

UNIDAD I

Métodos de Desarrollo de Software

Agenda



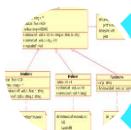
Ingeniería de Software



Modelos de Procesos – Ciclo de vida del Software



Métodos tradicionales



Paradigmas de la orientación a objetos



Proceso de Desarrollo de Software



Análisis de requisitos
(Requerimientos)



UNIVERSIDAD
DE LIMA



Definición Hardware - Software

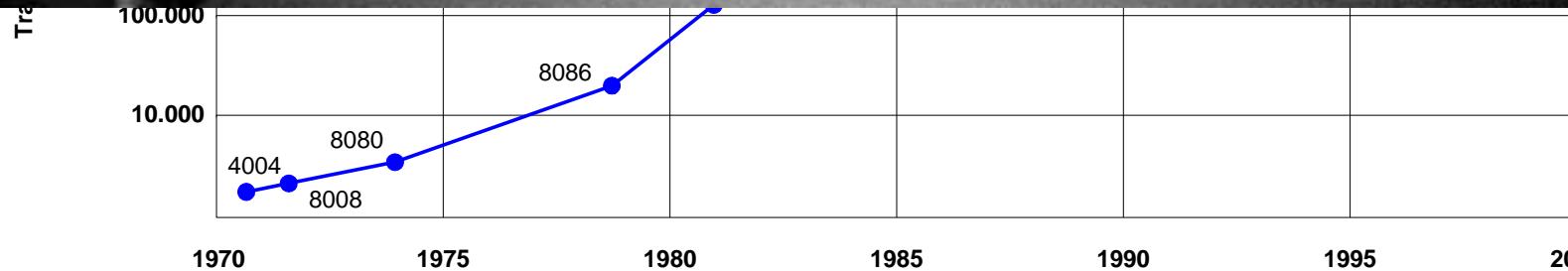
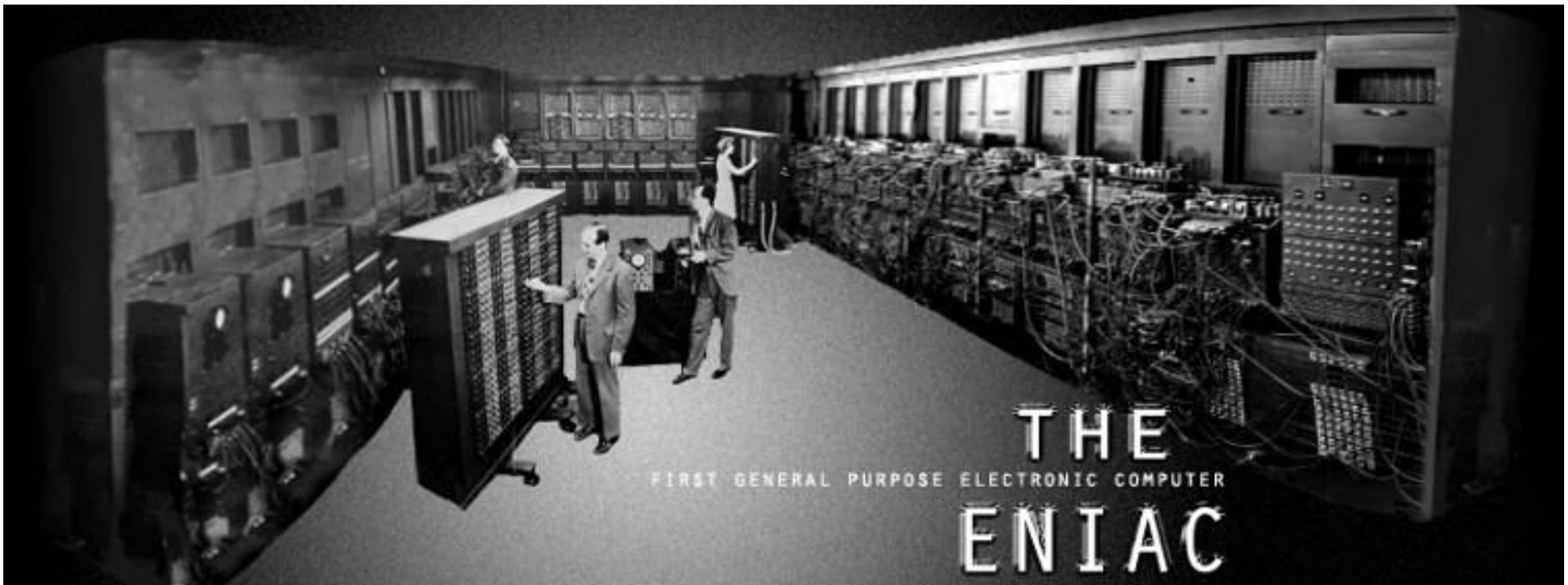
- El **hardware** es el conjunto de todos los elementos materiales como los dispositivos electrónicos y electromecánicos que pueden incluir
- En cambio, el **software** es intocable, existe como ideas, conceptos, símbolos, pero no tiene sustancia.

Una computadora sin software sería tan inútil como un libro con páginas en blanco

Desarrollo del hardware

- La aparición de componentes que cada dos años doblan la capacidad de sus antecesores (MTOPS).
- En **1946 ENIAC** ocupaba una superficie de 160 m², pesaba 30 toneladas, y ofrecía una capacidad de proceso de 30.000 instrucciones por segundo.
- En **2002** la Pentium IV a 2 Ghz ocupa una superficie de 217 mm² y tiene una capacidad de proceso de 5.300 MTOPS
- En la actualidad son cuatro los factores que imprimen un ritmo acelerado a la industria del hardware.

Desarrollo del hardware



PRODUCTION

Software gap—a growing crisis for computers

Shortage of programmers—and the fruits of their solitary art—is stunting growth of computer use and costing industry hard cash

The computer, man's most complex industrial product, can be cranked out in quantity by mass-production techniques. But it is powerless to solve problems, sort data, or store information without instructions.

The process of writing instructions—or programs—is a new human intellectual art, not a mechanical or electronic skill. And this factor is setting limits on the usefulness of the computer far below those imposed by electronic technology.

David B. Hertz, a consultant at McKinsey & Co., summed up the problem at an American Management Assn. meeting earlier this year: "The overriding issue is people—specifically, skilled computer personnel . . . Already, the supply is far short of the demand, and the gap is widening inexorably. For the foreseeable future, there is literally no possibility that we shall have enough trained people to go around."

The implication of this gap for business and science, he added, is that "use of computer systems five years hence will be seriously hobbed." There are only about 120,000 programmers in the U. S.—and right now there's a shortage of 55,000 or more of these new professionals, according to some estimates.

Special type. Programming is a young profession, born only a dozen years ago when computer makers first began to turn the fuzzy outlines of business problems into the precise electronic language of the machine. Since then, the number, power, and widespread application of computers has far outstripped the supply of programmers. Delays and extra costs have been the price to both users and computer makers.

To top it off, the computer has gotten so complex it is demanding



BUSINESS WEEK November 5, 1966

PRODUCTION 127

"Software Gap: A Growing Crisis for Computers", Business Week, 5 de noviembre de 1966,

Wanted: 500,000 Men to

By Stanley L. Englehardt

IF YOU know how to "talk to computers," chances are you've got it made. If you don't, you may be missing out on a great job opportunity.

People who talk to computers are called programmers. They instruct data-processing machines on how to perform specific jobs. Today there are about 40,000 of these specialists at work. In six years, experts say, 500,000 more will be needed. Many will require a

bachelor's, master's, or even doctor's degree. But close to 50 percent will move into this new profession with only high-school diplomas.

Here's why there's such a tremendous demand for programmers.

Computers are really very stupid multimillion-dollar collections of wires and transistors. Plug one in and it does nothing. Yell at it, curse, kick it—and still it remains mute. The reason: no instructions.

But once people write instructions,

You don't have to be a college man to get a good job in computer programming—today even high-school grads are stepping into excellent jobs with big futures

Feed Computers

the computer becomes a marvelous tool. It can tell the exact moment at which an astronaut should fire his retrorockets, or identify an obscure disease and prescribe a course of treatment. It can keep watch over huge inventories and write reorders when the stock gets low. Computers can prepare your paycheck, update accounts-receivable files—even print out past-due notices when you're late in paying bills.

Thousands of new computers are installed each year to do these jobs. Each

one must be programmed before it can start processing. This means anywhere from 1 to 100 people sitting down to figure out every possible step in a particular operation. These steps are translated into machine language, punched into cards, and fed into the computer. There they are stored for use during the solution of a problem.

Do you have what it takes to be a programmer?

Education is important, but most important is a quick mind, with the ability

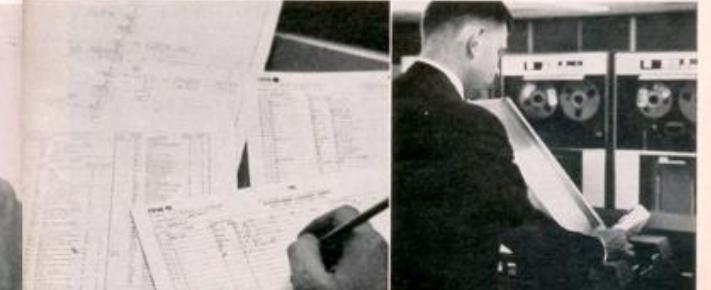
The computer programmer's daily work



1 From his boss, a systems analyst, the programmer receives his assignment to work out a program for one section of a job planned on a flow chart.

106 POPULAR SCIENCE JANUARY 1965

2 First step: Draw a block diagram showing basic data-handling and logic operations computer must perform. Standard symbols are used in the diagram.



3 Consulting a special dictionary, programmer spells out instructions in a "programming language"—a code describing standard sequences of machine operations.

4 Coded instructions, now punched into cards, are fed into a computer, which automatically translates steps into precise instructions it can follow.

107

Gene Bylinsky, "Se busca ayuda: 50,000 programadores", Fortune 75, no. 3 (1967)

UNIVERSIDAD
DE LIMA

años
60

Crisis del software

- El problema se *identificó por primera vez en 1968*,
- año en el que la organización NATO desarrolló la primera conferencia sobre desarrollo de software.
- En esta conferencia se acuñaron los términos “**crisis del software**” para definir a los problemas que surgían en el desarrollo de sistemas de software, e “**ingeniería del software**” para describir el conjunto de conocimientos que existían en aquel estado inicial.

