Universidad de las Ciencias Informáticas
Facultad de Bioinformática

Propuesta de pruebas de calidad para producto LIMS Control de Calidad del CIGB.

Trabajo de Diploma para optar por el título de Ingeniero en ciencias Informáticas.

Autores: Keyly Betancourt Rodríguez
Frank Gabriel Rodríguez Martell

Tutores: Ing. Reynier García Vistorte
Lic. Yosdenis Urrutia Badillo

Ciudad de la Habana, julio del 2007.
"Año 49 de la Revolución"

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año 2007

Keyly Betancourt Rodriguez

_____

Firma del primer autor

Ing. Reynier García Vistorte

_____

Firma del primer tutor

Frank Gabriel Rodríguez Martell

_____

Firma del segundo autor

Lic. Yosdenis Urrutia Badillo

_____

Firma del segundo tutor

Opinión del tutor del trabajo de diploma

Título: Propuestas de pruebas de calidad para producto LIMS Control de Calidad del CIGB.
Autores: Keyly Betancourt Rodríguez y Frank Gabriel Rodriguez Martell.

Los tutores del presente Trabajo de Diploma consideran que durante su ejecución los estudiantes mostraron las cualidades que a continuación se detallan:

Los estudiantes manifestaron una actitud disciplinada, independiente y responsable, mucha dedicación al trabajo, gestionando información y enfocada a la mejora continua de su trabajo**.**

Su trabajo es perfectamente aplicable al entorno de la UCI, los resultados poseen un criterio científico adecuado y traerán beneficios de funcionamiento a los productos que la apliquen, con ello su calidad frente al cliente.

Por todo lo anteriormente expresado consideramos que los estudiantes están aptos para ejercer como Ingenieros Informáticos; y proponemos que se le otorgue al Trabajo de Diploma la calificación de 5-Excelente. Además, consideramos que los resultados poseen valor para ser publicados.

Ing. Reynier García Vistorte	Lic. Yosdenis Urrutia Badillo
_____	_____
Nombre completo del primer tutor	Nombre completo del segundo tutor

Fecha: 06 de julio de 2007.

Pensamiento

"La responsabilidad nuestra es luchar porque la calidad del producto que aquí se haga sea de las mejores y la mejor posible…"

Che

Agradecimientos

- ✓ *A nuestro tutores Ing. Reynier García Vistorte y Lic. Yosdenis Urrutia Badillo así como a todo el colectivo de profesores del centro.*
- ✓ *Lic. Lázaro Castillo García. Empresa DESOFT S.A Ciudad de la Habana*
- ✓ *A todos nuestros familiares que siempre nos han tenido presente.*
- ✓ *A nuestros amigos por su preocupación constante.*
- ✓ *A todos los que de una forma u otra hicieron posible que llegáramos hasta aquí.*

Mencionándolos no basta para agradecerles todo su apoyo; sin embargo es la única forma de dejar constancia de lo que todos ustedes han hecho por nosotros.

- ✓ *Muy en especial a nuestra revolución y a nuestro comandante en jefe Fidel Castro Ruz quien nos facilito con su brillante idea de crear la Universidad la gran oportunidad de formar parte de este gran proyecto.*

Dedicatoria

Se lo dedicamos muy en especial a nuestros padres por la gran obra que han venido realizando, a los que nos enseñaron a dar nuestros primeros pasos, a los que lograron hacer de nosotros hombres de bien, a los que ni dedicándoles nuestras vidas recompensaremos jamás.

Resumen

Cuando hablamos de calidad de software hay que tener en cuenta las pruebas como un elemento crítico, ya que estas representan una revisión de las especificaciones, del diseño y de la codificación y constituyen una etapa dentro del desarrollo del producto LIMS Control de Calidad del CIGB.

Es de suma importancia seguir las estrategias de pruebas a la hora de aplicar las mismas, para lograr que estas se hagan en el menor tiempo posible y con la calidad requerida, además de lograr que arrojen los resultados esperados.

Los principales aspectos a ser evaluados en un producto software son la Fiabilidad (resistente a fallos), la Funcionalidad (hace lo que debe) y el Rendimiento (lleva a cabo su trabajo de manera efectiva). Las pruebas pueden hacerse a diferentes niveles dependiendo del objetivo de los mismos, a saber: pruebas de unidad (se prueban las unidades mínimas por separado, y normalmente se hace durante la implementación misma), de integración (varias unidades juntas), de sistema (sobre la aplicación o sistema completo) y de aceptación (realizado sobre el sistema global por los usuarios o terceros).

En este trabajo se define un grupo de pruebas que permitirán colaborar con la calidad del producto LIMS, basadas en los requerimientos planteados por la agencia regulatoria y específicamente por la **Extensión 21 CFR Parte 11** para este tipo de producto con el que nos encontramos trabajando. En este procedimiento se detallan las técnicas empleadas en cada prueba, quiénes participan, qué hacen, cuándo y cómo deben hacerlo; así como: qué artefactos se generan en todo este complejo proceso de pruebas de software.

Índice

Introducción

La Industria del Software en el mundo se ha desarrollado notablemente en los últimos años, sin embargo, los resultados alcanzados no cubren las expectativas inicialmente vislumbradas debido básicamente a que la productividad que se alcanza, en general, es baja, la cantidad de recursos a consumir -en tiempo principalmente- es alta y el trabajo realizado casi nunca tiene la calidad requerida. (1) (2) (3)

Cuando se profundiza en las raíces o causas de tales insatisfacciones afloran de manera reiterada la aplicación inadecuada de las técnicas de Ingeniería de Software, la no utilización de los roles y procesos apropiados para el desarrollo de las tareas de la empresa de software y el no utilizar modelos de calidad en ellas. (1) (4) (5)

El desarrollo de la industria de software implica también que los productos realizados deben de ser confiables, precisos, rápidos de utilizar, flexibles y además deben de estar bien documentados. Un producto cumple con estos requisitos cuando se le han aplicado las técnicas de Ingeniería de Software adecuadas, se han utilizados los roles apropiados para el desarrollo de las tareas en la empresa de software y en su etapa de comprobación se le han aplicado las pruebas necesarias para lograr el nivel de calidad requerido. (6)

La calidad es otro aspecto que ha tomado gran importancia en el mundo con el desarrollo de la industria de software, principalmente porque influye en la competencia y en la aceptación del producto por parte de los clientes. Esta calidad tuvo su origen durante la segunda guerra mundial, aplicado sobre el producto final obtenido de la manufactura, con objetivo de garantizar que éste cumpliera con los requerimientos del cliente. La evolución y la generación de nuevas técnicas, equipos y procedimientos, llevaron a la necesidad de evaluar constantemente los estándares de calidad, tanto de los procesos de producción como del producto final, incorporándose, además del aseguramiento de la calidad, aspectos de gestión de la calidad.

Pero ya alrededor de la década del 70, este tema se hizo motivo de preocupación a nivel mundial para especialistas, ingenieros, investigadores y comercializadores de software, los cuales han realizado gran

cantidad de investigaciones al respecto con dos objetivos fundamentales: ¿Cómo obtener un software con calidad? ¿Cómo evaluar la calidad del software? (7)

La tendencia de las empresas a implantar lo que se considera “calidad” en los productos, ha alcanzado a la industria del software, esto se basa principalmente en la idea de que la obtención de productos de mejor calidad, se asegura si los procesos de producción y mantenimiento de los mismos la presentan.

El Centro de Ingeniería Genética y Biotecnología (CIGB), siendo una institución de desarrollo dinámico que le ha permitido alcanzar un alto nivel en la investigación, desarrollo, producción y comercialización de productos biológicos obtenidos a través de los métodos de la biotecnología moderna, se ha visto en la necesidad de poner en marcha una solución LIMS (del inglés Laboratory Information Management System), que son Sistemas de Gestión de la Información del Laboratorio, para su control de calidad basada en Sistemas Informáticos que permiten la aplicación de técnicas de adquisición y gestión avanzada de la información producida en el laboratorio. La aplicación con éxito de LIMS siempre ha contribuido a mejorar la productividad y aumentar la exactitud de los laboratorios y ha mejorado la eficiencia y la calidad del servicio siempre que se tuviera presente lograr durante todo el proceso de producción del software la calidad del mismo.

Resumiendo la **situación problemática** como el desconocimiento de pruebas de calidad para productos LIMS que cumplan con los requisititos y regulaciones planteados por la FDA para este tipo de software.

El problema: ¿Cuales son las pruebas de calidad apropiadas para productos LIMS?

En **la actualidad** en la facultad se está trabajando en el producto LIMS control de calidad del Centro de Ingeniería Genética y Biotecnología (CIGB).

Surgiendo en la facultad **la necesidad** de contar con un grupo de pruebas que cumplan con los requisitos y normativas propuestas por la FDA para este tipo de software, permitiendo uniformar la filosofía de trabajo, en aras de lograr una mayor confiabilidad, mantenibilidad y facilidad del producto, a la vez que eleven la productividad, tanto para la labor de desarrollo como para el control de la calidad del software.

Para el desarrollo de este trabajo tuvimos que realizar varias investigaciones en el ámbito internacional así como nacional acerca de **los antecedentes** de la calidad en un software y en especifico la de un LIMS de control de calidad de esta envergadura. Para lo cual se tuvo que visitar varios sitios en internet, donde pudimos conocer que en el pasado, el control de calidad se limitaba solamente a los productos terminados, con base en la calificación cuantitativa y/o cualitativa de las características del producto y su comparación con los requerimientos del cliente.

Posteriormente, la aplicación del control de la calidad se extendió hacia la ejecución de los procesos, con el objeto de asegurar que la calidad esté presente en cada una de sus etapas, promoviendo la mejora continua del sistema de producción.

Bajo el esquema actual de mejoramiento de productos, procesos y servicios, ya no es suficiente con el sólo aseguramiento de la calidad, sino que se ha vuelto imprescindible que la alta dirección se involucre activamente en los aspectos que afectan la calidad de los procesos, de modo que sea ésta quien gestione la mejora continua. En México, los estándares de producción han sido desarrollados por las dependencias gubernamentales, contando con el apoyo de la iniciativa privada. En esencia, las empresas de la iniciativa privada son, los interesados directos en aplicar dichos estándares. Además, la firma de tratados y convenios comerciales internacionales ha hecho necesario que los países tiendan a armonizar sus normas, tanto en la fabricación de productos, como en la forma en que éstos han de ser certificados. (8)

Anteriormente, las pruebas de software se consideraban sólo una actividad que realizaba el programador para encontrar fallas en sus productos; con el paso de los años se ha determinado la importancia que tienen para garantizar el tiempo, el costo y la calidad del producto, de tal forma que actualmente son un proceso cuyo propósito principal es evaluar la funcionalidad del software respecto de los requerimientos establecidos al inicio.

¿Cuál es la nueva tendencia en las pruebas? Iniciarlas antes, dentro del proyecto y capacitar a especialistas responsables de esta actividad. El primer punto quiere decir que actualmente las especificaciones de pruebas se realizan al mismo tiempo que el diseño de software; la propuesta es iniciar el análisis del testware junto con el análisis del software. Ello habla de sondeos preventivos que permitan

ejecutar las pruebas tan pronto como el software esté listo y con ello no sólo descubrir errores, sino evitarlos. (9)

En la industria de software en Cuba existen problemas con la productividad de los trabajadores, los tiempos de entrega de los productos y de las documentaciones, la calidad de las pruebas y la falta de comunicación efectiva entre los usuarios, desarrolladores, administradores, clientes e investigadores [3].

La Industria Cubana del Software (InCuSoft) está llamada a convertirse en una significativa fuente de ingresos para el país, como resultado del correcto aprovechamiento de las ventajas del alto capital humano disponible. La promoción de la industria cubana del 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. El continuar la producción sostenida de software de alta calidad en prestaciones, imagen y soporte, para satisfacer las necesidades nacionales en estos sectores, tendrá una positiva repercusión en el incremento de la exportación

Siguiendo este principio, desde inicios del 2000, y teniendo como objetivo desarrollo de la Industria Cubana de Software, en el Centro de Referencia de Ingeniería de Software (CRIS) del Instituto Superior Politécnico “José Antonio Echeverría” (CUJAE), se vienen desarrollando un grupo de investigaciones relacionadas con la madurez de las organizaciones y la calidad en los procesos de desarrollo y en el producto final.

Se estudiaron propuestas de pruebas de calidad y se tomaron los aspectos positivos de las mismas. Se analizaron las pruebas que se llevan a cabo en el centro como guía para nuestro trabajo, analizando detalladamente las características y requerimientos fundamentales que deben de cumplir estas pruebas, para garantizar que el producto cumpla con todos los requisitos funcionales y no funcionales que caracteriza a un software de estas características.

Con la propuesta de estas pruebas de esperan importantes aportes para aquellas personas que se encuentren involucradas en este mundo de la calidad de software así como nuevas experiencias para

diseños posteriores que se realizaran para la liberación de nuevos software, siendo este trabajo las bases y la guía para nuevos planes de pruebas que se llevaran a cabo en nuestro centro.

Definiendo como **objeto de estudio**: Proceso de Calidad de Software.

Delimitando así el **campo de acción**: Pruebas de calidad para productos LIMS.

Métodos teóricos:

Histórico-lógico: En la primera fase de nuestra investigación se desarrolla un estudio del estado del arte de la problemática analizada; revisando de forma crítica cada uno de los documentos para poder resaltar la importancia de las diferentes estrategias de pruebas que se llevan a cabo en la actualidad en el mundo, así como la eficiencia y/o deficiencias de cada una de estas.

El analítico sintético: Es el método empleado para el procesamiento de la documentación, así como para precisar las diferentes características de las estrategias de pruebas a emplear para el LIMS Control de Calidad del CIGB

El Hipotético deductivo: Permite a partir del problema concreto plantear los objetivos específicos e hipótesis que en el transcurso de la investigación son resueltos siguiendo métodos científicamente bien fundamentados.

Analítico Sistémico: Nos plantea el problema como un todo, estudio de graficas, esquemas del uso de las pruebas de software y cada uno de los métodos y herramientas que utilizaremos.

Para dar seguimiento a este trabajo nos trazamos como **objetivo general**: Analizar y seleccionar las pruebas que permitan controlar la calidad del producto LIMS control de calidad del CIGB.

Objetivos específicos:

- Desarrollar políticas para elegir las posibles pruebas a proponer.
- Fundamentar las pruebas de software especificando sus objetivos.
- Realizar la propuesta de un grupo de pruebas de calidad que se ajuste a las normas ISO y FDA para este tipo de software.

Por lo que nos proponemos llevar a cabo el cumplimiento de estos objetivos desarrollando las siguientes **tareas**:

- Realización de una investigación que recoja todo lo relacionado con las pruebas de software que se deben aplicar para un producto LIMS, así como sus objetivos y pasos a seguir para poder aplicar las mismas.
- Buscar información referente a las normas ISO y FDA que deben de cumplir estos diseños de pruebas que se utilizaran para un producto LIMS.
- Analizar los aspectos teóricos y conceptuales de los diseños de prueba.
- Analizar los casos de uso del sistema del LIMS, así como los requisitos funcionales del sistema.

El trabajo se encuentra **estructurado** de la siguiente forma:

Capítulo1: Fundamentación teórica sobre las pruebas de calidad.
Incluye el estado del arte del tema tratado, a nivel internacional, nacional y de la Universidad, las tendencias, técnicas, tecnologías, metodologías y software usados en la actualidad para solucionar nuestro problema planteado.

Capítulo2: Catálogo de ideas y artefactos que se generan en el proceso de pruebas de calidad de software.
Se encuentra enmarcado fundamentalmente en los artefactos e ideas que se generan en todo el flujo de trabajo de un ingeniero de prueba, las funciones y las responsabilidades que debe de tomar cada miembro del grupo de prueba y el historial de revisiones.

Capitulo3: Propuesta de pruebas para un producto LIMS.
Aborda de forma general los tres enfoques principales de pruebas (caja blanca, caja negra y aleatoria), la necesidad de probar, las pruebas y técnicas de pruebas específicas que proponemos concretar para lograr un producto LIMS de calidad.

Capítulo I: Fundamentación Teórica.

1.0 Introducción

El presente capítulo tiene como objetivo exponer los fundamentos teóricos generales que sirven de punto de partida para determinar las pruebas de calidad necesarias para el LIMS del CIGB. En él se definen algunos conceptos que serán de utilidad en la comprensión de este trabajo y específicamente la propuesta de solución, también se hace alusión a algunas de las tecnologías, metodologías y herramientas que se usan en la actualidad en el área de la calidad de software y específicamente en el desarrollo de casos de pruebas para productos LIMS que implementa programas estrictos de Aseguramiento de la Calidad los cuales permiten cumplir con los imperativos legales impuestos por las Agencias Reguladoras y los requisitos de calidad definidos a nivel interno y por los propios consumidores finales del producto.

1.1 Estado del Arte

El tema de la calidad del software a tomado mucha fuerza en los últimos años, teniendo en cuenta el gran número de desventajas que puede proporcionar la realización de un producto software sin la calidad requerida, provocando que en el mundo hayan surgido un sin número de metodologías, procedimientos y estándares para el análisis, diseño, programación y prueba del software que permitan uniformar la filosofía de trabajo dirigidos al control y a la eficiencia de estos productos.

Las necesidades en Calidad del Software en los próximos años superan el ámbito convencional. Siempre se ha considerado un entorno, hoy casi superado, compuesto por el técnicas de medición y dictámenes producidos, el cumplimiento de los estándares de creación de software, control de cambios, estrategias de prueba, las revisiones formales, la aplicación de métodos y herramientas y los medios de gestión de la Calidad del Software.

Las nuevas perspectivas en la orientación a objetos, mejora de procesos de software, el estudio de componentes, la medición del software, el desarrollo y mantenimiento de software, la gestión y sistemas de calidad de software, normativa y certificación ... Internet, Intranet, plantean nuevos retos futuros y una constante actualización de los medios actuales. En cualquier caso siempre tendremos que luchar con la

evaluación de los niveles de concordancia del software, lo que plantea la búsqueda constante de nuevos medios de medición de las especificaciones. Es decir, la medición en los distintos ámbitos de la creación del software se plantea como el gran reto.

En cuanto al trabajo que se ha venido desempeñando en nuestro país podemos decir que ha sido satisfactorio aunque debe mejorar mucho más, se ha logrado un alto nivel en la calidad de los productos ,aplicando pruebas de caja negra y en ocasiones a algún software pruebas de caja blanca en algunos lugares como DESOFT, con diferentes software que nos permiten la revisión de gran parte de los códigos de los productos lanzados al mercado, ya específicamente en la UCI se ha trabajado muy fuerte en la organización y aplicación de estas pruebas a los software pero todavía falta mucho camino por andar ,solo se esta trabajando con las pruebas de caja negra ya que no se cuenta con la tecnología, ni la preparación necesaria para aplicar las pruebas de caja blanca ,las cuales si ya son específicamente al código, esta ultima no se ha podido aplicar por varios factores como la falta de recursos tecnológicos y herramientas de trabajo ,así como la necesidad de organizar una misma línea de trabajo en las facultades a la hora de desarrollar los productos.

1.2 Definición de calidad del software

La calidad del software es el conjunto de cualidades que lo caracterizan y que determinan su utilidad y existencia. La calidad es sinónimo de eficiencia, flexibilidad, corrección, confiabilidad, mantenibilidad, portabilidad, usabilidad, seguridad e integridad.

La calidad del software es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario". (10)

La calidad del software es medible y varía de un sistema a otro o de un programa a otro. Un software elaborado para el control de naves espaciales debe ser confiable al nivel de "cero fallas"; un software hecho para ejecutarse una sola vez no requiere el mismo nivel de calidad; mientras que un producto de software para ser explotado durante un largo período (10 años o más), necesita ser confiable, mantenible y flexible para disminuir los costos de mantenimiento y perfeccionamiento durante el tiempo de explotación.

1.3 ¿Cómo obtener un software de calidad?

La obtención de un software con calidad implica la utilización de metodologías o procedimientos estándares para el análisis, diseño, programación y prueba del software que permitan uniformar la filosofía de trabajo, en aras de lograr una mayor confiabilidad, mantenibilidad y facilidad de prueba, a la vez que eleven la productividad, tanto para la labor de desarrollo como para el control de la calidad del software.

La política establecida debe estar sustentada sobre tres principios básicos: tecnológico, administrativo y ergonómico.

- El principio tecnológico define las técnicas a utilizar en el proceso de desarrollo del software.
- El principio administrativo contempla las funciones de planificación y control del desarrollo del software, así como la organización del ambiente o centro de ingeniería de software.
- El principio ergonómico define la interfaz entre el usuario y el ambiente automatizado. La adopción de una buena política contribuye en gran medida a lograr la calidad del software, pero no la asegura. Para el aseguramiento de la calidad es necesario su control o evaluación. (7)

1.4 ¿Cómo controlar la calidad del software?

Para controlar la calidad del software es necesario, ante todo, definir los parámetros, indicadores o criterios de medición, ya que, como bien plantea Tom De Marco, "usted no puede controlar lo que no se puede medir".

Las cualidades para medir la calidad del software son definidas por innumerables autores, los cuales las denominan y agrupan de formas diferentes. Por ejemplo, John Wiley define métricas de calidad y criterios, donde cada métrica se obtiene a partir de combinaciones de los diferentes criterios. La Metodología para la evaluación de la calidad de los medios de programas de la Comisión de Investigaciones Científicas (CIC), de Rusia, define indicadores de calidad estructurados en cuatro niveles jerárquicos: factor, criterio, métrica, elemento de evaluación, donde cada nivel inferior contiene los indicadores que conforman el nivel precedente. Otros autores identifican la calidad con el nivel de complejidad del software y definen dos categorías de métricas: de complejidad de programa o código, y de complejidad de sistema o estructura.

Ciertamente todos los autores coinciden en que el software posee determinados índices medibles que son las bases para la calidad, el control y el perfeccionamiento de la productividad.

Una vez seleccionados los índices de calidad, se debe establecer el proceso de control, que requiere los siguientes pasos:

- Definir el software que va a ser controlado: clasificación por tipo, esfera de aplicación, complejidad, entre otros, de acuerdo con los estándares establecidos para el desarrollo del software.
- Seleccionar una medida que pueda ser aplicada al objeto de control. Para cada clase de software es necesario definir los indicadores y sus magnitudes.
- Crear o determinar los métodos de valoración de los indicadores: métodos manuales como cuestionarios o encuestas estándares para la medición de criterios periciales y herramientas automatizadas para medir los criterios de cálculo.
- Definir las regulaciones organizativas para realizar el control: quiénes participan en el control de la calidad, cuándo se realiza, qué documentos deben ser revisados y elaborados, son algunas de ellas (7)

1.5 Factores que determinan la calidad del software

Se clasifican en tres grupos:

1. Operaciones del producto: características operativas

– Corrección (¿Hace lo que se le pide?)

