Preparación Ingeniería de Software CENEVAL (Ingeniería de Requerimientos, Verificación y Validación de Software)

Juan Carlos Lavariega Jarquín
Departamento de Computación
A7-415
lavariega@itesm.mx
https://sites.google.com/site/cenevalbd/

CONTENIDO

- Ingeniería de requerimientos (3-50)
  - Definiciones generales.
  - Requerimientos Funcionales y No Funcionales
  - Documento de Requerimientos
  - Proceso de Ing. Requerimientos
  - Adquisición de Requerimientos
  - Validación y Mantenimiento
- Verificación y Validación de Software (51-142)
  - Definiciones generales
  - Aseguramiento de la Calidad en productos de software
  - Atributos de Calidad
  - Verificacion y Validacion estática
  - Verificacion y Validacion dinámica
  - Pruebas

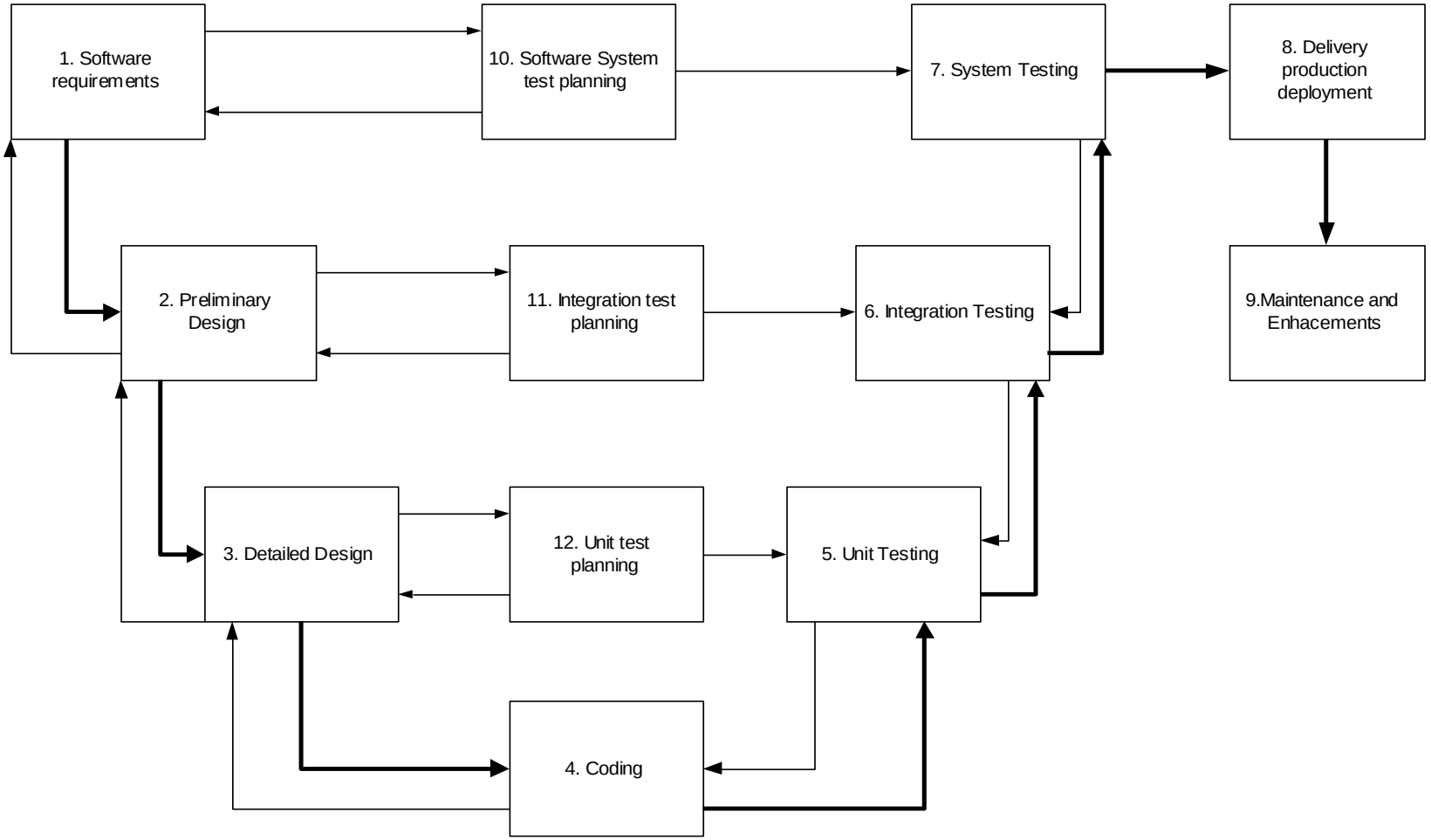
Ingeniería de Software

INGENIERIA DE REQUERIMIENTOS

Ingeniería de Software

- Ingeniería de Software es la aplicación de principios científicos a:
  - La transformación ordenada de problemas en una solución de software operacional (software funcionando)
  - El mantenimiento subsiguiente de ese software hasta el fin de su vida útil.

Proceso de desarrollo de Software (modelo V)



Proceso de desarrollo de Software

- Ingeniería de Requerimientos
  - Analizar el problema de software. Concluye con la especificación completa del comportamiento deseado del sistema a construir
- Diseño Preliminar
  - Descompone el sistema de software en sus componentes arquitectónicos e iterativamente descompone esos componentes en subcomponentes más y más pequeños (50-200 líneas de código).Cada subcomponente es documentado en términos de sus entradas, salidas y funciones.
- Diseño Detallado
  - Define y documenta algoritmos para cada modulo que se transformará en código.
- Codificación
  - Transformar los algoritmos definidos durante el diseño detallado en algún lenguaje computacional.
  - *Pero humanos no somos perfectos*

Proceso de desarrollo de Software

- Pruebas Unitarias
  - Revisar cada módulo por la presencia de bugs (module testing, functional testing)
- Pruebas de Integración
  - Conectar (integrar) conjuntos de modulos previamete probados independientemente .
- Pruebas del Sistema
  - Chequear que la totalidad (completamente integrado) del sistema de software integrado con el hardware en que operará se comporta de acuerdo a la especificación de requerimientos de software.
- Entrega, producción e instalación
  - Después de pruebas finales el software y el hardware que lo complementa son operacionales
- Mantenimiento y Mejoras
  - Continua detección y reparación de bugs y adición de nuevas capacidades

Proceso de desarrollo de Software

- Es inapropiado esperar hasta la etapa de pruebas para determinar como se harán éstas.
- Planeación de pruebas de Sistema
  - Determina como el sistema de software se probará para cumplir con los requerimientos. Desarrollo y documentación de planes de prueba. Examinar el documento de SRS para determinar si es verificable.
- Planeación de Pruebas de Integración
  - Genera documentos y procedimientos para efectuar una integración ordenada del sistema.
- Planeación de pruebas Unitarias
  - Genera documentos y procedimientos para probar cada módulo independientemente y exhaustivamente.
  -

Ingeniería de Requerimientos

- El proceso de establecer los servicios que el cliente requiere de un sistema y los limites bajo los cuales opera y se desarrolla.
- Los Requerimientos pueden ser Funcionales o No-Funcionales
  - Los Requerimientos funcionales describen servicios o funciones
  - Los Requerimientos No-funcionales son un límite en el sistema o en el proceso de desarrollo.

¿Qué es un Requerimiento?

- Es un rango de instrucciones abstractas de alto nivel de un servicio o de un sistema, limitado a detallar una especificación funcional matemática.
- Una capacidad del software requerida por el usuario para resolver un problema y lograr un objetivo
- Una capacidad del sw que debe ser alcanzada o poseída por un sistema o componente de un sistema para satisface un contrato, estándar, especificación u otra documentación formal impuesta.
- Así es inevitable como los Requerimientos pueden servir en una función dual
  - Puede ser la base para una declaración de un contrato, por lo tanto, deber estar abierto a interpretación.
  - Puede ser la base para el contrato en sí, por lo tanto, debe ser definido en detalle.
  - Ambas declaraciones serán llamadas Requerimientos.

¿Qué es un Requerimiento?

- Define que es lo que debe hacer el sistema y bajo que restricciones (o condiciones) va a operar.
  - El sistema debe mantener registro de todos los materiales en la biblioteca, incluidos libros, seriales, periódicos, y revistas, video y audio tapes, colecciones de transparencias, discos de computadoras y CD-ROMs (funcional)
  - El sistema debe permitir a los usuarios buscar un elemento por titulo, author, o ISBN. (funcional)
  - La interfaz de usuario del sistema deberá ser implantada usando un navegador de WWW. (no funcional)
  - El sistema debe soportar al menos 20 transacciones por segundo. (no funcional)
  - Las facilidades del sistema que estén disponibles a los usuarios debera ser demostrables en 10 minutos o menos (no funcional)

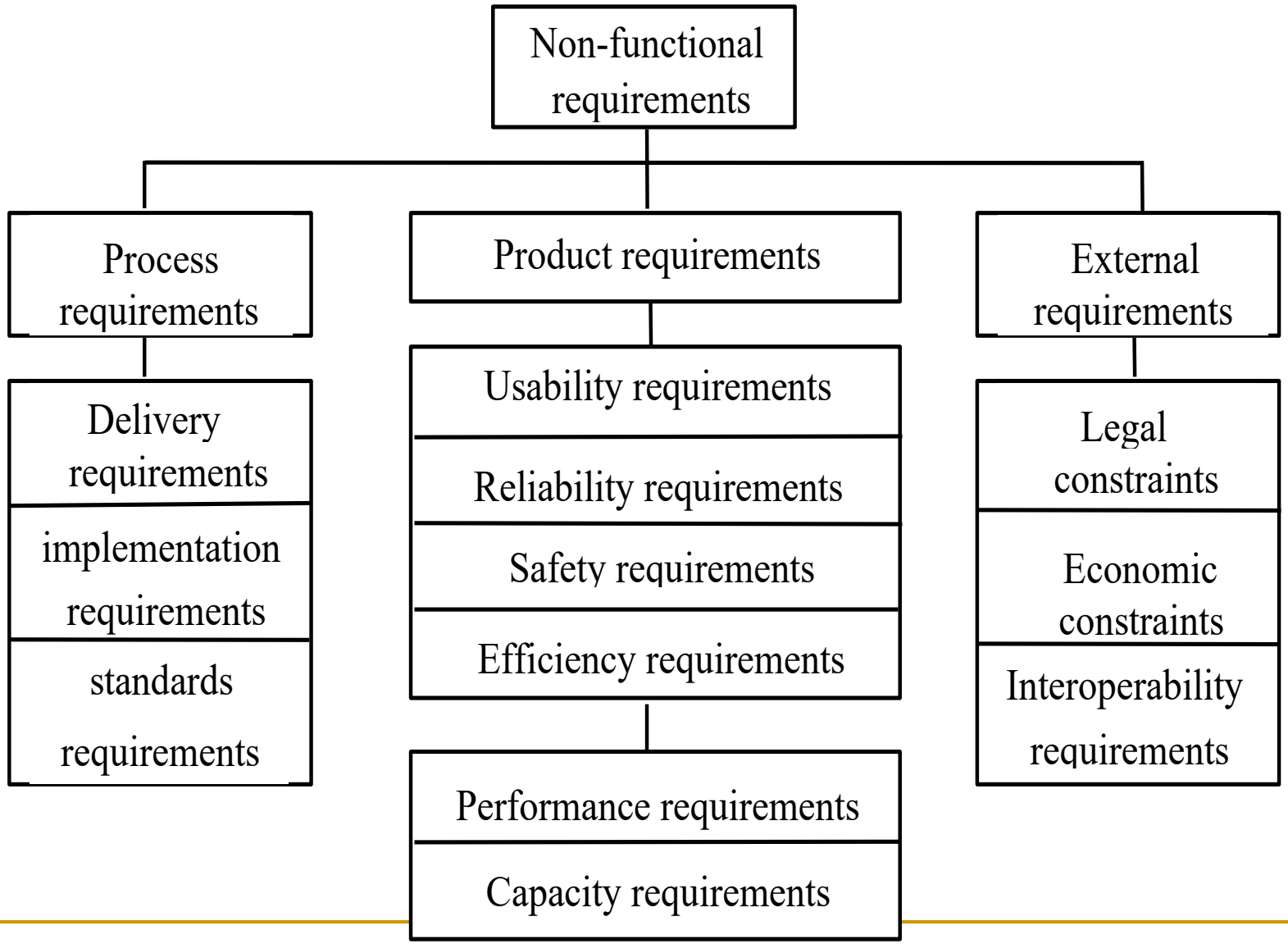
Tipos de Requerimientos

- Requerimientos que definen en términos muy generales los que el sistema deberá hacer.
- Requerimientos funcionales que definen parte de la funcionalidad del sistema.
- Requerimientos de implantación que indican como deberá ser implantado el sistema.
- Requerimientos de Rendimiento (Performance) que especifican cual es el rendimiento mínimo aceptable para el sistema.
- Requerimientos de Uso (Usability) que especifican el máximo tiempo aceptable para demostrar el uso del sistema.

Propiedades Sobresalientes (Reqts NoFuncionales)

- Las propiedades sobresaliente son las propiedades de un sistema considerado como un todo, y estas propiedades sobresalen una vez que todos los subsistemas han sido integrados.
- Ejemplos
  - Confiabilidad (Reliability)
  - Mantenible (Maintainability)
  - Rendimiento (Performance)
  - Uso (Usability)
  - Seguro (Security, Safe)

Clasificación de Req. No Funcionales



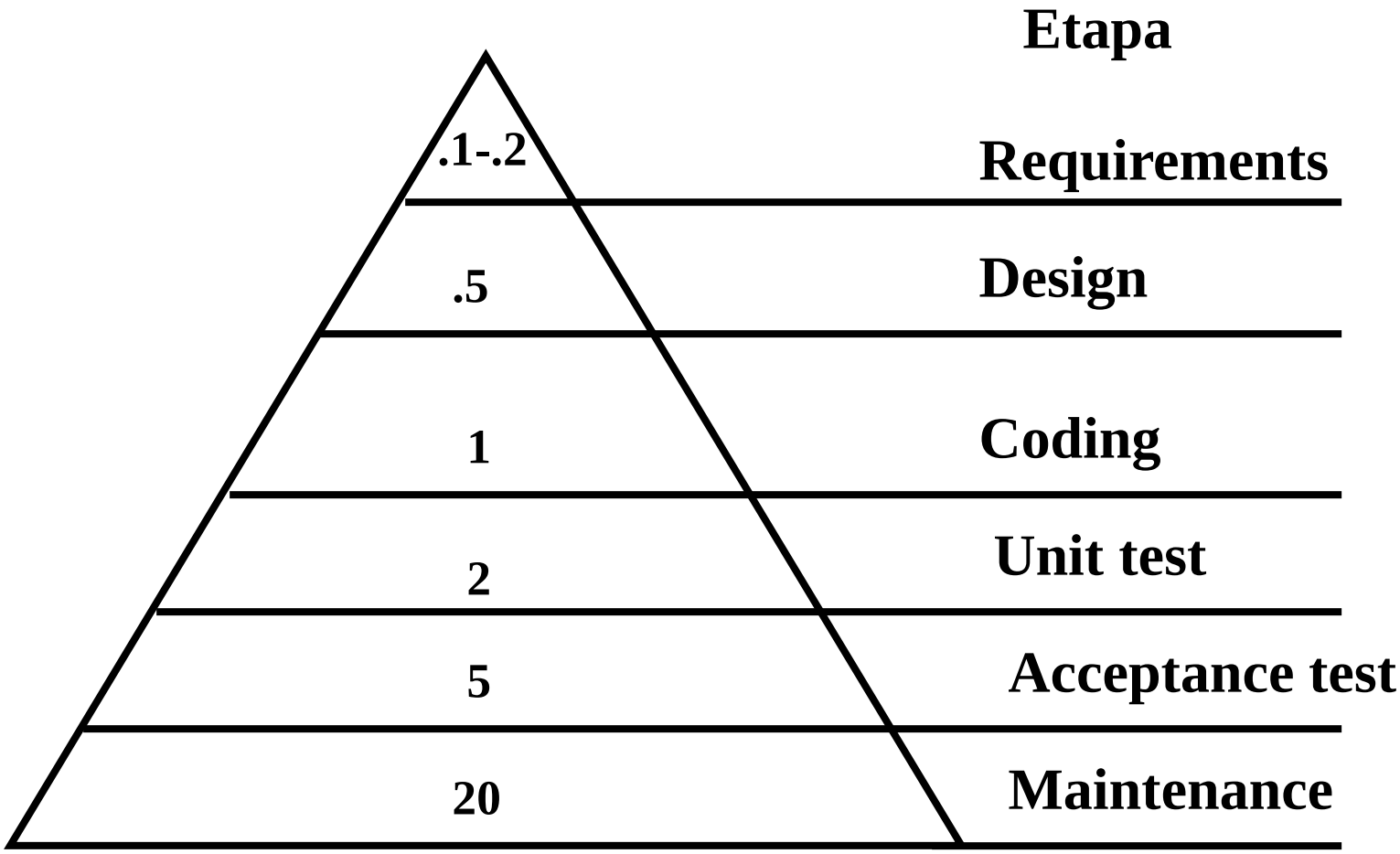
Relación entre Atributos de Calidad

	Availability	Efficiency	Flexibility	Integrity	Interoperability	Maintainability	Portability	Reliability	Reusability	Robustness	Testability	Usability
Availability								+		+		
Efiiciency			-		-	-	-	-		-	-	-
Flexibility		-		-		+	+	+			+	
Integrity		-			-				-		-	-
Interoperability		-	+	-			+					
Maintainability	+	-	+					+			+	
Portability		-	+		+	-			+		+	-
Reliability	+	-	+			+				+	+	+
Reusability		-	+	-	+	+	+	-			+	
Robustness	+	-						+				+
Testability	+	-	+			+		+				+
Usability		-								+	-	

El Problema de Requerimientos

- Mientras más tarde en detectarse un error en el ciclo de vida de un producto será más caro repararlo
- Muchos errores no se encuentran sino hasta mucho tiempo después de haber sido producidos
- Hay muchos errores de requerimientos
- Los errores típicos de requerimientos son hechos incorrectos, omisiones, inconsistencias, y ambigüedades
- Los errores en requerimientos pueden ser detectados

Costo Relativo de Correciones



Requerimientos Definición/Especificación

- Definición de Requerimientos
  - Una declaración en un Lenguaje Natural incluye los diagramas de los servicios del sistema y sus límites operacionales. Escrito para clientes.
- Especificación de Requerimientos
  - Un documento estructurado con descripción o detalle de los servicios del sistema. Escrito como un contrato entre el cliente y el contratista.
- Especificación de Software
  - Descripción detallada de software, la cual, puede servir como una base para diseño o implementación. Escrito para desarrolladores.

