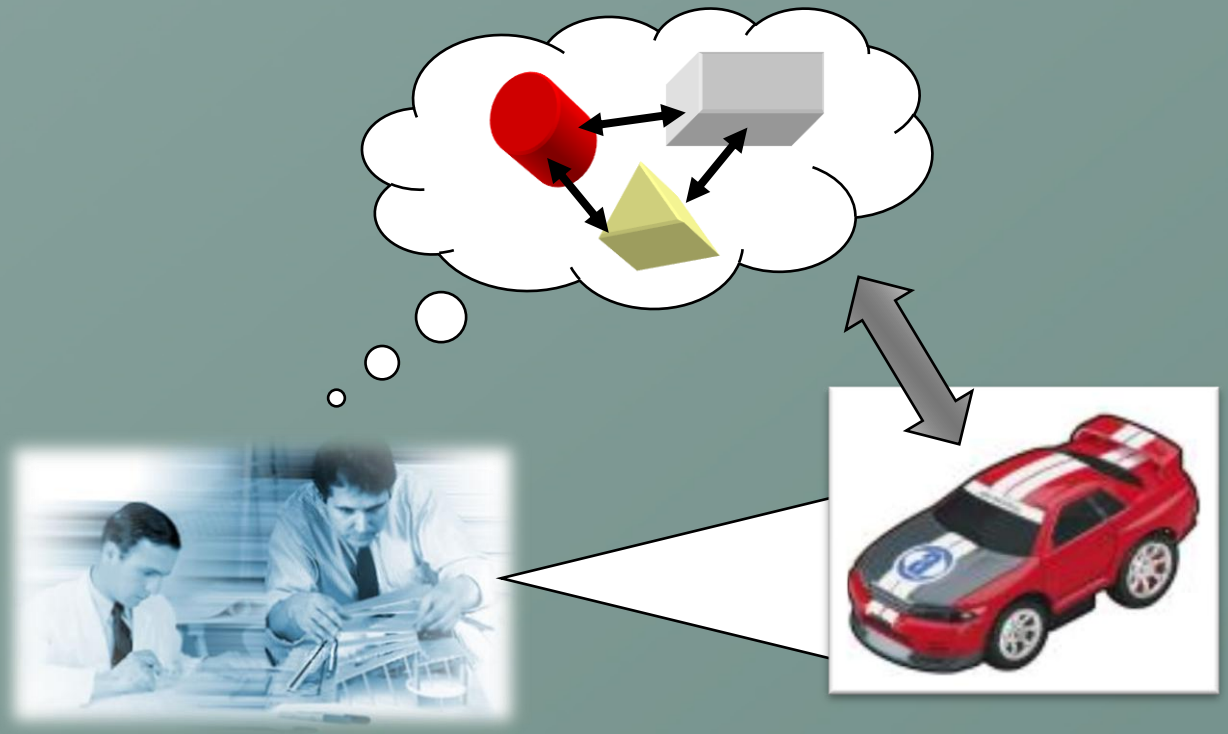
Modelos y Diagramas

- Un proceso de desarrollo de software debe ofrecer un conjunto de modelos que permitan expresar el producto desde cada una de las perspectivas de interés
- El código fuente del sistema es el modelo más detallado del sistema (y además es ejecutable). Sin embargo, se requieren otros modelos ...
- Cada modelo es completo desde su punto de vista del sistema, sin embargo, existen relaciones de trazabilidad entre los diferentes modelos



Modelo

El modelo es conocimiento depurado



Conocimiento implícito y explícito

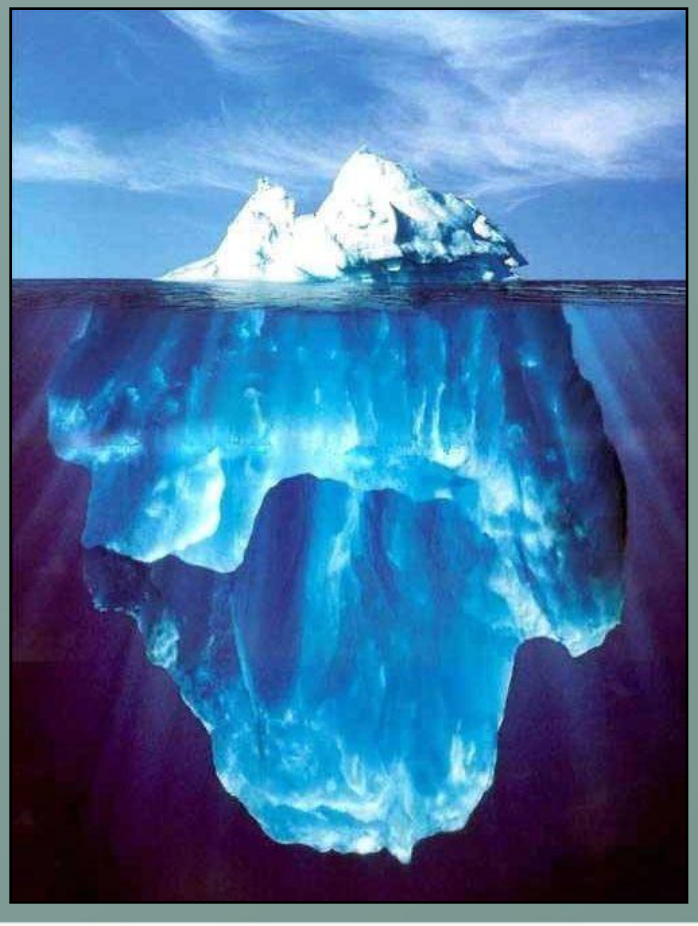
El conocimiento sobre un proyecto está fragmentado, repartido entre muchas personas y documentos, y está mezclado con otra información de tal manera que ni siquiera conocemos cuáles son los fragmentos de información que realmente necesitamos.

- **Conocimiento explícito**: este conocimiento puede expresarse en palabras y números y compartido en la forma de datos, fórmulas científicas, especificaciones, manuales, etc. Puede transmitirse entre las personas formal y sistemáticamente

- **Conocimiento implícito**: es en gran medida personal y muy difícil de formalizar, dificultando mucho la comunicación o el ser compartido por otros. Los pálpitos subjetivos, las intuiciones, los presentimientos caen dentro de esta categoría. Es difícil de verbalizar, dado que está íntimamente enraizado en las acciones y experiencias de una persona, además de los ideales, valores o emociones que esa persona pueda adoptar.