• El grado en que una aplicación satisface sus especificaciones y consigue los objetivos encomendados por el cliente

– Fiabilidad (¿Lo hace de forma fiable todo el tiempo?)

• El grado que se puede esperar de una aplicación lleve a cabo las operaciones especificadas y con la precisión requerida

– Eficiencia (¿Qué recursos hardware y software necesito?)

• La cantidad de recursos hardware y software que necesita una aplicación para realizar las operaciones con los tiempos de respuesta adecuados

– Integridad (¿Puedo controlar su uso?)

• El grado con que puede controlarse el acceso al software o a los datos a personal no autorizado

– Facilidad de uso (¿Es fácil y cómodo de manejar?)

• El esfuerzo requerido para aprender el manejo de una aplicación, trabajar con ella, introducir datos y conseguir resultados

2. Revisión del producto: capacidad para soportar cambios

– Facilidad de mantenimiento (¿Puedo localizar los fallos?)

• El esfuerzo requerido para localizar y reparar errores

– Flexibilidad (¿Puedo añadir nuevas opciones?)

• El esfuerzo requerido para modificar una aplicación en funcionamiento

– Facilidad de prueba (¿Puedo probar todas las opciones?)

• El esfuerzo requerido para probar una aplicación de forma que cumpla con lo especificado en los requisitos

3. Transición del producto: adaptabilidad a nuevos entornos

– Portabilidad (¿Podré usarlo en otra máquina?)

• El esfuerzo requerido para transferir la aplicación a otro hardware o sistema operativo

– Reusabilidad (¿Podré utilizar alguna parte del software en otra aplicación?)

• Grado en que partes de una aplicación pueden utilizarse en otras aplicaciones

– Interoperabilidad (¿Podrá comunicarse con otras aplicaciones o sistemas informáticos?

• El esfuerzo necesario para comunicar la aplicación con otras aplicaciones o sistemas Informático.

Son las técnicas y actividades de carácter operativo, utilizadas para satisfacer los requisitos relativos a la calidad, centradas en dos objetivos fundamentales:

– mantener bajo control un proceso

– eliminar las causas de los defectos en las diferentes fases del ciclo de vida

• En general son las actividades para evaluar la calidad de los productos desarrollados. (11)

1.6 Sobre la medición de la calidad del software

Es posible medir los principales atributos que forman o caracterizan a un software de buena calidad. La idea aquí es que la calidad del software se caracteriza por ciertos atributos: fiabilidad, flexibilidad,

robustez, comprensión, adaptabilidad, modularidad, complejidad, portabilidad, usabilidad, reutilización, eficiencia...., y que es posible medir cada uno de ellos, y por consiguiente, caracterizar o medir la calidad del software en cuestión.

Para cada atributo a medir, hay una medición confiable (objetiva) que puede llevarse a cabo. La idea es que, dado que el atributo deseable es difícil de medir, se mide otro atributo que está correlacionado con el primero. (12)

1) La fiabilidad (software confiable, pocos errores) se mide a través del número de mensajes de error, mientras más haya, contiene menos errores este software.
2) La flexibilidad (capacidad de adaptación a diferentes tipos de uso, a diferentes ambientes de uso) o adaptabilidad se mide viendo a cuántos estándares este software se adhiere.
3) La robustez (pocas fallas catastróficas, el sistema “no se cae”) se mide a través de pruebas y uso prolongado.
4) La comprensión (capacidad de entender lo que el sistema hace) se mide viendo la cantidad de comentarios que posee el software, y la extensión de sus manuales.
5) El tamaño de un programa se mide en bytes, el espacio que ocupa en memoria.
(Esta medición no tiene objeción, se mide lo que se quiere medir).
6) La eficiencia de un programa se mide en segundos, es la rapidez en su ejecución.
(Esta medición no tiene objeción, se mide lo que se quiere medir).
7) La modularidad de un programa se mide contando el número de módulos que lo forman.
8) La complejidad de un programa (qué tan fácil es entender el código) se mide contando el número de anidaciones en expresiones o postulados (“complejidad ciclomática”).
9) La portabilidad de un programa es la facilidad con que se eche a andar en otro sistema operativo distinto al que fue creado. Se mide preguntando a usuarios que han hecho estos trabajos.
10) La usabilidad de un programa es alta cuando el programa aporta gran valor agregado a nuestro trabajo. “Es indispensable contar con él”. Se mide viendo qué porcentaje de nuestras necesidades cubre ese programa.
11) La reutilización de un programa se mide por la cantidad de veces que partes del mismo se han reutilizado en otros proyectos de desarrollo de software.

12) La facilidad de uso (ergonomía) caracteriza a los programas que no cuesta trabajo aprender, que se amoldan al modo intuitivo de hacer las cosas. Se mide observando la cantidad de pantallas que interaccionan con el usuario, y su sofisticación.

1.6.1 Medir el proceso, en vez de medir el producto

En vez de medir la calidad del producto, midamos la calidad del proceso. Tener un buen proceso implica producir software de buena calidad.

Es tentador definir que un proceso produce software de buena calidad, y entonces medir las desviaciones que los programadores hacen con respecto de tal proceso. El problema es que no se sabe cómo deducir cuál proceso producirá buena calidad en el software.

Por ejemplo, si quiero hacer un software transportable, ¿qué proceso debo implantar, versus si lo que deseo es enfatizar la facilidad de uso del software?

Entonces la definición del proceso se vuelve muy subjetiva, un acto de fe. Se recurre a procesos que suenan o se ven razonables, o que han sido ensayados en otros lados con éxito. O que están dados por algún estándar o comité internacional. “Si tanta gente los usa, deben ser buenos.” Hay que reconocer que nuestra disciplina aún no es una ciencia o ingeniería o desarrollo sistemático, donde se puede deducir un proceso que garantice ciertas cualidades en el producto resultante. Este más bien es un arte o artesanía, donde cuenta “la inspiración”, “ver cómo lo hicieron los demás”, ciertas costumbres o tics que un programador le copió (inconscientemente, quizá) a su maestro.

1.7 Pruebas

Es la actividad en la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas donde los resultados se observan y registran y se realiza una evaluación de algún aspecto". Para Myers [MYERS, 1979], probar (o la prueba) es el "proceso de ejecutar un programa con fin de encontrar errores". El nombre "prueba", además de la actividad de probar, se puede utilizar para designar "un conjunto de casos y procedimientos de prueba". (9)

1.8 Caso de prueba

Es el conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular como, por ejemplo, ejercitar un camino concreto de un programa o verificar el

cumplimiento de un determinado requisito. También se puede referir a la documentación en la que se describen las entradas, condiciones y salidas de un caso de prueba. (9)

1.9 Enfoques de diseños de pruebas

Existen tres enfoques principales para el diseño de casos:

1.- El enfoque estructural o de caja blanca.

2.- El enfoque funcional o de caja negra.

3.- El enfoque aleatorio consiste en utilizar modelos en muchas ocasiones estadísticos) que representen las posibles entradas al programa para crear a partir de ellos los casos de prueba.

1.10 Niveles de pruebas

Tipos de Pruebas	Descripción
Unitarias	Su objetivo es probar el comportamiento de cada uno de los componentes de forma independiente. Este tipo de prueba deberá realizarse una vez sea implementado el componente. Se prueba la funcionalidad de una clase o conjunto de clases correlacionadas
Integración	Su objetivo es probar la unión de los componentes del sistema, una vez estos hayan rebasado las pruebas unitarias se deberá verificar que estos interactúan correctamente a través de las funcionalidades expuestas en sus interfaces, se verifican las interfaces entre las partes de una arquitectura. Este tipo de prueba deberá realizarse durante la fase de construcción, una vez sea implementado el componente.
Sistema	Su objetivo es comprobar la funcionalidad de todo el sistema
Implantación	Su objetivo es la comprobación de cualquier detalle de diseño interno hasta aspectos como las comunicaciones. Se deben verificar fundamentalmente los requisitos no funcionales definidos para el producto como por ejemplo, comprobar que el sistema puede gestionar los volúmenes de información requeridos, se ajusta a los tiempos de

	respuesta deseados y que los procedimientos de respaldo, seguridad e interfaces con otros sistemas funcionan correctamente. Se debe verificar también el comportamiento del sistema bajo las condiciones más extremas.
Aceptación	Su objetivo es el de verificar formalmente con el cliente que el sistema satisface todas sus necesidades.
Regresión	Es un conjunto de pruebas que se ejecutan nuevamente. Su objetivo es verificar que cambios ocurridos en un componente del sistema no provoca errores adicionales en otros componentes del mismo que ya han sido probados.

1.11 Modelos de Pruebas (TESTING)

Prueba es el proceso de ejecutar un programa con la intención de encontrar defectos, es como un proceso destructivo que determina el diseño de casos prueba y la asignación de responsabilidades.
Un test exitoso es aquél que encuentra muchos defectos, no lo opuesto. Además, un caso de prueba bueno es el que tiene una alta probabilidad de detectar un defecto aún no descubierto. (13)

Los pasos del proceso de pruebas son:
Planificación.- Al comenzar el desarrollo se establecen guías, métodos, estimaciones de recursos requeridos y estándares involucrados.
Identificación.- Se realiza una estimación más detallada de los recursos requeridos.
Especificación.- Es la descripción de las pruebas a nivel funcional (propósito) y a nivel detallado (ejecución paso a paso).
Ejecución.- Comprende el desarrollo de las pruebas tanto automatizadas como manuales, registrando los resultados.
Análisis de defectos.- Se identifican los defectos y sus probables causas, también se planean acciones correctivas.
Consumación.- Se registra la documentación y se prepara el equipo y los casos de prueba para uso posterior.

Defectos

La importancia de un defecto depende de la frecuencia, el costo de corrección, el costo de instalación y sus consecuencias. No hay manera universalmente correcta de dividir los defectos en categorías, las siguientes son típicamente utilizadas:

- Requerimientos.
- Características y funcionalidad.
- Estructura.
- Datos.
- Implementación y código.
- Integración.
- Arquitectura del sistema y del software.
- Definición y ejecución de pruebas.

De esta forma, los defectos se detectan examinando estructuras y diseños externos, interfaces del usuario, objetivos de diseño, requerimientos de los usuarios y ejecutando código.

Casos de prueba

Un buen caso de prueba es aquél que tiene una alta probabilidad de encontrar un defecto no descubierto, no aquél en el que el programa funciona correctamente.

El diseño de un buen caso de prueba generalmente es difícil; existen dos estrategias para su diseño:

- Pruebas de caja blanca: Se pretende un examen en detalle de los procedimientos internos, ejecutando caminos lógicos con condiciones o ciclos específicos.
- Pruebas de caja negra: Las pruebas se conducen a nivel de la interfaz, sin mayor interés por los detalles internos de la estructura lógica.

Las estrategias de Pruebas de caja blanca y Pruebas de caja negra pueden combinarse para proveer un enfoque que valide la interfaz y selectivamente asegure que los aspectos internos son los correctos.

Prueba de caja blanca

Esta estrategia utiliza la estructura de control del diseño de un procedimiento para derivar casos de prueba que garanticen que todos los caminos independientes dentro de un módulo han sido ejercitados por lo menos una vez, ejecutando todas las decisiones lógicas y ciclos presentes.

Entonces, se compone de cuatro pruebas de estructuras de control: Pruebas de caminos básicos, de condiciones, de ciclos y de flujo de datos.

Pruebas de caminos básicos.- Permiten derivar una medida de complejidad lógica del diseño de un procedimiento y usarla como guía para definir un conjunto básico de caminos de ejecución.

Los casos de prueba derivados para ejercitar el conjunto básico garantizan la ejecución de cada instrucción en el programa al menos una vez durante las pruebas.

Pruebas de condiciones.- Son métodos de diseño de casos de prueba que verifican cada condición o decisión en un programa. Los tipos de errores pueden ser: un operador booleano, una variable booleana, un operador relacional, etc.

Pruebas de ciclos.- Esta técnica se enfoca solamente a verificar los ciclos que se encuentren en el programa, identifica cuatro tipos de ellos: simples, concatenados, anidados y no estructurados. Para cada uno de ellos actúa de manera distinta y sugiere el rediseño de los ciclos no estructurados.

Prueba de flujo de datos.- Es un método de diseño de casos de prueba que selecciona caminos en un programa de acuerdo a la ubicación de las definiciones y uso de las variables en él. Dado que las instrucciones de un programa están relacionadas entre sí de acuerdo a las definiciones, el enfoque de flujo de datos es muy efectivo, aunque la selección de caminos es más compleja.

Prueba de caja negra

Los siguientes métodos son los más comúnmente utilizados:

Partición de equivalencias.- Es un proceso sistemático que identifica un conjunto de pruebas representativas de grandes conjuntos de otras pruebas posibles; la idea es que el producto bajo prueba se

comporte de la misma manera para todos los miembros de la clase. Sus pasos son la identificación de clases de equivalencia y la identificación de los casos de prueba.

Análisis de valores frontera.- Es una variante de la partición de equivalencias en que, en vez de seleccionar cualquier elemento como representativo de una clase de equivalencia, se seleccionan los bordes de una clase. Además, se definen clases de equivalencia de salida y no sólo de entrada como en el caso anterior.

Adivinanza de defectos.- Es un enfoque basado en intuición y experiencia, para identificar pruebas que probablemente expondrán defectos. La idea es formar una lista con los posibles defectos o situaciones propensas a error, y desarrollar pruebas de acuerdo a esa lista.

Aquí hay que probar con posibilidades como strings vacíos o nulos, así como con números negativos y el cero, que tienden a salir de las convenciones de los números positivos.
Algunos métodos no tan usados son:

Diagramas de causa-efecto.- Es un enfoque sistemático para seleccionar conjuntos de casos de prueba de alto rendimiento que exploran combinaciones en las condiciones de entrada. Se registran restricciones describiendo combinaciones de causas y efectos posibles, también se crean grafos de causa-efecto y se convierten en tablas de decisiones.
Pruebas sintácticas.- Es un enfoque sistemático para generar datos de entrada a un programa válidos e inválidos, aplicable a programas que tienen un lenguaje “oculto”, pero usualmente inefectivo para lenguajes explícitos y desarrollados formalmente.
Matriz de grafos.- Es una representación más simple de un grafo para organizar los datos.

Estrategias de pruebas

Una estrategia de prueba debe incorporar planificación, diseño de casos de prueba, ejecución, recolección de datos y evaluación, siguiendo un orden.
Algunas características generales de estrategia para el proceso de prueba son:

- Las pruebas son conducidas por el desarrollador del software y un grupo independiente.
- Comenzar en el nivel más bajo y trabajar hacia fuera hasta integrar completamente el sistema.

Inicialmente las pruebas se focalizan en cada módulo individualmente, asegurando que funciona de manera apropiada como una unidad, aquí las técnicas de White-box son más importantes; después, los módulos deben ensamblarse para formar un paquete completo, aquí las técnicas de Black-box son más importantes.

- El debugging es reconocido como una actividad aparte y no es mezclado con el proceso de prueba.

Para una implementación exitosa de una estrategia de prueba se debe:

-Especificar requerimientos del producto en forma cuantificable mucho antes de comenzar la prueba.
- Establecer explícitamente los objetivos de la prueba.
- Comprender a los usuarios y desarrollar un perfil para cada categoría.
- Desarrollar un plan que especifique pruebas de ciclo rápido.
- Usar revisiones formales como un filtro previo de prueba.
- Conducir revisiones técnicas para evaluar los casos de prueba y la estrategia de prueba.
- Desarrollar un enfoque de mejora continua para pruebas.

La prueba unitaria consiste en someter a pruebas un solo módulo de código.

La prueba de integración es una técnica sistemática para construir la estructura lógica del programa mientras se conducen pruebas para descubrir defectos asociados a las interfaces, para lo que se toman módulos probados (prueba unitaria) y se construye una estructura de programa de acuerdo al diseño.
Cualquier estrategia de integración debe ser incremental.

La prueba de regresión es la actividad que ayuda a asegurar que los cambios no introduzcan comportamientos no pretendidos o defectos adicionales. Los casos de prueba para regresión corresponden a tres clases diferentes:

- Un conjunto representativo de pruebas para ejecutar todas las funciones del software.
- Pruebas adicionales que se enfocan en funciones probablemente afectadas por un cambio.

- Pruebas que se enfocan a los componentes que han cambiado.

La prueba de usabilidad involucra el hacer que los usuarios trabajen con un producto de software y observar sus respuestas a él.
Históricamente ha sido uno de los muchos componentes de las pruebas de sistema, pero la tendencia es hacia darle un rol más importante.

Las características a probar son:

- Accesibilidad: Puede el usuario entrar, salir, navegar con facilidad…
- Respuesta: Puede el usuario hacer lo que desea, cuando lo desea y en forma clara…
- Eficiencia: Puede hacerlo con un mínimo de recursos…
- Comprensión: Entiende el usuario la estructura, ayudas y documentación…

La prueba de validación o aceptación tiene como objetivo determinar si el funcionamiento satisface al cliente o usuario, se compone de pruebas alfa y beta.

Pruebas alfa.- Son conducidas en los laboratorios del desarrollador para el usuario final, el programa está generalmente incompleto, pero dispone de cierta funcionalidad que le permite ser probado.
Pruebas beta.- Son conducidas en ambiente de trabajo reales por usuarios finales, las pruebas se desarrollan antes de sacar el producto al mercado. Una práctica que viene siendo muy popular es distribuir en forma gratuita una versión no final del producto para que sean los propios consumidores los que la prueben.

Una novedosa forma de prueba es llamada prueba online, la que considera derivar un test de un programa modelo y la ejecución del mismo test en un solo algoritmo.

Mediciones de prueba

Son mediciones de cobertura de las pruebas que proveen una evaluación cuantitativa de su alcance y dinamismo. Dan una idea de la complejidad real de un programa, ayudando a planear el esfuerzo de

prueba, a predecir cuántos errores existen y dónde pueden ocurrir. Las mediciones del número y tipo de defectos detectados proveen una idea precisa de la eficiencia en la verificación y las debilidades del proceso de desarrollo.

Las mediciones también:
- Proveen un indicador importante de calidad del producto.
- Sirven para predecir el número de errores que quedan.
- Proveen información sobre la calidad del producto durante las pruebas.

1.12 Herramientas de pruebas

La prueba es una fase del proceso "muy cara". Existen herramientas automáticas de prueba (**testing workbenches**) que permiten reducir el tiempo requerido para las pruebas y el coste total asociado.

Las herramientas tales como **Checking, Test, JTest 8.0** eliminan la complejidad de las pruebas. Estas herramientas de prueba están orientadas al modelado y al diseño de las aplicaciones, y al análisis estático del código, dicha herramienta en la que interviene en la monitorización del proceso de desarrollo del software y sus resultados cubren las necesidades de las organizaciones que se desean controlar la calidad del software antes de llegar a manos del cliente**.** Para ello la herramienta sigue un modelo de métricas en el que se integran medidas obtenidas automáticamente del proceso de desarrollo (actividad, requisitos, defectos y cambios) y de elementos analizables del software: código fuente, documentación del proyecto, scripts de construcción, y scripts de pruebas.

TEST es una herramienta líder en automatización de pruebas unitarias y de productos con código estándar, trabaja sobre clases escritas en la plataforma Microsoft .NET, sin requerir que los desarrolladores realicen un solo caso de prueba. Mejorara la fiabilidad, funcionalidad, seguridad, desarrollo y mantenimiento de las aplicaciones .NET. Puede probar cualquier fichero o paquete que haya sido construido utilizando el CLR de .NET. Facilita la creación y ejecución de pruebas definidas por el usuario basado en el framework de NUnit.

JTEST 8.0 brinda avances tecnológicos en la realización de pruebas, para ayudar a los equipos a verificar de manera automática la funcionabilidad de aplicaciones cada vez más complejas, en empresas con sistemas en permanente cambio (J2EE, servicios de SOA/Web, permite reducir el tiempo de entrega del software así como una disminución del riesgo de generar software defectuoso o con problemas de vulnerabilidad. Para reducir la complejidad de las pruebas, Jtest ofrece la generación y la ejecución automatizada de los casos de prueba a través de la simulación de un entorno real, posibilita una prueba en tiempo de ejecución que permite la temprana detección de defectos en el código que de otro modo pasarían inadvertidos hasta etapas avanzadas de desarrollo como el aseguramiento de la calidad.

La herramienta Jtest posee diferentes característica tales como:

- Prueba métodos individuales, clases, o grandes y complejas aplicaciones.
- Soporta Struts, spring, Hibernate, EJBs, JSPs, servlets.
- Genera casos de prueba funcionales JUnit que capturan el comportamiento real del código.
- Genera casos de prueba JUnit y Cactus que muestran la fiabilidad y capturan el comportamiento.
- Parametriza los casos de prueba para usarlos con múltiples valores (generados en tiempo de ejecución, definidos por el usuario o provenientes de otras. fuentes de datos).

1.13 Metodologías más utilizadas para el desarrollo de software a nivel mundial.

Una metodología de desarrollo no es más que un conjunto de actividades que tienen un orden lógico y que garantizan que al finalizar la última actividad de la metodología se tiene el software que se comienza a analizar en la primera actividad con la utilización de herramientas y técnicas.

Cuando se va a llevar a cabo la realización de un producto debemos utilizar diferentes metodologías de desarrollo de software con el fin de poder garantizar la calidad del software.

En el mundo existen tres metodologías fundamentales para el desarrollo de software:

XP (Extreme Programming), MSF (Microsoft Solution Framework) y RUP (Rational Unified Process), (14)

1.13.1 Metodología XP (Extreme Programming)

Es una de las metodologías de desarrollo de software más exitosa, utilizada para proyectos de corto plazo, y de pocos integrantes en el equipo, cuyo plazo de entrega era “ayer”. La metodología consiste en

una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto.

1.13.2 Metodología MSF (Microsoft Solution Framework)

Esta es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.

1.13.3 RUP (Rational Unified Process)

El "Proceso Unificado" es el resultado final de tres décadas de desarrollo y uso práctico. Esta es una de las causas que conlleva a que sea la metodología que mejor se ajusta a las necesidades que existen actualmente en el desarrollo de software, pues propone un Modelo iterativo e incremental, centrado en la arquitectura y guiado por casos de uso muy acorde con la naturaleza cambiante de los requisitos en muchos proyectos. Es una metodología que está totalmente respaldada por una excelente herramienta CASE: Rational Rose, con la que ya se tiene experiencia.

Esta metodología se divide en 4 **fases**:

- Inicio: El Objetivo en esta etapa es determinar la visión del proyecto.
- Elaboración: En esta etapa el objetivo es determinar la arquitectura óptima.
- Construcción: En esta etapa el objetivo es llevar a obtener la capacidad operacional inicial.
- Transmisión: El objetivo es llegar a obtener el reléase del proyecto.

