Recurrencia en el Desarrollo Software

Universo Santa Tecla

Índice

¿Por qué? 2
¿Qué? 3
Conceptos Recurrentes 4
¿Para qué? 7
¿Cómo? 8
Itinerario 9
Sintesis 10
Bibliografía 11
Ponente 12


Patrón
Conocimiento
Disciplina
Reusabilidad
Datos
Información
Sencillez
Compuesto
Eficacia
Coste
Software
Proyecto
Calidad
Clases
Síntesis
Complejidad necesaria
Compromiso
Evolución del Software
Unidad
Diseño
Orden
Ámbito
Mantenibilidad
Acoplamiento
Encapsulación
Martin Fowler
Kent Beck
Grady Booch
UNIVERSO SANTA TECLA

¿Por qué?

An Integrated, Breadth-First Computer Science Curriculum Based on *Computing Curricula 1991*

John T. Paxton[1] — paxton@cs.montana.edu
Rockford J. Ross[2] — ross@cs.montana.edu
J. Denbigh Starkey — starkey@cs.montana.edu

Computer Science Department
Montana State University
Bozeman, MT 59717

1 Background

Montana State University (MSU) was required to change from quarters to semesters beginning in autumn 1991. The Computer Science Department took the opportunity to completely revise the introductory course sequence required of all majors and minors. It had been felt for some time that a more *integrated* curriculum was necessary, one that would weave important subject areas—such as discrete math, data structures, and algorithms—into a cohesive unit. It had also been recognized that a *breadth-first* approach that introduced students to the major subject areas of the discipline (e.g., compilers, data bases, operating systems, etc.) would be superior to the existing curriculum.

Strong encouragement for such change appeared in two widely-referenced reports published during this period. "Computing as a Discipline" (often called the "Denning Report") [Denning *et al.* 1989], and *Computing Curricula 1991* [Tucker 1991]. The "breadth-first" models proposed there greatly influenced the design of our new curriculum.

The brevity of the required format for this paper makes a detailed discussion of the new curriculum impossible. A more complete description can be obtained from the authors [Paxton, Ross, and Starkey 1993-1].

[1]Support for part of the developmental work described in this paper came from a Montana State University Instructional Research Grant.
[2]Support for development of the formal laboratories described in this paper came in part from the Instrumentation and Laboratory Improvement Program of the National Science Foundation, grant number USE-9150298.

2 Curriculum Objectives

Until academic year 1991–92, the computer science introductory course sequence at MSU was fairly standard, based on the Curriculum 78 [ACM 1979] course models. Students took a year of programming followed by a year that included discrete math, data structures, and algorithms. The results had not been satisfactory.

- The compartmentalization of topics into separate courses kept the students from recognizing the correlations among them.
- The overwhelming majority of students—even the best ones—only endured discrete math, seldom appreciating its relevance or application to computer science.
- Students reached their junior years with no cohesive picture of the discipline of computing.
- Somehow, and perhaps most distressingly, the introductory courses seemed to leave computer science students disconnected from their discipline. The infrastructure that engenders comraderie and identification with the discipline was lacking.

We recognized that rather than being treated as a sequence of separate courses with separate topics, the curriculum needed to be viewed as an integrated entity with three primary objectives: (1) to give majors a clear understanding of the foundations of computer science in preparation for advanced courses, (2) to present minors with a comprehensive view of the discipline, and (3) to excite students about computer science.

3 Curriculum Design

A four-semester course sequence was settled upon for the new curriculum. Each course is a three-credit offer-

- **Computing Curricula, 1991.**
  - **ACM/IEEE-CS Joint Curriculum Task Force**

¿Qué?