Conocimiento implícito y explícito

Relación -de acuerdo a varios autores- entre conocimiento tácito y explícito



10% Conocimiento visible, comunicable, formalizable (Explícito)

90% Conocimiento oculto, ligado a la experiencia (Implícito)

Unified Modeling Language

Diagramas Estructurales

- Diagrama de Casos de Uso
- Diagrama de Clases
- Diagrama de Objetos

Diagramas de Comportamiento

- Diagrama de Estados
- Diagrama de Actividad

Diagramas de Interacción

- Diagrama de Secuencia
- Diagrama de Colaboración

Diagramas de Implementación

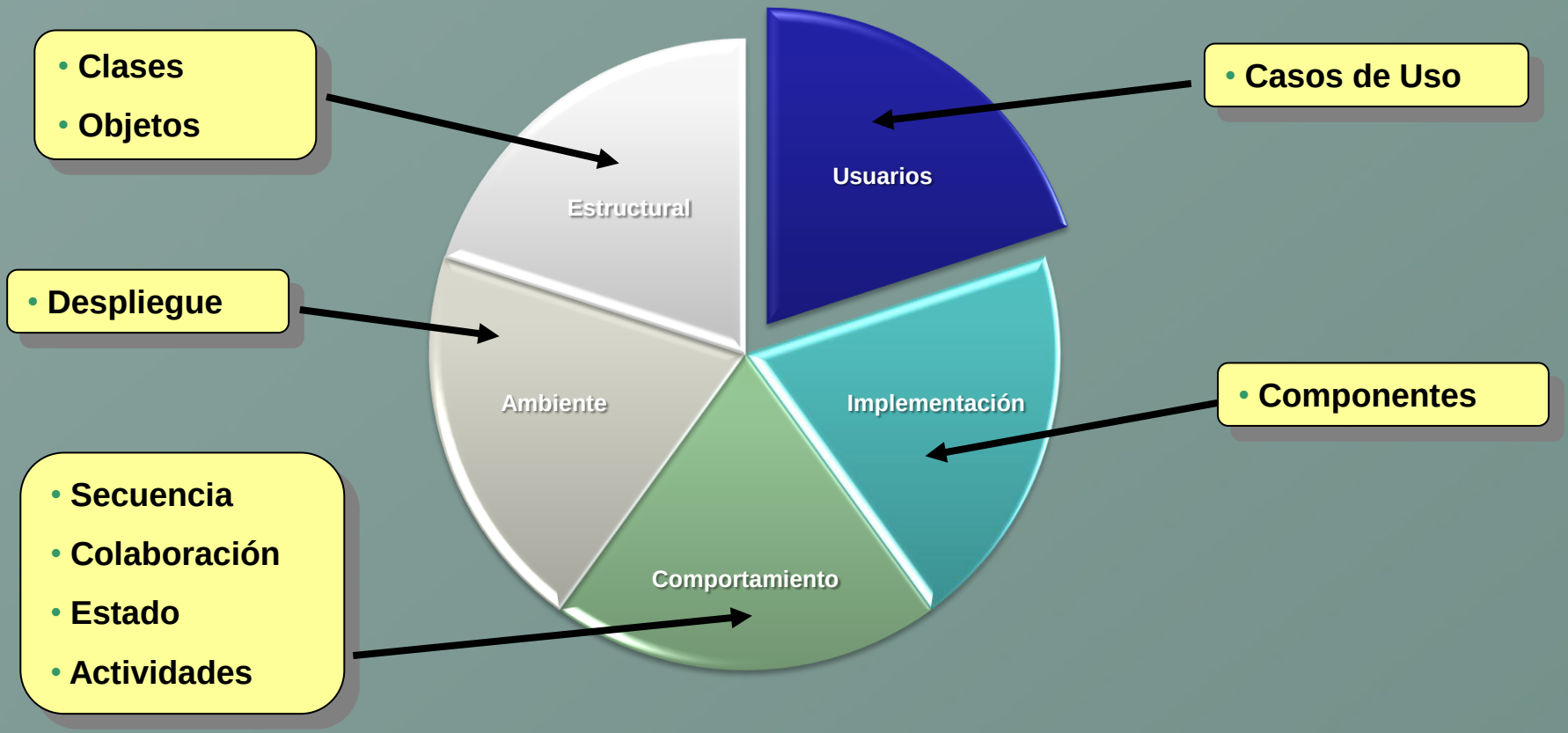
- Diagrama de Componentes
- Diagrama de Despliegue

Diagramas de UML

Los diagramas expresan gráficamente partes de un modelo.

UML

“Vistas y Diagramas”



Documentos y Diagramas

- La primera percepción de la utilidad de un diagrama UML -un diagrama de clases u objetos- puede ser cuando discutimos sobre un diseño de software.
- Mucha gente es naturalmente visual y los diagramas ayudan a captar cierto tipo de información: los diagramas UML son muy útiles para comunicar relaciones entre objetos y para mostrar interacciones.
- Pero no transmiten la definición conceptual de estos objetos. En una discusión deberemos rellenar estos significados hablando mientras vamos dibujando estos diagramas, o surgirán en un diálogo con otros participantes.

Documentos y Diagramas

- Es decir que un diagrama UML no puede transmitir dos de los aspectos más importantes de un modelo: el significado de los conceptos que representa y qué se supone que hacen los objetos.
- Esta es la función complementaria del lenguaje hablado.
- Los diagramas son un medio de comunicación y de explicación y facilitan el proceso de creación de ideas (brainstorming).

Documentos y Diagramas

- Deben ser mínimos: diagramas completos -aunque sean comprensibles- del modelo total de objetos fracasan en su intento de comunicar o explicar. Agobian al lector con detalles al mismo tiempo que carecen de los significados más importantes.
- Algunos prefieren utilizar los diagramas UML al revés: en vez de un diagrama con anotaciones en forma de texto que aclaren detalles, prefieren un documento escrito ilustrado con diagramas seleccionados y simplificados.

Documentos y Diagramas

- Siempre debemos recordar que *el modelo no es el diagrama* (lo mismo que el mapa no es el territorio), el propósito del mismo es ayudar a comunicar y explicar el modelo.

Documentos y Diagramas

- La comunicación hablada complementa el rigor de los diagramas y del código, pero aunque sea vital para conectar a todos con el modelo, a partir de una cierta cantidad de integrantes de un equipo se necesita la estabilidad y el poder compartir que ofrecen los documentos escritos.
  - Los documentos deben complementar al modelo y a las conversaciones.
  - Un documento no debería intentar hacer lo que el código hace perfectamente.
  - Los documentos deberían servir para seguir funcionando y estar actualizados.
  - Un documento debe estar involucrado en las actividades del proyecto.

“Scope” y “Span” de un Modelo

- **Scope:** tipo de fenómeno que se capta (ej. Mapa político, mapa físico); aspecto.
- **Span:** Individuos descriptos (ej.: Mapa de América, Mapa de Asia); segmento.

- Los modelos tienen un scope y un span.