1.14 Aspectos fundamentales relacionados con un LIMS

- Servicios de validación y de cumplimiento normativo de LIMS

Los especialistas profesionales de validación y cumplimiento normativo de Thermo combinan la experiencia en el trabajo de laboratorio con un profundo conocimiento de los requisitos normativos, lo cual les permite ayudar a las empresas de sectores regulados a enfrentarse a las cuestiones de validación del

LIMS y de cumplimiento de la CFR 21 Parte 11. Los servicios incluyen la consultoría y la planificación sobre validación, el desarrollo y la ejecución de pruebas, y la formación en validación. Thermo ofrece un enfoque de aplicación integrada de LIMS con una metodología de validación bien definida basada en las buenas prácticas de fabricación automatizada. (15)

- Una solución flexible y graduable

La integración de LIMS con los instrumentos de análisis no sólo automatiza las funciones del laboratorio, sino que también proporciona ventajas a la hora de compartir datos, que añaden valor a la inversión informática.

A los productos tradicionales de integración de instrumentos diseñados para el funcionamiento en un único PC les cuesta soportar los sistemas en red de hoy en día, que requieren unas soluciones más flexibles y graduables.

Entre los muchos beneficios obtenidos por la introducción de un LIMS están:

- . La revisión y visualización de datos más completa, flexible y accesible con una generación más rápida y efectiva de informes.
- Acceso definido y controlado a la información (muy difícil de llevar a cabo con métodos manuales).
- Permite habilidad en la definición, modificación y uso de Procedimientos Normalizados de Trabajo, estándares de calibración, procedimientos analíticos, etc.
- Obtención de resultados calculados automáticamente y verificación en la validez de los datos introducidos.
- Reducción de errores de trascripción por adquisición automática de información y eliminación de copias redundantes de información en papel.
- Validación de resultados obtenidos frente a Especificaciones de Calidad.
- Revisión y análisis más eficiente de la información eliminando procedimientos de búsqueda y recopilación muy costosos en tiempo y con claras fuentes de errores.
- Transferencia automática de información a otros Sistemas de la Entidad en cuestión.
- Uso más efectivo de los recursos humanos y técnicos del Laboratorio.

Un LIMS ayuda en todas las tareas asociadas con la gestión de las muestras y la información relativa a las mismas. La mayor parte de las empresas que disponen de un LIMS han estimado un incremento en la productividad entre 10-20%.

En la actualidad, la mayoría de la información y los documentos de un laboratorio se guardan electrónicamente en los ordenadores. Esta forma de almacenamiento en formato electrónico permite una manipulación de la información mucho mas fácil que en cualquier otro tipo de formato (papel, etc.). Sin embargo, la transferencia de dichos datos se hace difícil si no esta automatizada debidamente.

1.15 CMMI como parte del aseguramiento de la calidad de los productos de software

Las empresas productoras presentan modelos que son muy eficientes para velar por la calidad, como, CMMI (Capability Maturity Model® Integration) que estudia los procesos de desarrollo de software de una organización y produce una evaluación de la madurez de la organización estableciendo una escala de cinco niveles para lograr la madurez de la calidad del software, estos son definidos de la siguiente manera: (16)

Nivel 1 es el Inicial donde se debe establecer entre otras cosas un plan formal de Aseguramiento de la Calidad del Software así como la organización general del proyecto.

Para obtener nivel 2 de CMMI o Procesos Repetible (Gestionado) donde las pruebas hacen énfasis en las áreas de alto riesgo, el plan de Aseguramiento de la Calidad del Software y el plan del desarrollo del software son compatibles se deben realizar las siguientes tareas:

- Gestión de requisitos
- Planificación de proyectos
- Seguimiento y control de proyectos
- Gestión de proveedores
- Aseguramiento de la calidad
- Gestión de la configuración

Nivel 3: Proceso Definido es donde se trabaja en las Pruebas del Software que constituyen un elemento crítico para la calidad del software. Los procesos que hay que realizar para alcanzar este nivel en CMMI son:

- Desarrollo de requisitos
- Solución Técnica
- Integración del producto
- Verificación
- Validación
- Desarrollo y mejora de los procesos de la organización
- Definición de los procesos de la organización
- Planificación de la formación
- Gestión de riesgos
- Análisis y resolución de toma de decisiones

Nivel 4: Cuantitativamente Gestionado es el del Proceso administrativo

- Se puede seguir con indicadores numéricos (estadísticos) la evolución de los proyectos.
- Las estadísticas son almacenadas para aprovechar su aportación en siguientes proyectos.
- Los proyectos se pueden pedir cuantitativamente.

Nivel 5. Optimizado

Se basa en criterios cuantitativos se pueden determinar las desviaciones más comunes y optimizar procesos.

1.16 Conclusiones

Después de haber realizado un análisis de la información recopilada sobre temas tratados en el presente capitulo, como: qué es calidad de software, como garantizar la misma , qué son los diseños de casos de prueba así como los enfoques principales para el diseño de casos de pruebas entre otros conceptos y métodos, que son indispensables para el análisis y comprensión del proceso de diseño, así como el desarrollo de pruebas de calidad para productos LIMS(Sistemas de Gestión de la Información del Laboratorio (LIMS, del inglés Laboratory Information Management System), podemos determinar que para lograr la calidad y confiabilidad necesaria en un producto software y además que los clientes queden satisfechos, es necesario que los productos estén libres de errores, y esto trae consigo que los desarrolladores tomen conciencia de la influencia de este proceso en los resultados finales de los productos y la importancia del mismo para lograr competir en el mercado ya sea a nivel nacional o internacional.

Capítulo II: Catalogo de ideas y artefactos que se generan en el proceso de desarrollo de pruebas de calidad de software.

2.0 Introducción

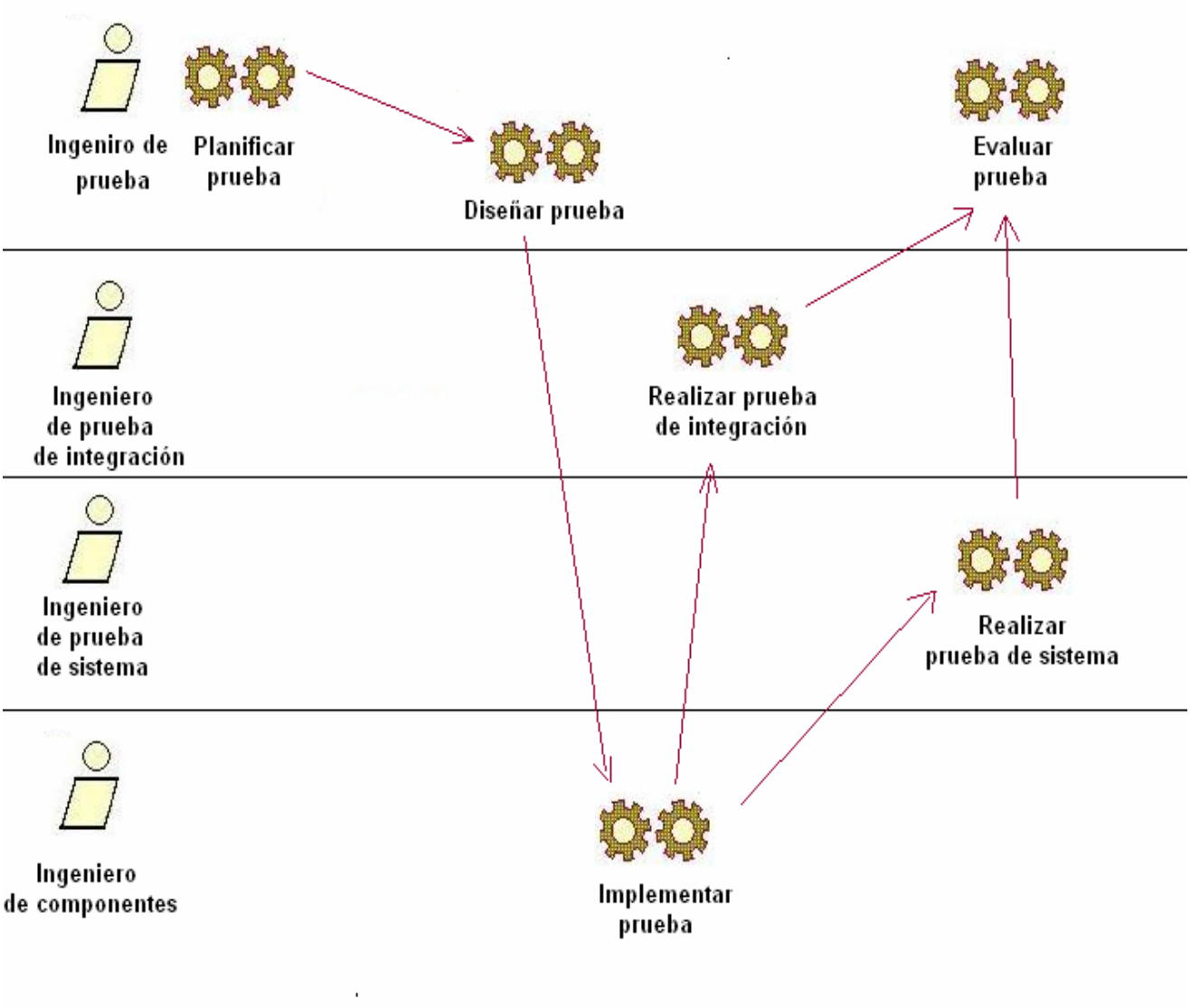
A lo largo de todo el ciclo de vida del producto de software, las personas involucradas en el equipo desde el inicio, se van dando cuenta de elementos funcionales y no funcionales que no deben dejar de probarse en su momento correspondiente. Para que esas ideas no se queden en el aire y de alguna manera se materialicen luego en alguno de los componentes del Modelo de Prueba, se hace necesario recogerlas de una manera sencilla y fácilmente comprensible por el ingeniero de pruebas.

Es por ello que se ha elaborado un artefacto llamado Catálogo de Ideas y artefactos de Prueba donde presentamos una hoja para el histórico de revisiones y la lista de distribución del documentos, y una hoja por cada una de las etapas del ciclo de vida del producto, donde se plasmarán las ideas de un modo sencillo, claro y fácil de comprender, haciendo alusión siempre a la iteración donde ha surgido la idea, el tipo de prueba al que pertenece (funcionalidad, integridad de la BD, seguridad, rendimiento, etc.) y una columna donde el Ingeniero de Prueba, luego de consultar con el Jefe de Proyecto su equipo, deberá consignar si la idea ha sido Aprobada para implementar en los Casos de Prueba, Rechazada o si ha encontrado otras ideas que sean similares o se encuentre esta Duplicada.

Es importante ir llenando el Catálogo a medida que surjan las ideas, pues todo el equipo es responsable por las pruebas del producto y por la calidad demostrada del mismo a la hora de la entrega al cliente.

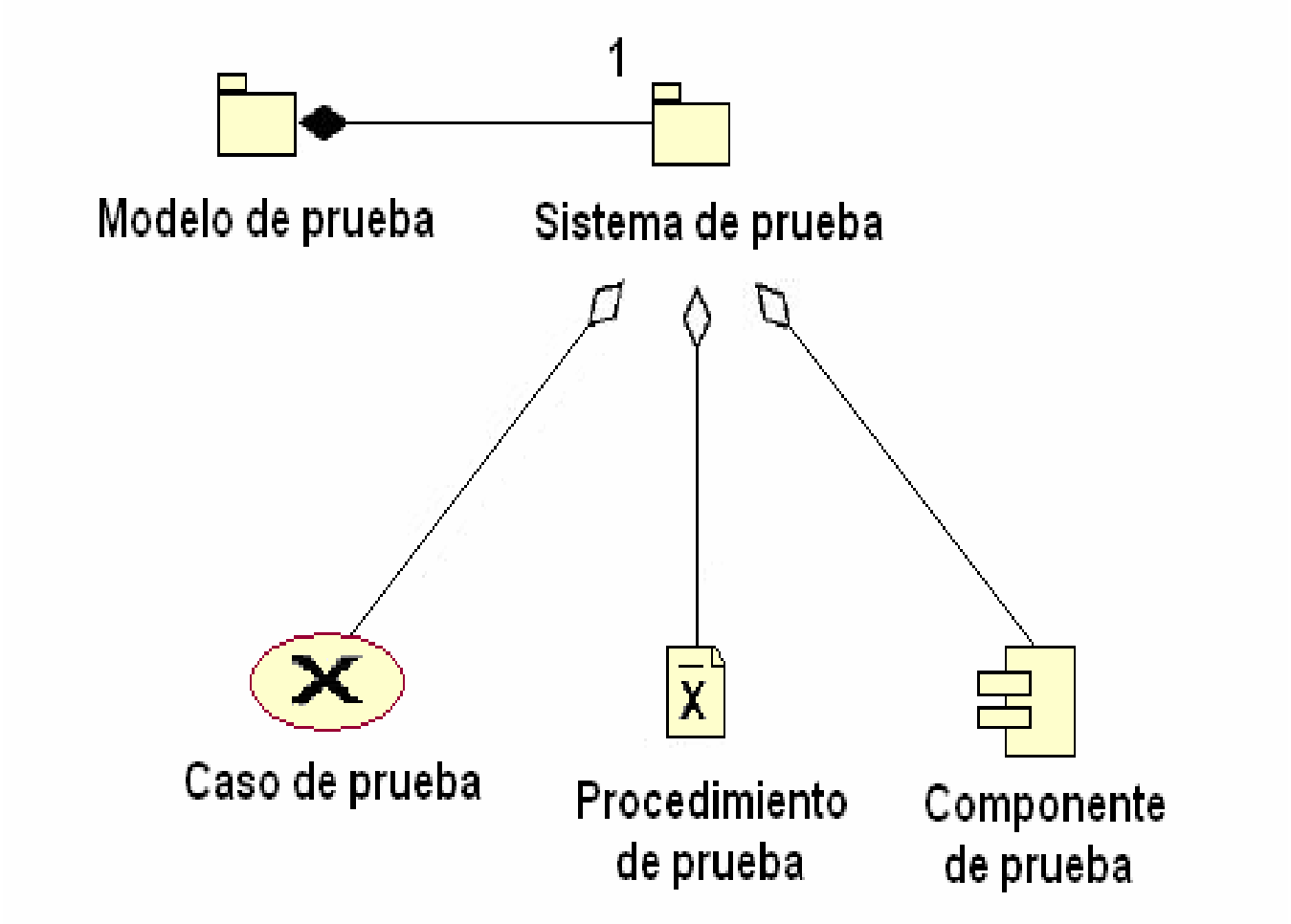
2.1 Flujo de Trabajo de Pruebas

(Este es el artefacto que recoge todo lo relacionado con el flujo de trabajo que se sigue en la aplicación de las pruebas de calidad de software, el mismo comienza cuando el ingeniero de pruebas planifica las pruebas, luego las diseña, para que acto seguido el ingeniero de componente las implemente, luego el ingeniero de prueba de integración realiza la prueba de integración y el ingeniero de pruebas del sistema realiza la prueba del sistema ,realizándose una evolución final de todo el proceso). (17)



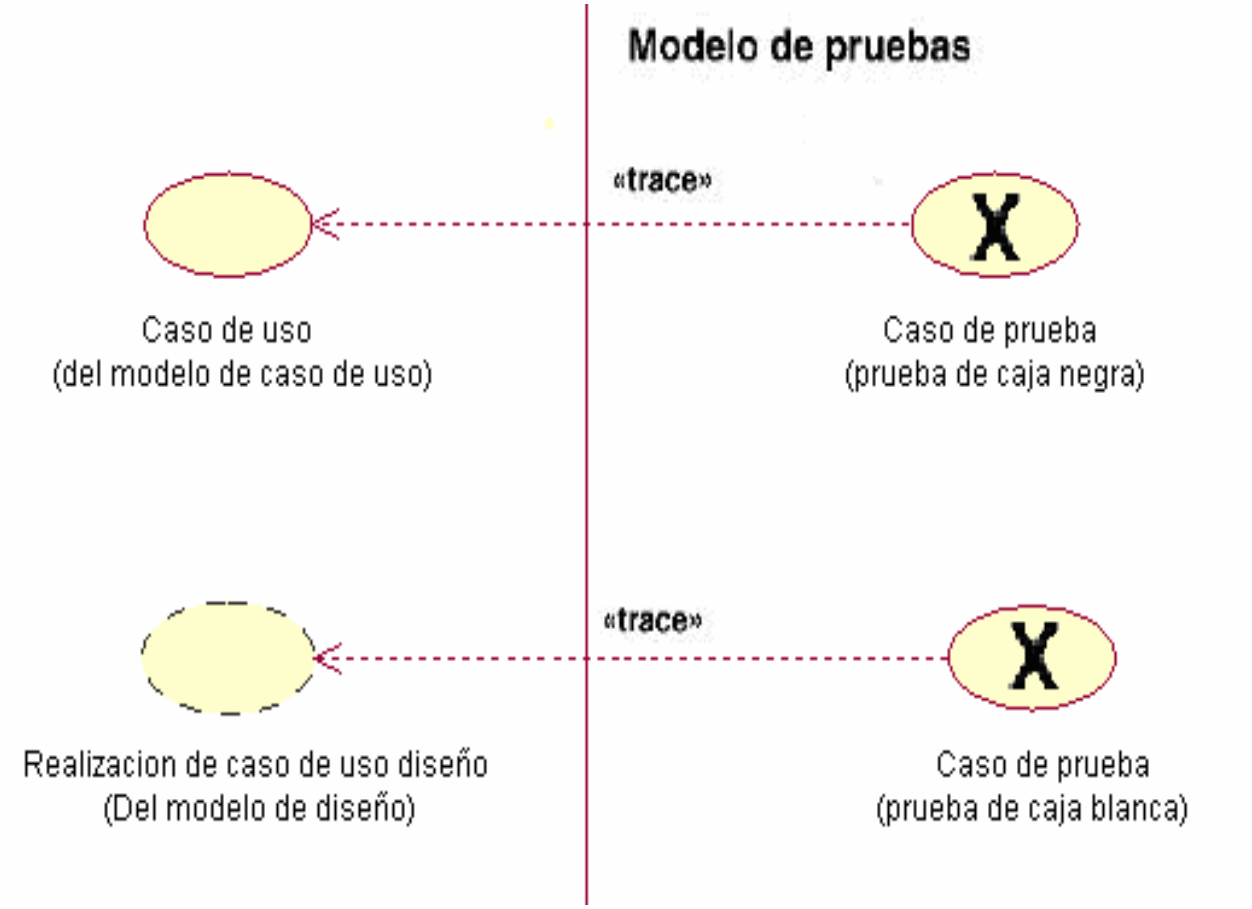
2.2 Modelo de Pruebas

Un Modelo de Pruebas no es más que una Colección de Casos de Prueba, Procedimientos de Prueba y Componentes de Prueba. Si es muy grande puede ser útil introducir paquetes para manejar su tamaño. (17)



2.3 Casos de Prueba

Los **Casos de Prueba** especifican una forma de probar el sistema, incluyendo los datos de entrada, resultados esperados y entorno en los cuales se debe probar (Sistema Operativo, Motor de BD, Framework, etc.). Un Caso de Prueba puede derivarse de, y por tanto seguir la traza de, un Caso de Uso del Modelo de Casos de Uso o de una realización de un Caso de Uso del Modelo de Diseño. (17)



En la práctica lo que se prueba puede venir dado por un requisito o una colección de requisitos del sistema cuya implementación justifica una prueba que es posible realizar y que no resulta muy costosa.

Casos de Prueba (CP) Comunes:

- Un CP que especifica cómo probar un Caso de Uso (CU) o un escenario específico de un CU. Este tipo de CP incluye la verificación del resultado de la interacción entre los actores y el sistema, que satisfaga pre – condiciones y post – condiciones especificadas por el CU y que se sigue la secuencia de acciones especificada en el mismo. Obsérvese que este tipo de CP basado en un CU funciona típicamente como una “Caja Negra”, o sea, verifica un comportamiento observable externamente del sistema.

- Un CP que especifica cómo probar una realización de un CU de diseño o un escenario específico de esta realización. Un CP de este tipo puede incluir la verificación de la interacción entre los componentes que implementan dicho CU. Obsérvese que este tipo de CP se comporta como una "Caja Blanca", es decir, verifica la interacción interna entre los componentes.

Se pueden especificar otros CP que verifiquen el sistema como un todo. Por ejemplo:

- CP de instalación, que verificarán que el sistema puede ser instalado en la plataforma del cliente y funciona correctamente.
- CP de configuración, que verifican que el sistema funciona en diferentes entornos de configuración. Ej. Diferentes configuraciones de red.
- Pruebas negativas que intenten provocar que el sistema falle para así detectar sus debilidades. Se especifican CP que intentan utilizar el sistema en formas para las que no ha sido diseñado, ej. Configuraciones de red incorrectas, capacidad de hardware insuficiente, carga de trabajo "imposible", etc.
- Pruebas de estrés que identifican cuando hay problemas con el sistema por recursos insuficientes o competencia por los recursos.

Muchos de estos casos pueden identificarse también como parte de los CU del sistema, dado que se corresponden casi siempre con especificaciones suplementarias o requisitos no funcionales del sistema.

2.4 Procedimiento de Pruebas

Un **Procedimiento de Prueba** (PP) especifica cómo realizar uno o varios CP o partes de estos. Puede constituir una instrucción para un individuo de cómo realizar un CP manualmente o puede especificar cómo interaccionar manualmente con una herramienta de automatización de pruebas para crear componentes ejecutables de prueba.