Crisis del software

¿Qué se tenía? :

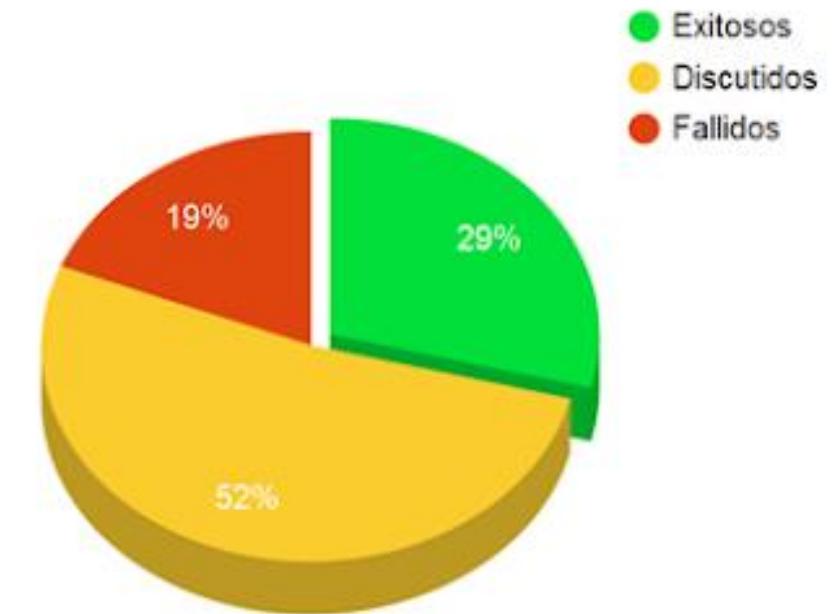
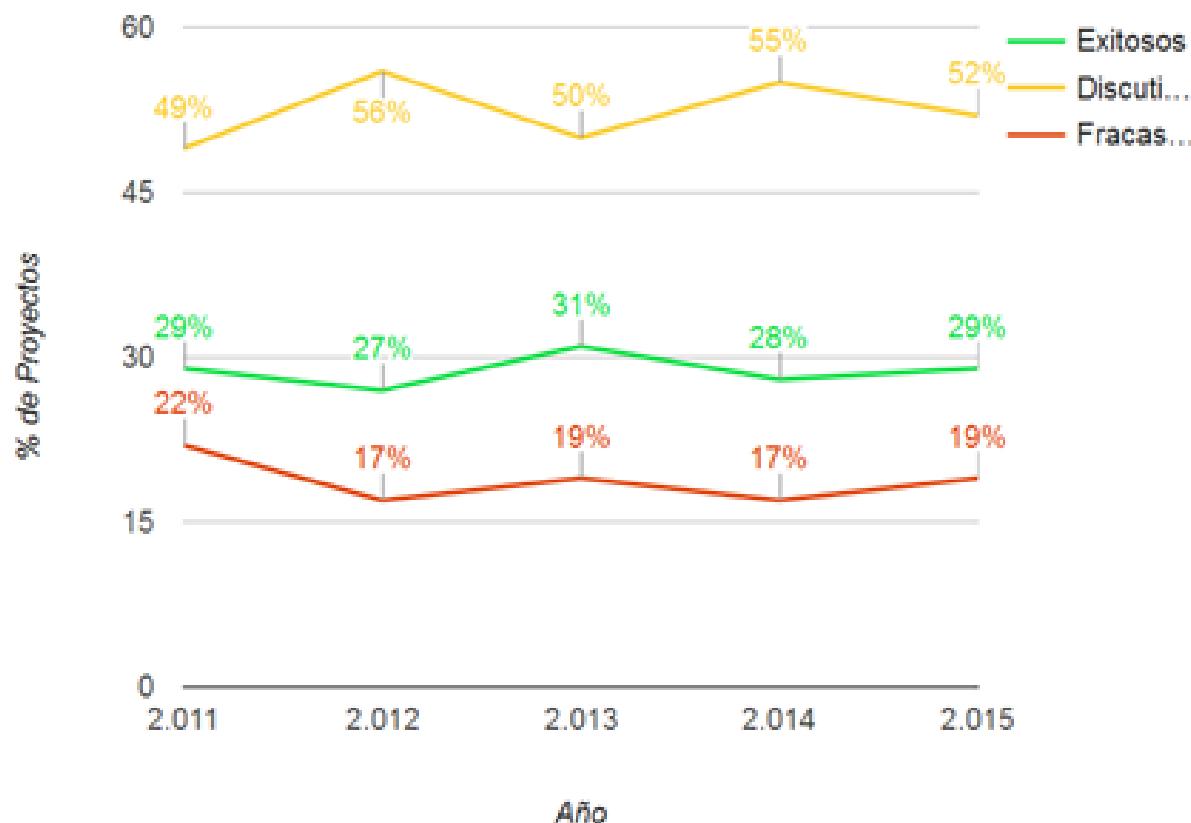
- En **1962** se publicó el primer algoritmo para búsquedas binarias.
- C. Böhm y G. Jacopini publicaron en **1966** el documento que creaba una fundación para la eliminación de "GoTo" y la creación de la programación estructurada.
- Edsger Dijkstra escribió su famosa carta "GoTo Statement Considered Harmful" en **1968**.
- La primera publicación sobre programación estructurada no vio la luz hasta **1974**, publicada por Constantine, Glenford y Wayne
- El primer libro sobre métrica de software fue publicado en **1977** por Tom Gilb.
- El primero libro sobre análisis de requisitos apareció en **1979**





Crisis del software

- Proyectos para el desarrollo de sistemas de software



* publicado por [Standish group](#)

¿ Cómo se manifiesta la crisis ?



Proyectos gestionados que exceden el presupuesto.



Proyectos gestionados con sobre tiempo



Software de baja calidad.



El software a menudo no satisface los requerimientos deseados



Los proyectos son inmanejables, con un código difícil de mantener.

La crisis del software fue dirigida por la implementación de varios procesos y metodologías, siendo la más notable el modelo de cascada de Royce

Ingeniería de Software

La Ingeniería del Software es una ingeniería muy joven que necesitaba:

- **Definirse a sí misma:** ¿Cuáles son las áreas de conocimiento que la comprenden?

→ **SWEBOK: Software Engineering Body of knowledge**

- **Definir los procesos que intervienen en el desarrollo, mantenimiento y operación del software**

→ **ISO/IEC 12207: Procesos del ciclo de vida del software**

- **De las mejores prácticas, extraer modelos de cómo ejecutar esos procesos para evitar los problemas de la “crisis del software”**

→ **CMM / CMMI
ISO/IEC TR 15504**

- **Definir estándares menores para dibujar criterios unificadores en requisitos, pruebas, gestión de la configuración, etc.**

→ **IEEE 830 - IEEE 1362 - ISO/IEC 14764 ...**



¿Qué es la Ingeniería de software ?

- Según la definición de la IEEE (1993), es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software.
- Ofrece métodos o técnicas para desarrollar y mantener software de calidad que resuelven problemas de todo tipo,
- El concepto de ingeniería del software surgió en 1968, tras una conferencia en Garmisch (Alemania) que tuvo como objetivo resolver los problemas de la crisis del software.

¿Qué es Ingeniería de Software?

- **Definición original:** “Establecimiento y uso de principios de ingeniería para obtener software económico que trabaje de forma eficiente en máquinas reales”. Fritz Bauer, 1968 (conferencia NATO)
- 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology
<http://www.idi.ntnu.no/grupper/su/publ/ese/ieee-se-glossary-610.12-1990.pdf> página 67

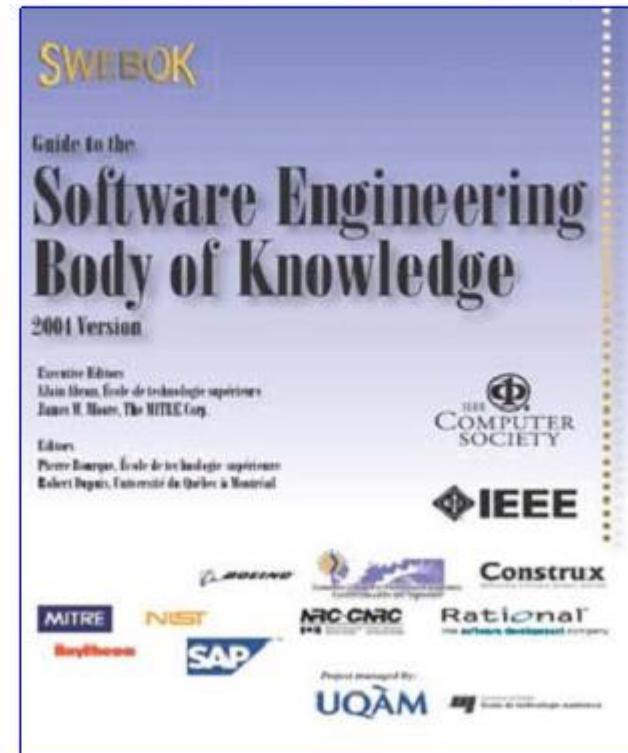
“The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.”

Peculiaridades de la Ingeniería del Software

- El “producto” es un software.
- Mucho desarrollo, poca disciplina ingenieril.
- Necesidad de describir y documentar lo que se va a producir.
- Cambios frecuentes en el producto.

SWEBOK

- Software Engineering Body of Knowledge
 - <http://www.swebok.org/>



SWEBOK

- La IEEE creó en **Mayo de 1993** su comité para la coordinación de la ingeniería de software (Software Engineering Coordinating Committee) dedicado **evaluar, planear y coordinar** acciones relacionadas para establecer la Ingeniería de Software como una profesión.
- Este comité publicó en 2001 la **Guia del Cuerpo de Conocimiento de Ingeniería de Software** (Guide to the Software Engineering Body of Knowledge) o SWEBOK

