FICH UNL
Facultad de Ingeniería y Ciencias Hídricas
UNIVERSIDAD NACIONAL DEL LITORAL

Ingeniería de software II

Verificación y Validación

Verificación y Validación

- Verificación y Validación (V&V):
  - Nombre dado a los procesos que permiten asegurar que el programa desarrollado:
    - satisface su especificación.
    - brinda la funcionalidad esperada.
  - Validación:
    - ¿Estamos construyendo el producto correcto?
      - Mostrar que el software cumpla con los requerimientos funcionales y no funcionales establecidos
  - Verificación:
    - ¿Estamos construyendo el producto correctamente?
      - Mostrar que el software cumpla con las expectativas del cliente.

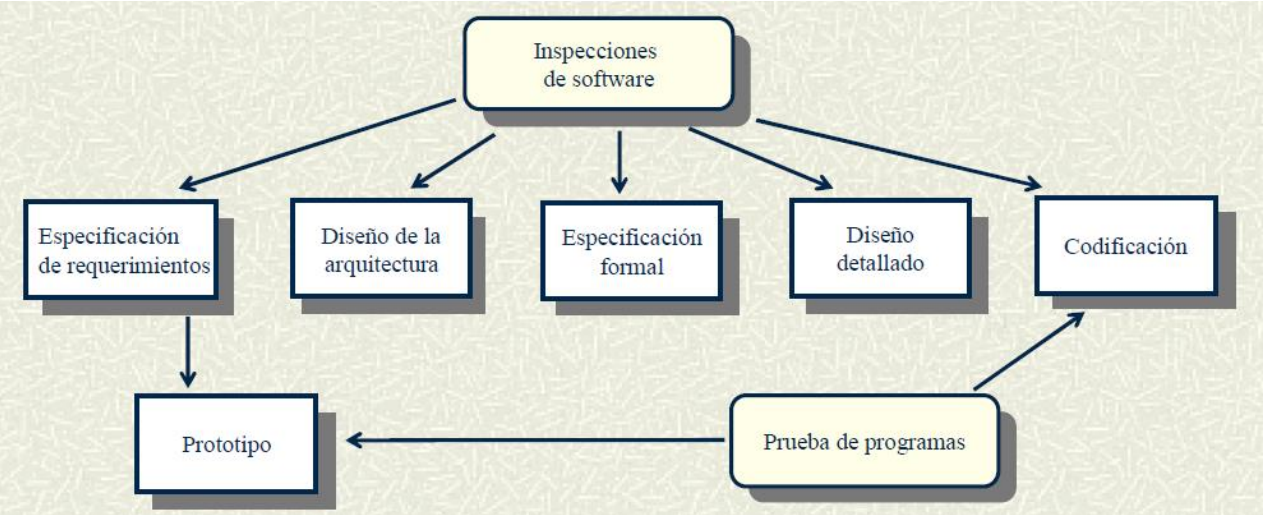
V&V

- Proceso de **comprobación**
  - comienza ni bien estén disponibles los requerimientos.
  - continúa a través de todas las etapas del proceso de desarrollo.
- Proceso para establecer **confianza** en que el sistema de software es adecuado

Software suficientemente eficaz para su uso esperado

V&V

- Aproximaciones complementarias:
  - Inspecciones de software
  - Pruebas de software



Inspecciones de software

- Inspecciones de software:
  - **Técnicas estáticas**
    - no requieren que el sistema se ejecute.
  - Analizan y comprueban:
    - representaciones del sistema:
      - ERS
      - diagramas de diseño
      - código fuente del programa.
  - Pueden aplicarse en todas las etapas del proceso.
  - Para descubrir errores, se utiliza:
    - conocimiento del sistema
    - dominio de aplicación
    - lenguaje de modelado
    - lenguaje de aplicación

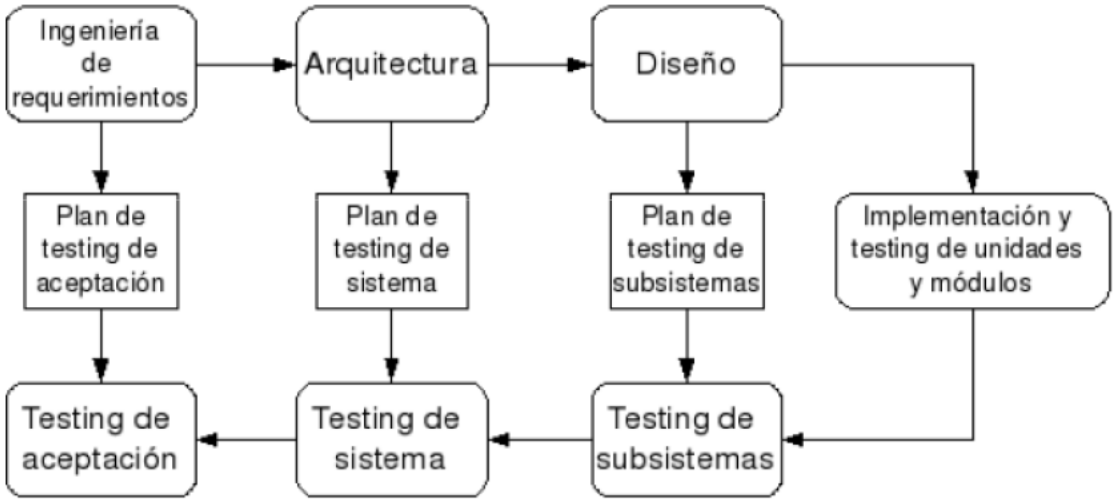
Pruebas de software

- Pruebas de software:
  - **Técnicas dinámicas**
    - Requieren un prototipo ejecutable del sistema.
  - Implican ejecutar una implementación del software con datos de prueba.
  - Se examinan:
    - Salidas
    - Entorno

Planificación de V&V

- Inicia en las primeras etapas del ciclo de desarrollo.
- Actividades de la gestión de V&V:
  - Utiliza estándares y procedimientos.
    - Define checklists para las inspecciones
    - Define el plan de pruebas (PP).