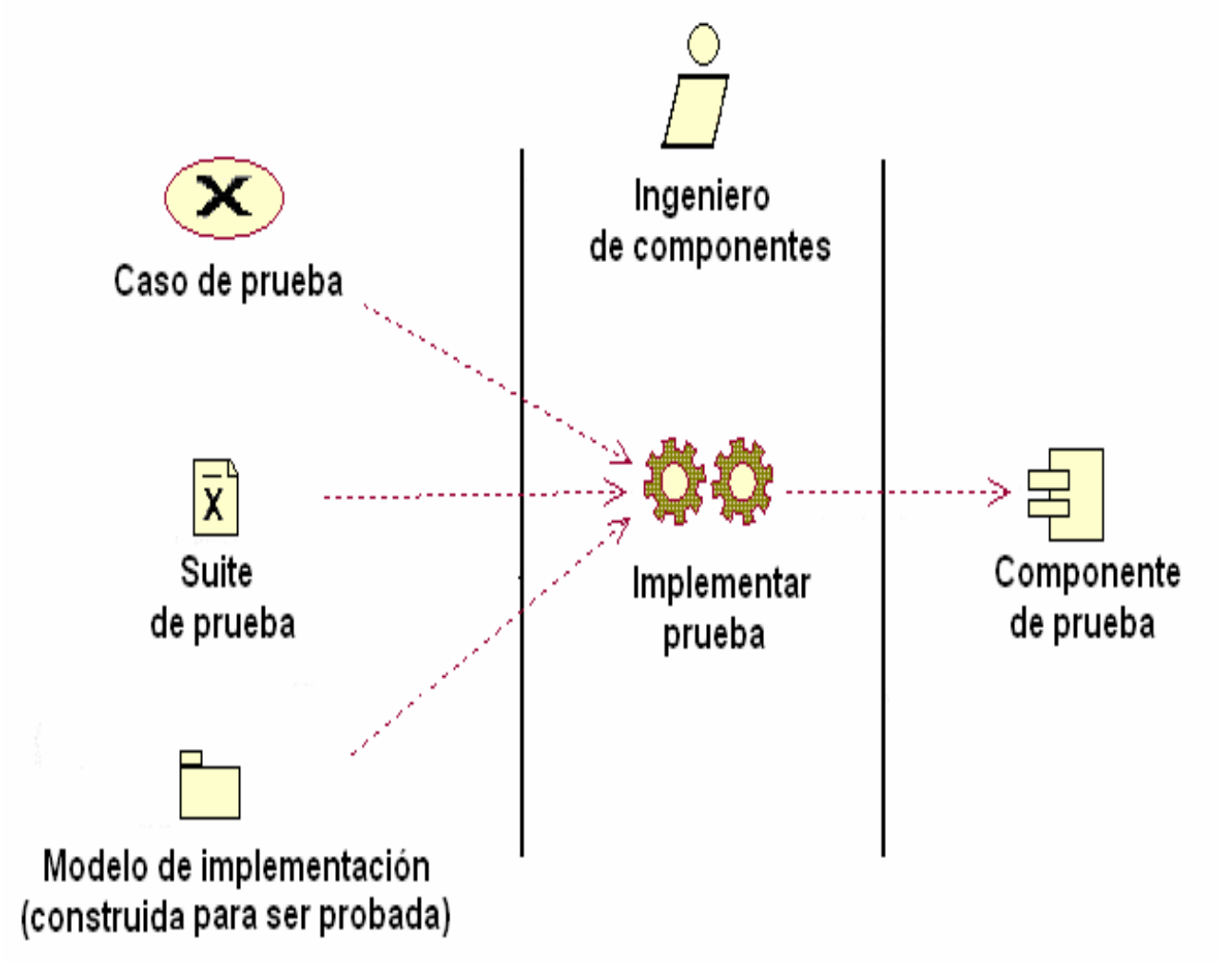
El cómo llevar a cabo un CP puede ser especificado por un PP, pero es a menudo útil utilizar un mismo PP para un conjunto de CP y reutilizar varios PP para un mismo CP.

Concretamente se propone unir estos dos artefactos en uno solo, los Casos de Prueba, pues estos pueden contener todas las acciones necesarias para validar los componentes y su funcionalidad, con datos diferentes, y en diferentes escenarios, creando, para cada Caso de Uso una Suite de Pruebas con los CP que responden a la validación de todos los tipos de datos, errores de diseño forzados y escenarios que supone el CU.

2.5 Componentes de Pruebas

Un **Componente de Prueba** automatiza una o varias Suite de Pruebas. Pueden ser desarrollados utilizando lenguaje de guiones o de programación, o ser grabados con una herramienta de automatización de pruebas. Se utilizan para probar los componentes en el modelo de implementación, proporcionando entradas de prueba, controlando y monitoreando la ejecución de los componentes a probar, para luego informar los resultados de las pruebas.

Pueden ser implementados utilizando tecnología de objetos. Si varios componentes de prueba tienen interacciones internas complejas con los componentes ordinarios en el modelo de implementación puede utilizarse un **Modelo de Diseño de Pruebas**, para modelar los componentes de prueba y representar vistas de alto nivel de ellos. Es por esto que se propone elaborar este modelo, similar al Modelo de Diseño. (17)

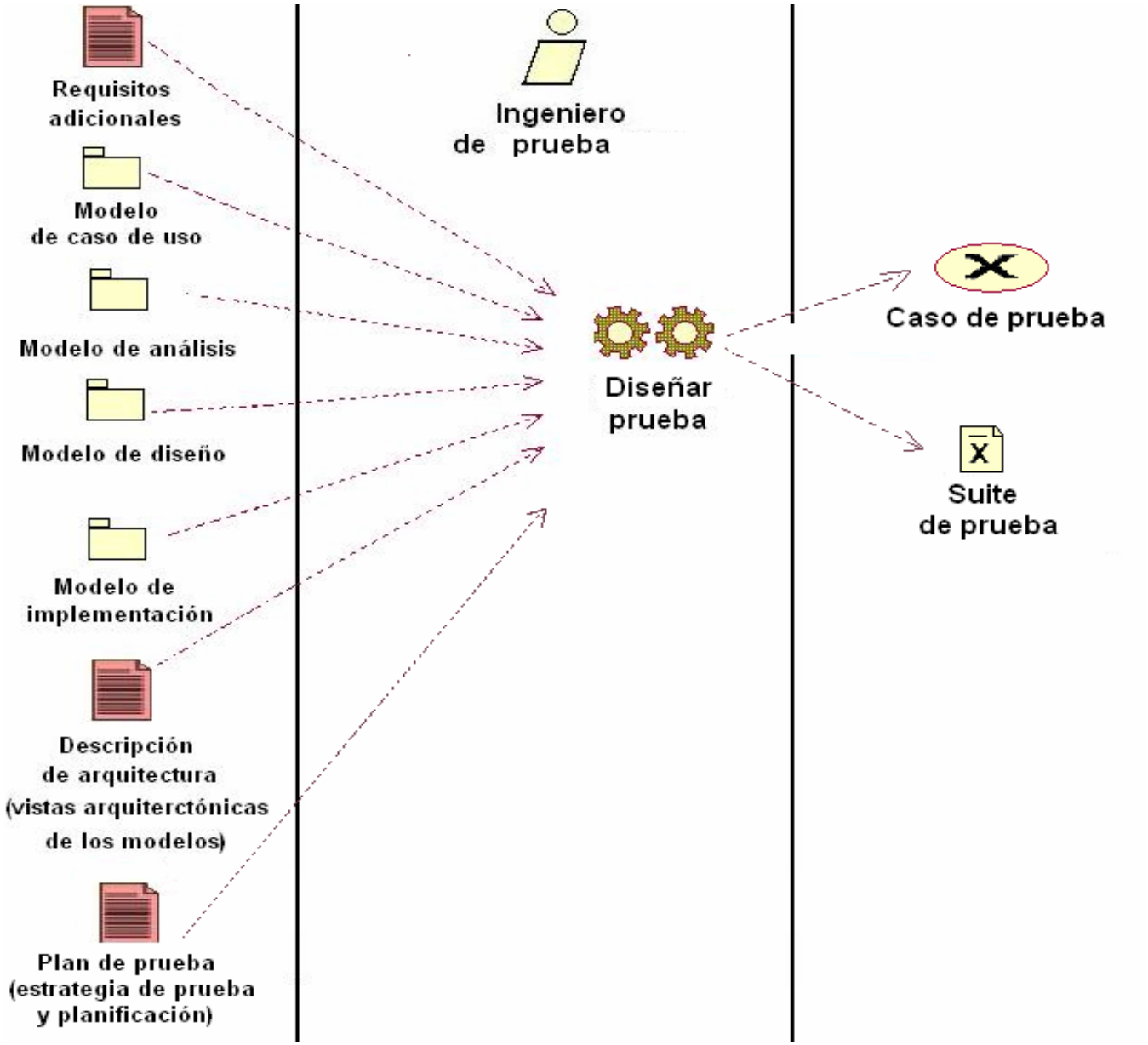


2.6 Suite de Pruebas

La **Suite de Pruebas** será entonces un conjunto de CP que validarán una determinada funcionalidad, módulo, o subsistema, según como sea más adecuado dividir su ejecución. (17)

El propósito de la actividad de diseño de pruebas es:

- Identificar y describir los CP
- Identificar y estructurar las Suites de Pruebas


Requisitos
adicionales
Modelo
de caso de uso
Modelo de análisis
Modelo de diseño
Modelo de
implementación
Descripción
de arquitectura
(vistas arquiterctónicas
de los modelos)
Plan de prueba
(estrategia de prueba
y planificación)
Ingeniero
de prueba
Diseñar
prueba
Caso de prueba
Suite
de prueba

2.7 Plan de Pruebas

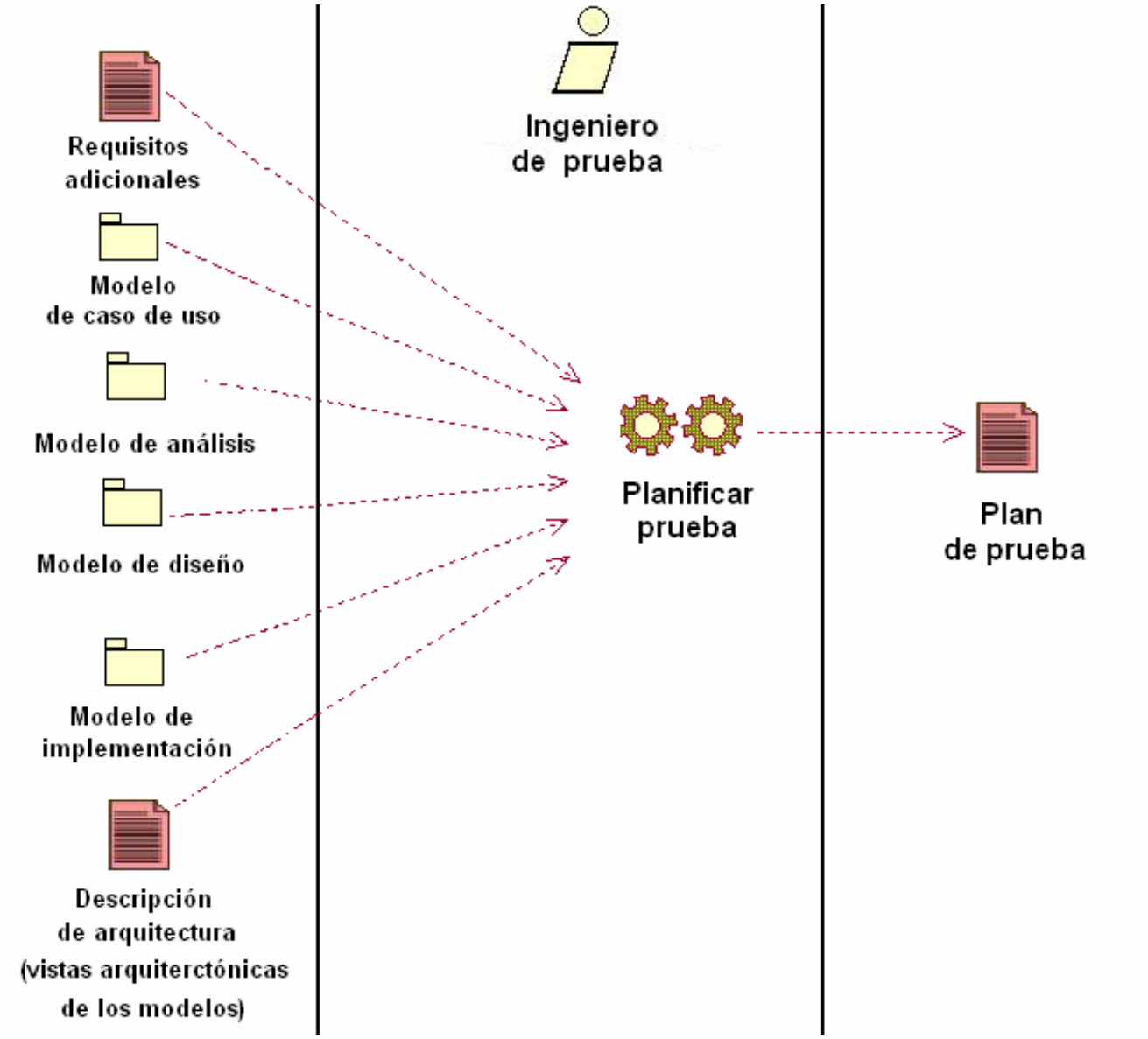
El **Plan de Pruebas** describe las estrategias, recursos y planificación de las pruebas del sistema. La estrategia incluye la definición del tipo de prueba a realizar para cada iteración y sus objetivos, el nivel de cobertura de prueba y de código necesario y el porcentaje de pruebas que deberían ejecutarse con un resultado específico. (17)

El propósito de la planificación de la prueba es planificar los esfuerzos en una iteración a través de las siguientes tareas:

- Describir estrategia de pruebas
- Estimar requisitos para el esfuerzo de pruebas. Ej. Sistemas, recursos humanos, habilidades, etc.
- Planificar esfuerzo de prueba

Para preparar el Plan de Pruebas los ingenieros se mueven en un rango de valores de entrada. El Modelo de Casos de Uso y los requisitos adicionales le ayudan a decidirse por un tipo adecuado de pruebas y a estimar el esfuerzo necesario para llevarlo a cabo. El ingeniero de pruebas utilizará también otros artefactos de entrada, como por ejemplo el Modelo de Diseño.

Los ingenieros o diseñadores de pruebas desarrollarán una estrategia de prueba para cada iteración (tipo de prueba, cómo y cuándo ejecutarlas, y cómo determinar si la prueba tiene éxito o no).
Este diagrama representa las entradas y el resultado de la actividad de planificación de las pruebas:



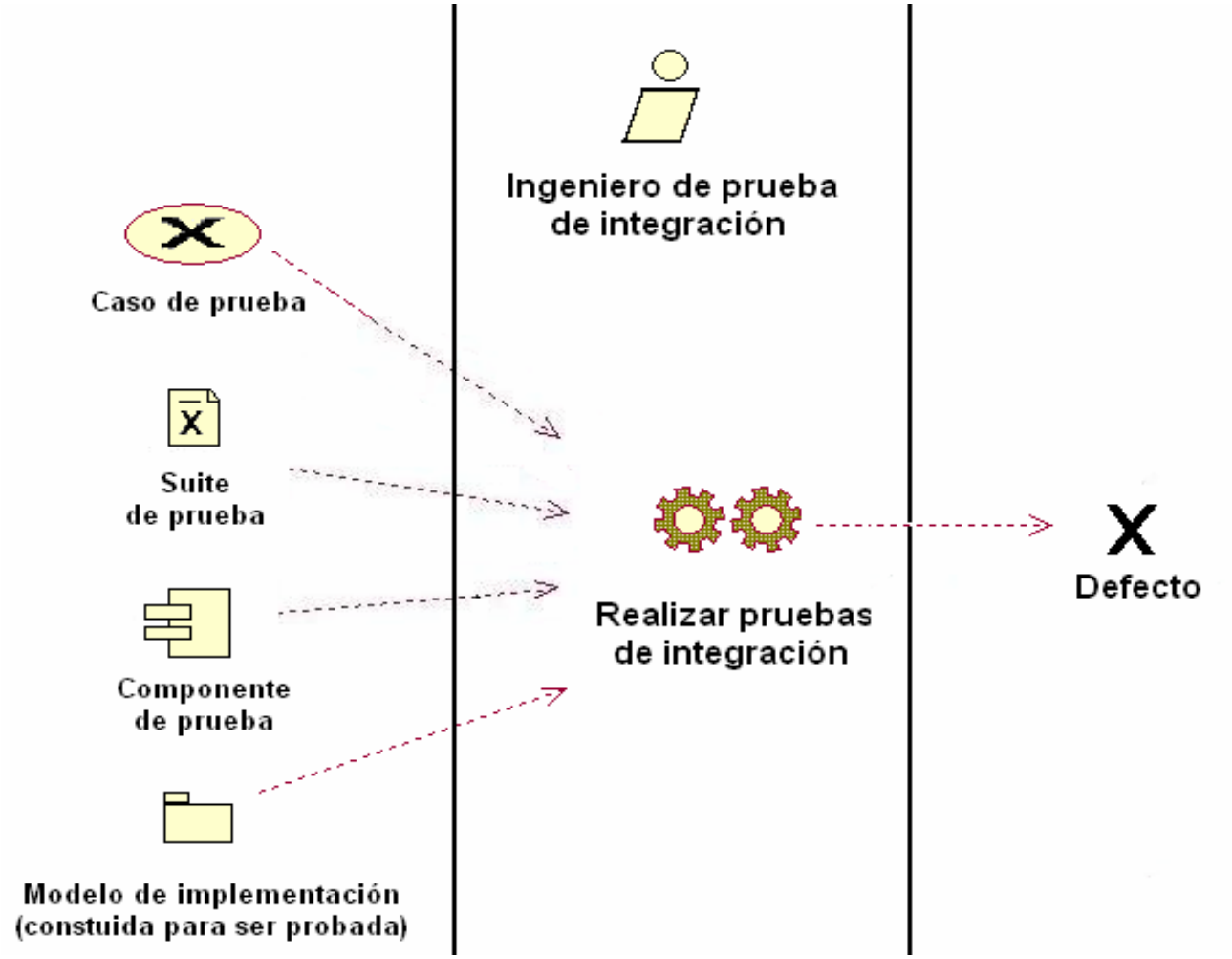
El principio general de la planificación de pruebas es diseñar CP con un solapamiento mínimo para probar los CU críticos y los más importantes que, casi siempre, se encuentran asociados a los riesgos más altos.

2.8 Defecto

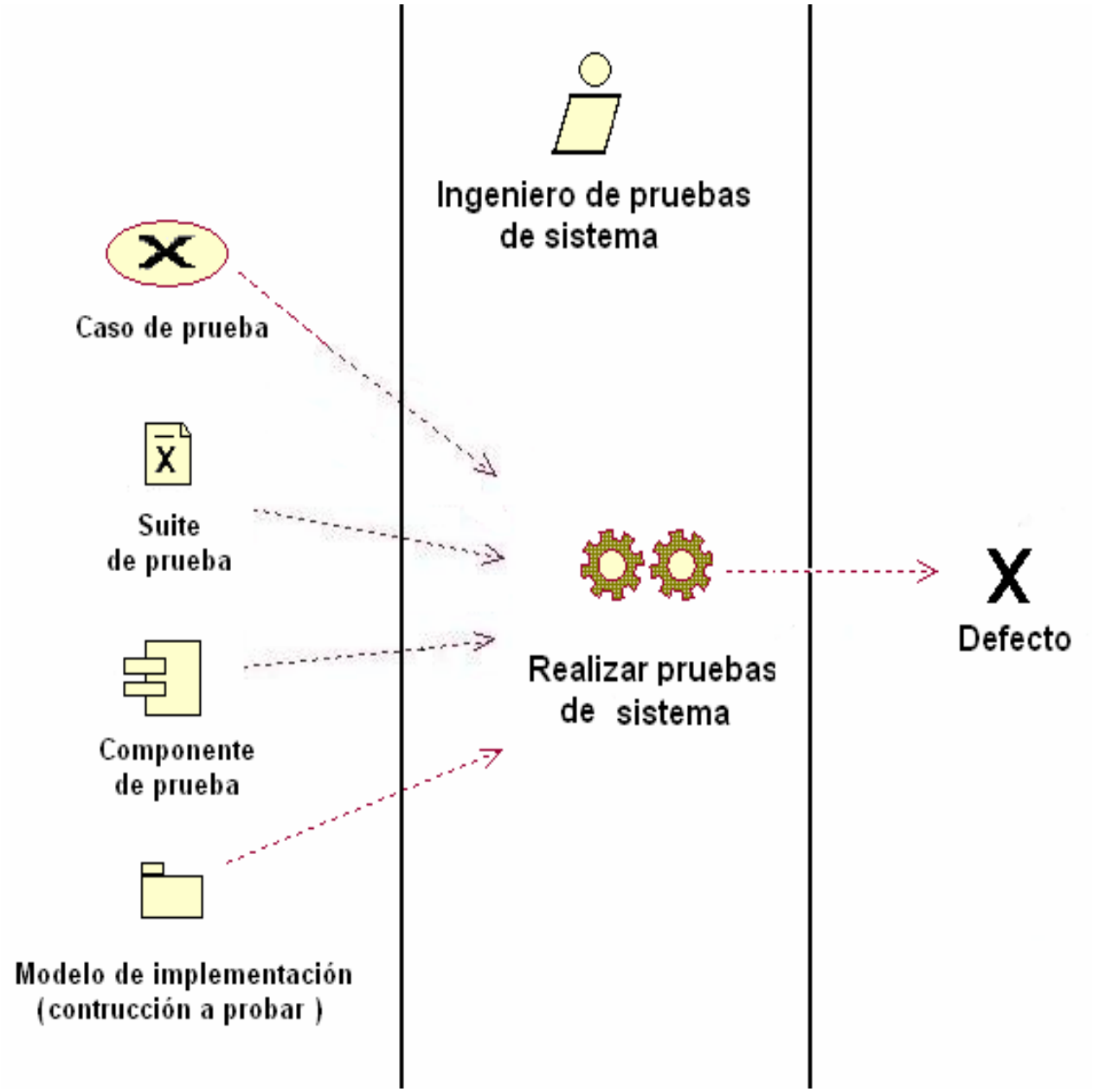
Un **Defecto** es una anomalía del sistema, como por ejemplo un síntoma de un fallo del software o un problema descubierto en una revisión. Puede ser utilizado para localizar problemas a controlar y corregir por los desarrolladores. (17)

Para encontrar los defectos se deben realizar Pruebas de Integración y Pruebas de Sistema.

Pruebas de Integración:



Pruebas de Sistema:



2.9 Evaluación de Pruebas

Una **Evaluación de Pruebas** es una evaluación de los resultados de los esfuerzos de prueba realizados, tales como la cobertura de pruebas, cobertura de código y defectos resueltos. En este proceso se detectan los defectos y se elabora el Plan de Resolución para resolverlos.

2.10 Sistema

En la siguiente tabla se establecen los recursos de sistema necesarios para realizar la verificación.

[Es recomendable que el sistema simule el entorno de producción, reduciendo los accesos y los tamaños de bases de datos si fuera apropiado.]

[Borre o agregue elementos a la lista]

Recurso	Nombre/Tipo
Servidor de base de datos	
Red o subred	
Nombre del servidor	
Nombre de la base de datos	
PC Cliente para pruebas	
Requerimientos especiales	
Repositorio de pruebas	
Red o subred	
Nombre del servidor	

2.11 Hitos del proyecto de verificación

[La verificación del [Nombre de proyecto] debe incorporar actividades de prueba para cada verificación identificada en las secciones anteriores. Se deben identificar los hitos del proyecto de verificación separados para comunicar los logros de estado de proyecto.]