Definiciones y Especificaciones

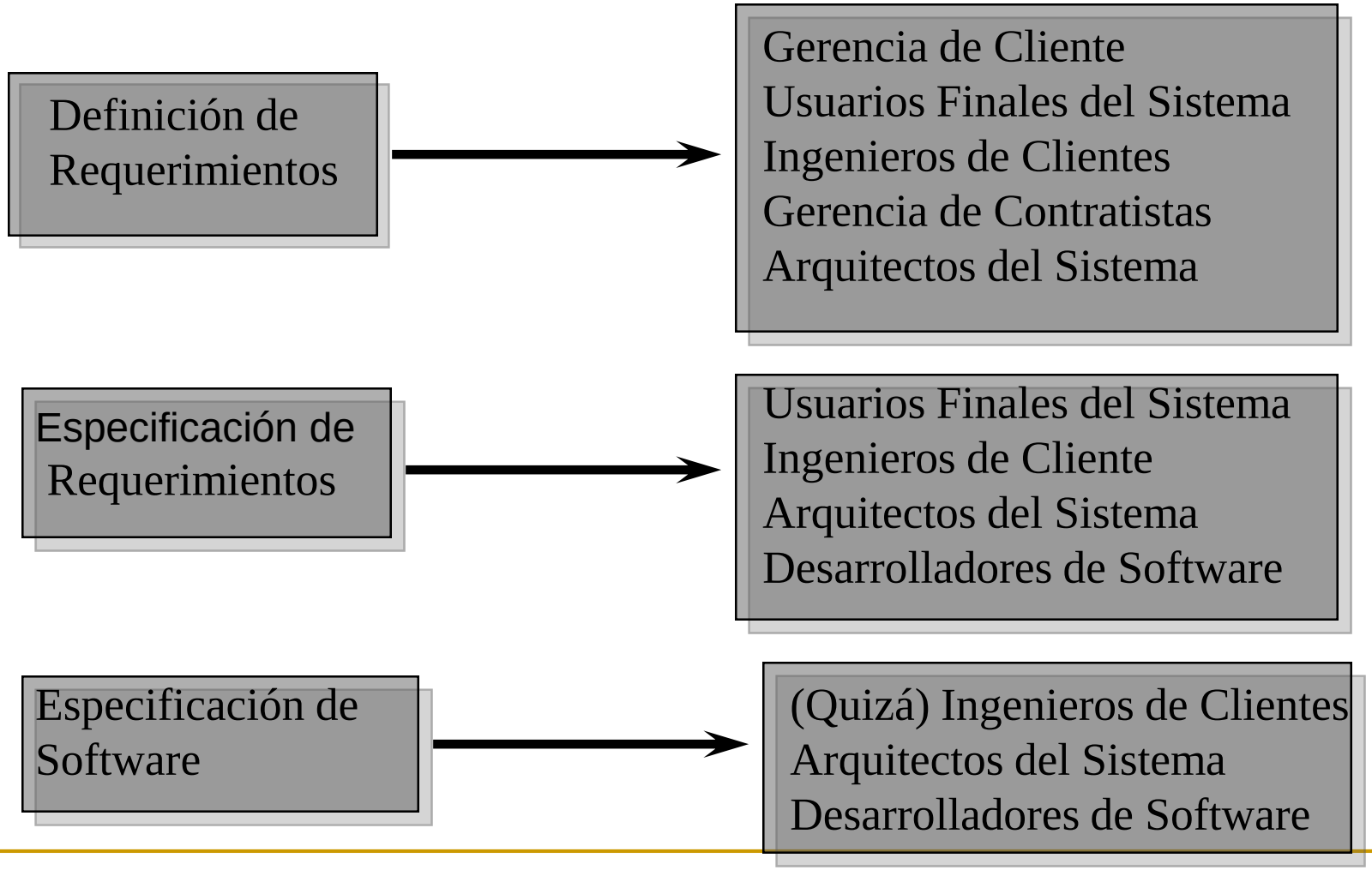
Definición de Requerimientos

1. El Software proporciona significado de representación y acceso a archivos externos creados por otras herramientas.

Especificación de Requerimientos

1.1 El usuario debe proporcionar facilidades para definir el tipo de archivos externos.
1.2 Cada tipo de archivo externo puede tener una herramienta asociada. La cual, será aplicada para el archivo.
1.3 Cada tipo de archivo externo será representado como un icono específico mostrado al usuario.
1.4 Las facilidades proporcionadas para la representación del icono en un tipo de archivo externo será definido por el usuario.
1.5 Cuando un usuario selecciona una representación de icono de un archivo externo, el efecto de la selección es aplicar las herramientas asociadas con el tipo de archivo externo al archivo representado por la selección del icono.

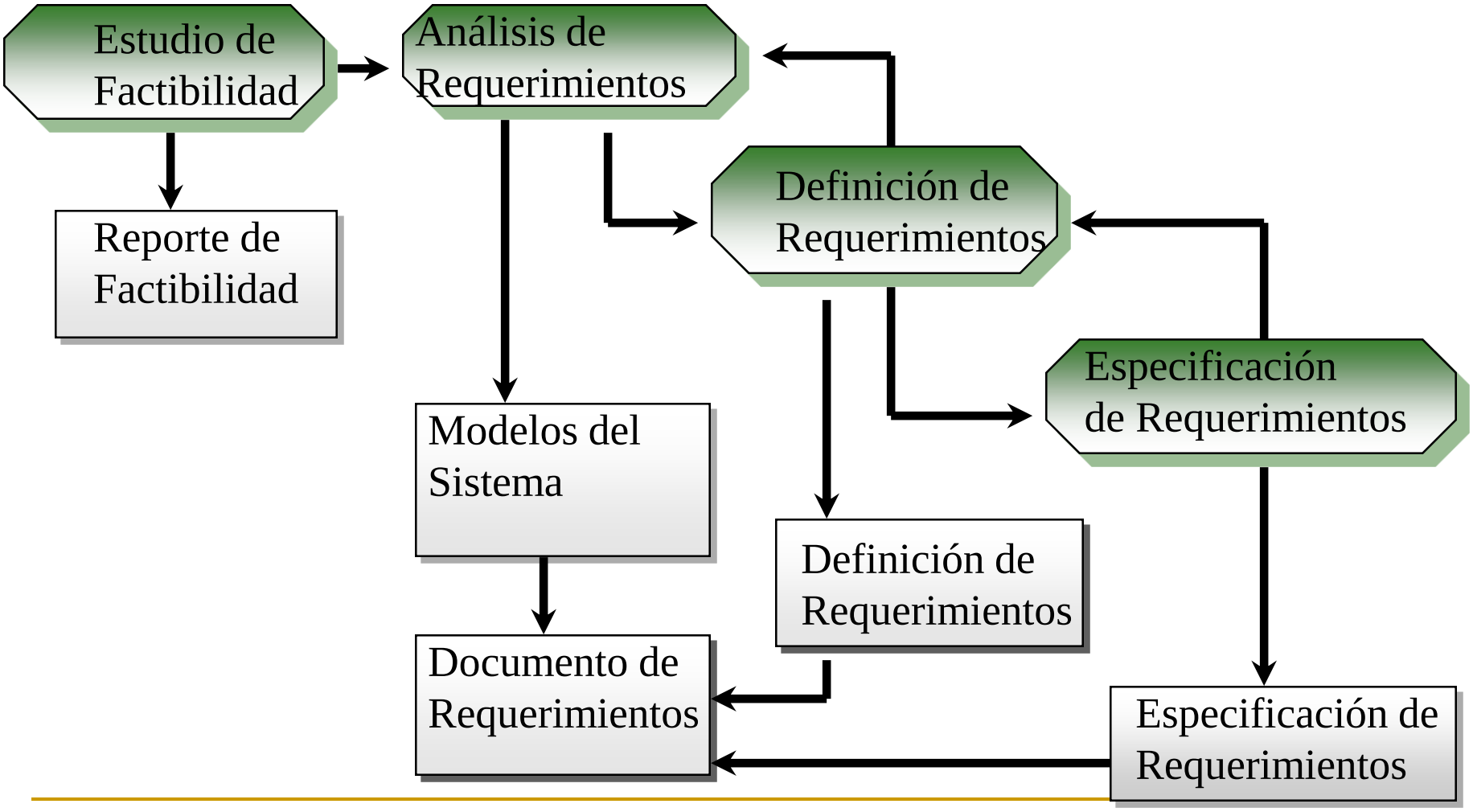
Lectores de Requerimientos



Proceso de Ingeniería de Requerimientos

- Estudio de Factibilidad
  - Encuentran los usuarios actuales que sus necesidades son satisfechas dada la tecnología y el presupuesto disponible?
- Análisis de Requerimientos
  - Encontrar que el sistema requiere del mantenimiento de intereses.
- Definición de Requerimientos
  - Definir los requerimientos en una forma comprensible para el cliente.
- Especificación de Requerimientos
  - Define los requerimientos en detalle.

El Proceso de Ingeniería de Requerimientos



Análisis de Requerimientos

- El proceso de estudiar las necesidades del usuario para llegar a una definición de los requerimientos del sistema o del software
- La verificación de los requerimientos del sistema/software
- Es un proceso iterativo de:
  - Identificación
  - Análisis
  - Documentación
  - Verificación

  Para llegar a una definición del comportamiento requerido del sistema

Metas del Análisis de Requerimientos

- Proceso
  - Mantener el proceso bajo nuestro control intelectual en todo momento
- Artefactos
  - Organizar los artefactos de tal forma que permita a otros comprender el producto a ser construido
  - La cantidad de esfuerzo debe ser proporcional al tamaño del producto

Documento de Requerimientos

- El documento de requerimientos es un documento formal usado para comunicar los requerimientos a clientes, ingenieros y administradores.
- El documento de requerimientos describe:
  - Los servicios y funciones que el sistema deberá proveer.
  - Las restricciones bajo las cuales el sistema deberá operar
  - Las propiedades generales del sistema (i.e. restricciones en las propiedades “sobresalientes” del sistema)
  - Definiciones de otros sistemas con los cuales el sistema deberá integrarse.
  - Información sobre el dominio de la aplicación.
  - Restricciones en el proceso de desarrollo.
  - Restricciones en equipo (hardware) en que correrá el sistema

Usuarios del Documento de Requerimientos

- Clientes del Sistema.
  - Especifican los requerimientos y los leen para verificar que cumplen con sus necesidades.
- Administradores del Proyecto
  - Usan el documento para planear un costo del sistema y el proceso de desarrollo del mismo.
- Ingenieros de sistemas
  - Usa el documento para entender el sistema siendo desarrollado.
- Ingenieros de Pruebas del sistema
  - Usa el documento para desarrollar pruebas de validación del sistema.
- Ingenieros del mantenimiento del sistema.
  - Use el documento para entender el sistema.

Estructura del Documento de Requerimientos

- IEEE/ANSI 830-1993 es el estándar propuesto para organizar la estructura del documento de requerimientos.
- Introducción
  - 1.1 Propósito del Documento de Requerimientos
  - 1.2 Alcance del producto
  - 1.3 Definiciones, acrónimos y abreviaciones
  - 1.4 Referencias
  - 1.5 Panorama del resto del documento

Estructura del Documento de Requerimientos

- 2. Descripción General
  - 2.1 Perspectiva del producto
  - 2.2 Funciones del producto
  - 2.3 Características del usuario
  - 2.4 Restricciones generales
  - 2.5 Supuestos y dependencias.
- 3. Requerimientos específicos
  - Cobertura de requerimientos funcionales, no-funcionales y de interfaz.
- 4. Apéndices
- Índice

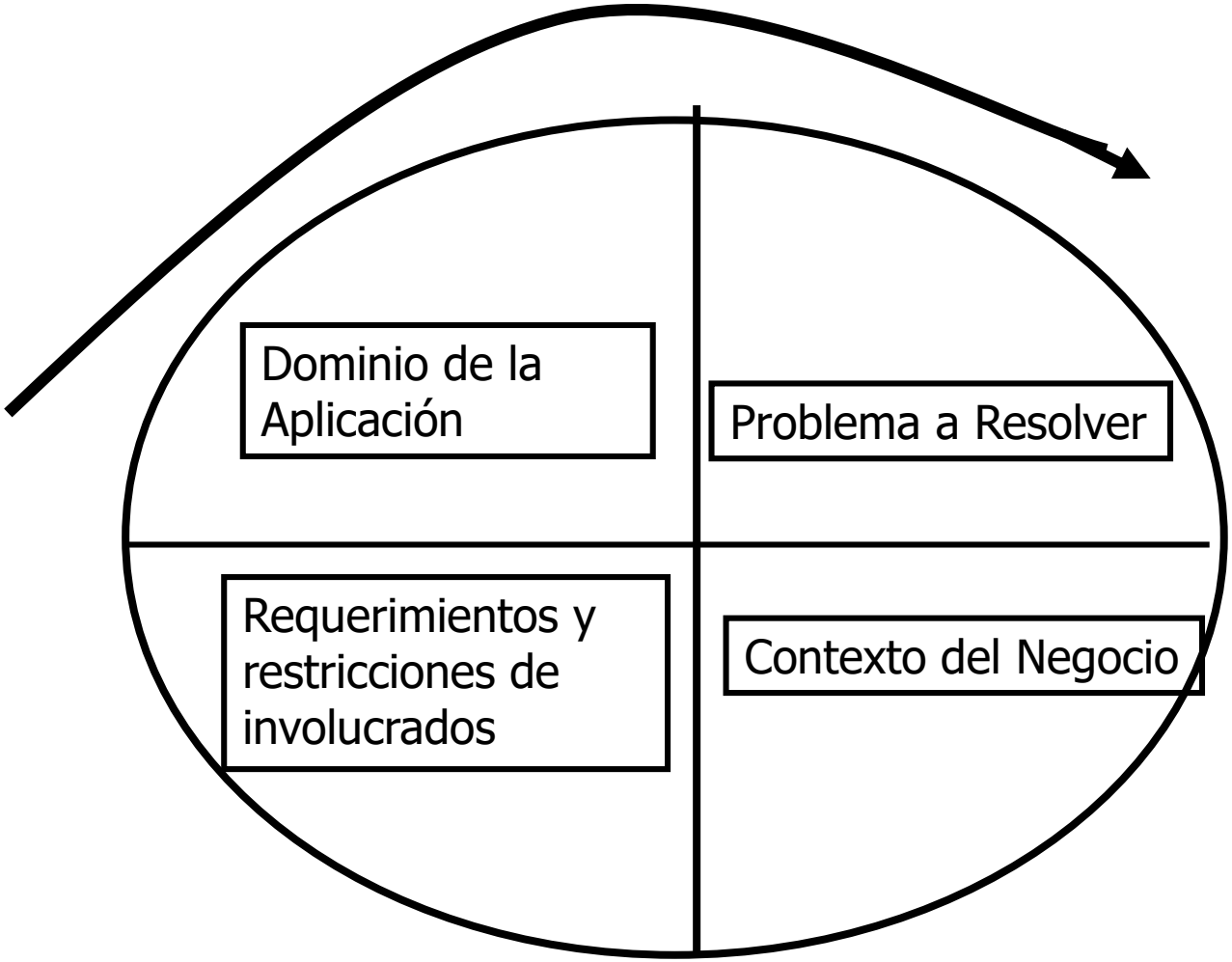
Estructura del Documento de Requerimientos

- Introducción.
  - Describe la necesidad de crear el sistema y cuales son sus objetivos.
- Glosario.
  - Define los términos técnicos usados.
- Modelos del Sistema.
  - Define los modelos que muestran los componentes del sistema y las relaciones entre ellos.
- Definición de Requerimientos Funcionales.
  - Define los servicios que serán proporcionados.

Estructura del Documento de Requerimientos

- Definición de Requerimientos No-funcionales.
  - Definir las limitantes del sistema y el proceso de desarrollo.
- Evolución del Sistema.
  - Definir las suposiciones fundamentales en las cuales el sistema se basa y se anticipan los cambios.
- Especificación de Requerimientos.
  - Especificación detallada de los requerimientos funcionales del sistema.
- Apéndices.
  - Descripción de la plataforma de Hardware del Sistema.
  - Requerimientos de la base de Datos (quizá como un modelo ER)
- Indice.