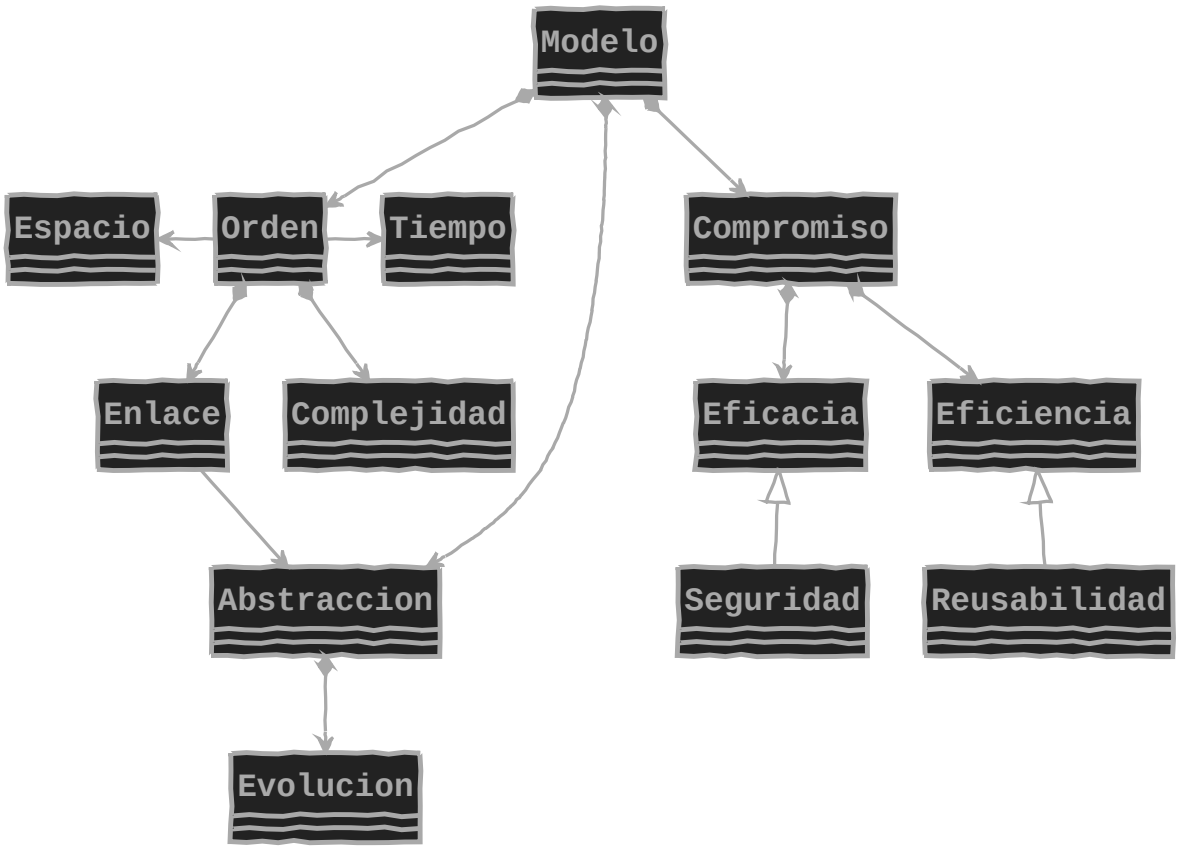
Conceptos Recurrentes

Concepto Recurrente	Definición	Ejemplos
Enlace	• Los procesos de hacer una abstracción más concreta al asociarle propiedades adicionales.	• *asociar (asignar) un proceso con un procesador,* • *asociar un tipo con un nombre de variable,* • *asociar un programa de objeto de biblioteca con una referencia simbólica a un subprograma,* • *instanciar en programación lógica,* • *asociar un método con un mensaje en un lenguaje orientado a objetos,* • *creando instancias concretas a partir de descripciones abstractas.*
Complejidad	• Los efectos del no lineal aumentan en complejidad a medida que aumenta el tamaño de un problema.	• *Este es un factor importante para distinguir y seleccionar métodos que se adapten a diferentes tamaños de datos, espacios de problemas y tamaños de programas. En grandes proyectos de programación, es un factor que determina la organización de un equipo de implementación.*
Modelos conceptuales y formales	• Diversas formas de formalizar, caracterizar, visualizar y pensar una idea o problema.	• *modelos formales en lógica,* • *teoría de conmutación y teoría de la computación,* • *paradigmas de lenguajes de programación basados en modelos formales,* • *modelos conceptuales como tipos de datos abstractos y modelos de datos semánticos, y* • *lenguajes visuales utilizados para especificar y diseñar sistemas, como el flujo de datos y diagramas entidad-relación.*
Coherencia e integridad	• Realizaciones concretas de los conceptos de coherencia e integridad en la informática, incluidos conceptos relacionados como corrección, solidez y fiabilidad.	• *Conjunto de axiomas que sirven como especificación formal,* • *la coherencia de la teoría con los hechos observados y* • *la coherencia interna de un lenguaje o diseño de interfaz,* • *Comportamiento del componente o del sistema con las especificaciones establecidas.* • *la adecuación de un conjunto dado de axiomas para capturar todos los comportamientos deseados,* • *la adecuación funcional de los sistemas de software y hardware, y* • *la capacidad de un sistema para comportarse bien en condiciones de error y situaciones imprevistas.*

