



UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA



Departamento de Informática
Universidad Técnica Federico Santa María

Procesos de Desarrollo y Ciclo de Vida del Software

Ingeniería de Software

Hernán Astudillo & Gastón Márquez
Departamento de Informática
Universidad Técnica Federico Santa María

Algo más de Ingeniería de Software

Ingeniería de Software

- Establecimiento y uso de principios con caracteres de ingeniería apropiados para obtener, eficientemente, software confiable, que opere eficaz y eficientemente en máquinas reales
- Concepto se acuñó en 1968, en Conferencia de la OTAN en Alemania, con la intención de que mediante el uso de filosofías y paradigmas de disciplinas ingenieriles establecidas se resolviera la denominada crisis del software

Ingeniería de Software

- Disciplina que se ocupa del desarrollo sistemático, eficaz y eficiente de software eficaz y eficiente
- Objetivos
 - maximizar calidad
 - maximizar productividad
 - minimizar riesgos

Ingeniería de Software

- Implicancias
 - constructores básicos más poderosos
 - mejores técnicas de control de calidad
 - mejores herramientas y métodos
- ...todo en la forma de procesos de software adecuados al tipo de problema que se resuelve y que permitan gestionar de mejor manera los principales riesgos relevantes para todos los stakeholders (involucrados o afectados)

Ciclo de vida del Software

Ciclo de vida

- El ciclo de vida es el conjunto de fases por las que pasa el sistema que se está desarrollando desde que nace la idea inicial hasta que el software es retirado o remplazado (muere).
- También a veces se denomina paradigma
- Algunas de las funciones más destacadas de un ciclo de vida
 - Determinar el orden de las fases del proceso
 - Establecer los criterios de transición para pasar de una fase a otra
 - Definir las entradas y salidas de cada fase
 - Describir los estados por los que se pasa en cada fase
 - Otros

Tipos de modelo de ciclo de vida

- Las principales diferencias entre distintos modelos de ciclo de vida están en:
 1. El alcance del ciclo dependiente de hasta dónde llegue el proyecto correspondiente.
 2. Las características (contenidos) de las fases en que dividen el ciclo. Esto puede depender del propio tema al que se refiere el proyecto, o de la organización.
 3. La estructura y la sucesión de las etapas, si hay realimentación entre ellas, y si tenemos libertad de repetirlas (iterar).

Modelos de ciclo de vida

- La ingeniería del software establece y se vale de una serie de modelos que establecen y muestran las distintas etapas y estados por los que pasa un producto software, desde su concepción inicial, pasando por su desarrollo, puesta en marcha y posterior mantenimiento, hasta la retirada del producto.
- A estos modelos se les denomina “Modelos de ciclo de vida del software”.
- El primer modelo concebido fue el de Royce, más comúnmente conocido como Cascada o “Lineal Secuencial”.
- Este modelo establece que las diversas actividades que se van realizando al desarrollar un producto software, se suceden de forma lineal.

Ciclo de Vida tradicional del Software

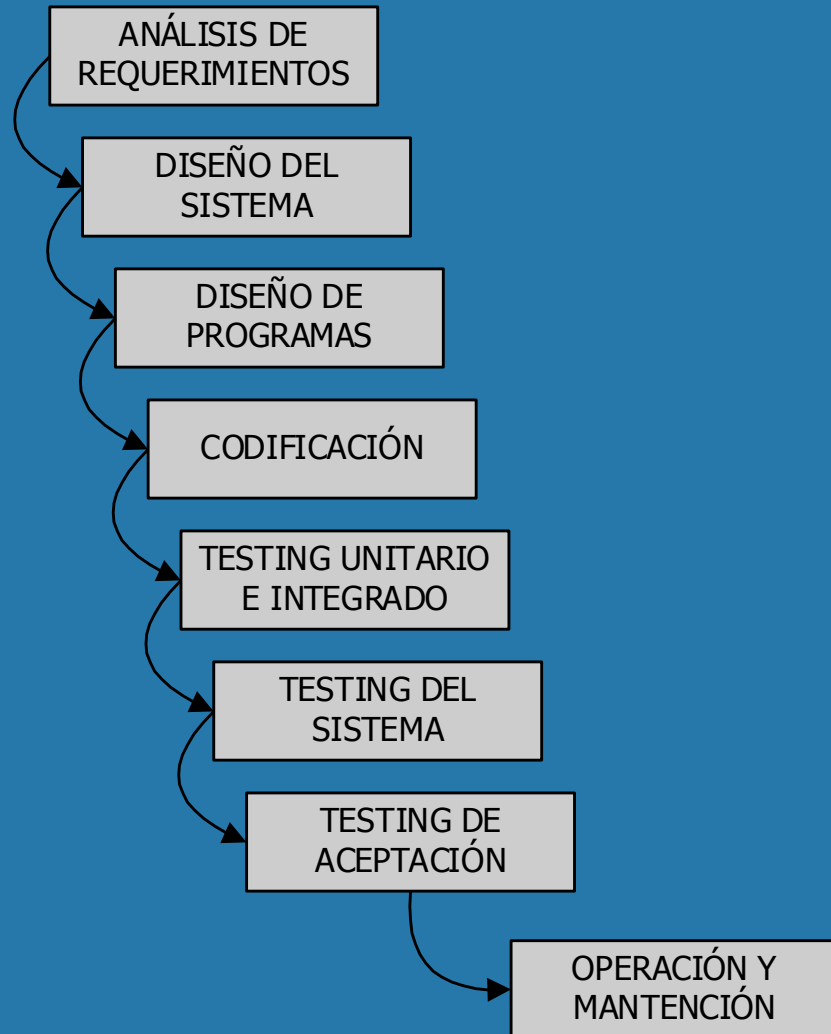


Estándares del Ciclo de Vida del Software [Yarif, 2010]

REQUERIMIENTOS	<ul style="list-style-type: none">• IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications• IEEE Std 1063-2001 IEEE Standard for Software User Documentation
ANÁLISIS Y DISEÑO	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• ANSI/IEEE 1008-1987 IEEE Standard for Software Unit Testing
PRUEBAS DEL SOFTWARE	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• ISO 12207
MANTENIMIENTO DEL SOFTWARE	<ul style="list-style-type: none">• IEEE Std 219-1998 IEE Standard for Software Maintenance

Modelos de desarrollo de software

Modelo en Cascada



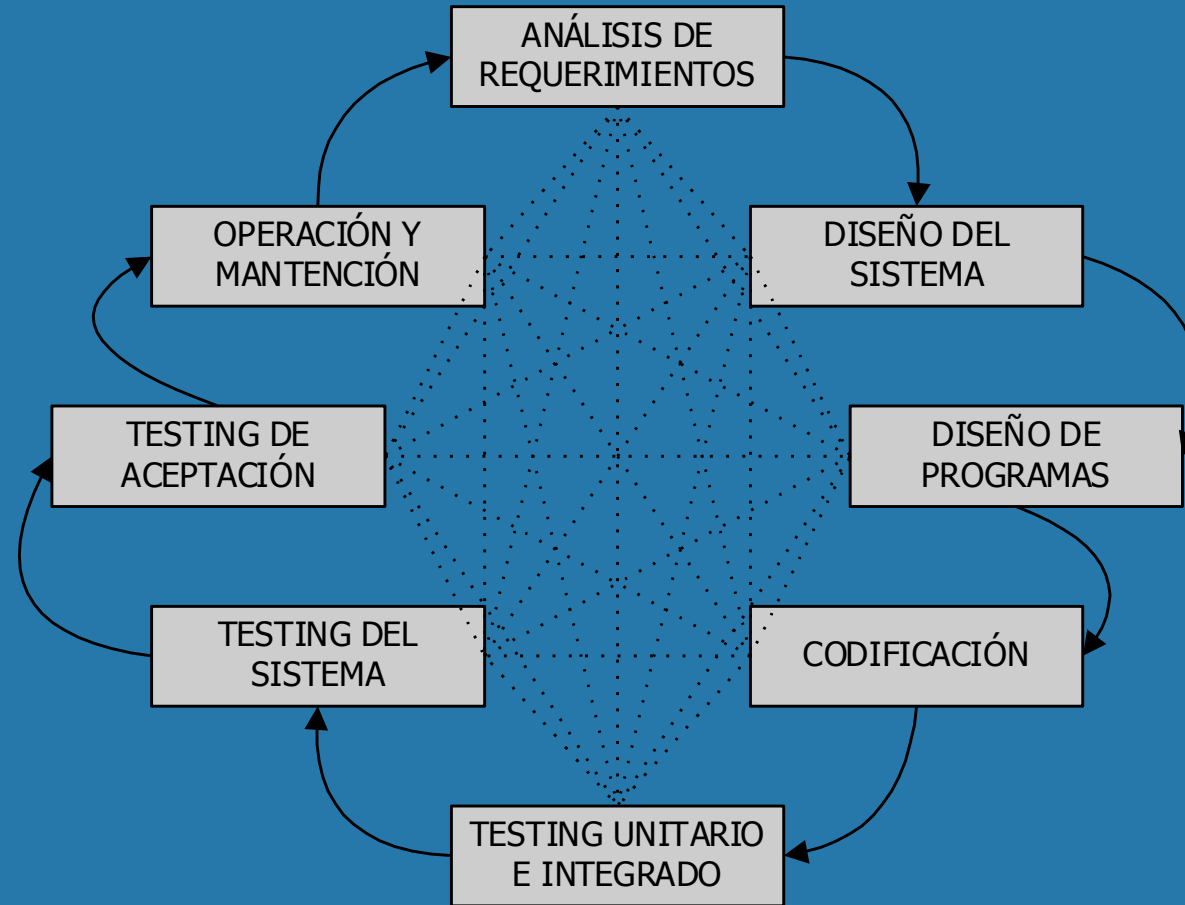
- Es el más antiguo (Royce, 1971)
- Secuencial: debe completarse un estado antes de comenzar el siguiente.
- Ventaja
 - todos ven claramente dónde se está y qué hacer
- Problema
 - no refleja la realidad