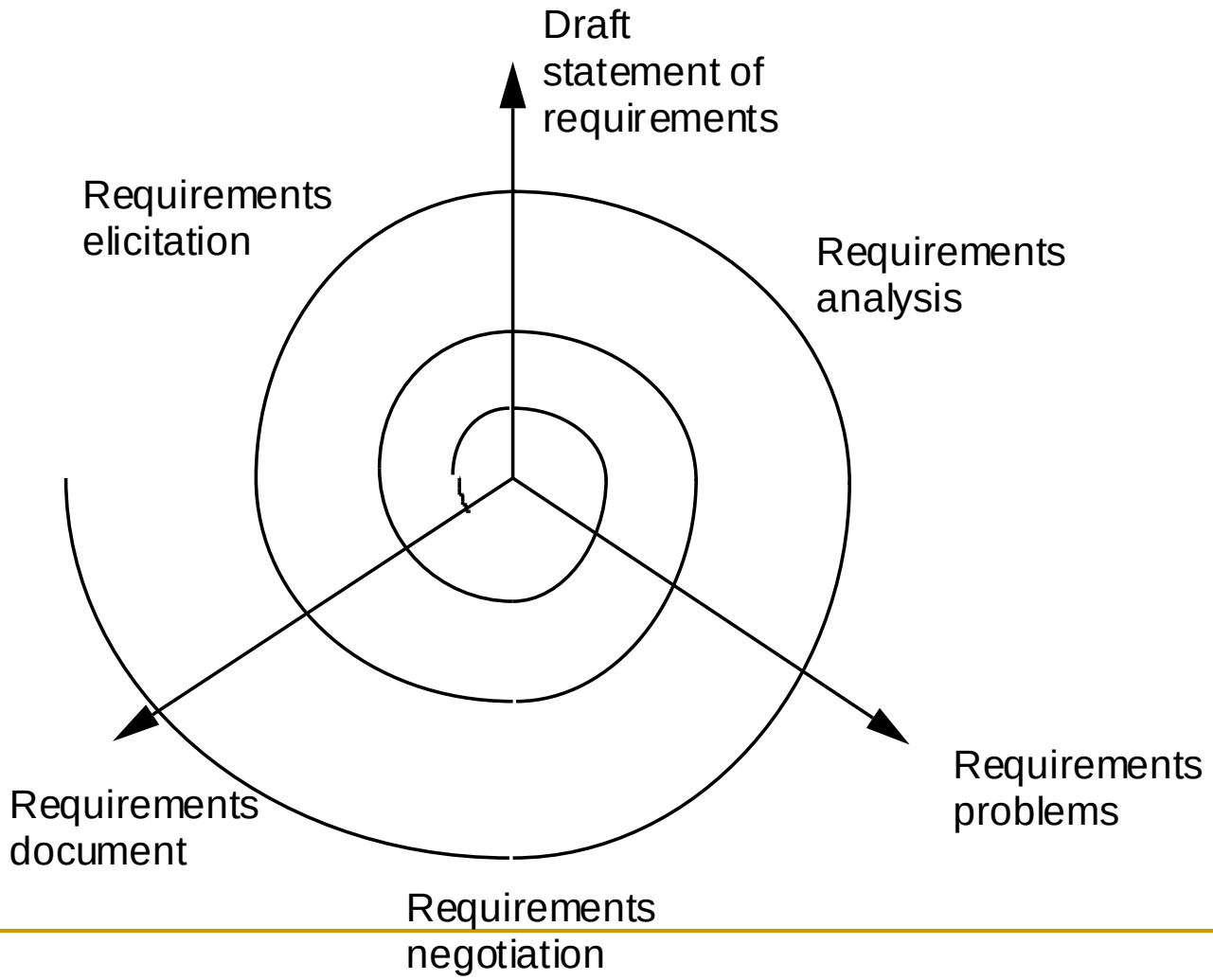
Componentes de adquisición de requerimientos



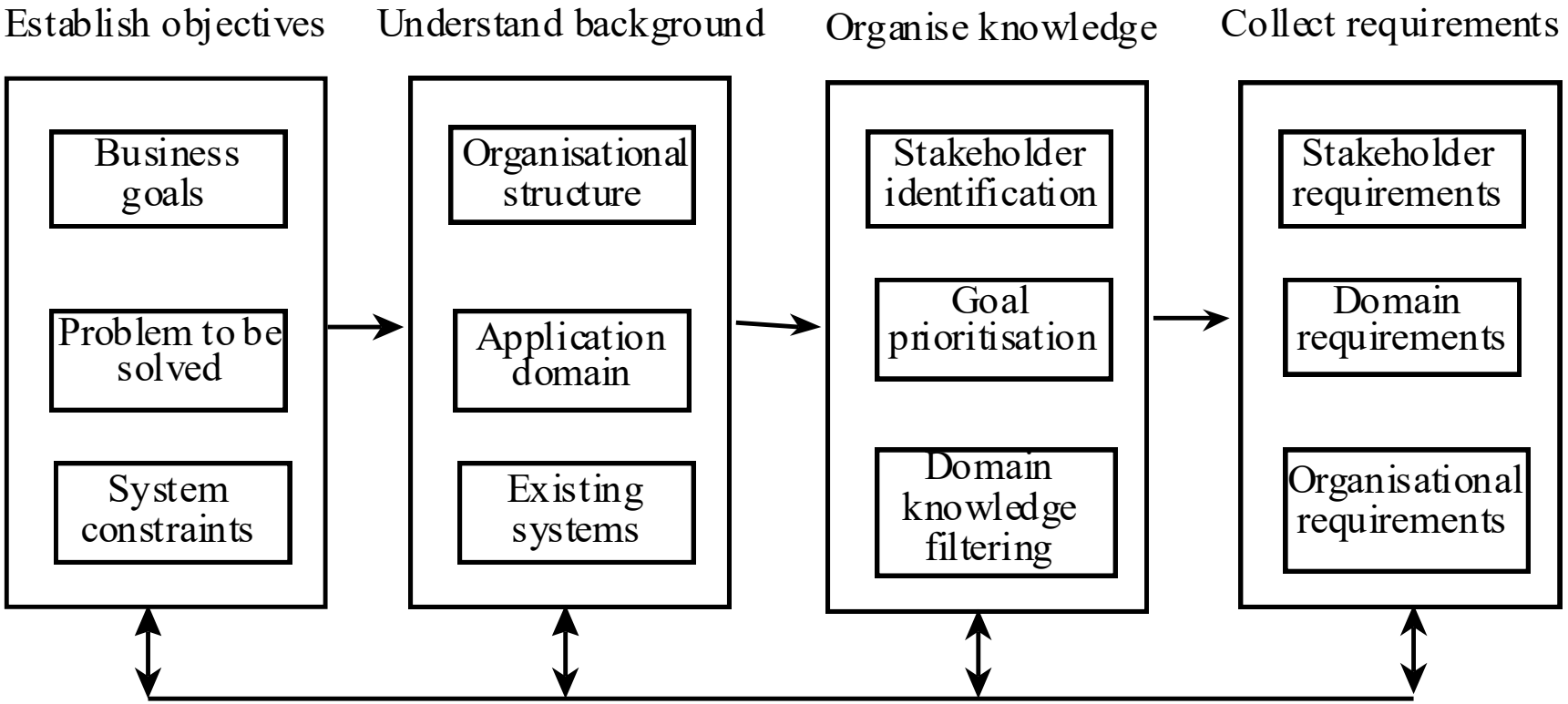
Actividades de adquisición

- Entendimiento del dominio de la aplicación.
  - Conocer el dominio de la aplicación es conocer las áreas generales donde el sistema se aplicará..
- Entendimiento del Problema
  - Los detalles del problema especifico del cliente donde el sistema será aplicado debe ser entendido.
- Entendimiento del Negocio
  - Se debe entender como interactuará el sistema y contribuirá a las metas generales del negocio.
- Entendimiento de las necesidades y restricciones de los involucrados en el sistema.
  - Se debe entender en detalle, las necesidades especificas de la gente que requiere un sistema para apoyar su trabajo.

Adquisición, Análisis y Negociación.



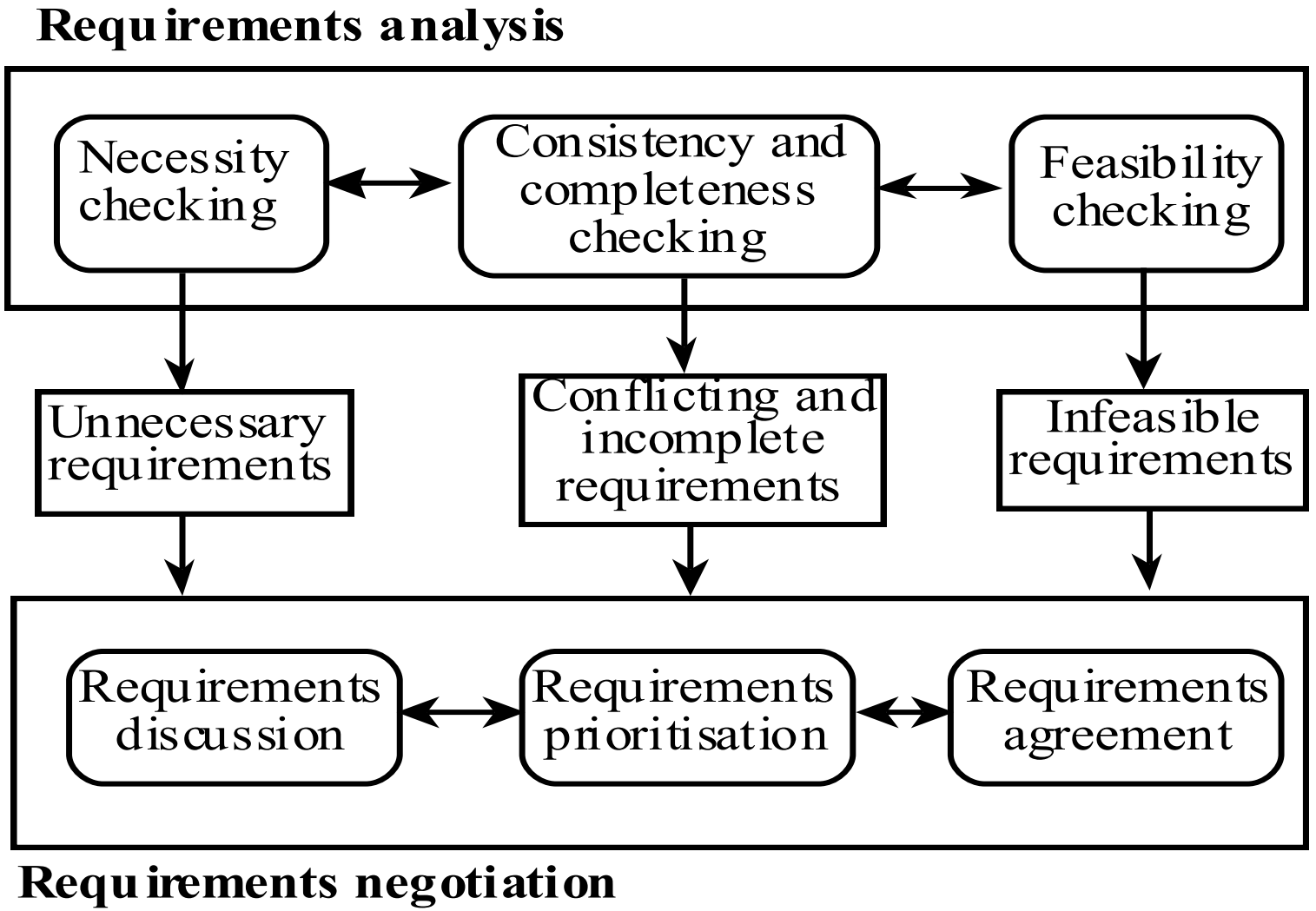
El proceso de adquisición de requerimientos.



Etapas de adquisición

- Definición de Objetivos
  - Se deben establecer los objetivos organizacionales, incluyendo las metas generales del negocio, una descripción general del problema a resolver, porque el sistema es necesario y las restricciones del sistema.
- Adquisición de conocimiento previo
  - Información previa acerca del sistema incluye información acerca de la organización donde el sistema será instalado, el dominio de la aplicación del sistema e información acerca de sistemas existentes.
- Organización de Conocimiento
  - La gran cantidad de conocimiento que ha sido recolectada en la etapa previa debe ser organizada y relacionada.
- Recolección de requerimientos de involucrados.
  - Los “accionistas” del sistema deben ser consultados para descubrir sus requerimientos

Análisis de Requerimientos y Negociación



Chequeos en Análisis

- Chequeo de Necesidades
  - La necesidad de los requerimientos es analizada. En algunos casos los requerimientos propuestos no contribuyen a las metas del negocio de la organización o del problema especifico a ser tratado en el sistema.
- Chequeo por consistencia y completitud.
  - Los requerimientos se revisan por consistencia y completitud. Consistencia significa que los requerimientos no serán contradictorios. Completitud significa que no hay servicios o restricciones que son necesariamente ausentes.
- Chequeo de factibilidad
  - Los requerimientos son chequeados para asegurar que son factibles en el contexto del presupuesto y tiempo de desarrollo disponible para el sistema.

Negociación de Requerimientos

- Discusión de Requerimientos
  - Los requerimientos que han sido identificados como problemáticos se discutidos y los involucrados deben presentar sus puntos de vista acerca de los mismos.
- Asignación de Prioridades de los Requerimientos
  - A los requerimientos en disputa se les asigna prioridades para identificar cuales son críticos y ayudar en el proceso de toma de decisiones.
- Acuerdo en Requerimientos
  - Se identifican soluciones a los problemas en los requerimientos y se llega a un acuerdo en un conjunto de requerimientos. Generalmente esto involucra hacer cambios a algunos requerimientos.

Técnicas de Adquisición.

- Técnicas especificas que pueden ser utilizadas para recabar conocimiento acerca de los requerimientos del sistema.
- El conocimiento que se recabe debe ser estructurado o estructurarse
  - Particiones – agregación de cocimientos relacionados
  - Abstracción – reconocimiento de generalidades
  - Proyección – organización de conocimiento de acuerdo a perspectivas.
- Problemas en la Adquisición
  - No hay tiempo suficiente
  - Preparación inadecuada de ingenieros
  - Involucrados “accionistas” no están convencidos de la necesidad del nuevo sistema.

Técnicas especificas de adquisición

- Entrevistas
  - Abiertas , Cerradas, Cuestionarios
- Escenarios
  - Casos de uso, narrativas, historias de usuarios
- Métodos de sistemas suaves (soft-system)
- Observaciones y análisis social
- Reutilización de requerimientos
- Uso de Prototipos
  - Desecho (Throwaway)
  - Evolutivos

Validación de Requerimientos

- Demostración de que los requerimientos que definen el sistema son lo que el cliente realmente quiere.
- Los costos de errores en los requerimientos son altos, por lo cual, la validación es muy importante.
  - Fijar un error de requerimiento después del desarrollo puede resultar en un costo 100 veces mayor que fijar un error en la implementación.
- El Prototipeo es una técnica importante de la validación de requerimientos.

Chequeo de Requerimientos

- Validación. Provee al sistema las funciones que mejor soporten las necesidades del cliente?
- Consistencia. Existe cualquier conflicto en los requerimientos?
- Completo. Están incluidas todas las funciones requeridas por el cliente?
- Realismo. Pueden los requerimientos ser implementados con la tecnología y el presupuesto disponible?

Revisión de Requerimientos

- Una revisión regular puede ayudar mientras la definición de requerimientos está siendo hecha.
- Tanto el cliente como el staff de contratistas deben estar involucrados en la revisión.
- La revisión debe ser formal (con los documentos completos) o informal. Una buena comunicación entre desarrolladores, clientes y usuarios puede resolver problemas en las primeras etapas.