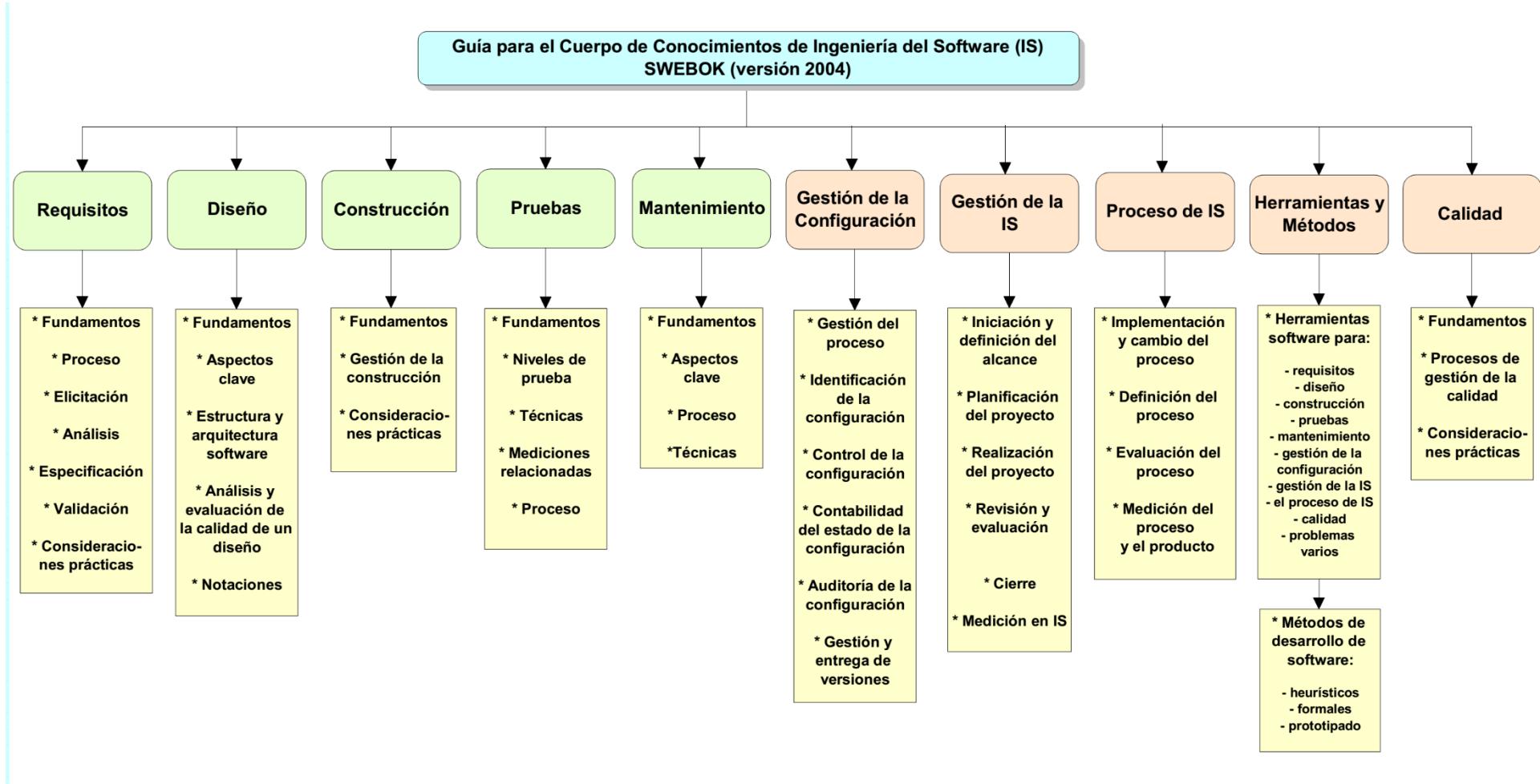


SWEBOK

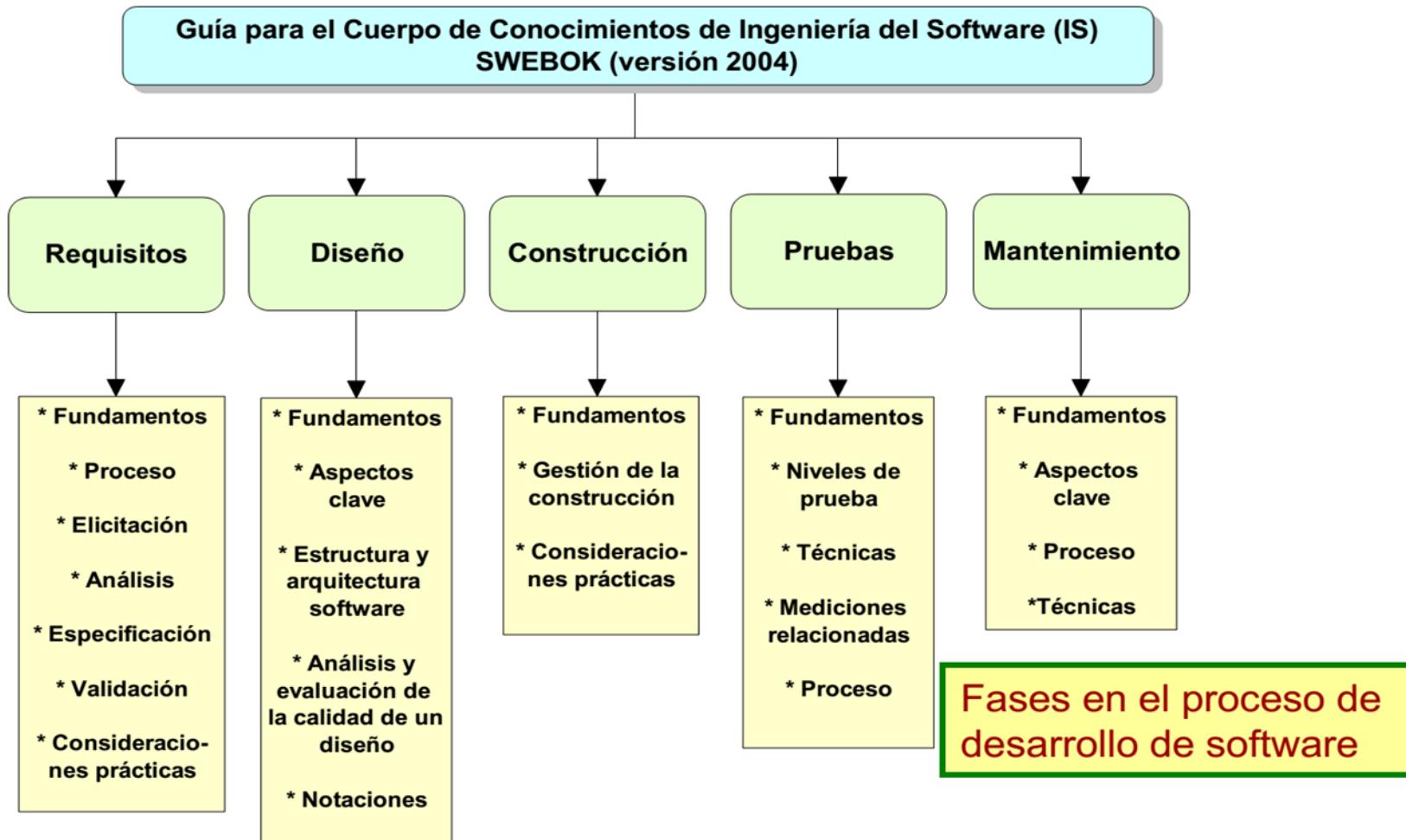
- Este documento tiene como propósito **proveer un consenso** sobre los límites de la ingeniería de software y acceso al cuerpo de conocimiento de la disciplina
- El cuerpo de conocimiento de la ingeniería de software se **divide en áreas** de conocimiento (*Knowledge area* o KA)



Cuerpo de Conocimientos - SWEBOK

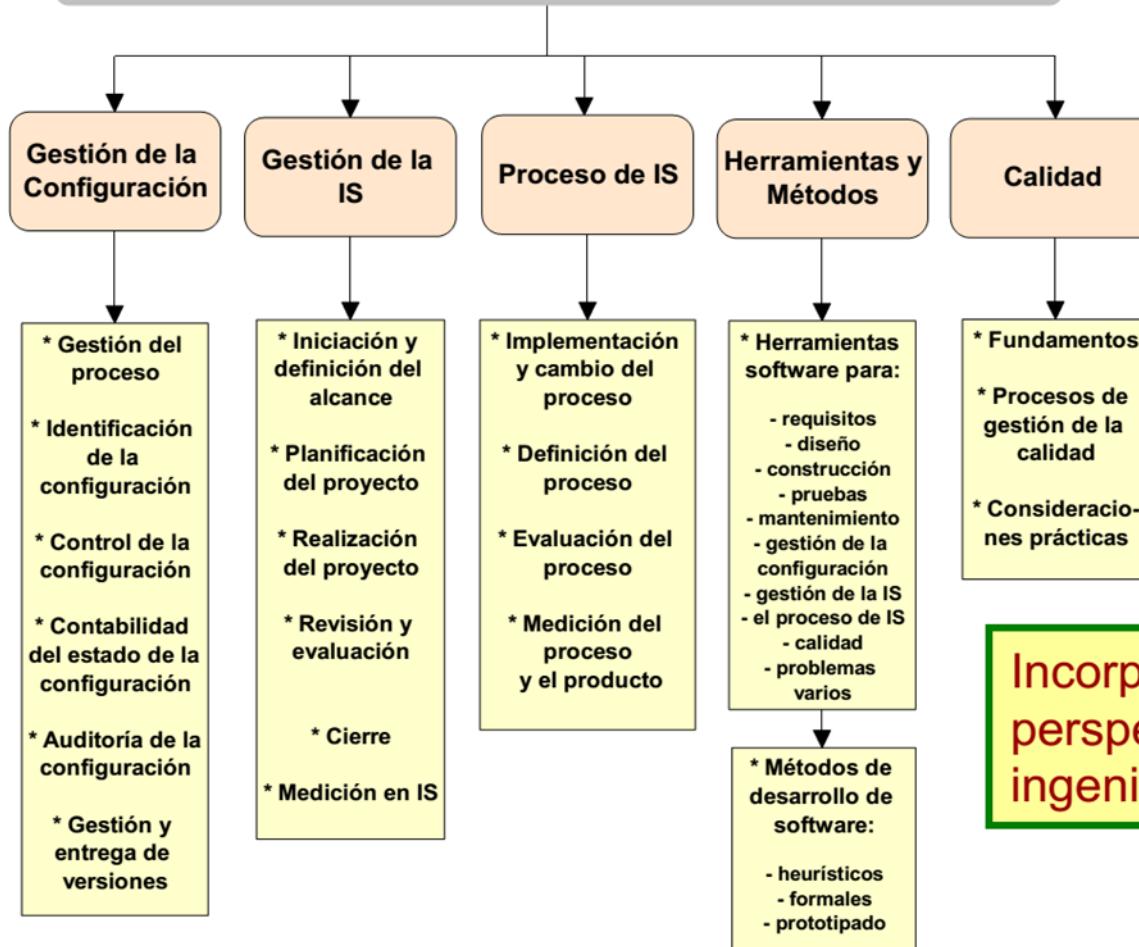


Cuerpo de Conocimientos - SWEBOK



Cuerpo de Conocimientos - SWEBOK

Guía para el Cuerpo de Conocimientos de Ingeniería del Software (IS)
SWEBOk (versión 2004)



- Gestión de la Configuración (gestión de productos)
- Gestión de la Ingeniería (gestión de proyectos)
- Proceso de Ingeniería (orientación a procesos)
- Herramientas y Métodos (tecnología de soporte)
- Calidad

Incorporación de la perspectiva de ingeniería

Areas de conocimiento v3

- 1 Requerimientos de software
- 2 Diseño de software
- 3 Construcción de software
- 4 Pruebas del software
- 5 Mantenimiento del software
- 6 Calidad de Software
- 7 Administración de la configuración
- 8 Administración de la Ingeniería de Software
- 9 Proceso de la Ingeniería de Software
- 10 Herramientas y métodos de la Ingeniería de Software

Áreas nuevas:

- Práctica profesional de la Ingeniería de Software
- Economía de la Ingeniería de Software
- Fundamentos de Computación
- Fundamentos matemáticos
- Fundamentos de Ingeniería

* En la edición 2014 existen 15 áreas de conocimiento.

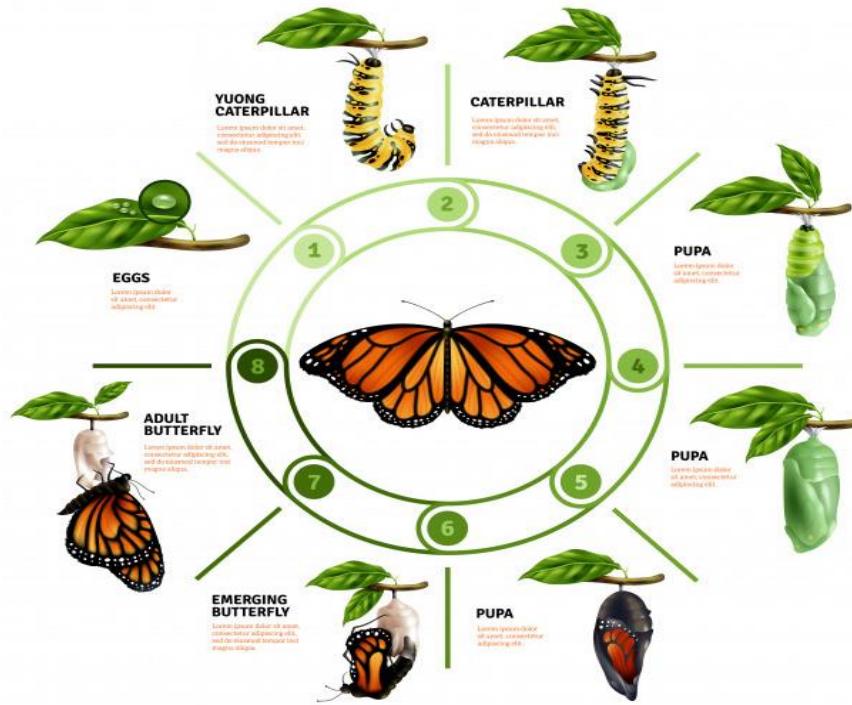
¿ciclo de vida?
¿ciclo de desarrollo?



UNIVERSIDAD
DE LIMA



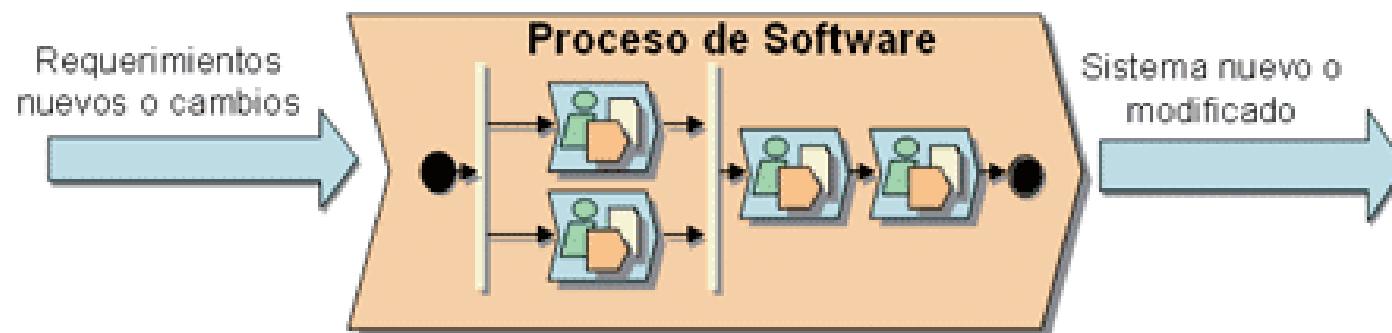
Ciclo de vida



Describe la vida de un producto de software desde su **definición**, pasando por su diseño, implementación, verificación, validación, entrega, y hasta su **operación y mantenimiento**

Proceso de Desarrollo de Software

- El SDP define el **qué, quién, cuándo y cómo** del desarrollo de software.
- Hay 4 actividades fundamentales que son comunes para todos los procesos de desarrollo de software :
 - La Especificación del software
 - El Desarrollo del software
 - La Validación del software
 - La Evolución del software



¿Cuál es la diferencia entre procesos desarrollo software y la metodología de desarrollo software?