Modelo en Cascada

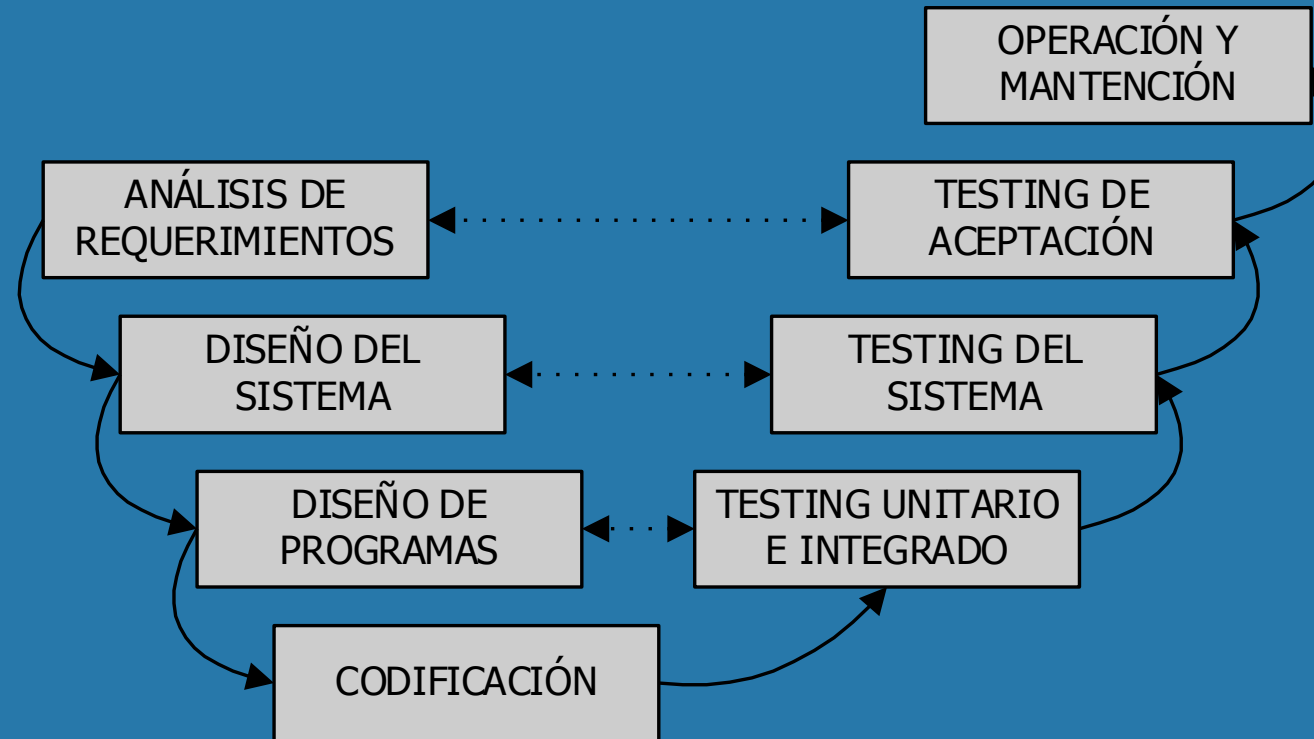
- ¿Cuándo es recomendable usar el modelo?
 - Cuando los requerimientos son bien conocidos
 - Los productos son estables
 - La tecnología es entendida
 - No existen requerimientos ambiguos
 - Ya existe experiencia de desarrollo en cascada en el equipo
 - El proyecto es corto

Modelo en Cascada

- En la práctica



Modelo V



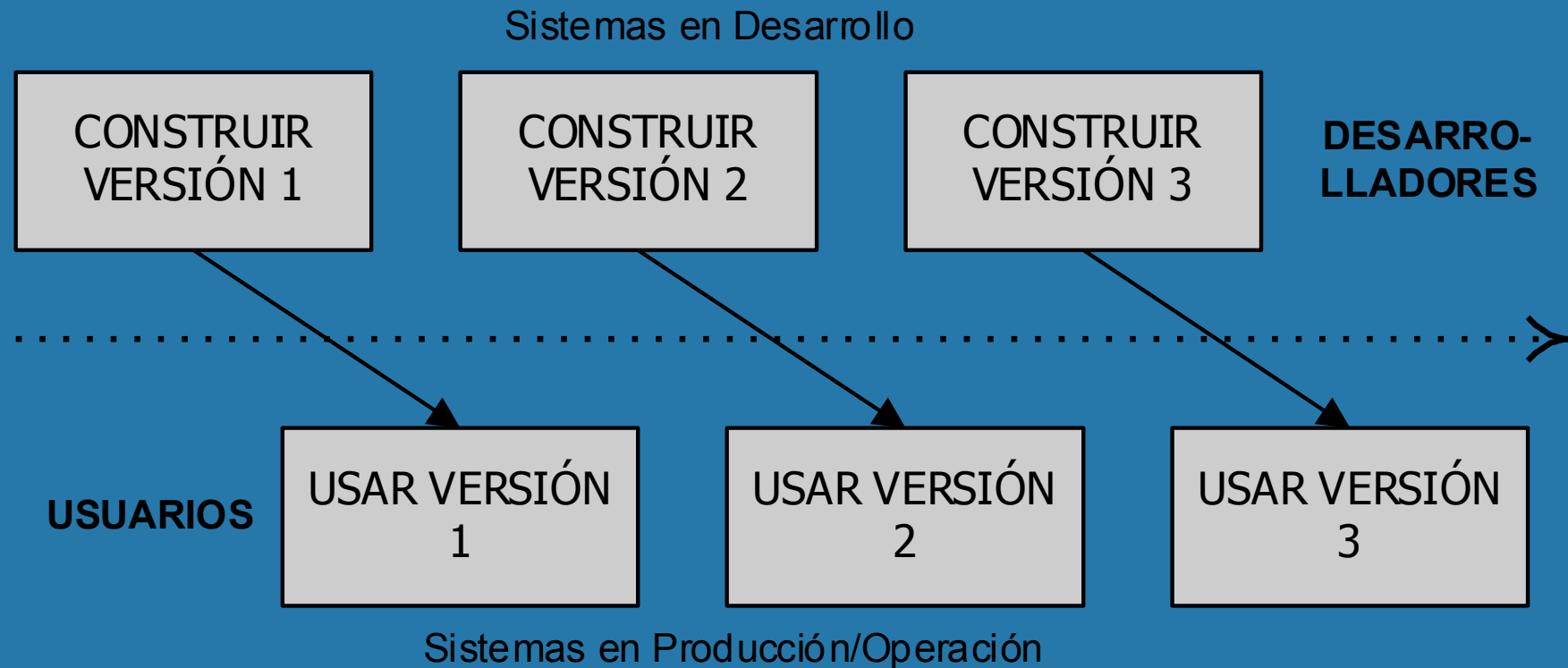
Modelo V

- Variación del modelo en cascada.
 - Cada actividad de prueba es contraparte de una actividad de desarrollo con el mismo ámbito.
- Se forman “ciclos” desarrollo-verificación-desarrollo...
 - Propuesto en Alemania en 1992

Modelo V

- ¿Cuándo es recomendable usar el modelo?
 - Cuando los son de poco a mediano alcance, donde los requerimientos están claramente definidos
 - Cuando los recursos técnicos están completamente disponibles y exista experiencia en el uso de ellos