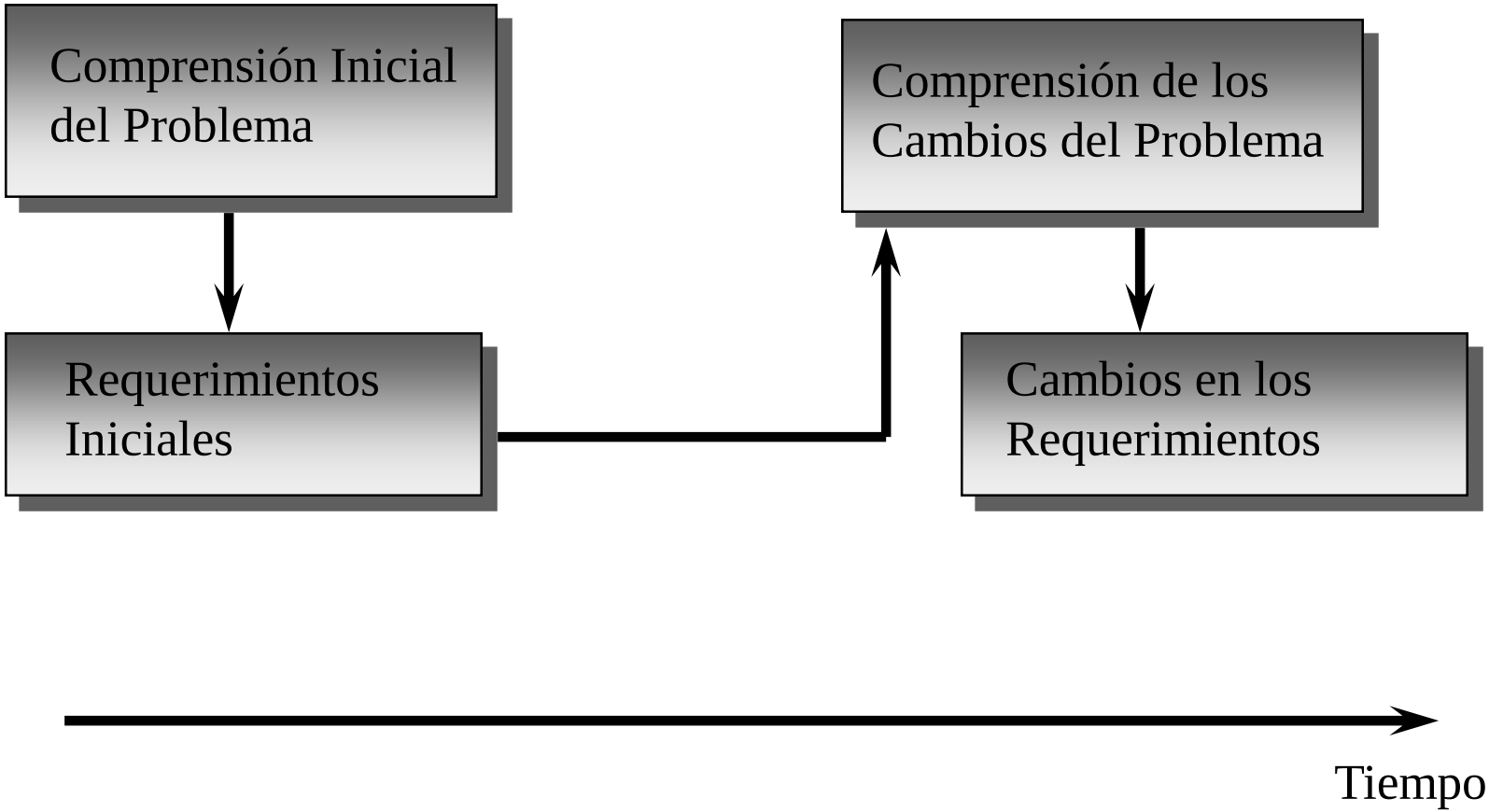
Chequeo de la Revisión

- Verificabilidad. Es el Requerimiento realmente probable?
- Entendibilidad. Es el Requerimiento comprendido propiamente?
- Probabilidad. Es el origen de los requerimientos claramente establecido?
- Adaptabilidad. Puede el requerimiento ser cambiado sin causar un gran impacto en otros requerimientos?

Evolución de Requerimientos

- Los requerimientos siempre involucran como comprender mejor el desarrollo de las necesidades de los usuarios y como los objetivos de la organización pueden cambiar.
- Es esencial planear posibles cambios en los requerimientos cuando el sistema sea desarrollado y utilizado.

Evolución de Requerimientos



Clases de Requerimientos

- Requerimientos Durables. Establecer requerimientos derivados de las actividades de la organización del cliente. Por ejemplo, un hospital siempre tendrá doctores, enfermeras, etc. Puede ser derivado de modelos de dominio.
- Requerimientos Volátiles. Los requerimientos cambian durante el desarrollo o cuando el sistema está en uso. En un hospital, los requerimientos se derivan de las políticas salud-cuidados.

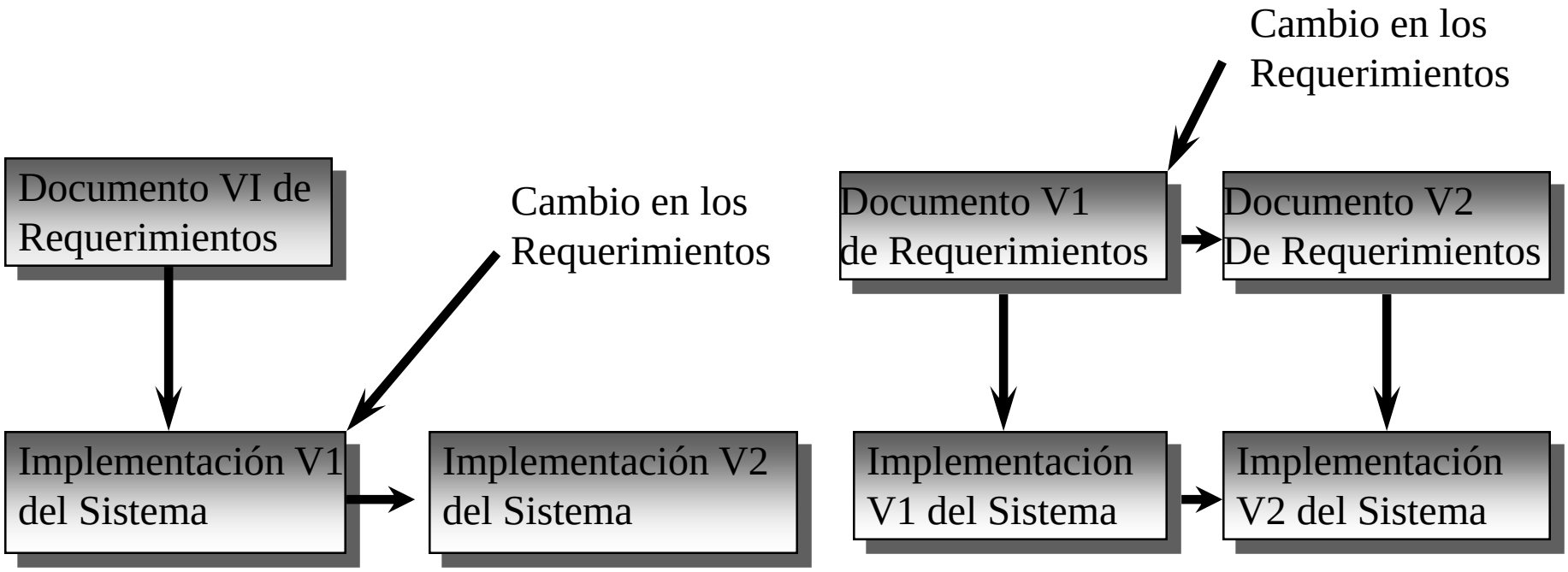
Clasificación de Requerimientos

- Requerimientos Cambiantes.
  - Los requerimientos que cambian por el ambiente del sistema.
- Surgimiento de los Requerimientos.
  - Requerimientos que surgen como una comprensión del desarrollo del sistema.
- Requerimientos en Consecuencial.
  - Requerimientos que resultan de la introducción del sistema a la computadora.
- Requerimientos Compatibles.
  - Requerimientos que dependen de otros sistemas o de otros procesos de la organización.

Cambios en el Documento de Requerimientos

- El documento de requerimientos debe ser organizado, de tal forma que los cambios en los requerimientos puedan ser hechos sin tener que re-escribir demasiado.
- Las referencias externas deben ser minimizadas y las secciones del documento deben ser tan modulares como sea posible.
- Los cambios son fáciles cuando se trata de un documento electrónico. Sin embargo, la falta de estándares para documentos electrónicos lo hace difícil.

Evolución Controlada



Inconsistencia de los Requerimientos y del Sistema

Consistencia de los Requerimientos y del Sistema

Ingeniería de Software

VERIFICACIÓN Y VALIDACION DE SOFTWARE

Aseguramiento de Calidad en Software

- Las actividades más importantes del aseguramiento de calidad están relacionadas a “asegurar que, al entregar un software, éste tenga muy pocos, si no es que ningún defecto”.

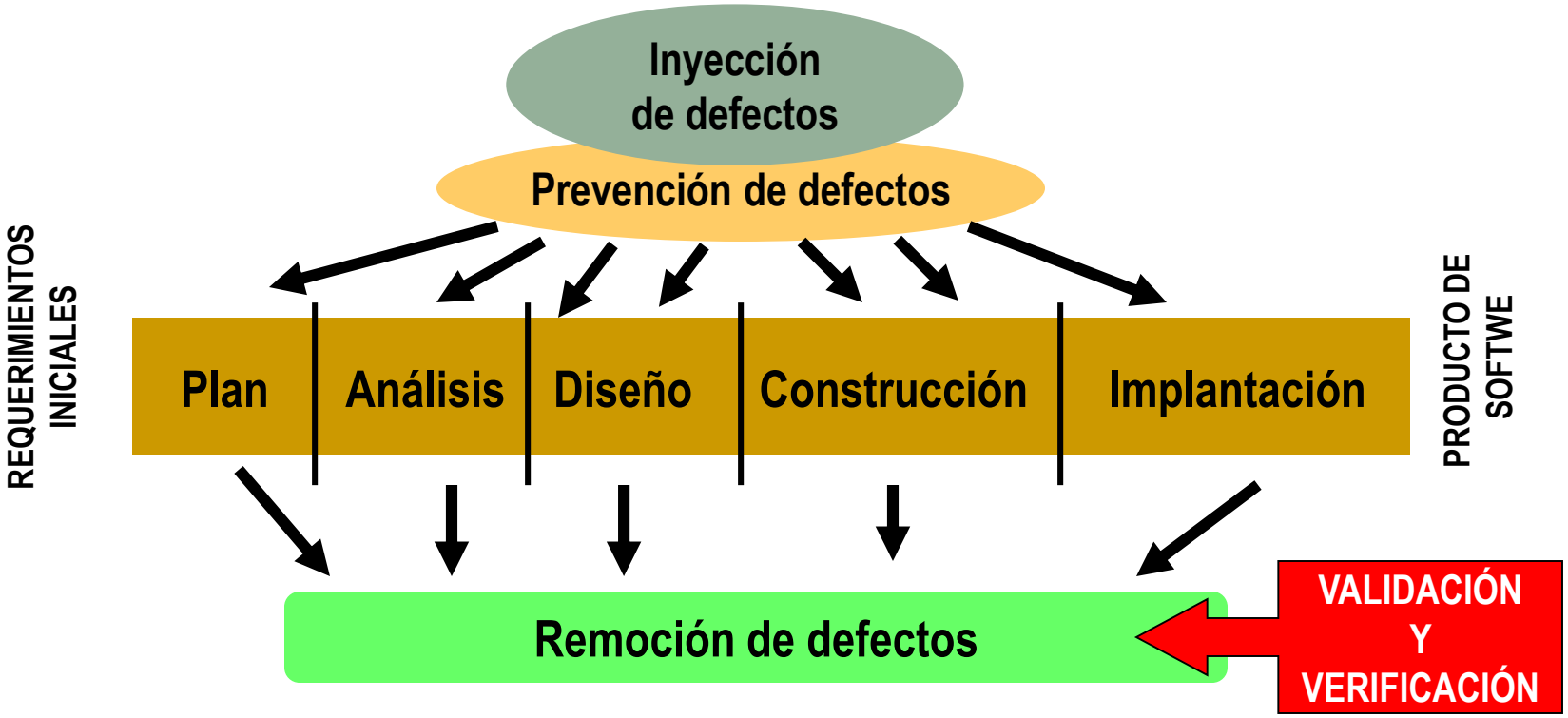
Aseguramiento de Calidad en Software

Aseguramiento de Calidad en Software

Métrica más aceptada

- Calidad SW =
  - Densidad de Defectos Entregados
  - Defectos / KLOC
- Defecto =
  - Inconsistencia con requerimiento
  - Defecto ≠ Error

Aseguramiento de Calidad en Software



Prevención de defectos

- El proceso de prevenir defectos está, en muchas ocasiones ligado a la eliminación de “la fuente del error”. Por ejemplo:
  - Si la fuente del error se debe a una “confusión humana, pues se debe proceder a Entrenar y Educar.
  - Si la fuente del error se debe a Diseños imprecisos que se desvían de las especificaciones del producto, existen métodos formales para eliminar ambigüedades y para revisar la correspondencia.
  - También hay herramientas que pueden ayudar a disminuir la inyección de errores (ej: un editor inteligente).
  - Etc.

Remoción o Reducción de defectos

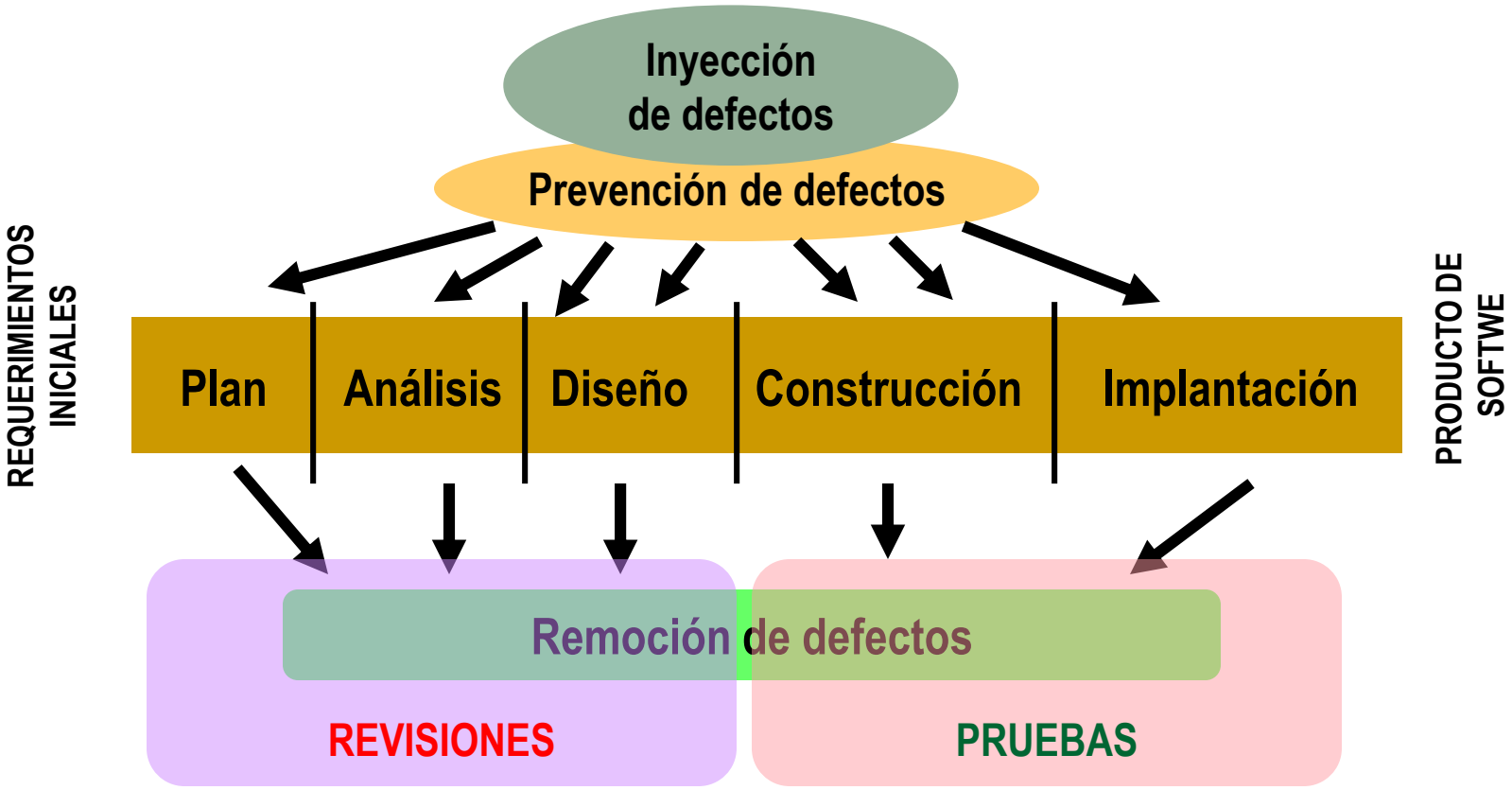
- El proceso de remoción (o al menos reducción) de defectos está compuesto por 2 actividades principales:
  - VERIFICACIÓN: Orientada al Proceso
    - ¿se está construyendo el producto en forma correcta? Do the things right.
  - VALIDACIÓN: Orientada al Producto
    - ¿se está construyendo el producto correcto? Do the right things.
  -

Definiciones según IEEE 1012 – 2004

- 3.1.35 validation:
- (A) The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.
- (B) The process of providing evidence that the software and its associated products satisfy system requirements allocated to software at the end of each life cycle activity, solve the right problem (e.g., correctly model physical laws, implement business rules, use the proper system assumptions), and satisfy intended use and user needs.

- 3.1.36 verification:
- (A) The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.
- (B) The process of providing objective evidence that the software and its associated products conform to requirements (e.g., for correctness, completeness, consistency, accuracy) for all life cycle activities during each life cycle process (acquisition, supply, development, operation, and maintenance); satisfy standards, practices, and conventions during life cycle processes; and successfully complete each life cycle activity and satisfy all the criteria for initiating succeeding life cycle activities (e.g., building the software correctly).