Modelos en Ingeniería de Software

- El producto (principal) a construir es intangible: El programa
- Los modelos que construimos serán intangibles también
  - A diferencia del ala del avión
- Los modelos en Ingeniería de Software son lenguajes formales con una denotación precisa en el mundo real
- Es imposible comunicar/analizar una descripción efectivamente si
  - La denotación no está bien definida
  - Los usuarios del modelo no la entienden y aplican rigurosamente
- Cómo defino una denotación?
  - Usando modelos con denotación “pre-definida” (Usos y costumbres en Ingeniería de Software)
  - Lenguaje natural, glosarios, etc..

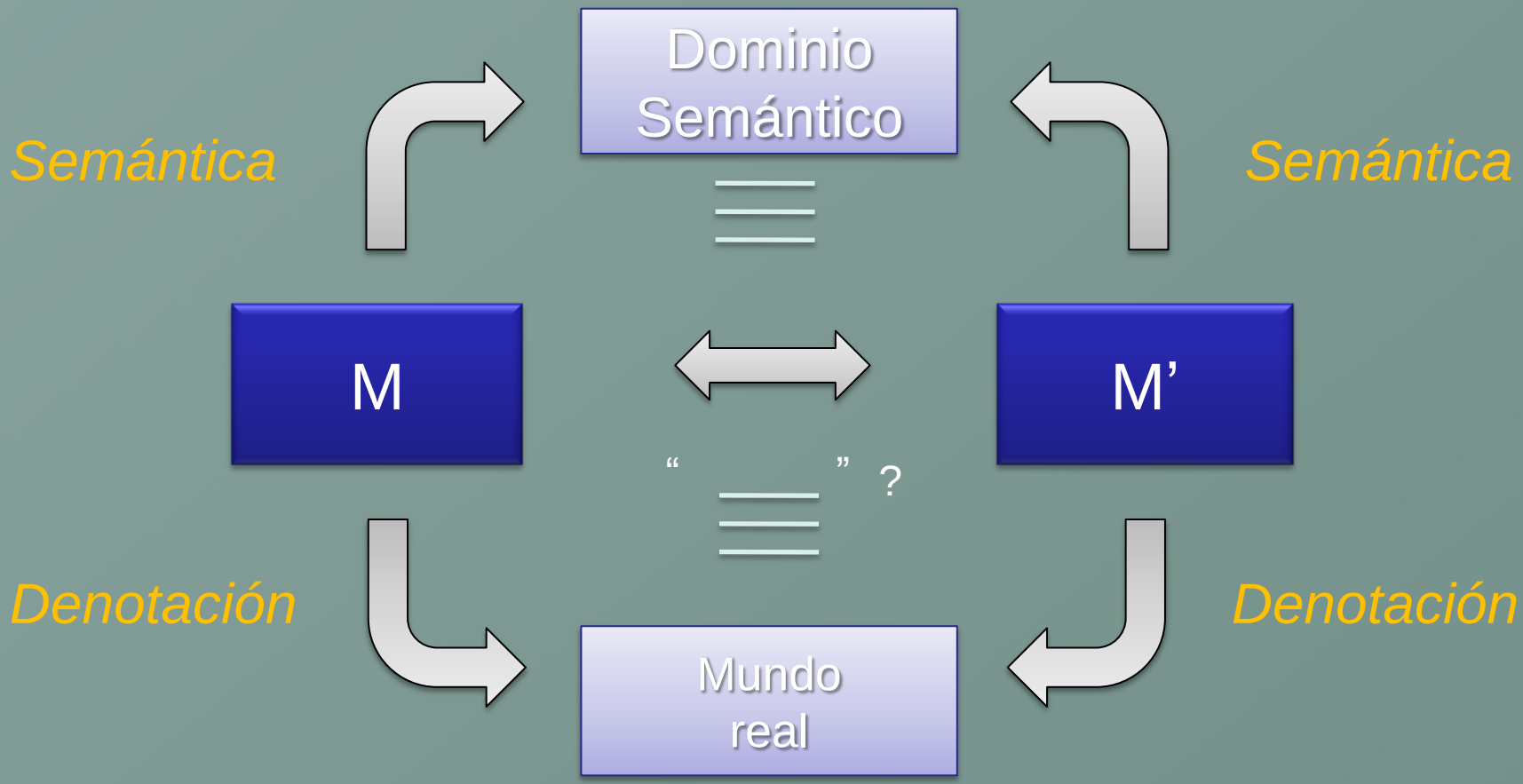
Lenguajes Formales

- Elementos de un lenguaje formal
  - Una Sintaxis (gráfica / texto) para describirlos
    - Cuales son los garabatos permitidos en una descripción?
  - Una Semántica para abstraer accidentes sintácticos
    - Cuales de estos garabatos significan lo mismo?
- ¿Qué lenguajes formales conocen? ¿Cuál es su sintaxis y semántica?
- ¿Cómo defino/documento/transmito una semántica?

  Hay muchos métodos, dos tradicionales son:

  1. Con una traducción a otro lenguaje formal
  2. Definiendo relaciones (de equivalencia o de orden) entre elementos sintácticos

Sintaxis, Semántica y Denotación



No es un Modelo, son muchos!

- Cada modelo enfoca en un aspecto del sistema para permitir análisis riguroso y escalable (scope y span)
- Un sistema se analiza desde múltiples modelos complementarios
- Es imposible vincular los distintos modelos formalmente manteniéndolos analizables y entendibles (es decir útiles)
  - ¿Lógica de primer orden?
  - Aquí las interpretaciones consistentes del los modelos en el dominio por parte del Ingeniero son fundamentales