El esfuerzo dedicado a V&V depende del tipo de sistema en desarrollo y de la experiencia organizacional.



Planificación de V&V

- El PP contiene:
  - Descripción del proceso de pruebas.
    - Fases de pruebas a realizar.
  - Requerimientos a probar.
    - Ej. casos de uso.
  - Productos de software a probar.
  - Calendarización de las pruebas.
  - Procedimientos de registro de las pruebas:
    - Registro de defectos, asignación, tiempos estimados de corrección, etc.
  - Requerimientos de hardware y software.
  - Restricciones.
    - Restricciones que afectan al proceso de pruebas
      - Ej personal dedicado.

Inspecciones de software

- Revisiones sistemáticas de documentos generados
  - Objetivo: detectar fallos.
- Ventajas:
  - No requieren que el programa se ejecute.
  - Pueden realizarse antes que se implemente.
  - Son más efectivas y menos costosas que las pruebas:
    - Inspecciones informales: descubren 60% de errores.
    - Inspecciones formales (Cuarto Limpio): descubren 90% de errores.
  - Se pueden inspeccionar atributos de calidad:
    - Ajuste a estándares, portabilidad, mantenibilidad.
  - Localización de:
    - Ineficiencias
    - Algoritmos inadecuados
    - Estilos de programación tendientes a un sistema difícil de mantener y actualizar.
  - Numerosos defectos se detectan en una sola sesión de inspección.
    - Reutilizan el conocimiento del dominio y del lenguaje de programación.

Inspecciones de software

- No reemplazan a las pruebas.
  - Son un complemento a utilizar en etapas previas.
- No pueden validar el comportamiento dinámico del software.
- A veces no es práctico inspeccionar un sistema completo.
  - A nivel “sistema” se utilizan las pruebas.
- Permiten identificar problemas en el mismo proceso de inspección
  - ayudar a diseñar formas más efectivas para probar el sistema.
- Requieren sobrecostos al inicio
  - conducen a ahorro en los costos posterior.
- Las versiones incompletas de un sistema se pueden inspeccionar.

Inspecciones de software

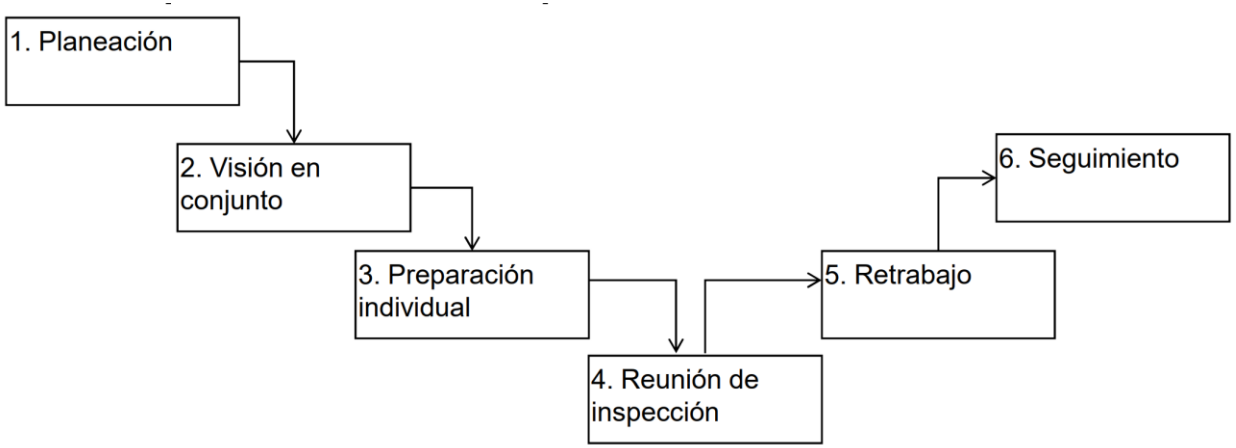
- Proceso
  - No más de 2 horas.
  - Dedicada a detectar defectos, anomalías y discordancias con los estándares.
    - No busca sugerir formas de corrección o cambios.
    - Checklist de errores comunes (revisado previamente por expertos en el tema).
  - Es importante la experiencia organizacional.
  - Proceso público de detecciones de errores.
    - Los errores cometidos se muestran a todo el equipo de programación

Inspecciones de programa

- Revisiones de código:
  - Objetivo: detección de defectos en el programa (código).
  - El equipo revisa cada línea del código fuente del programa.
- Integrantes:
  - Autor
  - Lector
    - Lee en voz alta al equipo el código
  - Probador
    - inspecciona el código desde una perspectiva de prueba
  - Moderador
    - organiza el proceso.
- Precondiciones del proceso:
  - Especificación precisa del código a inspeccionar.
  - Miembros del equipo familiarizados con estándares organizacionales.
  - Versión actualizada y sintácticamente correcta del código disponible.