- **El proceso de Software**, es el conjunto de actividades secuencial para realizar el desarrollo del software, tiene como propósito la producción eficaz y eficiente de un producto software que reúna los requisitos del cliente
- **La metodología de desarrollo software**, Metodología: Conjunto de estrategias, procedimientos, métodos o actividades que permitan el logro de un objetivo

Una metodología es el proceso de software junto a otros elementos.

¿Cuál es la diferencia entre procesos desarrollo software y la metodología de desarrollo software?

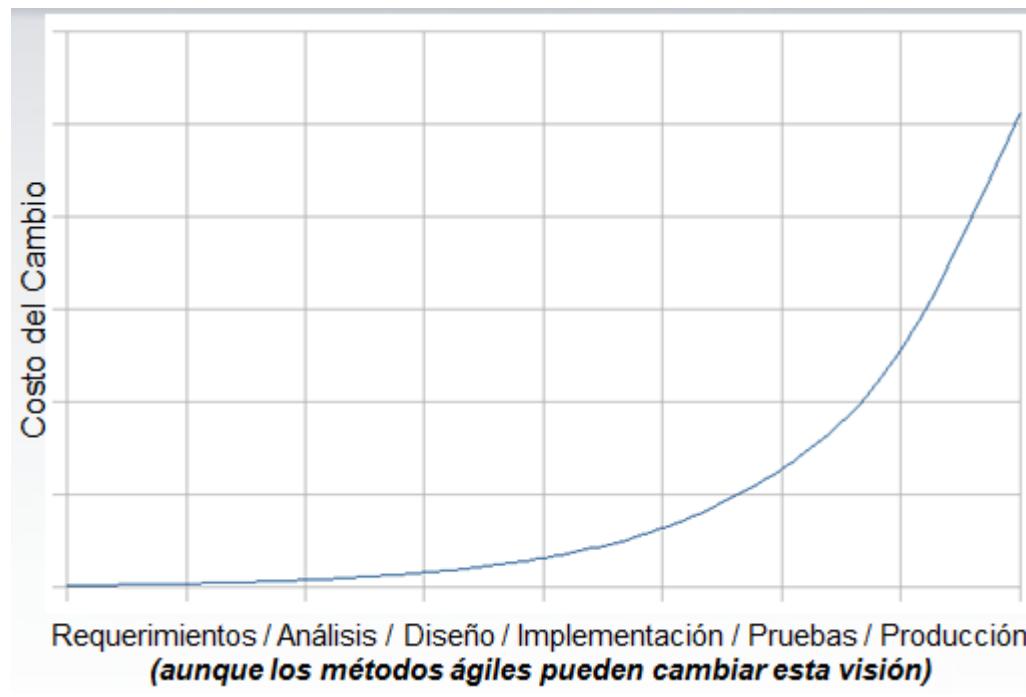


¿ por qué se requiere métodos para desarrollar software?

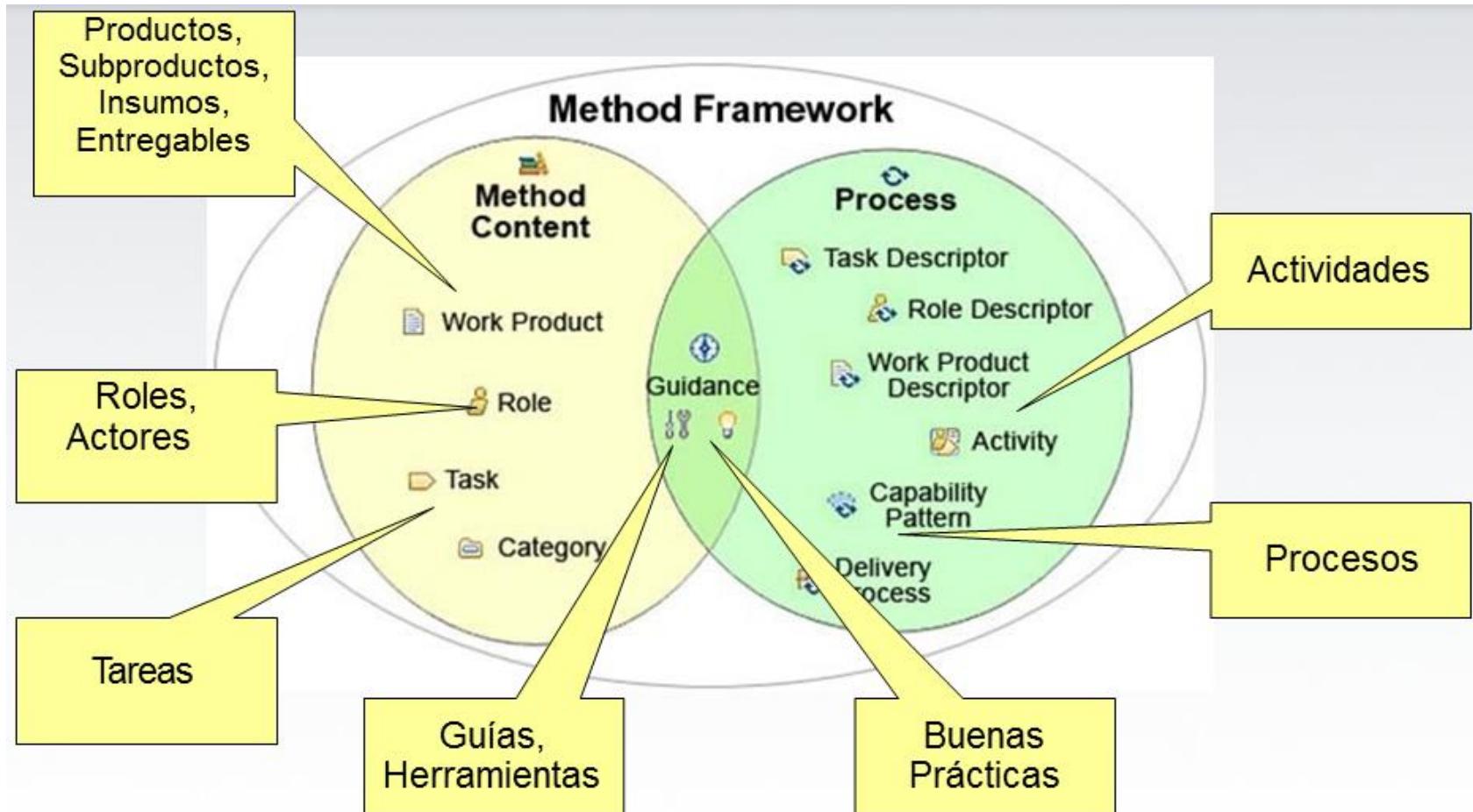


¿ Por qué se requieren métodos ?

- por lo complejo que es desarrollar software
- por el costo de los cambios,
- Por la naturaleza del software y otras razones



¿ qué aportan los métodos ?



y otros elementos adicionales...

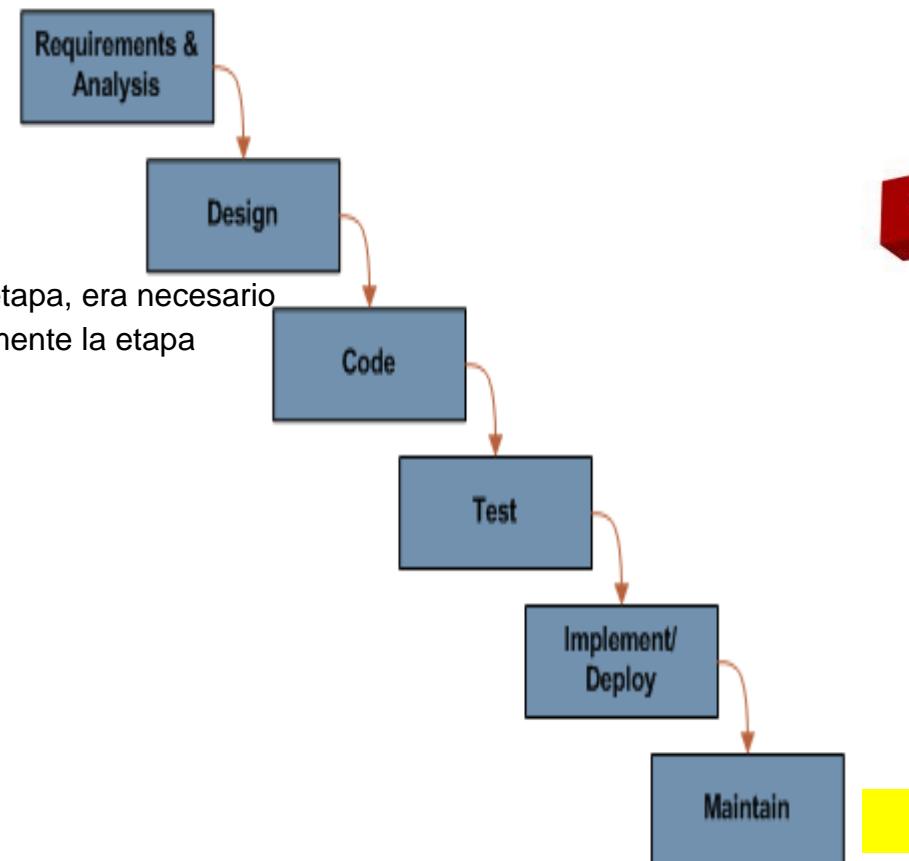
UNIDAD I: **Proceso de Desarrollo de Software**

¿ qué modelos de proceso existen ?

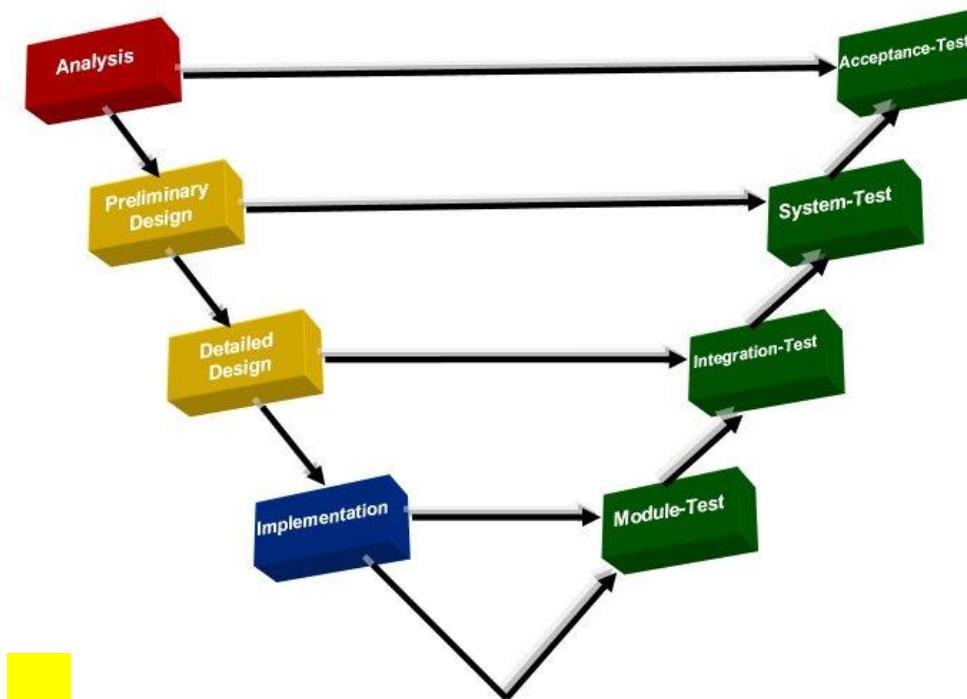


Modelo de proceso: Descripción simplificada (abstracción) de un proceso de desarrollo de software real.