Modelos de Ingeniería de Software

- Una visión desde los Requerimientos -

Grafos de refinamiento Y/O

Ing. Soft 2

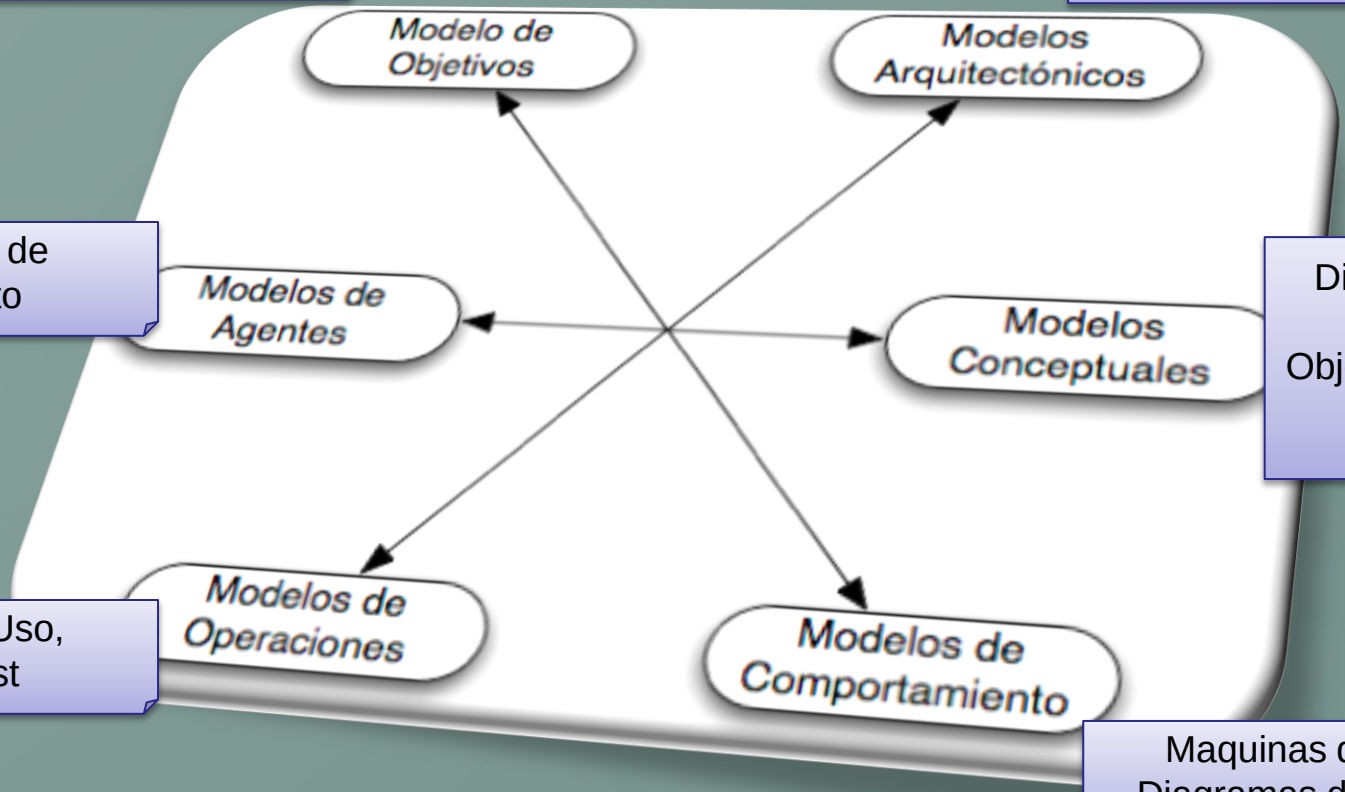


Diagrama de Contexto

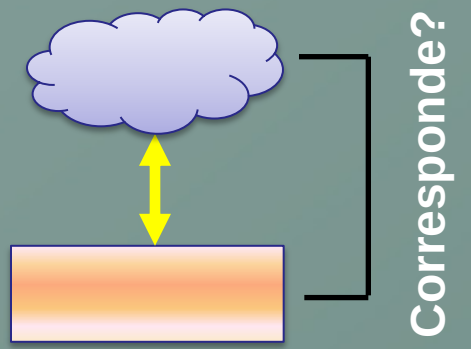
Diagramas de Clases, Objetos, Entidad - Relación

Casos de Uso, Pre/Post

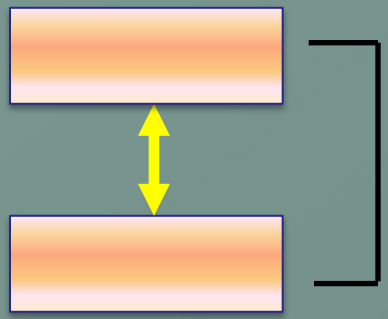
Maquinas de Estado, Diagramas de Secuencia

Validación y Verificación

- Validación: un proceso cuyo objetivo es incrementar la confianza de que una descripción formal se *corresponde* con la realidad (es decir, el mundo informal)
  - Ej. si la descripción del problema se corresponde con las necesidades reales.



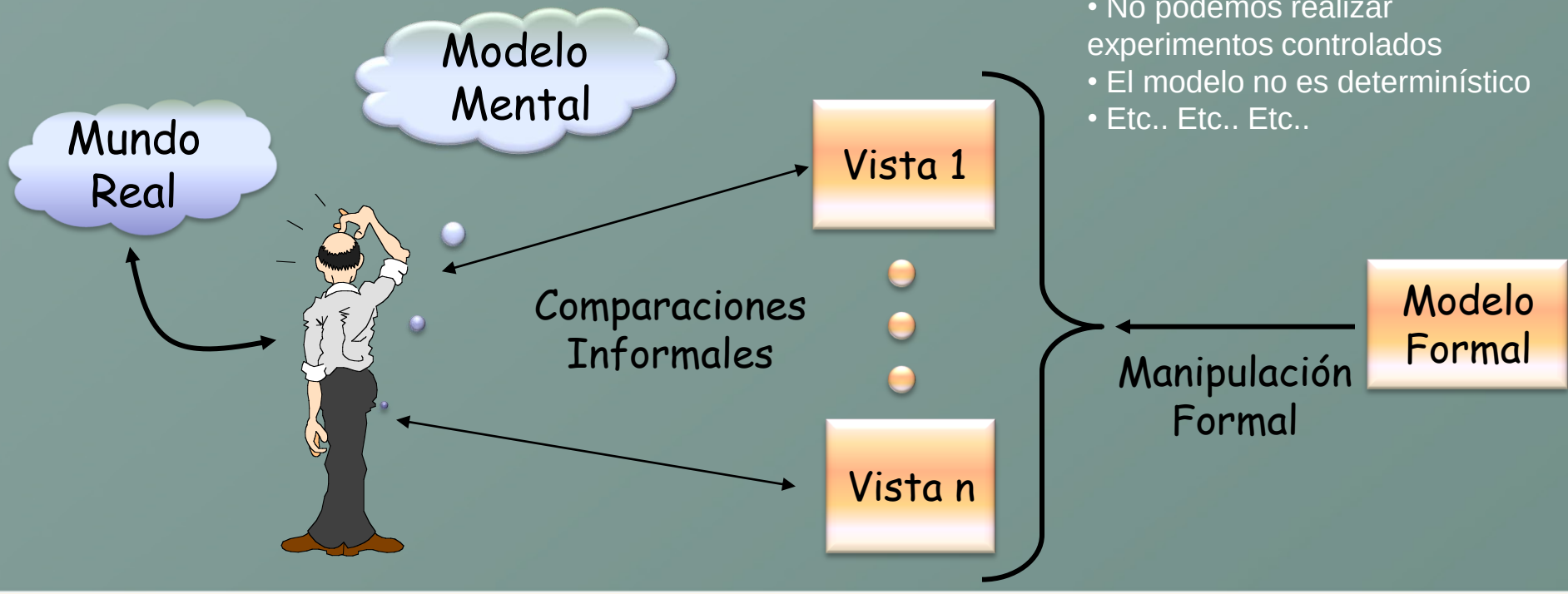
- Verificación: un proceso cuyo objetivo es garantizar que una descripción formal es *correcta* con respecto a otra
  - Ej. garantizar que la descripción del problema satisface la descripción de la solución.