Inspecciones de programa

- Se utiliza una lista de los errores comunes.
  - Se somete a discusión por el personal con experiencia
  - Se actualiza frecuentemente.
    - Es importante obtener experiencia a nivel organización.
- Límite:
  - Cantidad de código
  - 125 líneas de código por sesión
  - 2 horas.



Inspecciones de programa

- Comprobaciones

Clase de defecto	Comprobación de inspección
Defectos de datos	- Se inicializan todas las variables antes de que se utilicen sus valores? - Tienen nombres todas las constantes? - Límites de arreglos/matrices? - Separadores en cadenas? - Buffers o variables que se desborden?
Defectos de control	-Para cada sentencia condicional, es correcta la condición? - Se garantiza que termina cada bucle? - Están puestas correctamente entre llaves las sentencias compuestas? - En las sentencias case, se tienen en cuenta todos los posibles casos?
Defectos de entrada / salida	-Se utilizan todas las variables de entrada? - Se les asigna un valor a todas las variables de salida? - Pueden provocar errores los datos de entrada no esperados?
Defectos de interfaz	-Concuerdan los tipos de parámetros? - Están en el orden correcto los parámetros? - Las llamadas a funciones y métodos, tienen el número correcto de parámetros?
Defectos de gestión de almacenamiento	- Si se utiliza almacenamiento dinámico, se asigna correctamente el espacio de memoria? - Se limpian todas las variables al salir de la aplicación?
Defectos de manejo de excepciones	- Se tienen en cuenta todas las condiciones de error posibles?

Análisis estático automatizado

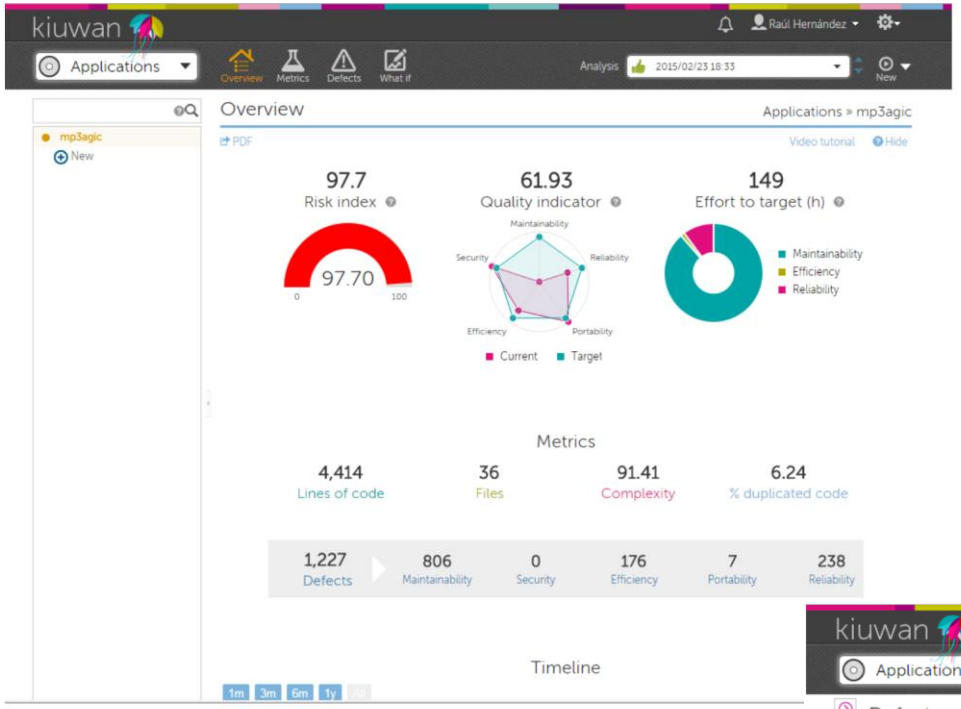
- Herramientas
  - Rastrean el código fuente de un programa
  - Detectan posibles fallas y anomalías.
  - No requieren que el programa se ejecute.
  - Son un complemento de los recursos de detección de errores provistos por el compilador del lenguaje.