Actividad que determina el hito	Esfuerzo	Fecha de comienzo	Fecha de finalización
Planificar la verificación			
Elaborar casos de prueba			
Ajuste y Control de Verificación			
Ejecutar la verificación			
Evaluar la verificación			

[Las últimas tres actividades se repiten en cada iteración. Debería incluir en esta tabla los datos de todas las iteraciones del proyecto.]

2.12 Entregables

[En esta sección enumere los documentos, herramientas e informes que se crearán, por quien, para quien y cuándo serán liberados.

Para cada entregable deberá indicar las fechas en que son liberadas todas las versiones del mismo.]

2.12.1 Modelo de Casos de Prueba

Documento	**Modelo de Casos de Prueba**
Creado por	El Responsable de verificación, [nombre del responsable de verificación].
Para quien	Es la guía para realizar las pruebas del sistema y lo usarán los Asistentes de verificación y el Responsable de verificación cuando se ejecuten las pruebas del sistema.
Fecha de liberación	Será liberado el [fecha de primera liberación].

2.12.2 Informes de Verificación

Documento	Se genera un documento **Informe de Verificación Unitaria** por cada prueba unitaria que se realice al sistema.
Creado por	Las personas que ejecutan las pruebas.
Para quien	Es el retorno para los implementadores de la tarea de verificación, que detalla los errores encontrados para que puedan ser corregidos.
Fecha de liberación	Será liberado luego de cada verificación unitaria. [Indique la versión y la fecha de liberación de todas las versiones de este informe.]

Documento	Se genera un documento **Informe Consolidación** por cada consolidación que se realice al sistema.
Creado por	Las personas que ejecutan las pruebas.
Para quien	Es el retorno para los implementadores de la tarea de consolidación, que detalla los errores encontrados para que puedan ser corregidos.
Fecha de liberación	Será liberado luego de cada consolidación. [Indique la versión y la fecha de liberación de todas las versiones de este informe.]

Documento	Se genera un documento **Informe de Verificación de Integración** por cada prueba de integración que se realice al sistema.
Creado por	Las personas que ejecutan las pruebas.
Para quien	Es el retorno para los implementadores de la tarea de verificación, que detalla los errores encontrados para que puedan ser corregidos.
Fecha de liberación	Será liberado luego de cada verificación de integración. [Indique la

	versión y la fecha de liberación de todas las versiones de este informe.]

Documento	Se genera un documento **Informe de Verificación de Sistema** por cada prueba de sistema que se realice.
Creado por	Las personas que ejecutan las pruebas.
Para quien	Es el retorno para los implementadores de la tarea de verificación, que detalla los errores encontrados para que puedan ser corregidos.
Fecha de liberación	Será liberado luego de cada verificación de sistema. [Indique la fecha de liberación de este informe.]

2.12.3 Evaluación de la verificación

Documento	Se genera un documento **Evaluación de la verificación** por cada prueba que se realice al sistema. Este documento contiene las fallas encontradas en el sistema, la cobertura de la verificación realizada y el estado del sistema.
Creado por	El Responsable de verificación, que toma como fuente de su trabajo los Informes de verificación.
Para quien	Es el resumen de la tarea de verificación y es el retorno para todo el equipo de trabajo del estado del sistema.
Fecha de liberación	Será liberado luego de cada verificación, unitaria, de integración y de sistema. [Indique la versión y la fecha de liberación de todas las versiones de este informe.]

2.12.4 Informe final de verificación

Documento	El documento **Informe final de verificación** es el resumen de la verificación final del sistema antes de que sea liberado al entorno del usuario.
Creado por	El Responsable de verificación, que toma como fuente de su trabajo los Informes de verificación.
Para quien	Indica el estado del sistema.
Fecha de liberación	Será liberado luego de la verificación final del sistema.

2.13 Dependencias [opcional]

[En esta sección se de hallan las dependencias, si existen, de las actividades de verificación respecto a otros elementos del sistema.]

2.13.1 Dependencia de personal [opcional]

[En esta sección se detallan las necesidades de personal para el equipo de verificación, la cantidad y habilidades.]

2.13.2 Dependencia de software [opcional]

[En esta sección se detallan los requisitos que debe cumplir el software a ser verificado para que se realice la verificación. Por ejemplo, que debe tener una verificación previa por parte del implementador, que debe estar en fecha disponible para verificar.]

2.13.3 Dependencia de hardware [opcional]

[En esta sección se detallan las necesidades de disponibilidad de hardware para realizar las tareas de verificación. Por ejemplo, horario, cantidad de horas que debe estar disponible para que las tareas de verificación se puedan realizar dentro del cronograma establecido.]

2.13.4 Dependencia de datos y base de datos de prueba [opcional]

[En esta sección se detallan las necesidades de disponibilidad de dato y de la base de datos para realizar las tareas de verificación. Por ejemplo, conjuntos de datos necesarios para la verificación, horarios de acceso a la base de datos que permitan que las tareas de verificación se puedan realizar dentro del cronograma establecido.]

2.14 Riesgos [opcional]

[En esta sección se detallan los riesgos detectados que puedan afectar la normal realización de las tareas de verificación.]

2.14.1 Planificación [opcional]

[En esta sección se plantean los riesgos relativos a la planificación. Por ejemplo, si una cronograma es muy ajustado un pequeño retraso en la liberación del software para verificar atrasa la verificación y por consiguiente las actividades que dependen de esta, provocando un retraso en el cronograma de todo el proyecto.]

2.14.2 Técnico [opcional]

[En esta sección se plantean los riesgos técnicos que afectan a la verificación. Por ejemplo, si existe un sistema anterior se deberá hacer una verificación en paralelo con el anterior en lugar de liberar la versión en un punto de trabajo a modo de prueba del sistema.]

2.14.3 Gestión [opcional]

[En esta sección se plantea como mediante la gestión se pueden mitigar los riesgos en las tareas de verificación.

2.15 Apéndice

2.15.1 Niveles de aceptación para los elementos verificados

[Se debe establecer un nivel de aceptación para los elementos verificados para poder establecer el estado en el que se encuentra el proyecto.

En esta sección defina niveles de aceptación y los criterios de pertenencia a cada nivel.

Como ejemplo de niveles de aceptación:

No aprobado: el elemento verificado tiene errores catastróficos (uno o varios) que impiden su uso o tiene errores críticos (uno o varios) que hacen que el elemento verificado no sea confiable. El usuario no puede depender de él para realizar el trabajo.

Aprobado con Observaciones: el elemento verificado no tiene errores catastróficos, ni errores críticos, pero tiene errores marginales (uno o varios) que hacen que el elemento de software se degrade en algunas situaciones.

Aprobado: el elemento verificado no tiene errores o tiene errores menores que no afectan el normal funcionamiento del elemento.]

2.15.2 Plan de Resolución de Errores

El **Plan de Resolución de Errores** no es más que un listado de defectos encontrados en una iteración de prueba con las correspondientes tareas para eliminarlos o mitigarlos. Es un documento que se genera al final de una iteración y se revisa posteriormente para repetir las pruebas que generaron defectos.

2.15.3 Niveles de gravedad de error

En muchas actividades del proceso de verificación se deben clasificar los errores según su nivel de gravedad. Se asigna un nivel de gravedad a los errores para poder capturar de alguna manera su impacto en el sistema. Además para poder evaluar la verificación y el sistema.

A continuación se da una sugerencia de cuatro niveles diferentes de gravedad de error:

Catastrófico: un error cuya presencia impide el uso del sistema.

Crítico: un error cuya presencia causa la pérdida de una funcionalidad crítica del sistema. Si no se corrige el sistema no satisfará las necesidades del cliente.

Marginal: un error que causa un daño menor, produciendo pérdida de efectividad, pérdida de disponibilidad o degradación de una funcionalidad que no se realiza fácilmente de otra manera.

Menor: un error que no causa perjuicio al sistema, pero que requiere mantenimiento o reparación. No causa pérdida de funcionalidades que no se puedan realizar de otra manera.

2.15.4 Asignar gravedad a los errores

En la tabla siguiente se enumeran instrucciones que pueden seguirse para asignar una gravedad a los errores

Gravedad	Tipos más comunes	Condiciones necesarias
1	El error bloquea la generación o la realización de pruebas de una característica El error afecta a las pruebas de una característica diferente de la que se está probando a la vez	El sistema no funciona. El usuario no puede ni siquiera empezar a utilizar partes importantes del sistema.
2	Los pasos definidos en la documentación no son viables Los resultados o el comportamiento de una función o proceso contradicen los resultados esperados (según se documentan en la especificación funcional) Los resultados o el comportamiento de una función o proceso contradicen los resultados esperados lógicamente Falta la funcionalidad documentada (en este caso la prueba está bloqueada) Falta documentación o es inadecuada	Si se cumplen las condiciones siguientes, la gravedad es 2: usuario no puede evitar que se produzca el problema de una forma sencilla usuario no puede averiguar una forma de evitar el error fácilmente sistema no cumple los requisitos comerciales principales Si no se cumplen las condiciones, la gravedad es 3.
3	La función o proceso no funciona Los resultados o el comportamiento de una función o proceso	Si se cumplen las condiciones siguientes, la gravedad es 3: El usuario puede evitar que se produzca el

	contradicen los resultados esperados (según se documentan en la especificación funcional) Los resultados o el comportamiento de una función o proceso contradicen los resultados esperados lógicamente. Errores poco importantes e imprecisiones en la documentación Erratas	problema de una forma sencilla El usuario puede averiguar una forma de evitar el error fácilmente El error no produce una experiencia desfavorable al usuario Los requisitos comerciales principales siguen siendo funcionales El error no bloquea un número suficiente de casos de prueba Si no se cumplen las condiciones, la gravedad es 2.
4	Sugerencias Mejoras futuras	Obviamente, no se trata de un error de producto para esta versión.

2.16 Histórico de revisiones

FECHA	VERSION	COMENTARIOS	AUTOR

Estructura fijada en el estándar:

- Identificador
- Descripción de las pruebas: elementos probados y entorno de la prueba.
- Anotación de los datos sobre cada hecho ocurrido (incluido el comienzo y el final de la prueba)
    - Fecha y hora
    - Identificador de informe de incidente
- Otras informaciones.

2.17 Funciones y responsabilidades

En la siguiente tabla se enumeran las funciones normales que pueden ser necesarias en un equipo de pruebas, junto con sus responsabilidades:

Función	**Responsabilidad**
Jefe de equipo	Definir los objetivos de las pruebas y generar el plan de pruebas maestro Elaborar el plan de generación y clasificación Revisar el plan de pruebas detallado Revisar los casos de pruebas detallados Revisar los errores introducidos en la herramienta de seguimiento de errores y supervisar su estado Celebrar reuniones para efectuar la clasificación Generar informes de estado semanales Resolver los problemas que bloqueen o atrasen las pruebas Revisar el análisis de impacto y elaborar documentos para administrar los cambios Hacer un seguimiento de la programación de las pruebas Asegurar que se consigue el nivel de pruebas apropiado para una versión en particular Dirigir la ejecución de la prueba de aceptación de la versión real Ejecutar casos de prueba Generar los informes de las pruebas
Ingeniero de pruebas	Generar el plan de pruebas detallado Revisar los casos de pruebas detallados Documentar los problemas encontrados durante la implementación Realizar la prueba de aceptación de la versión Ejecutar casos de prueba Informar de los errores en la herramienta de seguimiento de errores Volver a probar los errores corregidos
Ingeniero de	Preparar las versiones y modificaciones de versiones Realizar la prueba de comprobación de la versión

sistemas	Corregir los problemas de disponibilidad de hardware o software o asignar su solución a otra persona

2.18 Roles

En la tabla a continuación se muestra la composición de personal para el proyecto [Nombre del proyecto] en el área Verificación del Software.

Rol	**Cantidad mínima de recursos recomendada**	**Responsabilidades**
Responsable de verificación		Identifica, prioriza e implementa los casos de prueba. Genera el Plan de Verificación. Genera el Modelo de Prueba. Evalúa el esfuerzo necesario para verificar. Proporciona la dirección técnica. Adquiere los recursos apropiados. Proporciona informes sobre la verificación.
Asistente de verificación		Ejecuta las pruebas Registra los resultados de las pruebas. Recuperar el software de errores. Documenta los pedidos de cambio.
Administrador de Base de Datos		Realiza la gestión y mantenimiento del entorno de los datos (base de datos) de prueba y los recursos. Administra la base de datos de prueba.

2.19 Evaluar los riesgos del proyecto

Un riesgo es la probabilidad de que se produzca un suceso que pueda poner en peligro el sistema. Para evaluar los riesgos, el equipo de pruebas puede preparar una matriz que identifique riesgos y asigne uno de los siguientes factores de exposición a cada uno:

Probabilidad de pérdida. Se trata de la probabilidad de que se produzca el riesgo. Normalmente, son suficientes tres tipos. Por ejemplo, “No probable” (menos del 50%), “Posible” (50%) y “Muy probable” (más del 50%).

Tamaño de la pérdida. Se trata del impacto en el calendario del proyecto cuando se produce un suceso asociado con un riesgo. De nuevo, suele ser adecuado usar la clasificación siguiente: “Mínimo”, “Peligra la fecha final” y “Efecto significativo en la fecha final”.

Plan de contingencia. Es un plan para controlar las circunstancias del riesgo. Puede tratarse de incluir en la programación un número adicional de días para cumplir estas circunstancias, agregar personal y otros recursos, o cambiar la fecha de entrega.

En la tabla siguiente se presentan alguno de los riesgos comunes encontrados en proyectos de pruebas y en posibles planes de contingencia.

Riesgo	**Plan de contingencia**
El desarrollo no se ajusta a la programación	Determinar la posibilidad de comenzar la prueba inicial a la vez que las últimas etapas del desarrollo y agregar recursos de prueba y desarrollo
Los responsables de realizar las pruebas no están familiarizados con la aplicación	Agregar días adicionales para formar a los evaluadores en la aplicación
Las aplicaciones se basan en tecnologías recientes	Pueden producirse retrasos imprevistos debidos a la tecnología. La programación debe seguir siendo flexible
Aumenta el ámbito de los nuevos requisitos	Podría haber un incremento imprevisto en el ámbito debido a la naturaleza evolutiva de los requisitos.

.

2.20 Formulario de pedido de cambio

Identificación

Proyecto:

[No del proyecto al que pertenece el Pedido de Cambio**]**

Número:

[Número consecutivo identificador del Pedido de Cambio]

Tipo:

[Tipo de Pedido de Cambio: error o mejora]

Título

[Título del pedido de cambio que refleje de manera sintetizada el contenido del pedido]

Fecha de creación:

[Fecha de creación del pedido de cambio]

Creada por:

[Nombre de la persona que creó el pedido de cambio, y contacto de la misma**]**

Prioridad:[Prioridad del pedido de cambio, esta puede ser alta, media o baja en dependencia de la importancia del mismo]

Problema actual

Descripción:

[Breve descripción del pedido de cambio que abarque los aspectos más importantes del mismo]

Como repetir:

[Secuencia de pasos que permitan repetir el problema]

Nuevos requerimientos:

[Listado de nuevos requerimientos que desean ser agregados]

Condiciones bajo las que fue observado el problema:

[Condiciones de software bajo las que fue detectado el problema]

Ambiente actual: hardware:

[Hardware correspondiente al ordenador donde se detectó el problema]

Sistema operativo:

[Sistema operativo del ordenador donde se detectó el problema]

Cambio propuesto (creador)

Descripción:

[Descripción del cambio propuesto por parte del creador del pedido de cambio]

Costo estimado:

[Costo estimado del cambio por parte del creador del pedido]

Cambio propuesto (Equipo de desarrollo)

Acción:

[Explicación de la decisión tomada respecto al cambio. Pudiera ser aplazarlo para una próxima iteración del software]

Decisión tomada:

[Aceptar, rechazar]

Elementos de configuración afectados:

[Elementos de configuración que se ven afectados por el pedido de cambio]

Errores corregidos:

[Errores corregidos durante la implementación del pedido]

Nuevas funcionalidades:

[Nuevas funcionalidades que pudieron ser agregadas durante la implementación del pedido]

Resolución

Tiempo y costo estimado del cambio propuesto:

[Tiempo y costo estimado por parte del equipo de desarrollo para la realización del cambio]

Desarrollador:

[Nombre del desarrollador del pedido de cambio]

2.21 Conclusiones

Podemos decir a forma de conclusión que este capitulo servirá de guía al equipo de prueba que estará encargado de aplicar las propuestas de pruebas planteadas en nuestro trabajo para el producto LIMS control de calidad del CIGB, aclarando que si aprovechan todo este material al máximo a la hora de llevar a cabo este procesos interiorizan todas las ideas y artefactos aquí expuestos podrían definir múltiples casos de prueba de integración para cada caso de uso en dependencia de las condiciones de prueba que se tengan en cuenta.

Capítulo III: Propuesta de pruebas para un producto LIMS.

3.0 Introducción

Cuando se desarrolla un software, una de las actividades asociadas a este proceso es la prueba; de hecho, se ha establecido formalmente que éstas son fundamentales dentro de cada una de las etapas del proceso de desarrollo de un sistema. La principal razón es que a partir de ella se puede asegurar el cumplimiento de criterios mínimos de operabilidad y garantizar la calidad de los productos implementados.

Luego dirigimos este capitulo a las personas que en un futuro estarán encargados de probar este producto, esclareciéndole los aspectos fundamentales relacionados con las pruebas específicas que deben aplicársele a un producto LIMS, teniendo en cuenta las normas planteadas por la FDA. El capítulo comienza con una panorámica acerca de los requisitos con que debe cumplir un LIMS según la extensión CFR 21 Parte 11, luego consta de un Catalogo que servirá de guía para llevar a cabo todo el proceso de las pruebas, así como las estrategias generales de pruebas, y la pruebas especificas que proponemos para este tipo de producto.

3.1 La necesidad de probar

El desarrollo de un sistema LIMS implica una serie de actividades de producción en las que las posibilidades de que aparezca la falibilidad humana son enormes. Debido a la imposibilidad de trabajar y comunicarse de forma perfecta, el desarrollo de software debe ir acompañado de una actividad que garantice la calidad, ya que esta representa un requisito fundamental a la hora de comercializar los sistemas.

La prueba de software es un elemento crítico e imprescindible para la garantía de la calidad, y de ahí la necesidad de aplicarla, es por esto que hoy con el desarrollo de este trabajo queremos dejar bien claro todos los aspectos que se deben de tener en cuenta a la hora de realizar este trabajo relacionado con la calidad.

3.2 Técnicas de diseño de casos de prueba

El diseño de casos de prueba está totalmente mediatizado por la imposibilidad de probar exhaustivamente el software. Pensemos en un programa muy sencillo que sólo suma dos números enteros de dos cifras (del 0 al 99). Si quisiéramos probar, simplemente, todos los valores válidos que se pueden sumar, deberíamos probar 10.000 combinaciones distintas de cien posibles números del primer sumando y cien del segundo. Pensemos que aún quedarían por probar todas las posibilidades de error al introducir los datos (por ejemplo, teclear una letra en vez de un número).

La conclusión es que, si para un programa tan elemental debemos realizar tantas pruebas, pensemos en lo que supondría un programa medio.

En consecuencia, las técnicas de diseño de casos de prueba tienen como objetivo conseguir una confianza aceptable en que se detectarán los defectos existentes (ya que la seguridad total sólo puede obtenerse de la prueba exhaustiva, que no es practicable) sin necesidad de consumir una cantidad excesiva de recursos (por ejemplo, tiempo para probar o tiempo de ejecución). Toda la disciplina de pruebas debe moverse, por lo tanto, en un equilibrio entre la disponibilidad de recursos y la confianza que aportan los casos para descubrir los defectos existentes. Este equilibrio no es sencillo, lo que convierte a las pruebas en una disciplina difícil que está lejos de parecerse a la imagen de actividad rutinaria que suele sugerir.

Ya que no se pueden probar todas las posibilidades de funcionamiento del software, la idea fundamental para el diseño de casos de prueba consiste en elegir algunas de ellas que, por sus características, se consideren representativas del resto. Así, se asume que, si no se detectan defectos en el software al ejecutar dichos casos, podemos tener un cierto nivel de confianza (que depende de la elección de los casos) en que el programa no tiene defectos.

La dificultad de esta idea es saber elegir los casos que se deben ejecutar. De hecho, una elección puramente aleatoria no proporciona demasiada confianza en que se puedan detectar todos los defectos existentes. Por ejemplo, en el caso del programa de suma de dos números, elegir cuatro pares de sumandos al azar no aporta mucha seguridad a la prueba (una probabilidad de 4/10.000 de cobertura de posibilidades). Por eso es necesario recurrir a ciertos criterios de elección que veremos a continuación.

Existen tres enfoques principales para el diseño de casos:

 El enfoque estructural o de caja blanca. Consiste en centrarse en la estructura interna (implementación) del programa para elegir los casos de prueba. En este caso, la prueba ideal (exhaustiva) del software consistiría en probar todos los posibles caminos de ejecución, a través de las instrucciones del código, que puedan trazarse.

 El enfoque funcional o de caja negra. Consiste en estudiar la especificación de las funciones, la entrada y la salida para derivar los casos. Aquí, la prueba ideal del software consistiría en probar todas las posibles entradas y salidas del programa.

 El enfoque aleatorio consiste en utilizar modelos (en muchas ocasiones estadísticos) que representen las posibles entradas al programa para crear a partir de ellos los casos de prueba. La prueba exhaustiva consistiría en probar todas las posibles entradas al programa.

Estos enfoques no son excluyentes entre sí, ya que se pueden combinar para conseguir una detección de defectos más eficaz. Veremos a continuación una presentación de cada uno de ellos.

3.2.1 Pruebas estructurales. (Prueba de la caja blanca)

Como hemos dicho, las pruebas exhaustivas son impracticables. Podemos recurrir al clásico ejemplo de Myers [MYERS, 1979] de un programa de 50 líneas con 25 sentencias IF en serie, en el que el número total de caminos contiene 33,5 millones de secuencias potenciales (contando dos posibles salidas para cada IF tenemos 225 posibles caminos). El diseño de casos tiene que basarse en la elección de caminos importantes que ofrezcan una seguridad aceptable en descubrir defecto, y para ello se utilizan los llamados criterios de cobertura lógica. Antes de pasar a examinarlos, conviene señalar que estas técnicas no requieren el uso de ninguna representación gráfica específica del software, aunque es habitual tomar como base los llamados diagramas de flujo de control (flow graph charts o flowcharts). Como ejemplo de diagrama de flujo junto al código correspondiente.