Métodos tradicionales



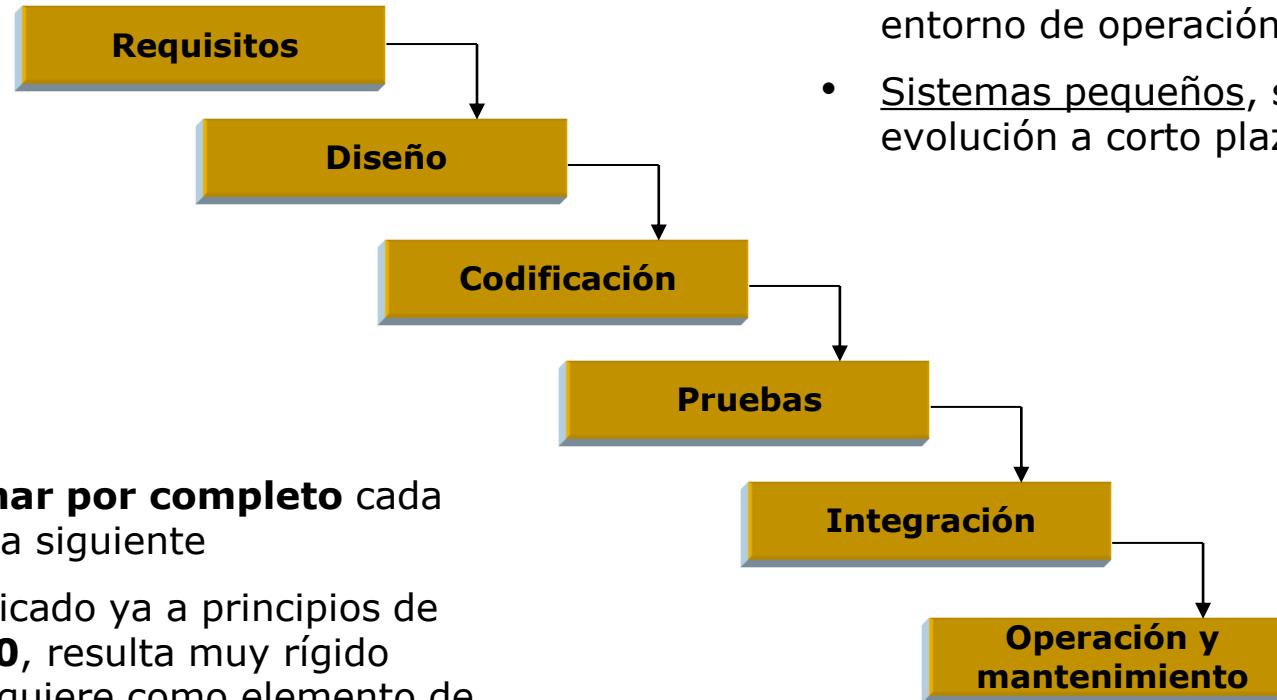
Waterfall Model –
Modelo cascada (Royce, 1970)
<http://www.waterfall-model.com/>



V- Waterfall Model –
Modelo V (Brook, 1986)
<http://www.waterfall-model.com/>

Lineal o secuencial

- Es necesario **terminar por completo** cada etapa para pasar a la siguiente
- Este modelo, identificado ya a principios de la **década de los 50**, resulta muy rígido porque cada fase requiere como elemento de entrada el resultado completo de la anterior.

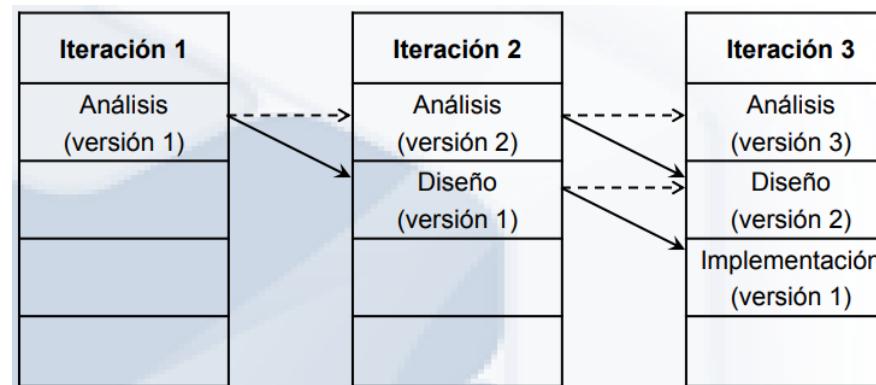


Resulta apropiado para:

- Desarrollar nuevas versiones de sistemas ya veteranos en los que el desconocimiento de las necesidades de los usuarios, o del entorno de operación no plantea riesgos.
- Sistemas pequeños, sin previsión de evolución a corto plazo.

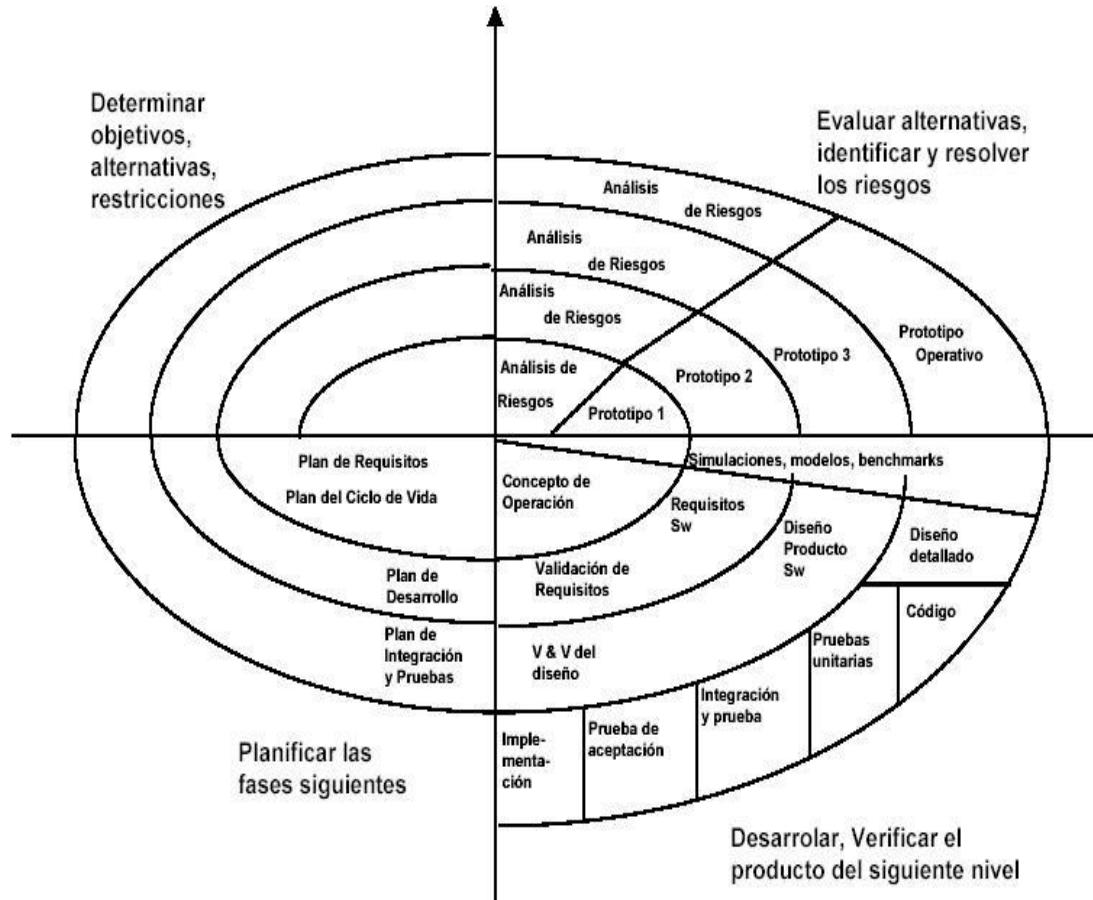
Modelos Evolutivos

- Son modelos iterativos, permiten desarrollar **versiones cada vez más completas y complejas**, hasta llegar al objetivo final.
- Los modelos “**Iterativo Incremental**” y “**Espiral**” son dos de los más conocidos y utilizados del tipo evolutivo.
- Una ventaja es que se obtiene una **rápida realimentación del usuario**, ya que las actividades de especificación, desarrollo y pruebas se ejecutan en cada iteración.
- Posibilita la **evolución en paralelo** de los distintos flujos de trabajo, y por tanto el trabajo en paralelo de distintos equipos de personas.
- Las distintas versiones de los **documentos producidas en cada iteración no necesariamente son compatibles entre sí**: se debe organizar bien la documentación,



<http://alistair.cockburn.us/Incremental+versus+iterative+development>

Espiral



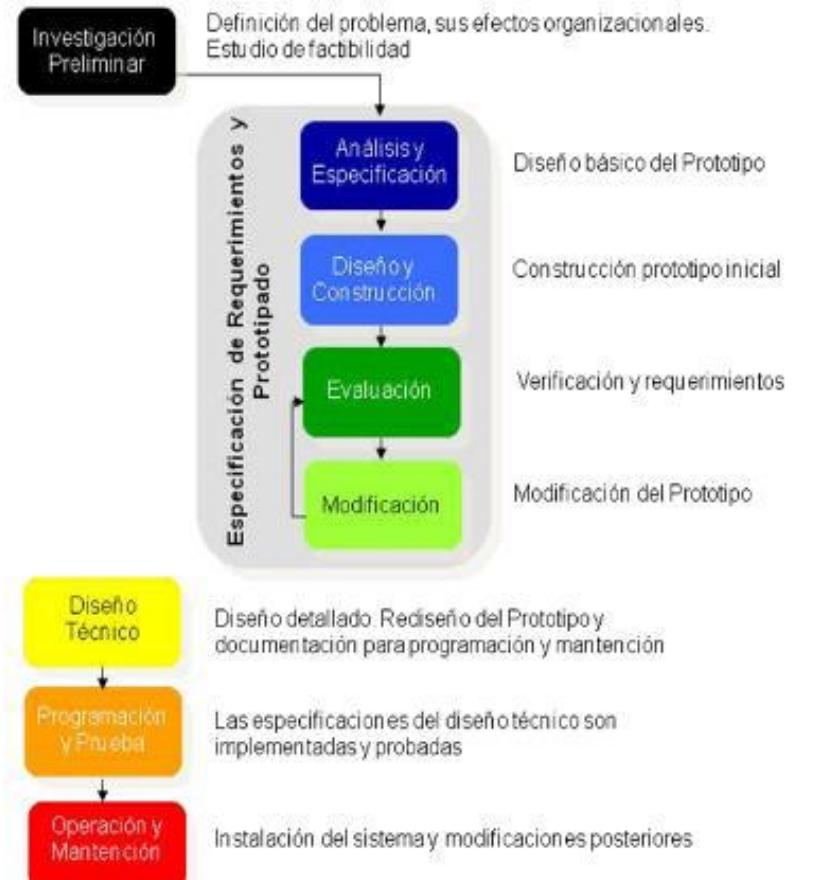
- Este modelo, definido por **Boehm en 1988**
- presenta un **desarrollo evolutivo**
- Introduce como elemento distintivo la actividad de **“análisis de riesgo”** para guiar la evolución del proceso de desarrollo.
- En **cada ciclo de la espiral se realiza una parte del desarrollo total**, a través de los cuatro tipos de actividades.
- En la planificación de cada vuelta se establece el contexto del desarrollo y se decide qué parte del mismo se abordará en el ciclo
- Este modelo permite múltiples combinaciones ya que en la planificación de cada ciclo se determina el avance que se va a ejecutar durante la vuelta.
- Si sólo se determina dar un pequeño paso, y después de conseguido, evaluar el resultado y planificar el siguiente paso, y antes de cada ejecución se analizan los riesgos, en ese caso, el modelo seguido es un modelo en espiral

Modelo de Prototipos

- Propuesto por **Gomma en 1984**
- Un prototipo **es una versión no terminada del producto** que se le entregará al cliente o usuario final.
- Sirve para identificar requisitos del software.
- El diseño rápido se centra en una representación de aquellos aspectos del software que serán visibles para el cliente: este diseño conduce a la creación de un prototipo el cual **es evaluado por el cliente** para luego dar su retroalimentación.
- El **cliente final se involucra mucho más** en el proyecto que con otras metodologías, haciendo de esta forma que el producto final llegue rápidamente aunque con un poco más de presión en el proceso

FASES:

- Planeación.
- Modelado.
- Elaboración del Prototipo.
- Desarrollo.
- Entrega y Retroalimentación.
- Comunicación con el Cliente.
- Entrega del Producto Final.



