laSalle

UNIVERSITAT RAMON LLULL

Escola Tècnica Superior d'Enginyeria La Salle

Treball Final de Màster

Màster Universitari en Enginyeria Informàtica i la seva gestió

Plan de pruebas: una propuesta

Alumne
Patricia Picanyol Mercadal

Professor Ponent
Maria Antonia Mozota Coloma

ACTA DE L'EXAMEN DEL TREBALL FI DE CARRERA

Reunit el Tribunal qualificador en el dia de la data, l'alumne

D. Patricia Picanyol Mercadal

va exposar el seu Treball de Fi de Carrera, el qual va tractar sobre el tema següent:

Plan de pruebas: una propuesta

Acabada l'exposició i contestades per part de l'alumne les objeccions formulades pels Srs. membres del tribunal, aquest valorà l'esmentat Treball amb la qualificació de

Barcelona,

VOCAL DEL TRIBUNAL VOCAL DEL TRIBUNAL

PRESIDENT DEL TRIBUNAL

Abstract

La finalidad de este proyecto es proponer un *proceso de pruebas* básico y útil que se pueda implantar de forma sencilla dentro de una empresa, basado en el estándar ISO-1926 y en las recomendaciones de la IEEE.

Para ello se han diseñado y estructurado una serie de pasos básicos y necesarios que permitirán una adaptación fácil y rápida del proceso dentro de una implementación.

Para complementar este proceso se ha creado y trabajado un documento básico que contendrá la información necesaria para empezar a familiarizarse con la *calidad del software*. Y se ha creado un manual básico para validar los principales atributos de las implementaciones software.

Resumen

Este proyecto pretende elaborar un *proceso de pruebas* simple y práctico que permita implantar procesos orientados a la calidad dentro de una empresa.

Se han estudiado y comparado las metodologías más significativas que se pueden encontrar actualmente en el mercado, y se han completado con los estándares relacionados con la calidad.

Se han diseñado y descrito una serie de fases y actividades a nivel práctico que facilitarán la instauración del proceso. Éstas se han detallado, explicado y relacionado, intentado ofrecer a los usuarios un manual práctico que les permita iniciar la implantación de un *proceso de pruebas*.

Se han completado estos pasos aportando la información necesaria para la creación de un *plan de pruebas*, según las recomendaciones de la IEEE, que permite guiar todo el proceso en un proyecto real.

También se ha recopilado la información básica relacionada con las pruebas, y se ha creado una pequeña guía para a validar los atributos más significativos de la *calidad del software* según las recomendaciones de la ISO-9126 y evaluarlos a través de un conjunto de métricas.

Finalmente se ha aplicado este proceso en un proyecto real, y se ha generado un *plan de pruebas* adaptado a los requerimientos facilitados por un cliente.

Índice

1. Introducción

1.1. Marco

La intención de este proyecto es crear un *proceso de pruebas* para el Departamento de Informática, en colaboración con el Departamento de Calidad de la Universidad La Salle-Ramón Llull. Existen muchas recomendaciones teóricas de cómo realizar un *proceso de pruebas*, pero nada que indique exactamente cuáles son las actividades que se deben realizar para llevarlo a cabo.

1.2. Estado del arte

Existen varias metodologías que intentan mejorar la *calidad del software*. Hemos seleccionado cuatro de las más utilizadas:

Métrica 3: es una metodología para planificar, desarrollar y mantener sistemas de información. Es la promovida por el Ministerio de Administraciones Públicas del Gobierno de España para trabajar con proyectos software relacionados con las administraciones públicas. Se basa en las normas ISO -12207 y la ISO-15504.

T-Maps: es una metodología para gestión del *proceso de pruebas*, propiedad de la empresa Sogeti. Se basa en cuatro elementos esenciales: Business-driven Test Management(BDTM), Proceso de Test Estructurado, Kit de Herramientas, y Adaptable.

Six Sigma: es una metodología para la mejora de procesos creada en Motorola en el año 1982 por el ingeniero Bill Smith, pensada como una estrategia de negocios y mejora de la calidad. Posteriormente fue mejorada y popularizada por General Electric. Se basa en el uso de herramientas estadísticas y en el estudio de los procesos.

RUP (Rational Unified Process): es un producto comercial propiedad de la compañía de IBM. El Proceso Unificado es un proceso de software que puede ser utilizado para una gran cantidad de sistemas de software, utiliza el lenguaje de modelado UML y un ciclo de vida en espiral. Es una metodología pensada para implementaciones orientadas a objetos.

A continuación se realiza una comparativa entre cada una de las metodologías descritas, basándonos en: cómo realizan la toma de requerimientos, cuáles son las fases que definen, qué pruebas recomiendan, documentación que utilizan y herramientas relacionadas.

Empezaremos mirando como recomiendan tomar los requerimientos cada una de las metodologías:

	Toma de requerimientos
Métrica 3	• Identificación de Subsistemas de Análisis (ver ASI 3) • Análisis de Casos de Uso (ver ASI 4) • Análisis de Clases (ver ASI 5) • Elaboración del Modelo de Datos (ver ASI 6) • Elaboración del Modelo de Procesos (ver ASI 7) • Definición de Interfaces de Usuario (ver ASI 8).
T-map	No define cómo se deben tomar los requerimientos
Six Sigma	No define cómo se deben tomar los requerimientos
Rup	• Casos de uso • Diagramas de actividad • Diagramas de despliegue • Vision • Etc.

Hay que destacar que tanto T-map como Six Sigma, al no ser metodologías orientas a la definición del software, no definen cómo deben tomarse los requerimientos del mismo (aunque T-map si que habla de ciertas aplicaciones para gestionarlos.)

A continuación se comparan las fases del ciclo de vida, definidas por cada una de las metodologías:

	Fases
Métrica 3	Define un proceso iterativo para la implementación del sistema, con las siguientes fases: • Viabilidad • Análisis • Diseño • Construcción • Implantación • Aceptación • Mantenimiento. La calidad se mide en la fase de aceptación.
T-map	Define un proceso para validar la calidad que se adapta al ciclo de vida del software, con las siguientes fases: • Planificación • Preparación • Especificación • Ejecución • Finalización En paralelo a todas ellas, define dos fases: • Control • Preparación y mantenimiento de la infraestructura.
Six Sigma	Define un proceso iterativo para validar la calidad, con las siguientes fases: • Definir • Medir • Analizar • Mejorar • Controlar
Rup	Define un ciclo de vida en espiral, con las siguientes fases: • Modelado de negocio • Requisitos • Análisis y Diseño • Implementación • Pruebas

- Despliegue

En paralelo, también define:

- Gestión del cambio y configuraciones
- Gestión del proyecto
- Entorno

La calidad se mide en la fase de pruebas.

RUP y Métrica 3, al estar orientadas al desarrollo del software, no definen unas fases concretas para validar las pruebas, pero si las incluyen dentro de su ciclo de vida. Six Sigma y T-map al ser orientadas a la calidad sí definen las fases concretas.

Las pruebas recomendadas por cada una de las metodologías son:

	Pruebas
Métrica 3	• Pruebas unitarias • Pruebas de integración • Pruebas del sistema • Pruebas de implantación • Pruebas de aceptación • Pruebas de regresión
T-map	• Pruebas de aceptación • Pruebas de sistema • Pruebas de desarrollo.
Six Sigma	Al no ser una metodología pensada solamente para el software, no define la tipología de pruebas.
Rup	• Pruebas unitarias • Pruebas de integración • Pruebas del sistema y de implantación • Pruebas de aceptación

Métrica 3 y RUP recomiendan pruebas muy similares. T-map coincide con ellas en aceptación y sistema. Six-Sigma no define pruebas en concreto ya que está pensada para cualquier proceso tecnológico y no sólo procesos de implementación software.

