Facultad 8

Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas

Título: Propuesta de un proceso de pruebas basado en RUP para Proyectos de Gestión.

Autores: Raúl Carralero Mulet
Fabian Felipe Cedeño González

Tutor: Lic. Lucy Cruz Águila. Especialista de Calidad Dirección de Desarrollo Empresa SOFTEL del MIC.

"Ciudad de la Habana, Junio, 2007"

Declaración de Autoría

Declaramos que Fabian Felipe Cedeño González y Raúl Carralero Mulet somos los únicos autores de este trabajo y autorizamos a la Empresa Softel y la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los 18 días del mes de junio del año 2007.

Fabian Felipe Cedeño González

_____

Raúl Carralero Mulet

_____

Lic. Lucy Cruz Águila

_____

Datos de contacto

Tutor: Lic. Lucy Cruz Águila

Especialista de calidad de la dirección de Desarrollo de la Empresa SOFTEL, graduada de Licenciatura en Cibernética-Matemática en el año 1988.

Ha desarrollado diferentes proyectos de gestión en la esfera de la salud, turismo, empresarial, ministerial, ha dirigido técnicamente proyectos de gestión en estas áreas.

Fue directora de tecnología y calidad de la empresa SOFTEL en el período del 2000 al 2003.

Ha participado en diferentes eventos científicos-técnicos en temas de calidad del desarrollo de software.

Ha recibido diferentes cursos de post-grados en ingeniería de software, programación y calidad.

Correo electrónico: lucycruz@softel.cu

Ubicación: SOFTEL, UCI, Cuba

“La calidad nunca es un accidente; siempre es el resultado de un esfuerzo de la inteligencia.”

John Ruskin

Agradecimientos

A mis padres, que son mi fuente de inspiración, por su eterno apoyo y comprensión durante el desarrollo de nuestra tesis.

A mi familia, por brindarme su apoyo, por hacerme ver que lo más importante en la vida es saber luchar por construir un futuro.

A nuestra tutora, por su ayuda, sus consejos y sus jalones de oreja cuando nos hicieron falta.

A Yasmín, por su amor y entrega, por su apoyo y ánimo cuando estaba cansado de trabajar.

Al colectivo de trabajadores de SOFTEL, por su apoyo.

A nuestros profesores y compañeros, que de una u otra forma nos han apoyado en la realización de nuestra tesis.

A los creadores de este proyecto de la Revolución, en especial a Fidel, nuestro principal guía.

Fabian

A mis padres, por tanto amor y confianza depositados en mi, por enseñarme a luchar por mis sueños, y entre ellos la realización de este trabajo de diploma.

A mi hermano querido por darme su apoyo durante esta etapa final de mis estudios.

A mi familia por haber depositado en mi toda su confianza en el desarrollo de esta tesis, no los defraudaré.

A Perla, por todo su amor y entrega en todos los momentos vividos, principalmente en estos días tan cargados de trabajo.

A nuestra tutora, por todo su tiempo y entrega para que nuestro trabajo saliera lo mejor posible.

A los compañeros de SOFTEL, por su apoyo.

A nuestros profesores y compañeros, que tanto apoyo nos brindaron para hacer realidad este sueño.

A Jorge, Susana, Rosa por su ayuda.

A Nuestro Comandante Fidel Castro y a la dirección de nuestra Revolución, por ser nuestro faro y guía.

Raúl

A nuestros padres…

Resumen

Para garantizar la calidad de software se realizan grandes esfuerzos, pues la construcción de los actuales sistemas exige poco tiempo de desarrollo y un correcto cumplimiento de los objetivos para los cuales se originan. Las pruebas de software son de vital importancia para alcanzar altos índices de efectividad en la creación de productos informáticos. El presente trabajo surge a partir de la necesidad de la aplicación de una correcta definición de esta disciplina que se ajuste a proyectos de gestión. A partir de un estudio realizado acerca del estado de las pruebas en nuestra universidad, en conjunto con el ámbito teórico de pruebas, el flujo de trabajo que propone RUP, las buenas prácticas de PSP y TSP, además de herramientas para la administración y la ejecución de pruebas, en ambiente de software libre y propietario, se propone una definición del proceso de prueba, aplicable a proyectos de gestión, y se describe detalladamente el flujo de trabajo resultante, con los respectivos roles; una fusión de los artefactos y entregables que se generan durante el proceso; y las actividades o tareas a realizar. Una correcta administración del proceso de pruebas, sumada a la responsabilidad de todas las personas involucradas en un proyecto y específicamente en el control de la calidad de software, siempre brinda buenos resultados.

Tabla de Contenidos

Introducción **1**
Capítulo 1 Fundamentación teórica de Pruebas **4**
Conceptos: 5
Objetivos de las Pruebas 6
1.1 Métodos de Prueba 7
1.1.1 Prueba de caja blanca 7
1.1.1.1 Prueba del camino básico 7
1.1.1.2 Notación de Grafo de Flujo. 8
1.1.1.3 Ejemplo de cómo elaborar un Grafo de Flujo. 10
1.1.1.4 Complejidad Ciclomática. 11
1.1.1.5 Prueba de condición 12
1.1.1.6 Prueba de flujos de datos 13
1.1.1.7 Prueba de bucles 13
1.1.2 Prueba de Caja Negra 15
1.1.2.1 Partición equivalente. 16
1.1.2.2 Análisis de Valores Límites: 18
1.2 Niveles de Prueba 18
1.2.1 Prueba de Unidad 18
1.2.2 Prueba de Integración 20
1.2.2.1 Integración descendente 21
1.2.2.2 Integración Ascendente 22
1.2.3 Prueba de regresión 23
1.2.4 Prueba de aceptación 23
1.2.5 Prueba del sistema 24
1.3 Estado actual de las pruebas en la UCI 27
1.4 Proceso Unificado de Desarrollo 31
1.4.1 Fases de RUP y flujo de pruebas: 31
1.4.2 Roles implicados: 33
1.4.3 Flujo de trabajo de prueba de RUP: 38
1.4.4 Descripción de las actividades fundamentales: 41
1.5 PSP 47
1.6 TSP 48

1.7 Herramientas 51
Capítulo 2 Propuesta de Flujo de Trabajo de Prueba 52
2.1 Propuesta de flujo de trabajo de Pruebas: 52
2.2 Tareas Fundamentales 54
Planificar pruebas 55
Verificar enfoque 58
Diseñar las pruebas: 60
Implementar las pruebas 62
Probar y evaluar 64
Mejorar calidad de las pruebas 66
2.3 Artefactos propuestos 67
2.4 Consideraciones: 70
Capítulo 3 Herramientas para la administración y ejecución de Pruebas 72
3.1 Herramientas administrativas: 72
3.1.1 Administración de pruebas: 72
3.1.2 Test Manager: 73
3.2 Herramientas de Seguimiento: 74
3.2.1 Rational ClearQuest 75
3.3 Herramientas de ejecución 76
3.3.1 Herramientas Propietarias 76
3.3.1.1 Rational Robot 77
3.3.2 Herramientas Libres 78
3.3.2.1 JUnit 78
3.3.2.2 JMeter: 78
3.4 Consideraciones: 80
Conclusiones: 81
Recomendaciones 82
Referencias bibliográficas 83
Bibliografía: 86
Anexos ..... Error! Bookmark not defined.
Glosario de términos: 89

Introducción

Las tecnologías de la información y las comunicaciones (TIC) se desarrollan a un ritmo elevado en la actualidad. Miles de sistemas alrededor del planeta se encuentran en funcionamiento, y cada día salen a la luz nuevos programas y aplicaciones, con un objetivo: facilitar y amenizar la vida del hombre.

En el mundo actual en el que se necesita obtener productos con alta calidad, que integran variadas tecnologías y cuyo tiempo de elaboración sea mínimo. Se impone mejorar el proceso de desarrollo de software, incrementando la productividad de los equipos involucrados en el desarrollo de los proyectos. (HUMPHREY, 1997)

Nuestro país se encuentra inmerso en una batalla por la informatización de la sociedad, la que ha hecho sentar sobre cuatro pilares fundamentales: la educación, la salud, la seguridad social y la cultura. Y podría definirse como el proceso de utilización ordenada y masiva de las tecnologías de la información y las comunicaciones para satisfacer las necesidades de información y conocimiento de todas las personas y esferas de la sociedad. Este proceso busca lograr más eficacia y eficiencia, que permitan una mayor generación de riquezas y hagan sustentable el aumento sistemático de la calidad de vida de los ciudadanos.

El desarrollo de una Industria Nacional de Software es una tarea de gran prioridad para el estado cubano, debido a la alta perspectiva económica que posee, así como para el aseguramiento de un grupo importante de actividades del país. (MORENO, 2003)

La Industria Cubana de Software (ICSW) está llamada a convertirse en una significativa fuente de ingresos nacional, como resultado del correcto aprovechamiento de las ventajas del considerable capital humano disponible. Nuestra universidad y el sistema de empresas cubanas vinculadas a este trabajo están jugando un papel importante en el desarrollo de la industria del Software, y en la materialización de los proyectos asociados al programa cubano de informatización.

La promoción de nuestra industria de software en el ámbito internacional ha tenido como línea estratégica aprovechar la enorme credibilidad que tiene Cuba en sectores tales como la salud, la educación y el deporte. Continuar la producción sostenida de software de alta calidad en

prestaciones, imagen y soporte, para satisfacer las necesidades nacionales en estos sectores, tiene ya una positiva repercusión en el incremento de la exportación. (CUBAMINREX, 2005)

Es por ello que se hace tan importante hoy en día obtener productos con una alta calidad a partir de procesos de desarrollos profesionales, no artesanales y bien organizados. Uno de los procedimientos que garantizan la competitividad de un producto es la ejecución del proceso de pruebas durante el desarrollo del software aplicando las mejores prácticas que se establecen para esta disciplina.

El desarrollo de sistemas de software implica una serie de actividades de producción en las que las posibilidades de que aparezca el fallo humano son enormes. Los errores pueden empezar a darse desde el primer momento del proceso, en el que los objetivos…pueden estar especificados de una forma errónea o imperfecta, así como en posteriores pasos del diseño y desarrollo. Debido a la imposibilidad humana de trabajar y comunicarse de forma perfecta, el desarrollo de software debe ir acompañado de una actividad que garantice la calidad. (DEUTSCH, 1979)

Las pruebas de software son un elemento crítico para la garantía de la calidad de software y representa una revisión final de las especificaciones, del diseño y de la codificación. (PRESSMAN, 1998)

Las pruebas de software se desarrollan en distintas etapas, con el propósito de lograr un producto con la menor cantidad de errores posibles y que cumpla con las necesidades del cliente. Es por ello que la efectividad de este proceso depende de la capacidad que tenga de brindar alta probabilidad de encontrar errores.

Existen dificultades a la hora de utilizar o llevar a cabo la aplicación de las pruebas actualmente, por lo que nos hemos planteado la siguiente situación problémica: No se llevan a cabo de forma correcta las tareas a desarrollar en cada fase del proceso de desarrollo correspondientes al flujo de trabajo de pruebas, los métodos, estrategias, tipos de pruebas y cómo administrar este proceso y estandarizar los procedimientos de trabajo y uso de herramientas que agilicen y garanticen niveles de productividad, profesionalidad y calidad competente. Además, la disciplina de prueba no se ve como un flujo de trabajo dentro del proceso de Ingeniería de Software.

Para dar respuesta al problema anterior se define el objeto de estudio: Flujo de trabajo de pruebas del ciclo de desarrollo de software y técnicas de PSP (Personal Software Process) y TSP (Team Software Process).

Como problema científico podemos decir que no se aplica una correcta definición de la disciplina de prueba generalizable en proyectos de gestión, planteándonos los siguientes objetivos:

- Obtener una definición del proceso de prueba generalizable para Proyectos de Gestión.
- Proponer un flujo de trabajo para desarrollar el proceso de prueba basados en RUP (Proceso Unificado de Desarrollo), y las buenas prácticas que brinda PSP (Proceso Personal de Software) y TSP (Proceso de Software en Equipo).
- Analizar diferentes herramientas para la ejecución, seguimiento y administración de las pruebas.

Las tareas realizadas para dar cumplimiento a los objetivos fueron:

- Estudiar el estado de las pruebas en nuestro país y particularmente en nuestra Universidad.
- Analizar los métodos, estrategias, tipos de pruebas y niveles de pruebas.
- Analizar el flujo de trabajo de pruebas de RUP y las técnicas de PSP y TSP.
- Proponer un flujo de prueba generalizable para proyectos de gestión.
- Analizar diferentes herramientas automatizadas para la ejecución, seguimiento y administración de las pruebas.

Esperándose como resultado una propuesta de definición del proceso de pruebas basado en las mejores prácticas para aplicar a proyectos de gestión desarrollados por nuestra Universidad y la empresa Softel.

El trabajo está estructurado de la siguiente forma:

Capítulo I Contiene información teórica de pruebas, niveles, tipos, métodos. Además del estudio del flujo de trabajo de pruebas que propone RUP (Rational Software Process), las técnicas de PSP (Personal Software Process) y TSP (Team Software Process).

Capítulo II Propuesta de un flujo de trabajo de pruebas, a partir de la vinculación de RUP con las técnicas de TSP y PSP.

Capítulo III Análisis de las herramientas que facilitan la administración, seguimiento y ejecución de las pruebas, tanto propietarias como libres.

Capítulo 1: Fundamentación teórica de Pruebas

El objetivo específico de las de pruebas es encontrar la mayor cantidad de errores posibles dentro de un software. Por este motivo probar es ejercitar un programa con la peor intención, para sacar a relucir todos los fallos que pueda contener.

Para un diseño de casos de pruebas efectivo un ingeniero de software tiene que entender los principios básicos que guían a las pruebas. Davis (DAVIS, 1995) sugiere un conjunto de principios de prueba que mostramos a continuación:

- A todas las pruebas se le debe hacer un seguimiento hasta los requisitos del cliente: ya que los defectos más graves (desde el punto de vista del cliente) son aquellos que impiden al programa cumplir sus requisitos.

- Las pruebas deberían de planificarse mucho antes de que empiecen.

- El principio de Pareto es aplicable a la prueba del software: esto implica que el 80 por ciento de todos los errores descubiertos durante las pruebas surgen al hacer un seguimiento de solo el 20 por ciento de todos los módulos del programa. El problema, por supuesto es aislar estos módulos sospechosos y probarlos concienzudamente.

- Las pruebas deberían empezar por “lo pequeño” y progresar “hacia lo grande”: Las primeras pruebas planeadas y ejecutadas se centran generalmente en módulos individuales del programa. A medida que avanzan las pruebas, desplazan su punto de mira en un intento de encontrar errores en grupos integrados de módulos y finalmente en el sistema entero.

- No son posibles las pruebas exhaustivas: es imposible cubrir todos los caminos durante la prueba. Pero si es posible, sin embargo, cubrir la lógica del programa y asegurarse de que se han aplicado todas las condiciones del diseño procedimental.

Conceptos:

Para la mejor comprensión mostramos algunos conceptos que son importantes para un mayor entendimiento, estos son:

De acuerdo a la IEEE (IEEE, 1991) el concepto de prueba se define como: “una actividad en la cual un sistema o uno de sus componentes se ejecuta dos en circunstancias previamente especificadas, los resultados se observan y registran y se realiza una evaluación de algún aspecto”.

Un concepto más específico dado por algunos desarrolladores de software es que las pruebas son: “Cualquier intento de demostrar que el software tiene propiedades por debajo de la calidad requerida”. (CIG_LABS, 2002)

Caso de prueba: “un conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular. También se puede referir a la documentación en la que se describen las entradas, condiciones y salidas de un caso de prueba (ACUÑA, 2007). En el anexo 1 se abordan temas referentes a la confección de casos de prueba.

Error: es un defecto, un resultado incorrecto o una acción humana que conduce a un resultado incorrecto. (ACUÑA, 2007)

Defecto: Un paso de procesamientos incorrectos en un programa. (ACUÑA, 2007)

Fallo: La incapacidad de un sistema o de alguno de sus componentes para realizar las funciones requeridas dentro de los requisitos de rendimiento especificados. (ACUÑA, 2007)

Objetivos de las Pruebas

Glen Myers (MYERS, 1979), establece varias normas como objetivos de las pruebas:

1. La prueba es un proceso de ejecución de un programa con la intención de descubrir un error.
2. Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
3. Una prueba tiene éxito si descubre un error no detectado hasta entonces.

Una buena prueba según Kaner, Falk y Nguyen (KANER, 1993) posee los siguientes atributos:

1. Una buena prueba tiene una alta probabilidad de encontrar un error.
2. Una buena prueba no debe ser redundante.
3. Una buena prueba debería ser ¨ la mejor de la cosecha ¨.
4. Una buena prueba no debería ser ni demasiado sencilla ni demasiado compleja.

Los objetivos anteriores suponen un cambio importante de punto de vista. Nos quitan la idea que, normalmente, tenemos de que una prueba tiene éxito si no descubre errores. Nuestro objetivo es diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo. (PRESSMAN, 2000)

Si la prueba se lleva a cabo con éxito, según los objetivos antes mencionados descubrirá errores en el software.

“La prueba no puede asegurar la ausencia de defectos, solo puede demostrar que existen defectos en el software”. (PRESSMAN, 2000)

1.1 Métodos de Prueba

1.1.1 Prueba de caja blanca

Sinónimos:

- pruebas estructurales
- pruebas de caja transparente

La prueba de Caja Blanca es un método de prueba que garantiza que se disminuya los errores existentes en los sistemas lográndose productos de mayor calidad y confiabilidad. Esto se logra ya que la prueba de caja blanca se basa en el diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivarlos. Mediante la prueba de la caja blanca según (PRESSMAN, 2005) el ingeniero del software puede obtener casos de prueba que:

1. Garanticen que se ejecuten por lo menos una vez todos los caminos independientes de cada módulo, programa o método, a través de las pruebas de camino básico descrita en el epígrafe 1.1.1.1.
2. Ejerciten todas las decisiones lógicas en las vertientes verdadera y falsa, a través de las pruebas de condición descritas en el epígrafe 1.1.1.2.
3. Ejerciten las estructuras internas de datos para asegurar su validez, a través de la prueba de flujo de datos descrita en el epígrafe 1.1.1.3.
4. Ejecuten todos los bucles en sus límites operacionales a través de las pruebas de bucles descritas en el epígrafe 1.1.1.4.

Es por ello que se considera a la prueba de Caja Blanca como uno de los tipos de pruebas más importantes que se le aplican a los software, logrando como resultado que disminuya en un gran por ciento el número de errores existentes en los sistemas y por ende una mayor calidad y confiabilidad. (PRESSMAN, 2005)

1.1.1.1 Prueba del camino básico

La prueba del camino básico es una técnica de prueba de la Caja Blanca propuesta por (MCCABE, 1976).

Esta técnica permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico.

La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el Grafo de Flujo asociado y se calcula su complejidad ciclomática. Los pasos que se siguen para aplicar esta técnica son:

1. A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
2. Se calcula la complejidad ciclomática del grafo.
3. Se determina un conjunto básico de caminos independientes.
4. Luego de tener elaborados los Grafos de Flujos y los caminos a recorrer, se preparan los casos de prueba que forzarán la ejecución de cada uno de esos caminos. Se escogen los datos de forma que las condiciones de los nodos predicados estén adecuadamente establecidas, con el fin de comprobar cada camino.

Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecuten por lo menos una vez cada sentencia del programa.

1.1.1.2 Notación de Grafo de Flujo.

Para aplicar la técnica del camino básico se debe introducir una sencilla notación para la representación del flujo de control, el cual puede representarse por un Grafo de Flujo.
Cada nodo del grafo corresponde a una o más sentencias de código fuente. Todo segmento de código de cualquier programa se puede traducir a un Grafo de Flujo.
Para construir el grafo se debe tener en cuenta la notación para las instrucciones.

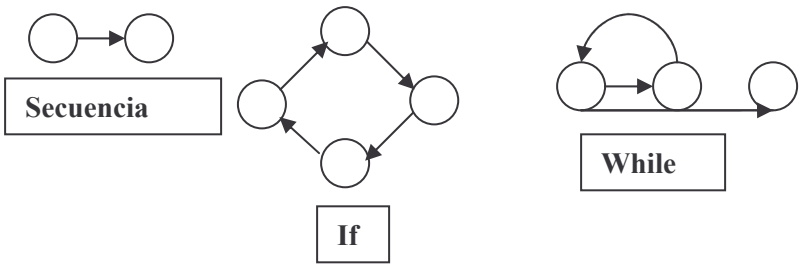


Figura 1.1 Notación de grafos de flujo para las instrucciones

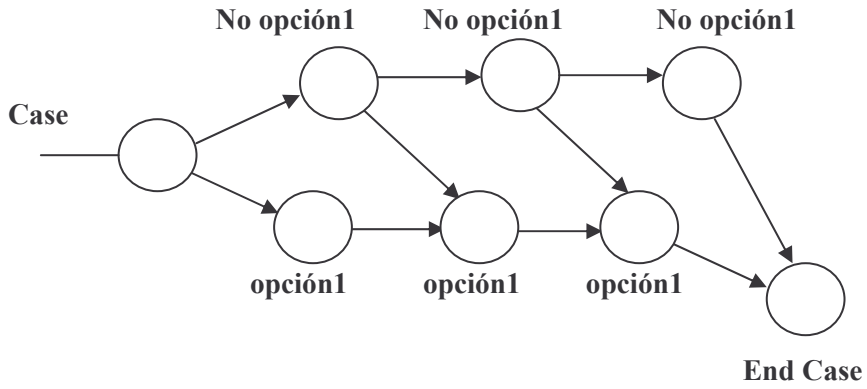


Figura 1.2 Notación de grafos de flujo para la instrucción Case.

Un Grafo de Flujo está formado por tres componentes fundamentales estos son:

Nodo:
Representa una o más secuencias procedimentales. Un solo nodo puede corresponder a una secuencia de procesos o a una sentencia de decisión figura 1.3. Puede ser también que hallan nodos que no se asocien, se utilizan principalmente al inicio y final del grafo.

Arista:
Una flecha dentro del grafo que representa el flujo de control. Una arista debe terminar en un nodo, incluso aunque el nodo no represente ninguna sentencia procedimental figura 1.3.

Región:
La región es un área delimitada por las aristas y nodos. También se incluye el área exterior del grafo, contando como una región más. Las regiones se enumeran y la cantidad de regiones es equivalente a la cantidad de caminos independientes del conjunto básico de un programa figura 1.3.

Un ejemplo de representación de Grafo de Flujo es el mostrado en la Figura 1.3 en el cual aparecen sus componentes:

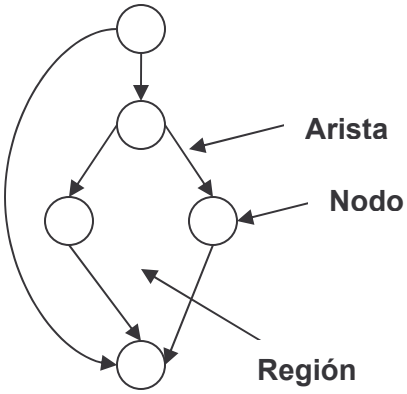


Figura 1.3 Características de los grafos.