TRADICIONAL



AGILE



Modelos Agiles

- A diferencia de las metodologías tradicionales, funcionan mas como una combinación de estas para lograr un objetivo. Su finalidad siempre será el crear software de una forma más rápida.
- Consiste principalmente en trabajar con menos documentación que las metodologías tradicionales .

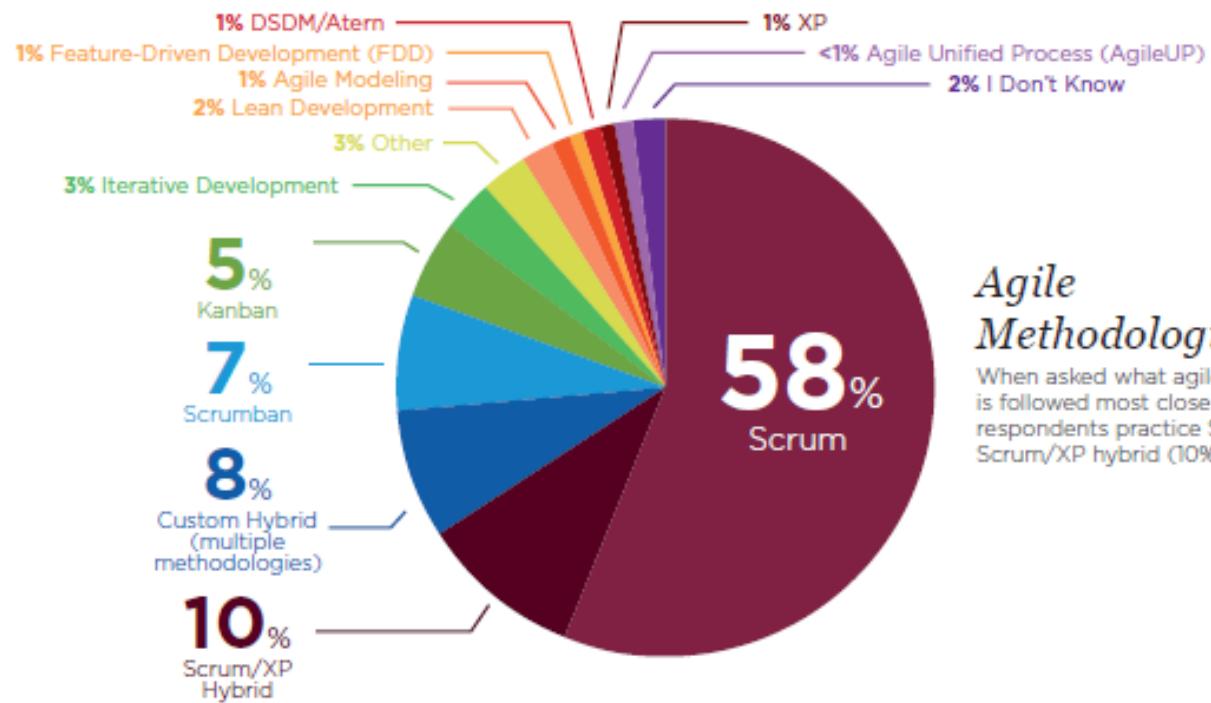
Manifiesto ágil:

1. Se prioriza al Individuo y las Interacciones del Equipo
2. Software funcional en lugar de demasiada documentación
3. Colaboración con el Cliente en lugar de Contratos
4. Posibilidad de hacer cambios de planes a medio proyecto



Uso de los métodos

AGILE METHODS AND PRACTICES



Agile Methodologies Used

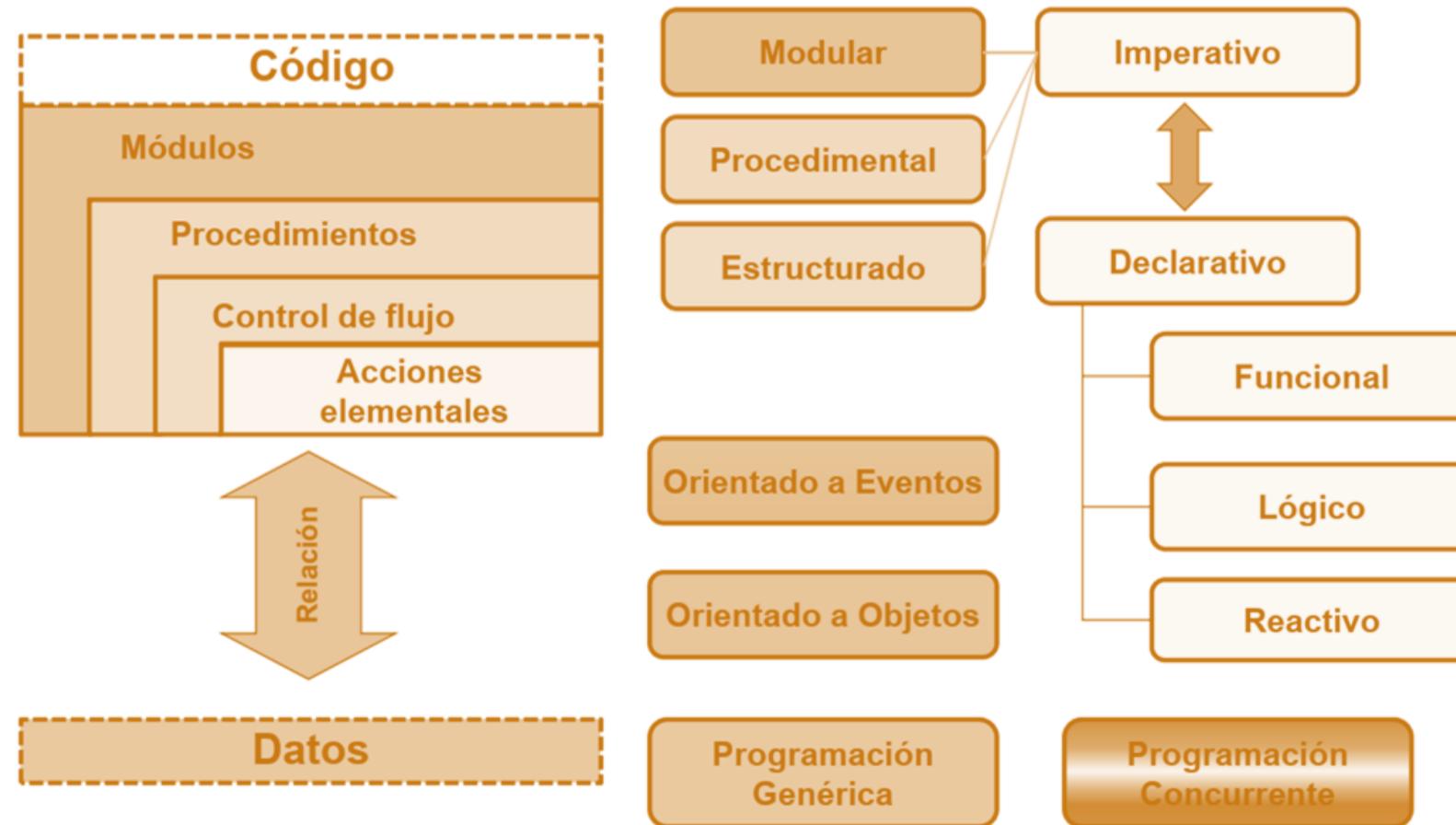
When asked what agile methodology is followed most closely, nearly 70% of respondents practice Scrum (58%) or Scrum/XP hybrid (10%).

En el gráfico siguiente, se puede observar la tendencia (2017) en cuanto al uso de metodologías ágiles.



Orientación a objetos

Paradigma de programación



Paradigmas de la orientación a objetos

Problemas:

- Pocos sistemas lograban terminarse
- Pocos se terminaban con los requisitos iniciales
- No todos los que se terminaban y cumplían con los requisitos, se usaban según lo planificado

“Adaptar el software a nuevos requerimientos imposibles de haber sido planificados inicialmente, era imposible”

¿Qué es la Orientación a Objetos?

- El concepto surge en los **lenguajes de programación**
 - Se organiza el software como una colección de objetos discretos que encapsulan
- **Estructuras de Datos y Comportamiento.**
 - Un sistema OO funciona mediante la colaboración entre los objetos que se comunican entre sí.
- El concepto se extiende a los **métodos de análisis y diseño**
 - Se utilizan los objetos del mundo real como base para construir modelos
 - Los elementos que forman los sistemas del mundo real se corresponden con objetos software



- **Características:**
 - Raza
 - Color
 - Estatura
 - Edad
 - Etc.
- **Acciones posibles:**
 - Comer
 - Dormir
 - Sentarse
 - Ladrar
 - Etc.

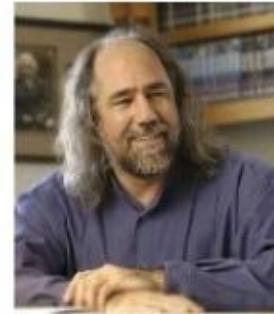
Recordemos conceptos básicos

- Clase y Objeto
 - Atributos
 - Operaciones
 - Comportamiento
 - Identidad
- Interfaz
- Asociación
- Agregación
- Herencia
- Polimorfismo
- Paso de mensajes
- Encapsulación



Unified Modeling Language (UML)

- Estándar de la **OMG** (Object Management Group) para el modelado de sistemas software
- Es **ampliamente usando** en la industria del software y existen muchas herramientas.
- Define hasta **14 tipos de diagramas** para modelar
- Basado en el trabajo de “**los 3 amigos**”