Concepto Recurrente	Definición	Ejemplos
Eficiencia	• Medidas de costo en relación con recursos como espacio, tiempo, dinero y personas.	• *evaluación teórica de la complejidad espacial y temporal de un algoritmo,* • *la viabilidad, la eficiencia con la que se puede lograr un determinado resultado deseable (como la finalización de un proyecto o la fabricación de un componente) y* • *eficiencia de un determinado resultado*
Evolución	• El hecho del cambio y sus implicaciones. El impacto del cambio en todos los niveles y la resiliencia y adecuación de abstracciones, técnicas y sistemas frente al cambio.	• *modelos formales para representar aspectos de los sistemas que varían con el tiempo, y* • *capacidad de un diseño para resistir las demandas ambientales cambiantes y los requisitos,* • *herramientas e instalaciones cambiantes para la gestión de la configuración.*
Niveles de abstracción	• La naturaleza y el uso de la abstracción en la informática; el uso de la abstracción para gestionar la complejidad, estructurar sistemas, ocultar detalles y capturar patrones recurrentes; la capacidad de representar una entidad o sistema mediante abstracciones que tienen diferentes niveles de detalle y especificidad.	• *niveles de descripción de hardware,* • *niveles de especificidad dentro de una jerarquía de objetos,* • *la noción de genéricos en lenguajes de programación y* • *los niveles de detalle proporcionados en la solución de un problema a partir de especificaciones a través del código.*
Ordenar en el espacio	• Los conceptos de localidad y proximidad en la disciplina de la informática.	• *ubicación física, como en las redes o la memoria,* • *ubicación organizacional (por ejemplo, de procesadores, procesos, definiciones de tipo y operaciones asociadas) y* • *ubicación conceptual (por ejemplo, alcance, acoplamiento y cohesión del software).*
Ordenar en el tiempo	• El concepto de tiempo en el orden de los eventos.	• *parámetro en modelos formales (por ejemplo, en la lógica temporal),* • *sincronizar procesos que se extienden por el espacio,* • *ejecución de algoritmos.*