Cualquier representación del diseño procedimental se puede traducir a un grafo de flujo. Cuando en un diseño se encuentran condiciones compuestas (uno o más operadores OR, AND, NAND, NOR lógicos en una sentencia condicional), la generación del grafo de flujo se hace un poco más complicada. (PRESSMAN, 2000)

1.1.1.3 Ejemplo de cómo elaborar un Grafo de Flujo.

En la figura 1.4 se tiene un segmento de código que posee una sentencia while y luego una if, de cada una de estas condiciones salen dos nodos, que se consideran nodos predicados por tener dos a más aristas saliendo de él (ej. Nodo 2 y 4).

```
public int Buscar(string  ISBN)
{
bool finish = false;    1
int i = 0;     1
while(i < CatalogoPeliculas.Count && !finish)    2
{
finish = (CatalogoPeliculas[i] as CE_Pelicula).ISBN == ISBN;   3
i++;     3
}
if (finish)          4
return i - 1;       5
else
return -1;   6
}     7
```

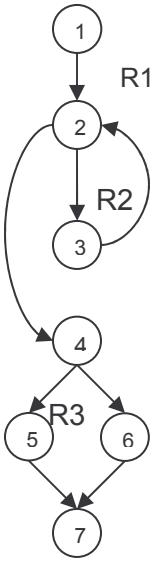


Figura 1.4 Representación de un Grafo de Flujo.

1.1.1.4 Complejidad Ciclomática.

La complejidad ciclomática es una métrica de software extremadamente útil pues proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez.

Un camino independiente es cualquier camino del programa que introduce por lo menos un nuevo conjunto de sentencias de procesamiento o una nueva condición. El camino independiente se debe mover por lo menos por una arista que no haya sido recorrida anteriormente. (PRESSMAN, 2005)
Basándonos en la figura 1.4 unos ejemplos de caminos independientes serian:
Camino 1: 1 – 2 – 4 – 6 - 7
Camino 2: 1 – 2 – 3 – 2 - 4 – 5 - 7

Si se diseñan pruebas que fuercen el recorrido de esos caminos, se garantiza que se ejecute al menos una vez cada sentencia del programa y que cada condición se ejecute en sus variantes verdadera y falsa. Se debe tener en cuenta que de un mismo diseño procedimental se pueden derivar varios conjuntos básicos. (ISABEL, 2002; PRESSMAN, 2000)

Hay tres formas fundamentales de calcular la complejidad ciclomática y utilizaremos como ejemplo la figura 1.4:

1. El número de regiones del grafo de flujo coincide con la complejidad ciclomática.

Por lo tanto como existen tres regiones R1, R2 y R3 la complejidad ciclomática es igual a tres.

2. La complejidad ciclomática, V (G), se define como:

V (G) = A - N + 2

V (G)= 8 – 7 + 2

V (G)= 3

Donde: A es el número de aristas del grafo y N es el número de nodos.

3. La complejidad ciclomática, V (G), también se define como:

V (G) = P + 1

V (G) = 2 + 1

V (G) = 3

Donde: P es el número de nodos predicado contenido en el grafo.

1.1.1.5 Prueba de condición.

La prueba de condición es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa (PRESSMAN, 2005). Los tipos de errores de una condición pueden se los siguientes:

- Error en operador lógico (existencia de operadores lógicos incorrectos, desaparecidos o sobrantes).
- Error en variables lógicas.
- Error en paréntesis lógicos.
- Error en operador relacional.
- Error en expresión aritmética.

Este método se centra en la prueba de cada una de las condiciones del programa. Las estrategias de prueba de condiciones tienen generalmente dos ventajas.

1- La cobertura de la prueba de una condición es sencilla.
2- La cobertura de la prueba de las condiciones de un programa da una orientación para generar pruebas adicionales del programa.

El propósito de la prueba de condición es detectar no solo los errores en las condiciones de un programa, sino también otros errores en dicho programa.
En (PRESSMAN, 2005) se puede encontrar el siguiente consejo: Cada vez que decides efectuar una prueba de condición, deberás evaluar cada condición en un intento por descubrir errores. ¡Este es un escondrijo ideal para los errores!

1.1.1.6 Prueba de flujos de datos.

El método de prueba de flujos de datos selecciona caminos de pruebas de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa (PRESSMAN, 2005).

Las estrategias de prueba de flujo de datos son útiles para seleccionar caminos de prueba de un programa que contenga sentencias IF o de bucles anidados. Dado que las sentencias de un programa están relacionadas entre sí de acuerdo con las definiciones de las variables, el enfoque de prueba de flujo de datos es efectivo para la protección contra errores. Sin embargo, los problemas de medida de la cobertura de la prueba y de selección de caminos de pruebas para la prueba de flujo de datos son más difíciles que los correspondientes problemas para la prueba de condición.

Un consejo a tener en cuenta lo menciona Pressman (PRESSMAN, 2005) cuando dice: Es poco realista asumir que la prueba del flujo de datos puede ser usada ampliamente cuando probamos grandes sistemas. En cualquier caso, puede ser utilizada en áreas del software que sean sospechosas.

1.1.1.7 Prueba de bucles.

Los bucles son la piedra angular de la inmensa mayoría de los algoritmos implementados en software. Este tipo de técnica de caja blanca se centra en la validez de las construcciones de bucles. Se recomienda, además de un análisis de camino básico que aísle todos los posibles caminos del bucle, un conjunto especial de pruebas adicionales para cada tipo de bucle, en un intento de descubrir errores de inicialización, de indexación o de incremento y errores en los límites de los bucles. Se pueden definir cuatro clases diferentes de bucles según Beizer (BEIZER, 1990): bucles simples, bucles concatenados, bucles anidados y bucles no estructurados.

1.1.1.7.1 Bucles simples.- Se les puede aplicar el siguiente conjunto de pruebas.

- Saltar totalmente el bucle.
- Pasar una sola vez por el bucle.

- Pasar dos veces por el bucle.
- Hacer *m*-pasos por el bucle *m* < *n*.
- Hacer *n - 1*, *n* y *n + 1* pasos por el bucle.

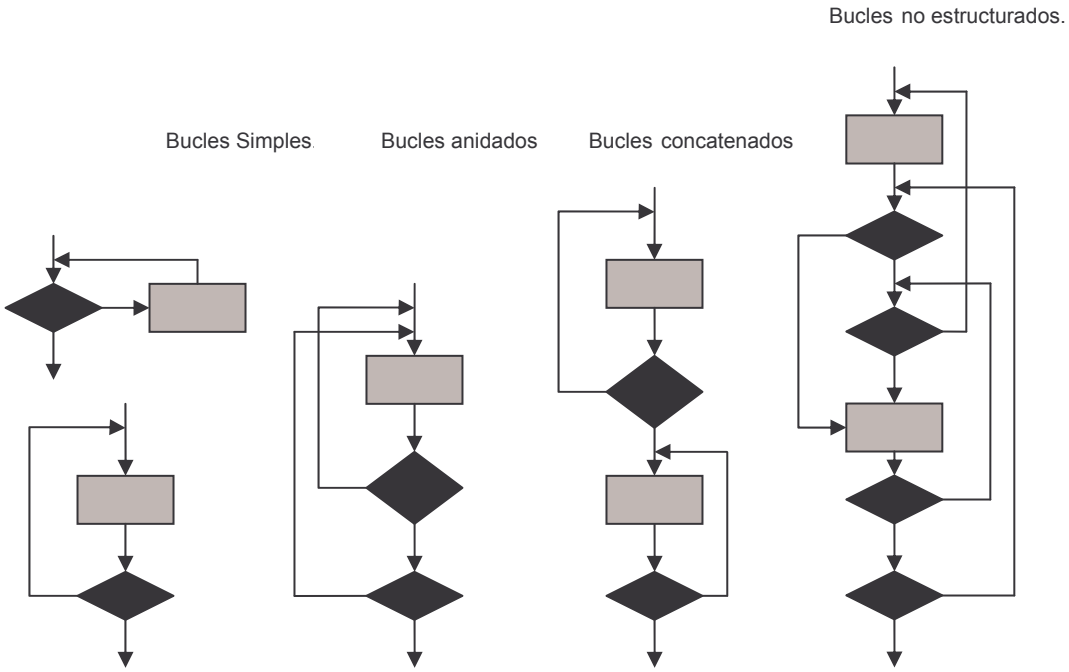


Figura 1.5 Tipos de Bucles

1.1.1.7.2 Bucles anidados.

El enfoque a realizar no consiste en anidar las pruebas de los bucles simples, ya que el crecimiento de pruebas seria exponencial, a medida que aumenta el anidamiento, por el contrario se propone el siguiente conjunto de pruebas según (BEIZER, 1990) que ayuda a reducir el número de pruebas:

- Comenzar en el bucle más interno. Disponer todos los demás bucles en sus valores mínimos.
- Llevar a cabo las pruebas de bucles simples para los bucles más internos dejando los externos con los iteradores en sus valores mínimos. Añadir otras pruebas para los valores fuera de rango o para valores excluidos.
- Progresar hacia afuera, llevando a cabo las pruebas para el siguiente bucle, pero manteniendo todos los demás bucles exteriores en sus valores mínimos y los demás bucles anidados con valores "típicos".
- Continuar hasta que se hayan probado todos los bucles.

1.1.1.7.3 Bucles concatenados.

Se pueden probar mediante el enfoque anteriormente definido para los simples, siempre que estos sean independientes unos de otros, en otro caso se recomienda el enfoque de los bucles anidados.

1.1.1.7.4 Bucles no estructurados.

Se recomienda su no utilización porque puede brindar dificultades.

1.1.2 Prueba de Caja Negra

La prueba de Caja Negra se centra principalmente en los requisitos funcionales del software. Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejecuten completamente todos los requisitos funcionales de un programa.

La prueba de Caja Negra no es una alternativa a las técnicas de prueba de la Caja Blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la Caja Blanca (PRESSMAN, 2005).

Muchos autores como (PRESSMAN, 1998) (MYERS, 1979) (BEIZER, 1995) consideran que estas pruebas permiten encontrar:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a las Bases de Datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación.

Para preparar los casos de pruebas de caja negra hacen falta un número de datos de prueba que ayuden a la ejecución de estos casos y que permitan que el sistema se ejecute en todas sus variantes, pueden ser datos válidos o inválidos para el programa en dependencia de si lo que se quiere es hallar un error o probar una funcionalidad. Los datos se escogen atendiendo a las especificaciones del problema, sin importar los detalles internos del programa, a fin de verificar que el programa se ejecute correctamente.

Para diseñar casos prueba de caja negra existen varias técnicas, entre ellas están: (PRESSMAN, 2000)

1. Técnica de la Partición Equivalente: esta técnica divide el campo de entrada en clases de datos que tienden a ejecutar determinadas funciones del software.
2. Técnica del Análisis de Valores Límites: esta Técnica prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.

1.1.2.1 Partición equivalente.

Una partición equivalente es una técnica de diseño de casos de prueba de Caja Negra que divide el dominio de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba (PRESSMAN, 2005). El diseño de estos casos de prueba para la partición equivalente se basa en la evaluación de las clases de equivalencia.

El diseño de casos de prueba para la partición equivalente se basa en una evaluación de las clases de equivalencia para una condición de entrada. Una clase de equivalencia representa un conjunto de estados válidos o inválidos para condiciones de entrada. Regularmente, una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica. (PRESSMAN, 1997)

Las clases de equivalencia se pueden definir de acuerdo con las siguientes directrices:

- Si un parámetro de entrada debe estar comprendido en un cierto rango, aparecen tres clases de equivalencia: por debajo, en el rango y por encima de este.
- Si una entrada requiere un valor concreto, aparecen tres clases de equivalencia: por debajo, en el rango y por encima de este.
- Si una entrada requiere un valor de entre los de un conjunto, aparecen dos clases de equivalencia: en el conjunto o fuera de él.
- Si una entrada es booleana, hay dos clases: si o no.

Los mismos criterios se aplican a las salidas esperadas: hay que intentar generar resultados en todas y cada una de las clases.

Aplicando estas directrices se ejecutan casos de pruebas para cada elemento de datos del campo de entrada a desarrollar. Los casos se seleccionan de forma que ejerciten el mayor número de atributos de cada clase de equivalencia a la vez.
Para aplicar esta técnica de prueba se tienen en cuenta los siguientes pasos:

1- Identificar las clases de equivalencias válidas e inválidas lo cual se hace tomando cada condición de entrada y aplicándole las directrices antes expuestas figura 1.6.

Condición externa	Clases de equivalencia válidas	Clases de equivalencia Inválidas
Condición de entrada	Clases válidas para esa condición de entrada	Clases inválidas para esa condición de entrada

Figura 1.6 - Características de los grafos.

2- Teniendo en cuenta las siguientes reglas:

- Si una condición de entrada especifica un rango, entonces se identifica una clase de equivalencia válida y dos inválidas.
- Si una condición de entrada especifica la cantidad de valores, identificar una clase de equivalencia válida y dos inválidas.
- Si una condición de entrada especifica un conjunto de valores de entrada y existen razones para creer que el programa trata en forma diferente a cada uno de ellos, identificar una clase válida para cada uno de ellos y una clase inválida.
- Si una condición de entrada especifica una situación de tipo “debe ser”, identificar una clase válida y una inválida.
- Si existe una razón para creer que el programa no trata de forma idéntica ciertos elementos pertenecientes a una clase, dividirla en clases de equivalencia menores.

3- Identificar los casos de pruebas, con un identificador único a cada clase de equivalencia.

4- Definir los casos teniendo en cuenta lo siguiente:

a. Escribir un nuevo caso de prueba que cubra tantas clases de equivalencia válidas no cubiertas como sea posible hasta que todas las clases de equivalencia hayan sido cubiertas por casos de prueba.
b. Escribir un nuevo caso de prueba que cubra una y solo una clase de equivalencia inválida hasta que todas las clases de equivalencias inválidas hayan sido cubiertas por casos de pruebas.

1.1.2.2 Análisis de Valores Límites.

Los errores tienden a darse más en los límites del campo de entrada que en el centro (PRESSMAN, 2005). Por ello, se ha desarrollado el análisis de valores límites (AVL) como técnica de diseño de casos de prueba de caja negra. Esta técnica de diseño de casos de prueba que complementa a la técnica de partición equivalente. En lugar de seleccionar cualquier elemento de una clase de equivalencia, el AVL lleva a la elección de casos de prueba en los extremos de la clase. La identificación del AVL es similar en muchos aspectos a las que proporciona la partición equivalente:

Si una condición de entrada, especifica un rango determinado por los valores a y b, se deben diseñar casos de prueba para los valores a y b y para los valores justo por debajo y justo por encima de a y b.
Si una condición de entrada, especifica un número de valores, se deben desarrollar casos de prueba que ejecuten los valores máximos y mínimos. También se deben probar los valores justo por encima y justo por debajo.
Si las estructuras de datos internas tienen límites preestablecidos hay que asegurarse de diseñar un caso de prueba que ejecute la estructura de datos en sus límites.

1.2 Niveles de Prueba.

1.2.1 Prueba de Unidad.

La prueba de unidad centra el proceso de verificación en la menor unidad del diseño del software: el componente o módulo (PRESSMAN, 2005). Usando la descripción del diseño procedimental como guía, se prueban los principales caminos de control para descubrir errores dentro del módulo. Esta prueba siempre está orientada a caja blanca y este paso se puede llevar a cabo en paralelo para múltiples módulos.

Procedimientos para la prueba de unidad:
El diseño de casos de prueba de unidad comienza una vez que se ha desarrollado, revisado y

verificado en su sintaxis a nivel de fuente. Un repaso de la información del diseño proporciona directrices para establecer los casos de prueba que más probabilidades de descubrir errores en cuanto a las categorías previamente mencionadas. Cada caso de prueba debe ir acompañado de un conjunto de resultados esperados.

La prueba de unidad se hace más simple cuando se diseña un módulo con un alto grado de cohesión. Cuando un módulo solo se dirige a una función se reduce el número de casos de prueba y los errores se pueden predecir y descubrir más fácilmente.

En (PRESSMAN, 2005) podemos encontrar un consejo que expresa: Hay ocasiones en que no dispones de los recursos para hacer una prueba unitaria compleja. En esta situación, selecciona los módulos críticos y aquellos con alta complejidad ciclomática y realiza sobre ellos la prueba unitaria.

Pruebas de Unidad en el Contexto Orientado a Objetos (OO).

El software orientado a objetos cambia el concepto de unidad. El encapsulamiento dirige la definición de clases y objetos. Esto significa que las clases empaquetan los atributos y las operaciones que manipulan estos datos. En vez de módulos individuales, la menor unidad a probar es la clase u objeto encapsulado. Una clase puede tener un número de operaciones y una operación puede existir como parte de un número de clases diferentes, esto significa que la prueba de unidad cambia.

No podemos probar con detalle una operación aisladamente (la vista convencional de prueba de unidad) sino como parte de una clase.

La prueba de clases para el software OO es equivalente a la prueba de software convencional. A diferencia de la prueba de unidad de este tipo de software, el cual tiende a centrarse en el detalle algorítmico de un módulo y los datos que fluyen a lo largo de la interfaz de este, la prueba de clases para software OO esta dirigido por las operaciones encapsuladas por la clase y el estado del comportamiento de la clase.

1.2.2 Prueba de Integración.

La prueba de integración es una técnica sistemática para construir la estructura del programa mientras que al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción. El objetivo es tomar los módulos probados mediante la prueba de unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño (PRESSMAN, 2005).

Se pueden plantear desde un punto de vista estructural o funcional.
Las pruebas estructurales de integración son similares a las pruebas de caja blanca; pero trabajan a un nivel conceptual superior. Nos referimos a llamadas entre módulos. Se trata pues de identificar todos los posibles esquemas de llamadas y ejercitarlos para lograr una buena cobertura.

Las pruebas funcionales de integración son similares a las pruebas de caja negra. Aquí se trata de encontrar fallos en la respuesta de un módulo cuando su operación depende de otro(s) módulo(s). Estas pruebas se van basando más y más en la especificación de requisitos del usuario.

Pruebas de Integración en el contexto Orientado a Objeto (OO).
Debido a que el software orientado a objetos no tiene una estructura de control jerárquica, las estrategias convencionales de integración ascendentes y descendentes poseen un significado pequeño. Adicionalmente, la integración, una a una, de operaciones en una clase (el enfoque de integración convencional) es a menudo imposible debido a las interacciones directas e indirectas de los componentes que conforman la clase (BERARD, 1993).

Existen dos estrategias diferentes para pruebas de integración en sistemas orientados a objetos. (BINDER, 1994).

- Prueba basadas en hilo: integra el conjunto de clases necesarios para responder a una entrada o evento del sistema. Cada hilo se integra y prueba individualmente. Se aplica la prueba de regresión para asegurar que no ocurren efectos colaterales.
- Pruebas basadas en uso: comienza la construcción del sistema probando aquellas clases servidor. Después de probar las clases dependientes, se comprueba la próxima capa de clases, llamadas clases dependientes. Esta secuencia de capas de pruebas, de clases dependientes continúa hasta construir el sistema por completo.

1.2.2.1 Integración descendente.

La integración descendente es un planteamiento incremental a la construcción de la estructura de programas. Se integran los módulos moviéndose hacia abajo (figura1.7) por la jerarquía de control comenzando por el módulo de control principal (programa principal) (PRESSMAN, 2005).

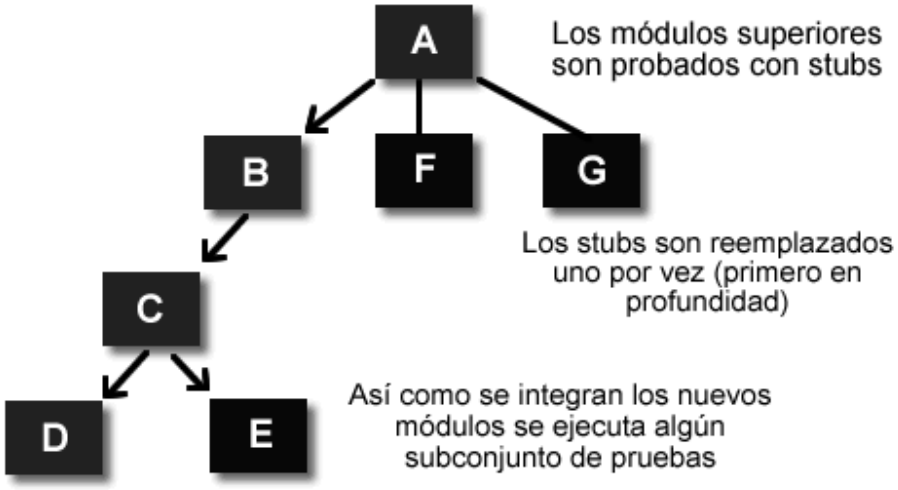


Figura 1.7 Integración descendente

El proceso de integración se realiza en una serie de cinco pasos:

1- Se usa el módulo de control principal como controlador de la prueba, disponiendo de resguardos para todos los módulos directamente subordinados al módulo principal.
2- Dependiendo del enfoque de integración elegido se van sustituyendo los resguardos subordinados uno a uno por los módulos reales.
3- Se llevan a cabo pruebas cada vez que se integra un nuevo módulo.
4- Tras terminar cada conjunto de pruebas se reemplaza otro resguardo con el módulo real.
5- Se hace la prueba de regresión (epígrafe 1.4.3) para asegurar que no se hayan introducido nuevos errores.

La integración descendente verifica los puntos de decisión o control principales al principio del proceso de prueba.

1.2.2.2 Integración Ascendente

La prueba de integración ascendente empieza con la construcción y la prueba de los módulos de los niveles más bajos de la estructura del programa (figura 1.8). Dado que los módulos se integran de abajo hacia arriba, el proceso requerido de los módulos subordinados a un nivel dado siempre está disponible y se elimina la necesidad de resguardo (PRESSMAN, 2005).

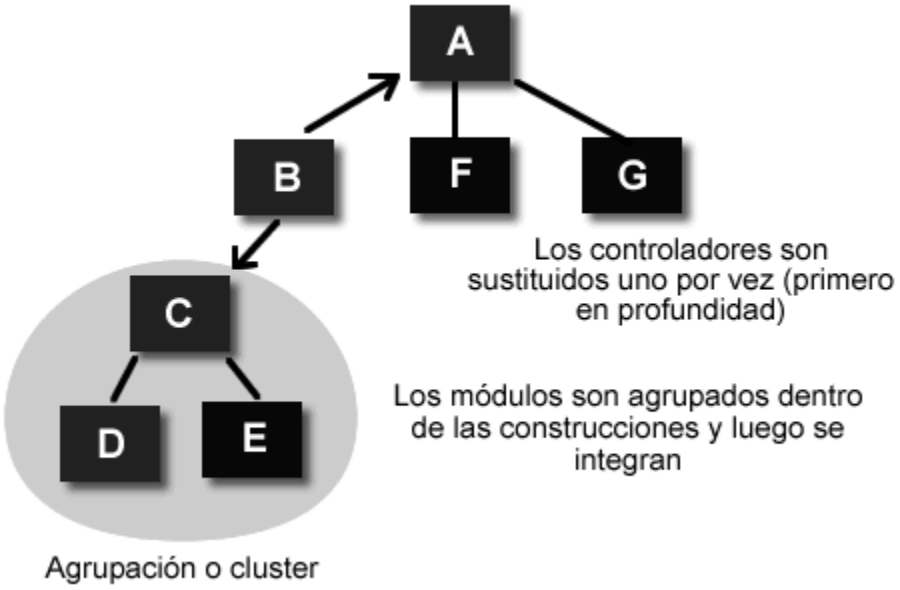


Figura1.8 Integración ascendente

Se puede implementar una estrategia de integración ascendente mediante los siguientes pasos:

1- Se combinan los módulos de bajo nivel en grupos (construcciones) que realicen una función específica del software.
2- Se escribe un controlador (programa de control de prueba) para controlar la entrada y salida de los casos de prueba.
3- Se prueba el grupo.
4- Se eliminan los controladores y se combinan los grupos moviéndose hacia arriba por la estructura del programa.