Grady Booch



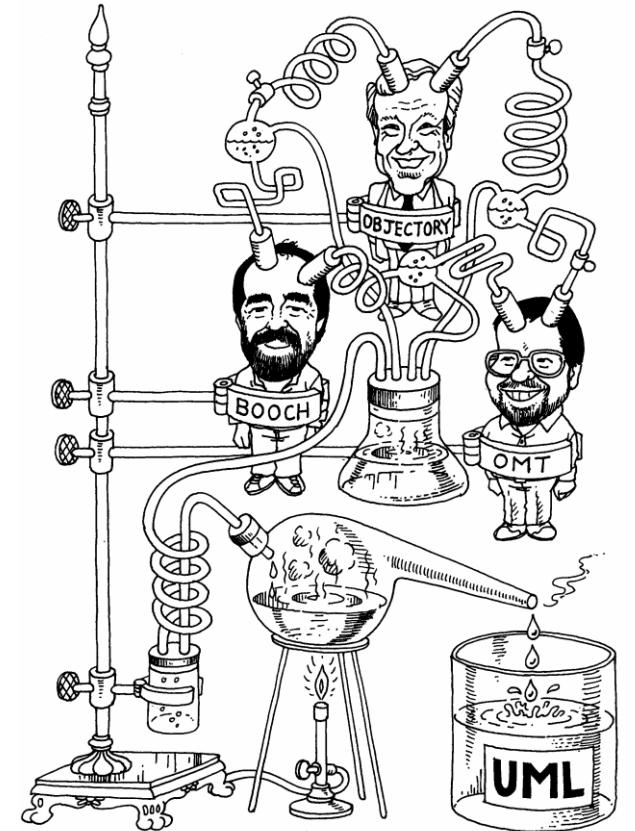
James Rumbaugh



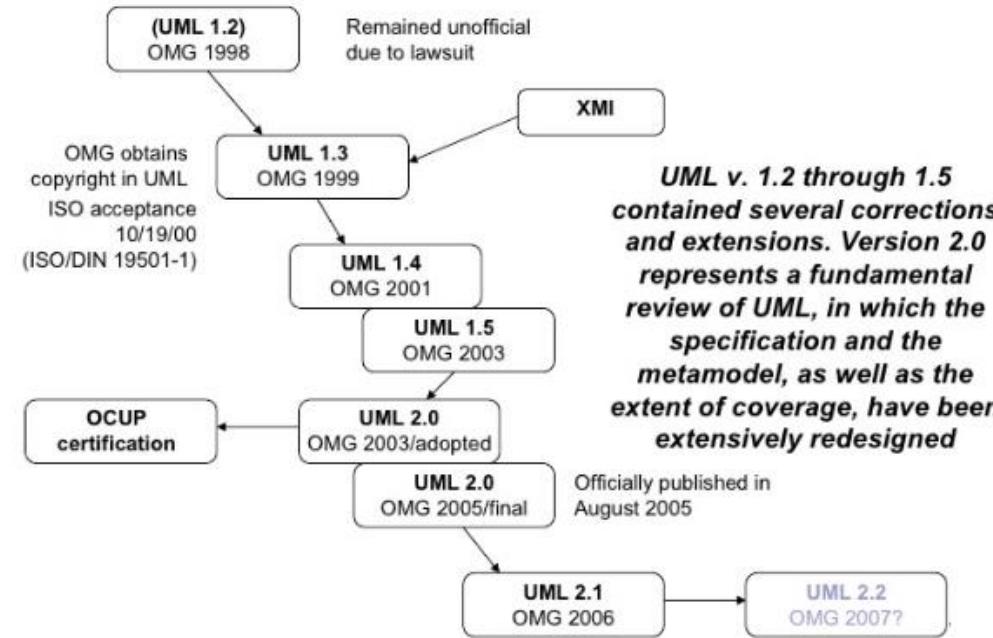
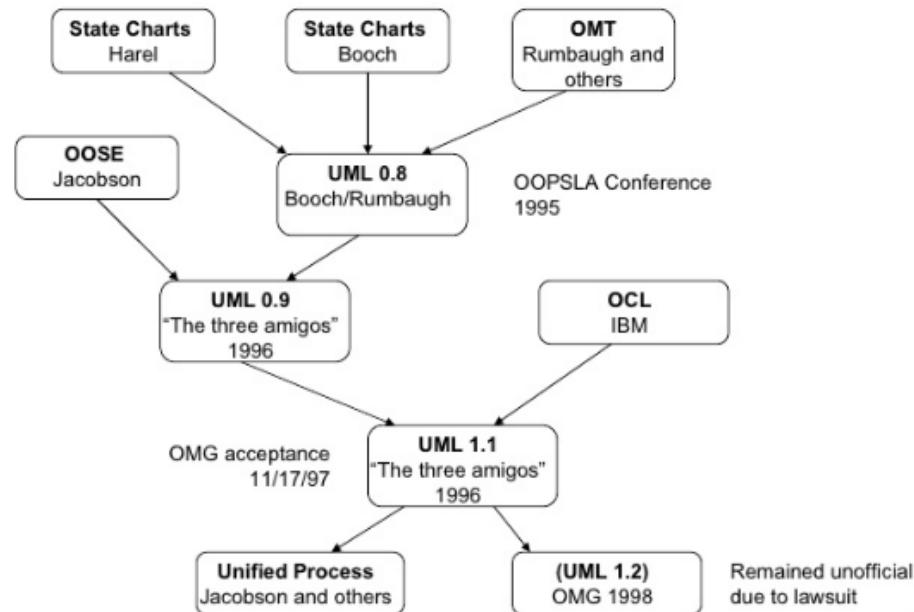
Ivar Jacobson

Unified Modeling Language (UML)

- En **1994** Grady Booch manifiesta la necesidad de unificar criterios
- James Rumbaugh se une a Booch en octubre de ese año
- Ambos elaboran la **versión 0.8** del Unified Method
- En **1995** Ivar Jacobson completa el trío de “amigos” y cambia el enfoque: **nace el UML**



Historia



Versión actual 2.5.1 (Dic 2017)

Modelado Orientado a Objetos con UML

Qué es UML(Unified Modeling Language)?:

- Lenguaje de Modelado Unificado.
- Es un **lenguaje** estándar para escribir planos (modelos) de software.
- Utilizado para **expresar gráficamente** el proceso de generación de software.
- UML **es independiente** del lenguaje de implementación del software.



Modelado Orientado a Objetos con UML

- **Lenguaje:** Proporciona la sintaxis, vocabulario y las reglas necesarias para la representación conceptual y física de un sistema software.
- **Modelado:** El UML es visual. Mediante su sintaxis se modelan distintos aspectos del mundo real, que permiten una mejor interpretación y entendimiento de éste.
- **Unificado:** Unifica varias técnicas (orientada a objetos, enfocada al usuario...) de modelado en una única

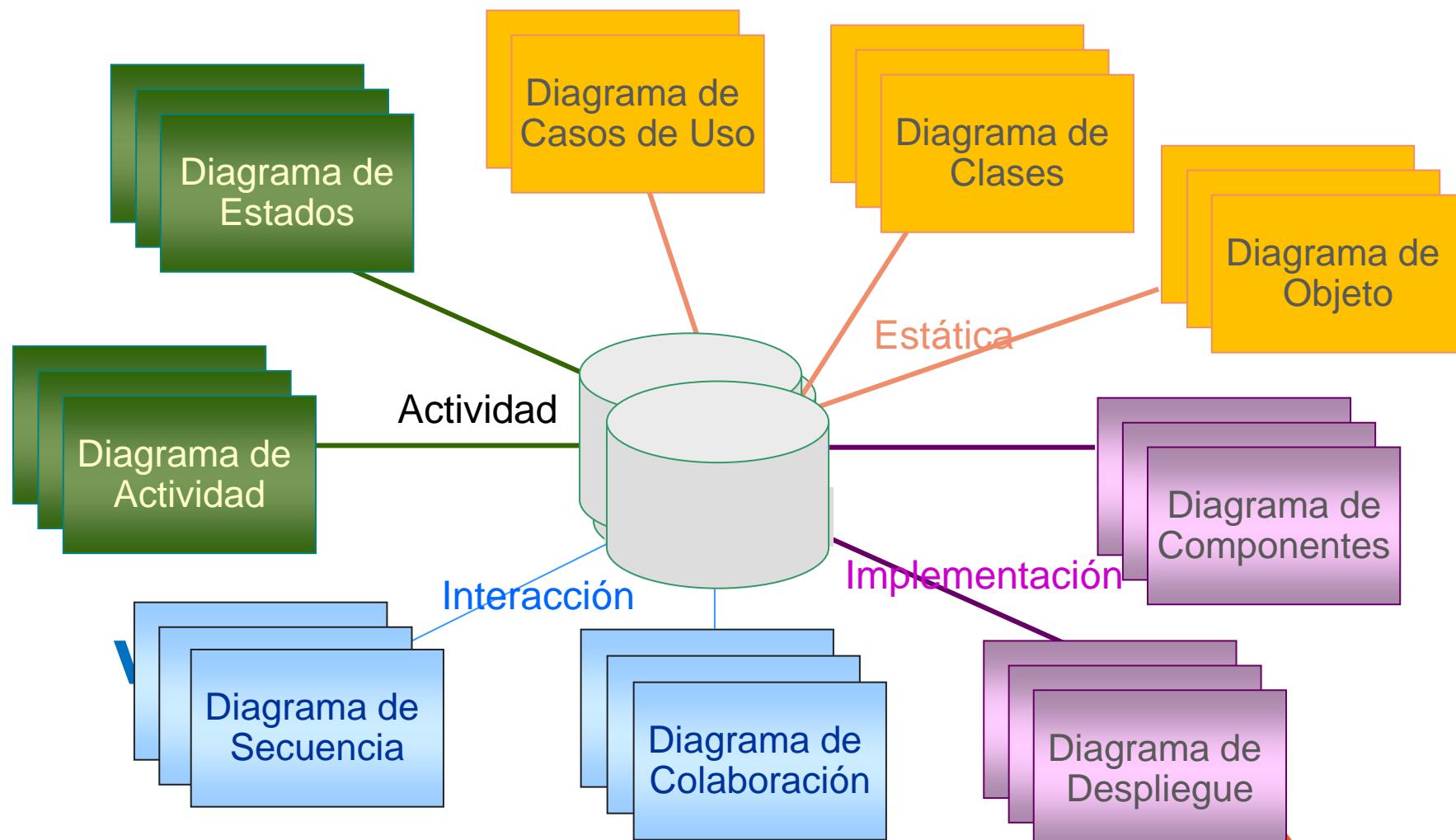


Cual es el uso de UML?

- **Visualizar:** Representar y Comunicar Ideas. Detrás de cada símbolo de UML hay una semántica bien definida.
- **Especificar:** Modelos precisos, no ambiguos, completos.
- **Construir:** Trasladar en forma directa a un lenguaje de programación.
- **Documentar:** Los artefactos construidos durante un proyecto.

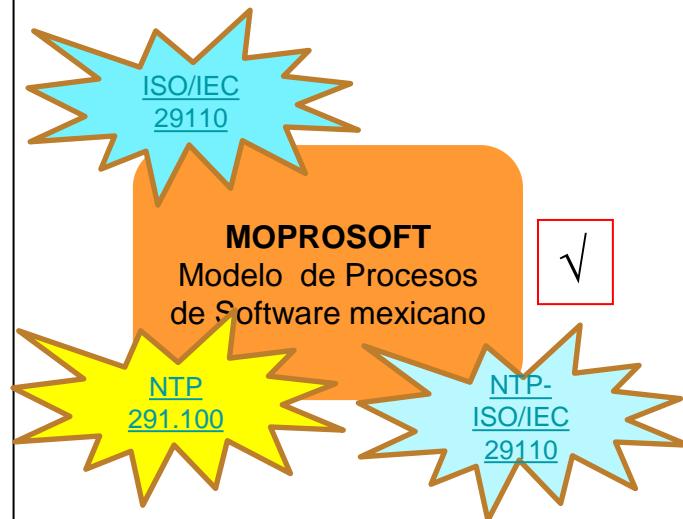
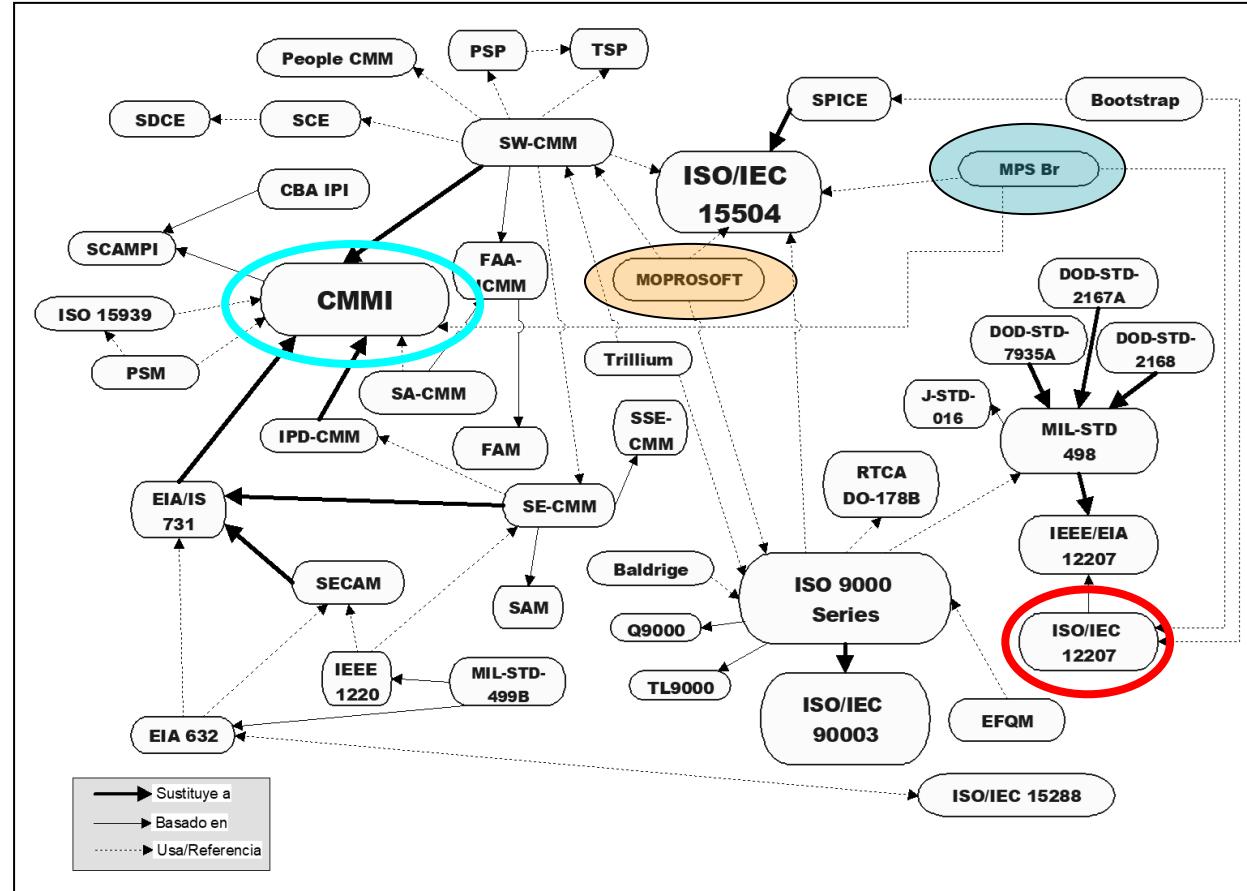