Análisis estático automatizado

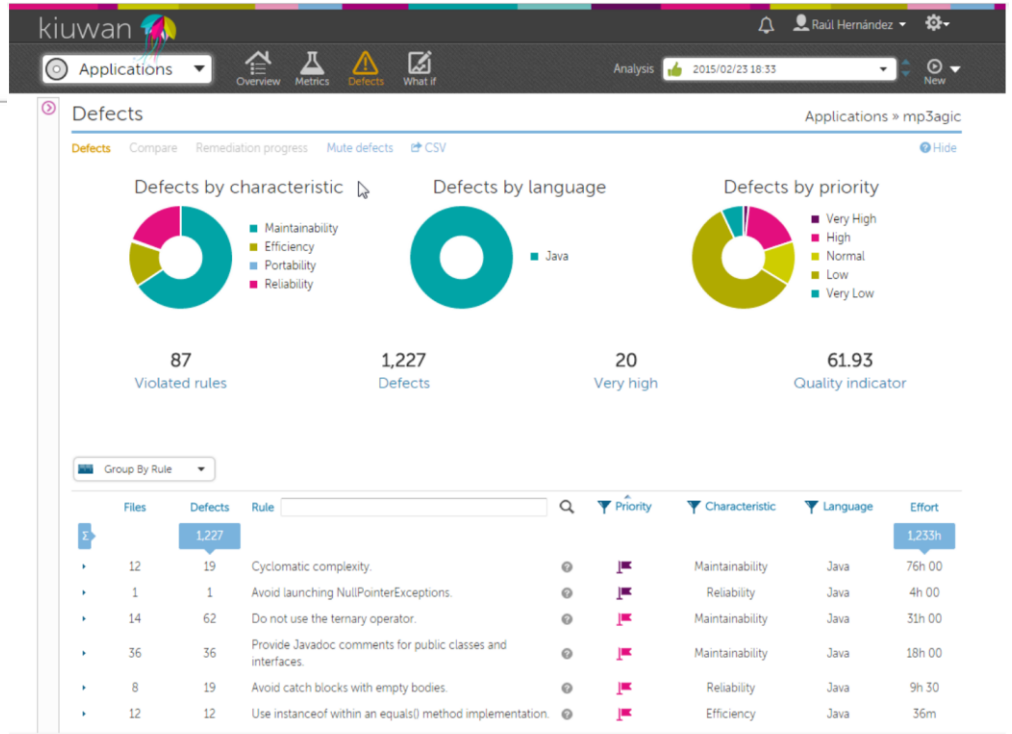
- Comprobaciones que hace el análisis estático:
  - Flujo de control:
    - Identifica y señala los bucles con múltiples puntos de salida y las secciones de código no alcanzable.
  - Utilización de los datos:
    - Señala como se utilizan las variables del programa:
      - Variables sin utilización previa
      - Variables que se declaran dos veces
      - Variables declaradas y nunca utilizadas.
      - Condiciones lógicas con valor invariante

Análisis estático automatizado

- Comprobaciones que hace el análisis estático:
  - Interfaces: Verifica la declaración de las operaciones y su invocación
  - Trayectoria (análisis semántico): Identifica todas las posibles trayectorias del programa y presenta las sentencias ejecutadas en cada trayectoria.
- Ejemplos:
  - Analizador estático: LINT, bajo Linux, para el lenguaje C.
  - Analizador de código Zend para php
  - Analizador semántico Quick Fix
  - Analizador estático Kiuwan
- Debe utilizarse siempre junto a inspecciones directas del código, pues existen errores que no pueden detectar, por ejemplo la inicialización de una variable a un valor incorrecto



Indicadores de primer nivel en Kiuwan



Kiuwan visión de Defectos

Estrategia de prueba

- Estrategia de prueba de software
  - Proporciona una guía que describe los pasos que deben realizarse como parte de la prueba
  - Plan
    - Esfuerzo, tiempo y recursos
    - Diseño de casos de prueba.
    - Recolección y evaluación de resultados.

Probar

- Es el proceso de detectar diferencias (errores) entre el comportamiento esperado y el comportamiento actual del sistema.

(IEEE, Standard for Software Test Documentation, 1983)

- Establecer fehacientemente que un programa hace lo que se supone que tiene que hacer

(Bill Hetzel, 1975)

- El proceso de ejecutar un programa o sistema con el objeto de encontrar errores

(Myers, 1979)

- El proceso de ejecutar un programa utilizando un caso de prueba construido expresamente para asegurar que la conducta exhibida es la esperada.

Prueba

- Objetivo
  - Encontrar errores.
    - Antes que lo haga el usuario final.
  - Asegurar la implementación de todas las funcionalidades.
  - Prevenir problemas futuros
  - Asegurar la satisfacción del cliente
    - Proceso de mejora.

Una medida de la calidad del software.

- Costo por no probar:
  - Tiempo
  - Dinero
  - Intangible: confiabilidad, imagen, etc.

Pruebas

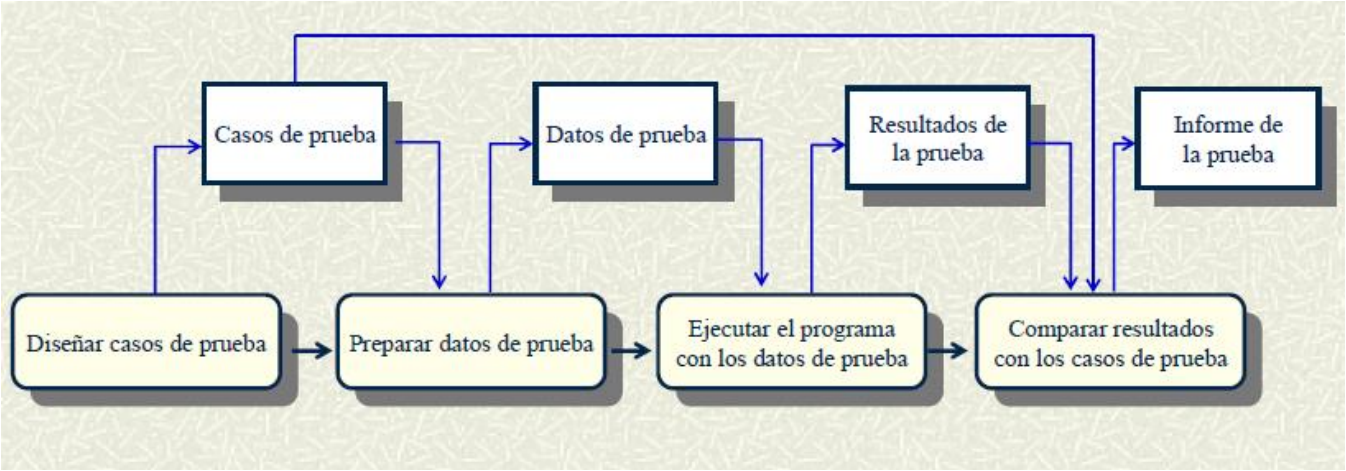
- **Proceso general de pruebas:**
  - **Se realiza en cuatro etapas:**
    - Pruebas de las unidades (pruebas de métodos y clases)
      - se prueban programas individuales como funciones u objetos.
    - Pruebas de integración o de subsistemas:
      - se prueban agrupaciones de clases relacionadas.
    - Prueba de sistema:
      - se prueba el sistema como un todo.
    - Prueba de aceptación:
      - prueba del sistema en el entorno real de trabajo con intervención del usuario final.
- El descubrimiento de un defecto en una etapa requerirá la repetición de las etapas de prueba anteriores.