A medida que la integración progresa hacia arriba disminuye la necesidad de controladores de prueba. De hecho, si dos niveles superiores de la estructura del programa se integran de forma descendente se puede reducir el número de controladores y se simplifica la integración de grupos.

1.2.3 Prueba de regresión.

Cada vez que se añade un nuevo módulo como parte de la prueba de integración, el software cambia (PRESSMAN, 2005). En el contexto de una estrategia de prueba de integración, la prueba de regresión es la actividad que ayuda a asegurar que los cambios (debidos a las pruebas u otros motivos) no introducen comportamientos no deseados o errores adicionales.
La prueba de regresión se puede hacer manualmente, volviendo a realizar un subconjunto de todos los casos de prueba o utilizando herramientas automatizadas de reproducción de captura. Las herramientas de reproducción de captura permiten obtener casos de prueba y resultados para la subsiguiente reproducción y comparación. Las pruebas de regresión contienen tres clases diferentes de casos de prueba:

1- Una muestra representativa de pruebas que ejercite todas las funciones del software.
2- Pruebas adicionales que se centran en las funciones del software que se van a ver probablemente afectadas por el cambio.
3- Pruebas que se centran en los componentes del software que han cambiado.

A medida que avanza la prueba de integración, el número de pruebas de regresión puede crecer demasiado. Por lo que las pruebas de regresión deben diseñarse para incluir solo aquellas pruebas que traten una o más clases de errores en cada una de las funciones principales del programa.

1.2.4 Prueba de aceptación.

Prueba de aceptación del usuario es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el software fue construido.

La experiencia muestra que aún después del más cuidadoso proceso de pruebas por parte del desarrollador, quedan una serie de errores que sólo aparecen cuando el cliente se pone a usarlo. Por esta razón los desarrolladores ejercitan unas técnicas denominadas” prueba *alfa*” y “prueba *beta*” que consisten en:

- La prueba *alfa* se lleva a cabo en el lugar de desarrollo pero por un cliente. Se usa el software de forma natural pero con el desarrollador como espectador del usuario y registrando errores y problemas de uso. Y se llevan a cabo en entornos controlados.

- La prueba *beta* es realizada por los usuarios finales en sus puestos de trabajo, normalmente el desarrollador no está presente. Así la prueba beta es una aplicación en vivo del software en un entorno que no puede ser controlado por el desarrollador. El cliente registra todos los problemas que encuentra durante la prueba e informa al desarrollador. Como resultado de los problemas informados el desarrollador lleva a cabo modificaciones y prepara una nueva versión del producto para toda clase de clientes.

1.2.5 Prueba del sistema.

Estas pruebas no las realiza únicamente el desarrollador del software, sin embargo, los pasos dados durante el diseño y la prueba pueden mejorar considerablemente la probabilidad de éxito en la integración del software en el sistema.

Uno de los objetivos de las pruebas de sistema es verificar que el comportamiento externo del sistema de software satisface los requisitos establecidos por los clientes y futuros usuarios del mismo. (JAVIER, 2006)

Algunos pasos para realizar pruebas de sistema:

- Diseñar camino de manejo de errores que prueben toda la información procedente de todos los elementos del sistema.
- Llevar a cabo una serie de pruebas que simulen la presencia de datos en mal estado o de otros posibles errores en la interfaz del software.
- Registrar los resultados de las pruebas como evidencia en el caso de que se le señale con el dedo.
- Participar en la planificación y el diseño de pruebas en el sistema para asegurarse de que el software se prueba correctamente.

Aunque cada prueba tiene un propósito diferente, todos trabajan para verificar que se han integrado correctamente todos los elementos del sistema y que realizan las funciones apropiadas. A continuación tratamos los diferentes tipos de pruebas del sistema (BEIZER, 1984).

Tipos de pruebas del Sistema:

- *Prueba de Recuperación*: Es una prueba del sistema que fuerza el fallo del software de muchas formas y verifica que la recuperación se lleva a cabo apropiadamente.
- *Prueba de Seguridad:* Asegurar que los datos o el sistema solamente, es accedido por los actores deseados.
- *Prueba de Resistencia:* Están diseñadas para enfrentar a los programas con situaciones anormales. Se prueba a que potencia se puede poner a funcionar antes que falle.
- *Prueba de Rendimiento:* Está diseñada para probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado. Dentro de esta prueba encontramos otras como:
  - Benchmark: es un tipo de prueba que compara el rendimiento de un elemento nuevo o desconocido a uno de carga de trabajo de referencia conocido.
  - Contención: Enfocada a la validación de las habilidades del elemento a probar para manejar aceptablemente la demanda de múltiples actores sobre un mismo recurso (registro de recursos, memoria, etc.)
  - Carga: Usada para validar y valorar la aceptabilidad de los límites operacionales de un sistema bajo carga de trabajo variable, mientras el sistema bajo prueba permanece constante. La variación en carga es simular la carga de trabajo promedio y con picos que ocurre dentro de tolerancias operacionales normales.

Otros tipos de pruebas de sistema son:

Funcionalidad

Función: Prueba fijando su atención en la validación de las funciones, métodos, servicios y casos de uso.

Volumen: Enfocada a verificar las habilidades de los programas para manejar grandes cantidades de datos, tanto como entrada, salida o residente en la BD.

Usabilidad

Usabilidad: Prueba enfocada a factores humanos, estéticos, consistencia en la interfaz de usuario, ayuda sensitiva al contexto y en línea asistente, documentación de usuarios y materiales de entrenamiento.

Fiabilidad

Integridad: Enfocada a la valoración de la robustez (resistencia a fallos).

Estructura: Enfocada a la valoración, adherencia y a su diseño y formación. Este tipo de prueba es hecho a las aplicaciones Web asegurando que todos los enlaces están conectados, que el contenido deseado es mostrado, y no hay contenido huérfano.

Soportabilidad

Configuración: Enfocada a asegurar que funciona en diferentes configuraciones de hardware y software. Esta prueba es implementada también como prueba de rendimiento del sistema.

1.3 Estado actual de las pruebas en la UCI.

Con el objetivo de conocer y profundizar sobre el estado de las pruebas de Software, como parte importante en la producción de los mismos y la inmersión de nuestra Universidad en tal tarea, tanto en el mercado nacional como internacional se llevó a cabo entrevistas al personal implicado en las pruebas y a una serie de proyectos que se llevan a cabo en nuestra universidad, con el objetivo de conocer el estado actual de las pruebas, la plantilla utilizada se encuentra en el anexo 2.

Los resultados de las entrevistas realizadas arrojaron los siguientes datos:

El 60% de los proyectos tienen definido un plan de prueba y con igual por ciento están los que usan algún estándar para las pruebas. El 70% de los planes de pruebas son revisados con anterioridad por los responsables de su ejecución y luego por el equipo de pruebas, en iguales condiciones se encuentran los que llevan a cabo alguna estrategia, donde se especifican los objetivos de dichas pruebas.

Solo el 50% lleva en paralelo el proceso de planificación y ejecución de pruebas en el desarrollo de software, donde la metodología más utilizada es RUP y la figura 1.9 que mostramos a continuación nos brinda el estado de la utilización de los artefactos que deben utilizarse en cada fase.

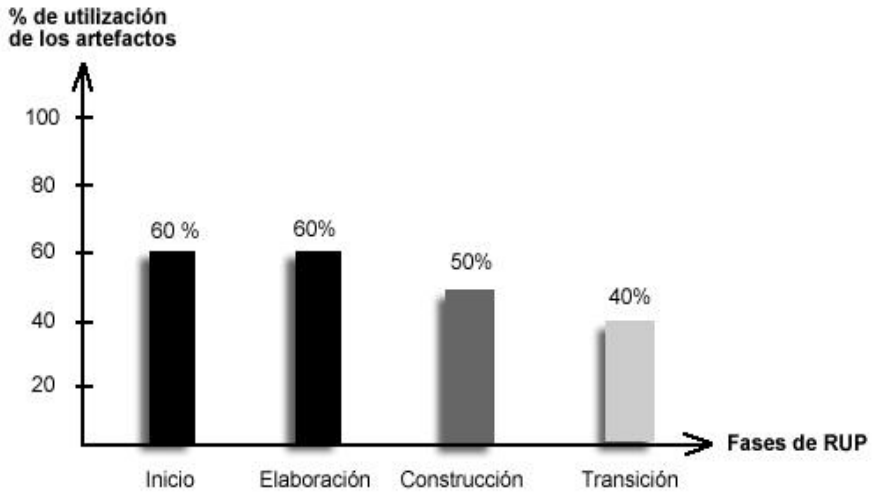


Figura 1.9 Utilización de artefactos en las fases del desarrollo.

La aplicación de las técnicas de PSP es muy pobre, pues solo un 30% de los integrantes del equipo planifica las actividades que realiza y un 20% lleva el registro de tiempo por el cual se rige y el 100% de los defectos encontrados son reportados. Las de TSP son más utilizadas pero todavía falta mucho pues el 90% de los equipos define un plan con anterioridad pero solo el 50%, distribuye correctamente el personal que va a ocupar los roles y analizan los riesgos y un 80% documenta los resultados obtenidos cuando es terminado el esfuerzo de pruebas la siguiente grafica (figura 1.10) muestra el por ciento de utilización de cada técnica.

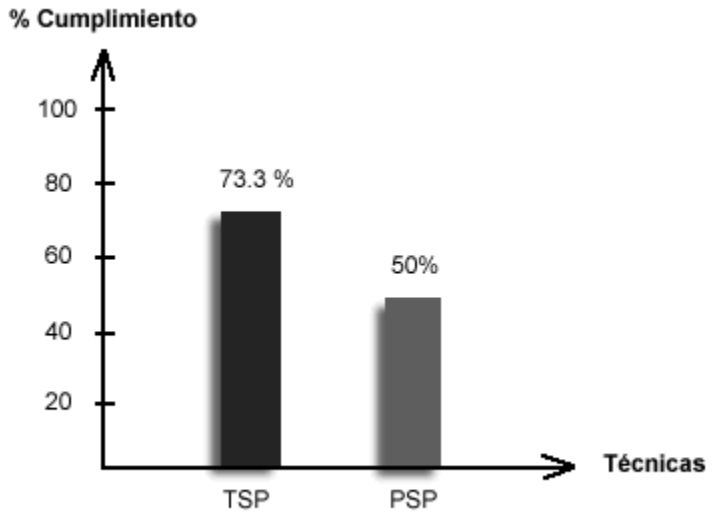


Figura 1.10. Buenas prácticas.

En muchos de los proyectos no existen personas que cumplan con el rol de probadores, lo que significa que en la etapa que le corresponde probar, el sistema no cumple con lo establecido para ello. Por lo general las pruebas de Caja Blanca no se le aplican a los sistemas y en los casos que se realicen son llevadas a cabo por los mismos desarrolladores. Lo mismo ocurre con la prueba de Caja Negra aunque esta sí es mucho más utilizada la figura 1.11 nos muestra el por ciento de aplicación de estos métodos de pruebas y la figura 1.12 muestra los tipos de pruebas que más se aplican. Esto trae consigo las pruebas no se documenten como es debido y nos enfrentemos a programas que a menudo poseen bajos índices de calidad.

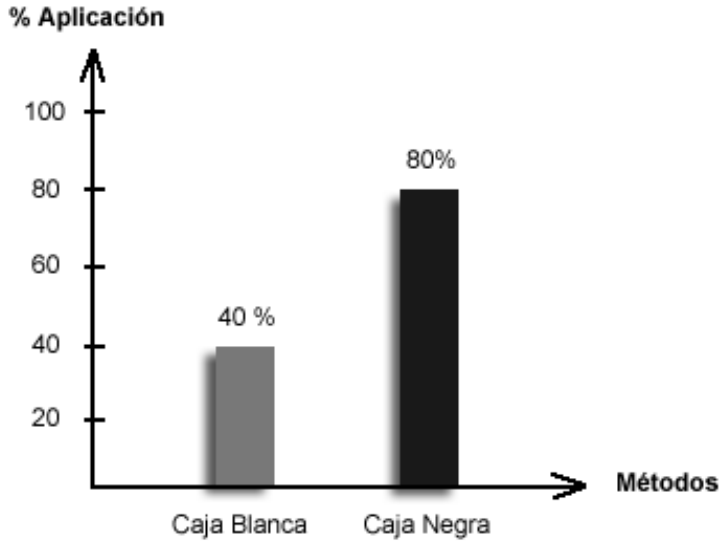


Figura 1.11. Métodos de Pruebas.

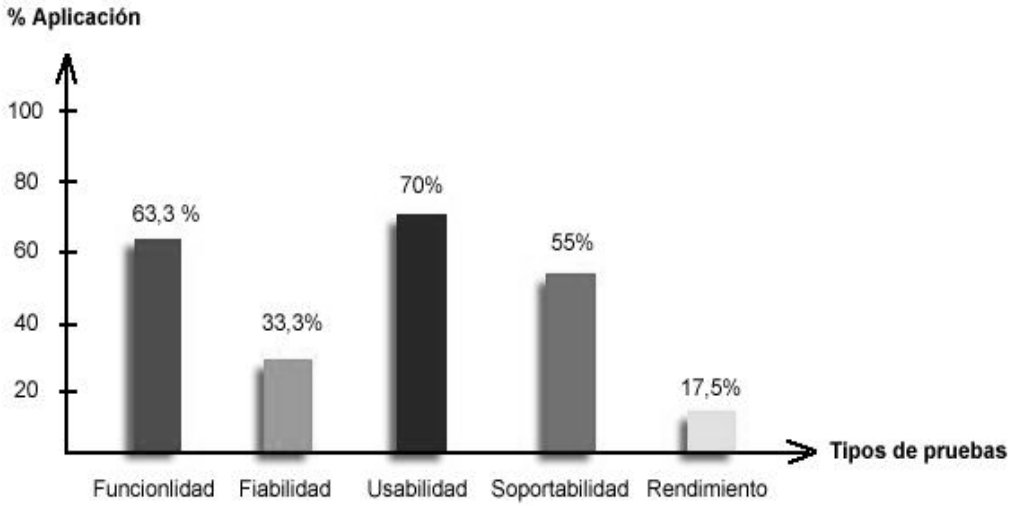


Figura 1.12. Tipos de Pruebas.

En los niveles de pruebas, se realizan pruebas que ofrecen un gran potencial para encontrar errores y lograr una mayor implementación e ejecución de estas, la figura 1.13 muestra la realización de las mismas en cada uno de los niveles.

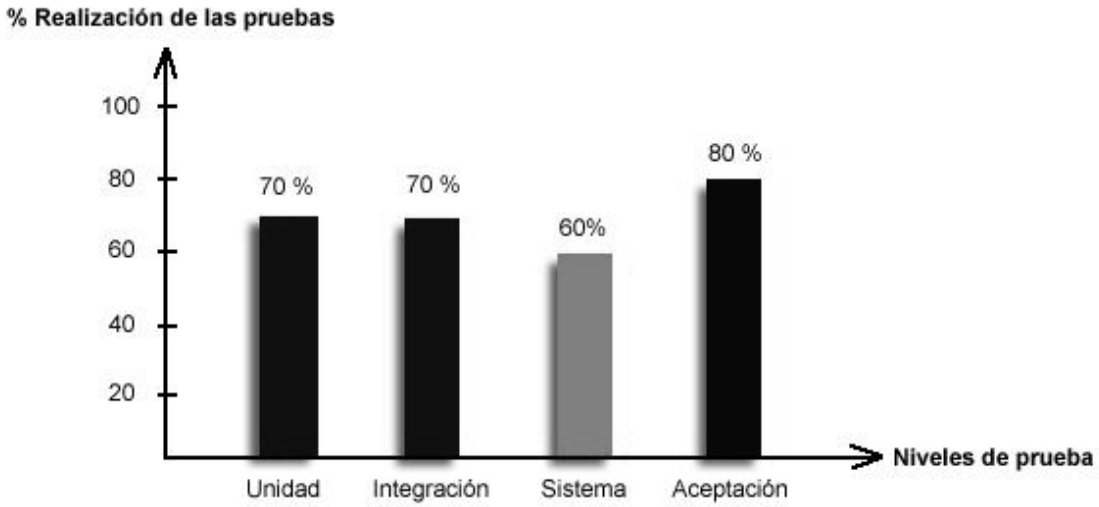


Figura 1.13. Niveles de Prueba.

El uso de herramientas automatizadas que agilicen y faciliten la realización de dichas pruebas, es muy pobre, son muy poco los proyectos en nuestra universidad que hacen suyo el uso de tan importantes herramientas para ahorrar tiempo y esfuerzo.

1.4 Proceso Unificado de Desarrollo.

El proceso de desarrollo Rational Unified Process (RUP), que será el utilizado como hilo fundamental para llevar a cabo nuestra propuesta, cuenta con un varias disciplinas, dentro de las cuales se encuentra el flujo de trabajo de pruebas, que define los principales pasos que deben llevarse a cabo durante todo el proceso, además de una descripción de los roles implicados, los principales artefactos y las herramientas de la suite de Rational que mejoran el desarrollo de las pruebas, este estudio se basa en la ayuda del Rational (RATIONAL, 2003).

1.4.1 Fases de RUP y flujo de pruebas.

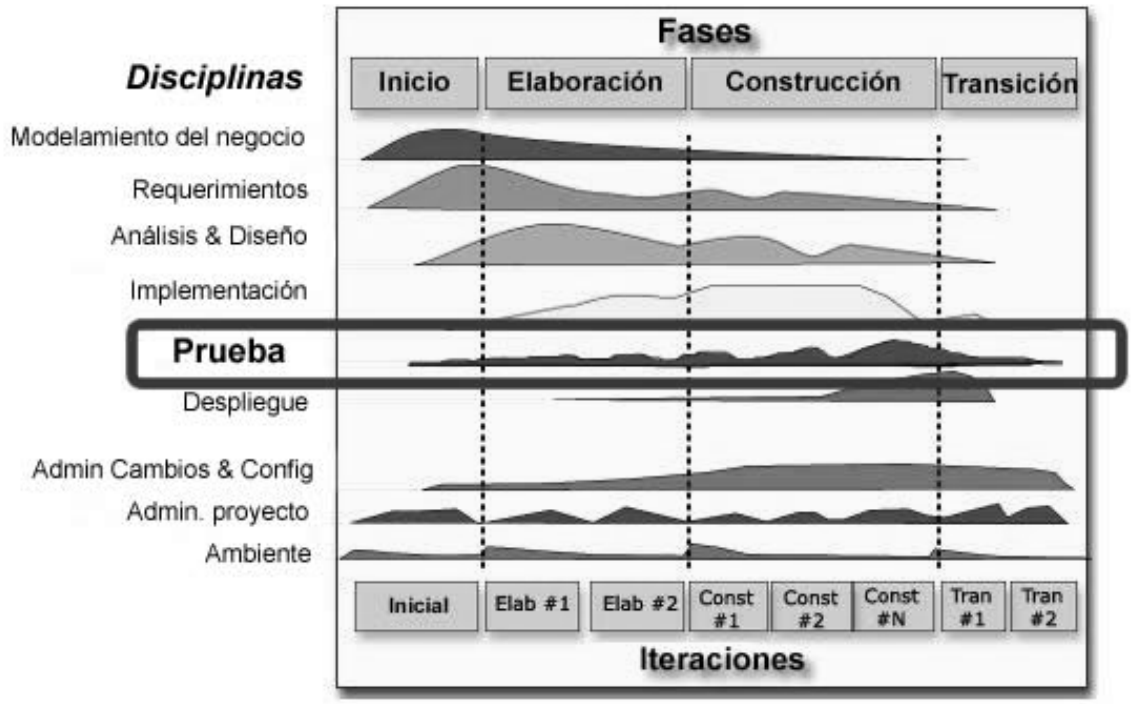


Figura 1.14 Ciclo de vida del desarrollo de software (RUP).

Durante las cuatro fases del desarrollo de software que propone RUP, se realiza un esfuerzo de prueba, que varía su complejidad a medida que avanza el proyecto (Figura 1.14).

Inicio:

Los ingenieros de pruebas se van poniendo al corriente de la naturaleza general del sistema, van considerando que pruebas requerirá y van desarrollando algunos planes provisionales de prueba. Se seleccionan el personal que ocupará los roles que van a participar en el desarrollo de las pruebas.

Se puede generar el plan de prueba general, que a partir de este momento comienza a alimentarse de todo el ciclo de desarrollo y un modelo inicial de pruebas.

Elaboración:

El objetivo es asegurarse de que los subsistemas de todos los niveles (subsistemas de servicio y subsistemas del diseño) y de todas las capas (desde la capa del sistema hasta las capas específicas de la aplicación) funcionen. Se prueban los casos de pruebas arquitectónicamente significativos. Se obtienen los casos de pruebas a partir de artefactos resultantes de otros flujos (Modelo de Casos de Uso, Modelo de Despliegue, Modelo de Implementación, entre otros). Sólo se pueden probar los componentes ejecutables. Se determina cómo probar los requisitos en el conjunto de construcciones para lo cual se preparan casos y procedimientos de pruebas.

Al empezar por las capas más bajas de la arquitectura se prueban los mecanismos de distribución, almacenamiento, recuperación (persistencia) y concurrencias de objetos, así como otros mecanismos de las capas inferiores del sistema. Con esto no solo se prueba la funcionalidad sino también el rendimiento. Todas las capas no son necesarias de probar, sino es necesario probar cómo las capas superiores hacen uso de las inferiores. Se seleccionan los objetivos que evaluarán la línea base de la arquitectura. Al diseñar las pruebas se toman como base estos objetivos para identificar los casos de pruebas necesarios y se preparan procedimientos de pruebas para comprobar la sucesiva integración de subsistemas.

Construcción:

En esta fase las pruebas son una actividad fundamental. Al planificar las pruebas los ingenieros de pruebas seleccionarán los objetivos que comprueben las sucesivas construcciones, y por último el propio sistema.

Se realizan las pruebas de unitarias y de integración, informando los resultados para tomar las medidas necesarias en casos de errores. Se realizan también pruebas del sistema al alcanzarse el status de versión parcial del sistema, informando los resultados para tomar las medidas necesarias en casos de errores. En esta fase se deben evaluar las pruebas a medida que transcurren las pruebas de integración y del sistema comprobando que éstas alcancen los objetivos del plan de pruebas. Si una prueba no alcanza sus objetivos, los casos y procedimientos de pruebas deberán ser modificados para lograrlos.