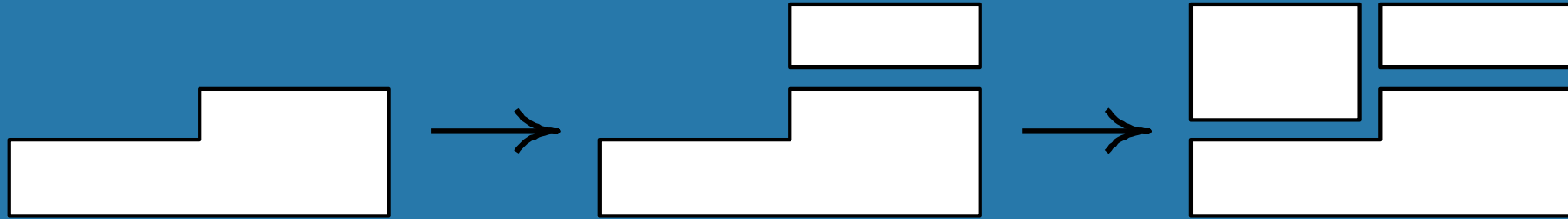
Modelo de Desarrollo en Fases



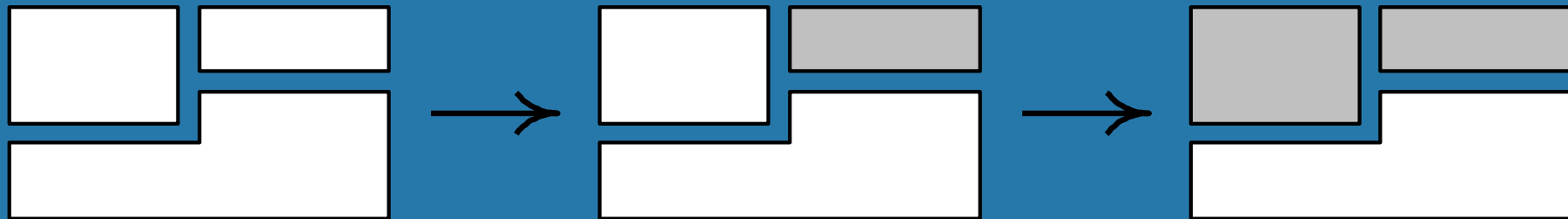
Modelo de Desarrollo en Fases

- El mercado exige entregas prontas
- Idea: desarrollar en fases
 - Diseñar el sistema para entregarlo por partes.
 - El usuario obtiene tiene funcionalidad parcial... mientras se desarrolla el resto.
- Hay dos sistemas funcionando en paralelo:
 - Desarrollo: siguiente versión (release) que reemplazará a la actual de Producción.
 - Producción: usado por el Cliente.
- (Veremos en SCM que hay 3 ambientes)
 - Desarrollo -> Prueba -> Producción

Desarrollo Iterativo-Incremental

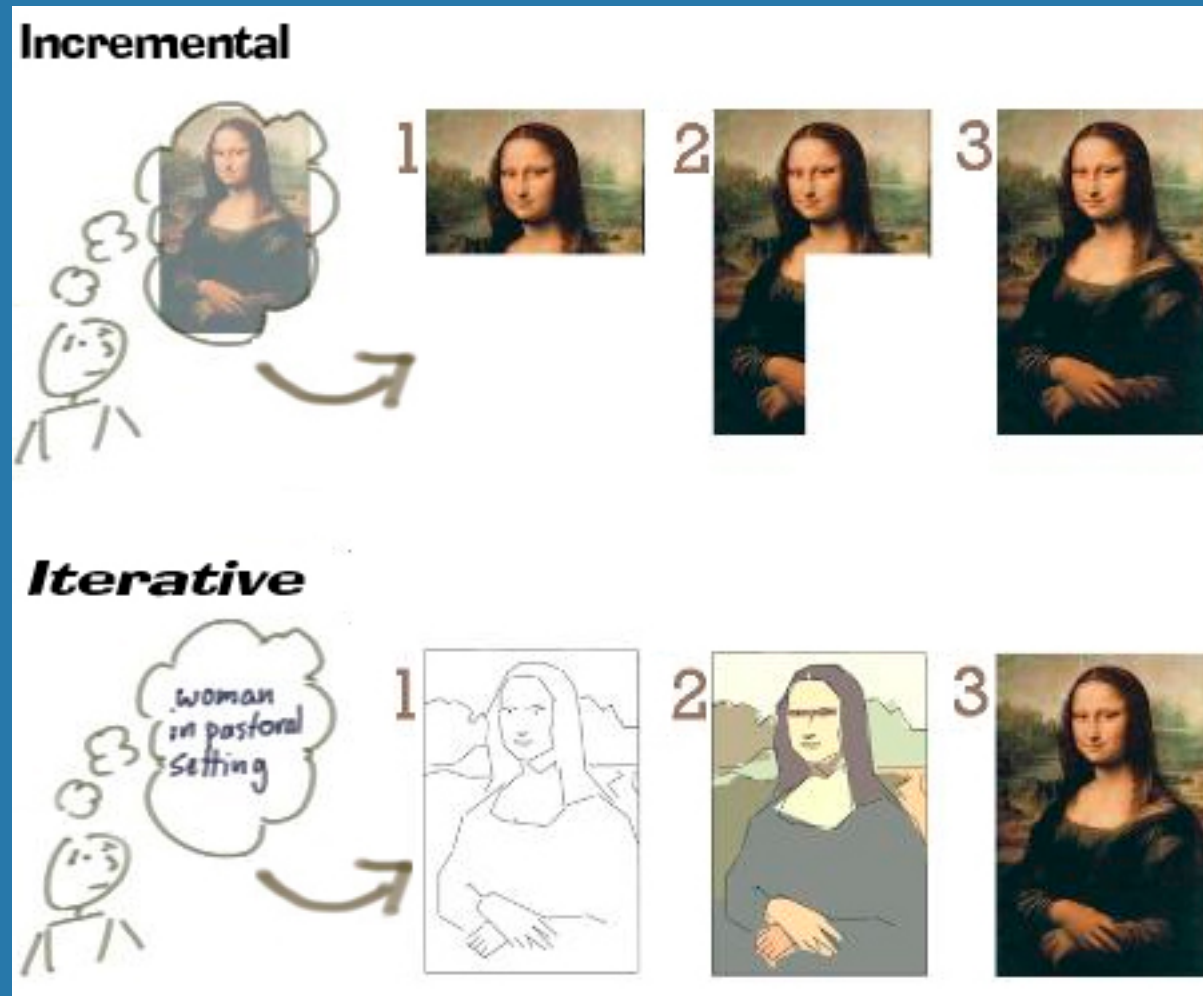


- Desarrollo Incremental



- Desarrollo Iterativo

Desarrollo Iterativo-Incremental



Desarrollo Iterativo-Incremental

- Hay dos maneras básicas de organizar el desarrollo de las versiones:
 - **Incremental.** El sistema se especifica y luego es particionado en subsistemas, por funcionalidad. Se comienza con un subsistema pequeño y se va agregando funcionalidad en cada nueva versión
 - **Iterativo.** Se entrega el sistema completo el comienzo y se van haciendo cambios y mejoras en la funcionalidad, en cada nueva versión