Pruebas

- Pruebas de defectos:
  - Buscan las inconsistencias entre un programa y su especificación.
  - Buscan los errores en el código.
  - **Demuestran la presencia, y no la ausencia, de defectos**
- Pruebas estadísticas:
  - Buscan demostrar que se satisface la especificación operacional y su fiabilidad.
  - Se diseñan para reflejar la carga de trabajo habitual.
  - Sus resultados se procesan estadísticamente para estimar su fiabilidad
    - Cantidad de caídas del sistema
    - Tiempos de respuesta.

Pruebas

- Objetivo:
  - exponer los defectos latentes en un sistema de software antes de entregar el sistema.
- Se contrapone con las pruebas de aceptación, en las que se pretende demostrar que un sistema cumple con su especificación.
- **Prueba de defecto exitosa: expone un defecto = el sistema se desempeña incorrectamente.**
- Proceso:



Pruebas

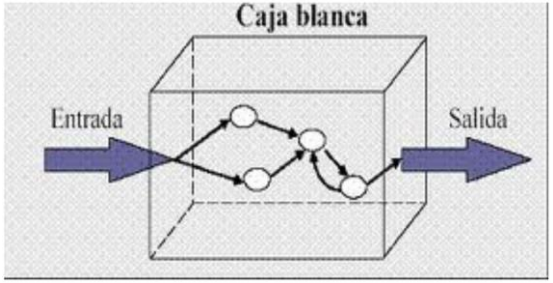
- Las pruebas exhaustivas, donde se prueba cada posible secuencia de ejecuciones del programa, no son prácticas.
- Se debe llevar a cabo pruebas sobre un subconjunto de posibles casos de prueba.
  - Definir políticas organizacionales para pruebas.
  - Tener en cuenta experiencia en la utilización del sistema.
- Pruebas de caja negra o funcionales:
  - Se derivan de la especificación de los requerimientos.
    - El comportamiento del sistema está determinado por sus entradas y las salidas esperadas.
- Pruebas de caja blanca o estructurales:
  - Se conoce la estructura del software a probar.
  - Se utilizan principalmente para probar ciertas porciones de código.

Pruebas caja negra o funcionales

- Pruebas de caja negra:
  - Es una política de selección de casos de pruebas basado en la especificación del componente o programa.
  - Las pruebas se seleccionan en función de la especificación y no de la estructura interna del software.

Pruebas de caja blanca o estructurales

- Pruebas de caja blanca
  - Se seleccionan en función del conocimiento que se tiene de la implementación del módulo.
  - Se suelen aplicar a módulos pequeños.
    - El tester analiza el código y deduce cuántos y qué conjuntos de valores de entrada han de probarse para que al menos se ejecute una vez cada sentencia del código.
  - Se pueden refinar los casos de prueba que se identifican con pruebas de caja negra.

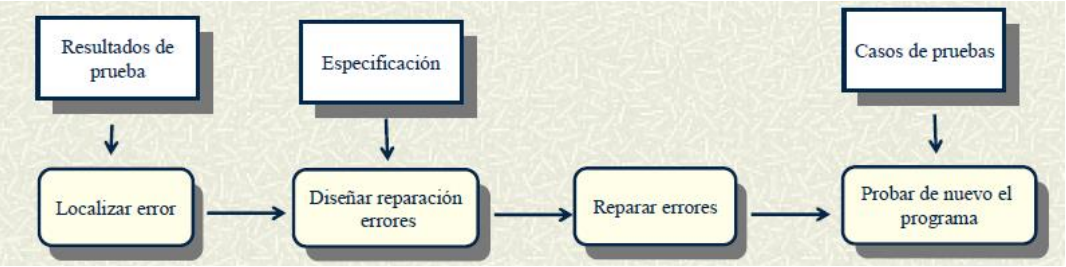


Proceso de depuración

- Depuración:
  - Proceso de eliminación de los errores que se descubren durante las fases de prueba.
- Es un proceso independiente que no tiene porqué estar integrado:
  - La verificación y validación establece la existencia de defectos en el programa.
  - La depuración es el proceso que localiza el origen y corrige estos defectos.

Proceso de depuración

- Proceso de depuración:
  - Proceso que localiza y corrige los errores descubiertos durante la verificación y validación.
- No siempre los errores se detectan cerca del punto en que se generaron.
- Se utilizan herramientas de depuración, que facilitan el proceso.
- Después de reparar el error, hay que volver a probar el sistema (pruebas de regresión).
- La solución del primer fallo puede dar lugar a nuevos fallos.