Dificultades de la Validación

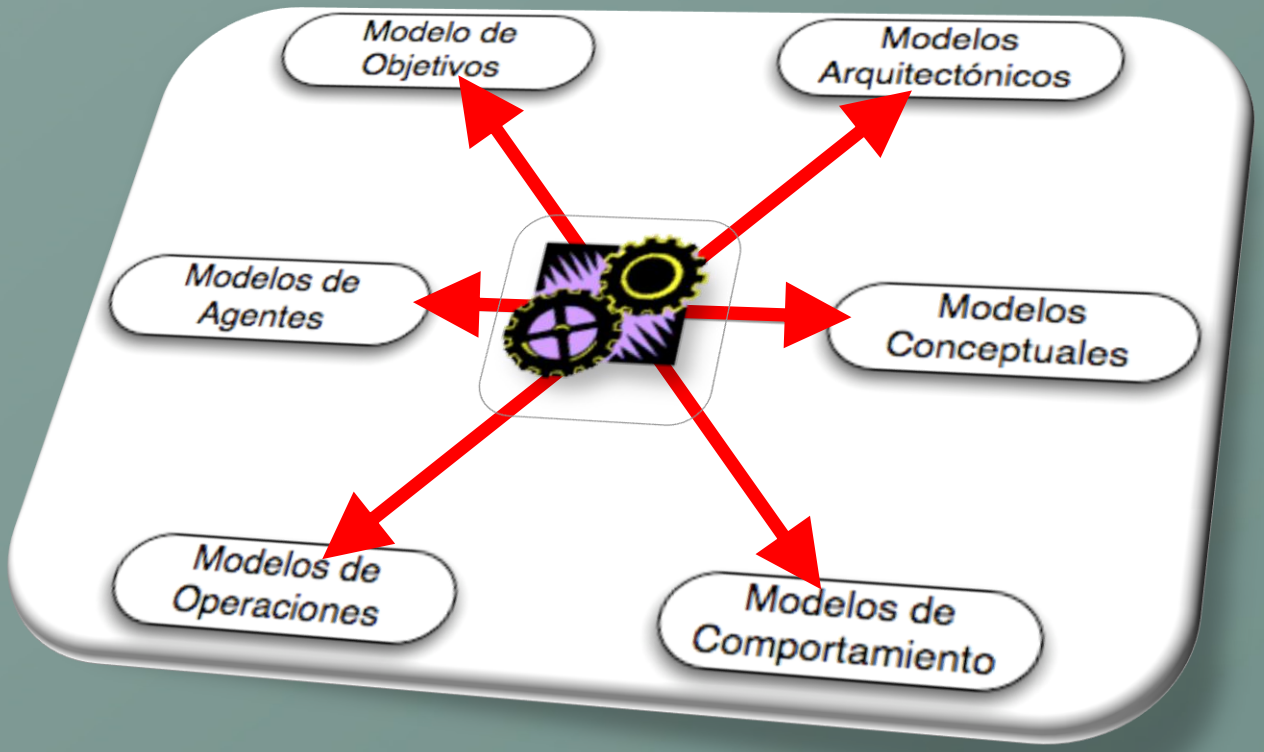
- Es imposible: Formal vs. Informal

• El sistema real no existe
• El problema es intangible
• Los stakeholders no están disponibles
• Observar el mundo no es inocuo (Hawthorne)
• No podemos realizar experimentos controlados
• El modelo no es determinístico
• Etc.. Etc.. Etc..



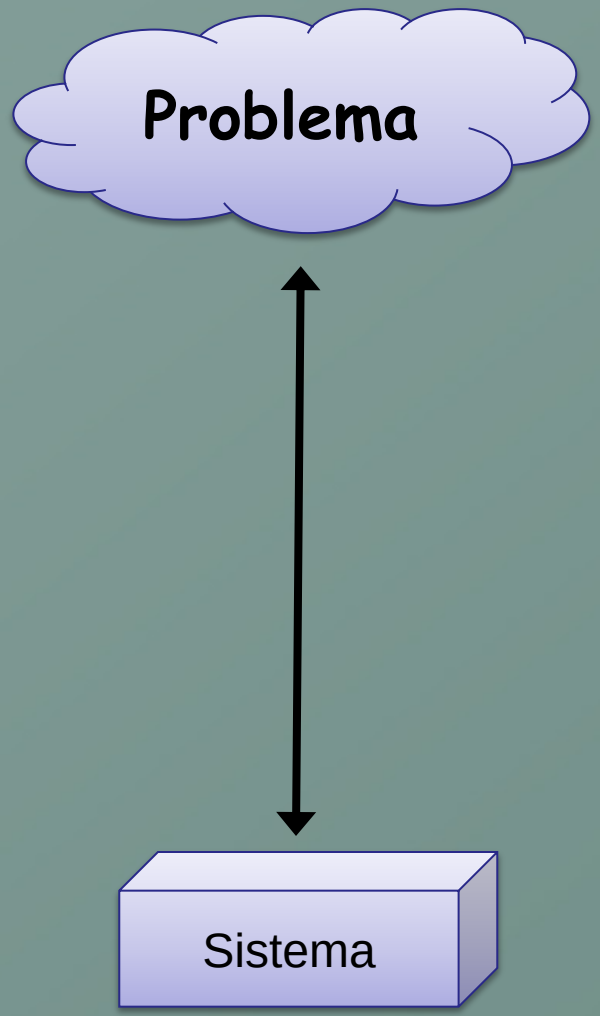
Dificultades de la Verificación

- Vínculos entre modelos son difíciles de formalizar
- ¿Cómo comparo modelos con scope y span diversos?
- Complejidad de la prueba puede ser excesiva (tiempo y espacio)