A continuación se describen algunas herramientas relacionadas con cada una de las metodologías. Éstas pueden estar recomendadas por la empresa propietaria, o pueden haber sido creadas para la metodología en concreto:

	Herramientas
Métrica 3	Se adapta a las herramientas de la organización.
T-map	Recomienda herramientas relacionadas con la calidad de las siguientes empresas: • Microsoft • IBM • Oracle • HP • Borland • Dassault Systemes
Six Sigma	Algunas herramientas relacionadas: • Arena (software) • ARIS Six Sigma • @RISK for Six Sigma • iGrafx Process for Six Sigma • Quik Sigma Professional by Promontory Management Group • SigmaXL • SigmaFlow(BPA)
Rup	Las herramientas creadas por IBM para gestionar la calidad son: • Rational Quality Manager: Solución de gestión de calidad de ciclo vital para la planificación de pruebas, el control del flujo de trabajo y la generación de informes de mediciones. • Rational Functional Tester: Automatización de pruebas funcionales para garantizar la distribución de software, basada en la calidad. • Rational Performance Tester: Herramienta de análisis, ejecución y creación de pruebas de rendimiento, para validar la escalabilidad y fiabilidad de las aplicaciones. • Rational PurifyPlus: Completo conjunto de herramientas de análisis de tiempo de ejecución, diseñadas para mejorar la fiabilidad y el rendimiento de las aplicaciones. • Rational Service Tester for SOA Quality: Automatización de pruebas funcionales, para validar las aplicaciones SOA y los servicios web.

Finalmente se compara la documentación que generan cada una de las metodologías. Esta documentación permite que todas las partes implicadas en el proyecto puedan conocer los resultados de las actividades realizadas en cada una de las fases:

	Documentación

Métrica 3	• Análisis coste/beneficio
	• Casos de uso
	• Diagrama de clases
	• Diagrama de componentes
	• Diagrama de descomposición
	• Diagrama de despliegue
	• Diagrama de estructura
	• Diagrama de flujo de datos (dfd)
	• Diagrama de interacción
	  o Diagrama de secuencia
	  o Diagrama de colaboración
	• Diagrama de paquetes
	• Diagrama de transición de estados
	• Modelado de procesos de la organización sadt (structured analysis and design technique)
	• Modelo entidad/relación extendido
	• Normalización
	• Optimizaciónreglas de obtención del modelo físico a partir del lógico
	• Reglas de transformación técnicas matriciales

T-map	**Business-driven Test Management** consta de los siguientes pasos: • Formular la misión y recolectar objetivos de las pruebas. • Determinar las categorías de riesgo. • Determinar cobertura alta/baja. • Realizar una estimación global para las pruebas y la planificación creadas. • Asignar técnicas de pruebas. • Proporcionar al cliente información sobre el proceso de test. **Master Test Plan.** Se compone de 2 fases: planificación de la totalidad del proceso de test y control de la totalidad del proceso de test. Su principal objetivo es detectar los defectos más importantes con el menor coste y el menor tiempo posibles.
Six Sigma	• Árbol Critico para la Calidad (CPC) • Diagrama del proceso • Histograma • Diagrama de Pareto • Hoja de Análisis del Proceso • Diagrama de Causa efecto • Diagrama de Dispersión • Diagrama de Afinidad • Diagrama de Movimiento

	• Diagrama de Control
Rup	• Diagrama de clases • Diagrama de componentes • Diagrama de objetos • Diagrama de estructura compuesta Diagrama de despliegue • Diagrama de actividades • Diagrama de casos de uso • Diagrama de estados • Diagrama de secuencia • Diagrama de comunicación • Diagrama global de interacciones • Vision • Supplementary Specifications • Glossary artefacts

1.3. Descripción del problema

Existe mucha documentación que explica la teoría de cómo validar el software y de cómo implementar un *proceso de pruebas*, pero no se explica exactamente cuáles son las actividades que se deben realizar y cómo deben realizarse. Además, cada Empresa u Organización tiene sus propios procedimientos, y es recomendable adaptar el *proceso de pruebas* a éstos.

La intención de este proyecto es encontrar un *proceso de pruebas* que se adapte a las necesidades de las implementaciones software realizadas en el marco del Departamento de Informática.

1.4. Solución propuesta

Para realizar este proyecto, se ha llevado a cabo un estudio de las metodologías y procesos orientados a la calidad que hay actualmente en el mercado. Se ha buscado la información relacionada con las mismas y se ha contrastado con la metodología usada dentro del Departamento de Informática.

También se han consultado los estándares ISO y las documentaciones recomendadas por la IEEE. Estos documentos proponen los conocimientos básicos y las herramientas necesarias para poder diseñar un *proceso de pruebas* adaptado a las necesidades de cada proyecto.

Partiendo de esta información se ha diseñado un proceso básico que permite mejorar la metodología de trabajo actual de Departamento de Informática, orientándola a la calidad de los procesos.

En este proyecto se presenta una solución básica para poder empezar esta implantación. Se han descrito los pasos básicos que se deben seguir para realizar un *proceso de pruebas* adaptado a la necesidades de la empresa. También se ha definido un documento básico para poder empezar a trabajar con procesos orientados a la calidad, relacionándolo con los pasos básicos del proceso de *proceso de pruebas*. Y finalmente se han escogido y detallado los atributos más relevantes para poder validar futuras implementaciones.

2. Proceso de pruebas

El *proceso de pruebas* está formado por una serie de actividades estáticas y dinámicas que permiten planificar, preparar y evaluar productos software. Comprueba que se satisfacen unos requerimientos específicos, y demuestra que son adecuados para un propósito, detectando también sus defectos.

Durante el *proceso de pruebas* se validará el software y todos los trabajos relacionados con éste. Por ejemplo: los requerimientos, el diseño, el material necesario para ejecutarlo y el entrenamiento del usuario.

Requerimientos: son la condición o capacidad que necesita el usuario para solventar un problema o lograr un objetivo. Éste debe ser alcanzado por un sistema o un componente de un sistema, para satisfacer un contrato, un estándar, una especificación u otro documento impuesto formalmente.

Para garantizar la construcción del software de manera correcta y con un coste reducido, es recomendable relacionar este proceso con el ciclo de vida de desarrollo.

Para reducir los errores introducidos en el código y conseguir minimizarlos al final del ciclo de vida, se deben diseñar las pruebas al inicio del mismo.