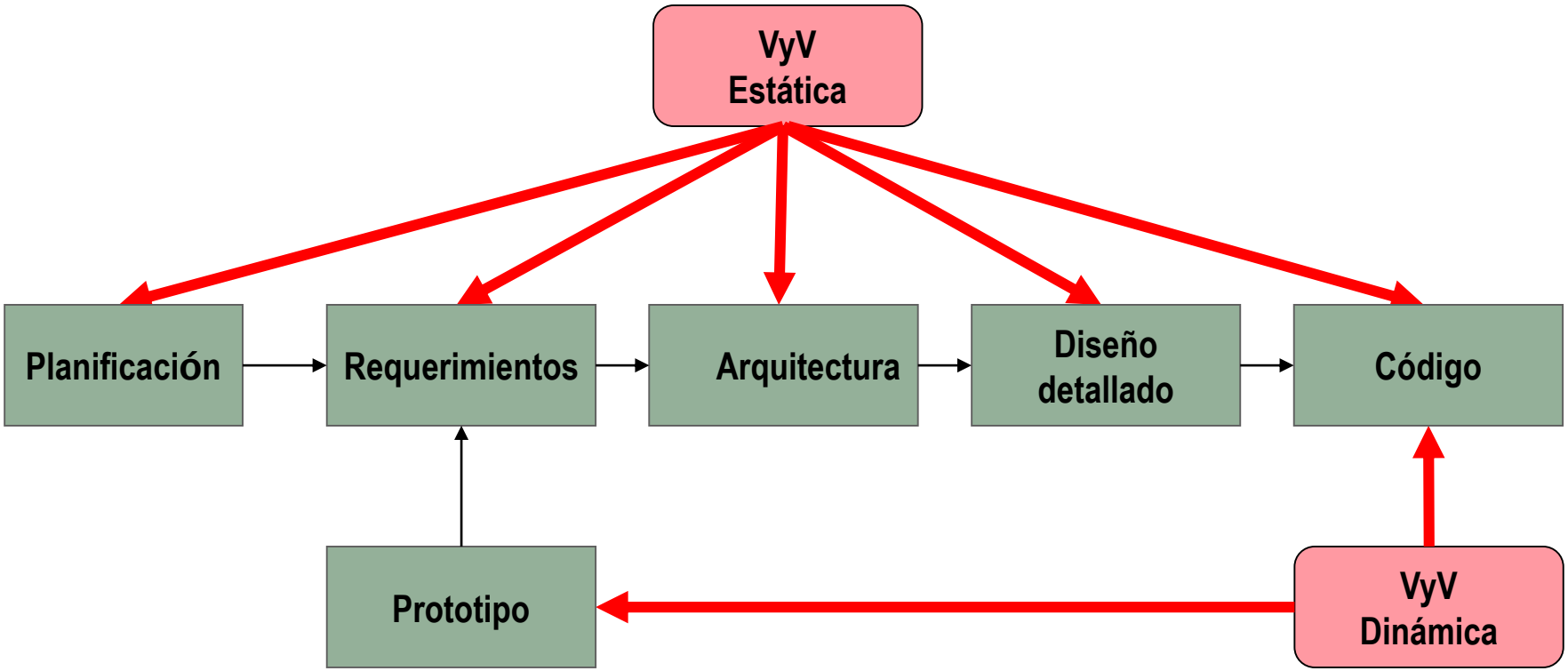
Aseguramiento de Calidad en Software



Verificación y Validación

- Se distinguen 2 tipos:
  - Verificación y Validación Dinámica
    - Ejercitar y observar comportamiento del producto
    - Normalmente = pruebas del sistema
  - Verificación y Validación Estática
    - Analizar representación estática del sistema
    - Normalmente = revisiones de los artefactos (productos) de trabajo

Verificación y Validación



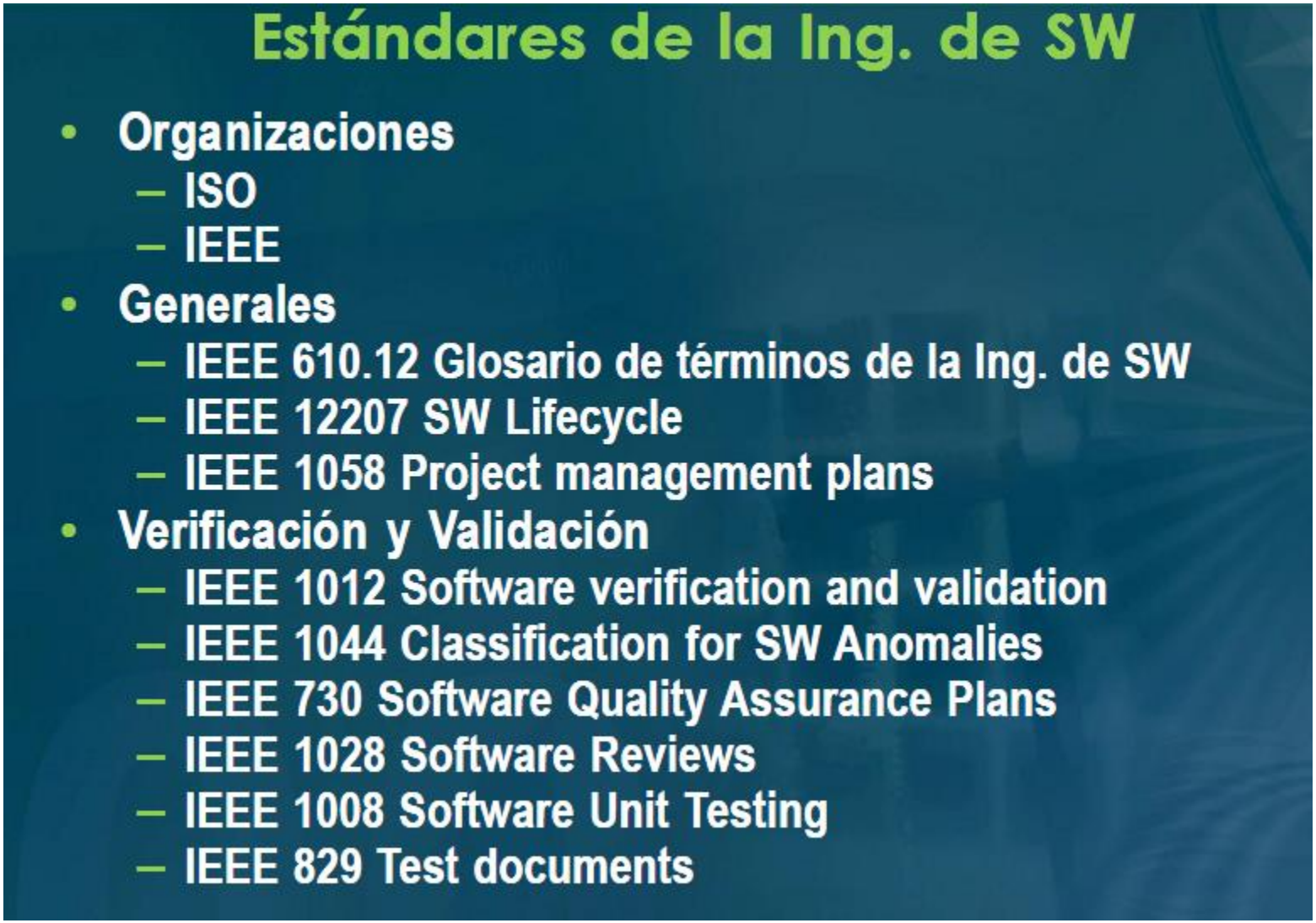
= ARTEFACTO = Producto de Trabajo

Atributos de Calidad

- El proceso de Desarrollo de Software tiene diferentes Vistas, asi como Roles y Responsabilidades, algo que es MUY importante es que éstas estén contenidas en un marco de referencia de calidad (Quality Framework).

- Hoy en día, el marco de referencia más utilizado en el área por la comunidad de Ingeniería de Software es el ISO-9126 .

(Recordatorio rápido de estándares)



Atributos de Calidad

- El ISO 9126 proporciona un conjunto de características como parte de los elementos de calidad de un software:
  - Functionality
  - Reliability
  - Usability
  - Efficiency
  - Maintainability
  - Portability

Atributos de Calidad

- Functionality: Conjunto de atributos relacionados a la existencia de un grupo de funciones que satisfacen las necesidades. Estos atributos incluyen:
  - Suitability
  - Accuracy
  - Interoperability
  - Security

Atributos de Calidad

- Reliability: Conjunto de atributos relacionados a la capacidad de mantener un cierto nivel de “performance” durante un período de tiempo. Estos atributos incluyen:
  - Maturity
  - Fault Tolerance
  - Recoverability

Atributos de Calidad

- Usability: Conjunto de atributos relacionados con el “esfuerzo” que requiere para ser usado. Estos atributos incluyen:
  - Understandability
  - Learnability
  - Operability

Atributos de Calidad

- Efficiency: Conjunto de atributos relacionados con la relación entre el nivel de “performance” y los recursos que requiere. Estos atributos consideran:
  - Time behavior
  - Resource behavior

Atributos de Calidad

- Maintainability: Conjunto de atributos relacionados con el esfuerzo requerido para realizar modificaciones. Esto incluye:
  - Analyzability
  - Changeability
  - Stability
  - Testability

Atributos de Calidad

- Portability: Conjunto de atributos relacionados con la capacidad del software para ser transferido de un ambiente a otro. Esto incluye:
  - Adaptability
  - Installability
  - Conformance
  - Replaceability

Calidad en el Software

- Adicional a los atributos “de calidad” que establece el framework ISO9126, es necesario agregar otras definiciones MUY importantes:
  - Confiabilidad = Densidad de Defectos entregados
  - Defecto = se refiere a algún “problema” con el software, ya sea en su comportamiento externo o a sus características internas. (el pópulo lo denomina BUG - a mismatch between implementation and specification)

Calidad en el Software

- Según el estándar IEEE 610.12, un defecto puede ser:
  - Failure = The inability of a system or component to perform its required functions within specified performance requirements.
  - Fault = An incorrect step, process, or data definition in a computer program.
  - Error = A human action that produces an incorrect result.

Revisiones de Software

- ¿qué es una revisión de software?
  - Según IEEE (en su estándar 1028):

    - A process or meeting during which a software product is presented to project personnel, managers, users, customers, user representatives, or other interested parties for comment or approval.

Ventajas de las revisiones de Software

- No requieren de código ejecutable, por lo que puede ser realizada desde el inicio
  - Por lo tanto, es menos costosa
- Se encuentran varios defectos a la vez
- Encuentra hasta un 85% de los defectos
- Se localiza la posición exacta del defecto
- Refuerza el uso de estándares (como IEEE)
- Mejora la capacitación

Desventajas de las revisiones de Software

- Requiere del tiempo de los expertos
- No es útil para verificar las características no-funcionales
- Validan cumplimiento de lo que se especificó, en vez de lo que realmente desea el cliente
- Difícil de implementar (es vista como “improductiva” por los ingenieros) ya que puede consumir “tiempo valioso” de desarrollo.
- Aunque muchas veces son obvias, resulta “difícil” vender sus bondades.

Revisiones

¿Las revisiones son caras?

- Supongan un software al que
  - No se le realizan revisiones durante su desarrollo
  - Se le encuentran 200 defectos en las pruebas internas
- Supongan además que
  - Arreglar un requerimiento defectuoso en la fase de análisis toma 15 minutos
  - Realizar una revisión del documento de requerimientos toma 80 HH (5 personas, 2 días)
- ¿Cuántas HH se hubieran ahorrado si se hubiera revisado el documento de requerimientos?

- De acuerdo a estadísticas mundiales
  - 40% de los defectos ocurren en los requerimientos
  - Arreglar un requerimiento defectuoso durante las pruebas internas cuesta 40 veces más que arreglarlo en el análisis
  - Una revisión detecta en promedio el 70% de los defectos

¿Las revisiones son caras ? Solución

- Proyecto sin revisiones:
  - Arreglar un defecto en análisis = 0.25 HH
    - Entonces arreglarlo en pruebas = 10 HH (0.25*40)
  - Cantidad de defectos en requerimientos = 80 (200 * 40%)
  - Arreglarlos en pruebas = 800 HH (80 defectos * 10 HH)
- Proyecto con revisión del documento de requerimientos:
  - Defectos encontrados por revisión = 56 (80 * 70%)
    - Defectos que pasan a pruebas = 24 (80 – 56)
  - Arreglar defectos en análisis = 14 HH (56 defectos * 0.25 HH)
  - Arreglar defectos en pruebas = 240 HH (24 defectos * 10 HH)
  - Revisar documento de requerimientos = 80 HH
  - Total de tiempo invertido = 334 HH (14 + 240 + 80)
- Ahorro = 466 HH (58%)

Revisiones de software

- Informales
  - No hay proceso definido (son ad hoc)
  - No existen roles
  - Usualmente no planeadas
- Formales
  - Objetivos definidos
  - Proceso documentado
  - Roles definidos y personas entrenados en ellos
  - Check-lists, reglas, estándares y métodos para encontrar defectos
  - Reporte del resultado
  - Recolección de datos para el control del proceso

Tipos de revisiones

- El proceso de revisión más formal es el denominado INSPECCIÓN
- Los más informales son: las pruebas de escritorio y los procesos construidos “ad hoc”.
- Y, entre lo formal y lo informal están los Walkthrough y PairProgramming.

Tipos de revisiones de SW

Artefactos que se pueden revisar:

- Entre los diferentes “productos” que pueden pasar por un proceso de revisión están:
  - Contrato, Presupuesto, Programación de actividades (schedule)
  - El Software Project Management Plan, el Software Quality Assurance Plan
  - El SRS y el SDD
  - El modelo lógico: ERD, Class Model, Object Model, DFD,…
  - Los diagramas de Casos de Uso, de Interacción y de Estados..
  - Pseudocódigo
  - Los manuales de usuario, la documentación operativa,..
  - Test Plan, Acceptance Test Plan, System Test Plan, Integration Test Plan,…
  - Etc…

Pruebas de Software

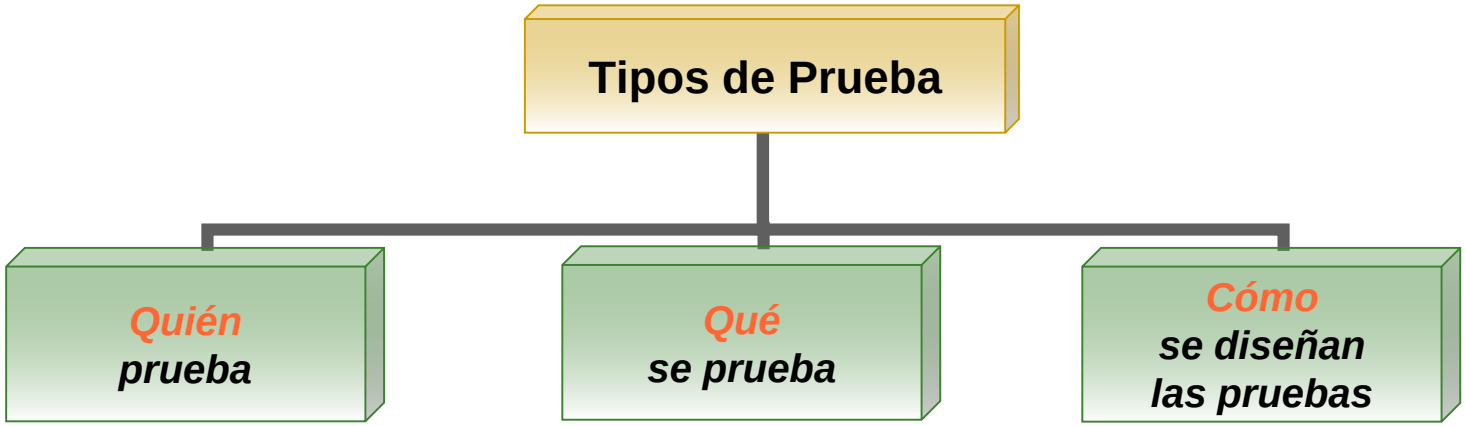
- Prueba…tipicamente se relaciona con:
  - Ejecución de programa(s)
  - Es para descubrir defectos
  - Se hace antes de la entrega
- No asegura ausencia de defectos

Pruebas de Software

- Elementos a considerar en las pruebas:
  - **Planificación**
  - **Seguimiento**
  - **Progresivas**
  - **Equipo independiente**
  - **No exhaustivas**
  - **Problema del oráculo**
  - **…**

Tipos de pruebas

- Las pruebas dependen de:



Dependiendo de quién prueba

- **Internas**
  - Por ingenieros que desarrollan SW
- **Externas**
  - Por el cliente
    (a veces junto con desarrolladores)

Pruebas externas (con el cliente)

- **Pruebas alfa**
  - Instalaciones del desarrollador
  - Grupo muy reducido de clientes
- **Pruebas beta**
  - Instalaciones del cliente, ambiente de “laboratorio”
  - Grupo más amplio de clientes
- **Pruebas piloto**
  - Instalaciones del cliente, ambiente de producción
  - Conjunto reducido de departamentos del cliente