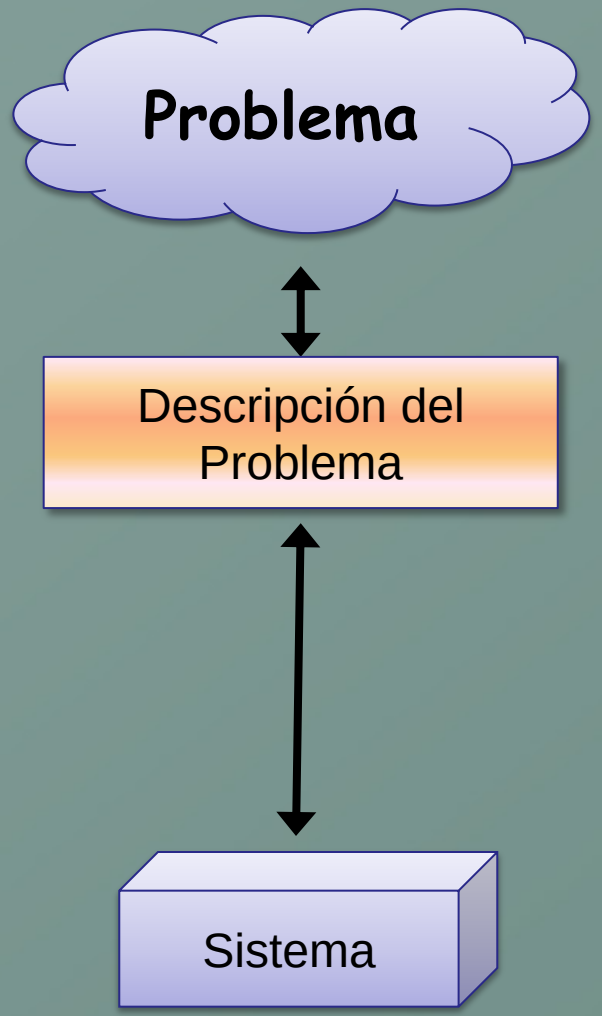
Validación es todo

- El objetivo final es resolver el problema
- ¿Cómo validamos un sistema complejo?
- ¿Qué pasa si el sistema construido no es válido?



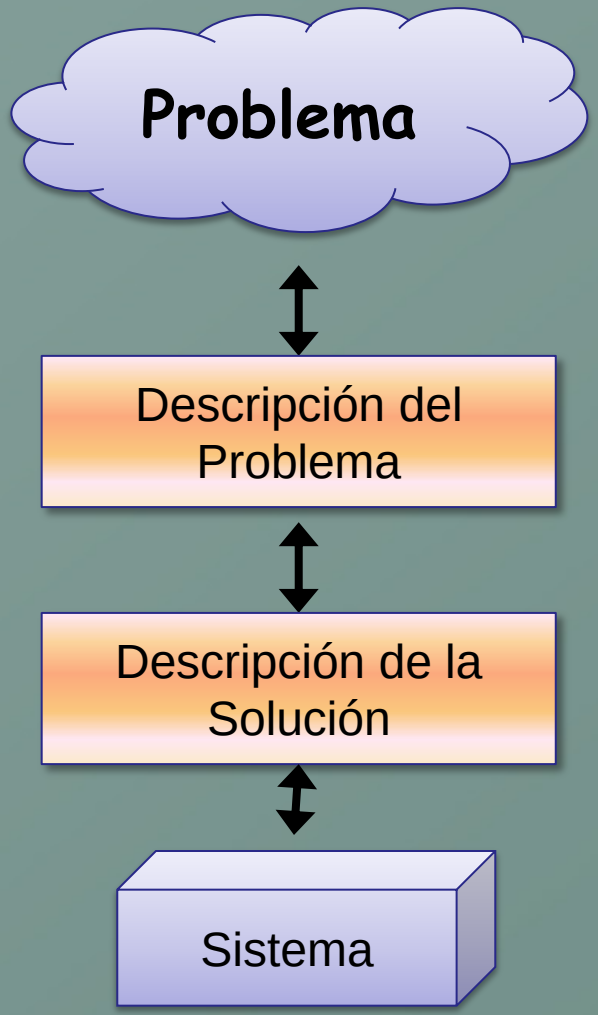
Achicar el Gap a Validar

¡Aquí la denotación de la descripción del problema es fundamental!