Proceso de depuración

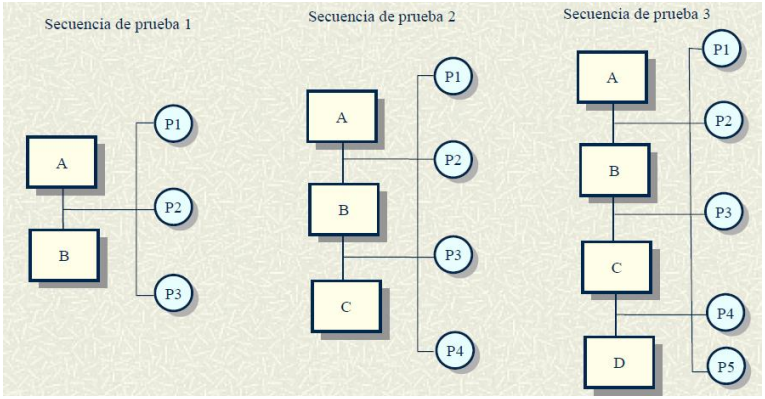
- Los depuradores “conocen” los errores comunes de los programadores
- Realizan una comparación contra patrones observados.
- Localización del defecto:
  - Conocimiento sobre el tipo de defecto
  - Patrón de salida
  - Lenguaje y proceso de programación.

Pruebas de integración

- Pruebas del sistema: implican integrar varios componentes y probar el sistema integrado.
- Existen dos fases de pruebas de sistema:
  - Pruebas de Integración:
    - Equipo de testing tiene acceso al código del sistema.
    - Cuando descubre un problema, se intenta localizar el componente que lo origina y que debe ser depurado.
    - Objetivo: encontrar defectos en el sistema.
  - Pruebas de Entrega:
    - se prueba la versión a entregar al cliente sin disponer del código.
    - Son pruebas de caja negra
    - Cuando participa el cliente en estas pruebas, se denominan Pruebas de Aceptación.
    - Si la entrega es lo suficientemente buena, el cliente la acepta para su uso.

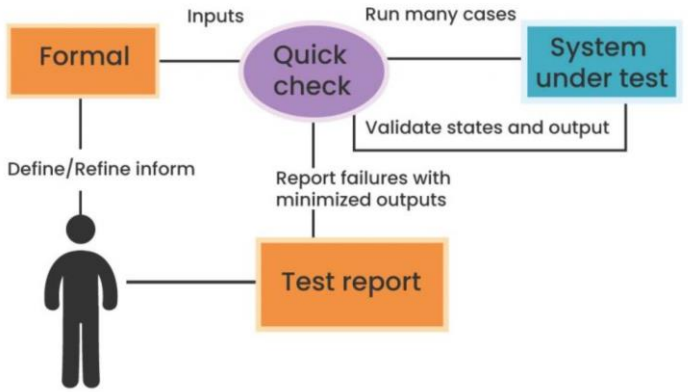
Pruebas de integración

- Los componentes a integrar pueden ser de distintos tipos:
  - Componentes comerciales
  - Componentes reutilizados que han sido adaptados
  - Componentes nuevos.
- La integración implica:
  - identificar grupos de componentes que proporcionan alguna funcionalidad del sistema
  - integrarlos añadiendo código para hacer que funcionen conjuntamente.
- Dos estrategias:
  - Descendente
  - Ascendente
  - Ambas.



Monkey testing

- Técnica de testing de caja negra
  - Definición de hitos a ser probados
    - elementos de carácter unitario
    - elementos cuya funcionalidad es proporcionada por varias capas de la arquitectura de la aplicación
    - Ej. campos de un formulario
    - Se les proporciona datos de entrada aleatorio
      - puede tener acotado el patrón de generación.
        - Tipos:
          - Dumb Monkey Testing
          - Brilliant Monkey Testing
          - Smart Monkey Testing.



Generación de casos de prueba

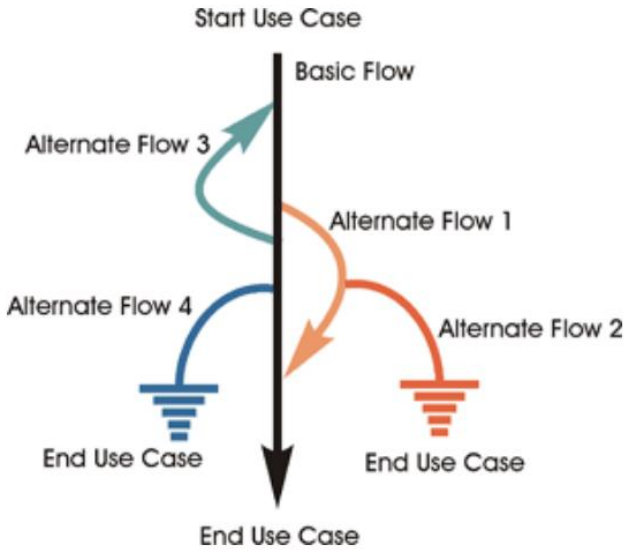
- Generando casos de prueba a partir de casos de uso (Heumann, 2001)
  - Ayuda a iniciar el proceso de testing en etapas tempranas en el ciclo de desarrollo.
  - CU
    - Cliente
    - Desarrollador
    - Documentación
    - Tester