En el recuadro adjunto pueden consultarse algunas recomendaciones para dibujar los grafos de flujo de los programas para poder generar los casos de prueba correspondientes. También existen herramientas que dibujan el grafo de flujo de un programa sólo con facilitar el código fuente del mismo como entrada.

Una posible clasificación de criterios de cobertura lógica es la que se ofrece abajo. Hay que destacar que los criterios de cobertura que se ofrecen están en orden de exigencia y, por lo tanto, de coste económico. Es decir, el criterio de cobertura de sentencias es el que ofrece una menor seguridad de detección de defectos, pero es el que cuesta menos en número de ejecuciones del programa.

- Cobertura de sentencias. Se trata de generar los casos de prueba necesarios para que cada sentencia o instrucción del programa se ejecute al menos una vez.

- Cobertura de decisiones. Consiste en escribir casos suficientes para que cada decisión tenga, por lo menos una vez, un resultado verdadero y, al menos una vez, uno falso. En general, una ejecución de pruebas que cumple la cobertura de decisiones cumple también la cobertura de sentencias.

- Cobertura de condiciones. Se trata de diseñar tantos casos como sea necesario para que cada condición de cada decisión adopte el valor verdadero al menos una vez y el falso al menos una vez. No podemos asegurar que si se cumple la cobertura de condiciones se cumple necesariamente la de decisiones.

- Criterio de decisión/condición. Consiste en exigir el criterio de cobertura de condiciones obligando a que se cumpla también el criterio de decisiones.

- Criterio de condición múltiple. En el caso de que se considere que la evaluación de las condiciones de cada decisión no se realiza de forma simultánea (por ejemplo, según se ejecuta en el procesador), se podría considerar que cada decisión multicondicional se descompone en varias decisiones unicondicionales. Es decir, una decisión como IF ((a=1) AND (c=4)) THEN se convierte en una concatenación de dos decisiones: IF(a=1) y IF(c=4). En este caso, debemos conseguir que todas las combinaciones posibles de resultados (verdadero/falso) de cada condición en cada decisión se ejecuten al menos una vez.

La cobertura de caminos (secuencias de sentencias) es el criterio más elevado: cada uno de los posibles caminos del programa se debe ejecutar al menos una vez. Se define camino como la secuencia de sentencias encadenadas desde la sentencia inicial del programa hasta su sentencia final. Como hemos visto, el número de caminos en un programa pequeño puede ser impracticable para las pruebas.

Para reducir el número de caminos a probar, se habla del concepto de camino de prueba (test path): un camino del programa que atraviesa, como máximo, una vez el interior de cada bucle que encuentra. La idea en la que se basa consiste en que ejecutar un bucle más de una vez no supone una mayor seguridad de detectar defectos en él. Sin embargo, otros especialistas recomiendan que se pruebe cada bucle tres veces: una sin entrar en su interior, otra ejecutándolo una vez y otra más ejecutándolo dos veces. Esto último es interesante para comprobar cómo se comporta a partir de los valores de datos procedentes, no del exterior del bucle (como en la primera iteración), sino de las operaciones de su interior.

Si trabajamos con los caminos de prueba, existe la posibilidad de cuantificar el número total de caminos utilizando algoritmos basados en matrices que representan el grafo de flujo del programa. Así, en [SHOOMAN, 1983] y en [BEIZER, 1990] se ofrecen diversos métodos basados en ecuaciones, expresiones regulares y matrices que permiten tanto calcular el número de caminos como enumerar dichos caminos expresados como series de arcos del grafo de flujo. Saber cuál es el número de caminos del grafo de un programa ayuda a planificar las pruebas y a asignar recursos a las mismas, ya que indica el número de ejecuciones necesarias. También sirve de comprobación a la hora de enumerar los caminos.

Los bucles constituyen el elemento de los programas que genera un mayor número de problemas para la cobertura de caminos. Su tratamiento no es sencillo ni siquiera adoptando el concepto de camino de prueba. Pensemos, por ejemplo, en el caso de varios bucles anidados o bucles que fijan valores mínimo y máximo de repeticiones. Se puede consultar una presentación detallada de su tratamiento en [BEIZER, 1990].

3.2.1.1 Utilización de la complejidad ciclomática de McCabe.

La utilización de la métrica de McCabe [MCCABE, 1976] ha sido muy popular en el diseño de pruebas desde su creación. Esta métrica es un indicador del número de caminos independientes que existen en un grafo.

El propio McCabe definió como un buen criterio de prueba la consecución de la ejecución de un conjunto de caminos independientes, lo que implica probar un número de caminos igual al de la métrica. Se propone este criterio como equivalente a una cobertura de decisiones, aunque se han propuesto contraejemplos que invalidan esta suposición.

La complejidad de McCabe V (G) se puede calcular de las tres maneras siguientes a partir de un grafo de flujo G:

- V (G) = a - n + 2, siendo a el número de arcos o aristas del grafo y n el número de nodos.

- V (G) = r, siendo r el número de regiones cerradas del grafo.

- V (G) = c + 1, siendo c el número de nodos de condición

- V (G) = 14 - 11 + 2 = 5. Los arcos han sido identificados con las marcas desde a1 hasta a14. Los nodos están numerados del 1 al 11.

- V (G) = 5. Las regiones o áreas cerradas (limitadas por aristas) del grafo son cinco, que hemos numerado en el grafo. Como puede verse, se ha marcado un área (región 5) añadiendo un arco ficticio desde el nodo 11 al nodo 1. Esto se debe a que las fórmulas de McCabe sólo son aplicables a grafos fuertemente conexos, es decir, aquellos para los cuales existe un camino entre cualesquiera dos nodos que se elijan. Los programas, con un nodo de inicio y otro de final, no cumplen esta condición. Por eso, debemos marcar dicho arco o, como alternativa, contabilizar la región externa al grafo como una más.

- V (G) = 4 + 1. Los nodos de condición son el 2, el 4, el 5 y el 6. Todos ellos son nodos de decisión binaria, es decir, surgen dos aristas de ellos. En el caso de que de un nodo de condición (por ejemplo, una sentencia Case-of) partiera n arcos (n>2), debería contabilizarse como n-1 para la fórmula (que equivale al número de bifurcaciones binarias necesarias para simular dicha bifurcación “n-aria”).

Una vez calculado el valor V (G) podemos afirmar que el número máximo de caminos independientes de dicho grafo es cinco.

El criterio de prueba de McCabe consiste en elegir cinco caminos que sean independientes entre sí y crear casos de prueba cuya ejecución siga dichos caminos. Para ayudar a la elección de dichos caminos, McCabe [MCCABE, 1982] creó un procedimiento llamado “método del camino básico”, consistente en realizar variaciones sobre la elección de un primer camino de prueba típico denominado camino básico.

En nuestro caso, un posible conjunto de caminos (descritos como secuencias de nodos visitados) podría ser el siguiente:

- 1-2-11
- 1-2-3-4-10-2
- 1-2-3-4-5-10-2
- 1-2-3-4-5-6-7-9-4-10-2-11
- 1-2-3-4-5-6-8-9-4-10-2-11

Hemos subrayado los elementos de cada camino que lo hacen independiente de los demás. Conviene aclarar que algunos de los caminos quizás no se puedan ejecutar solos y requieran una concatenación con algún otro. A partir de estos caminos, el diseñador de las pruebas debe analizar el código para saber los datos de entrada necesarios para forzar la ejecución de cada uno de ellos. Una vez determinados los datos de entrada hay que consultar la especificación para averiguar cuál es la salida teóricamente correcta para cada caso.

Puede ocurrir también que las condiciones necesarias para que la ejecución pase por un determinado camino no se puedan satisfacer de ninguna manera. Nos encontraríamos entonces ante un «camino imposible». En ese caso, debemos sustituir dicho camino por otro posible que permita satisfacer igualmente el criterio de prueba de McCabe, es decir, que ejecute la misma arista o flecha que diferencia al imposible de los demás caminos independientes.

La experimentación con la métrica de McCabe ha dado como resultado las siguientes conclusiones [BEMER, 1990]:

- V (G) marca un límite mínimo de número de casos de prueba para un programa, contando siempre cada condición de decisión como un nodo individual.

- Parece que cuando V (G) es mayor que diez la probabilidad de defectos en el módulo o en el programa crece bastante si dicho valor alto no se debe a sentencias Case-of o similares. En estos casos, es recomendable replantearse el diseño modular obtenido, dividiendo los módulos para no superar el límite de diez de la métrica de McCabe en cada uno de ellos.

3.2.2 Prueba funcional. (Prueba de la caja negra)

La prueba funcional o de caja negra se centra en el estudio de la especificación del software, del análisis de las funciones que debe realizar, de las entradas y de las salidas. Lamentablemente, la prueba exhaustiva de caja negra también es generalmente impracticable: pensemos en el ejemplo de la suma visto en el apartado 3.2.1.De nuevo, ya que no podemos ejecutar todas las posibilidades de funcionamiento y todas las combinaciones de entradas y de salidas, debemos buscar criterios que permitan elegir un subconjunto de casos cuya ejecución aporte una cierta confianza en detectar los posibles defectos del software. Para fijar estas pautas de diseño de pruebas, nos apoyaremos en las siguientes dos definiciones de Myers que definen qué es un caso de prueba bien elegido:

- El que reduce el número de otros casos necesarios para que la prueba sea razonable. Esto implica que el caso ejecute el máximo número de posibilidades de entrada diferentes para así reducir el total de casos.

- Cubre un conjunto extenso de otros casos posibles, es decir, nos indica algo acerca de la ausencia o la presencia de defectos en el conjunto específico de entradas que prueba, así como de otros conjuntos similares.

Veremos a continuación distintas técnicas de diseño de casos de caja negra.

3.2.2.1 Particiones o clases de equivalencia

Esta técnica utiliza las cualidades que definen un buen caso de prueba de la siguiente manera:

- Cada caso debe cubrir el máximo número de entradas.

- Debe tratarse el dominio de valores de entrada dividido en un número finito de clases de equivalencia que cumplan la siguiente propiedad: la prueba de un valor representativo de una clase permite suponer “razonablemente” que el resultado obtenido (existan defectos o no) será el mismo que el obtenido probando cualquier otro valor de la clase.

El método de diseño de casos consiste entonces en:

- Identificación de clases de equivalencia.

- Creación de los casos de prueba correspondientes.

Para identificar las posibles clases de equivalencia de un programa a partir de su especificación se deben seguir los siguientes pasos:

 Identificación de las condiciones de las entradas del programa, es decir, restricciones de formato o contenido de los datos de entrada.

 A partir de ellas, se identifican clases de equivalencia que pueden ser:

- De datos válidos.

- De datos no válidos o erróneos.

La identificación de las clases se realiza basándose en el principio de igualdad de tratamiento: todos los valores de la clase deben ser tratados de la misma manera por el programa.

 Existen algunas reglas que ayudan a identificar clases:

- Si se especifica un rango de valores para los datos de entrada (por ejemplo, “el número estará comprendido entre 1 y 49”), se creará una clase válida (1 " número " 49) y dos clases no válidas (número < 1 y número > 49).

- Si se especifica un número de valores (por ejemplo, “se pueden registrar de uno a tres propietarios de un piso”), se creará una clase válida (1 " propietarios " 3) y dos no válidas (propietarios < 1 y propietarios > 3).

- Si se especifica una situación del tipo “debe ser” o booleana (por ejemplo, “el primer carácter debe ser una letra”), se identifican una clase válida (“es una letra”) y una no válida (“no es una letra”).

- Si se especifica un conjunto de valores admitidos (por ejemplo, “pueden registrarse tres tipos de inmuebles: pisos, chalés y locales comerciales”) y se sabe que el programa trata de forma diferente cada uno de ellos, se identifica una clase válida por cada valor (en este caso son tres: piso, chalé y local) y una no válida (cualquier otro caso: por ejemplo, plaza de garaje).

- En cualquier caso, si se sospecha que ciertos elementos de una clase no se tratan igual que el resto de la misma, deben dividirse en clases menores.

La aplicación de estas reglas para la derivación de clases de equivalencia permite desarrollar los casos de prueba para cada elemento de datos del dominio de entrada. La división en clases deberían realizarla personas independientes al proceso de desarrollo del programa, ya que, si lo hace la persona que preparó la especificación o diseñó el software, la existencia de algunas clases (en concreto, las no consideradas en el tratamiento) no será, probablemente, reconocida.

El último paso del método es el uso de las clases de equivalencia para identificar los casos de prueba correspondientes. Este proceso consta de las siguientes fases:

 Asignación de un número único a cada clase de equivalencia.

 Hasta que todas las clases de equivalencia hayan sido cubiertas por (incorporadas a) casos de prueba, se tratará de escribir un caso que cubra tantas clases válidas no incorporadas como sea posible.

 Hasta que todas las clases de equivalencia no válidas hayan sido cubiertas por casos de prueba, escribir un caso para una única clase no válida sin cubrir.

La razón de cubrir con casos individuales las clases no válidas es que ciertos controles de entrada pueden enmascarar o invalidar otros controles similares. Por ejemplo, en un programa donde hay que “introducir cantidad (1-99) y letra inicial (A-Z)” ante el caso “105 &” (dos errores), se puede indicar sólo el mensaje “105 fuera de rango de cantidad” y dejar sin examinar el resto de la entrada (el error de introducir “&” en vez de una letra).

Veamos un ejemplo de aplicación de la técnica. Se trata de una aplicación bancaria en la que el operador deberá proporcionar un código, un nombre para que el usuario identifique la operación (por ejemplo, “nómina”) y una orden que disparará una serie de funciones bancarias.

Especificación

- Código área: número de 3 dígitos que no empieza por 0 ni por 1.
- Nombre de identificación: 6 caracteres.
- Órdenes posibles: “cheque”, “depósito”, “pago factura”, “retirada de fondos”.

Aplicación de las reglas

- Código
- Número, regla 3, booleana:
- Clase válida (número)
- Clase no válida (no es número)
- Regla 5, la clase número debe subdividirse; por la regla 1, rango, obtenemos:
- Subclase válida (200 código 999)
- Dos subclases no válidas (código < 200; código > 999)

- Nombre de id., número específico de valores, regla 2:
- Clase válida (6 caracteres)
- Dos clases no válidas (más de 6; menos de 6 caracteres)
- Orden, conjunto de valores, regla 4:
- Una clase válida para cada orden (“cheque”, “depósito”); 4 en total.
- Una clase no válida (“divisas”, por ejemplo).

En la tabla 1 se han enumerado las clases identificadas y la generación de casos (presuponiendo que el orden de entrada es código-nombre-orden) se ofrece a continuación.

Condición de entrada	Clases válidas	Clases inválidas
Código área	(1) 200 " código " 999	(2) código <200 (3) código >999 (4) no es número
Nombre para identificar la operación	(5) seis caracteres	(6) menos de 6 caracteres (7) más de 6 caracteres
Orden	(8) «cheque» (9) «depósito» (10) «pago factura» (11) «retirada fondos»	(12) ninguna orden válida

Tabla 1. Tabla de clases de equivalencia del ejemplo.

Casos válidos:

- 300 Nómina Depósito (1) (5) (9)
- 400 Viajes Cheque (1) (5) (8)
- 500 Coches Pago-factura (1) (5) (10)
- 600 Comida Retirada-fondos (1) (5) (11)
- Casos no válidos:
- 180 Viajes Pago-factura (2) (5) (10)
- 1032 Nómina Depósito (3) (5) (9)
- XY Compra Retirada-fondos (4) (5) (11)
- 350 A Depósito (1) (6) (9)
- 450 Regalos Cheque (1) (7) (8)
- 550 Casa &%4 (1) (5) (12)

3.2.2.2 Análisis de valores límite (AVL).

Mediante la experiencia (e incluso a través de demostraciones) se ha podido constatar que los casos de prueba que exploran las condiciones límite de un programa producen un mejor resultado para la detección de defectos, es decir, es más probable que los defectos del software se acumulen en estas condiciones. Podemos definir las condiciones límite como las situaciones que se hallan directamente arriba, abajo y en los márgenes de las clases de equivalencia.

El análisis de valores límites es un técnica de diseño de casos que complementa a la de particiones de equivalencia. Las diferencias entre ambas son las siguientes:

 Más que elegir “cualquier” elemento como representativo de una clase de equivalencia, se requiere la selección de uno o más elementos tal que los márgenes se sometan a prueba.

 Más que concentrarse únicamente en el dominio de entrada (condiciones de entrada), los casos de prueba se generan considerando también el espacio de salida.

El proceso de selección de casos es también heurístico, aunque existen ciertas reglas orientativas. Aunque parezca que el AVL es simple de usar (a la vista de las reglas), su aplicación tiene múltiples matices que requieren un gran cuidado a la hora de diseñar las pruebas. Las reglas para identificar clases son las siguientes:

 Si una condición de entrada especifica un rango de valores (“-l.0 valor +1.0”) se deben generar casos para los extremos del rango (-1.0 y +1.0) y casos no válidos para situaciones justo más allá de los extremos (- 1.001 y + 1.001, en el caso en que se admitan 3 decimales).

 Si la condición de entrada especifica un número de valores (“el fichero de entrada tendrá de 1 a 255 registros”), hay que escribir casos para los números máximo, mínimo, uno más del máximo y uno menos del mínimo de valores (0, 1, 255 y 256 registros).

 Usar la regla 1 para la condición de salida (“el descuento máximo aplicable en compra al contado será el 50%, el mínimo será el 6%”). Se escribirán casos para intentar obtener descuentos de 5,99%, 6%, 50% y 50,01 %.

 Usar la regla 2 para cada condición de salida (“el programa puede mostrar de 1 a 4 listados”). Se escriben casos para intentar generar 0, 1, 4 y 5 listados.

En esta regla, como en la 3, debe recordarse que:

- Los valores límite de entrada no generan necesariamente los valores límite de salida (recuérdese la función seno, por ejemplo).

- No siempre se pueden generar resultados fuera del rango de salida (pero es interesante considerarlo).

 Si la entrada o la salida de un programa es un conjunto ordenado (por ejemplo, una tabla, un archivo secuencial, etc.), los casos se deben concentrar en el primero y en el último elemento.

Es recomendable utilizar el ingenio para considerar todos los aspectos y matices, a veces sutiles, en la aplicación del AVL.

3.2.2.3 Conjetura de errores.

La idea básica de esta técnica consiste en enumerar una lista de equivocaciones que pueden cometer los desarrolladores y de las situaciones propensas a ciertos errores. Después se generan casos de prueba en base a dicha lista (se suelen corresponder con defectos que aparecen comúnmente y no con aspectos funcionales). Esta técnica también se ha denominado generación de casos (o valores) especiales, ya que no se obtienen en base a otros métodos sino mediante la intuición o la experiencia.

No existen directrices eficaces que puedan ayudar a generar este tipo de casos, ya que lo único que se puede hacer es presentar algunos ejemplos típicos que reflejan esta técnica. Algunos valores a tener en cuenta para los casos especiales son los siguientes:

- El valor cero es una situación propensa a error tanto en la salida como en la entrada.
- En situaciones en las que se introduce un número variable de valores (por ejemplo, una lista), conviene centrarse en el caso de no introducir ningún valor y en el de un solo valor. También puede ser interesante una lista que tiene todos los valores iguales.
- Es recomendable imaginar que el programador pudiera haber interpretado algo mal en la especificación.
- También interesa imaginar lo que el usuario puede introducir como entrada a un programa. Se dice que se debe proveer toda clase de acciones de un usuario como si fuera “completamente tonto” o, incluso, como si quisiera sabotear el programa.

3.2.3 Pruebas aleatorias.

En estas simulamos la entrada habitual del programa creando datos de entrada en la secuencia y con la frecuencia con las que podrían aparecer en la práctica, de forma continua sin parar., Esto implica usar una herramienta denominada un generador de pruebas, a las que se alimenta con una descripción de las entradas y las secuencias de entrada posibles y su probabilidad de ocurrir en la práctica.

Este enfoque de prueba es muy común en la prueba de compiladores en la que se generan aleatoriamente códigos de programas que sirven de casos de prueba para la compilación.

Si el proceso de generación se ha realizado correctamente, se crearán eventualmente todas las posibles entradas del programa en todas las posibles combinaciones y permutaciones. También se puede conseguir, indicando la distribución estadística que siguen, que la frecuencia de las entradas sea la apropiada para orientar correctamente nuestras pruebas hacia lo que es probable que suceda en la práctica. No obstante, esta forma de diseñar casos de prueba es menos utilizada que las técnicas de caja blanca y de caja negra.

3.3 Requerimientos a probar

En la lista a continuación se presentan los elementos, casos de uso, requerimientos funcionales y requerimientos no funcionales, que serán verificados.

La integración de LIMS con los sistemas de la empresa debe estar basada en los siguientes requerimientos:

Lista de los requerimientos más importantes a ser verificados. En esta sección indique qué elementos serán verificados. (18)

Diseño: basado en la web, con rastro cero de cliente para mejorar las actualizaciones, el mantenimiento y la validación en una gran base de usuarios.

Posibilidades de configuración: flexibles para los distintos tipos de laboratorio, sin codificación.

Funcionalidad: robusto, estable y fiable para garantizar una utilización a gran escala sin problemas.

Integración: servicios de Internet para la integración con distintos instrumentos y aplicaciones de empresa.
Internacionalización: disponible en múltiples idiomas
Informes: informes realizados a través de Internet, lo cual ofrece incluso a los usuarios ocasionales de fuera del laboratorio el acceso a los datos pertinentes.
Posibilidad de graduarlo: capaz de soportar un amplio y creciente número de usuarios y laboratorios.
Seguridad: avanzado sistema de permisos para garantizar un acceso seguro a los datos en toda la organización.
Estándares: herramientas, idiomas y bases de datos estándar en el sector.
Soporte: disponible en múltiples regiones e idiomas Acceso a Internet: acceso remoto para la entrega de muestras, la visualización de resultados.

3.4 Estrategia de Pruebas

[Esta sección presenta el enfoque recomendado para la verificación. Describe como se verificarán los elementos.

Para cada tipo de prueba, proporcione una descripción de la prueba y porque será implementada y ejecutada.

Si un tipo de prueba no será implementada y ejecutada, indique brevemente cual es la prueba que no se implementará o ejecutará y una justificación de ello.

Se indicarán las técnicas usadas y el criterio para saber cuando una prueba se completó (criterio de aceptación).

Las pruebas se deben ejecutar usando bases de datos conocidas y controladas en un ambiente seguro.]

3.5 Tipos de pruebas especificas para producto LIMS.

[En las secciones a continuación, que son los tipos de pruebas a realizar, para cada tipo de prueba en lugar de explicar el contenido de cada sección y subsección se incluyen ejemplos.]