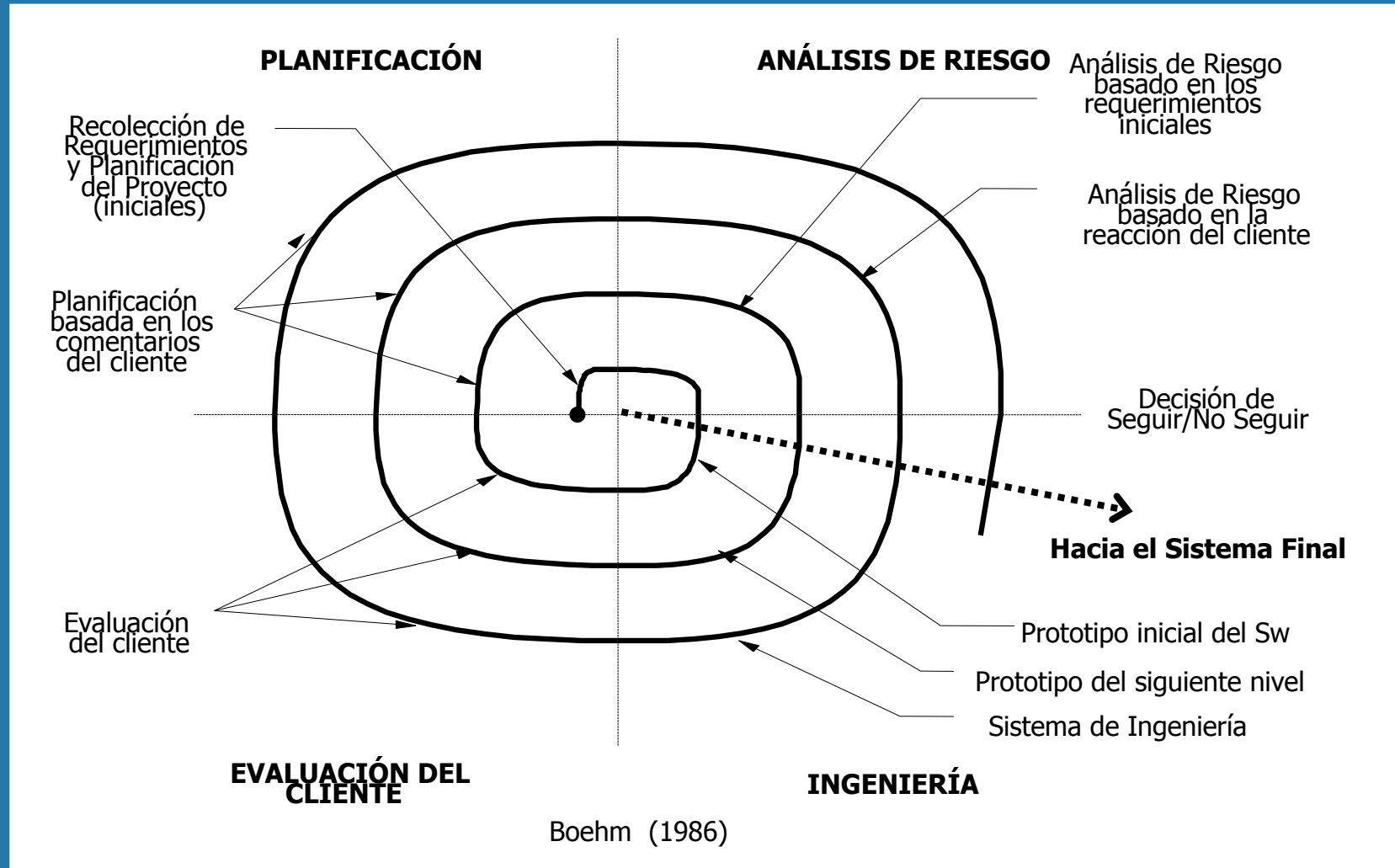
Desarrollo Iterativo-Incremental

- Ventajas
 - El entrenamiento de los usuarios puede comenzar tempranamente, aunque falte funcionalidad
 - En ambiente de prueba o de producción
 - Versiones frecuentes permiten encontrar y resolver problemas, a partir de las versiones operativas
 - Perversión: “usuario beta”
 - El equipo de desarrollo puede enfocarse en diferentes áreas de experticia en cada versión
 - Esfuerzo focalizado

Desarrollo Iterativo-Incremental

- ¿Cuándo es recomendable usar el modelo?
 - Los requerimientos del sistema completo están bien definidos
 - Los requerimientos estratégicos están abordados
 - Existe una gran necesidad de que el producto esté en el mercado lo antes posible
 - Una nueva tecnología está empezando a ser usada
 - Existe alto riesgo en los objetivos del proyecto

Modelo Espiral



Modelo Espiral

- El orden de ejecución de las actividades de desarrollo es determinado por análisis de Riesgo
- Modelo iterativo
 - Planificación → Análisis de Riesgo → Ingeniería → Evaluación → Planificación → ...
 - En cada iteración se evalúa las alternativas y se elige una para eliminar o minimizar el mayor riesgo restante
- Propuesto por Boehm en 1986

Modelo Espiral

- Ventajas
 - Alto manejo del riesgo
 - Buen manejo de proyectos críticos
 - Buena documentación
 - Se pueden agregar funcionalidades extras al proyecto
 - El software es realizado en etapas tempranas del proyecto

Modelo Espiral

- Desventajas
 - El modelo puede ser costoso
 - Se necesita experiencia en el análisis de riesgo
 - El éxito del proyecto depende fuertemente del análisis del riesgo
 - No funciona para pequeños proyectos

Modelo Espiral

- ¿Cuándo es recomendable usar el modelo?
 - Los costos y el riesgo es importante
 - Pequeños a grandes proyectos
 - Cuando los usuarios no saben lo que quieren
 - Los requerimientos son complejos
 - Nuevas líneas de productos
 - Cambios significativos son esperados

Modelos utilizados por la industria

- De acuerdo con un estudio realizado por [Appelo, NOOP.NL, 2008], los modelos que son utilizados por la industria son:
 - Scrum
 - Extreme Programming
 - Lean Software Development
 - Unified Process
 - Rational Unified Process
 - Dynamic Systems Development Method
 - Prince2
 - Project Management Body of Knowledge
 - Capability Maturity Model Integration
 - Feature Driven Development
 - Microsoft Solutions Framework
 - Open Unified Process
 - Essential Unified Process
 - Agile Unified Process
 - Enterprise Unified Process
 - Crystal
 - Evo

Tabla comparativa [Sorensen, 2008]

	Waterfall	Incremental	Boehm Spiral
STRENGTHS			
Allows for work force specialization	X	X	X
Orderliness appeals to management	X	X	X
Can be reported about	X	X	X
Facilitates allocation of resources	X	X	X
Early functionality		X	X
Does not require a complete set of requirements at the onset		X(4)	X
Resources can be held constant		X	
Control costs and risk through prototyping			X
WEAKNESSES			
Requires a complete set of requirements at the onset	X		
Enforcement of non-implementation attitude hampers analyst/designer communications	X		
Beginning with less defined general objectives may be uncomfortable for management		X	X
Requires clean interfaces between modules		X	
Incompatibility with a formal review and audit procedure		X	X
Tendency for difficult problems to be pushed to the future so that the initial promise of the first increment is not met by subsequent products		X	X
(4) The incremental model may be used with a complete set of requirements or with less defined general objectives.			
Table 2: Strengths and Weaknesses of Models.			

¿Veamos un ejemplo?

- Modelo seleccionado: Cascada
- Proyecto: Libro de direcciones virtual
- **Requerimientos** (Timeframe: 2 semanas):
 - El usuario debe crear nuevos contactos
 - El usuario puede ver sus contactos
 - El usuario puede importar contactos de otros programas
 - El usuario puede enviar correos a sus contactos
 - El usuario puede agregar fotos a sus usuarios

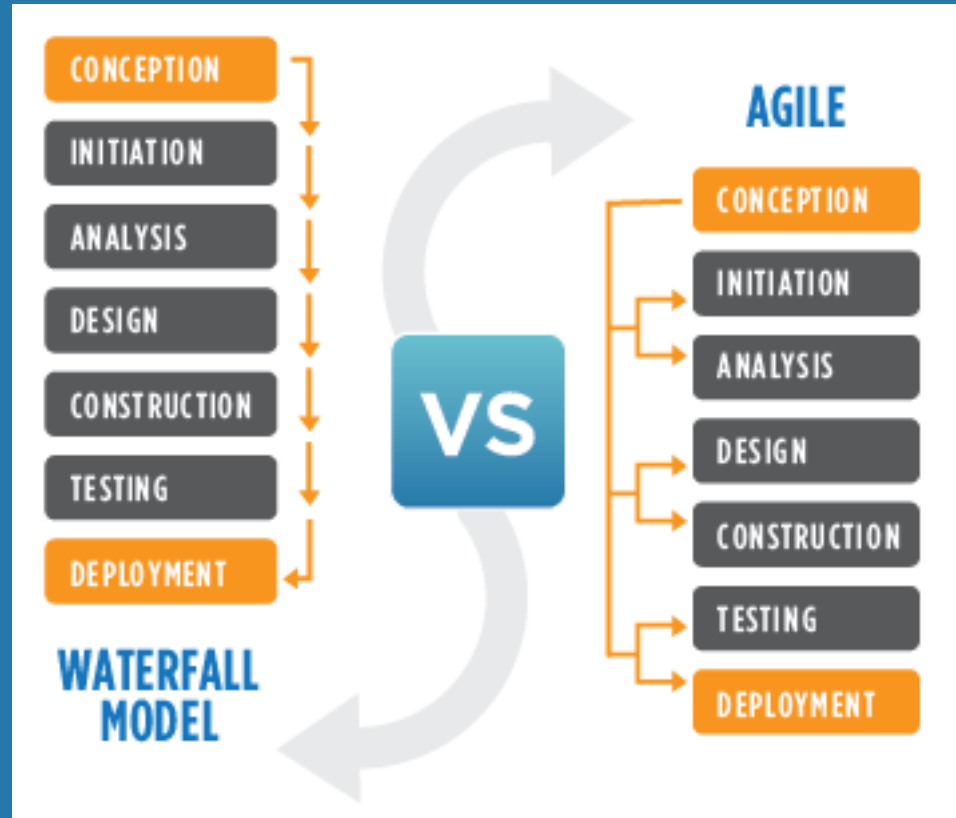
¿Veamos un ejemplo?

- **Análisis** (Timeframe: 1 semana)
 - Los ingenieros toman los requerimientos y los analizan, realizando las preguntas necesarias. El Product Manager actualiza los documentos necesarios.
- **Diseño** (Timeframe: 3 semanas)
 - Los ingenieros crean el diseño funcional, incluyendo la base de datos y workflows
- **Implementación** (Timeframe: 1 semana)
 - Los ingenieros desarrollan las funcionalidades y se preparan para el testing

¿Veamos un ejemplo?