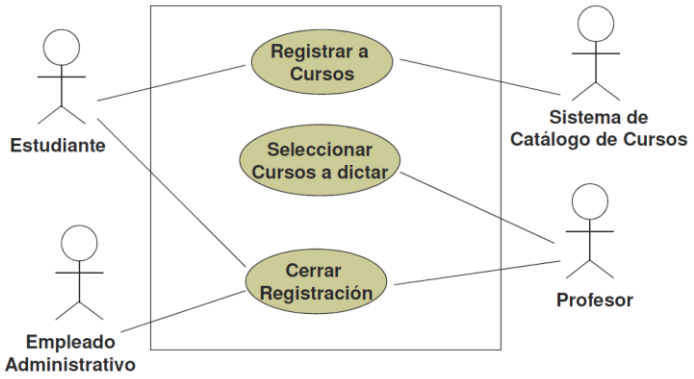
1. Casos de uso
2. Escenarios de casos de uso
3. Casos de prueba (condiciones válidas e inválidas)
4. Procedimientos de prueba (con datos reales)

Los casos de prueba son clave para el proceso porque identifican y comunican las condiciones que se implementarán en la prueba y son necesarios para verificar la implementación exitosa y aceptable de los requisitos del producto.

Generación de casos de prueba

- Generar casos de prueba a partir de un CU:
  - Flujo básico de eventos
  - Flujo alternativo (define “desvíos” al flujo normal y excepciones).





Flujo básico

CU “Registrar a cursos”

1. **Login**: Este caso de uso comienza cuando un estudiante accede al sitio web de la universidad. El sistema le solicita y el estudiante ingresa su ID y password.
2. **Seleccionar “Crear un Cronograma”**: El sistema muestra las funciones disponibles al estudiante, quien selecciona “Crear un Cronograma”.
3. **Obtener información de cursos**: El sistema recupera una lista de cursos disponibles del Sistema de Catálogo de Cursos y la muestra al estudiante.
4. **Seleccionar cursos**: El estudiante selecciona cuatro cursos principales y dos alternativos de la lista de cursos disponibles.
5. **Confirmar Cronograma**: El estudiante indica que el cronograma está completo. Para cada curso seleccionado, el sistema verifica que el estudiante cumple los prerrequisitos necesarios.
6. **Mostrar Cronograma completo**: El sistema muestra el cronograma conteniendo los cursos seleccionados y el nro de confirmación.

Flujo alternativo

CU “Regsitrar a cursos”

1. **Estudiante No identificado**: En el paso 1 del flujo básico: Login, si el sistema determina que el ID o la password del estudiante no son válidos, se muestra un mensaje de error.
2. **Salir**: El sistema permite al estudiante salir en cualquier momento durante el caso de uso. El estudiante puede elegir guardar un cronograma parcial antes de salir: todos los cursos no marcados como “inscripto”, son marcados como “seleccionado”. El cronograma se guarda en el sistema. El CU termina.
3. **Prerrequisitos no cumplidos**, **curso completo o conflictos de cronograma**: En el paso 5 del flujo básico: Confirmar Cronograma, si el sistema determina que los prerrequisitos no son cumplidos, que el curso está completo o que existen conflictos de cronograma, el sistema no inscribirá al estudiante en el curso. Se muestra un mensaje avisando que puede seleccionar otro curso. El CU vuelve al Paso 4, Seleccionar cursos, del flujo básico.

Flujo alternativo

CU “Regsitrar a cursos”

4. **Sistema de Catálogo de Cursos no disponible:** En el paso 3 del flujo básico: Obtener información de curso, si el sistema no está funcionando, se muestra un mensaje y el CU termina.
5. **Registración cerrada**: Si cuando comienza el CU, se determina que la registración ha sido cerrada, se muestra un mensaje y el CU termina.

Escenarios de casos de uso

- Escenarios de CU
  - Son instancias de un caso de uso.
  - Describen un camino completo del caso de uso
    - con alternativas y excepciones
    - sin alternativas y excepciones
  - Debe:
    - Generar un escenario por combinación de flujos básicos y alternativos del caso de uso.
    - Determinar lo que se espera que el sistema haga, y para quién y con quién lo hará.
    - Especificar escenarios de uso que cubran todos los posibles caminos a través de las funciones del sistema.

Escenarios posibles

- CU “Registrar cursos”

1. Estudiante no Identificado.
2. Salir.
3. Prerrequisitos no cumplidos, curso completo o conflictos de cronograma.
4. Sistema de Catálogo de Cursos no disponible.
5. Registración cerrada.

Escenario 1	Flujo Básico			
Escenario 2	Flujo Básico	Flujo Altern. 1		
Escenario 3	Flujo Básico	Flujo Altern. 2		
Escenario 4	Flujo Básico	Flujo Altern. 3		
Escenario 5	Flujo Básico	Flujo Altern. 3	Flujo Altern.1	
Escenario 6	Flujo Básico	Flujo Altern. 3	Flujo Altern.1	Flujo Altern. 2
Escenario 7	Flujo Básico	Flujo Altern. 4		
Escenario 8	Flujo Básico	Flujo Altern. 3	Flujo Altern.4	

Generación de casos de prueba

- Caso de prueba:
  - conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular
  - ejercitar o verificar la conformidad de una funcionalidad específica.
  - Son necesarios para verificar la implementación exitosa y aceptable de los requerimientos del producto.
- Proceso de 3 pasos:
  1. Para cada caso de uso, generar un conjunto completo de escenarios de caso de uso.
  2. Para cada escenario identificar al menos un caso de prueba y las condiciones que lo harían “ejecutarse”.
  3. Para cada caso de prueba, identificar los valores de los datos con los cuales probar la funcionalidad.