3.5.1 Prueba de integridad de los datos y la base de datos

3.5.1.1 Objetivo de la prueba

[Asegurar que los métodos y procesos de acceso a la base de datos funcionan correctamente y sin corromper datos

3.5.1.2 Técnica

[Invoque cada método o proceso de acceso a la base de datos con datos válidos y no válidos.]

[Inspeccione la base de datos para asegurarse de que se han guardado los datos correctos, que todos los eventos de la base de datos ocurrieron correctamente, o repase los datos devueltos para asegurar que se recuperaron datos correctos por la vía correcta.]

3.5.1.3 Criterio de aceptación

[Todos los métodos y procesos de acceso a la base de datos funcionan como fueron diseñados y sin datos corruptos.]

3.5.1.4 Consideraciones especiales

[La prueba requiere un entorno de administración de DBMS o controladores para ingresar o modificar información directamente en la base de datos.

Los procesos deben ser invocados manualmente.

Se deben usar bases de datos pequeñas para aumentar la facilidad de inspección de los datos para verificar que no sucedan eventos no aceptables.]

3.5.1.5 Fase de ejecución: Construcción.

3.5.2 Prueba de Funcionalidad

[La prueba de funcionalidad se enfoca en requerimientos para verificar que se corresponden directamente a casos de usos o funciones y reglas del negocio. Los objetivos de estas pruebas son verificar la aceptación de los datos, el proceso, la recuperación y la implementación correcta de las reglas del negocio. Este tipo de prueba se basa en técnicas de caja negra, que consisten en verificar la aplicación y sus procesos interactuando con la aplicación por medio de la interfaz de usuario y analizar los resultados obtenidos.]

3.5.2.1 Objetivo de la prueba

[Asegurar la funcionalidad apropiada del objeto de prueba, incluyendo la navegación, entrada de datos, proceso y recuperación.]

3.5.2.2 Técnica

[Ejecute cada caso de uso, flujo de caso de uso, o función usando datos válidos y no válidos, para verificar lo siguiente:

Se obtienen los resultados esperados cuando se usan datos válidos.

Cuando se usan datos no válidos se despliegan los mensajes de error o advertencia apropiados.

Se aplica apropiadamente cada regla del negocio.]

3.5.2.3 Criterio de aceptación

[Todas las pruebas planificadas se realizaron. Todos los defectos encontrados han sido debidamente identificados.]

3.5.2.4 Consideraciones especiales

[Identificar o describir aquellos elementos o problemas (internos o externos) que impactaron en la implementación y ejecución de las pruebas de funcionalidad.]

3.5.2.5 Fase de ejecución: Construcción.

3.5.3 Prueba de Ciclo del Negocio

[Esta prueba debe simular las actividades realizadas en el proyecto en el tiempo. Se debe identificar un período, que puede ser un año, y se deben ejecutar las transacciones y actividades que ocurrirían en el período de un año. Esto incluye todos los ciclos diarios, semanales y mensuales y eventos que son sensibles a la fecha.]

3.5.3.1 Objetivo de la prueba

[Asegurar que la aplicación funciona de acuerdo a los requerimientos del negocio.]

3.5.3.2 Técnica

[La prueba debe simular ciclos de negocios realizando lo siguiente:

Las pruebas de funcionalidad se deben modificar para aumentar la cantidad de veces que se ejecuta cada función, simulando varios usuarios diferentes en un período determinado.

Todas las funciones sensibles a la fecha se deben ejecutar con fechas válidas y no válidas o períodos de tiempo válido y no válido.

Para cada prueba realizada verificar lo siguiente:

Se obtienen los resultados esperados cuando se usan datos válidos.

Cuando se usan datos no válidos se despliegan los mensajes de error o advertencia apropiados.

Se aplica apropiadamente cada regla del negocio.]

3.5.3.3 Criterio de aceptación

[Todas las pruebas planificadas se realizaron. Todos los defectos encontrados han sido debidamente identificados.]

3.5.3.4 Consideraciones especiales

[Las fechas del sistema y eventos requieren actividades de soporte especiales. Se requieren las reglas del negocio para identificar apropiadamente los requerimientos y procedimientos a ser verificados.]

3.5.3.5 Fase de ejecución: Construcción.

3.5.4 Prueba de Interfaz de Usuario

[Esta prueba verifica que la interfaz de usuario proporcione al usuario el acceso y navegación a través de las funciones apropiada. Además asegura que los objetos presentes en la interfaz de usuario se muestren como se espera y conforme a los estándares establecidos por la empresa o de la industria.]

3.5.4.1 Objetivo de la prueba

[Verificar que: la navegación a través de los elementos que se están probando reflejen las funciones del negocio y los requerimientos, incluyendo manejo de ventanas, campos y métodos de acceso; los objetos de las ventanas y características, como menús, tamaño, posición, estado funcionen de acuerdo a los estándares.]

3.5.4.2 Técnica

[Crear o modificar pruebas para cada ventana verificando la navegación y los estados de los objetos para cada ventana de la aplicación y cada objeto dentro de la ventana.]

3.5.4.3 Criterio de aceptación

[Cada ventana ha sido verificada exitosamente siendo consistente con una versión de referencia o estándar establecido.]

3.5.4.4 Consideraciones especiales

[No todas las propiedades de los objetos se pueden acceder.]

3.5.4.5 Fase de ejecución: Transición

3.5.5 Prueba de Performance

[En esta prueba se miden y evalúan los tiempos de respuesta, los tiempos de transacción y otros requerimientos sensitivos al tiempo. El objetivo de la prueba es verificar que se logren los requerimientos de performance. La prueba de performance es implementada y ejecutada para poner a punto los destinos de pruebas de performance como función de condiciones de trabajo o configuraciones de hardware.]

3.5.5.1 Objetivo de la prueba

[Verificar la performance de determinadas transacciones o funciones de negocio bajo ciertas condiciones:

- condiciones de trabajo normales conocidas.
- peores casos de condiciones de trabajo conocidas.]

3.5.5.2 Técnica

[Usar procedimientos de prueba desarrollados para verificar funciones o ciclos de negocio.

Modificar archivos de datos para aumentar el número de transacciones o los procedimientos de prueba para aumentar el número de iteraciones de ocurrencia de transacciones.

Las pruebas se deben ejecutar en una máquina (mejor caso de prueba un solo usuario, una sola transacción) y se debe repetir con múltiples usuarios (virtuales o reales).]

3.5.5.3 Criterio de aceptación

[Con una transacción o un usuario: Éxito completo de la prueba sin fallas y dentro del tiempo esperado o requerido.

Con múltiples transacciones y varios usuarios: Éxito completo de la prueba sin fallas y dentro de un tiempo aceptable.]

3.5.5.4 Consideraciones especiales

[Las pruebas de performance deben incluir un trabajo de fondo en el servidor. Esto se puede realizar de distintas formas:

Enviar transacciones directamente al servidor, generalmente en la forma de consultas (SQL).

Crear usuarios virtuales para simular muchos clientes, generalmente varios cientos. Se pueden usar herramientas de Emulación de Terminar Remota para lograr este objetivo. Esta técnica también se usa para cargar la red con “trafico”.

Usar muchos clientes físicos, cada uno corriendo procedimientos de prueba.

La prueba de performance se debe realizar en una máquina dedicada para permitir control total y medición exacta. Las bases de datos usadas para las pruebas de performance deben tener un tamaño similar a las reales.]

3.5.5.5 Fase de ejecución: Transición.

3.5.6 Prueba de Carga

[La prueba de carga somete los objetos a verificar a diferentes cargas de trabajo para medir y evaluar los comportamientos de performance y la habilidad de los objetos de continuar funcionando apropiadamente bajo diferentes cargas de trabajo. El objetivo es determinar y asegurar que el sistema funciona apropiadamente en circunstancias de máxima carga de trabajo esperada. Además evaluar las características de performance, como tiempos de respuesta, tiempos de transacciones y otros elementos sensitivos al tiempo.]

3.5.6.1Objetivo de la prueba

[Verificar el comportamiento de performance de determinados componentes del software bajo condiciones de trabajo diferentes.]

3.5.6.2 Técnica

[Usar pruebas desarrolladas para funciones o ciclos de negocios y modificar archivos de datos para aumentar el número de transacciones o las pruebas para aumentar la cantidad de ocurrencia de transacciones.]

3.5.6.3 Criterio de aceptación

[Para múltiples transacciones y múltiples usuarios: Realización exitosa de las pruebas sin fallas y dentro del tiempo aceptable.]

3.5.6.4 Consideraciones especiales

[La prueba de carga debe realizarse en una máquina dedicada para tener control total y exactitud de mediciones.

Las bases de datos usadas para la prueba deben tener un tamaño similar a las reales.]

3.5.6.5 Fase de ejecución: Transición.

3.5.7 Prueba de Esfuerzo (stress, competencia por recursos, bajos recursos)

[La prueba de esfuerzo en un tipo de prueba de performance implementada y ejecutada para encontrar errores cuando hay pocos recursos o cuando hay competencia por recursos. Poca memoria o poco espacio de disco pueden revelar fallas en el software que no aparecen bajo condiciones normales de cantidad de recursos. Otras fallas pueden resultar al competir por recursos compartidos como bloqueos de bases de datos o ancho de banda de red. La prueba de esfuerzo también puede usarse para identificar el trabajo máximo que el software puede manejar.]

3.5.7.1 Objetivo de la prueba

[Verificar que el software funciona apropiadamente y sin error bajo condiciones de esfuerzo, como son:

- poca memoria o sin disponibilidad de memoria en el servidor
- cantidad máxima de clientes conectados
- múltiples usuarios realizando la misma operación sobre los mismos datos
- peor caso de volumen de operaciones.

El objetivo de la prueba de esfuerzo es también identificar y documentar las condiciones bajo las cuales el sistema falla y no continua funcionando apropiadamente.]

3.5.7.2 Técnica

[Usar las pruebas desarrolladas para Performance y Prueba de Carga.

Para probar recursos limitados, las pruebas se deben ejecutar en una sola máquina, y se debe reducir o limitar la memoria en el servidor.

Para las pruebas de esfuerzo restantes, deber usarse múltiples clientes, cualquiera que ejecute las mismas pruebas o pruebas complementarias para producir el peor caso de volumen de operaciones.]

3.5.7.3 Criterio de aceptación

[Todas las pruebas planeadas se ejecutaron y se alcanzaron o excedieron los límites del sistema sin que el software fallara o las condiciones bajo las que ocurre una falla en el software están fuera de las condiciones especificadas.]

3.5.7.4 Consideraciones especiales

[Las pruebas de esfuerzo de red pueden requerir herramientas de red para cargar la red con mensajes o paquetes.

La cantidad de disco del servidor usada por el sistema debe ser reducida temporalmente para restringir el espacio disponible para crecimiento de la base de datos.

Sincronizar el acceso simultáneo de varios clientes accediendo a los mismos datos.]

3.5.7.5 Fase de ejecución: Transición.

3.5.8 Prueba de Volumen

[La Prueba de Volumen somete el software a grandes cantidades de datos para determinar si se alcanzan límites que causen la falla del software. La Prueba de Volumen identifica la carga máxima continua que puede manejar el software a prueba en un período dado.]

3.5.8.1Objetivo de la prueba

[Verificar que el software funciona correctamente con volúmenes de datos grandes:

Máximo (real o físicamente posible) número de clientes conectados, o simulados, todos realizando la misma operación (peor caso de operación) por un período de tiempo extenso.

Máximo tamaño de base de datos y múltiples consultas ejecutadas simultáneamente.]

3.5.8.2 Técnica

[Usar pruebas desarrolladas para Prueba de Performance y Prueba de Carga.

Se deben usar múltiples clientes, ejecutando las mismas pruebas o pruebas complementarias para producir el peor caso de volumen de operaciones o mezcla en un período de tiempo extenso.

Se debe crear el tamaño máximo de base de datos (real, escalado o con datos representativos) y múltiples clientes ejecutando consultas simultáneamente por un período de tiempo extenso.]

3.5.8.3 Criterio de aceptación

[Todas las pruebas planificadas se ejecutaron y se han alcanzado o excedido los límites especificados sin que el software falle.]

3.5.8.4 Consideraciones especiales

[¿Qué período de tiempo se considera aceptable para condiciones de gran volumen?]

3.5.8.5 Fase de ejecución: Construcción.

3.5.9 Prueba de Seguridad y Control de Acceso

[La Prueba de Seguridad y Control de Acceso se enfoca en dos áreas de seguridad:

Seguridad en el ámbito de aplicación, incluyendo el acceso a los datos y a las funciones de negocios.

Seguridad en el ámbito de sistema, incluyendo conexión, o acceso remoto al sistema.

La seguridad en el ámbito de aplicación asegura que, basado en la seguridad deseada los actores están restringidos a funciones o casos de uso específicos o limitados en los datos que están disponibles para ellos.

La seguridad en el ámbito de sistema asegura que, solo los usuarios con derecho a acceder al sistema son capaces de acceder a las aplicaciones y solo a través de los puntos de ingresos apropiados.]

3.5.9.1 Objetivo de la prueba

Seguridad en el ámbito de aplicación:[Verificar que un actor pueda acceder solo a las funciones o datos para los cuales su tipo de usuario tiene permiso.]

Seguridad en el ámbito de sistema: [Verificar que solo los actores con acceso al sistema y a las aplicaciones, puedan acceder a ellos.]

3.5.9.2 Técnica

Seguridad en el ámbito de aplicación: [Identificar y hacer una lista de cada tipo de usuario y las funciones y datos sobre las que cada tipo tiene permiso.]

[Crear pruebas para cada tipo de usuario y verificar cada permiso creando operaciones específicas para cada tipo de usuario.]

[Modificar el tipo de usuario y volver a ejecutar las pruebas para los mismos usuarios. En cada caso, verificar que las funciones o datos adicionales están correctamente disponibles o son denegados.

Acceso en el ámbito de sistema: [Ver consideraciones especiales más abajo.]

3.5.9.3 Criterio de aceptación

[Para cada tipo de actor conocido las funciones y datos apropiados están disponibles, y todas las operaciones funcionan como se espera y ejecutan las pruebas de Funcionalidad de la aplicación.]

3.5.9.4 Consideraciones especiales

[El acceso al sistema debe ser discutido con el administrador del sistema o la red. Esta prueba no puede requerirse como tal, es una función del administrador del sistema o de la red.]

3.5.9.5 Fase de ejecución: Transición.

3.5.10 Prueba de Fallas y Recuperación

[Las Pruebas de Fallas y Recuperación aseguran que el software puede recuperarse de fallas de hardware, software o mal funcionamiento de la red sin pérdida de datos o de integridad de los datos.

La Prueba de Recuperación es un proceso en el cual la aplicación o sistema se expone a condiciones extremas, o condiciones simuladas, para causar falla, como fallas en dispositivos de Entrada/Salida o punteros a la base de datos inválidos. Los procedimientos de recuperación se

invocan y la aplicación o sistema es monitoreado e inspeccionado para verificar que se recupera apropiadamente la aplicación o sistema y se logre la recuperación de datos.]

3.5.10.1 Objetivo de la prueba

[Verificar que los procesos de recuperación (manual o automáticos) recuperen apropiadamente la base de datos, aplicaciones y sistema a un estado conocido y deseado. En la prueba se incluyen los siguientes tipos de condiciones:

- interrupción de energía al cliente
- interrupción de energía al servidor
- interrupción de comunicaciones mediante los servidores de la red
- interrupción de comunicación o pérdida de energía de los discos del servidor o con los controladores
- ciclos incompletos (procesos de filtro de datos interrumpidos, procesos de sincronización de datos interrumpidos)
- punteros a la base de datos o claves inválidos
- elementos de datos en la base de datos inválidos o corruptos.]

3.5.10.2 Técnica

[Se deben usar las pruebas creadas para probar Funcionalidad y Ciclos de negocio para crear una serie de operaciones. Una vez logrado el punto de comienzo deseado, se deben realizar o simular las siguientes acciones, individualmente:

Interrumpir la energía del cliente: apagar el PC.

Interrumpir la energía del servidor: simular o iniciar el proceso de apagado del servidor.

Interrupción por medio de los servidores de red: simular o iniciar la pérdida de comunicación con la red (desconectar físicamente la comunicación o apagar el servidor de red o router

Interrumpir la comunicación o quitar la energía de los discos del servidor o sus controladores: simular o eliminar físicamente la comunicación con uno o más controladores de disco o los discos.]

Una vez que se lograron o simularon estas condiciones, se deben invocar los procedimientos de recuperación.

Las pruebas de ciclos incompletos utilizan la misma técnica excepto que los procesos de bases de datos deben ser abortados a sí mismos o terminados prematuramente.

Las últimas dos pruebas requieren que se logre un estado conocido de la base de datos. Se deben corromper manualmente campos de la base de datos, punteros y claves trabajando directamente sobre la base de datos (utilizando herramientas para la base de datos).

Se deben ejecutar las pruebas de Funcionalidad y Ciclo de negocio y verificar que los ciclos se completen.]

3.5.10.3 Criterio de aceptación

[En todos los casos, la aplicación, la base de datos y el sistema deben, en la realización procedimientos de recuperación, volver a un estado conocido y deseable. Este estado incluye corrupción de datos limitada al los campos, punteros o claves corruptos conocidos, y reportes indicando los procesos u operaciones que no se completaron debido a las interrupciones.]

3.5.10.4 Consideraciones especiales

[Los procedimientos para desconectar cables (simulando falta de energía o pérdida de comunicación) no son deseables o factibles. Se pueden requerir métodos alternativos, como software de diagnóstico. Se requieren los grupos de recursos de Sistemas, Bases de datos y Red.

Estas pruebas deben ejecutarse fuera del horario de trabajo normal o en una máquina aislada.]

3.5.10.5 Fase de ejecución: Transición

3.5.11 Prueba de Configuración

[La Prueba de Configuración verifica el funcionamiento del software con diferentes configuraciones de software y hardware.]

3.5.11.1 Objetivo de la prueba

[Verificar que el software funcione apropiadamente en las configuraciones requeridas de hardware y software.]

3.5.11.2 Técnica

[Usar las pruebas de Funcionalidad.

Abrir y cerrar varias sesiones de software que no son objeto de prueba, como parte de la prueba o antes de comenzar la prueba.

Ejecutar operaciones seleccionadas para simular la interacción del actor con el software objeto de prueba y con el software que no es objeto de prueba.

Repetir los procedimientos anteriores minimizando la memoria convencional disponible en la máquina cliente.]

3.5.11.3 Criterio de aceptación

[Por cada combinación de software objeto de prueba y software que no es objeto de prueba, todas las operaciones son completadas exitosamente sin fallas.]

3.5.11.4 Consideraciones especiales

[Todo el software que no es objeto de prueba que es necesario y debe estar accesible.

¿Qué aplicaciones se usan normalmente?

¿Qué información se maneja en las aplicaciones que se usan normalmente, y que tamaño de información?

Los sistemas, red, servidores de red, bases de datos, etc., deben ser documentados como parte de esta prueba.]

3.5.11.5 Fase de ejecución: Transición.

3.5.12 Prueba de Instalación

[La Prueba de Instalación tiene dos propósitos. Uno es asegurar que el software puede ser instalado en diferentes condiciones (como una nueva instalación, una actualización, y una instalación completa o personalizada) bajo condiciones normales y anormales. Condiciones anormales pueden ser insuficiente espacio en disco, falta de privilegios para crear directorios, etc.

El otro propósito es verificar que, una vez instalado, el software opera correctamente. Esto significa normalmente ejecutar un conjunto de pruebas que fueron desarrolladas para Prueba de Funcionalidad.]

3.5.12.1 Objetivo de la prueba

[Verificar que el software objeto de prueba se instala correctamente en cada configuración de hardware requerida bajo las siguientes condiciones:

- instalación nueva, una nueva máquina, nunca instalada previamente con [Nombre del proyecto]
- actualización, máquina previamente instalada con [Nombre del proyecto], con la misma versión
- actualización, máquina previamente instalada con [Nombre del proyecto], con una versión anterior.]

3.5.12.2 Técnica

[Manualmente o desarrollando programas, para validar la condición de la máquina destino (nueva, nunca instalado, misma versión, versión anterior ya instalada).

Realizar la instalación.

Ejecutar un conjunto de pruebas funcionales ya implementadas para la Prueba de Funcionalidad.]

3.5.12.3 Criterio de aceptación

[Las pruebas de funcionalidad de [Nombre de proyecto] se ejecutan exitosamente sin fallas.]

3.5.12.4 Consideraciones especiales

[¿Qué operaciones se deben ser seleccionar para realizar una prueba confiable de que la aplicación [Nombre del proyecto] ha sido exitosamente instalada sin dejar fuera ningún componente importante?]

3.5.12.5 Fase de ejecución: Transición.

3.5.13 Prueba de Documentos

[La Prueba de Documentos debe asegurar que los documentos relacionados al software que se generen en el proceso sean correctos, consistentes y entendibles. Se incluyen como documentos los Materiales para Soporte al Usuario, Documentación Técnica, Ayuda en Línea y todo tipo de documento que forme parte del paquete de software.]

3.5.13.1 Objetivo de la prueba

[Verificar que el documento objeto de prueba sea:

Correcto, esto es, que cumpla con el formato y organización para el documento establecido en el proyecto.

Consistente, esto es, que el contenido del documento sea fiel a lo que hace referencia. Si el documento es Documentación de Usuario, que la explicación de un procedimiento sea exactamente como se realiza el procedimiento en el software, si se muestran pantallas que sean las correctas.

Entendible, esto es, que al leer el documento se entienda correctamente lo que expresa y sin ambigüedades, además que sea fácil de leer.]

3.5.13.2 Técnica

[Para verificar que el documento es correcto se debe comparar con el estándar definido si existe o con las pautas de documentación y ver que el documento cumple con ellas.

Para verificar que el documento es Consistente se debe ejecutar el programa siguiendo el documento en caso de los Materiales de Soporte al Usuario y comprobar que lo que se explica en estos documentos es exactamente lo que se ejecuta en el programa. En caso de Documentación Técnica se debe revisar el código al cual corresponde la documentación y comprobar que dicha describe el código.

Para verificar que el documento es entendible, debe comprobar que se entiende correctamente, que no tiene ambigüedades y que sea fácil de leer.]

3.5.13.3 Criterio de aceptación

[El documento expresa exactamente lo que debe expresar, no hay diferencias entre lo que está escrito y el objeto de la descripción (operación de software, código de programa, decisiones técnicas) y se entiende fácilmente.]

3.5.13.4 Consideraciones especiales

[Enumere las consideraciones que considere importantes para la verificación de documentos]

3.5.13.5 Fase de ejecución: Inicio, Elaboración, Construcción, Transición.

3.6 Planificación de las pruebas propuestas

Esta sección presenta una planificación para la realización de las pruebas específicas propuestas para un producto LIMS. Las cuales a pesar de ser muy novedosas en nuestro país y en especifico en nuestra universidad si se ha trabajado en algunas de las mas importantes aquí propuestas.