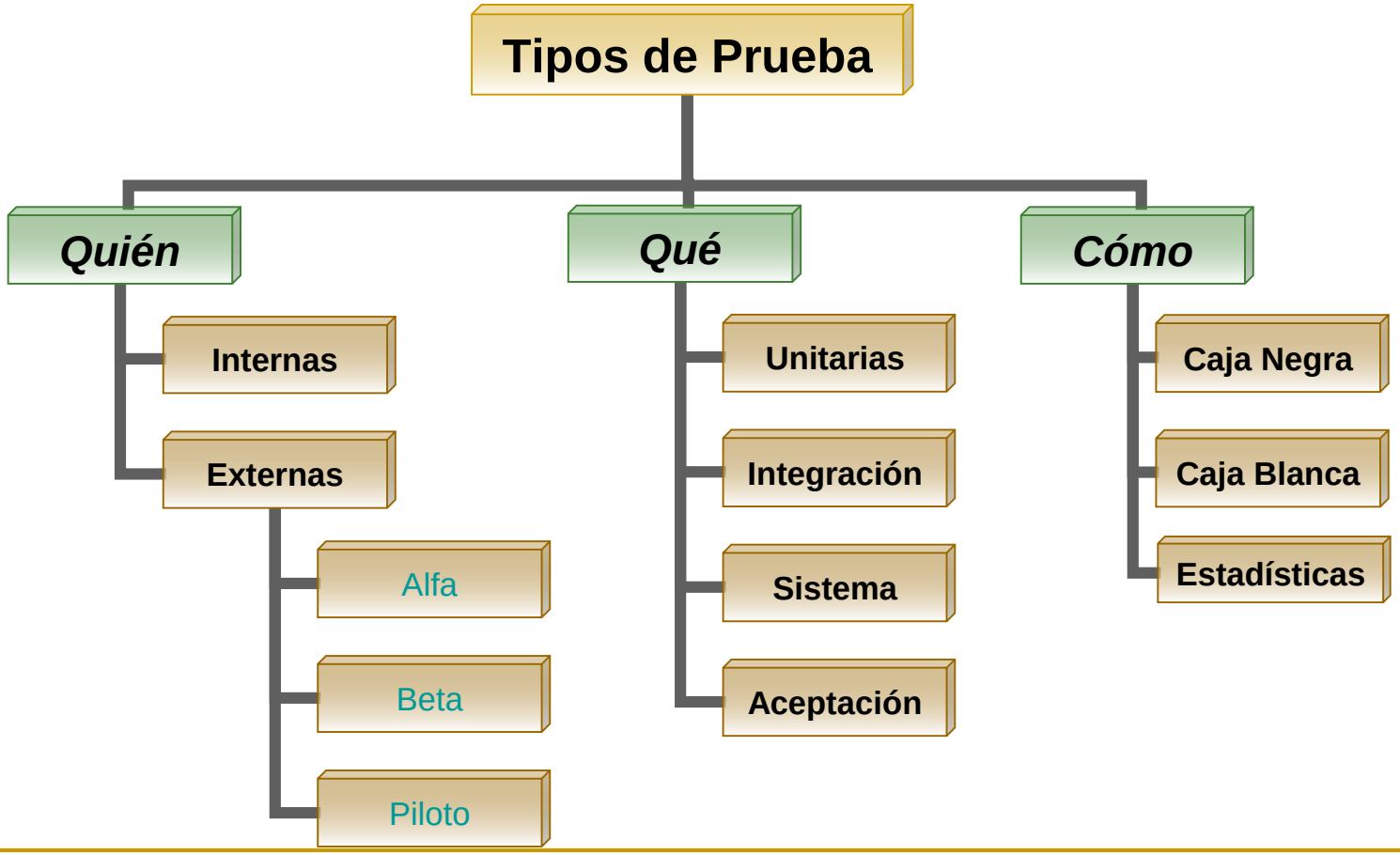
Dependiendo de qué se prueba

- **Comunmente se tienen:**
  - **Pruebas Unitarias**
    - Cada programa individualmente
  - **Pruebas de Integración**
    - Módulos o subsistemas ya integrados
  - **Pruebas del Sistema**
    - Características técnicas del sistema completo
  - **Pruebas de Validación o Aceptación**
    - Funcionalidad del sistema completo

Dependiendo de cómo se diseñan

- **Pruebas de caja negra (Black-box)**
  - Se toma en cuenta funcionalidad
- **Pruebas de caja blanca (White-box ó Glass-box)**
  - Se toma en cuenta cómo está desarrollado (estructurales)
- **Estadísticas**
  - Se diseñan para lo más probable

Clasificación de pruebas



¿quién se encarga de las Pruebas?

- **Test manager**
  - *manage and control a software test project*
  - *supervise test engineers*
  - *define and specify a test plan*
- **Software Test Engineers and Testers**
  - *define test cases, write test specifications, run tests*
- **Independent Test Group**
- **Development Engineers**
  - *Only perform unit tests and integration tests*
- **Quality Assurance Group and Engineers**
  - *Perform system testing*
  - *Define software testing standards and quality control process*

Plan y Diseño de Pruebas

- **Plan de pruebas**
  - **Qué** se va a probar del sistema
    - Incluye tipos de pruebas a realizar
  - **Dónde** se va a probar
  - **Cuándo** se va a probar
  - **Quién** va a probar
  - **Riesgos** y **contingencias**
- **Diseño de pruebas**
  - Detalle de **cómo** se va a realizar CADA prueba,
    - Conjunto de casos de prueba (**Test Cases**) =muchos se generan a la par de la definición de los UseCases

Estrategias para el diseño de Pruebas

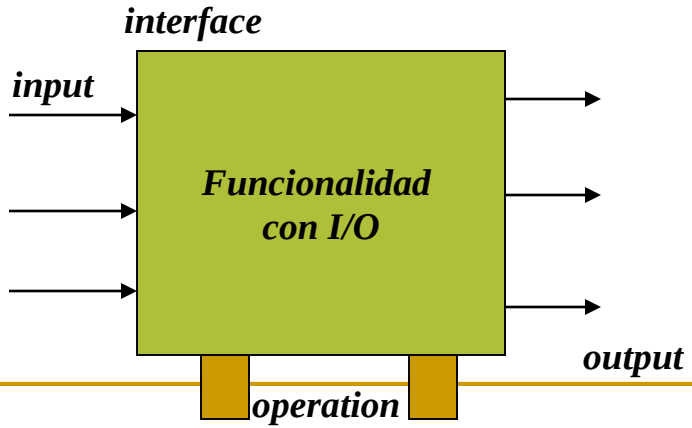
- Dos enfoques son los mas conocidos
  - Black Box Testing
    - Funcionalidad
  - White Box Testing
    - Estructura del código

BlackBox Testing

- También denominadas "**Pruebas de Comportamiento**" ó Funcionales ó *Data-driven* ó *Interface-driven*
- Se basan en ejecutar el programa con un conjunto de datos, tomando como base la especificación de funcionalidad esperada. No le interesa cómo está construido el programa.

BlackBox Testing

- Prueba requerimientos funcionales del software
  - Requisitos funcionales → casos de pruebas
- Intenta corregir:
  - Funcionalidad incorrecta o faltante
  - Errores de interfase
  - Errores en las Bases de Datos
  - Errores en el comportamiento



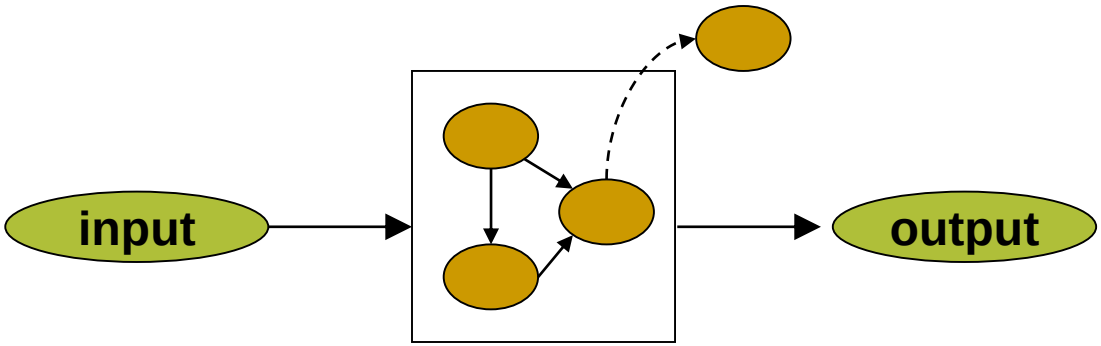
BlackBox Testing

- Con este tipo de pruebas, al Tester sólo le interesa determinar bajo qué circunstancias el software NO se comporta como está establecido en las especificaciones.
- Encontrar un error pudiera implicar una búsqueda exhaustiva.

BlackBox Testing

- Métodos de Caja Negra más comunes:
  - ***Equivalence partitioning (Particiones Equivalentes)***
  - ***Boundary-value analysis (Valores límite)***
  - ***Cause-effect graphing (Basados en grafos)***
  - ***Error guessing***
  - ***State space exploration***
  - ***…y muchos otros…***

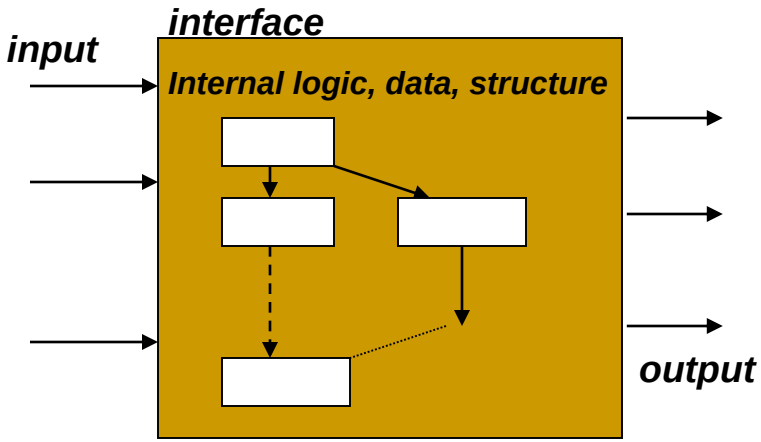
WhiteBox Testing



- También denominadas ***Estructurales*** ó *Code-driven* , *Glass-box*
- Se basan en consierar la “ejecución” de las posibles “rutas” en un fragmento de código.

WhiteBox Testing

- Prueba la estructura interna del software
- Intenta:
  - Probar “en ejecución” TODOS los estatutos (no necesariamente todas las rutas/combinaciones porque sería prácticamente “infinito”)
  - Encontrar errores de codificación



WhiteBox Testing

- Con este tipo de pruebas, en teoría, cada estatuto es probado al menos en una ocasión
- Desgraciadamente este tipo de pruebas no asegura “la funcionalidad”.

WhiteBox Testing

- Métodos de Caja Blanca más comunes:
  - ***Control Flow analysis (Coverage)***
    - ***Camino Básico (basic path)***
    - ***Estructuras de control***
      - ***Ciclos, decisiones,…***
    - ***Estatutos,***
    - ***…***
    - ***Data flow analysis***
  - ***…y muchos otros…***

BlackBox/WhiteBox Testing

- NO es posible hacer un buen proceso de pruebas usando sólo pruebas de Caja Negra o sólo pruebas de Caja Blanca.
- Las de **Caja Blanca** están más asociadas con la idea de hacer "**Test in the small**", mientras que las de **Caja Negra** son más apropiadas para hacer "**Test in the large**".

BlackBox/WhiteBox Testing

- Cada una de estas técnicas tiene un mayor grado de éxito en los distintos niveles de pruebas:

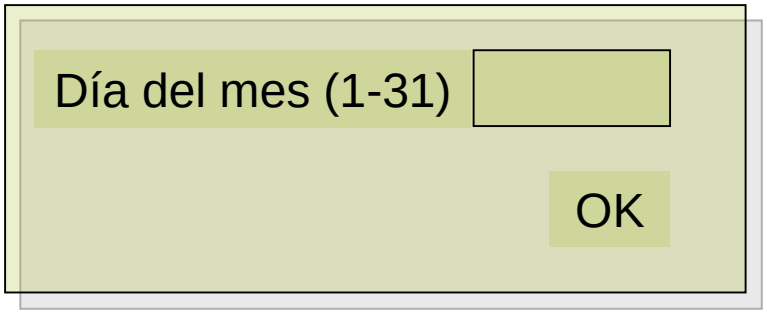
- **WHITEBOX, usado en**:
  - **Unit Testing, Integration Testing, Regression Testing**

- **BLACKBOX, usado en**:
  - **Integration Testing, System Testing, Beta Testing, Regression Testing.**

PRUEBAS FUNCIONALES (BLACK BOX)

Considerar la siguiente aplicación…

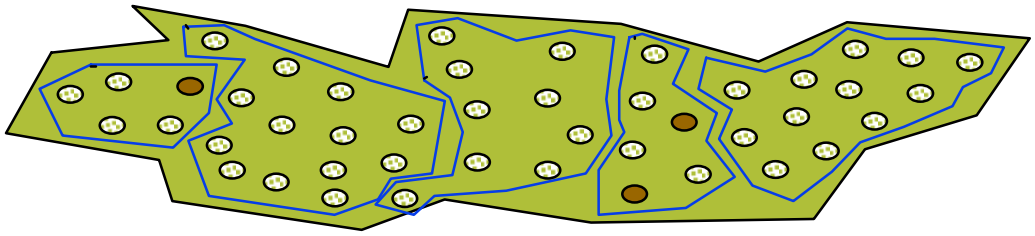
- En una escuela reciben pagos de colegiaturas cada mes.
  - ***Si la persona paga entre el día 1 y 10 no tiene ningún recargo***
  - ***Si paga entre el 11 y 20 tiene un recargo del 2%***
  - ***Si paga el día 21 o después el recargo es del 4%.***
- Se ha realizado un programa que sólo solicita el día del mes en que se paga (en este ejercicio el mes no importa) y despliega el % de recargo a cobrar. Se anexa a continuación la pantalla:



- Diseñen los casos de prueba…¿qué debemos considerar? ¿porqué?

Equivalence Partitioning

- **Divide** (particiona) cada entrada en conjuntos "**equivalentes**" (tipicamente llamados ***Clases de datos***).
- Directriz: diseñar al menos 2 casos de prueba por cada partición
  - Uno válido y otro inválido, o
  - Uno dentro y otro fuera de la partición
- Ejemplos de posibles particiones
  - Si la entrada es alfanumérica
    - Nulo, letras, números, especiales, etc.



Equivalence Partitioning

- Ejemplos de posibles particiones
  - **Numérico**
    - Negativos, cero, positivos, reales, con coma
    - ... más alfanuméricos
  - **Rango**
    - Dentro del rango, fuera del rango
  - **Tabla de rangos**
    - Dentro de cada rango, fuera del menor de los rangos, fuera del mayor de los rangos

Boundary-value analysis

- Cuando se está trabajando con Particiones equivalentes es **MUY** importante tomar en consideración los **valores límites** de cada uno de los rangos.
- **Directrices**:
  - **Rango entre a y b**
    - Diseñar 4 casos de prueba: a-1, a, b y b+1
  - **De M a N datos**
    - Diseñar 4 casos de prueba: M-1, M, N y N+1 datos
  - **De 0 (cero) a N datos**
    - Diseñar 4 casos de prueba: 0, 1, N y N+1 datos

Respuesta al ejercicio…(paso 1 de 3)

- Paso 1: **Analizar posibles entradas de datos**
  - ***Clases de datos (partición equivalente)***
    - Letras
    - Nulo
    - Negativo
    - Positivo
    - Cero
    - Flotante
    - Con coma
    - Un valor entre 1-10
    - Un valor entre 11-20
    - Un valor entre 21-31
  - ***Valores límite***
    - Rango 1-10 → 0, 1, 10,11
    - Rango 11-20 → 10, 11, 20, 21
    - Rango 21-31 → 20, 21, 31, 32

Respuesta al ejercicio…(paso 2 de 3)

- **Paso 2: Eliminar repetidos**
  - Clases de datos (partición equivalente)
    - Letras
    - Nulo
    - Negativo
    - **Positivo**
    - **Cero**
    - Flotante
    - Con coma
    - **Un valor entre 1-10**
    - **Un valor entre 11-20**
    - **Un valor entre 21-31**
  - Valores límite
    - Rango 1-10 → 0, 1, **10**, **11**
    - Rango 11-20 → 10, 11, **20**, **21**
    - Rango 21-31 → 20, 21, 31, 32

Respuesta al ejercicio…(paso 3 de 3)

- **Paso 3: diseñar casos de prueba con los datos resultantes**

ID	Entrada	Salida esperada
1	“abc”	Error
2	Nulo	Error
3	-4	Error
4	3.48	Error
5	1,039	Error
6	0	Error
7	1	0%
8	10	0%
9	11	2%
10	20	2%
11	21	4%
12	31	4%
13	32	Error

¿qué consideraciones se tendrían qué hacer si la interfaz tuviera que capturar también el mes y validar que todo fuera congruente?

Otras estrategias de Caja Negra

- Las estrategias **Model-Based Testing** son otro tipo de pruebas de caja negra.
- Tipicamente trabajan sobre "**modelos**" que representan el **comportamiento** del software que se va a construir.
- Con este tipo de pruebas, se intenta atacar las fallas en las primeras etapas del Ciclo de Vida (durante las etapas de modelación).
- En este tipo de pruebas, dado que el software aún no existe, se requieren de "**test-oracles**" para producir (inventar) lo que serían "salidas esperadas".