- **Testing** (Timeframe: 2 semanas)
 - Se prueban las funcionalidades
- **Entrega** (Timeframe: 0 semanas)
 - El producto final es entregado
- Tiempo total: 9 semanas
- Notar que cualquier cambio que ocurra en modelo, el proyecto se retrasa.
- Debate (ya veremos este tema en profundidad 😊) → ¿Qué ocurre si usamos un *modelo ágil*?

¿Veamos un ejemplo?



Procesos de Desarrollo de Software

Procesos de Software

- Todas las organizaciones involucradas en el desarrollo de software necesitan establecer, gestionar y soportar el trabajo de desarrollo.
- El término “proceso de desarrollo de software” tiende a unificar todas las actividades y prácticas que cubren esas necesidades [Järvi et al., 2005].
- El principal objetivo de tal proceso es garantizar la construcción de un software adecuado, conforme a sus especificaciones y dentro de los límites de tiempo y costo [Combemale et al., 2006].
- Modelar el proceso de software es una forma para mejorar el desarrollo y la calidad de las aplicaciones resultantes.

SPEM [1]

- La Ingeniería de Procesos de Software (SPE, *Software Process Engineering*) consiste en modelar, diseñar, mejorar y aplicar procesos utilizando lenguajes de modelado de procesos (PML, *Process Modelling Language*).
- Uno de tales lenguajes es SPEM (*Software Process Engineering Metamodel*) [SPEM, 2008], una especificación de OMG (*Object Management Group*) que está basado en MOF (*MetaObject Facility*) y es un metamodelo UML (*Uniform Model Language*).

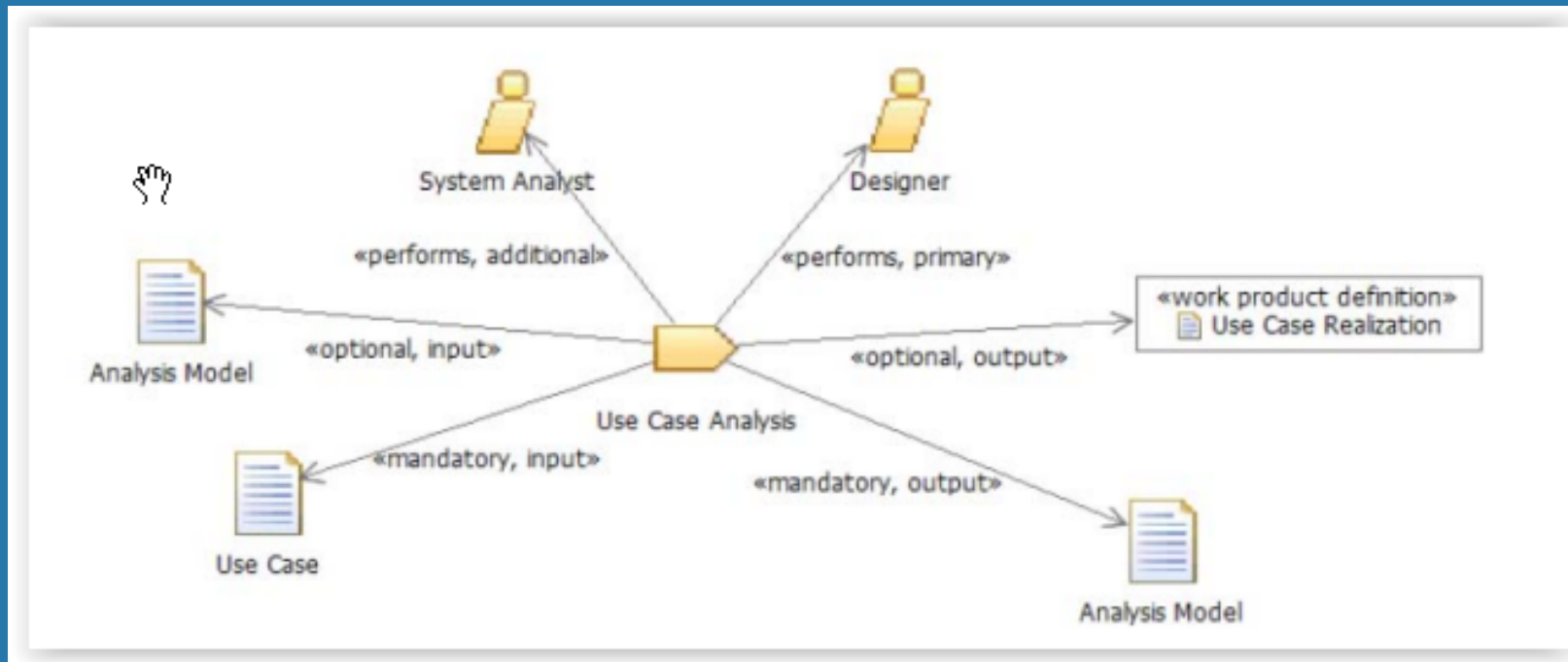
SPEM [2]

- SPEM permite representar una familia de procesos de desarrollo de software y sus componentes.
- Para ello provee un conjunto de elementos de modelado de procesos para describir cualquier proceso de desarrollo de software sin agregar modelos o restricciones de alguna área o disciplina específica, tal como gestión de proyectos o análisis.







SPEM [3]

- SPEM proporciona una sintaxis y estructura para cada aspecto de los procesos desarrollo, incluyendo:
 - Roles.
 - Tareas.
 - Artefactos.
 - Lista de verificación
 - Productos de trabajo.
 - Técnicas y herramientas.
 - Estructuras de trabajo.
 - Capacidad de rastreo y refinamiento.
 - Ayuda sensible al contexto, guía y lineamientos.
 - Descripción textual de elementos.