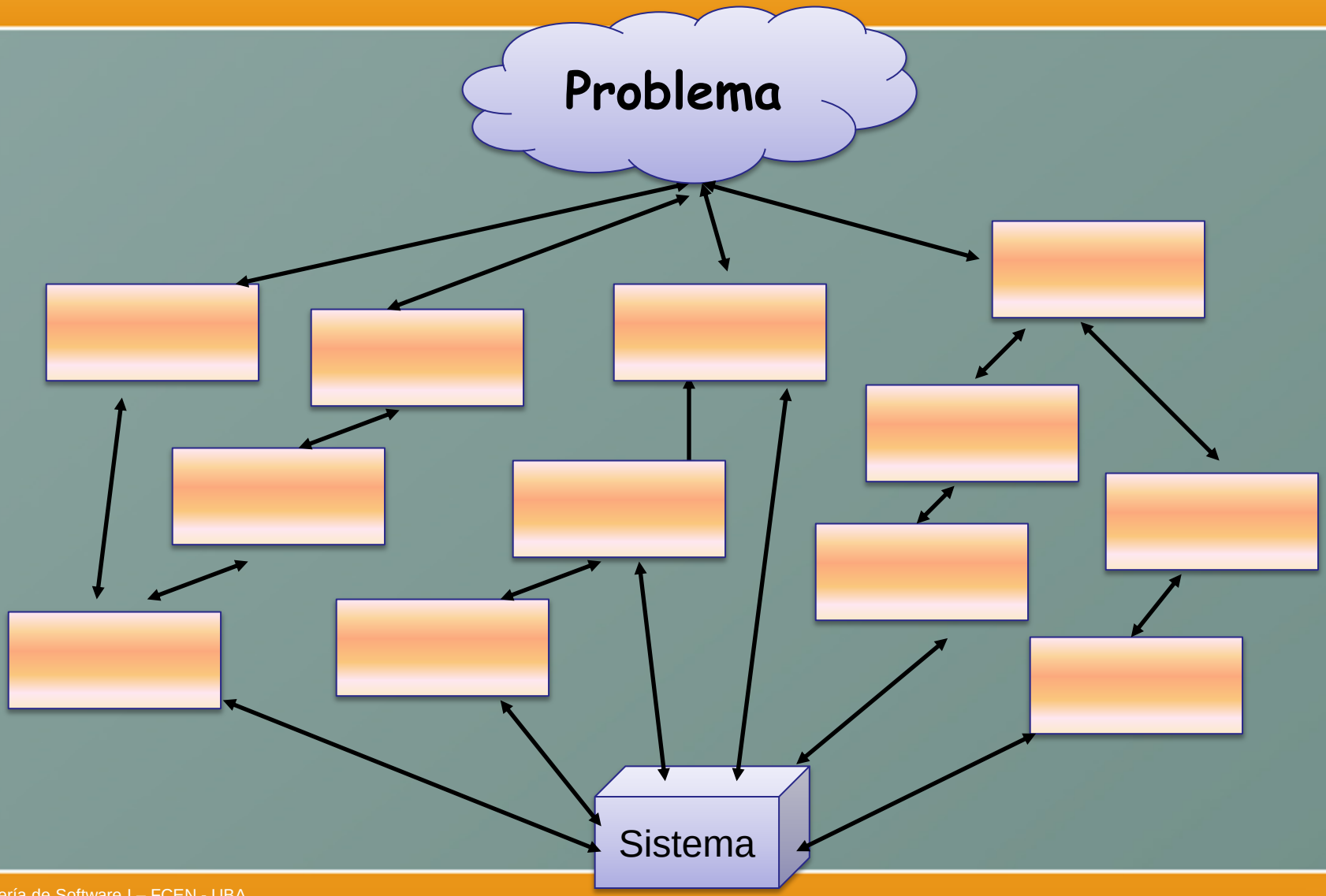


Achicar el gap a Verificar

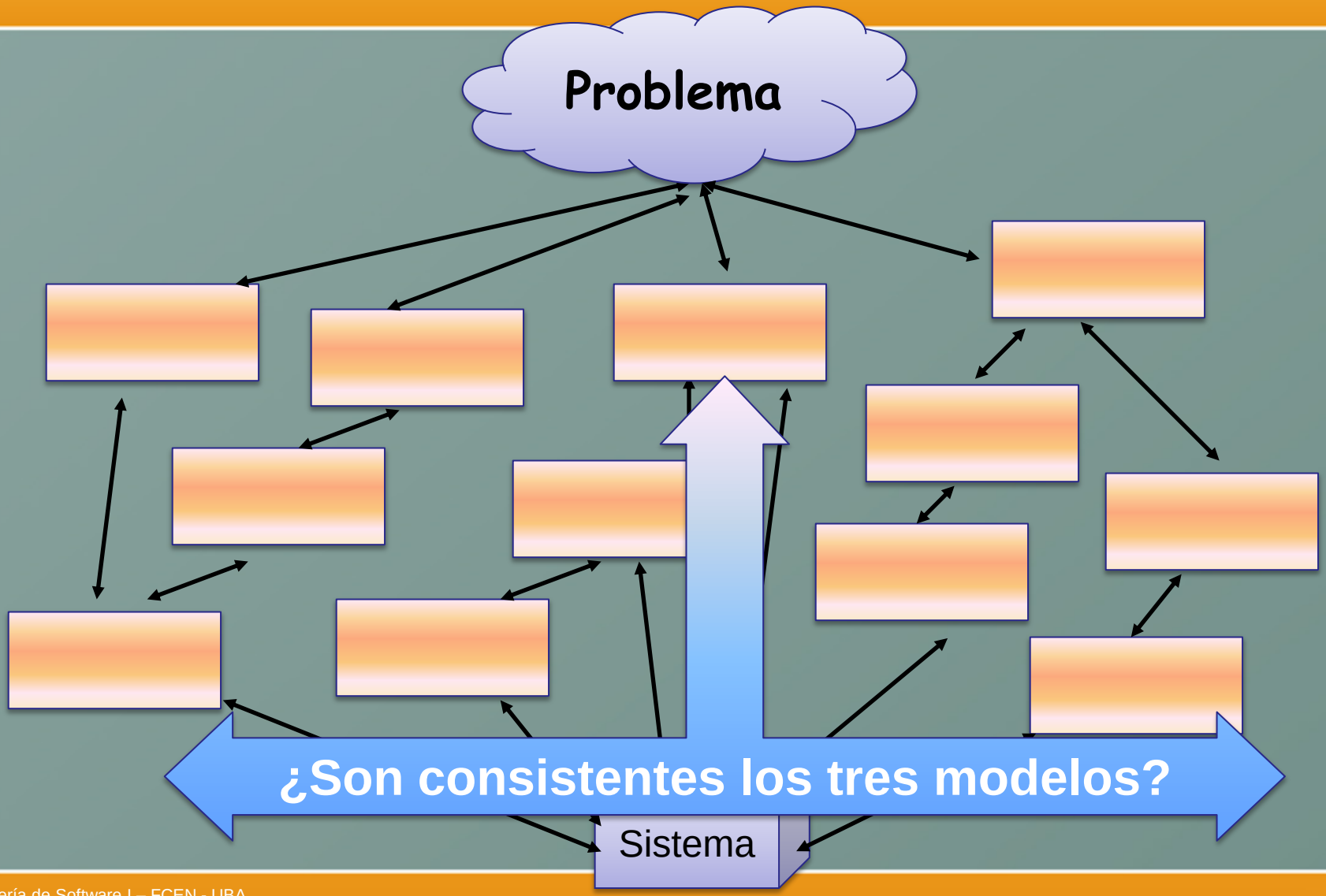
¿Y aquí…?