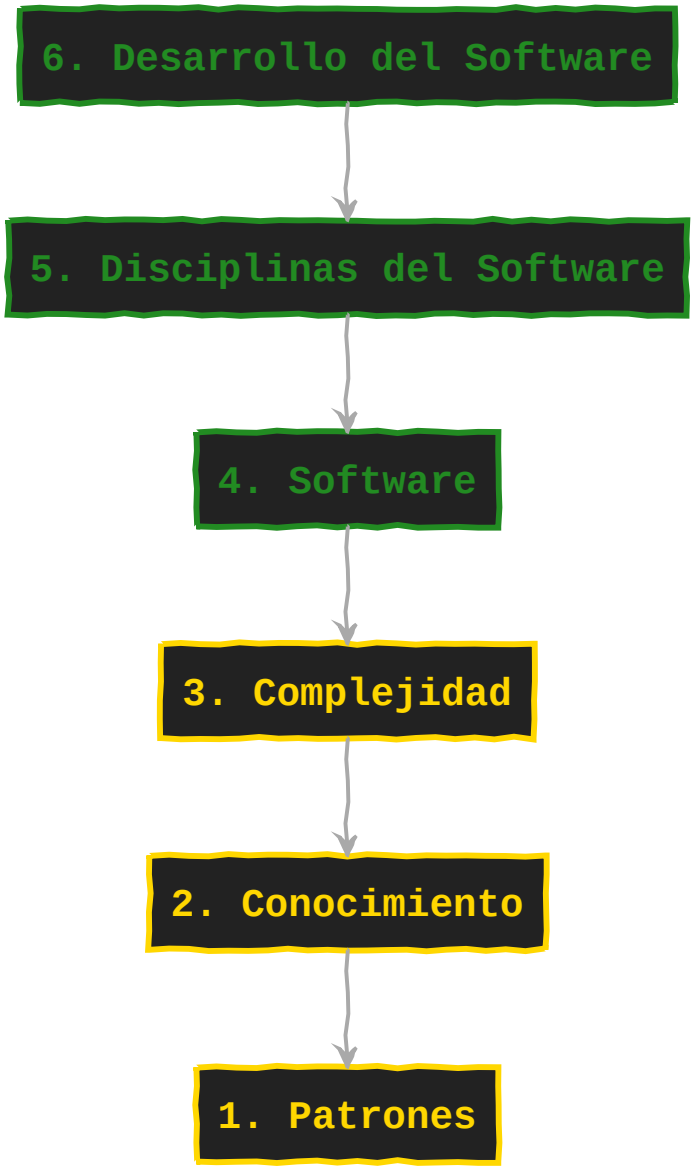
Concepto Recurrente	Definición	Ejemplos
Reutilización	• La capacidad de una técnica, concepto o sistemas en particular para responder de manera adecuada para ser reutilizada en un nuevo contexto o situación.	• *probabilidad, la reutilización de bibliotecas de software y componentes de hardware,* • *tecnologías que promueven la reutilización de componentes de software y* • *abstracciones de lenguaje que promueven el desarrollo de módulos de software reutilizables.*
Seguridad	• La capacidad de los sistemas de software y hardware para responder de manera adecuada y defenderse de solicitudes inapropiadas e imprevistas; la capacidad de una instalación de computadora para resistir eventos catastróficos (por ejemplo, desastres naturales e intentos de sabotaje).	• *verificación de tipos y otros conceptos en lenguajes de programación que brindan protección contra el uso indebido de objetos y funciones de datos,* • *cifrado de datos, otorgamiento y revocación de privilegios por parte de un sistema de administración de bases de datos,* • *características en las interfaces de usuario que minimizan los errores del usuario, medidas de seguridad física en la computadora*
Compromisos y consecuencias	• El fenómeno de los compromisos en la informática y las consecuencias de dichos compromisos. Los efectos técnicos, económicos, culturales y de otro tipo de seleccionar una alternativa de diseño sobre otra. Los compromisos son un hecho fundamental de la vida en todos los niveles y en todas las áreas temáticas.	• *compromisos de espacio-tiempo en el estudio de algoritmos,* • *compromisos inherentes a objetivos de diseño en conflicto (por ejemplo, facilidad de uso versus integridad, flexibilidad versus simplicidad, bajo costo versus alta confiabilidad, etc.),* • *compromisos de diseño en hardware y* • *compromisos implícitas en los intentos de optimizar la potencia informática frente a una variedad de limitaciones.*

¿Para qué?


Modelo
Espacio
Orden
Tiempo
Compromiso
Enlace
Complejidad
Eficacia
Eficiencia
Abstraccion
Seguridad
Reusabilidad
Evolucion

¿Cómo?