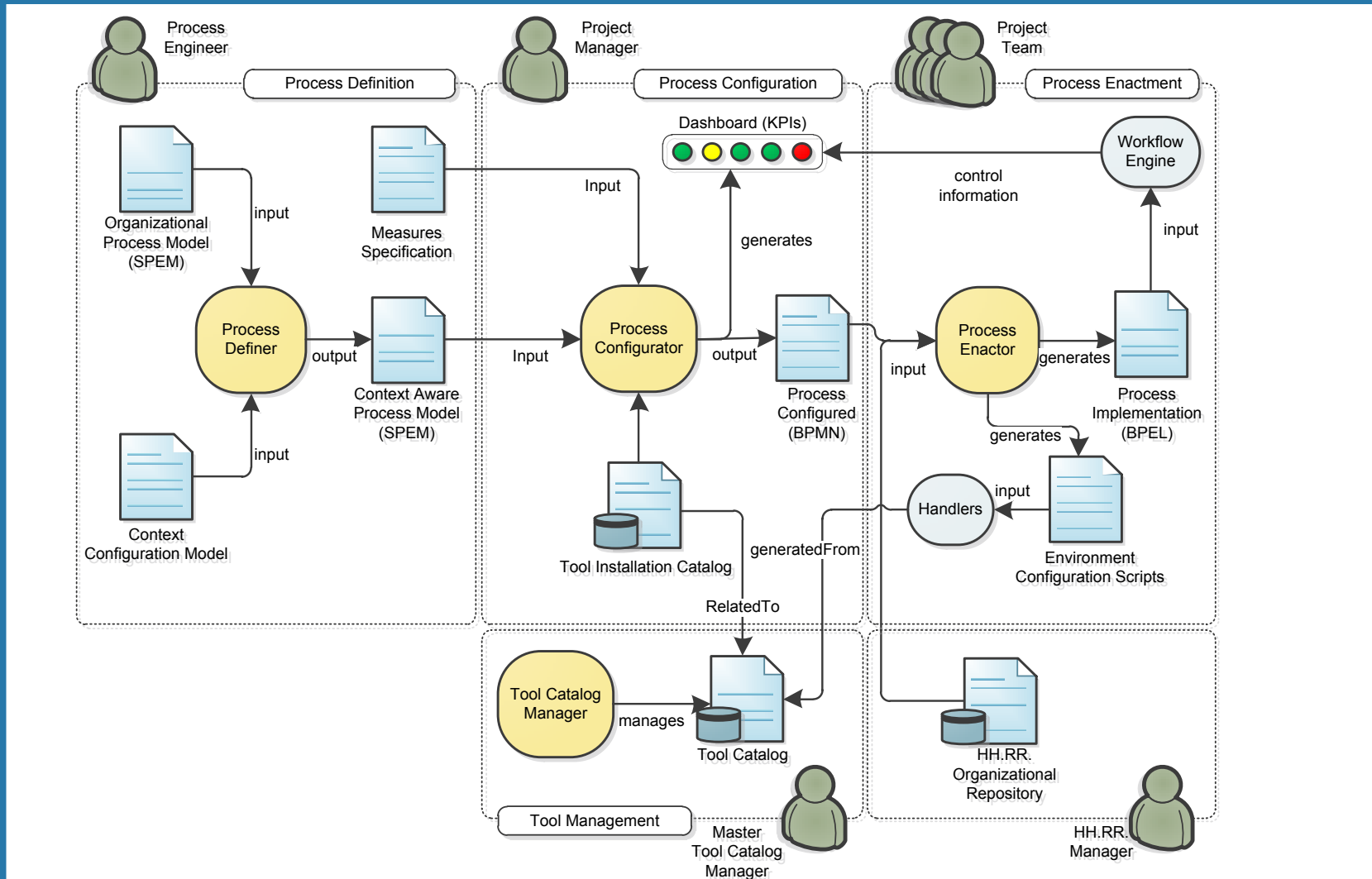
SPEM – Notación para procesos



SPEM – elementos

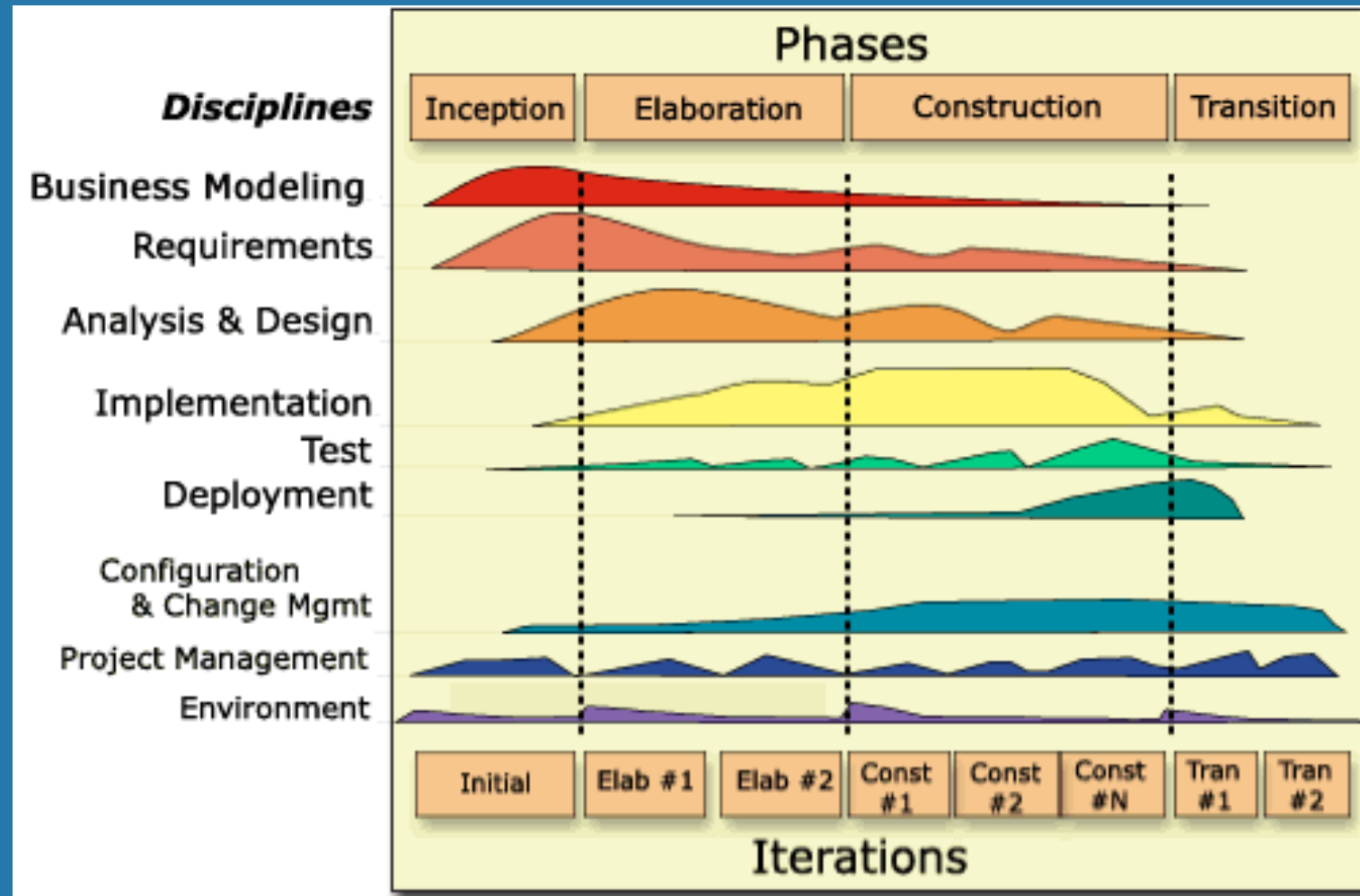
- Roles 
- Workproducts 
 - Artefactos
 - Resultados (“outcomes”): intangibles
 - Entregables = artefactos + resultados
- Tareas 
- Guías  checklists  plantillas 

SPeM – ejemplo



RUP – Rational Unified Process

RUP – Vista global



RUP – Rational Unified Process

- RUP fue desarrollado por Rational en 1999
- Framework de procesos para describir procesos de desarrollo específicos
 - Modelo de proceso centrado en arquitectura
 - Enfoque iterativo-incremental
- Instanciación
 - Manual o con IBM Rational Method Composer
 - Ej: AgileUP (Ambler), BasicUP (IBM/Eclipse), EnterpriseUP...
- RUP tiene 2 dimensiones
 - Horizontal (tiempo): ciclo de vida en fases e iteraciones
 - Vertical (workflows): disciplinas de proceso; agrupan actividades
- Artefactos
 - Entradas y salidas de actividades

RUP – Elementos clave

- Todo proyecto tiene 4 fases
 - Concepción: det. visión, requisitos (alto nivel), ámbito -> decisión de factibilidad
 - Elaboración: det. requisitos (detalle, 80%), arquitectura, prototipos -> arquitectura & plan
 - Construcción: análisis, diseño, desarrollo, testing -> beta desplegable
 - Transición: despliegue, post-mortem -> producto liberado
- Cada fase puede tener iteraciones
- Disciplinas (9)
 - Modelado de negocio, Requisitos, Análisis y Diseño, Implantación, Prueba (test), Despliegue, Gestión de Cambio y Configuración, Gestión del Proyecto, Ambiente
- Roles (36, en 6 grupos)
- Workproducts → “artefactos” (76)