Los motivos principales para aplicar el *proceso de pruebas* en un proyecto son los siguientes:

1. Las pruebas permiten determinar si un producto software satisface unos requerimientos específicos: Las pruebas permiten validar que el diseño se ajusta a los requerimientos y que el código satisface el diseño.
2. Las pruebas demuestran que el producto se ajusta a un propósito: Los requerimientos pueden no ser correctos o estar incompletos. Por eso es recomendable evaluar si el producto cumple las necesidades del usuario o el propósito de forma razonable.
3. Las pruebas ayudan a detectar defectos: El propósito más común del *proceso de pruebas* es hallar fallos en el producto. Esto permite mejorar el producto en el momento de ponerlo en uso y prevenir futuros fallos. Este proceso ayuda a mejorar la *calidad del software*, el ciclo de vida y reducir defectos en futuros proyectos

2.1. Principios del proceso de pruebas

Antes de empezar un *proceso de pruebas* hay que tener en cuenta los siguientes principios:

1. Realizar pruebas **puede mostrar que existen defectos pero no puede garantizar que no existen**. Este proceso reduce la probabilidad de encontrar fallos.
2. **Es imposible probar todas las posibles combinaciones del software** excepto en los casos triviales. Por eso mismo es recomendable analizar los riesgos y las prioridades enfocando los esfuerzos en estos dos puntos.
3. **Las actividades del *plan de pruebas* deben empezar lo antes posible** en el ciclo de vida de un producto o sistema, y deben centrarse en los objetivos definidos.
4. Normalmente **un número reducido de módulos contienen la mayoría de los defectos** encontrados en la fase de pruebas.
5. Si las mismas pruebas son ejecutadas una y otra vez en el tiempo sobre el mismo conjunto de pruebas **se deben analizar y revisar las pruebas continuamente** o no se encontrarán nuevos defectos. Para evitarlo es recomendable crear nuevas pruebas, y así ejercitar diferentes partes del programa y hallar nuevos defectos.
6. **Las pruebas se realizan de diferente modo según el contexto**.
7. **Encontrar y corregir defectos es innecesario si el sistema no cumple los requerimientos** o no es un producto utilizable.

Para validar correctamente la *calidad del software* es necesario saber qué significa la calidad para un software en concreto y cómo se va a comprobar que se cumple.

2.2. Nivel de las pruebas

Antes de empezar a explicar los niveles de pruebas, es importante tener claras cuales son las diferencias entre validar y verificar un producto. Para **validar** un sistema se debe comprobar que es correcto para un uso en concreto. Para ello, se estudia si los resultados obtenidos de las diferentes funciones del producto son las adecuadas.

Para **verificar** un sistema se debe comprobar que es correcto en función de unas especificaciones. Para ello se mira si el producto cumple los requerimientos y definiciones de las fases anteriores. Por ejemplo, una función puede ser válida porque realiza una suma correctamente, pero no se podrá verificar que es correcta si no suma los valores indicados en las especificaciones.

Existen varios niveles de pruebas que permiten validar diferentes características del sistema:

- Pruebas de componentes: verifica cada componente de forma individual. Busca los defectos y prueba las funcionalidades del software de aquellas partes que se pueden ejecutar de forma independiente. Se pueden usar *Stubs* o *Drivers* para simular el software que falta. Un *Stub* es un componente ficticio que es llamado por el componente que se quiere probar. Un *Driver* es un componente ficticio que llaman al componente que se quiere probar.
- Pruebas de integración: verifica los posibles defectos entre componentes y cómo interactúan al ser integrados o unificados.
- Pruebas de sistema: permite validar el software bajo la perspectiva que tendrá el futuro usuario y verifica que se cumplan los requerimientos. También permite comprobar que los requerimientos están bien especificados.
- Pruebas de aceptación: valida las necesidades de los usuarios, los requerimientos y el proceso de negocio para determinar si el sistema satisface el criterio de aceptación del usuario.
-

2.3. Tipos de pruebas

Existen varios tipos de pruebas clasificados según los siguientes conceptos:

- Funcionales: estas pruebas se usan para validar el correcto funcionamiento de un sistema. Para asegurar la funcionalidad de un programa se deben garantizar las siguientes características: idoneidad, precisión, seguridad e interoperabilidad.
- No funcionales: validan los atributos de un componente o sistema que no están relacionados con las funcionalidades, por ejemplo, eficiencia, usabilidad o portabilidad.
- Estructurales o de caja blanca: son pruebas basadas en el análisis de la estructura interna de un componente o sistema. Para diseñar estas pruebas es necesario conocer el funcionamiento interno del sistema.
- Regresión: consiste en probar un programa que ha sido probado y modificado anteriormente, y asegurar que no se hayan introducido defectos en otras áreas del programa.

2.4. Técnicas de las pruebas

Existen dos grandes familias de técnicas: estáticas y dinámicas.

Estas suelen complementarse ya que se usan para validar diferentes tipos de fallos.

Técnicas estáticas

Las **técnicas estáticas** permiten evaluar un componente o sistema, tanto a nivel de implementación como de especificación, sin necesidad de ejecutarlo.

Las técnicas estáticas permiten disponer de información sobre la calidad de un sistema desde el inicio, permitiendo reducir los costes durante la implementación. También pueden aportar información para mejorar y evitar errores en un futuro.

Estas son algunas de las técnicas más utilizadas:

- ***Informal reviews***: son técnicas que no se basan en procedimientos formales (documentados). Son revisiones que se realizan sin seguir ningún procedimiento.
- ***Walkthorughts***: esta técnica consiste en realizar la presentación paso a paso de un documento. Se pretende recoger información para mejorar el documento y establecer una comprensión común de su contenido con terceras personas.
- ***Tecnical reviews***: es una técnica que consiste en realizar un análisis entre un grupo de personas de todas las actividades que se quieren realizar en el *proceso de pruebas*. Se decidirá entre todas cuales serán las técnicas que se usarán y actividades se deberán realizar.
- ***Inspection:*** es un tipo de revisión que se basa en revisar de forma visual de los documentos para detectar defectos.
  Por ejemplo: Violaciones de las normas de desarrollo y la no conformidad a la documentación de nivel superior. Siempre se basa en un procedimiento documentado.
- ***Static Analysis***: es técnica se encarga validar los artefactos del software. Por ejemplo los requerimientos o el código sin tener que ejecutarlos.
- ***Control flow****:* es una técnica que realizar una representación abstracta de todas las posibles secuencias de los eventos (caminos) en la ejecución de un componente o sistema. Esta representación permitirá validar que todas las ejecuciones del sistema siguen el camino especificado.
- ***Data flow****:* es una técnica que realiza una representación abstracta de las secuencias y de los posibles cambios del estado de los objetos del sistema, dónde el estado de un objeto puede ser: creación, destrucción o uso. Esta representación se usará para comparar más adelante si los cambios de estado de los objetos que realiza el sistema son los deseados.

Técnicas dinámicas

Las **técnicas dinámicas** implican la ejecución de un componente o sistema. Utilizan una serie de valores de entrada y examinan los valores de salida.

Permiten así mismo detectar defectos y determinar los atributos de *calidad del software*.

Basadas en estructura (*White box*)

Existen varias técnicas basadas en estructura: *Statement coverage, Decision coverage, Multiple condition, etc.*

- ***Statement coverage:*** esta técnica de caja blanca que valora el porcentaje de sentencias ejecutables que han sido ejercitadas por un juego de pruebas. Esta información permite saber si se han probado todas las sentencias que se querían verificar.
- ***Decision coverage:*** es una técnica estructural que valora el porcentaje de los resultados relacionados con las condiciones del código que han sido ejercitados por un juego de pruebas (*test suite*). Una cobertura de decisión del 100% implica una cobertura de sentencia del 100%.
- ***Multiple condition:*** es una técnica de caja blanca en la que los casos de prueba están pensados para ejecutar combinaciones que validen el funcionamiento de una única condición de diferentes formas.

Basadas en la experiencia (*Experience based*)

Existen varias técnicas basadas en la experiencia: *Error guessing, Exploratory testing, etc.*

- ***Error guessing:*** Técnica de diseño de pruebas donde la experiencia del probador es usada para anticiparse a los defectos que pueden presentarse en el componente o sistema sometido a prueba, como resultado de un error cometido, y para diseñar pruebas específicas
- ***Exploratory testing****:* Técnica de diseño de pruebas donde el probador controla activamente el diseño de las pruebas y cómo éstas se realizan. Utiliza la información obtenida durante el *proceso de pruebas* para diseñar nuevas y mejores futuras pruebas.