Transición:

Se pueden realizar pruebas Beta (prueba realizadas en organizaciones representativas “clientes beta”), pruebas Alfa (se realizan en la empresa que desarrolla el software; pero fuera de la organización de desarrollo) y validaciones por terceros (una empresa especializada en pruebas realiza pruebas de aceptación por encargo del cliente). Se recopilan y analizan los resultados de estas pruebas con el objetivo de llevar a cabo acciones.

En esta fase se buscan pequeñas deficiencias que pasaron desapercibidas durante la fase de construcción y que pueden ser corregidas en el marco de la línea base de la arquitectura existente.

A continuación se realiza una descripción detallada de cada aspecto del flujo:

1.4.2 Roles implicados.

Los distintos roles se organizan por las responsabilidades en las actividades y el desarrollo de los artefactos de prueba en grupos lógicos. Cada rol puede ser asignado para una o varias personas y cada persona puede cubrir varios roles. El conocimiento necesario en cada momento y para cada rol varía en dependencia de los tipos de pruebas ejecutadas y las fases del ciclo de vida del proyecto.

Jefe o administrador de prueba.

Es el responsable de las principales actividades de la prueba, quien dirige el comportamiento de estas y define los tipos de pruebas necesarias, además de manejar los resultados de estas. Es por ello que este debe ser seleccionado por sus aptitudes y capacidad para realizar esta tarea, ya que será la cabeza de la prueba.

Actividades o tareas que realiza:

- Trabajar acorde a la misión.
- Identificar los elementos que motiven o faciliten las pruebas.
- Comprometerse con las pruebas.
- Evaluar y auditar la calidad.
- Evaluar y mejorar las pruebas.

Es responsable de:

- Plan de pruebas.
- Resumen de los resultados de las pruebas.
- Negociar o acordar las pruebas.

- Asegurarse de una correcta planificación y administración de recursos.
- Verificar el progreso y efectividad de las pruebas.

Modifica:

- Lista de defectos.
- Solicitudes de cambio.

Analista de prueba:

El analista de prueba es responsable de identificar y definir las pruebas necesarias, monitorear el progreso y los resultados de la prueba en cada ciclo de prueba, evaluar la experiencia general adquirida y las actividades de la prueba. El rol de manera general tiene la responsabilidad de representar las necesidades de los clientes que no se relacionan directamente, o no tienen un conocimiento profundo acerca de la representación del proyecto. Este rol es considerado una especialización del administrador de prueba.

Las tareas o actividades que debe realizar son:

- Identificar los objetivos (blancos) de la prueba.
- Identificar las ideas de la prueba.
- Definir los detalles de las pruebas.
- Definir necesidades de evaluación y trazabilidad.
- Determinar los resultados de las pruebas.
- Verificar los cambios en la construcción.

Es responsable de:

- Lista de ideas de las pruebas.
- Casos de prueba.
- Probar el modelo de análisis (condiciones alternativas para simular las pruebas en varios ambientes de configuración).
- Colección y administración de datos de pruebas.
- Colección de los resultados de las pruebas.
- Evaluar los resultados de cada ciclo de prueba.

Modifica:

- Plan de pruebas.
- Resumen de los resultados de las pruebas.
- Requisitos de cambio.
- Pautas para las pruebas.

Diseñador de prueba.

Es el responsable de definir el alcance de la prueba y asegurar su correcta implementación. Identifica las técnicas, herramientas y guías de trabajo apropiadas para la implementación de la prueba, y brinda la información de los recursos necesarios para ejecutar la prueba. Este rol es llamado por arquitecto de prueba o arquitecto de automatización de prueba.

Actividades o tareas que realiza:

- Definir enfoque de las pruebas.
- Definir la configuración del ambiente de las pruebas.
- Identificar los mecanismos de pruebas.
- Estructurar la implementación de las pruebas.
- Definir los elementos de las pruebas.
- Desarrollar las pautas de las pruebas.

El rol es responsable de:

- Documento de prueba de arquitectura.
- Especificación de interfaz de prueba.
- Configuración del ambiente de prueba.
- Definir la suite de pruebas.
- Identificar y describir apropiadas técnicas de prueba.
- Identificar herramientas apropiadas de soporte.
- Definir y mantener la arquitectura de automatización de pruebas.
- Especificar y verificar las condiciones de las configuraciones de prueba.

Modifica:

- Plan de pruebas.
- Script de pruebas.

Verificador o Probador.

Es responsable de ejecutar o producir las principales pruebas del software y es quien obtiene directamente los resultados y analizar los fallos que se producen.

Debe realizar las siguientes actividades o tareas:

- Implementar pruebas.
- Implementar la suite de pruebas.
- Ejecutar la suite de pruebas.

- Analizar fallos de prueba.

Es responsable de:

- Test log (salida de ejecución de las pruebas).
- Test Script (Pasos o instrucciones para realizar la prueba).
- Identificar la implementación más apropiada para la prueba.
- Implementar pruebas individuales.
- Verificar la forma en que se ejecutan las pruebas.
- Analizar y recuperarse de los errores de ejecución.

Modifica:

- Requisitos de cambio confeccionado por el jefe de control de cambios.
- Suite de pruebas.

Roles Adicionales.

Revisor.

Es responsable de organizar el tiempo de los miembros del equipo de trabajo en las tareas y construcción de los artefactos asignados. Una persona que adopta este rol tiene la responsabilidad de coordinar todo el proceso de revisión, con el objetivo de que esta se realice de forma apropiada.

Coordinador de revisión.

Es responsable de facilitar las revisiones formales e inspecciones, y asegurarse de que estas ocurren en el momento requerido y de la forma establecida. Organiza todo el proceso de revisión.

Revisor técnico.

Contribuye a la regeneración del proceso de revisión. Realiza una revisión técnica de los artefactos del proyecto. Y principalmente cumple con seis actividades fundamentales, revisando:

- Modelo de casos de uso del negocio.
- Modelo de análisis del negocio.
- Requerimientos.
- Arquitectura.
- Diseño.
- Código.

Tareas comunes.

La persona que ejecuta cualquiera de los roles que se describen en RUP puede, con los permisos o privilegios adecuados, realizar check-in y check-out de artefactos relacionados con el producto para el mantenimiento en el sistema de control de configuración. Puede actualizar la solicitud de cambios dentro de las reglas establecidas por el proyecto.

Realiza las siguientes tareas o actividades:

- Crear desarrollo del espacio de trabajo.
- Hacer cambios.
- Entregar cambios.
- Actualizar espacio de trabajo (repositorio).
- Proponer solicitud de cambios.
- Actualizar solicitud de cambios.

1.4.3 Flujo de trabajo de prueba de RUP.

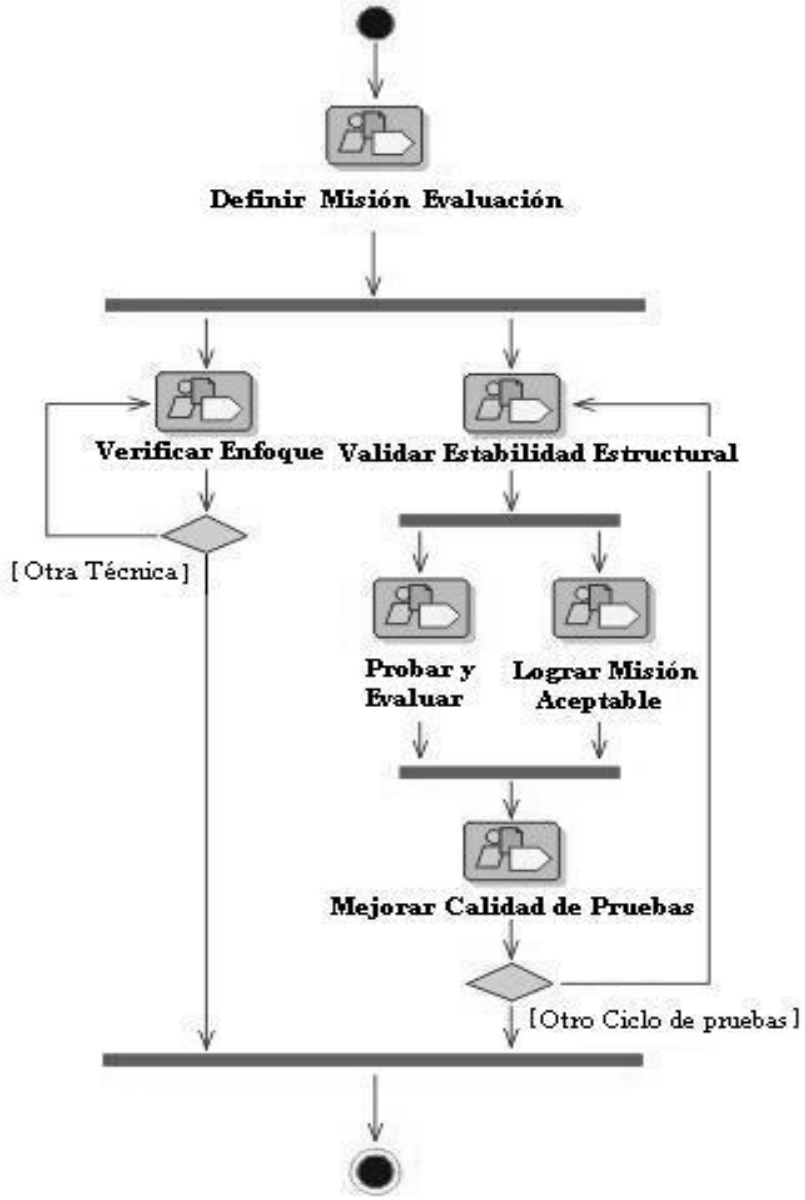


Figura 1.15 Flujo de trabajo de pruebas (RUP).

La prueba se alinea con la iteración que el resto del equipo de desarrollo sigue. Generalmente, cada iteración contiene al menos un ciclo de prueba (figura 1.16).

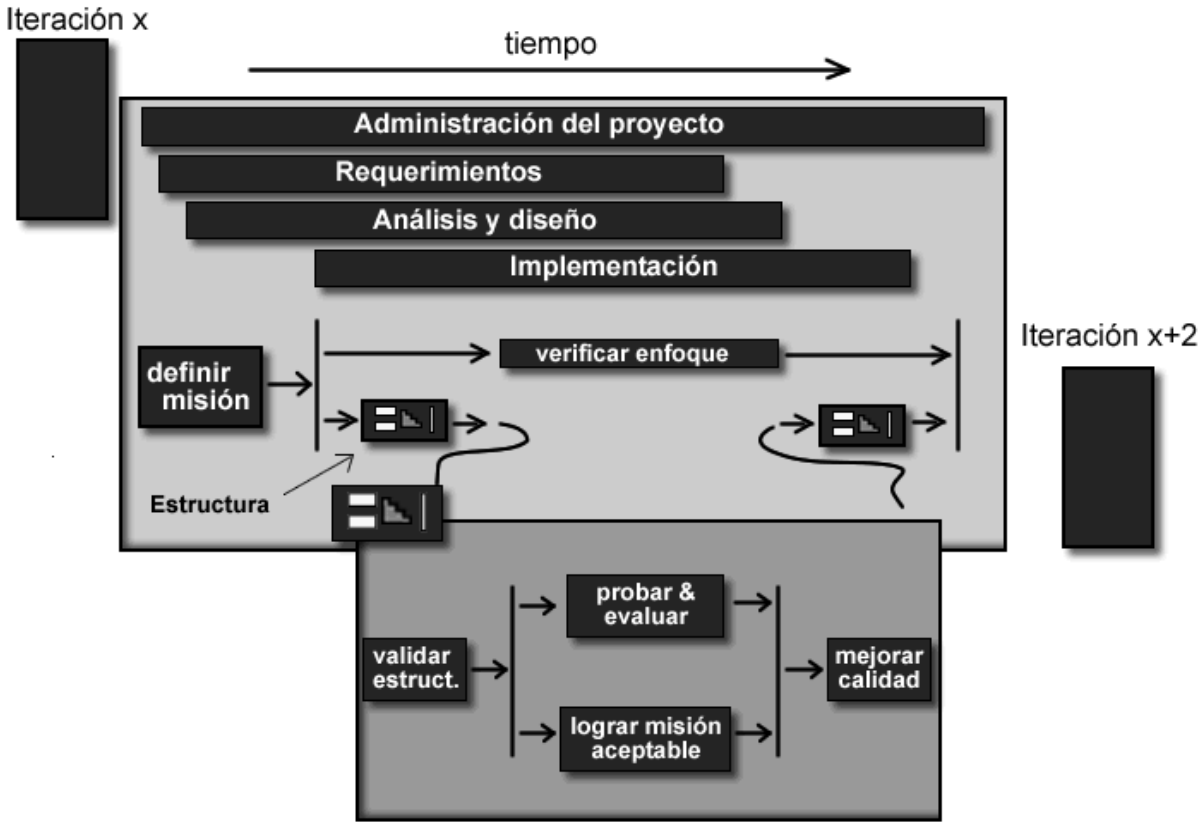


Figura 1.16 Ciclo de vida e iteraciones de RUP

La iteración comienza con una investigación por el equipo de prueba, quienes negocian con el administrador de proyecto, y otros stakeholders en lo que se refiere a las pruebas más útiles para emprender en las iteraciones (*Definir la misión de la evaluación, figura 2.4*)

Un subconjunto de los miembros del equipo de prueba puede estar investigando nuevas técnicas de pruebas. Este esfuerzo intenta probar qué técnicas son factibles de usar. Así el equipo de prueba puede contar con estas técnicas en iteraciones posteriores (*Verificar el enfoque de prueba, figura 2.5*).

Para cada estructura (build) construida por el equipo de desarrollo, se realizan los cuatros siguientes flujos de trabajo de la disciplina de prueba. Se valida que la estructura es lo suficiente estable para comenzar la evaluación (*Validar la estabilidad estructural, figura 2.6).* Se realiza la evaluación (*Probar y evaluar, figura 2.7*). A medida que se está probando y evaluando, se entrega un resultado útil de la evaluación de las pruebas para los stakeholder, este resultado está determinado en términos de la misión de evaluación (*Lograr la misión aceptable, figura 2.8*).

Se mejora la calidad de las pruebas realizadas teniendo en cuenta que pueden seguir usándose en otras iteraciones (*Mejorar la calidad de las pruebas, figura 2.9*).

1.4.4 Descripción de las actividades fundamentales.

Definir la misión de evaluación. Su función fundamental es identificar el método apropiado de la prueba para la iteración y estar de acuerdo con los stakeholder de las metas correspondientes que dirigirán las pruebas.

A partir del plan de iteración y el plan de desarrollo de software se define el plan de prueba, se identifican los objetivos y la estrategia prueba. Se define además cómo monitorear y evaluar el progreso.

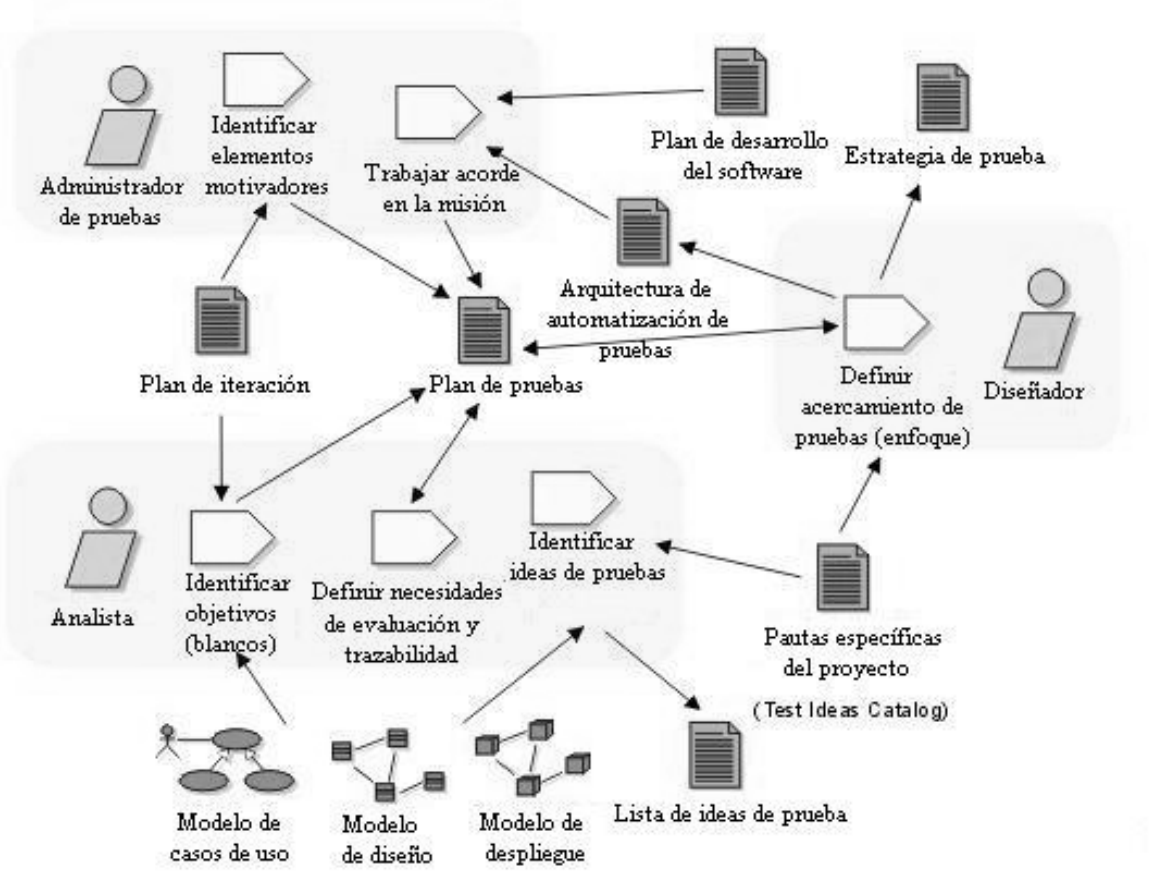


Figura 1.17 Definir la misión de evaluación (RUP).

Verificar el enfoque de prueba. Su función fundamental es demostrar que diferentes técnicas facilitarían la prueba planificada, es verificar demostrando que el enfoque trabajará, producirá resultados precisos y es apropiado para los recursos disponibles.

El objetivo es lograr un entendimiento de las restricciones y limitaciones de cada técnica al aplicarlas en un contexto del proyecto dado: encontrar una solución de implementación apropiada para cada técnica o encontrar técnicas alternativas que puedan usarse. Esto ayuda a mitigar el riesgo de descubrir muy tarde técnicas que no son factibles aplicarlas en el proyecto.

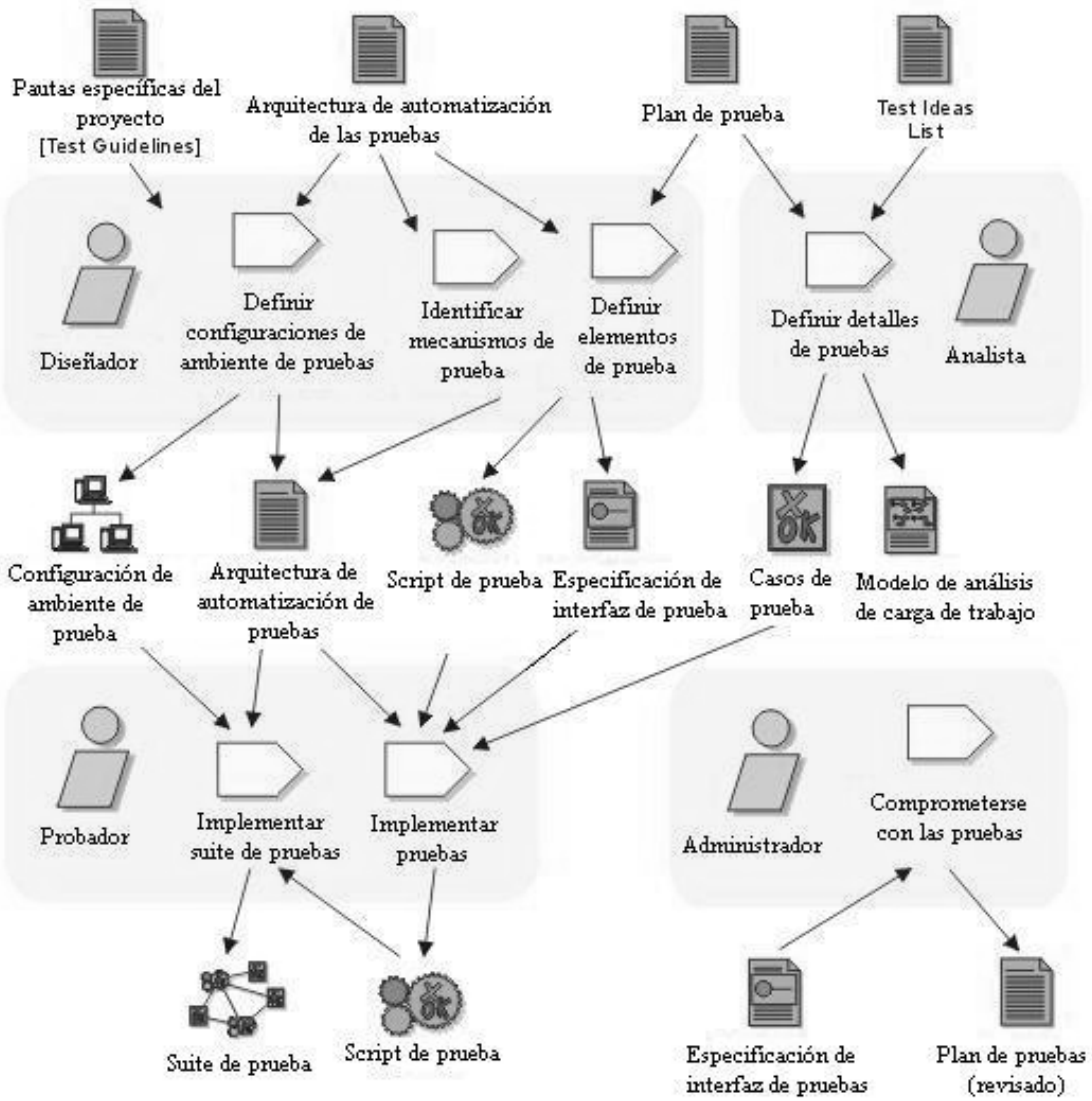


Figura 1.18 Verificar enfoque de prueba (RUP).

Validar estabilidad estructural. Su función fundamental es validar que la estructura (build) es suficientemente estable para la prueba detallada y para comenzar la evaluación. Es importante definir los detalles de la prueba, implementarla, ejecutarla, analizar los fallos y determinar los resultados de la prueba.