Diagramas de UML



Estándares de ciclo de vida del software

Y las NTP
(Normas
técnicas
peruanas



MPS.BR
Mejora del Proceso de Software Brasileño



UNIVERSIDAD
DE LIMA



Estándares del Ciclo de Vida del Software

REQUERIMIENTOS

- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications
- IEEE Std 1063-2001 IEEE Standard for Software User Documentation

ANÁLISIS Y DISEÑO

- IEEE Std 1471-2000 IEEE Recommended Practice for Architectural Description of Software Intensive
- IEEE Std 1471-2000 IEEE Recommended Practice for Architectural Description of Software-Intensive Systems

CONSTRUCCIÓN DEL SOFTWARE

- ANSI/IEEE 1008-1987 IEEE Standard for Software Unit Testing

PRUEBAS DEL SOFTWARE

- IEEE Std 1012-1998 IEEE Standard for Software Verification and Validation
- IEEE Std 730™-2002 IEEE Standard for Software Quality Assurance Plans
- IEEE Std 829-1998 IEEE Standard for Software Test Documentation

INTEGRACIÓN DEL SOFTWARE

- ISO 12207

MANTENIMIENTO DEL SOFTWARE

- IEEE Std 219-1998 IEE Standard for Software Maintenance

Estándares de los Procesos de Apoyo del Software

PROCESOS DE APOYO DEL CICLO DE VIDA: GESTIÓN DE LA CONFIGURACIÓN

- IEEE Std 828-1998 IEEE Standard for Software Configuration Management Plans
- ANSI/IEEE 1042-1987 IEEE Guide to Software Configuration Management
- ISO 12207



UNIVERSIDAD
DE LIMA



Estándares del Ciclo de Vida del Software(Perú)

REQUERIMIENTOS	IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications IEEE Std 1063-2001 IEEE Standard for Software User Documentation
ANÁLISIS Y DISEÑO	IEEE Std 1471-2000 IEEE Recommended Practice for Architectural Description of Software-Intensive Systems
CONSTRUCCIÓN DEL SOFTWARE	ANSI/IEEE 1008-1987 IEEE Standard for Software Unit Testing NTP ISO/IEC 12207
PRUEBAS DEL SOFTWARE	NTP ISO/IEC 9126-1:2004 NTP ISO/IEC 9126-2:2004 NTP ISO/IEC 9126-3:2005
MANTENIMIENTO DEL SOFTWARE	IEEE Std 219-1998 IEEE Standard for Software Maintenance





Requisitos de Software

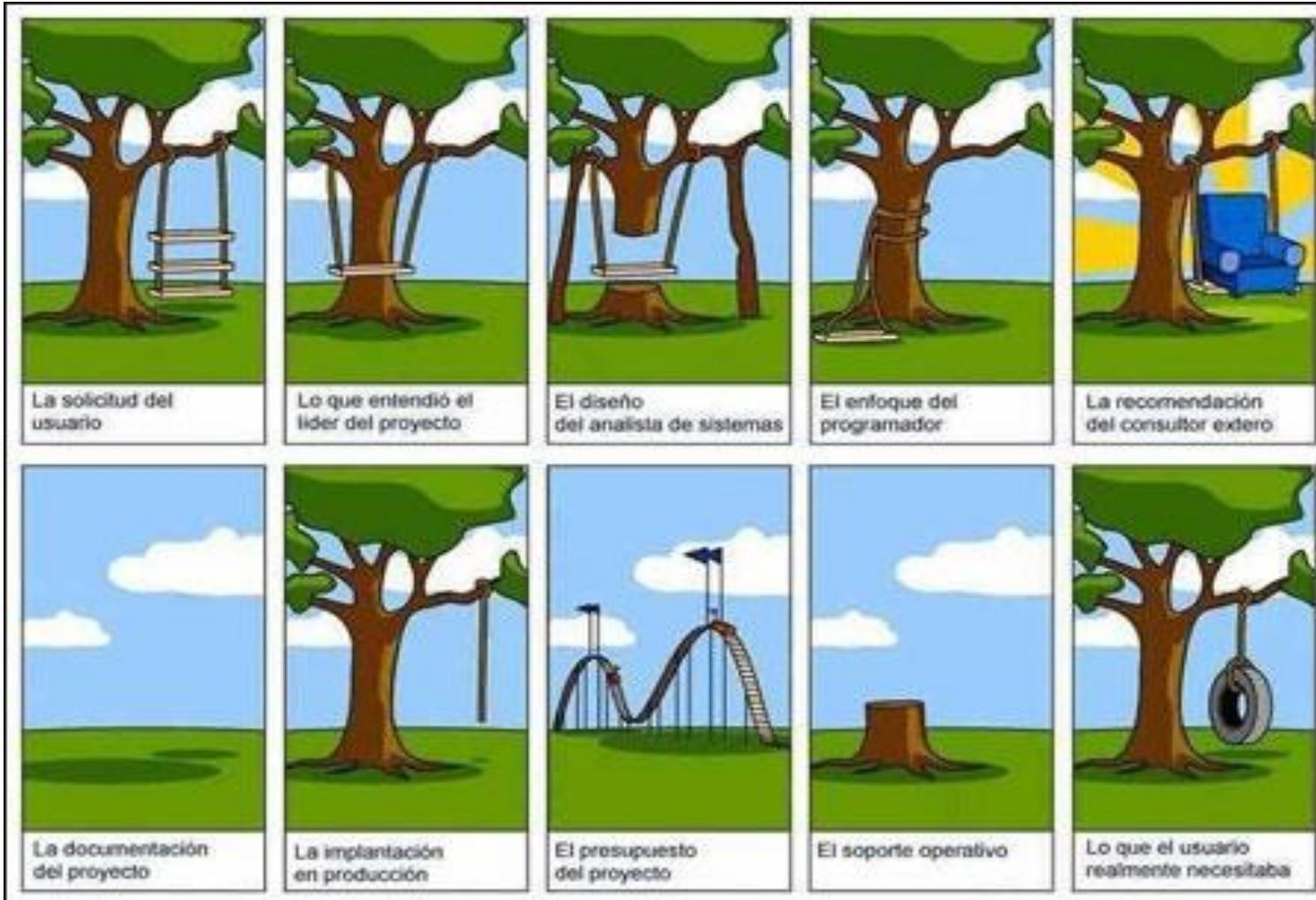
Ingeniera de Requisitos, que es?

Es el **proceso de especificar** los requerimientos a través del **estudio de las necesidades** de los involucrados (stakeholders), analizar sistemáticamente y refinar dichas especificaciones.

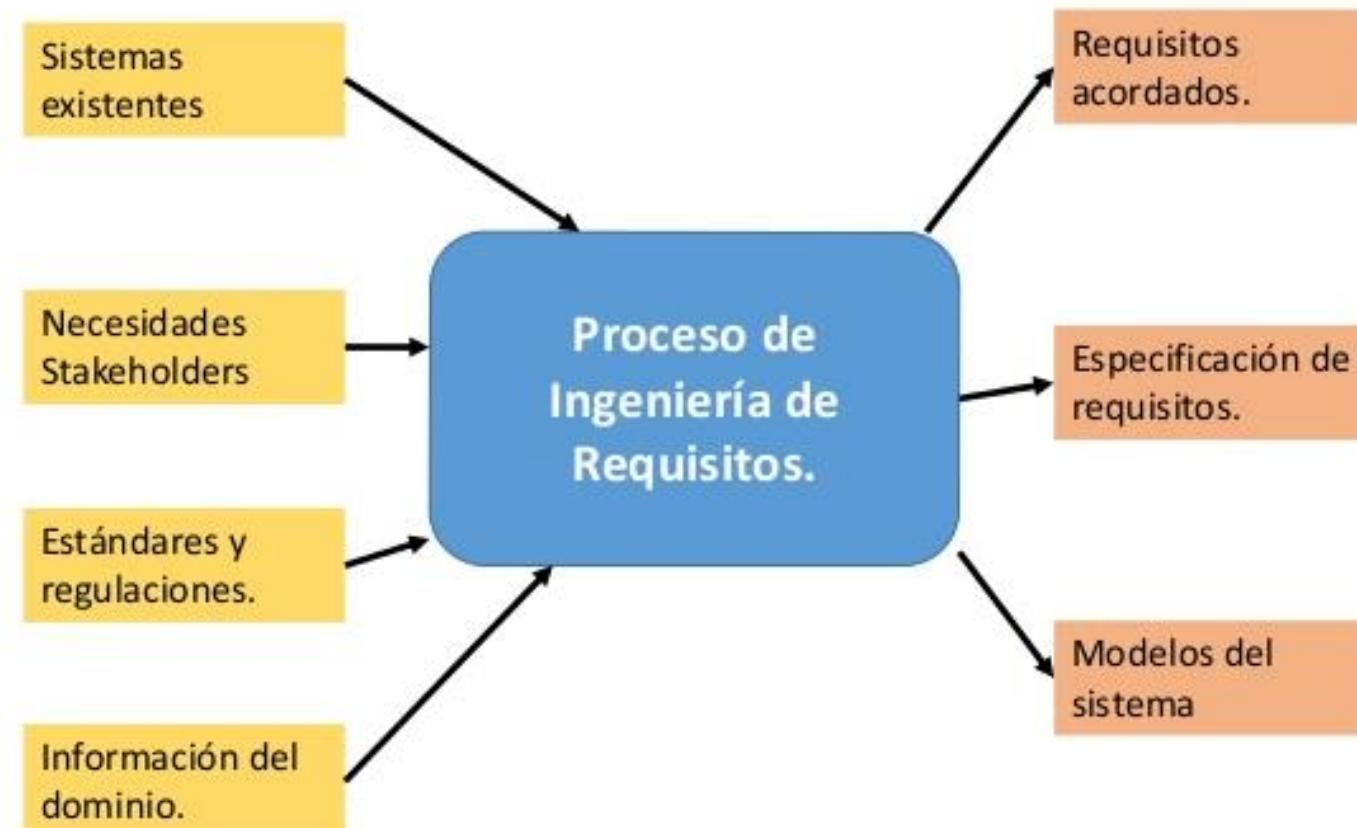
Objetivos:

1. Asegurar que el producto final **cumpla con los requerimientos** propuestos por el usuario.
2. Establecer y mantener **acuerdos** entre los usuarios y el equipo de desarrollo acerca del tratamiento **a los cambios a los requerimientos**.

Requisitos, que son:



El proceso de Ingeniería de Requisitos



Ingeniería de Requerimientos