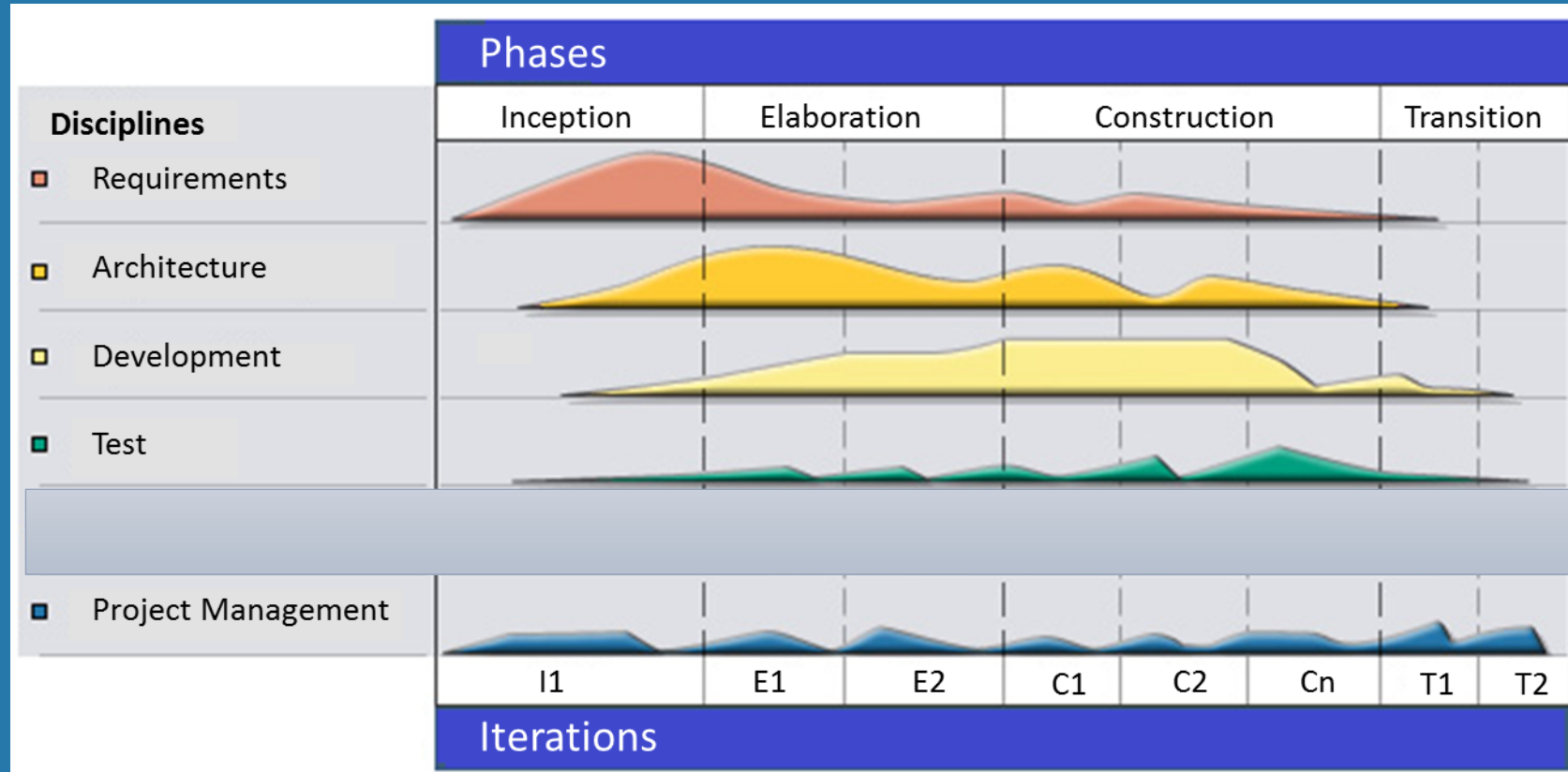
RUP – adopción

Cuadro 22. Empresas con personal certificado

TIPO DE CERTIFICACIÓN	No. DE EMPRESAS	%
ITIL	75	25,17%
PMI	32	10,74%
COBIT	8	2,68%
PSP/TSP	6	2,01%
VAL IT	5	1,68%
RUP	5	1,68%
ESCM	3	1,01%
TOGAF	3	1,01%
SIX SIGMA	1	0,34%
Otro	160	53,69%
TOTAL	298	100,00%

- Fuente: estudio de productos y servicios de Fedesoft (2012), Colombia, Fáber Giraldo

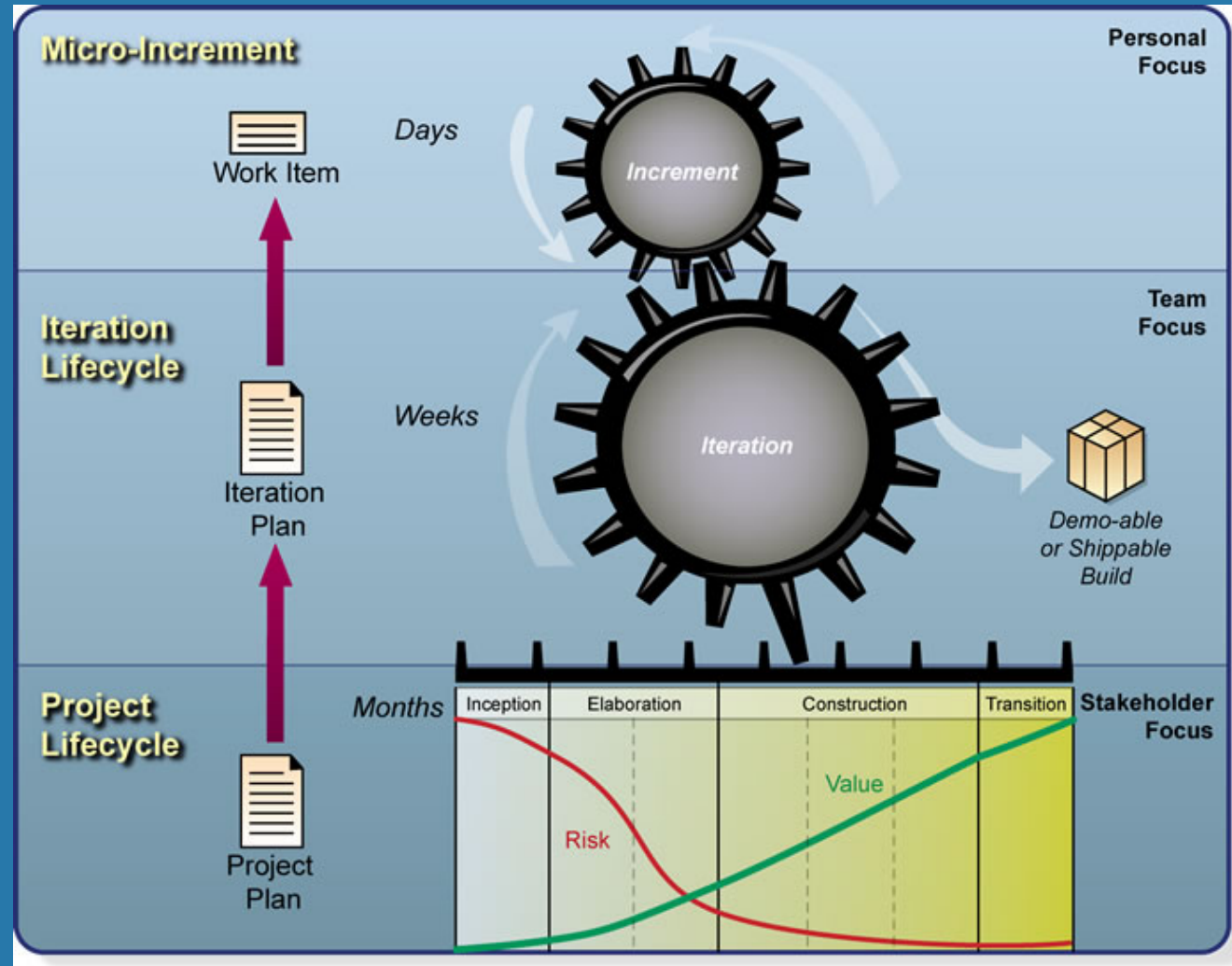
OpenUP



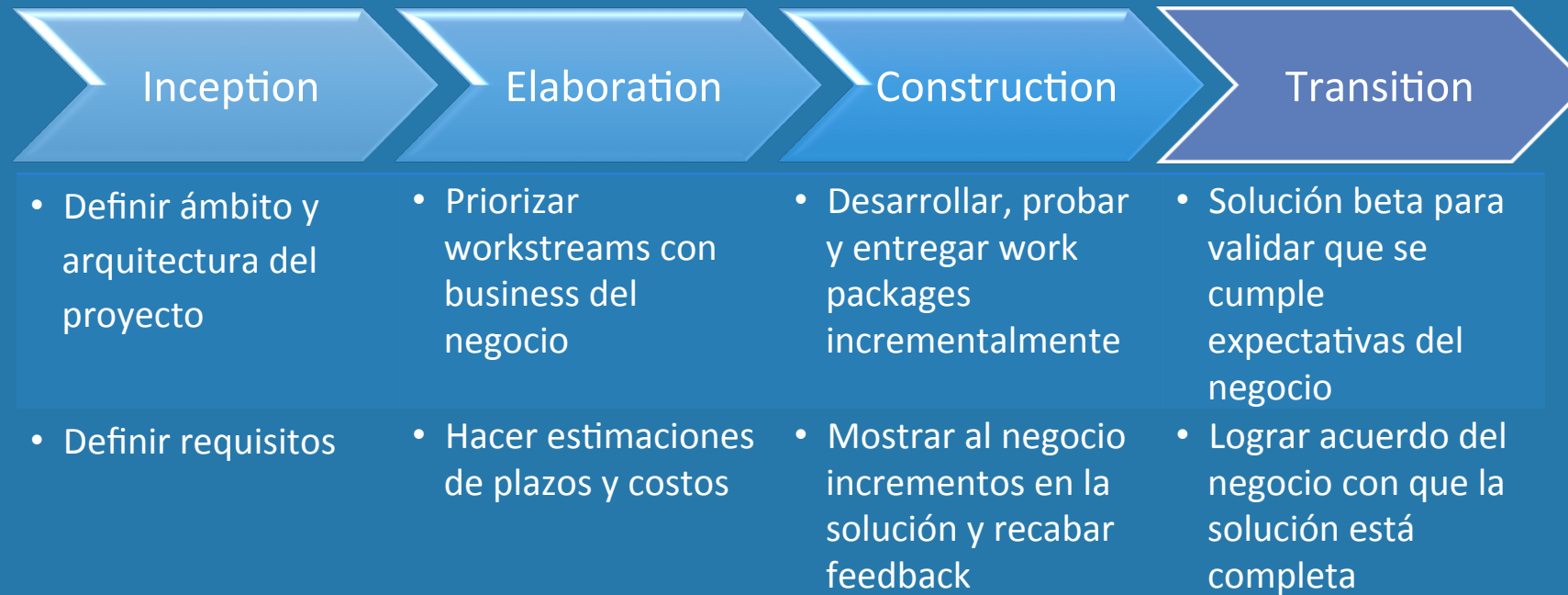
OpenUP

- Desarrollado por IBM y Eclipse Foundation
 - <http://epf.eclipse.org/wikis/openup/>
- Instanciación
 - EPF Composer (EPF = Eclipse Process Framework, open source)
- Versión ligera de RUP
 - Disciplinas: RUP = 9, OpenUP = 5
 - Roles: RUP = 36 (en 6 grupos), OpenUP = 7
 - Workproducts: RUP = 76, OpenUP = 17

OpenUP



OpenUP



EPF Composer

- EPF Composer es una herramienta que permite generar marcos de procesos de software para una organización.
- Se basa en modelos reutilizables que permiten tomar las mejores prácticas del mercado e integrarlas en el framework de procesos organizacionales.
- Ambiente de desarrollo Eclipse
 - Permite autorizar, parametrizar y publicar métodos
 - Añadir, eliminar y cambiar elementos de acuerdo con las necesidades de su proceso
 - Publicar y comunicar el contenido para servir de guía a su equipo de trabajo

EPF Composer (composer)

The image displays two overlapping screenshots of the EPF Composer application. The left screenshot shows the 'define_vision' task configuration, and the right screenshot shows the 'rup_analysis_class' work product configuration.