Basadas en las especificaciones (*Black box*)

Existen varias técnicas basadas en las especificaciones: *Use case testing, State transtion, Decision tables*, *Equivalence partitioning*, *Boundary value analysis*, *etc.*

- ***Use case testing****:* una técnica de caja negra en la que se ejecutan todos los casos de casos de uso definidos para el usuarios y se verifica su correcto funcionamiento.
- ***State transtion****:* es una técnica de caja negra en la cual se ejecutan casos de prueba que se validan todas las transiciones de los objetos del sistema y sus respuestas. Tanto para estados validos como para estados invalidados.

- ***Decision tables***: es una técnica de caja negra en la que los casos de prueba están diseñara para ejecutar combinaciones de datos de entrada y/o estimulas que se muestran en una tabla en una tabla de decisiones. Esta tabla contiene los resultados esperados y permite compararlos con los resultados obtenidos.
- ***Equivalence partitioning****:* es una técnica de caja negra en la que los casos de prueba están diseñados para ejecutar particiones equivalentes representativas. En principio, los casos de prueba están diseñados para cubrir cada partición menos una vez.
  Una partición equivalente es una porción de una entrada o una salida del dominio del sistema para la que se supone que el comportamiento del componente o sistema será siempre el mismo.
- ***Boundary value analysis****:* es una técnica de caja negra que valida todos los valores frontera del sistema. Se crean casos de uso que ejecutan las funciones con estos valores. Un valor frontera es un valor de entrada o el valor de salida que se encuentra al borde de una partición de equivalencia o en la distancia más pequeña a cada lado de un borde, por ejemplo, el mínimo y el máximo valor de todos los posibles valores que puede tomar un entero.

3. Fases fundamentales del proceso de pruebas

El *proceso de pruebas* está compuesto por una serie de fases que permiten verificar los requerimientos del sistema y validar la calidad del producto.

En el siguiente diagrama de actividad se muestran las fases básicas y cuáles son los resultados de cada una de ellas. Dichas fases deben adaptarse a las necesidades del software y a la metodología de trabajo usada para la implementación del sistema.

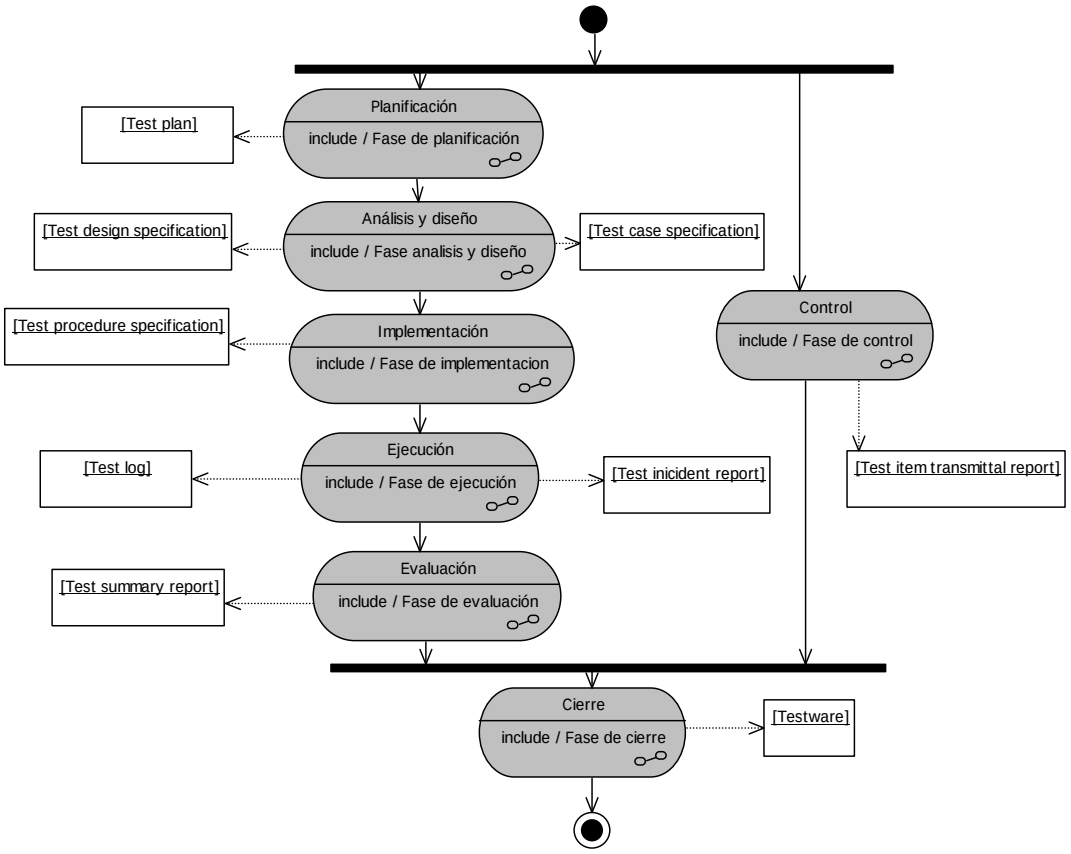


ILUSTRACIÓN 1 DIAGRAMA DEL PROCESO DE PRUEBAS

Las fases fundamentales del *proceso de pruebas* son:

- **Planificación**: durante esta fase se definen y documentan las metas y objetivos de las partes implicadas en el proyecto.
- **Análisis y diseño:** durante esta fase los objetivos generales de las pruebas se transforman en condiciones tangibles y en diseños.
- **Implementación:** en esta fase se crean los conjuntos de pruebas y el entorno de las mismas, a partir de las condiciones obtenidas en la fase de análisis y diseño.
- **Ejecución:** se aplican todas las pruebas creadas en la fase anterior y se registran los resultados.

- **Evaluación:** se analizan los resultados registrados en el proceso de ejecución.
- **Cierre:** se crea un entregable con toda la información del *proceso de pruebas*, que servirá como histórico para consolidar la experiencia.
- **Control:** permite saber si el proceso está funcionando correctamente y realizar un seguimiento del mismo.

El resultado de cada una de estas fases será un documento o entregable que se usará en la siguiente fase o en futuros procesos de pruebas. Los artefactos obtenidos como resultado son:

- ***Test plan:*** este documento es el resultado de la fase de planificación. Describe el ámbito, el enfoque, los recursos y el programa de las actividades previstas.
- ***Test design specification:*** este documento es el resultado de la fase de diseño y análisis. Especifica las condiciones de las pruebas para un ítem.
- ***Test case specification:*** es el resultado de la fase de diseño y especificación. En este documento se pueden consultar los valores de entrada, los resultados esperados y demás información necesaria para ejecutar una prueba.
- ***Test procedure specification***: este documento es el resultado de la fase de implementación. Indica la secuencia de acciones para la ejecución de una prueba.
- ***Test ítem transmittal report***: describe la estructura de documentos, datos e información del *proceso de pruebas*.
- ***Test log:*** es el registro cronológico de los detalles relevantes ocurridos durante la ejecución de las pruebas.
- ***Test incident report:*** es el documento que recopila cualquier evento ocurrido durante la ejecución y que debe ser investigado.
- ***Test summary report***: es el documento final donde se recopilan todos los resultados y conclusiones de las pruebas.
- ***Testware***: es la recopilación de todos los artefactos producidos durante el *proceso de pruebas*.