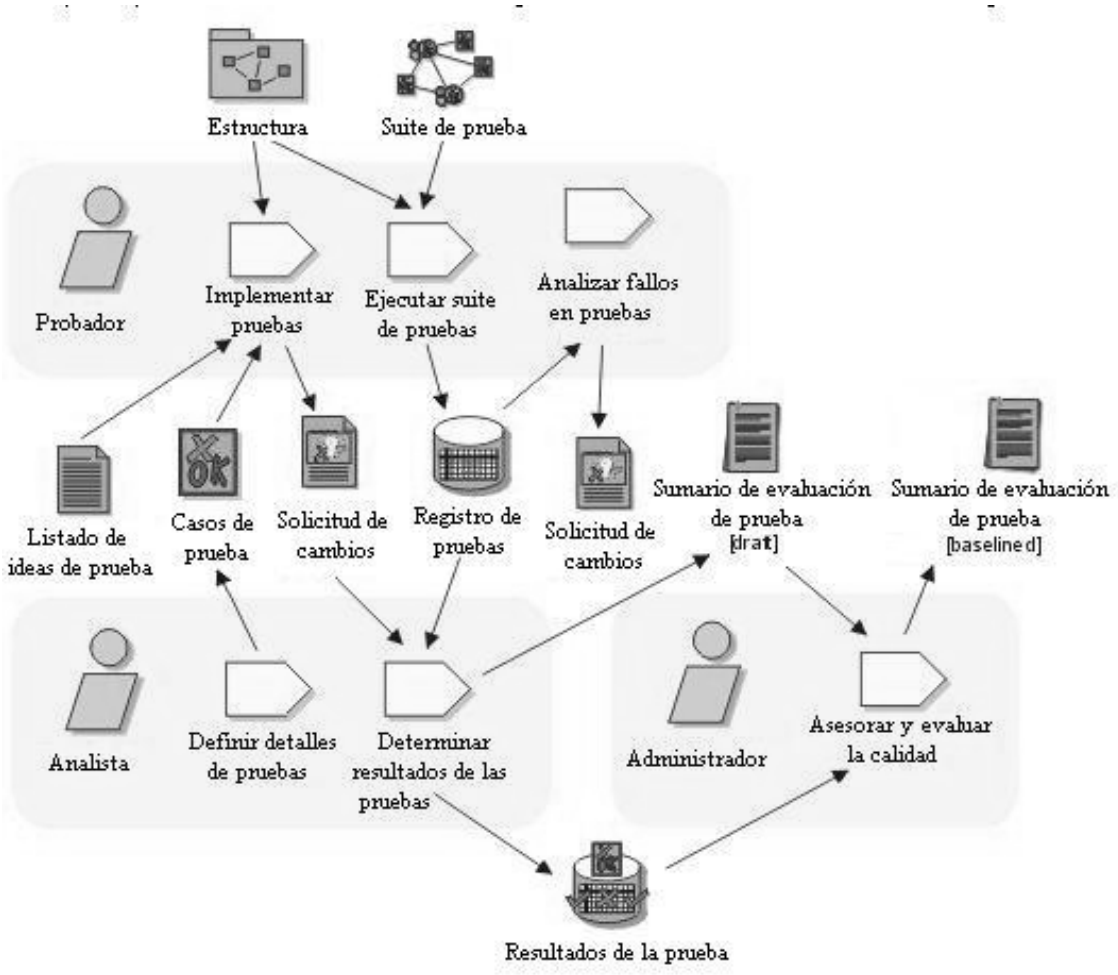


Figura 1.19 Validar estabilidad estructural (RUP).

Probar y evaluar. Realizar pruebas adecuadas a lo ancho y en profundidad para permitir una evaluación suficiente de los elementos que son objetivo de las pruebas. Esta evaluación suficiente es dirigida por la misión de evaluación actual y por los elementos motivadores de prueba.

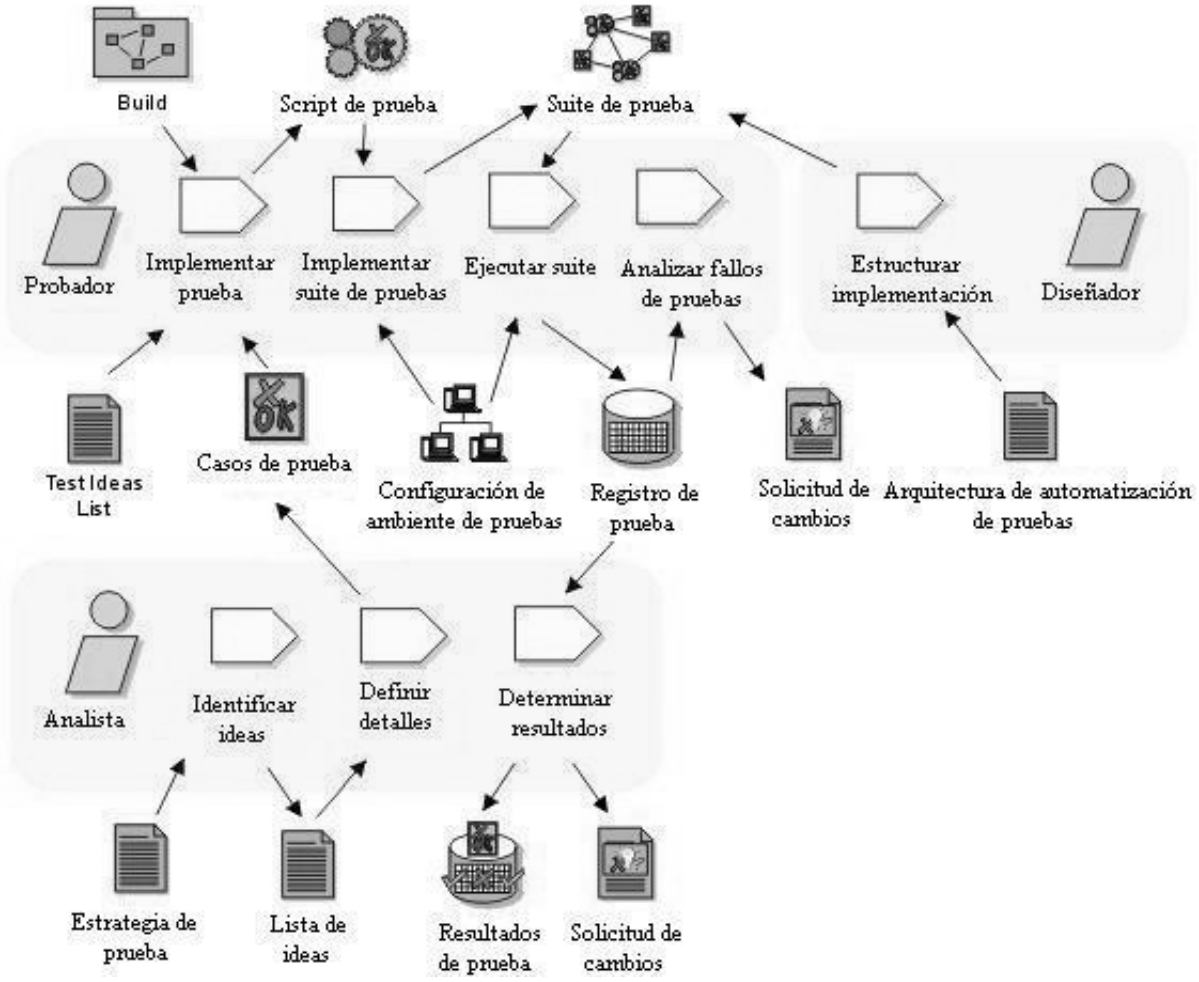


Figura 1.20 Probar y evaluar (RUP).

Lograr la misión aceptable. Su función fundamental es entregar un resultado útil de la evaluación de las pruebas para los stakeholder, este resultado se obtiene en términos de la misión de evaluación. En la mayoría de los casos significaría enfocar el esfuerzo en ayudar al equipo de proyecto en lograr los objetivos del plan de iteración que se aplica al ciclo de prueba actual.

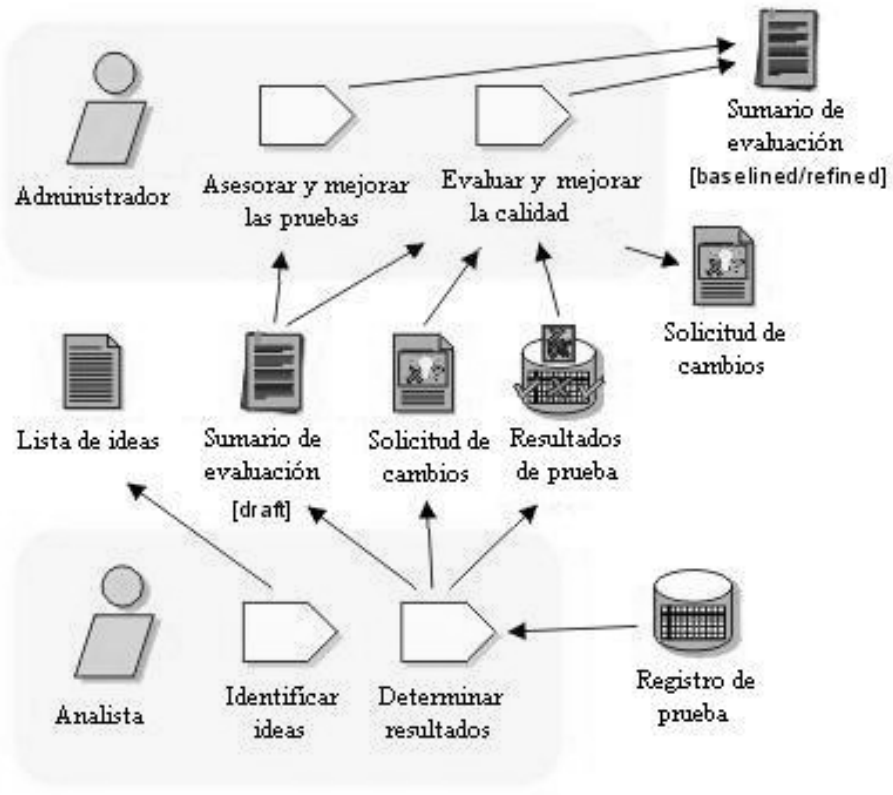


Figura 1.21 Lograr misión aceptable (RUP).

Mejorar la calidad de las pruebas. Su función fundamental es mantener y mejorar la calidad de las pruebas. Esto es especialmente importante si la intención es re-usar las ventajas desarrolladas en el actual ciclo de prueba en ciclos de pruebas posteriores.

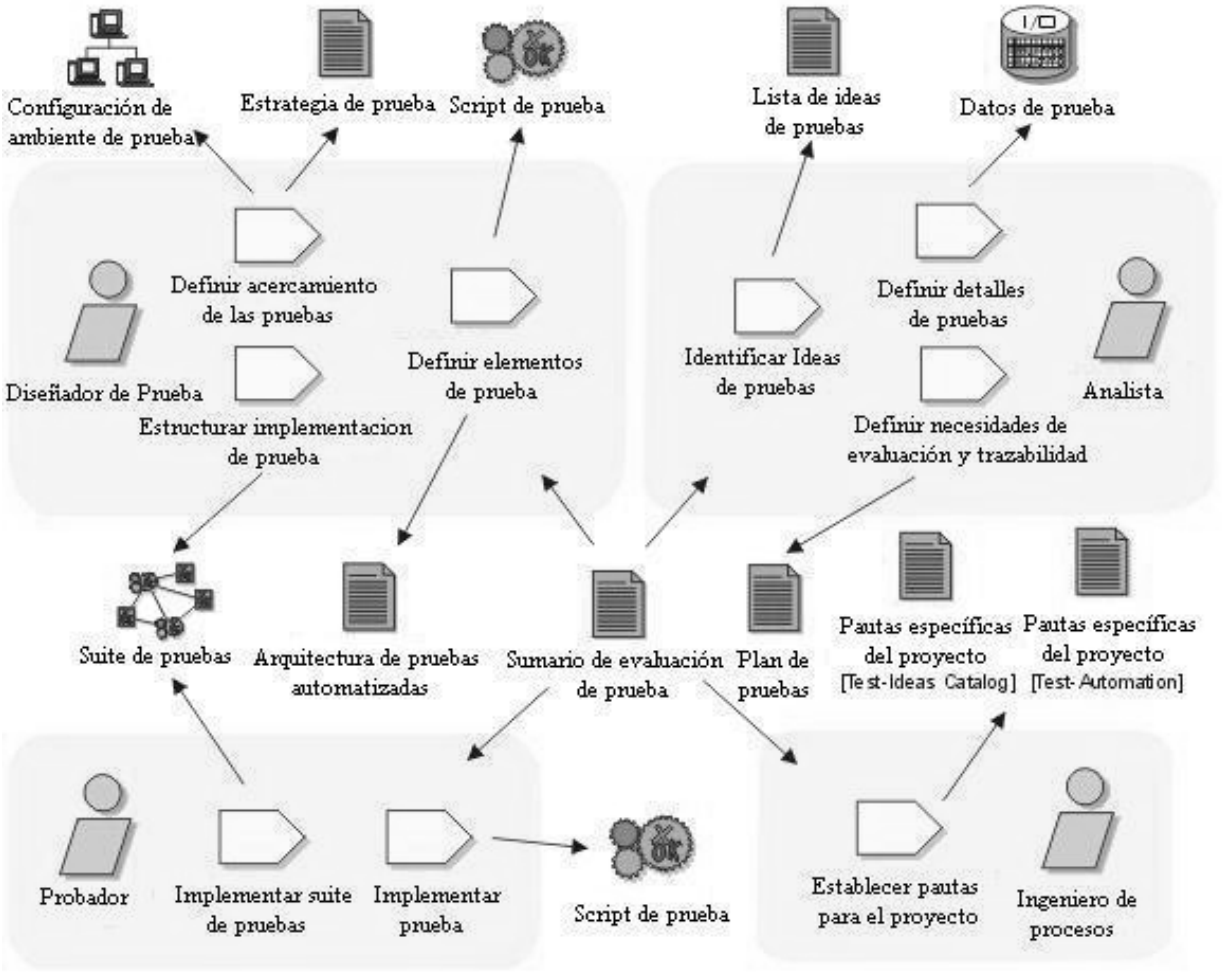


Figura 1.22 Mejorar la calidad de las pruebas (RUP).

Cada actividad incluye una serie de tareas o actividades que se describen en el anexo 4.

1.5 PSP.

El Proceso de Software Personal (Personal Software Process, o PSP) constituye un conjunto de métodos y habilidades que los ingenieros de software han desarrollado teniendo como objetivo principal garantizar que cada persona implicada en un proyecto planifique debidamente todas las actividades que deberá realizar, y por consiguiente, que haga bien su trabajo. (HUMPHREY, 2001)

Como se ha visto en el capítulo anterior, las pruebas se realizan para detectar errores, que de una forma son producidos por las personas que realizan el proyecto. Si cada una de ellas es capaz de revisar su trabajo, llevar a cabo un control de sus errores y corregirlos, entonces la etapa de prueba requiere de un menor tiempo y esfuerzo por parte del equipo.

Con el PSP, los ingenieros se centran en hacer programas limpios y libres de defectos desde el principio. La razón para la cual se crea esta estrategia, es que una vez que el ingeniero hace un producto poco sólido, no hay herramienta que pueda dejarlo libre de defectos…Así, para obtener un producto de alta calidad al final de la prueba, debes poner un producto de alta calidad al comienzo de la prueba. (HUMPHREY, 2001)

El objetivo de la introducción del PSP en el presente trabajo, es llegar a una propuesta donde se apliquen estas técnicas que consideramos tan importantes para las pruebas de software. Las técnicas que se propone vincular son:

1. Cada integrante del equipo realiza un cuaderno del ingeniero, donde se hace:
- Registro de tiempo.
- Resumen semanal de actividades.
- Control de tareas finalizadas.
- Registro de defectos.

2. El administrador de prueba realiza un diagrama de Gantt, independientemente del plan de pruebas, para definir las pautas y el control de las actividades de cada uno de sus trabajadores.

De esta manera se puede mejorar el desarrollo de las pruebas y de cada una de las actividades del ciclo. Y se puede facilitar la obtención de los resultados de las actividades del flujo de trabajo de pruebas que se propone en el capítulo 2.

1.6 TSP.

El Proceso de Software en Equipo (Team Software Process, o TSP), es un conjunto de procesos definidos y estructurados que enfatizan el balance entre procesos, producto y trabajo en equipo. Muestra cómo aplicar prácticas de ingeniería de software conocidas en ambiente de equipo. (ANTHONY LATTANZE)

Tiene como objetivo principal alcanzar un producto de alta calidad, poner a funcionar el proyecto de una forma productiva y bien manejada, y terminar en tiempo.

Es necesario que el equipo de prueba se organice, en aras de lograr una correcta distribución del esfuerzo de prueba, las relaciones de trabajo, determinación de los roles y llegar a los acuerdos necesarios para cumplir debidamente con las tareas.

Los roles deberán seleccionarse de acuerdo a los siguientes conocimientos:

Jefe o Administrador de prueba:

Conocimientos necesarios:

- Conocer los diferentes tipos y técnicas de pruebas.
- Conocimientos generales de todos los aspectos de Ingeniería de Software.
- Experiencia en variedad de tipos de pruebas, técnicas y herramientas.
- Relaciones con personas, diplomacia.
- Conocimientos de planificación y administración.
- Conocer o estudiar el producto que está bajo prueba.
- Identificar o diagnosticar los problemas y sus posibles soluciones.
- Conocer la arquitectura del sistema y la forma en que trabaja.
- Seleccionar el personal del equipo y ser exigente con el personal subordinado.

Cuando se necesita una prueba automatizada, además de las relacionadas anteriormente, debe adicionalmente tener:

- Experiencia en el uso de herramientas automatizadas.
- Conocimientos de diagnóstico y corrección de errores.

Es responsable de:

- Plan de pruebas.
- Resumen de los resultados de las pruebas.
- Negociar o acordar las pruebas.
- Asegurarse de una correcta planificación y administración de recursos.
- Verificar el progreso y efectividad de las pruebas.
- Brindar un alto nivel de calidad para la resolución de defectos.

Realiza:

- Plan de pruebas.
- Selección de roles o equipo de trabajo.
- Establecer las pautas para las pruebas.
- Verificación del plan de pruebas (general y de cada iteración).
- Determinar los resultados.
- Obtener la solicitud de cambios.

Analista:

- Conocimientos analíticos.
- Mente abierta, curiosa e investigativa.
- Atención a los detalles y tenacidad.
- Entender los fallos y errores de determinado software.
- Conocimiento del sistema o la aplicación bajo prueba.
- Experiencia en variedad de tipos de pruebas.

Diseñador:

- Experiencia en variedad de pruebas.
- Diagnóstico y habilidad en solución de problemas.
- Conocimientos de hardware y software así como su instalación.
- Experiencia en el uso de herramientas automatizadas de prueba.
- Conocimientos de programación.
- Experiencia en liderazgo de equipos de programación, así como habilidades en diseño.
- Conocer el sistema o la aplicación bajo prueba.

Probador:

- Conocer diferentes técnicas de prueba.
- Diagnosticar problemas y posibles soluciones.
- Conocimientos del sistema o la aplicación que se está probando.
- Conocimientos de redes y de arquitectura de sistema.
- Tener experiencia en herramientas automatizadas para pruebas.
- Conocimientos de programación.
- Saber diagnosticar y depurar defectos.

Una buena práctica para el logro de los objetivos de las pruebas es utilizar las siguientes técnicas que brinda el TSP:

1. Seleccionar o repartir correctamente los roles.
2. Definir una estrategia de desarrollo.
3. Documentar los resultados. (Resumen de evaluación de pruebas)

1.7 Herramientas.

Durante los últimos años numerosas empresas, instituciones y desarrolladores de software han dedicado tiempo y esfuerzo a la confección de herramientas que hacen posible administrar, controlar, optimizar y ejecutar las pruebas de manera automática.

Se hace necesario llevar a cabo un control exacto de cada uno de los pasos que se siguen durante el desarrollo de una prueba, procesar toda la información de entrada y de salida, y permitir conocer el estado de esta en cualquier momento.

En el capítulo 3 se hace referencia a algunas de las herramientas que brindan grandes utilidades para la administración, seguimiento y ejecución de las pruebas. Estas se encuentran divididas en herramientas para plataformas libres, y herramientas para software propietario.

Capítulo 2: Propuesta de Flujo de Trabajo de Prueba

Esta sección parte del estudio detallado del flujo de trabajo de pruebas que propone RUP en conjunto con las buenas prácticas de PSP y TSP, con el objetivo de lograr una estrecha vinculación entre ellos, para lograr alcanzar una propuesta de flujo de trabajo en la disciplina de prueba de software, que se pueda aplicar a proyectos de gestión.

2.1 Propuesta de flujo de trabajo de Pruebas.

A partir de lo que se ha interpretado del flujo de trabajo que propone RUP, las técnicas de PSP y TSP, que se consideran buenas prácticas dentro de la ingeniería de software, en conjunto con las habilidades desarrolladas en la universidad, llegamos a la siguiente propuesta (Figura 2.1). Esta deberá ser desarrollada por un Grupo de Prueba (GP), que forma parte del equipo de desarrollo de software y los resultados de su trabajo son muy útiles para el proceso de la garantía de la calidad.

El GP es parte del equipo del proyecto de desarrollo de software en el sentido de que se ve implicado en el proceso de especificación (Planificación y especificación de los procedimientos de prueba) a lo largo de un proyecto.

Este grupo debe constar de cuatro roles fundamentales que se definen en el TSP, el Administrador de prueba, Analista de prueba, Diseñador de prueba y Probador, los que se seleccionan según las habilidades que poseen.

El flujo cuenta con seis actividades fundamentales, que se explican detalladamente a continuación.