Model Based Testing

- Se utilizan **herramientas "formales" de modelación** de comportamiento como:
  - **Gramáticas**
  - **Máquinas de Estados Finitos**
  - **Tablas de Decisión**
  - **Cadenas de Markov**
  - **Redes de Petri**
  - Entre otros…

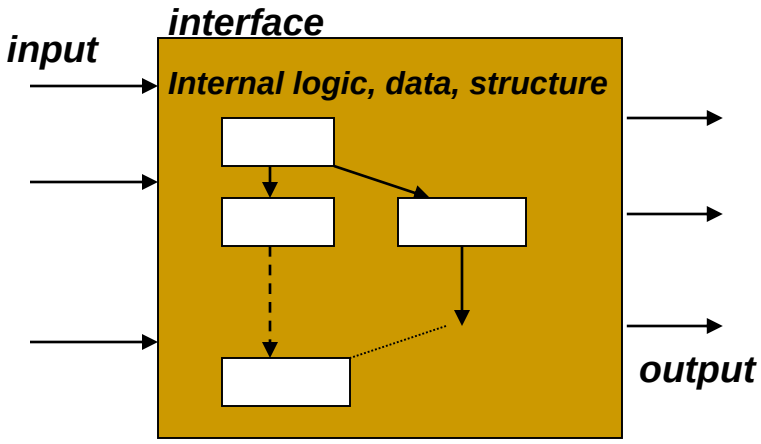
Pruebas de Software

White Box Testing

Pruebas de ESTRUCTURA

WhiteBox Testing

- Prueba la **estructura** interna del software
- Intenta:
  - Probar "**en ejecución**" TODOS los estatutos (no necesariamente todas las rutas/combinaciones porque sería prácticamente "infinito")
  - Encontrar ***errores de codificación***



WhiteBox Testing

- Las pruebas de caja blanca se derivan de la implementación (**código**).
- La cantidad de **TestCases** depende de la estructura propia del código.
- El objetivo es "**cubrir**" (ejercitar) ciertos aspectos de la codificación, a través de:
  - ***Path coverage***
  - ***Statement coverage***
  - ***Branch coverage***
- También hay pruebas basadas en Datos (no sólo en estatutos).

Path Coverage

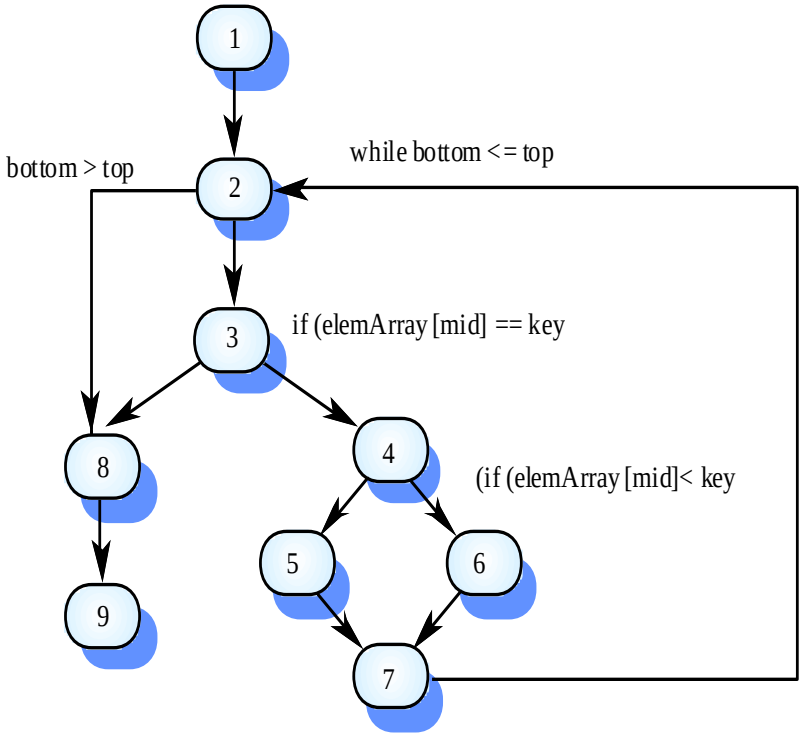
- El objetivo de estas pruebas es asegurarnos que el conjunto de TestCases contemple el "**pasar**" por cada uno de los **caminos** (***paths***) definidos por el código, al menos una vez.
- En estas pruebas el código es representado por un **Grafo**.
- La cantidad de rutas dentro del grafo puede ser enorme. Por esta razón se utilizan sólo los **Caminos Básicos** (Basic Path).
- La cantidad de caminos básicos **(#CB**) determina la **cantidad mínima** de **TestCases** que se requieren.
- ***Inconvenientes***:
  - La cantidad de pruebas necesarias puede ser enorme.
  - Hacer un cambio en el código puede duplicar la cantidad de pruebas requeridas.

Path Coverage:ejemplo

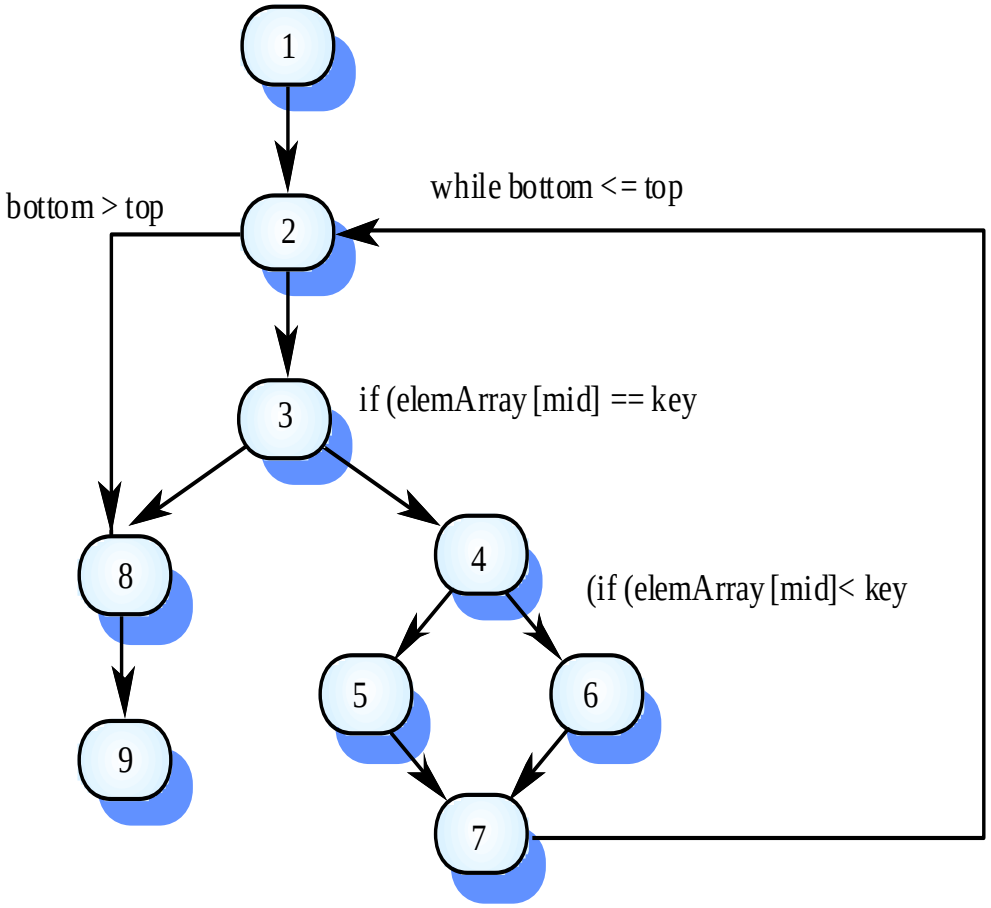
```
class BinSearch {

// This is an encapsulation of a binary search function that takes an array of
// ordered objects and a key and returns an object with 2 attributes namely
// index - the value of the array index
// found - a boolean indicating whether or not the key is in the array
// An object is returned because it is not possible in Java to pass basic types by
// reference to a function and so return two values
// the key is -1 if the element is not found

        public static void search ( int key, int [] elemArray, Result r )
        {
                int bottom = 0 ;
                int top = elemArray.length - 1 ;
                int mid ;
                r.found = false ; r.index = -1 ;
                while ( bottom <= top )
                {
                        mid = (top + bottom) / 2 ;
                        if (elemArray [mid] == key)
                        {
                                r.index = mid ;
                                r.found = true ;
                                return ;
                        } // if part
                        else
                        {
                                if (elemArray [mid] < key)
                                        bottom = mid + 1 ;
                                else
                                        top = mid - 1 ;
                        }
                } //while loop
        } // search
} //BinSearch
```



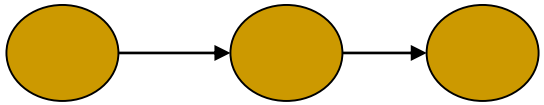
Path Coverage - Ejemplo



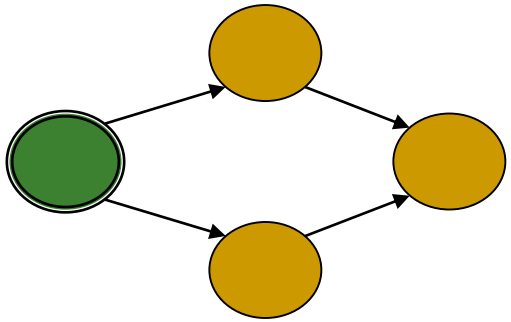
- Caminos o RUTAS INDEPENDIENTES.
  - **1, 2, 3, 8, 9**
  - **1, 2, 3, 4, 6, 7, 2**
  - **1, 2, 3, 4, 5, 7, 2**
  - **1, 2, 3, 4, 6, 7, 2, 8, 9**
- Los casos de uso deberían considerar estas rutas.

Path Coverage - Notación usada

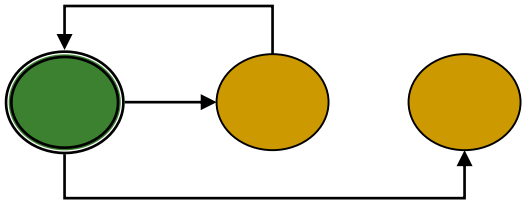
Secuencia



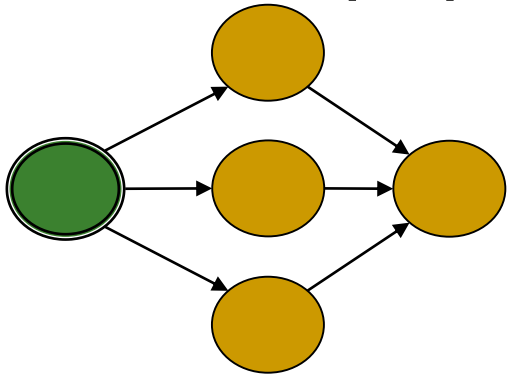
Decisión Sencilla (if)



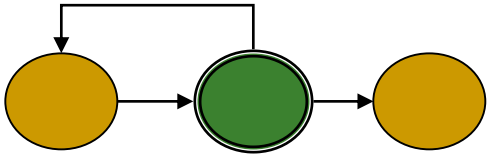
Ciclo (while, for)



Decisión Múltiple (switch)



Ciclo (do)



= Nodo Predicado

Path Coverage -- Basic Paths

- Pensar en cubrir todas las posibles combinaciones (generadas por ciclos y condiciones) es prácticamente imposible.
- Por esta razón, se utiliza la estrategia de **Caminos Básicos** (***Basic Path Testing***).
- Es muy utilizada en Pruebas de Unidad (**Unit Testing**).
- La determinación de las rutas básicas se basa en la ***Complejidad Ciclométrica del Grafo***.

Path Coverage -- Basis Paths

- **Complejidad Ciclométrica** (*Cyclometric Complexity*)
  - Es una **métrica** de software.
  - Es una herramienta cuantitativa de medición para la complejidad de un programa.
  - Al usar esta métrica dentro del concepto de ***Caminos Básicos***, permite determinar la ***cantidad mínima*** de rutas independientes que deben contemplarse en los **TestCases**.

Complejidad Ciclomática

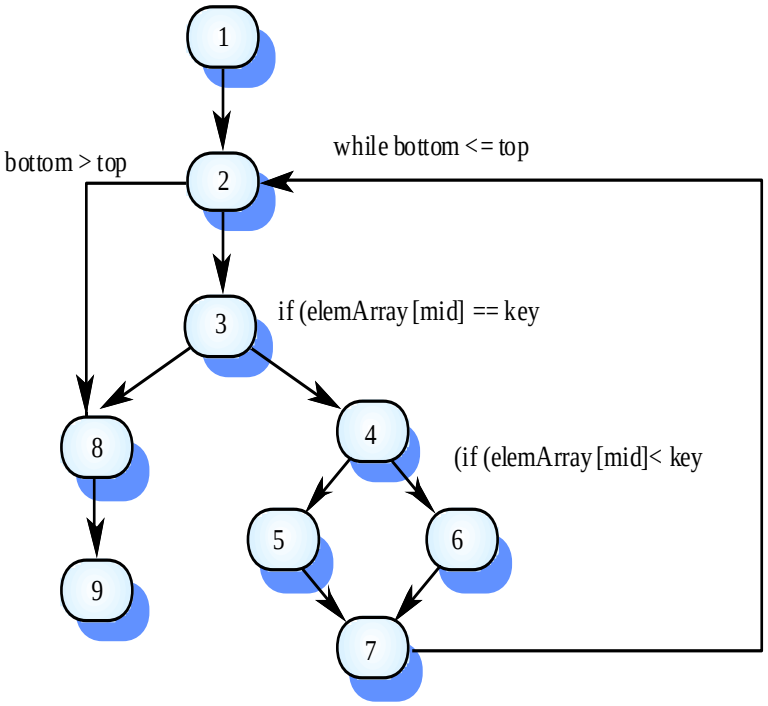
- **Complejidad Ciclomática** (CC)
  - Complejidad lógica de una rutina
  - Cantidad de caminos independientes (CBs)

  - **CC** = #Arcos – #Nodos + 2

  ó

  - **CC** = #NodosPredicado + 1
    - *Se calcula asi sólo si código es:*
      - *Estructurado y*
      - *Sin decisiones múltiples*

Ejemplo de Complejidad Ciclomática



CC para Búsqueda Binaria

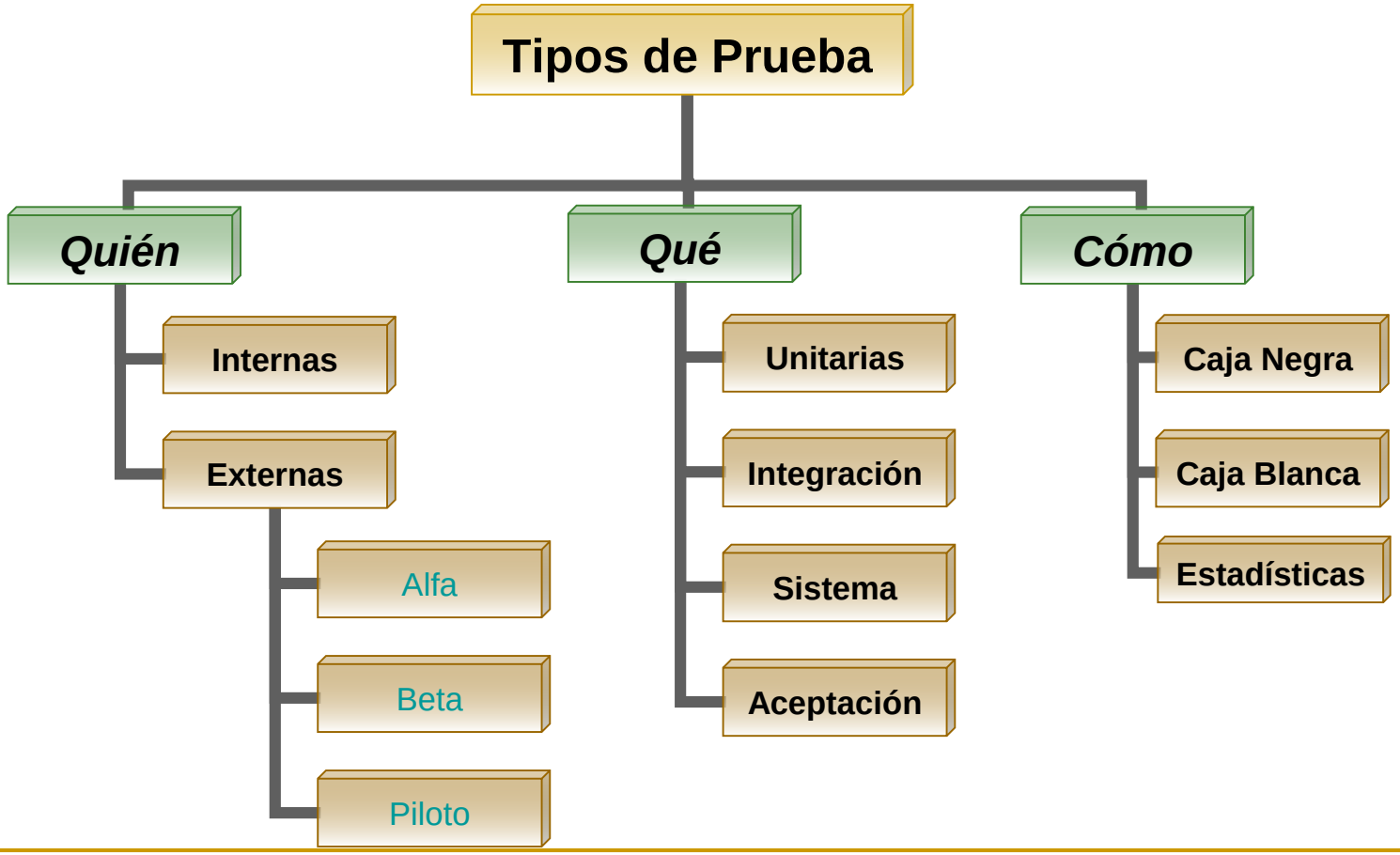
V(G) = N# Arcos – N# Nodos+ 2
= 11 – 9 + 2
= 4

V(G) = N# NodosPredicado + 1
= 3 + 1 *(nodes 2, 3 & 4)*
= 4

Complejidad Ciclomática

- Puntos importantes:
  - Se asegura que cada estatuto **se ejecutará una vez**, unque, con eso, no se prueben todas “las posibles combinaciones”.
  - Es recomendable usar “**Nodos condensados**” (en los que se agrupan todos los estatutos que no tengan decisiones involucradas).
  - Es muy importante considerar que todas las expresiones condicionales son **SIMPLES** (si hay expresiones complejas, se deben fragmentar).