Left Screenshot: define_vision Task

Task: define_vision

General Information

Name: define_vision
Presentation name: Define Vision
Brief description: Define the vision for the future system. Des

Detail Information

Purpose: The solution is proposed for a pro the development team to express

Main description:

Key considerations:

Alternatives:

Version Information

Version: 1.0.0
Change date: Wednesday, February 28, 2007
Change description:

Authors:

Right Screenshot: rup_analysis_class Work Product

Work Product (Artifact): rup_analysis_class

General Information

Name: rup_analysis_class
Presentation name: Analysis Class
Unique ID:
Brief description: This work product specifies elements of an early conceptual model for 'things in the system which have responsibilities and behavior'.

Detail Information

Purpose: Analysis classes are used to capture the major "clump" system.

Main description: Analysis Classes specify elements of an early conceptual model of the system which have responsibilities and behavior. prototypical classes of the system, and are a 'first-pass' abstraction that the system must handle. Analysis classes are maintained in their own right, if a "high-level", concept is defined. Analysis classes also give rise to the major

Key considerations:

Notation

Brief outline:

Representation options:

Tailoring

Impact of not having:

Work Product (Artifact): rup_analysis_class

Representation options:

CRC Card technique - see [WIR90] for details of this technique). On the front side of the card, capture the name and description of the class. An example for a Course in a course registration system is listed below.

Class Name	Description
Course	The Course is responsible for maintaining information about a set of course sections having a common subject, requirements and syllabus.

Responsibilities: To maintain information about the course.

Name	Description	Type
Course Title	The name of the course	string
Description	A short description of the course	string

Attributes:

On the back of the card, draw a diagram of the class:

```

graph LR
    Course[Course] -- "0..*" --> Section[Section]
    Section -- "1..*" --> Professor[Professor]
    Section -- "1..*" --> Student[Student]
    Section -- "0..*" --> Textbook[Textbook]
    Section -- "1..*" --> Room[Room]
    Course -- "pre-requisite" --> Section
    
```

EPF Composer (lector)

The screenshot displays the Eclipse Process Framework Composer (EPF Composer) application. The title bar indicates the file path: C:\cmsynergy\uschi\ccm_wa\logon\OpenUP~uschi\OpenUP. The menu bar includes File, Edit, Search, SYNERGY/CM, Configuration, Window, and Help. The toolbar contains icons for file operations and a search icon. The left sidebar shows a tree view of the project structure, with 'OpenUP' selected. Under 'OpenUP Disciplines', 'Requirements' is expanded, and 'Define Vision' is selected. The main content area displays the 'Task: Define Vision' task details.

Task: Define Vision

Define the vision for the future system. Describe the problem and features based on Stakeholder requests.

Disciplines: [Requirements](#)

[Expand All Sections](#) [Collapse All Sections](#)

Purpose

The solution is proposed for a problem that everybody agrees on. Stakeholders collaborate with the development team to express and document their problems, needs, and potential features for the system to be, so the project team can better understand what has to be done.

[Back to top](#)

Relationships

Roles	Primary Performer: <ul style="list-style-type: none"> Analyst 	Additional Performers: <ul style="list-style-type: none"> Architect Project Manager Stakeholder
Inputs	Mandatory: <ul style="list-style-type: none"> None 	Optional: <ul style="list-style-type: none"> Vision Work Items List
Outputs	<ul style="list-style-type: none"> Glossary Vision Work Items List 	
Process Usage	<ul style="list-style-type: none"> Initiate Project > Define Vision 	

[Back to top](#)

Steps

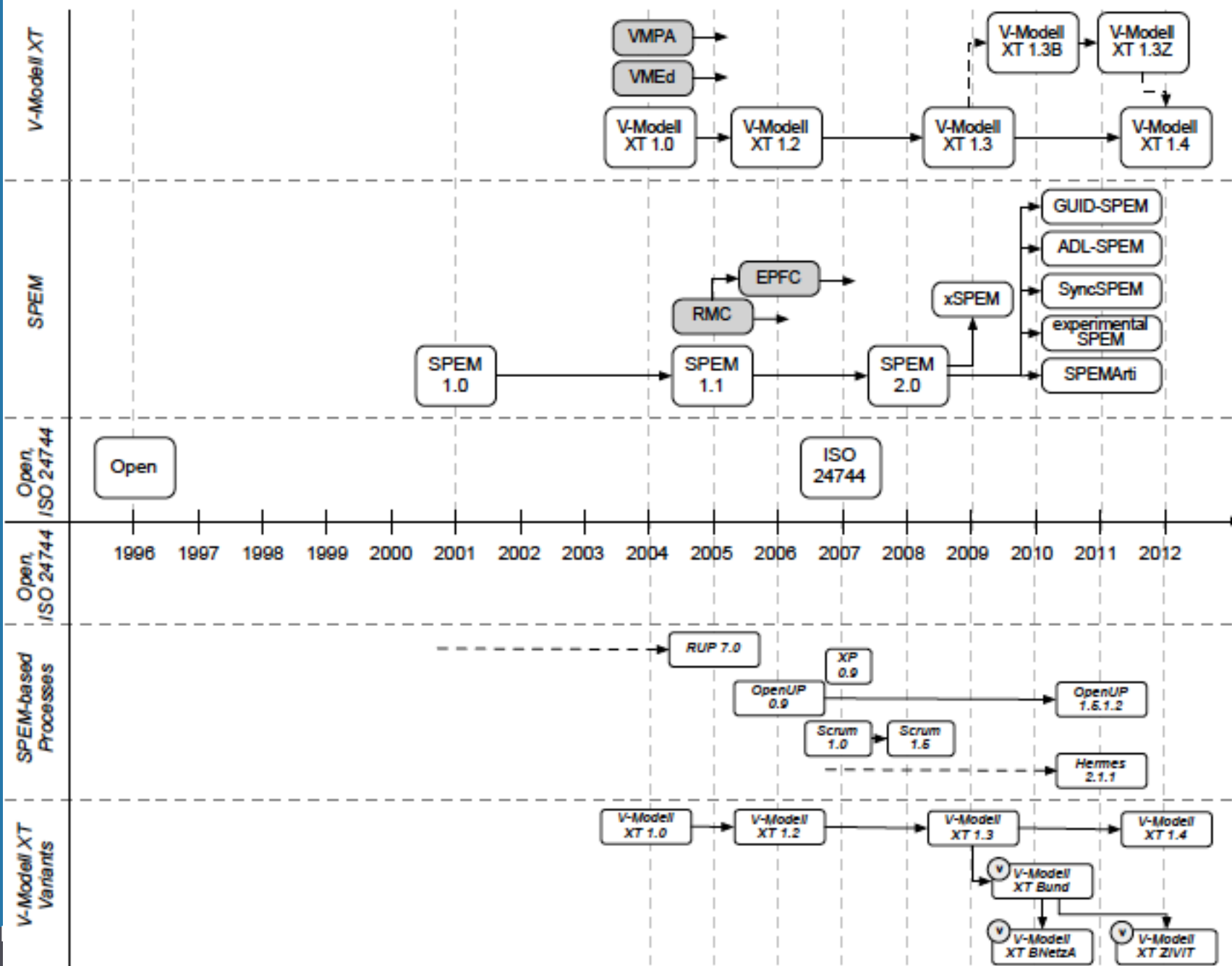
[Expand All Steps](#) [Collapse All Steps](#)

[Identify Stakeholders](#)

Evolución de los Procesos de Desarrollo de Software

Ev

o [1]



Evolución de los Procesos de Desarrollo [2]

- Según el artículo presentado por Kuhrmann et al. (2013), existe una fuerte demanda por la investigación de problemas relacionados a los Procesos de Desarrollo en general.
- Estas investigaciones incluyen como insertar estándares aprobados por la industria para mejorar los procesos de desarrollo.
- Otro aspecto fundamental son los paradigmas de modelamiento



UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA



Departamento de Informática
Universidad Técnica Federico Santa María

Procesos de Desarrollo y Ciclo de Vida del Software

Ingeniería de Software

Hernán Astudillo & Gastón Márquez
Departamento de Informática
Universidad Técnica Federico Santa María