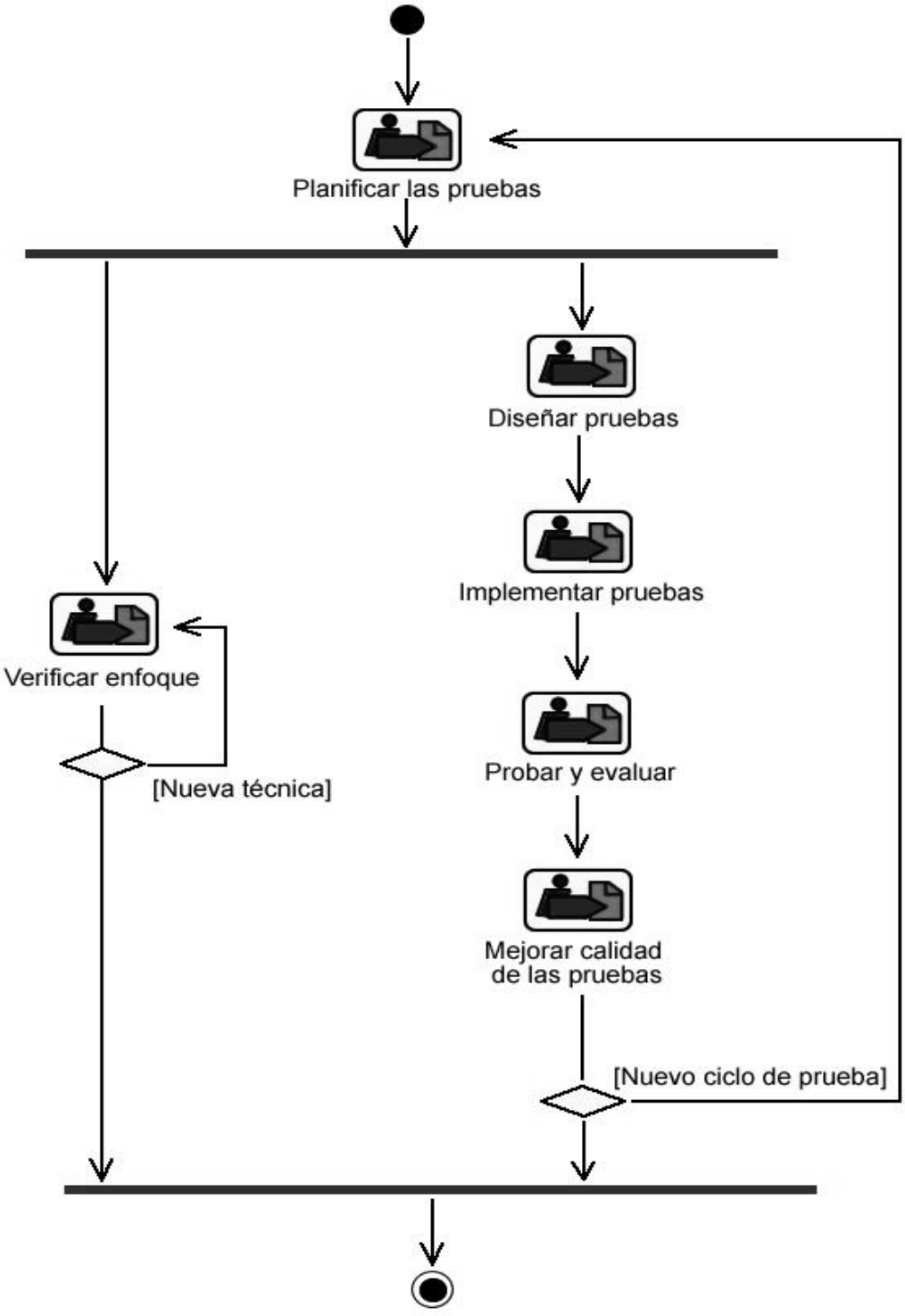


Figura 2.1 Flujo propuesto

2.2 Tareas Fundamentales.

Antes de comenzar las actividades el jefe o administrador de pruebas se reúne con el equipo de pruebas, y se realizan las siguientes actividades:

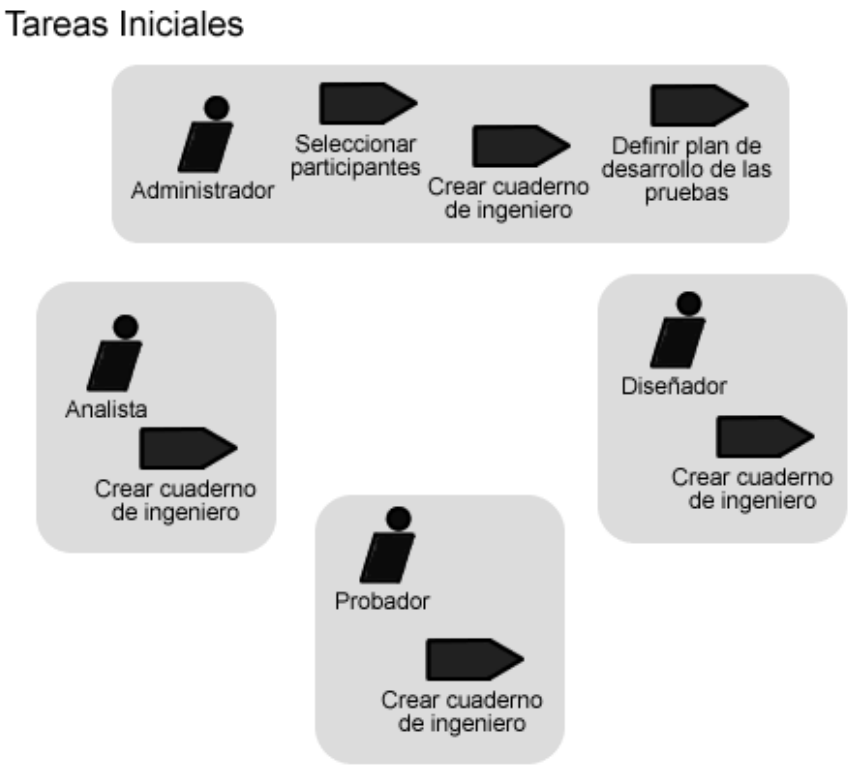


Figura 2.2 Tareas Iniciales

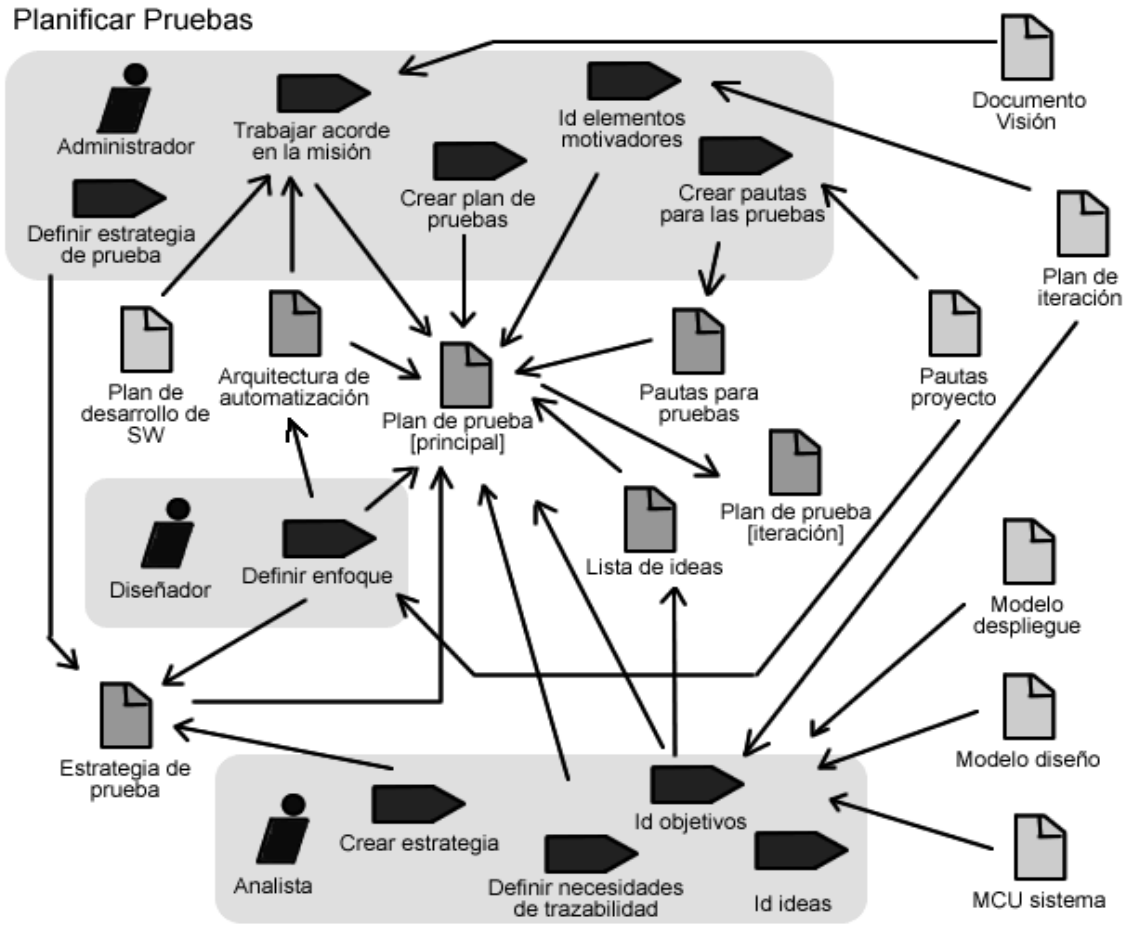


Figura 2.3 Planificar pruebas

Planificar pruebas:

Su función principal será identificar el método apropiado para ejecutar las pruebas durante todo el proyecto, aquí se define el plan de pruebas general y la estrategia general de prueba. Surge un plan de pruebas para la iteración a partir del plan general y una estrategia para la iteración, derivada de la estrategia principal y atendiendo a los objetivos específicos de la iteración. Se van a definir los objetivos de las pruebas, las pruebas posibles a realizar y las pautas a seguir durante el desarrollo del proceso.

- Trabajar acorde en la misión (Administrador)

  Se especifican los recursos necesarios para cada ciclo. Se logra un acuerdo apropiado para el cumplimiento de objetivos y confección de entregables. Todo a partir de los principales objetivos y las necesidades de aplicar pruebas en la iteración.

- Identificar elementos motivadores (Administrador).
  Se buscan elementos que faciliten y sirvan de estimulación para el trabajo con las pruebas.

- Definir enfoque de las pruebas. (Diseñador)
  Se establecen los tipos de pruebas que se van a realizar a partir de las necesidades de cada iteración. Estos constituirán una línea base para que se cumplan eficientemente los objetivos. Se seleccionan las técnicas posibles a aplicar para las pruebas.

- Definir estrategia de prueba. (Administrador)
  Se establecen las pruebas que se van a realizar durante el proceso, y a qué niveles dentro de la iteración. La estrategia propuesta se muestra en el anexo 3.

- Definir las necesidades de trazabilidad (Analista)
  Se define la relación de los elementos a probar con su entorno dentro del sistema. Qué elementos del software se afectan con cada posible cambio que se realice durante el proceso de prueba o corrección de errores.

- Crear las pautas para las pruebas (Administrador)
  Se toman las pautas del proyecto, y dentro del espacio de tiempo designado a las pruebas se realizan las pautas para el proceso de prueba.

- Crear plan de pruebas principal (Administrador).
  Se crea un plan de prueba a partir de los objetivos iniciales y generales del proyecto. A partir de este se van generando planes para cada iteración según las necesidades.

- Crear estrategia de prueba (Analista)
  Se crea a partir de la estrategia definida por el administrador.

- Identificar ideas de pruebas (Analista)
  Se identifican las ideas de las pruebas que serán exploradas para adquirir una aceptable calidad en los blancos de prueba seleccionados. Identificar un número suficiente de ideas para validar los blancos seleccionados contra los elementos que motivan las pruebas.

- Identificar objetivos (blancos). (Analista)
  Identificar los elementos individuales del sistema, tanto hardware como software, que necesitan ser probados.

Artefactos entrada:
- Plan de desarrollo de software.
- Pautas del proyecto.
- Plan de iteración.
- Modelo de casos de uso del sistema (MCU).
- Modelo de diseño.
- Modelo de despliegue.
- Arquitectura de automatización de pruebas (se genera cuando se define el enfoque).
- Documento visión.

Artefactos resultantes:
- Plan de prueba general (dentro del este se incluye: estrategia de prueba, lista de ideas, arquitectura de automatización y pautas para las pruebas).
- Plan de prueba de iteración (incluye la especificación de las tareas a realizar en cada iteración en específico.)

En las siguientes actividades cuando se hace referencia al plan de pruebas, se trata del plan de pruebas para la iteración.

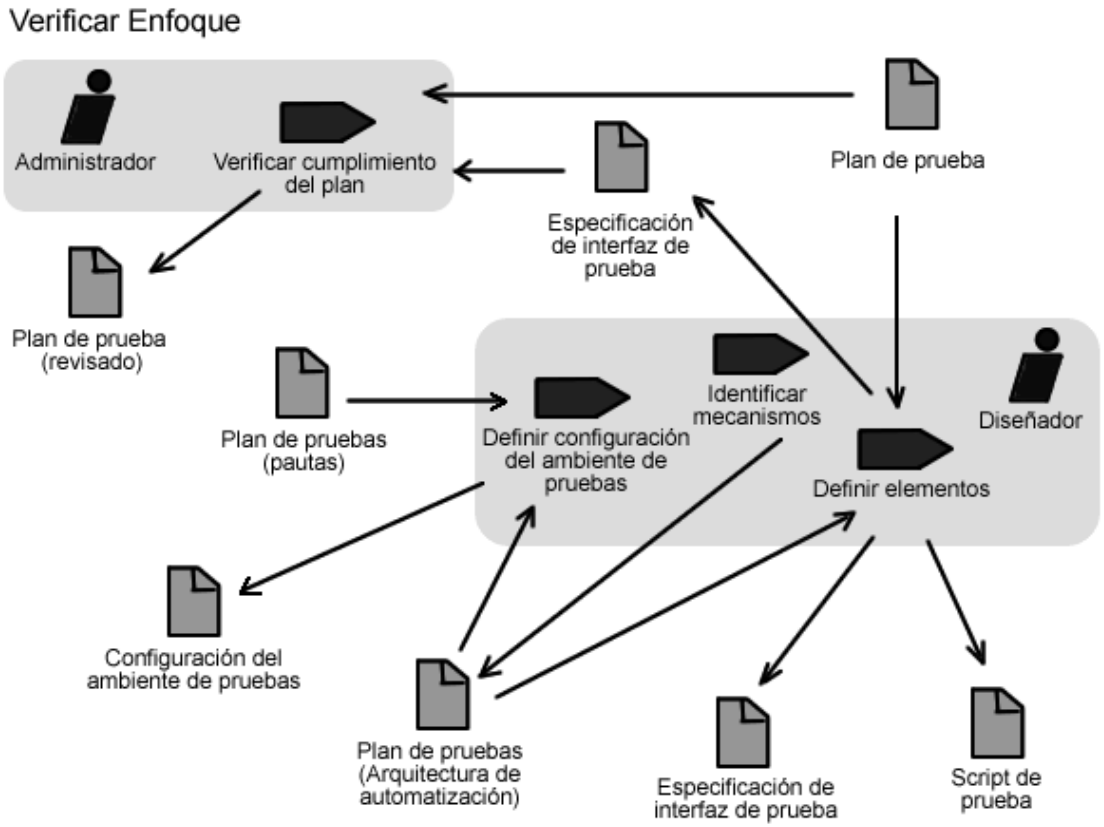


Figura 2.4 Verificar enfoque

Verificar enfoque:

El enfoque es una línea guía que posee los objetivos fundamentales y hacia qué punto deben ir dirigidas las pruebas en la iteración. Su función fundamental es seleccionar las técnicas que facilitarían la prueba planificada, el enfoque permitirá realizar las actividades según los recursos disponibles. El objetivo es lograr un entendimiento de las restricciones y limitaciones de cada técnica al aplicarlas en cualquier contexto del proyecto: encontrar una solución de implementación apropiada para cada técnica o encontrar técnicas alternativas que puedan usarse.

- Definir configuración del ambiente de pruebas. (Diseñador)

  Se definen los requerimientos para evaluar los ambientes necesarios que dan soporte al esfuerzo de prueba y las condiciones necesarias que deben estar presentes a la hora de ejecutar las pruebas, tanto hardware como software.

- Identificar mecanismos fundamentales. (Diseñador)

  Se identifican los mecanismos generales de las soluciones técnicas necesarias para facilitar las pruebas. Para determinar un alcance general y características claves de estos mecanismos.

- Definir elementos de pruebas. (Diseñador)

  Se identifican elementos necesarios para dar soporte a los blancos que serán probados. Identificar los elementos físicos de la infraestructura de implementación de la prueba requeridos para habilitar las pruebas bajo cada configuración de ambiente de prueba. Definir los requerimientos de diseño de software que será necesario satisfacer para habilitar el software y que se pueda muestrear físicamente.

- Verificar el cumplimiento del plan de pruebas. (Administrador)

  Se analiza cuando se estime conveniente el cumplimiento del plan de pruebas. Se recomienda hacerlo varias veces dentro de cada iteración.

Artefactos entrada:

- Plan de pruebas.
- Especificación de interfaz de prueba.

Artefactos resultantes:

- Configuración del ambiente de pruebas.
- Script de prueba.
- Especificación de interfaz de prueba.
- Plan de pruebas (revisado).

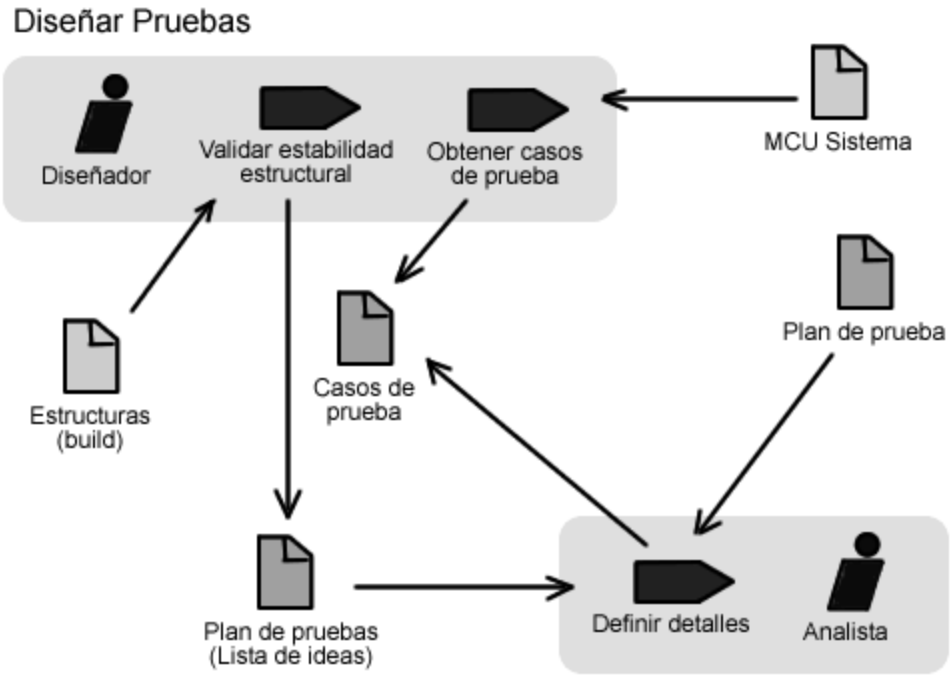


Figura 2.5 Diseñar las pruebas

Diseñar las pruebas:

A partir de las principales ideas de las pruebas que se van a realizar, se llevará a cabo un análisis de la forma en que serán implementadas las pruebas, el diseño de las pruebas y los diferentes casos de pruebas a implementar. Guiándose siempre por el enfoque que deben llevar las pruebas y las pautas específicas que se han trazado.

- Definir detalles de las pruebas. (Analista)
  Se definen las condiciones individuales y necesarias para llevar a cabo las pruebas en contextos específicos. Se identifican los puntos potenciales de observación y control de los artículos relacionados con las pruebas. Se proveen recursos utilizables para brindar soportes a las pruebas.

- Validar estabilidad estructural. (Diseñador)
  Se analiza si las estructuras o construcciones dentro del software bajo construcción son lo suficientemente estables y sólidas para ponerlas a prueba.

- Obtener los casos de prueba a partir de los casos de uso del sistema. (Diseñador)
  Se crean los casos de prueba como se muestra en el anexo 2.

Artefactos entrada:

- Plan de pruebas.
- Modelo de casos de uso del sistema.
- Estructuras (build)

Artefactos resultantes:

- Casos de prueba.
- Plan de pruebas (lista de ideas)

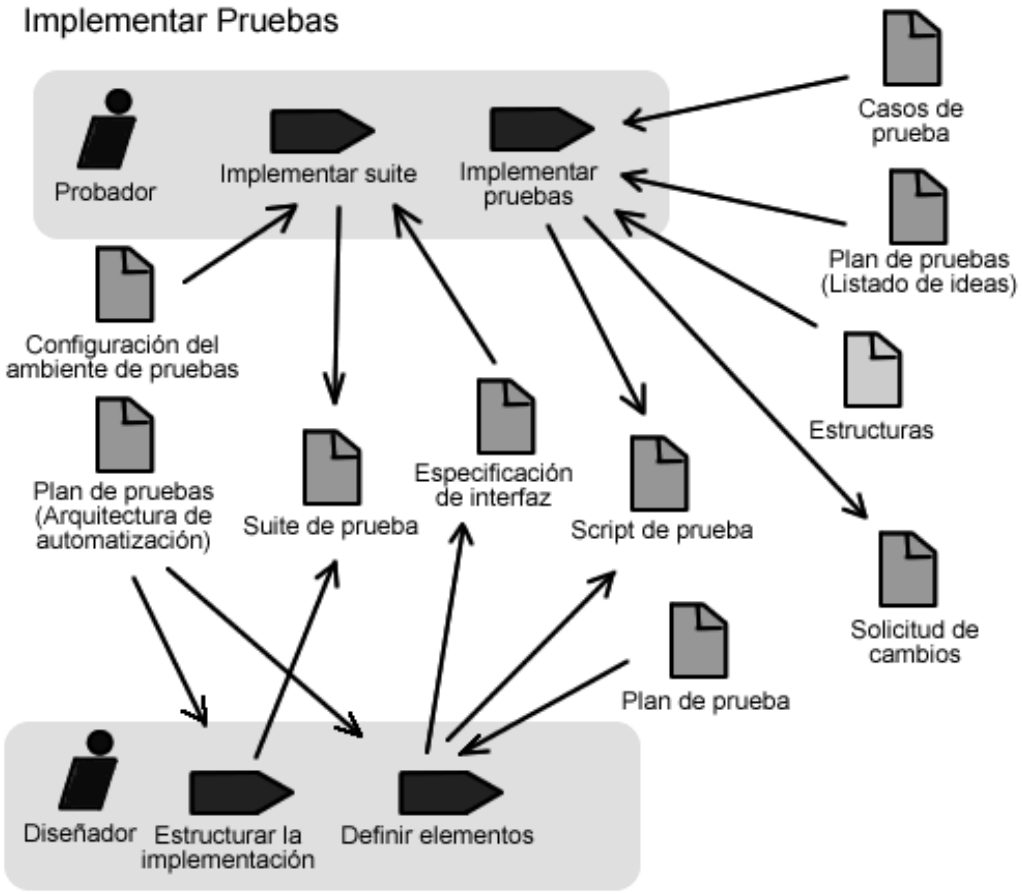


Figura 2.6 Implementar las pruebas

Implementar las pruebas:

Con el objetivo de realizar correctamente las pruebas planificadas y diseñadas. Se van a construir componentes útiles para la ejecución de pruebas que sirven de complemento a la suite de pruebas guiándose por la estrategia de prueba establecida y el enfoque de las pruebas. Se pone a punta el proceso para comenzar la ejecución.

- Definir elementos de pruebas. (Diseñador)

  Se identifican los elementos necesarios para dar soporte a los blancos que serán probados. Se identifican los elementos físicos de la infraestructura de implementación de la prueba requeridos para habilitar las pruebas bajo cada configuración de ambiente de prueba. Se definen los requerimientos de diseño de software que será necesario satisfacer para habilitar el software y que se pueda muestrear físicamente.

- Estructurar la implementación. (Diseñador)
  Establecer la estructura en la cual residirá la implementación de la suite de prueba. Asignar responsabilidades para las áreas de implementación de la suite de prueba y su contenido. Brindar las suites de prueba requeridas. Se crean los componentes o herramientas necesarios para ejecutar algún tipo de prueba específica.