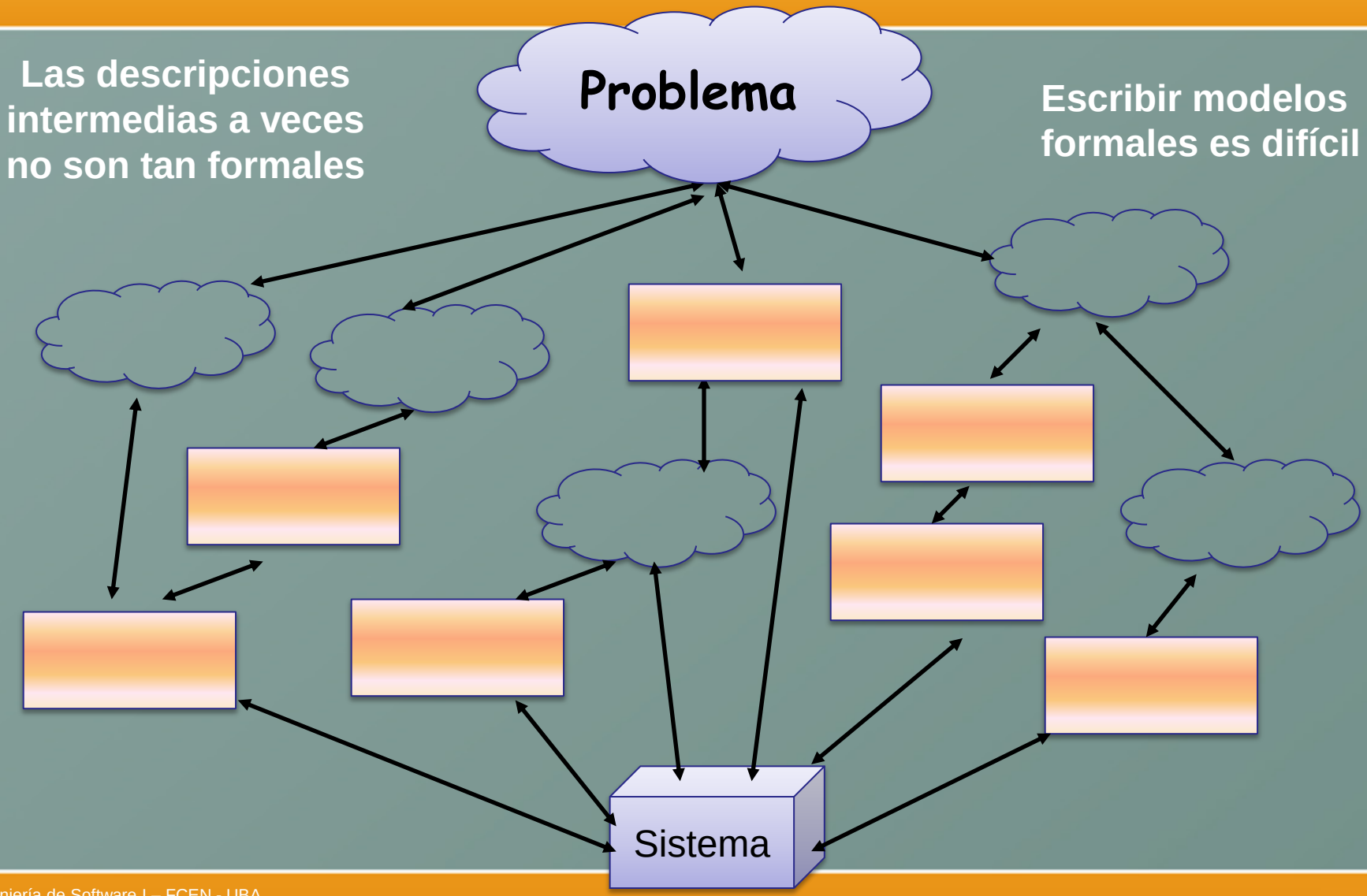
Muchas Descripciones y Gaps



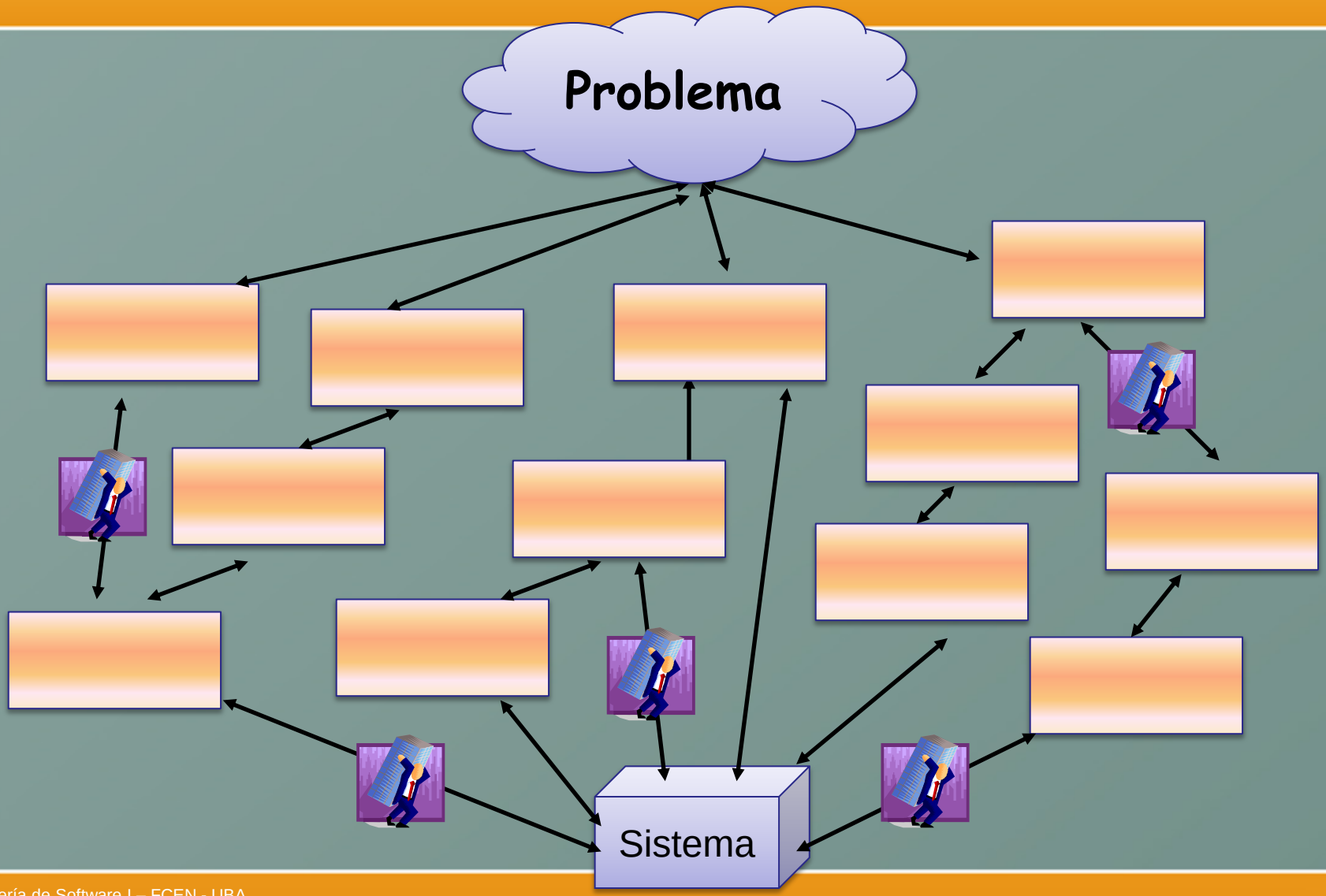
Nuevos Problemas



Otro problema nuevo



Decidibilidad y Complejidad de Verificación



Formalidad y Rigurosidad

- La Ingeniería de Software intenta ayudar a viajar
  - de lo informal al lo formal
  - del mundo real (con necesidades y problemática) prácticamente imposible de formalizar al mundo software (HW + códidgo fuente) completamente formal

- Este viaje no puede hacerse formalmente…
- Pero si puede hacerse más o menos rigurosamente

- La ingeniería de software intenta hacer este viaje lo más rigurosamente posible (dentro de los límites de presupuesto - tiempo y $)

Resumen

- Qué es la Ingeniería de Software
- Porqué lo visto hasta ahora no alcanza
- El rol central de modelado
- Modelos
  - Sintaxis, Semántica y Denotación
  - Scope & Span
  - Verificación y Validación