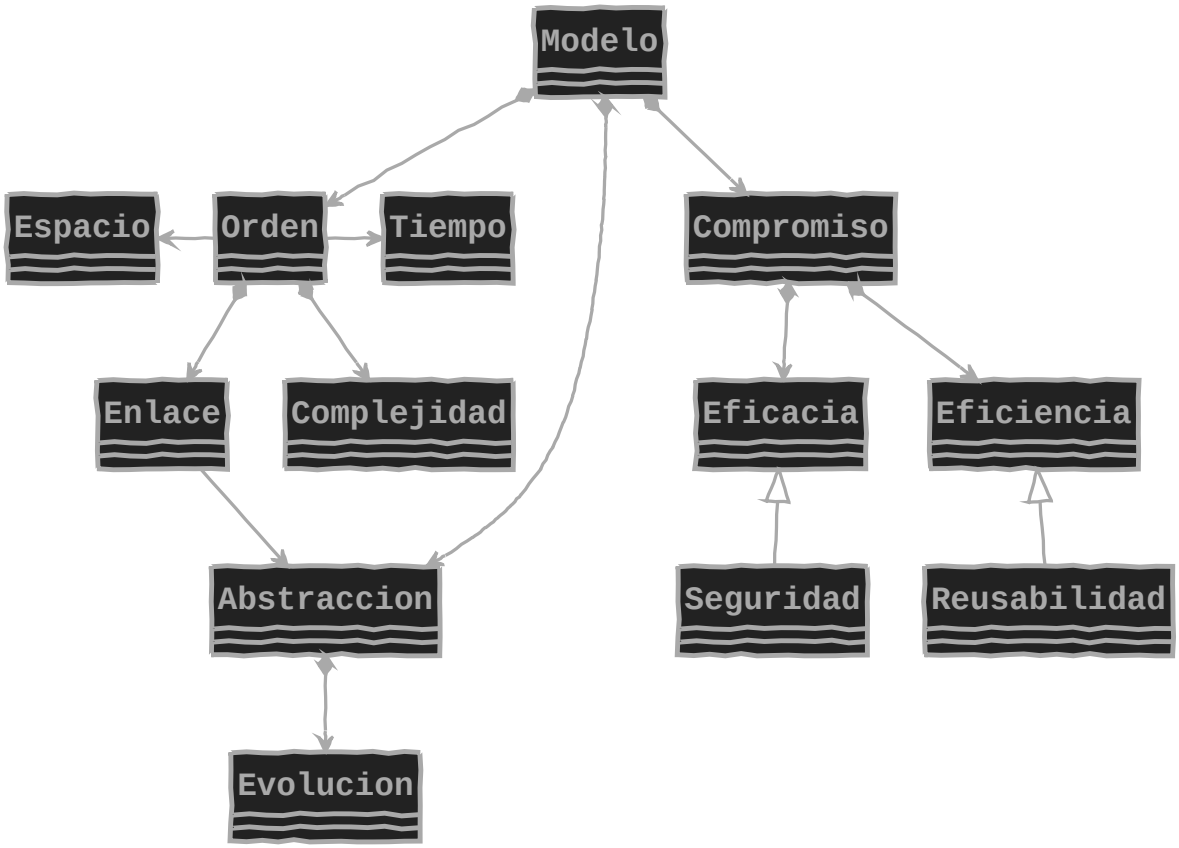
Itinerario



- **1. Patrones**
  - Reusabilidad
  - Modelos
  - Abstracción
- **2. Concocimiento**
  - Orden
  - Enlace
  - Coherencia
  - Seguridad
  - Evolución
- **3. Complejidad**
  - Complejidad
  - Eficiencia
  - Compromisos
- **4. Software**
- **5. Disciplinas del Software**
- **6. Evolución del Software**

Sintesis


UNIVERSO SANTA TECLA
Patrón
Conocimiento
Datos
Información
Sencillez
Compuesto
Disciplina
Reusabilidad
Software
Síntesis
Calidad
Proyecto
Clases
Objetos
Complejidad necesaria
Compromiso
Orden
Ámbito
Diseño
Unidad
Eficiencia
Eficacia
Análisis
Coste
Tiempo
Grady Booch
Kent Beck
Alan Turing
Charles Babbage
Ada Lovelace


Modelo
Espacio
Orden
Tiempo
Compromiso
Enlace
Complejidad
Eficacia
Eficiencia
Abstraccion
Seguridad
Reusabilidad
Evolucion

Bibliografía

Obra, Autor y Edición	Portada	Obra, Autor y Edición	Portada

Ponente

• ***Luis Fernández Muñoz*** ◦ *Formador/Consultor* ◦ *Linkedin*		• *Doctor en Inteligencia Artificial por la UPM* • *Ingeniero en Informática por la UMA* • *Diplomado en Informática por la UPM* • *Profesor Titular de ETSISI de la UPM*