- Implementar la suite de pruebas. (Probador)
  Se ensambla el conjunto de pruebas a ser ejecutadas junto a la captura de registros de prueba de valor. Facilitar la profundidad y el alcance de la prueba para la ejercitación de combinación de pruebas de interés.

- Implementar pruebas. (Probador)
  Implementar los artefactos que habiliten la validación del producto de software a través de su ejecución física. Desarrollar pruebas que puedan ser ejecutadas en conjunto con otras pruebas como parte de una gran infraestructura de pruebas.

Artefactos entrada:
- Configuración de ambiente de pruebas.
- Plan de pruebas.
- Estructuras.
- Casos de prueba.
- Especificación de interfaz de prueba.

Artefactos resultantes:
- Suite de pruebas.
- Script de prueba.
- Especificación de interfaz de prueba.
- Solicitud de cambios.

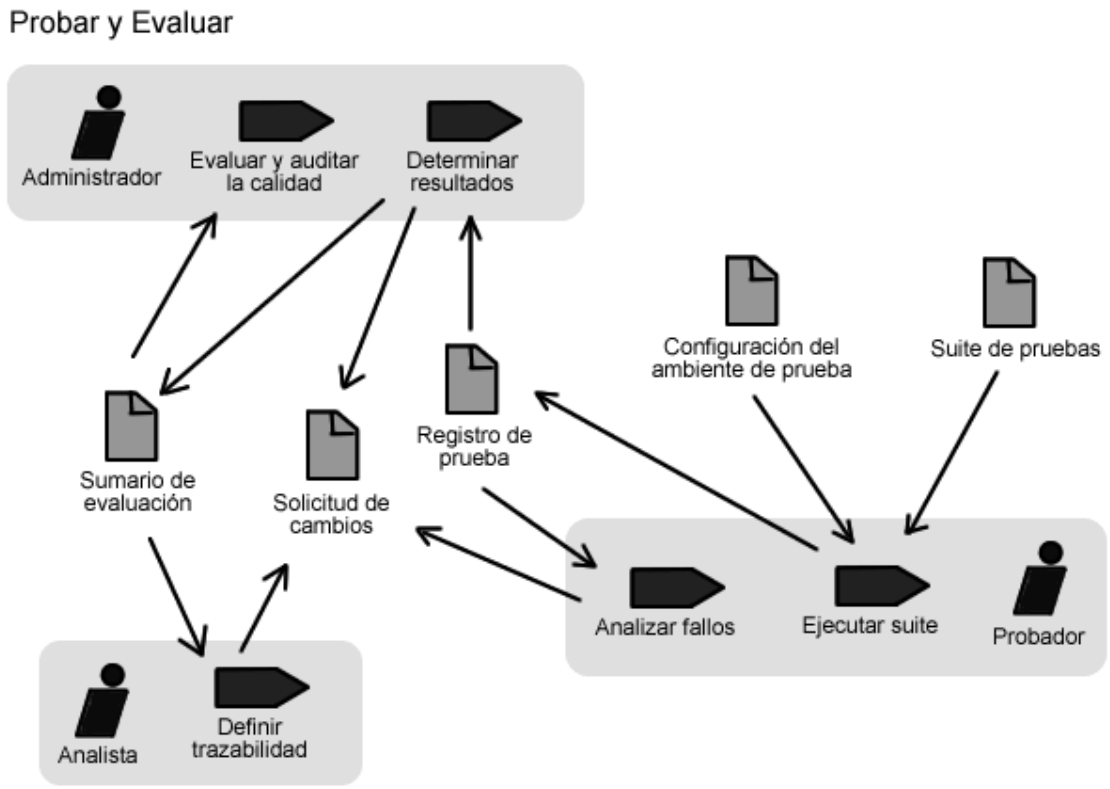


Figura 2.7 Probar y evaluar

Probar y evaluar:

Se ejecutan las pruebas para permitir una evaluación suficiente de los elementos objetivos. Se realizan las pruebas unitarias, de integración, de sistema.

- Ejecutar suite de pruebas. (Probador)
  Se ejecutan todas las pruebas requeridas. Se capturan los resultados de estas, para facilitar la evaluación en curso del producto.

- Determinar resultados de las pruebas. (Administrador)
  Se hace un resumen de las evaluaciones de la calidad del producto. Se identifican y capturan los resultados de prueba detalladamente. Se proponen las acciones apropiadas para corregir fallos en la calidad.

- Analizar fallos. (Probador)
  Se investigan los detalles de los registros de pruebas, y se analizan los fallos que ocurrieron durante la implementación y ejecución de las pruebas. Se rectifican fallos como consecuencia de errores en los procedimientos de pruebas. Se almacenan los resultados más importantes.

- Definir trazabilidad. (Analista)
  Se analiza y se definen todos los artefactos o componentes que van a ser afectados a partir de los resultados obtenidos.

- Evaluar y auditar la calidad (Administrador).
  Se identifica y obtienen los posibles defectos que podrían impactar directamente contra la calidad del software. Se monitorea el progreso del software, a través de sugerencias al Comité de Control de Cambios. Además se vela porque los defectos se corrijan en tiempo para prevenir retrasos en el proyecto.

Artefactos entrada:
- Suite de pruebas.
- Configuración de ambiente de pruebas.
- Registro de prueba.

Artefactos resultantes:
- Registro de pruebas.
- Solicitud de cambio.
- Sumario de evaluación de prueba.

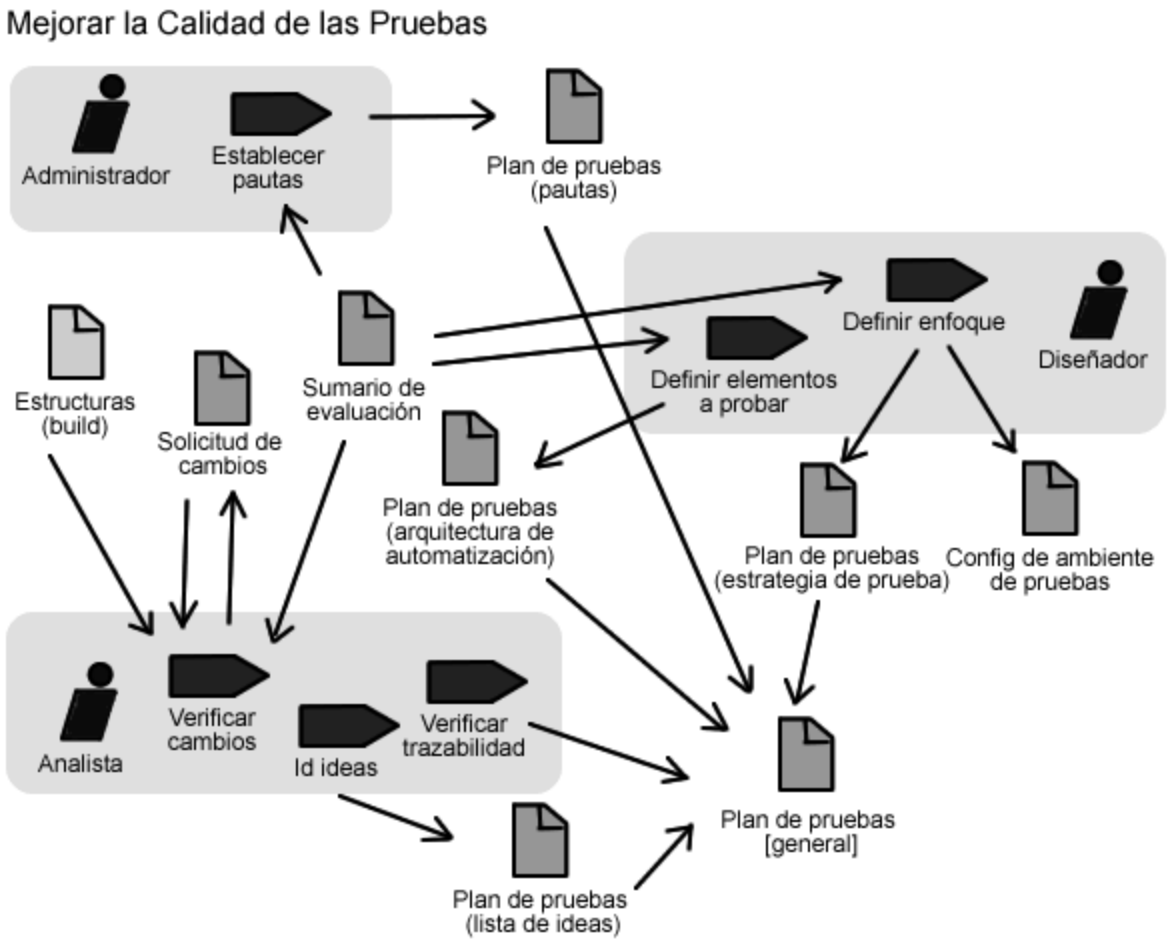


Figura 2.8 Mejorar la calidad de las pruebas

Mejorar calidad de las pruebas:

Su función fundamental es mantener y mejorar la calidad de las pruebas. Esto es especialmente importante si la intención es reutilizar las ventajas desarrolladas (componentes creados, experiencia adquirida) en el actual ciclo de prueba en ciclos de pruebas posteriores.

- Definir enfoque para pruebas posteriores. (Diseñador)

  A partir de los resultados obtenidos se establecen los objetivos para la próxima iteración.

- Definir elementos a probar. (Diseñador)

  Se definen los elementos que necesitan ser probados nuevamente.

- Identificar ideas de las pruebas. (Analista)

  Se obtienen ideas para pruebas posteriores a partir de la experiencia obtenida.

- Verificar cambios (Analista).

  Se verifican los cambios que deben haberse realizado en los componentes o artefactos afectados durante la corrección.

- Establecer pautas. (Administrador)

  Se establecen nuevas pautas a partir de las dificultades de las pruebas.

- Verificar la trazabilidad (Analista)

  Se verifica que se mantengan las relaciones de trazabilidad entre los componentes del sistema.

Artefactos entrada:
- Sumario de evaluación de prueba.
- Plan de pruebas.
- Solicitud de cambio.
- Estructuras.

Artefactos resultantes:
- Plan de prueba.
- Configuración de ambiente de pruebas.
- Solicitud de cambios (revisada).

2.3 Artefactos propuestos.

1. **Plan de prueba:** (Anexo 5) Define las metas y los objetivos de la prueba dentro del alcance del proyecto o la iteración, la estrategia de prueba, la lista de ideas acerca de las pruebas, la arquitectura de automatización, la configuración del ambiente de pruebas, se establecen las pautas para todo el proceso de pruebas, los elementos que son registrados, las medidas que se tomarán, los recursos requeridos y los entregables que se producirán.

   - El plan de pruebas principal define una estrategia para todo el esfuerzo de prueba a lo largo de todo el ciclo de vida del desarrollo del software. Opcionalmente se puede incluir el plan de prueba dentro del plan de desarrollo de software o como sección de este.
   - A partir del plan de pruebas principal se definen las tareas en el plan de pruebas para cada iteración. El plan de pruebas principal cubre fundamentalmente toda la información logística y de todo el proceso, abarcando todo el ciclo de vida.

- La estrategia de prueba se define en el plan principal, al comienzo de cada iteración se especifica qué parte de esta se va a aplicar durante el ciclo.

2. **Casos de prueba:** (Anexo 6) Define un sistema específico de entradas de la prueba, condiciones de ejecución, y resultados previstos. La documentación de casos de la prueba permite que sean repasados para su completitud y su corrección, y deben considerarse antes de que se planee y se expenda el esfuerzo de prueba. Esto es mayormente usado donde la entrada, las condiciones de ejecución y los resultados esperados son particularmente complejos. Contiene la descripcion de cada caso en uno de sus epígrafes.

   - Los casos de pruebas son recomendados para la mayoría de los proyectos donde las condiciones requeridas para conducir una prueba específica son extensas o complejas. Estos deben ser documentados correctamente constituyendo un entregable.
   - En la mayoría de los otros casos se recomienda mantener el listado de ideas de pruebas y los registros de prueba implementados e instanciados por la descripción de los casos de pruebas. Algunos proyectos contornearán simplemente casos de prueba a un alto nivel y diferirán de los detalles del Script de prueba. Otro estilo usado comúnmente es documentar la información del caso de prueba como comentario dentro de los registros de prueba.

3. **Script de prueba:** Puede tomar la forma de cualquier instrucción documentada textual que se ejecuta manualmente o las instrucciones legibles por computadora que permiten la ejecución automatizada de la prueba.

4. **Suite de pruebas:** Paquete usado para agrupar una colección de pruebas individuales relacionadas (script de pruebas) en conjuntos significativos. Un paquete usado para agrupar colecciones de pruebas, para ordenar la ejecución de esas pruebas y para proporcionar un sistema útil y relacionado de información del registro de pruebas del cual pueden obtenerse los resultados. También requerido para definir cualquier secuencia de ejecución en los registros de pruebas que se requiera para que las pruebas funcionen correctamente.

5. **Registro de prueba:** Una colección de salida cruda capturada durante una ejecución única de unas o más pruebas, representa generalmente la salida resultante de la ejecución de una suite de prueba para una corrida de un ciclo de prueba.

- Muchos proyectos que realizan pruebas automatizadas tienen de alguna forma un registro de las pruebas que se realizan. Donde difieren estos proyectos es en la acción de conservar o desechar estos registros después que los resultados de las pruebas se han determinado.
- Se pueden conservar los registros de las pruebas si se necesita satisfacer requerimientos de la audiencia o stakeholders, si se desea realizar un análisis de cómo una prueba cruda provoca cambios en los resultados o comportamiento del software a largo plazo, o si no se está seguro de la salida resultante del análisis que se ha realizado.

6. **Especificación de interfaz de prueba:** Define un sistema requerido de comportamientos hechos por un clasificador (específicamente, una clase, un subsistema o componente) para los propósitos de la prueba. Los tipos comunes incluyen el acceso a pruebas, el comportamiento tropezado, diagnostico de registro y oráculos de las pruebas. Cada interfaz de la prueba debe proporcionar un grupo único y bien definido de servicios.

7. **Sumario de evaluación de prueba:** (Anexo 7) Organiza y presenta un análisis sumario de los resultados de la prueba y de las medidas de la prueba para la revisión. Además, el resumen de la evaluación de la prueba puede contener una declaración general de la calidad relativa y proporcionar las recomendaciones para el esfuerzo futuro de la prueba. Contiene los resultados de las pruebas en uno de sus apéndices.

    - Cuando se trata de un proyecto mayormente informativo, puede ser apropiado simplemente registrar los resultados y no crear los resúmenes formales de evaluación de la prueba. En otros casos, el sumario de evaluación de la puede ser incluido como una sección dentro de otros artefactos de asesoramiento.

8. **Solicitud de cambios:** Se obtiene a partir del sumario de evaluación de pruebas. Mediante esta se proponen los cambios a los artefactos del desarrollo de software. Se utilizan para documentar y seguir los defectos, solicitudes de mejoras o cualquier tipo de cambio en el producto. Su principal ventaja es que brinda un expediente de decisiones que asegura el mantenimiento de la trazabilidad y el entendimiento de los cambios que se van realizando.

2.4 Consideraciones.

La propuesta consta de varios puntos que consideramos de importancia. Son los siguientes:

Plan de pruebas:

El propósito del plan de pruebas es dejar de forma explicita el alcance, el enfoque, los recursos requeridos, el calendario, los responsables y el manejo de riesgos de un proceso de pruebas. (TERUEL, 2001)

Está constituido por un conjunto de pruebas. Cada prueba debe:

- Dejar claro qué tipo de propiedades se quieren probar (corrección, robustez, fiabilidad, amigabilidad, etc.).
- Dejar claro cómo se mide el resultado.
- Especificar en qué consiste la prueba (hasta el último detalle de cómo se ejecuta).
- Definir cual es el resultado que se espera (identificación, tolerancia, etc.) y la forma de decidir que el resultado se corresponde con lo esperado.

Estas mismas ideas se suelen agrupar diciendo que un caso de prueba consta de tres bloques de información:

1. El propósito de la prueba.
2. Los pasos de ejecución de la prueba.
3. El resultado que se espera.

Todos y cada uno de esos puntos deben quedar perfectamente documentados. La información referente a la confección de estos se puede encontrar en el anexo 2.

El plan de pruebas señala el enfoque, los recursos y el esquema de actividades de prueba, así como los elementos a probar, las características, las actividades de prueba, el personal responsable y los riesgos.

Las actividades obtenidas en el flujo que se propone, en conjunto con los artefactos y los roles implicados, pueden ser aplicadas a cualquier proyecto de gestión, ya que el grupo de prueba puede garantizar el éxito de estas después de una correcta planificación del proceso.

Estrategia de prueba:

La propuesta de estrategia de prueba se incluye dentro de la plantilla del plan de prueba. Esta se muestra en el anexo 3.

Capítulo 3: Herramientas para la administración y ejecución de Pruebas

En esta sección se exponen algunas de las herramientas que están a nuestro alcance sus principales utilidades y beneficios. Algunas de estas se utilizan actualmente en nuestra universidad, el uso de otras es aún incipiente. La automatización de las pruebas dentro de un proceso de software trae consigo que se pueda agilizar y optimizar este proceso.

El objetivo es mostrar algunas de las características fundamentales de estas, así como los primeros pasos para su uso y explotación en aras de mejorar el proceso de pruebas.

3.1 Herramientas administrativas.

Para la administración de las pruebas realizamos el estudio de una de las herramientas de la Suite de Rational, pero antes de comenzar explicamos por qué es tan importante.

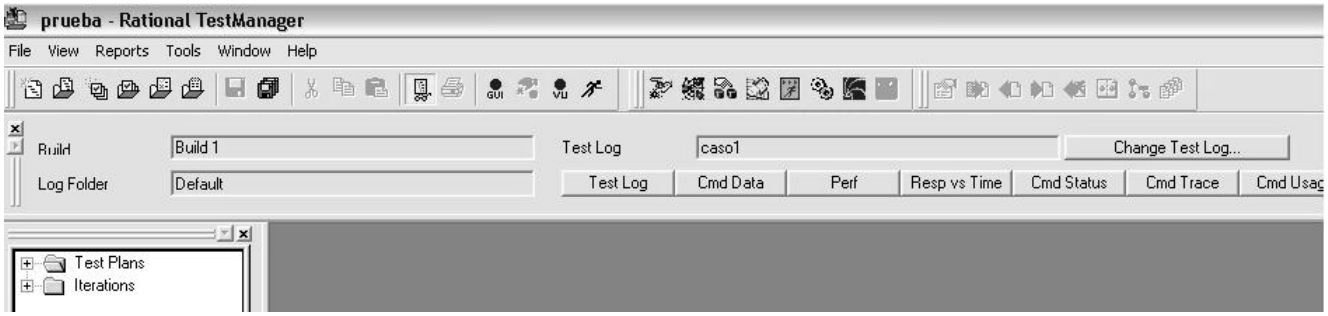
3.1.1 Administración de pruebas.

El objetivo de la administración de pruebas es definir los pasos a seguir para lograr un manejo integral del proceso de pruebas a lo largo del ciclo de vida de un proyecto. Contar con un ciclo de pruebas metódico que abarque las diferentes fases del desarrollo, comenzando con el análisis de la documentación del software, pasando al diseño de las pruebas en el cual se especifican casos y procedimientos, para posteriormente ejecutarlas sobre el software ya configurado, culminando con una evaluación y análisis de errores. Como complemento de estas actividades se procede a la localización y corrección de defectos y al análisis estadístico de los errores.

A lo largo del flujo de trabajo de pruebas, independientemente de la metodología o herramienta utilizada para la realización de estas, debe llevarse a cabo una correcta administración del proceso, lo que garantiza que se detecte una mayor cantidad de errores dentro del software, sin que se pierda una mayor cantidad de tiempo y recursos, ya que gran parte del esfuerzo total en la construcción de un software es derivado de las pruebas. Además, este proceso de administración de pruebas brinda para beneficio del administrador de prueba, una mayor facilidad en el control o chequeo de cada una de las tareas a realizar por parte de cada rol involucrado, desde el momento en que comienza la planificación de las pruebas hasta que se culmina el esfuerzo de prueba.

La herramienta propuesta para la administración del proceso de pruebas es el Test Manager, por sus prestaciones y su capacidad de vinculación con otras herramientas de la suite.

3.1.2 Test Manager.



Es un framework abierto y extensible que une todas las herramientas, utensilios y datos de pruebas para ayudar al equipo a definir y refinar sus objetivos. Contiene y organiza el plan de pruebas que guía al equipo para lograr esos objetivos, contiene las claves de toda la información necesaria para evaluar el estado del sistema en cualquier momento durante el ciclo de vida del proyecto. Permite la administración de las pruebas, ejecución y reportes de las pruebas. Soporta desde pruebas manuales hasta pruebas automatizadas de diversos tipos incluyendo pruebas unitarias, muestreo funcional de regresión y pruebas de rendimiento.

Resolución de la complejidad de las pruebas y administración de los asesoramientos:

Rational TestManager trata de automatizar y simplificar las tareas fundamentales. La herramienta la permite al equipo comenzar, dar seguimiento y probar toda la funcionalidad requerida de una aplicación, ayudando asegurar que ningún requerimiento crítico de negocio quede sin probar.

Beneficios:

- Proporciona una vista clara de todas las actividades y resultados de las pruebas.

- Soporte para ejecución de prueba local y remota: Ejecuta las pruebas en máquinas locales o remotas. La ejecución en paralelo de las pruebas está limitada solo por el número de recursos de sistema disponibles.

- Evaluación detallada de la prueba: Posee un visor de registro integrado para cada corrida de prueba, incluyendo el estado de las pruebas, información del y análisis de tiempo de ejecución.

- Generación de reportes significativos: Incluye una serie de reportes predefinidos gráficos y textuales que capturan los aspectos cruciales de la calidad de la aplicación y la completitud de las pruebas. Existen reportes especiales para análisis de pruebas de rendimiento, incluyendo reportes correlación, que combinan métricas de tiempo de respuesta con la utilización de recursos desde máquinas remotas.

- Integración profunda con las herramientas de la Suite**:** Se vincula directamente con herramientas de la suite como el Rational Robot, ClearQuest, Rational Administrador, ClearCase.

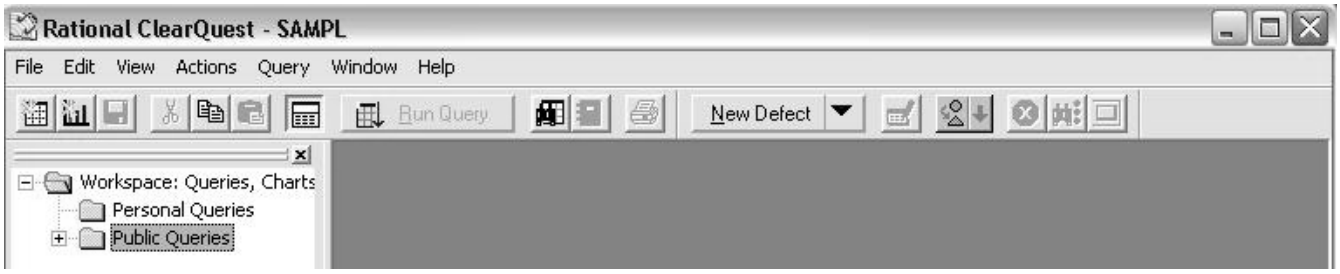
En el anexo 8 se muestran los primeros pasos para su utilización.