Clasificación de pruebas



UNIT Testing

- Se basa en "***Test in the small***".
- Trabaja con **UNIDADES** de código, tipicamente se piensa en:
  - ***Funciones***
  - ***Clases (incluso por métodos)***
  - ***Paquetes***
- Una buena parte de las ***pruebas de unidad*** son llevadas a cabo por los mismos **desarrolladores**.
- Si no se realiza un buen proceso de **UNIT Testing**, los procesos de Pruebas de Integración y pruebas del Sistema (***Integration Test y System Test***) se volverán sumamente complicados.

UNIT Testing

- El proceso de **UNIT Testing** debe estar fuertemente ligado a **Requerimientos**.
- Ya que uno de los elementos bases para realizar estas pruebas es el “contraste” contra los requerimientos (el **CONTRATO** establecido para esa unidad de código)
- Las pruebas de unidad se pueden realizar a distintos niveles de **granularidad**.
- Las pruebas de unidad pueden utilizar técnicas de **Caja Negra** (***Particiones, Valores Límite,…***), asi como técnicas de **Caja Blanca** ***(Estatutos, Rutas, …***)

UNIT Testing

- Hoy en día, la mayor parte de las pruebas de unidad se hace en forma "**automática**" usando herramientas (y en muchos casos librerías de los mismos lenguajes) que permiten programar las pruebas "indispensables".

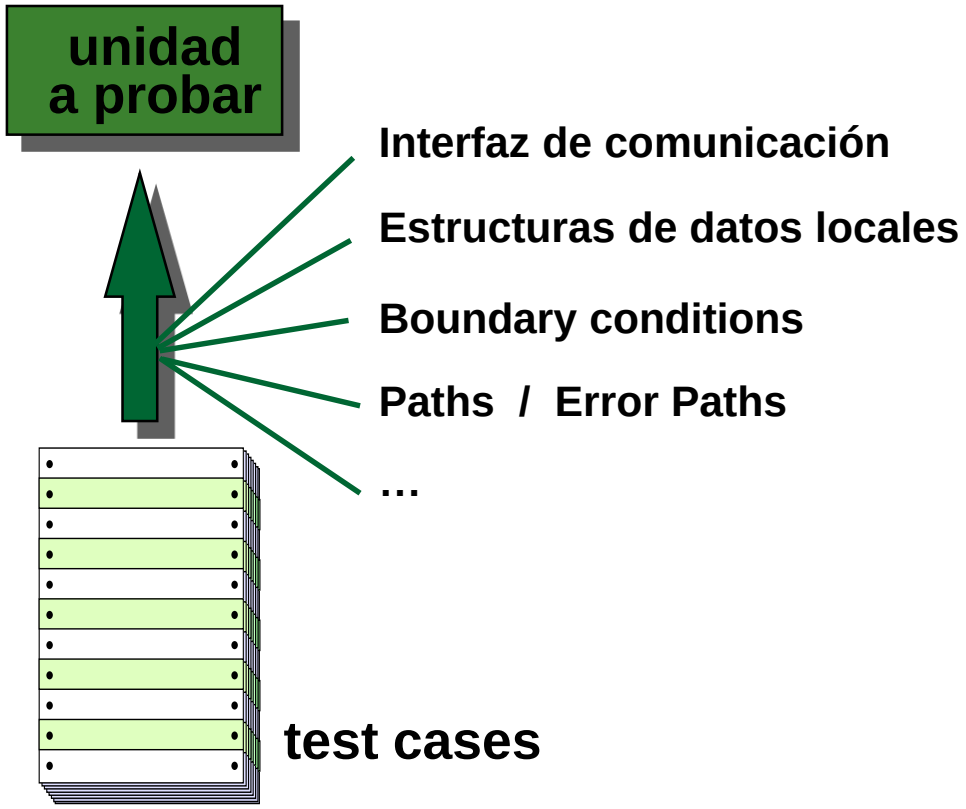
- Como ejemplo: ***JUnit*** (para Java) es un **framework** de uso "industrial" que facilita el proceso de pruebas de unidad.

UNIT Testing

- El proceso básico de "**automatización**" contempla **diseñar código** para utilizar el *código a probar* (UNIT)
- Y, contrastar contra "**valores conocidos**" válidos e inválidos.
- Esto se hace considerando el **Contrato** (**especificación**) establecido para la unidad a probar.

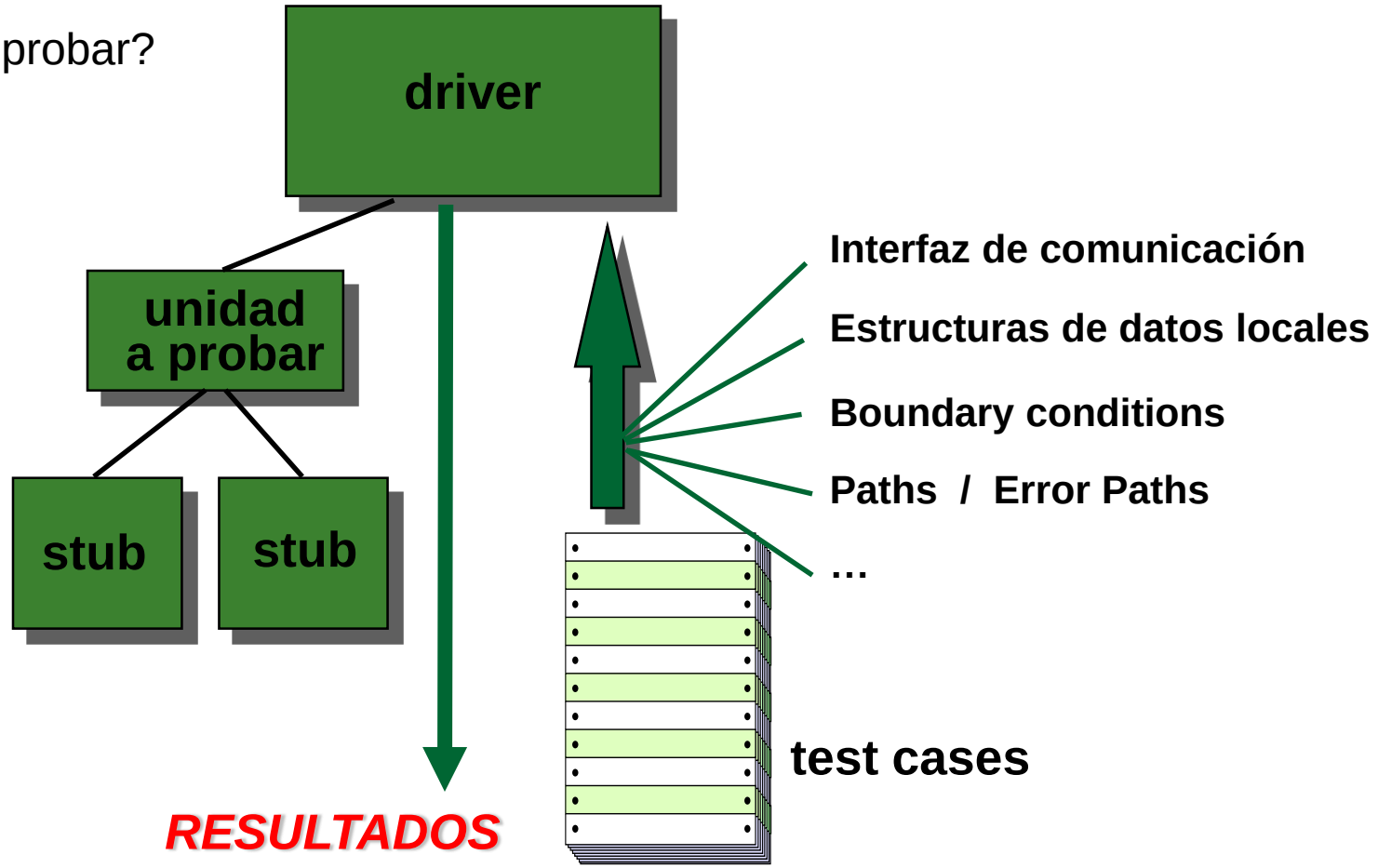
UNIT Testing

¿qué probar?



UNIT Testing

¿cómo probar?

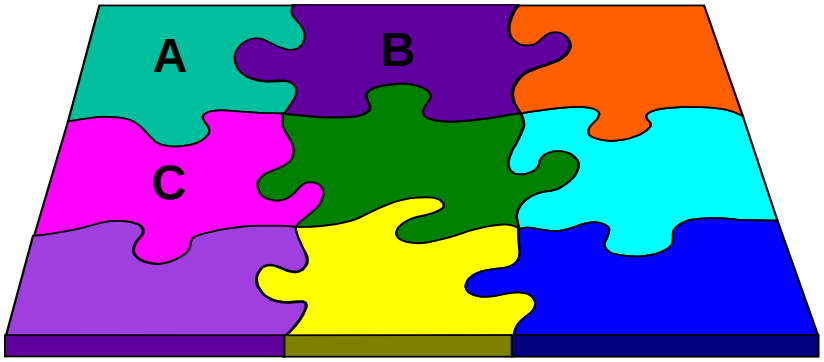


UNIT Testing

- **DRIVER** (***o TEST DRIVER o TEST HARNESS***):
  - Código que sirve para probar el "**código**"
  - Contiene los casos de prueba para el "**código**".
- **STUB**
  - Módulos "falsos" que se utilizan para interactuar con el "código" a probar.
  - Son módulos que se crean para proveer datos o recibir datos del módulo a probar.
  - "simulan" el comportamiento esperado del módulo para responder al módulo que se está probando.

INTEGRATION Testing

- Una vez que se tienen un conjunto de “**Unidades**” ya probadas, es necesario verificar que éstas se “integren” correctamente entre sí.
- Porque no es lo mismo funcionar de manera independiente, que en forma coordinada, atendiendo solicitudes de otros módulos.



INTEGRATION Testing

- Se pueden seguir distintas estrategias para realizar las pruebas de integración:
  - ***DBIT: Decomposition Based Integration Testing***
  - ***CGBIT: Call Graph Based Integration Testing***
  - ***PBIT: Path Based Integration Testing***

- Las más comunmente usadas son las Pruebas Basadas en Descomposición (**DBIT**).

DB - INTEGRATION Testing

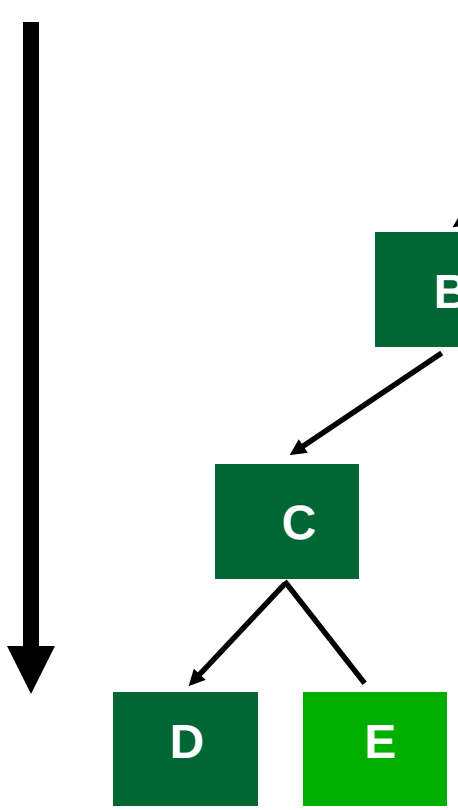
- Las pruebas de integración por descomposición utilizan un proceso **INCREMENTAL**, lo que lo hace más “fácil” de implementar, o al menos, menos caótico.

- La descomposición puede llevarse a cabo:
  - **TOP-DOWN**
  - **BOTTOM-UP**
  - **SANDWICH**

DB - INTEGRATION Testing

- Estrategia **Top-Down**:
  - Inicia con el Módulo Princial y gradualmente va agregando módulos.
    - El Sistema se prueba cada vez que un nuevo módulo se agrega.
    - Puede usar estrategia Breath-first ó Depth-first.
- Estrategia **Bottom-Up**:
  - Inicia con módulos atómicos (hojas en el árbol de llamadas)
    - Se apoya en Test-drivers para probar “clusters” (grupos) de módulos.
    - Gradualmente va agregando los módulos a los que se llama y se va moviendo hacia arriba.

DBIT- Top-Down Testing



El módulo principal se prueba con el apoyo de stubs

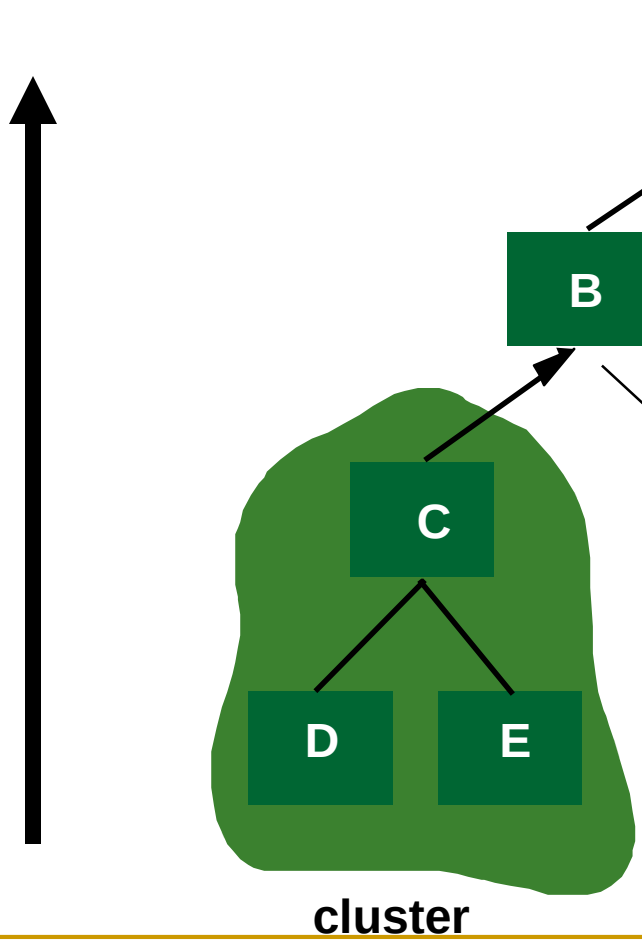
Conforme se van agregando módulos los stubs se van eliminado uno a uno

Cada vez que se agregan módulos es necesario volver a "correr" un subconjunto de las pruebas.

DBIT- Top-Down Testing

- **Ventajas**:
  - Valida funciones de decisión y de control mucho más “temprano”
  - Resulta MUY útil cuando se tiene un problema “serio” de control entre los módulos.
- **Desventajas**:
  - La construcción de los STUBS necesarios (en los primeros niveles) requiere mucho esfuerzo (de hecho es lo que comunmente “más cuesta” en este proceso).
  - No necesariamente será factible hacer una integración completa (por cuestiones de lógica de la aplicación).

DBIT- Bottom-Up Testing



Los test-drivers se van sustituyendo -uno a la vez- con estrategia depth-first

Algunos módulos se agrupan en "unidades lógicas" para formar clusters

DBIT- Bottom-UpTesting

- **Ventajas**:
  - Dado que inicia probando los módulos del nivel más bajo, NO requiere el uso de STUBS.
- **Desventajas**:
  - Es imposible tener TODO el sistema antes de terminar la integración.
  - No podrá detectar problemas fuertes de Interfaz/Control hasta muy avanzado el proceso

INTEGRATION Testing

- Es MUY importante probar inicialmente la integración de los **módulos críticos**:
  - Están relacionados a múltiples REQUERIMIENTOS
  - Tienen un papel importante en el control de la aplicación
  - Son de naturaleza compleja
  - Tienen altos niveles de acoplamiento con otros módulos
  - Están directamente ligados al performance de la aplicación
  - Son parte de la RUTA CRÍTICA del proyecto

System Testing

- Se basa en "***Test in the large***".
- Trabaja con el **Sistema** "completo", tipicamente busca desarrollar pruebas tipo:
  - ***Stress testing***
  - ***Load testing***
  - ***Performance testing***
  - ***Networking testing***
  - ***Etc…***
- Su función principal es determinar que la implantación del sistema se haga correctamente.

- Roger S. Pressman (2005). *Ingeniería de Software, un enfoque práctico, 6a. ed., México. McGraw-Hill.*
- Kendall y Kendall (2005). *Análisis y diseño de sistemas, 6a. ed., México, Pearson Educación. 752 p.*
- Larman C. (2003). *UML y patrones: una introducción al análisis y diseño orientado a objetos y al proceso unificado, 2a. Ed., Madrid, Pearson.*
- Robbins, Stephen P. (2001), *Administración, 6a. ed., México, Prentice Hall.*
- Rumbaugh, E. y Jacobson, I. y Booch, G. (2007). *El Lenguaje Unificado de Modelado, Manual de referencia*
- Sommerville, Ian (2005). *Ingeniería del software, 7a. ed., Madrid, Pearson - Adidson Wesley, 687 p.*

- Levine Gutiérrez, Guillermo. *Computación y programación moderna* : perspectiva integral de la informática / Guillermo Levine Gutiérrez ;

QA
76.5
.L48
2001