3.1. Fase: Planificación

La fase de planificación del *proceso de pruebas* permite definir y documentar todas las metas y objetivos de las partes implicadas en el proyecto.

Durante esta fase también deben quedar claros cuáles son los riesgos que se intentan minimizar con este trabajo. Basándose en esta información se podrá definir el enfoque del proyecto y realizar un plan para las pruebas. Es importante que los *stakeholders* tengan estos puntos claros antes de empezar la planificación. Para ello se debe documentar, especificar y validar esta información con todas las personas implicadas en el proyecto.

Para conducir la planificación, la organización debe definir unas políticas o estrategias que facilitarán las actividades del proceso.

Durante la planificación se debe crear un plan de actividades y un plan de control, y definir como se medirá el progreso. Esto permitirá especificar las actividades que se realizarán y en qué puntos se alcanzarán los objetivos del proyecto que pueden implicar cambios en el plan. El control y la monitorización son puntos muy importantes en esta fase.

En los siguientes puntos se mostrará qué actividades componen la planificación, qué se debe realizar en cada una de ellas y cómo se relacionan con la información que debe documentarse en el *plan de pruebas*.

3.1.1. Actividades

Para realizar de forma correcta el proceso de planificación es recomendable seguir los siguientes pasos:

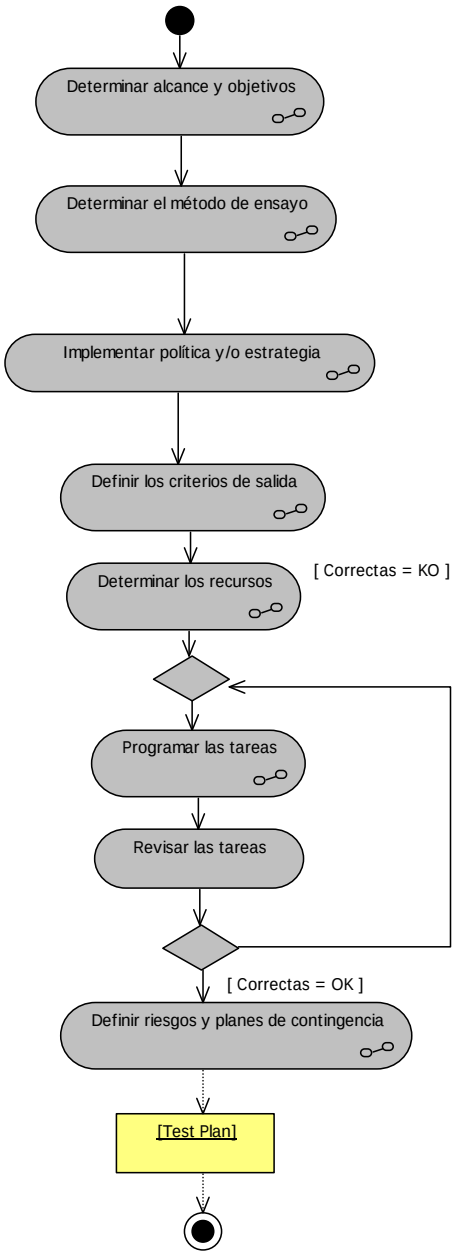


ILUSTRACIÓN 2 DIAGRAMA DE LA FASE DE PLANIFICACIÓN

Hay que tener en cuenta que cada uno de estos pasos se adapta a las necesidades del producto o de la empresa, dado que cierta información o forma de trabajo puede venir predefinida por los requerimientos del proyecto. Como por ejemplo, el formato de los entregables o los programas que monitorizarán el proceso.

1. Determinar alcance y objetivos

Estos son las actividades que se deben seguir para determinar el alcance y los objetivos:

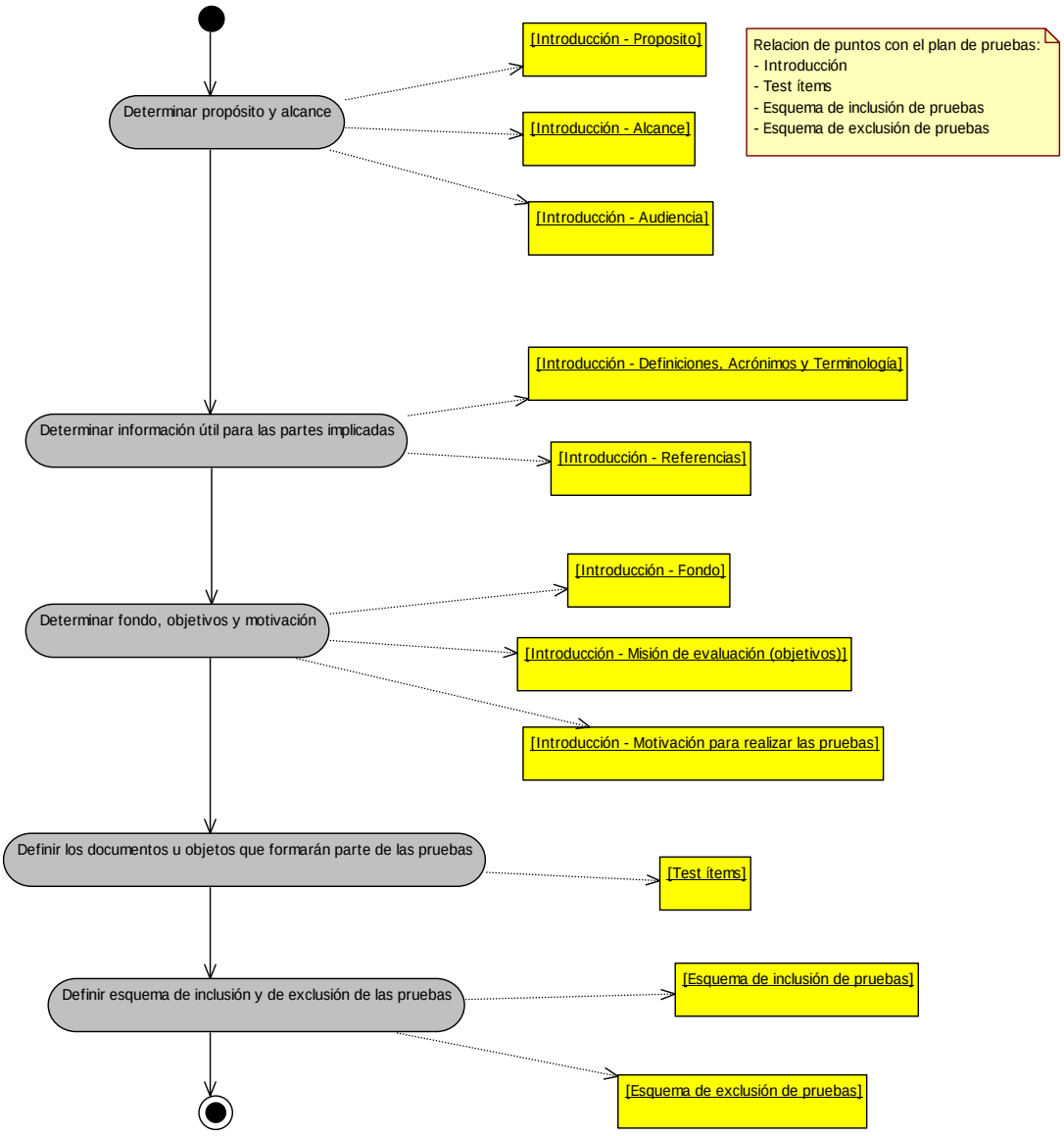


ILUSTRACIÓN 3DIAGRAMA DE LA ACTIVIDAD DETERMINAR ALCANCE Y OBJETIVOS

Para poder implementar y detectar las pruebas necesarias para un producto hay que conocer exactamente cuál es la intención del *proceso de pruebas*. Por eso mismo es importante **definir cuál es el alcance y cuál es el propósito** de las mismas. Para poder identificarlos se puede analizar la siguiente información:

- Los riesgos del negocio: sirven para priorizar las pruebas que se van realizar.
- Los riesgos del producto: permiten saber cuáles son las partes más importantes a probar.
- Los riegos del proyecto: determinan el *proceso de pruebas* y los riesgos del mismo.