3.2 Herramientas de Seguimiento.

El control de cada reporte, ejecución o movimiento que se haga dentro de un sistema en prueba garantiza el mantenimiento de la trazabilidad que se establece dentro de un proyecto. Se hace necesario controlar los cambios originados a partir de defectos encontrados en el software, ya que afectan a los artefactos, y la única manera de lograr eficiencia a la hora de corregir errores es llevando a cabo un estricto control de todo lo que se ve afectado durante la corrección.

Una herramienta que garantiza lo expuesto anteriormente, estudiada durante el desarrollo del trabajo es el ClearQuest.

3.2.1 Rational ClearQuest.



Independientemente de la plataforma que se use, Windows, Unix o la Web, ClearQuest captura, da seguimiento y maneja eficientemente diferentes tipos de cambio. Permite manejar de forma eficiente la trazabilidad dentro del proyecto.

La integración con otras soluciones de desarrollo asegura que todos los miembros del equipo están unidos dentro de un mismo proceso de seguimiento de defectos y solicitudes de cambio, lo que resulta de gran importancia para lograr un producto de calidad.

Provee de los siguientes beneficios:

- Con él se puede dar seguimiento a los defectos y solicitudes de mejoras, asignar actividades y evaluar el estado real de los proyectos a través de todo el ciclo de vida.
- Captura, da seguimiento y maneja los cambios eficientemente.
- Ya sea que se trabaje bajo Windows, UNIX o la Web, la interfaz totalmente personalizable y el motor de flujo de trabajo son capaces de adaptarse a cualquier proceso de desarrollo.

Rational ClearQuest proporciona un seguimiento flexible de defectos y cambios en toda la organización.

- Seguimiento basado en actividad de cambios y defectos.
- Soporte robusto y flexible para flujos de trabajo, que incluye notificaciones por correo electrónico y opciones de envío.
- Soporte completo para consultas con generación de multitud de informes y gráficos.
- Interfaz Web para acceder fácilmente desde cualquier navegador Web estándar.

En el anexo 9 se muestran los primeros pasos para su utilización.

3.3 Herramientas de ejecución.

El desarrollo de la automatización de pruebas se ha incrementado paulatinamente y hoy en día se puede contar con herramientas que contribuyen en este sentido, que garantizan la ejecución de numerosos tipos de prueba.

Las herramientas para plataformas propietarias están más integradas, y como ejemplo de ello tenemos la Suite de Rational. Sin embargo, para plataformas libres estas se encuentran más dispersas, y surgen con la necesidad de resolver un problema en específico, aunque no por esto carecen de importancia, ni dejan de utilizarse, por el contrario, cada día aparecen en mayor número.

3.3.1 Herramientas Propietarias.

Las herramientas automatizadas para ambientes propietarios generalmente forman parte de un conjunto o suite, como la de Rational, que contiene herramientas que hacen posible la ejecución de procesos de forma simultánea. A continuación se muestran algunos ejemplos.

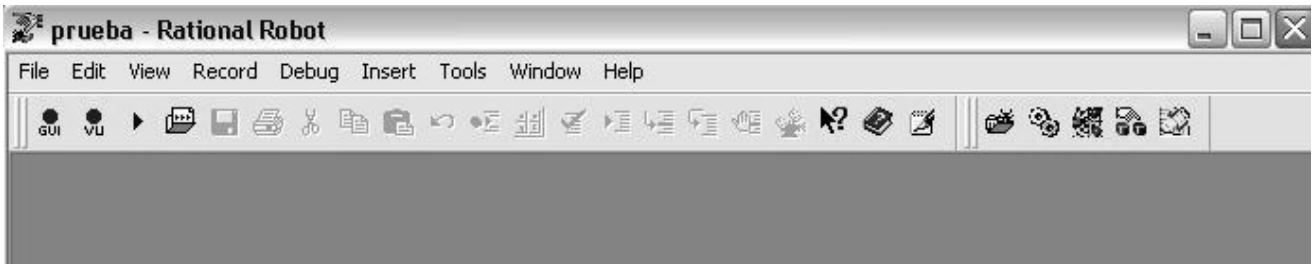
Rational Manual Tester:
Mejora la autoría y la ejecución de la realización de pruebas manuales, por medio de nuevas técnicas de diseño de pruebas.

Rational Funcional Tester:
Para pruebas funcionales automatizadas y avanzadas para aplicaciones Java, Web y VS.NET.

Rational Performance Tester:
Verifica un tiempo de respuesta y una escalabilidad aceptables para las aplicaciones, según las cargas variables de varios usuarios.

Rational Test RealTime:
Permite la realización de pruebas de componentes y el análisis de ejecución para el software incrustado y en tiempo real de varias plataformas

3.3.1.1 Rational Robot.



Rational Robot permite a los equipos de pruebas automatizar las pruebas de regresión de aplicaciones .NET, Java, Web y otras aplicaciones basadas en interfaz gráfica de usuario.

Es una herramienta para pruebas de configuración, regresión y funcionales, para entornos en los que se desarrollan las aplicaciones utilizando más de un lenguaje de programación. Facilita la transición de los equipos de pruebas manuales a pruebas automatizadas. La realización de pruebas de regresión con Rational Robot ayuda a la automatización, ya que resulta fácil de utilizar y ayuda a los equipos de pruebas a conocer los procesos de automatización mientras se trabaja.

Permite a los diseñadores de pruebas a identificar más defectos al ampliar sus scripts de pruebas, para abarcar una mayor parte de la aplicación y para definir los casos de prueba. Proporciona casos de prueba para objetos comunes, como menús, listas, mapas de bits y casos de prueba especializados para los objetos específicos del entorno de desarrollo.

Da soporte a múltiples tecnologías de interfaz de usuario para cualquier entorno: desde Java y la Web hasta todos los controles de VS.NET, incluidos VB.NET, J #, C # y C++ gestionado.

Ventajas:

- Permite un ahorro de tiempo a la hora de pruebas de regresión, permitiendo realizar estas sobre las nuevas versiones de la aplicación si las pruebas han sido bien planificadas.
- Permite programar la verificación de modo que la prueba solo requiere personal a la hora de analizar los resultados.

Inconvenientes:

- Requiere de un pequeño período de adaptación.
- No siempre se pueden capturar correctamente las acciones del usuario.

En el anexo 10 se muestran los primeros pasos para su utilización.

3.3.2 Herramientas Libres.

Dentro de las herramientas libres se analizaron dos de estas, que se han probado y se han puesto en funcionamiento en algunos de los proyectos de la universidad proporcionando buenos resultados.

3.3.2.1 JUnit.

Es un conjunto de librerías creadas que son utilizadas en programación para hacer pruebas unitarias de aplicaciones Java. Es un conjunto de clases que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, devolverá un fallo en el método correspondiente.

Es también un medio de controlar las pruebas de regresión, necesarias cuando una parte del código ha sido modificado y se desea ver que el nuevo código cumple con los requerimientos anteriores y que no se ha alterado su funcionalidad después de la nueva modificación. El propio framework incluye formas de ver los resultados que pueden ser en modo texto, gráfico o como tarea.

En el anexo 11 se muestran los primeros pasos para su utilización.

3.3.2.2 JMeter.

Es una herramienta Java, que permite realizar pruebas de rendimiento y pruebas funcionales sobre aplicaciones Web. Es una herramienta de carga para llevar acabo simulaciones sobre cualquier recurso de Software.

Inicialmente diseñada para pruebas de estrés en aplicaciones Web, hoy en día, su arquitectura ha evolucionado no sólo para llevar acabo pruebas en componentes habilitados en Internet (HTTP), sino además en Bases de Datos, programas en Perl, etc. JMeter puede ser utilizado para realizar pruebas de rendimiento con tipos de recursos estáticos y dinámicos: ficheros, scripts de perl, objetos java, bases de datos y consultas, servidores FTP, entre otros. Puede ser utilizado para simular pruebas de carga en un servidor, pruebas de red, pruebas de robustez sobre objetos y permite analizar el rendimiento global con distintos tipos de carga. Puede utilizarse para realizar un

análisis gráfico del rendimiento o para comprobar el comportamiento bajo carga pesada concurrente. (*Guía de JMeter*, 2005)

Además, posee la capacidad de realizar desde una solicitud sencilla hasta secuencias de requisiciones que permiten diagnosticar el comportamiento de una aplicación en condiciones de producción. En este sentido, simula todas las funcionalidades de un navegador, o de cualquier otro cliente, siendo capaz de manipular resultados en determinada requisición y reutilizarlos para ser empleados en una nueva secuencia.

En el anexo 12 se muestran los primeros pasos para su utilización.

Se conoce gran variedad de herramientas de software libre que se pueden utilizar, como podemos encontrar en (HOWER, 2007).

Herramientas para pruebas de carga y de funcionalidad.

Test Maker

Utilidad libre para pruebas funcionales de aplicaciones Web. Ofrece una lengua basada en scripts XML. Incluye capacidad para comprobar y supervisar sistemas de correo electrónico usando protocolos SMTP, POP3, IMAP. Herramienta Java que sirve para cualquier plataforma.

Site Stress

Herramienta para hacer pruebas de Stress a aplicaciones Web. Simula la actividad del usuario final en un sitio Web, para la prueba de la confiabilidad de funcionamiento y de infraestructura. Puede generar una carga infinitamente escalable del usuario conectado por la red, y proporciona divulgación del funcionamiento, el análisis, y recomendaciones de la optimización.

E-Carga

La herramienta de prueba de la carga del Web puede simular centenares o a millares de usuarios conectados simultáneamente a una aplicación; mediante una interfaz de usuario. Esto evita que colapse el sistema por exceso de conexiones cuando se pone en funcionamiento.

Herramientas para pruebas de Java.

Koalog

Herramienta que verifica la compilación de este lenguaje. Posee cobertura para conversión y generación en lenguaje binario, con XML, HTML, látex, CSV, texto, entre otros. Combina la sesión para permitir la compilación de los resultados totales para las ejecuciones distintas. Integra con JUnit.

JCover

Herramienta de análisis de cobertura de prueba de código. Trabaja con código fuente o con archivos compilados. Recolecta medidas de la cobertura de ramas, declaraciones, métodos, clases, archivo, paquete y produce informes en formatos múltiples.

JTest

Herramienta para prueba de unidad y de conformidad integrada, cubre los estándares para Java. Genera y ejecuta automáticamente las pruebas y los pasos de Junit, el código sigue 400 estándares de codificación y puede corregir muchos automáticamente.

3.4 Consideraciones.

La optimización del proceso de pruebas, en gran medida depende de la automatización de las pruebas, el uso de herramientas implica un menor tiempo de esfuerzo, y una detección más rápida de errores, en cualquiera de los niveles de prueba tratados anteriormente.

Con el avance de las tecnologías se espera que poco a poco se vayan incorporando nuevas herramientas que contribuyan a lograr mayores índices en la calidad de software y que el trabajo manual en las pruebas se vaya sustituyendo hasta donde sea posible.

Conclusiones.

En este trabajo que nace a partir de una necesidad actual de la universidad y la Industria Cubana del Software, se logra agrupar información acerca de las pruebas de software. Proponiendo un proceso de prueba generalizable y flexible.

A partir de las mejores prácticas de la aplicación de RUP, TSP y PSP se llega a una propuesta de flujo de trabajo que reúne las principales tareas, roles y actividades que deben realizarse en un proyecto para garantizar que las pruebas se realicen durante todo el proceso de desarrollo de software. Además de una estrategia de prueba con una visión general del proceso de pruebas, partiendo de los módulos hasta llegar al sistema, realizando pruebas de regresión durante todo el proceso para no arrastrar errores hasta los niveles superiores de la construcción.

Se obtienen artefactos fundamentales a partir de los propuestos en otros procesos y los utilizados actualmente en nuestra universidad y la empresa Softel. Con el objetivo de mejorar el trabajo con estos y hacerlos más generales.

Finalmente se realizó un estudio de herramientas que pueden utilizarse en todo lo referente a Administración, Seguimiento y Ejecución de pruebas, brindando una caracterización general de cada una de ellas, así como brindar una guía de los primeros pasos con algunas.

El resultado de un proyecto es un producto que va tomando forma durante cada una de las fases del desarrollo. La calidad final de un producto esta ligada estrechamente a la calidad del proceso de desarrollo de software, y dentro de este, depende en gran medida de las pruebas que se realicen, independientemente de un buen diseño, codificación y aspectos inherentes a la construcción de un producto.

Recomendaciones.

El presente trabajo está encaminado a mejorar la calidad de las pruebas y por consiguiente, elevar los índices de calidad requeridos para los productos que se realizan en nuestro país, y específicamente en nuestra universidad y la empresa Softel.

Recomendamos que:

- Se continúe el estudio y la profundización de todos los aspectos que se refieren a las pruebas de Software.
- Se tenga en cuenta la propuesta obtenida, a la hora de realizar pruebas en los diferentes proyectos de la Universidad, para que sea evaluada, y teniendo en cuenta los resultados de la evaluación del proceso, se continúe perfeccionando y enriqueciendo.
- Para conformar los equipos de pruebas y la distribución de los roles, analizar los planes de estudio para darle participación a estudiantes de los diferentes años.
- En cuanto las herramientas automatizadas de pruebas:
  - Se profundice en el estudio de la integración entre las herramientas de la suite de Rational que se relaciona con pruebas.
  - Se continúe investigando y profundizando acerca de las herramientas libres para ejecución de pruebas.

Referencias bibliográficas.

ACUÑA, C. J. *Pruebas de Software* Ingeniería del Software Universidad Rey Juan Carlos: [Consultado el: 12 de abril de 2007]. Disponible en: http://kybele.escet.urjc.es/documentos/ISG/%5BISG-2006-07%5DPruebasSoftware.pdf

ANTHONY LATTANZE, M. R.-L., AND JAMES TOMAYKO. *Teaching the Personal Software Processsm and the Team Software*
Processsm Carnegie Mellon University,

BEIZER, B. *Black-Box Testing.* Wiley, 1995.

---. *Software System Testing and Quality Assurance.* Van Nostrand Reinhold, 1984.

---. *Software Testing techniques.* Van Nostrand Reinhold, 1990. vol. Segunda Edición,

BERARD, E. V. . *Essays on Object-Oriented Software Engineering.* Addison-Wesley, 1993. vol. 1,

BINDER, R. V. *Testing OO Systems: A Status Report.* American Programmer, 1994. vol. 7, 23-28 p.

CIG_LABS. *The Home of Groundbreaking Software Quality* [Consultado el: 12 de 2006]. Disponible en: http://:www.cigitallabs.com/resources/definitions/software_testing.html.

CUBAMINREX. *La informatización en Cuba* 2005, Disponible en: http://www.cubaminrex.cu/Sociedad_Informacion/Cuba_SI/Informatizacion.htm.

DAVIS, A. *Principles of Software Development.* McGraw-Hill, 1995. 201 p.

DEUTSCH, M. *Verification and Validation.* Software Engineering, R. Jensen y C. Tonies ed. Prentice Hall, 1979. 329-408 p.

Guia de JMeter osmosis Latina, [Consultado el: 12/05 de 2007]. Disponible en: http://www.osmosislatina.com/jmeter/basico.htm

HOWER, R. *Software QA/Test Resource Center* [Consultado el: 5/05 de 2007]. Disponible en: http://www.softwareqatest.com/qatweb1.html#LOAD.

HUMPHREY, W. *Introducción al Proseso Software Personal.* Addison Wesley, 2001, ISBN 84-7829-052-4.

---. *"Managing technical people: innovation, teamwork, and the software process".* Addison Wesley Longman, 1997.

IEEE. *Metrics* IEEE, (IEEE std-1995).

ISABEL, A. M. *Tema 8. Pruebas del software.* Universidad de Alicante, 2002.

JAVIER, J. G. M. J. E., MANUEL MEJÍAS Y ANTONIA M. REINA. *MODELOS DE PRUEBAS PARA PRUEBAS DEL SISTEMA.* 2006, Disponible en: http://www.lsi.us.es/~javierj/publications/MDA14.pdf.

KANER, C. J. F., Y H. Q. *Testing Computer Software.* 1993. vol. 2, Edición, Van Nostrand Reinhold.

MCCABE, T. *A Software Complexity Measure.* IEEE Trans. Software Engineering, 1976. vol. 2, 308-320 p.

MORENO, B. En *Discurso en el Primer Taller de Calidad de Software. Informática 2003, La Habana, Cuba. 2003*.

MYERS, G. *The Art of Software Testing.* Wiley, 1979.

PRESSMAN, R. S. *Can Internet-based Applications Be Engineered?* . in IEEE Software, 1998. 104 - 110 p.

---. *Ingenieria de Software: UN Enfoque Práctico.* Felix Varela ed. La Habana: 2005. vol. 1, 281 p.

---. *Software Engineering: A Practitioner's Approach.* McGraw-Hill, 2000. ISBN 0-07-709677-0.

---. *Software Engineering: a practitioner's approach.* European ed. McGraw Hill, 1997.

RATIONAL, S., CORPORATION. *Ayuda del Rational.* Estados Unidos: 2003,

TERUEL, A. *Prueba repetibles y mantenibles.* Universidad Simón Bolívar 2001.

Bibliografía.

1. ACUÑA, C. J. *Pruebas de Software Ingeniería del Software Universidad Rey Juan Carlos: [Consultado el: 12 de abril de 2007]. Disponible en: http://kybele.escet.urjc.es/documentos/ISG/%5BISG-2006-07%5DPruebasSoftware.pdf*

2. BEIZER, B. *Black-Box Testing. Wiley, 1995.*

3. *---. Software System Testing and Quality Assurance. Van Nostrand Reinhold, 1984.*

4. *---. Software Testing techniques. Van Nostrand Reinhold, 1990. vol. Segunda Edición,*

5. BERARD, E. V. *Essays on Object-Oriented Software Engineering. Addison-Wesley, 1993. vol. 1,*

6. BINDER, R. V. *Testing OO Systems: A Status Report. American Programmer, 1994. vol. 7, 23-28 p.*

7. CIG_LABS. *The Home of Groundbreaking Software Quality [Consultado el: 12 de 2006]. Disponible en: http://:www.cigitallabs.com/resources/definitions/software_testing.html.*

8. CUBAMINREX. *La informatización en Cuba 2005, Disponible en: http://www.cubaminrex.cu/Sociedad_Informacion/Cuba_SI/Informatizacion.htm.*

9. DAVIS, A. *Principles of Software Development. McGraw-Hill, 1995. 201 p.*

10. DEUTSCH, M. *Verification and Validation. Software Engineering, R. Jensen y C. Tonies ed. Prentice Hall, 1979. 329-408 p.*

11. HUMPHREY, W. *Introducción al Proseso Software Personal. Addison Wesley, 2001, ISBN 84-7829-052-4.*

12. ---. "Managing technical people: innovation, teamwork, and the software process". Addison Wesley Longman, 1997.

13. IEEE. Metrics IEEE, (IEEE std-1995).

14. ISABEL, A. M. Tema 8. Pruebas del software. Universidad de Alicante, 2002.

15. JAVIER J. GUTIÉRREZ, M. J. E., MANUEL MEJÍAS Y ANTONIA M. REINA.

16. MODELOS DE PRUEBAS PARA PRUEBAS DEL SISTEMA. 2006, Disponible en: http://www.lsi.us.es/~javierj/publications/MDA14.pdf.

17. KANER, C. J. F., Y H. Q. Testing Computer Software. 1993. vol. 2, Edición, Van Nostrand Reinhold.

18. MCCABE, T. A Software Complexity Measure. IEEE Trans. Software Engineering, 1976. vol. 2, 308-320 p.

19. MORENO, B. En Discurso en el Primer Taller de Calidad de Software. Informática 2003, La Habana, Cuba. 2003.

20. MYERS, G. The Art of Software Testing. Wiley, 1979.

21. PRESSMAN, R. S. Can Internet-based Applications Be Engineered? . in IEEE Software, 1998. 104 - 110 p.

22. ---. Ingeniería de Software: UN Enfoque Práctico. Félix Varela ed. La Habana: 2005. vol. 1, 281 p.

23. ---. Software Engineering: A Practitioner's Approach. McGraw-Hill, 2000. ISBN 0-07-709677-0.

24. ---. Software Engineering: a practitioner's approach. European ed. McGraw Hill, 1997.

25. RATIONAL, S., CORPORATION*. Ayuda del Rational. Estados Unidos: 2003,*

26. TERUEL, A*. Prueba repetibles y mantenibles. Universidad Simón Bolívar 2001.*

27. CÉSAR, M. D. O., V. Team Software Process (TSP): Integración de Equipos de Desarrollo de Alto Rendimiento Carnegie Mellon University.

28. COLLADO, M. Técnicas de pruebas del software, estrategias de pruebas del software [Consultado el: 24/4 de 2007]. Disponible en: http://lml.ls.fi.upm.es/ftp/ed2/0203/Apuntes/pruebas.ppt#284,1,Pruebas de software.

29. FERRÓN MARÍA JOSÉ, G. J. P. TESTING (Pruebas) [Consultado el: 24/4 de 2007]. Disponible en: http://www.ldc.usb.ve/~teruel/ci3711/test1/index.html.

30. IVAR JACOBSON, G. B., JAMES RUMBAUGH. El Proceso Unificado de Desarrollo de Software. Traducido por: Salvador Sánchez, M. A. S., Francisco Javier Dúran, Carlos Canal. Addinson Wesley, 1999. ISBN 84-7829-036-2.

31. LÓPEZ, F. P. Casos de Uso: Una Guía para la definicion de los Casos de Prueba. 2004.

32. LUÍS FERNÁNDEZ SANZ, J. J. C.-G. Innovación, Calidad e Ingeniería del Software. 2005, vol. 1, Disponible en: www.ati.es. ISBN 1885-4486.

33. TESTMANAGER, R. User's Guide. Rational Software Corporation, 2003, Disponible en: http://www.rational.com.

34. WILL, H. J., W. OVER. The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers

35. Pennsylvania: Carnegie Mellon University, 1997.

Glosario de términos:

FTP: Protocolo de transferencia de archivo (File Transfer Protocol).

GUI: Interfaz gráfica de usuario (Graphic User Interface).

HTTP: Protocolo de transferencia de hipertexto (Hyper Text Transfer Protocol).

IEEE: Institute of Electrical and Electronics Engineers, que establece un Glosario estandarizado de terminología de Ingeniería de Software.

VU: Lenguaje rational para testeo virtual (para aplicaciones C).

Puntos de verificación: puntos de comprobación automática del estado de la aplicación.

Script: Texto que especifica un guión o acciones a realizar.

Software de Gestión: Permite el procesamiento de datos, el cálculo interactivo y el procesamiento de información comercial, que constituye su mayor área de aplicación del software.

SQA: Gestión de la calidad de Software (Software Quality Assurement).

Stakeholder: Participante, interesado, actor del sistema, beneficiado con el producto.