Paso 1: Generar escenarios

- Leer la descripción textual del CU e identificar la combinación del flujo básico con los alternativos y crear una matriz de escenarios.
- Para el ejemplo (sin considerar anidamientos alternativos):

Nombre del Escenario	Flujo de comienzo	Flujo Alternativo
Escenario 1 – Registración Exitosa	Flujo Básico	
Escenario 2 – Estudiante no identificado	Flujo Básico	A1
Escenario 3 – Usuario Cancela	Flujo Básico	A2
Escenario 4 – Sistema de Catálogo de Cursos no disponible	Flujo Básico	A4
Escenario 5 – Registración cerrada	Flujo Básico	A5
Escenario 6 – No se puede inscribir	Flujo Básico	A3

Paso 2: Identificar casos de prueba

- Deben analizarse los escenarios y revisarlos con la descripción textual del CU
- Identificar las condiciones o elementos de datos requeridos para ejecutar cada escenario.
- Debe existir, al menos, un caso de prueba para cada escenario.
- La cantidad de casos de prueba a considerar para cada escenario depende del grado de generalidad utilizado para describir los flujos alternativos
  - Ej. A3. Prerrequisitos no cumplidos, curso completo o conflictos de cronograma

Paso 2: Identificar casos de prueba

ID Caso	Escenario / Condición	ID Estud	Passw	Cursos seleccion	Cumple Prerreq.	Curso abierto	Cronog. abierto	Result. Esperado
CP 1	Esc. 1 – Registración Exitosa	V	V	V	V	V	V	Cronog. y nro confirmac.
CP 2	Esc. 2 – Estudiante no identificado	I	N/A	N/A	N/A	N/A	N/A	Mens. error. Vuelta a pantalla login
CP 3	Esc. 3 – Usuario Cancela	V	V	N/A	N/A	N/A	N/A	Pantalla login
CP 4	Esc. 4 – Sistema de Catálogo de cursos no disponible	V	V	N/A	N/A	N/A	N/A	Mensaje de Error: vuelta paso 2
CP 5	Esc. 5 – Registración cerrada	V	V	N/A	N/A	N/A	N/A	Mensaje de Error: vuelta paso 2
CP 6	Esc. 6 – Inscripción negada: curso lleno	V	V	V	V	I	V	Mensaje de Error: vuelta paso 3
CP 7	Esc. 6 – Inscripción negada: no cumple prerrequisitos	V	V	V	I	V	V	Mensaje de Error: vuelta paso 4
CP 8	Esc. 6: Inscripción negada: conflicto cronograma	V	V	V	V	V	I	Mensaje de Error: vuelta paso 4

Paso 2: Identificar casos de prueba

- Referencias:
  - *V (valor válido),*
  - *I (valor inválido)*
  - *N/A (no* aplica).
- Matriz:
  - Paso intermedio
  - Muestra condiciones que deben ser evaluados para probar cada caso.
- Cada columna de la matriz debería tener al menos una I
  - indica la prueba de una condición inválida.
- En el ejemplo, los casos RC3, RC4 y RC5 tienen los mismos valores de prueba porque por espacio se dejó de declarar alguna condición (por ejemplo, si la registración está abierta, el sistema de catálogos disponible, etc.).

Paso 3: Identificar valores de datos para probar

- Una vez identificados todos los casos de prueba
  - deben ser revisados y verificados
    - para asegurar su corrección
    - para identificar casos de prueba redundantes o faltantes.
- El paso final es sustituir las "*I*" *y* "*V*" *en la* matriz generada en el paso anterior por valores de datos reales.
- Objetivo:
  - implementar o ejecutar los casos de prueba identificados.

Paso 3: Identificar valores de datos para probar

ID Caso	Escenario / Condición	ID Estud	Passw	Cursos selecc.	Cumple Prerreq.	Curso abierto	Cronog. abierto	Result. Esperado
RC 1	Esc. 1 – Registración Exitosa	jheumann	abc123	M101 E201 S101	Si	Si	Si	Cronog. y nro confirmac.
RC 2	Esc. 2 – Estudiante no identificado	jheumann1	N/A	N/A	N/A	N/A	N/A	Mens. error. Vuelta a pantalla login
RC 3	Esc. 3 – Salidas válidas del usuario	jheumann	abc123	N/A	N/A	N/A	N/A	Pantalla login
RC 4	Esc. 4 – Sistema de Catálogo de cursos no disponible	jheumann	abc123	N/A	N/A	N/A	N/A	Mensaje de Error: vuelta paso 2
RC 5	Esc. 5 – Registración cerrada	jheumann	abc123	N/A	N/A	N/A	N/A	Mensaje de Error: vuelta paso 2
RC 6	Esc. 6 – Inscripción negada: curso lleno	jheumann	abc123	M101 E201 S101	Si	M101 completo	Si	Mensaje de Error: vuelta paso 3
RC 7	Esc. 6 – Inscripción negada: no cumple prerrequisitos	jheumann	abc123	M101 E201 S101	No para E201	Si	Si	Mensaje de Error: vuelta paso 4
RC 8	Esc. 6: Inscripción negada: conflicto cronograma	jheumann	abc123	M101 E201 S101	Si	Si	Conflicto E201y S101	Mensaje de Error: vuelta paso 4