- Los riesgos técnicos: permiten conocer mejor la tipología de pruebas que hay que realizar.
- El software, componentes o sistemas que están dentro del alcance de las pruebas: es importante saber qué se va a probar y qué no, ya que el coste de validar el proyecto entero generalmente es demasiado grande.

Una vez se conoce la intención de las pruebas, se debe documentar la **información importante para las partes implicadas** en el proyecto. Como por ejemplo definiciones, acrónimos y terminologías, que pueden resultar útiles para aquellas personas no implicadas directamente en el proyecto. También es interesante crear una referencia a otros documentos que pueden ser útiles durante el proceso, como el ERS o el plan de riesgos.

Una vez ha quedado bien definida toda la información de los puntos anteriores hay que **determinar los objetivos del proceso (misión de evaluación), el fondo y la motivación de las pruebas**. Estos deben estar relacionados con las siguientes acciones:

- Mostrar defectos: si la intención de nuestro de *proceso de pruebas* es detectar errores en el código o en la ejecución del software.
- Ajustar los requerimientos: si la intención es garantizar que se cumplen los requerimientos del proyecto.
- Ajustar el propósito: si la intención es verificar que el software se ajusta a las necesidades del usuario.
- Medir atributos: si se busca asegurar que el producto cumple una serie de condiciones.
- Medir calidad: si se quiere asegurar la calidad del producto que se está entregando.

Para acabar de delimitar el alcance y la misión de evaluación del *plan de pruebas* se deberán **definir todos los documentos u objetos que formarán parte de las pruebas** como por ejemplo:

- Partes del sistema que se validarán
- Documentos resultantes
- Dispositivos de almacenaje de resultados
- Entregables

Finalmente, según los objetivos que se quieran alcanzar se decidirá qué tipos de pruebas (funcionales, no funcionales, estructurales, etc.) se deben realizar, y cuáles son las técnicas que pueden interesar. Para ello se **creará un esquema de inclusión de pruebas**, donde se indicarán las pruebas incluidas en el plan y las razones de su inclusión. También es importante **indicar el esquema de exclusión de pruebas** y la razón por la cual han sido excluidas.

2. Determinar el método de ensayo (test approach)

Método de ensayo (*test approach*): es la implementación de la estrategia de prueba para un proyecto en concreto. Incluye las decisiones basadas en el objetivo de las pruebas del proyecto, la evaluación de las técnicas de diseño, los criterios de éxito y los tipos de prueba que se ejecutarán.

Estas son las actividades que se deben seguir para determinar el método de ensayo:

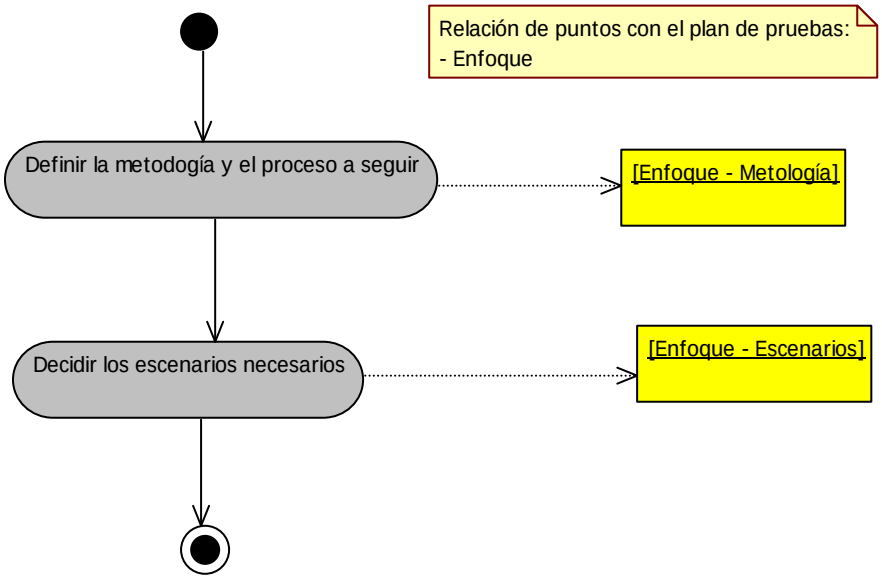


ILUSTRACIÓN 4 DIAGRAMA DE LA ACTIVIDAD DETERMINAR EL MÉTODO DE ENSAYO

Se definirá la metodología para realizar las pruebas dependiendo de la información obtenida en el punto anterior. Se decidirá:

- Cómo se realizarán las pruebas: manuales, automáticas, etc.
- Qué técnicas se usarán: basadas en experiencias, estructura, etc.
- Qué necesitan las pruebas: programas para ejecutar las pruebas, equipo, entorno, etc.
- Extensión de las pruebas (cobertura o *coverage*).

También se decidirán **cuáles son los escenarios necesarios** para realizar las pruebas en función de:

- Los requerimientos del sistema
- La metodología definida
- Los esquemas de exclusión e inclusión de las pruebas

Cobertura de prueba (*test coverage*): es el grado, expresado como porcentajes, en que un ítem de cobertura específico ha sido usado por un conjunto de pruebas.

Se analizará la información relacionada con la estrategia de las pruebas:

- **Quién necesita involucrarse**: qué personas han de participar en el proceso.
- **Cuándo necesita involucrarse**: en qué momento del proceso se realizará cada actividad y qué relación tendrá el *proceso de pruebas* con el proceso de desarrollo del sistema.
- **Qué se producirá como parte de las pruebas**: cuáles serán los entregables del proceso y cómo podrán las partes implicadas conocer los resultados de las pruebas y su progreso.

3. Implementar política y/o estrategia

Estas son las actividades que se deben seguir para implementar la políticas y la estrategia del proceso de pruebas:

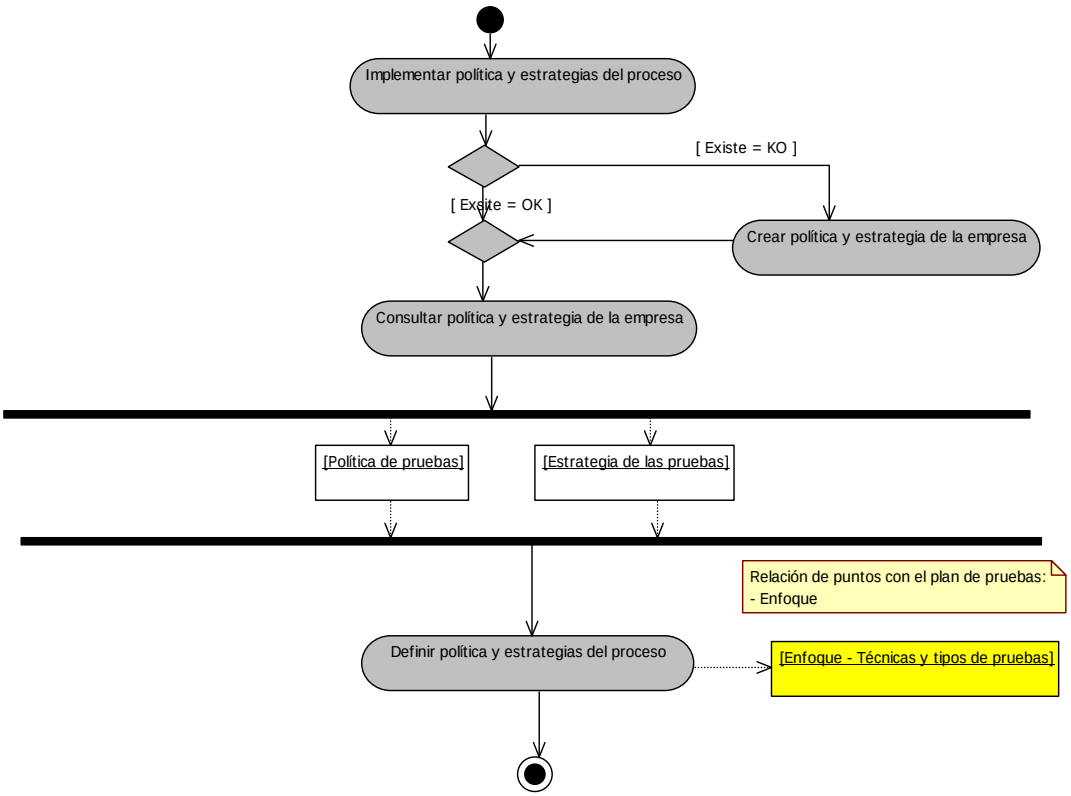


ILUSTRACIÓN 5 DIAGRAMA DE LA ACTIVIDAD IMPLEMENTAR POLÍTICA Y/O ESTRATEGIA

En este paso se revisa toda la información relativa a niveles de pruebas, principios, enfoque y objetivos necesarios para realizar nuestro proceso.
Es recomendable que las empresas generen una serie de documentación que marcará las directrices del proceso. También se deberá facilitar la reutilización de las pruebas

y guiar al equipo durante la planificación. Es recomendable que las empresas definan los siguientes documentos:

- **Política de prueba (*test policy*)**: Documento de alto nivel que describe los principios, el enfoque y los objetivos de la organización referidos al *proceso de pruebas*.
- **Estrategia de prueba (*test strategy*):** Descripción de alto nivel de los niveles de pruebas que se deberán realizar y las pruebas dentro de estos niveles para una organización o programa (uno o más proyectos).

Hay que tener en cuenta que **si ya existe** una política de pruebas o una estrategia, se debe asegurar de que lo planeado se ajusta a ambas especificaciones, y **si no existe** se debe llegar a un acuerdo con los agentes involucrados, documentarlo y justificarlo.

Con toda esta información se definirá la política y la estrategia de para este proceso en concreto. Para ello se documentará cada una de las técnicas que se realizarán en el proceso. La información necesaria para cada técnica puede ser:

- Nombre de la técnica
- Objetivos de la técnica
- Explicación de la técnica
- Estrategia
- Herramientas requeridas
- Consideraciones especiales
- Criterios de salida para esa técnica

4. Definir los criterios de salida

Estas son las actividades que se deben seguir para definir los criterios de salida:

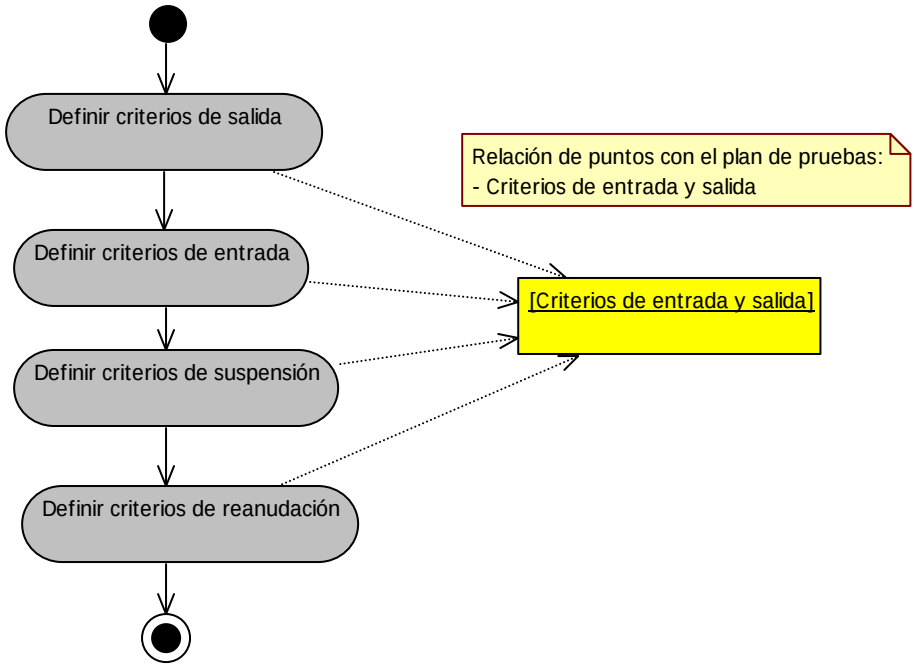


ILUSTRACIÓN 6 DIAGRAMA DE LA ACTIVIDAD DEFINIR LOS CRITERIOS DE SALIDA

Criterio de salida (*exit criteria*): es el conjunto de condiciones específicas y genéricas, acordado con las partes interesadas, para permitir que un proceso se termine de forma oficial. El propósito del criterio de salida es prevenir que una tarea sea considerada completada cuando aún hay partes pendientes que no han sido finalizadas. Este criterio es usado por el *proceso de pruebas* para informar en contra y para planificar cuándo detener la realización de pruebas.

Es muy importante decidir cuál será o serán los criterios de salida. Un posible criterio de salida podría ser: cuando se han realizado correctamente todas las pruebas.
Hay que tener en cuenta que no sólo son necesarios los criterios de salida. Durante esta fase también se deben documentar los criterios de entrada, los criterios de suspensión y los criterios de reanudación.
Criterios de entrada: son las condiciones necesarias para poder iniciar una tarea del *plan de pruebas*.
Criterios de suspensión: son las condiciones por las cuales se detendrá o cancelará una tarea del *plan de pruebas*.
Criterios de reanudación: son las condiciones por las cuales se podrá volver a iniciar una tarea del *plan de pruebas* después de ser suspendida.