Para realizar este planteamiento nos basamos en la experiencias adquiridas a lo largo del curso del proyecto en el que nos desempeñamos como ingenieros de prueba, por citar un ejemplo podemos mencionar el software vendido al hermano país de Venezuela, Registro y Notaria, donde tuvimos la oportunidad de participar en las pruebas realizadas al mismo y tomarlo hoy como referencia para realizar este planteamiento empírico, que no deja de estar cerca de la realidad, ya que fue elaborado con una base sólida.

Se señala la fase de desarrollo, según RUP, más idónea en la cual se podría ir aplicando cada prueba durante todo el proceso de desarrollo del sistema.

Se debe decir que esta propuesta puede variar en cierto rango, su flexibilidad depende de muchos factores como el tiempo que se tenga para llevar a cabo todo este proceso de prueba, así como el factor equipamiento de trabajo, preparación del personal y magnitud del sistema.

NOMBRE DE LA PRUEBA	# DE ITERACIONES	INTERVALO DE TIEMPO	# DE PROBADORES	FASES DE DESARROLLO SEGÚN RUP	PROBADORES: NOMBRE Y APELLIDOS
1.Prueba de integridad de los datos y la base de datos	**10**	**24h**	**4**	**Construcción**	
2.Prueba de Funcionalidad	**6**	**72h**	**6**	**Construcción**	

3.Prueba de Ciclo del Negocio	**4**	**48h**	**2**	**Transición**	
4.Prueba de Interfaz de Usuario	**8**	**12h**	**3**	**Transición**	
5.Prueba de Performance	**6**	**48h**	**2**	**Transición**	
6.Prueba de Carga	**10**	**36h**	**1**	**Transición**	
7.Prueba de Esfuerzo	**6**	**24h**	**1**	**Transición**	
8.Prueba de Volumen	**8**	**24h**	**2**	**Construcción**	
9.Prueba de Seguridad y Control de Acceso	**12**	**8h**	**3**	**Transición**	
10.Prueba de Fallas y Recuperación	**10**	**12h**	**2**	**Transición**	
11.Prueba de Configuración	**4**	**36h**	**1**	**Transición**	
12.Prueba de Instalación	**5**	**48h**	**1**	**Transición**	
13.Prueba de Documentos	**12**	**72h**	**2**	**Inicio Elaboración Construcción Transición**	

3.7 Conclusiones

En este capítulo se expuso una breve descripción general de los tres enfoques principales de prueba existentes(caja blanca, caja negra y el enfoque aleatorio, también se abordó la necesidad de probar y una explicación mas detallada de las pruebas especificas que deben de aplicársele a un producto LIMS ,dejando esclarecido el objetivo de la cada una de las pruebas y las técnica utilizadas, Para la propuesta de estas pruebas se tuvieron en cuenta un conjunto de puntos clave para lograr el buen desarrollo de las mismas y su aplicación con la mayor calidad.

Entre esos puntos clave podemos citar los siguientes:
Las pruebas intentan descubrir la presencia de errores, no la ausencia de los mismos.
Asegurar que los métodos y procesos de acceso a la base de datos funcionan correctamente y sin corromper datos.
Se deben probar los componentes individualmente y luego se procede a su integración para crear el sistema.
Las pruebas de caja negra se basan en la especificación del sistema.

Conclusiones

Dada la importancia de obtener una mejora continua en productos, procesos y servicios, en este documento se presentan las características generales del proceso que se lleva a cabo para aplicar las pruebas de calidad a un producto LIMS. Así mismo, se expone brevemente la historia de la aplicación del control de calidad en el mundo, la participación de los organismos de normalización y aspectos importantes de la extensión CFR 21 Parte 11. Como requisito fundamental, se describen las características primordiales de un sistema de calidad, además de otros requisitos técnicos y administrativos que debe cumplir un sistema LIMS.

Se argumentó la necesidad de que los productos realizados sean confiables y precisos, crece la exigencia por parte de clientes y usuarios en general, de que a los sistemas se les hayan aplicado las técnicas de Ingeniería de Software adecuadas, y en su etapa de comprobación hayan sido probados con las técnicas necesarias para lograr el nivel de calidad requerido**.**

Es por ello que resulta vital, que se tenga en cuenta como parte importantísima del proceso de realización de sistemas, velar por que la calidad de los mismos, esté a la altura del desarrollo que se exige en estos tiempos.

Al planificarse las pruebas para el producto LIMS control de calidad del CIGB, se llegaron a las siguientes conclusiones:

- Se debe aplicar rigurosamente la propuesta de este grupo de pruebas de calidad que se ajustan a las normas ISO y FDA para este tipo de software.
- Las pruebas a los productos en la etapa de desarrollo es un factor de gran influencia en la calidad y por ende en la aceptación del producto en el mercado, y por parte de los clientes.
- Se alcanzo un amplio conocimiento acerca de las técnicas de pruebas existentes, las estrategias para su aplicación y pasos a seguir.

- Se plantearon pruebas como la de seguridad de los datos y la base de datos que serán las encargadas de asegurar la parte principal de un LIMS, que sin duda son los datos e información que en la misma será almacenada.
- Se logró desarrollar una adecuada documentación de todo el proceso de prueba propuesto para este producto, que servirá de apoyo para la corrección de los errores encontrados en cada uno de los módulos probados.
- Podemos decir que llevando un proceso de prueba a cabalidad y con seriedad según la guía aquí propuesta, se poseerá mayor calidad y confiabilidad el producto LIMS sin duda alguna.
- Los desarrolladores deben de tomar conciencia de la influencia de este proceso en los resultados finales de los productos y la importancia del mismo para lograr competir en el mercado.

Por demás podemos plantear que constituye una necesidad para la industria cubana de software, definir y aplicar un riguroso sistema que incluya todos los criterios para determinar la calidad, específicamente el criterio de pruebas, pues es un punto clave en la determinación del nivel de calidad de cualquier software. Siendo esta la manera en la que podemos garantizar que las empresas puedan entrar a competir en el mercado mundial de la industria de la informática y aportar al país un gran beneficio económico.

Con el estudio realizado y la propuesta de pruebas que se han llevado a cabo en el trabajo “Propuestas de pruebas de calidad para producto LIMS control de calidad del CIGB”, se ha cumplido el objetivo propuesto al inicio del trabajo el cual consistió en proponer las pruebas del producto LIMS, con vistas a la obtención del mayor número de faltas existentes en la misma, logrando con su ulterior eliminación obtener una aplicación con un mínimo de errores y por consiguiente mayor calidad.

Recomendaciones

Luego de haber estudiado el proceso de pruebas y conocer las facilidades que brinda a los desarrolladores de los sistemas una adecuada detección de errores, en el momento oportuno se recomienda:

- ✓ Aplicar las pruebas al LIMS antes de su definitiva implantación.
- ✓ Utilizar los argumentos, casos ejemplos y resultados de este trabajo para lograr que los grupos de desarrollo tengan conocimiento de las técnicas de prueba y que las incluyan como una fase más dentro del desarrollo de los sistemas.
- ✓ Que este trabajo sirva de complemento a la bibliografía que se emplea en las asignaturas Ingeniería de Software y Administración de Empresas, que abarcan temas de calidad dentro de la Universidad de las Ciencias Informáticas.
- ✓ Utilizar el Plan de Pruebas propuesto para el Control de la Calidad en los proyectos de Bioinformática, pues se fundamentan tanto en las normas ISO, como en las regulaciones de la FDA.
- ✓ Evitar que el programador pruebe sus propios programas, ya que intentaría (consciente o inconscientemente) demostrar que funcionan sin problemas.
- ✓ Al generar casos de prueba, se deben incluir tanto datos de entrada válidos y esperados como no válidos e inesperados.
- ✓ Las pruebas deben centrarse en dos objetivos
    1. Se deben evitar los casos desechables, es decir, los no documentados ni diseñados con cuidado.
    2. No deben hacerse planes de prueba suponiendo que, prácticamente, no hay defectos en los programas y, por lo tanto, dedicando pocos recursos a las pruebas.
- ✓ Las pruebas se deben realizar en el entorno en el que se utilizará el sistema (lo que incluye al personal que lo maneja).

.

Bibliografía

http://gidis.ing.unlpam.edu.ar/downloads/pdfs/Calidad_software.PDF(15/10/2006)

http://bvs.sld.cu/revistas/aci/vol3_3_95/aci05395.htm.(aci05395 (1/3/2007)

http://www.calidaddelsoftware.com/eventos/solopruebas2006.pdf(15/12/2006)

http://www.calidaddelsoftware.com/modules.php?name=News&file=article&sid=156(13/2/2007)

http://www.thermo.com/eThermo/CMA/PDFs/Product/productPDF_24068.pdf.producto(16/12/2006)

http://www.imt.mx/Espanol/Publicaciones/pubtec/pt185.pdf.laboratorios.normas(15/5/2007)

http://bibliotecnica.upc.es/PFC/arxius/migrats/36146-2.pdf.tesis.industrial (14/3/2007)

http://www.ati.es/novatica/2000/145/juagra-145.pdf(15/3/2007)

http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html(27/1/2007

http://homepages.mty.itesm.mx/al1031749/Prueba_Testing.doc. (15/12/2006)

http://www.cic.ipn.mx/aguzman/papers/165%20mitos,%20creencias%20y%20superst%20sobre%20la%20calidad%20del%20software.pdf(17/3/2007)

http://www.monografias.com/trabajos20/pruebas-de-software/pruebas-software.shtml#teoroico(9/4/2007)

http://thermofinnigan.com/eThermo/CMA/PDFs/Product/productPDF_24068.pdf(12/5/2007)

http://www.quiminet.com.mx/ar0/ar_%25FC%2512%25FD%251D%2592%2512%2517%2587.htm(11/2/2007)

http://www.als-es.com/home.php?location=herramientas/entorno-pruebas#contenido(24/3/2007)

Metodología DESOFT S.A Ciudad de la Habana 2005 (Grupo de autores) (21/3/2007)

Referencias Bibliográficas

1. **Alvares, S., Febles, A. y Fernández, H.** *Aplicación del modelo CMM a la empresa Segurimática.* Facultad de Ingeniería Industrial, ISPJAE. Cuidad de la Habana : s.n., 2000. Tesis.

2. *Discurso pronunciado en la inauguración de la Convención Informática 2000.* **Lage, Carlos.** Ciudad de la Habana : s.n., 2000.

3. *Lineamientos estratégicos para la informatización de la sociedad cubana.* SIME. 1997.

4. **Febles, Ailyn.** *Case corporativo para el proceso de control de cambios.* CUJAE. Ciudad de la Habana : s.n., 2000. Tesis de maestría.

5. **Baeza, Yates.** *¿The Jaguar of the Pacific Rim?, Communications of the ACM.* 1995. Vol. 38.

6. **Hernández, Yanko y Banderas, Idael.** *Case para la planificacion y control de configuración de software.* Instituto Superior Politécnico José Antonio Echeverría. Cuidad de la Habana : s.n., 2001. Tesis.

7. **Fernández Carrasco, Oscar M., García León, Delba y Beltrán Benavides, Alfa.** *Un enfoque actual sobre la calidad del software.* 1995.

8. **Hernández Guzmán, Andrés, Fabela Gallego, Manuel de Jesús y Martínez Madrid, Miguel.** *Sistemas de Calidad y Acreditación aplicados a laboratorios de prueba.* 2001.

9. **Calderón Hernández, Manuel Alexander.** *Modelo para Pruebas se software, auditoría en entorno de prueba Microsoft.Net.* 2004.

10. *Standard for Software Configuration Management Plans.* IEEE, American National Standar Institute. 1990.

11. *Calidad de software.* **Cueva Lovelle, Juan Manuel.** Pampa, España : s.n., 1999.

12. **Guzmán Arenas, Adolfo.** *Mitos, creencias y supersticiones sobre la calidad del software y de su enseñanza.* octubre, 2003.

13. **Mendoza Rivera, Oscar Santiago.** *Investigación sobre modelos de prueba.*

14. **Mendoza Sánchez, María A.** *Metodología de desarrollo de software.* 2004.

15. *Requisitos que debe cumplir un Lims.* mayo, 2004, Thermo Electron Corporation, Vol. 8.0, p. 8.

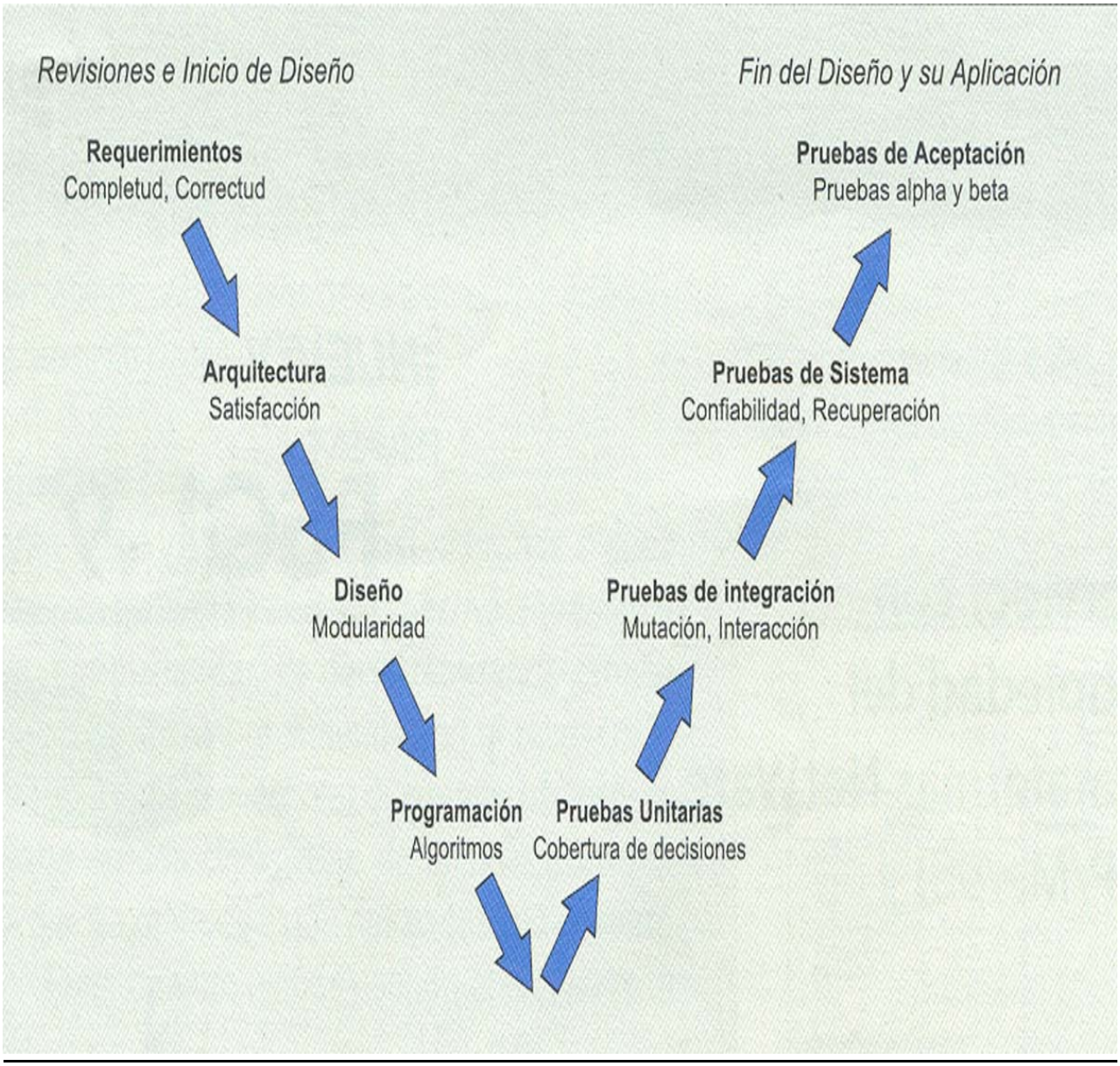
16. **García, Joaquin.** *CMM-CMMI.* 2003.

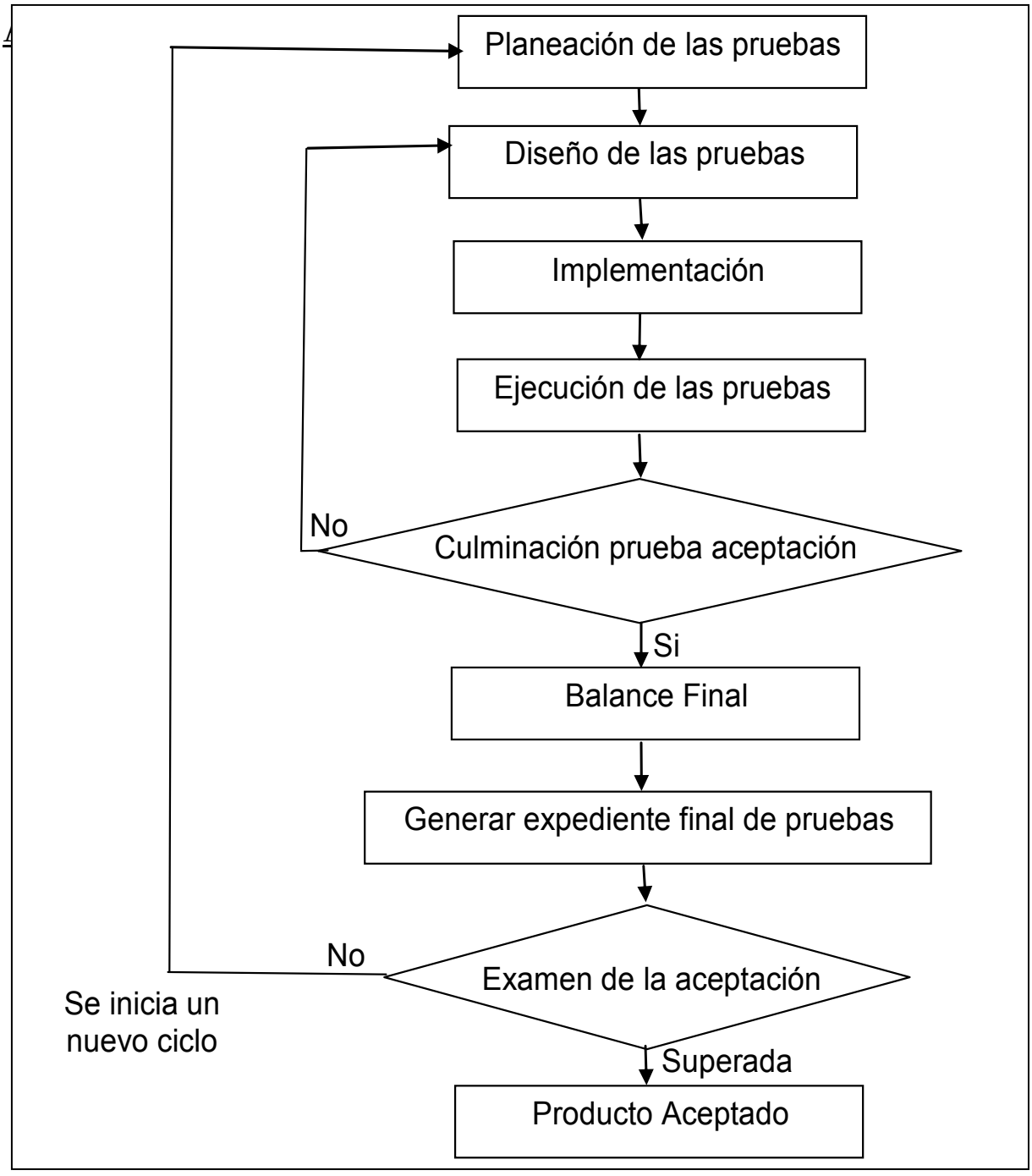
17. *Metodología DESOFT s.a.* Ciudad de la Habana : s.n., 2005.

18. *Requerimientos de la FDA cfr parte 21.* Vol. I.

Anexo

Anexo#1: Modelo V de pruebas.




Planeación de las pruebas
Diseño de las pruebas
Implementación
Ejecución de las pruebas
No
Culminación prueba aceptación
Si
Balance Final
Generar expediente final de pruebas
No
Examen de la aceptación
Se inicia un nuevo ciclo
Superada
Producto Aceptado

Glosario de Términos

Aseguramiento de la Calidad: Conjunto de acciones planificadas y sistemáticas que son necesarias para proporcionar la confianza de que un producto o servicio satisface los requisitos de calidad preestablecidos.

Base de datos: Conjunto no redundante de información almacenada en memoria organizada independientemente de su utilización y su implementación en máquinas accesibles en tiempo real y compatibles con usuarios concurrentes con necesidad de información diferente y no predicable en tiempo.

Defecto: Para un software es un proceso, una definición de datos o un paso de procesamiento incorrectos en un programa".

Fallo: "La incapacidad de un sistema o de alguno de sus componentes para realizar las funciones requeridas dentro de los requisitos de rendimiento especificados".

Control de la Calidad: Técnicas y actividades de carácter operativo utilizadas para satisfacer los requisitos de calidad.

Metodología: Un sistema de principios y normas generales de organización y estructuración teórico-práctica de actividades.

Artefacto: Los artefactos son productos de trabajo intermedios o finales que se producen y se utilizan durante un proyecto. Los artefactos se usan para capturar y difundir la información del proyecto. Un artefacto puede ser:
Un documento. Por ejemplo, el documento de Visión.
Un modelo. Como el Modelo de Casos de Uso o el Modelo de Diseño.

Requerimiento: Cualquier necesidad de un área usuaria que debe cubrirse mediante una solución de tipo informático.

Cliente: Una persona u organización, interna o externa a la organización productora que toma responsabilidad financiera por el sistema. El cliente es el último destinatario del producto desarrollado y sus artefactos.

Caso de uso: Descripción del comportamiento del sistema en término de secuencia de acciones.

Estándar: Que tiene el tamaño, la forma o cualquier otra característica que sigue al modelo. Se aplica a lo que se produce en serie. Que sigue una tendencia muy extendida. Aquello que se considera modelo.

Interfaz: Colección de operaciones que son usadas para especificar un servicio de una clase o un componente.

Interfaz gráfica: Tipo de interfaz que permite a los usuarios comunicarse con un programa mediante funcionalidades gráficas. Normalmente incluyen una combinación de gráficos, barras de menús e íconos.

Modelo: Cosa que ha de servir de objeto de imitación. Objeto, construcción u otra cosa con un diseño del que se reproduce más iguales. Esquema teórico de un sistema o de una realidad compleja que se elabora para facilitar su comprensión y estudio.

Rol: Papel, cometido o función que tiene o desempeña que interpreta un actor.