5. Determinar los recursos para las pruebas (test resources).

Estas son las actividades que se deben realizar para determinar los recursos de las pruebas:

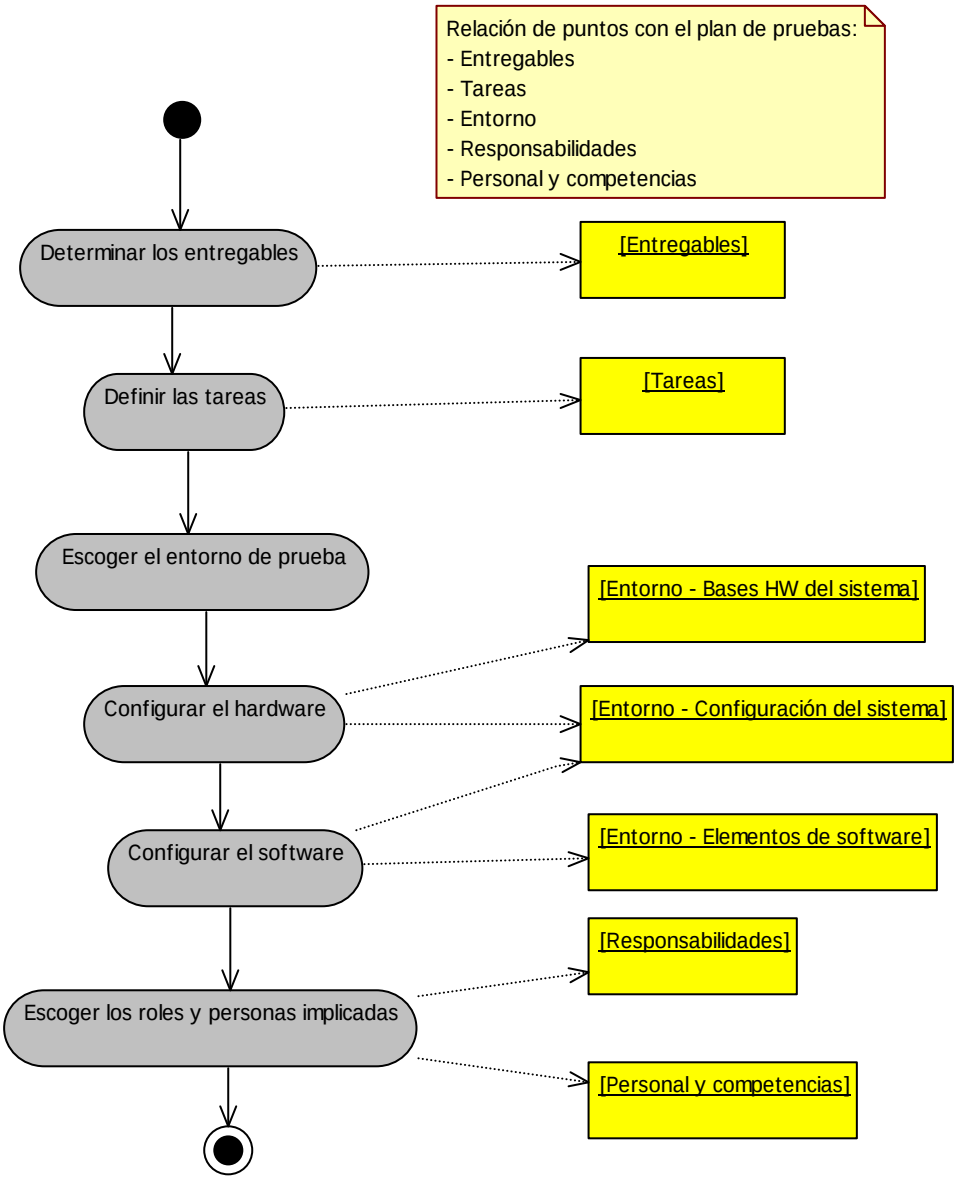


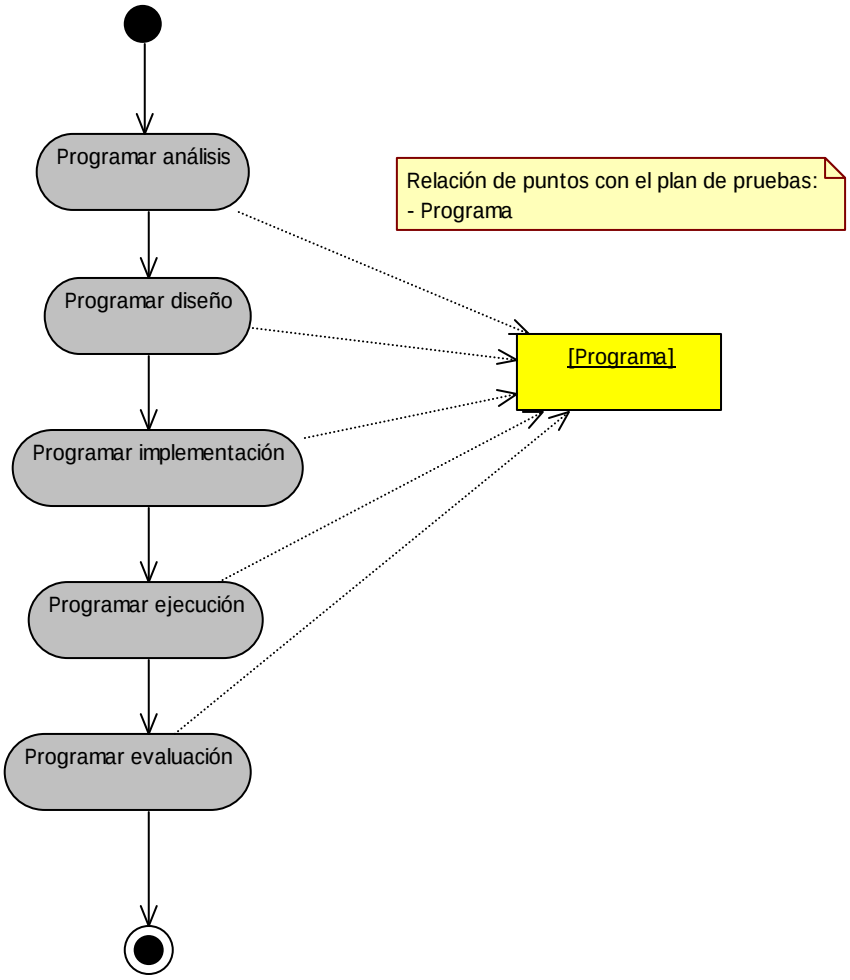
ILUSTRACIÓN 7 DIAGRAMA DE LA ACTIVIDAD DETERMINAR LOS RECURSOS

Es importante decidir exactamente qué personas formarán parte del proceso, en qué momento intervendrán y cuáles serán sus funciones. También se debe definir como se configurará el sistema para realizar las pruebas. Para obtener toda esta información se deben realizar las siguientes actividades:

- **Definir los entregables:** indicar todo aquello que se entregará como resultado final del proceso. Por ejemplo: que documentación se generará, que código se implementará y entregará para realizar las pruebas, etc.
- **Definir las tareas:** realizar a alto nivel un listado de todas las tareas que serán necesarias para realizar el proceso.
- **Escoger el entorno de prueba**: definir bajo qué condiciones se realizarán las pruebas. Si es necesario puede existir más de un entorno, pero solamente en casos muy justificados ya que trabajar con más de uno puede resultar costoso y complicar el proceso.
- **Configurar el *hardware*:** especificar las condiciones hardware necesarias para ejecutar las pruebas y la configuración adecuada.
- **Configurar el *software*:** definir qué otros programas son necesarios para ejecutar las pruebas y validarlas. Se debe documentar la configuración de estos programas.
- **Escoger los roles y las personas implicadas**: definir exactamente los roles necesarios para cada una de las actividades. Estos pueden variar en función de los recursos disponibles y las necesidades del sistema.

6. ***Programar las tareas***

Estas son las actividades que se deben realizar para programar las tareas:



-

ILUSTRACIÓN 8 DIAGRAMA DE LA ACTIVIDAD PROGRAMAR LAS TAREAS

Con toda la información recopilada en los puntos anteriores ya se podrán programar todas las tareas necesarias para completar el *proceso de pruebas*. Las fases que englobarán todas las tareas son las siguientes:

- Análisis (*test analysis*)
- Diseño (*design tasks*)
- Implementación de las pruebas (*test implementation*)
- Ejecución
- Evaluación.

Para gestionar todas las actividades de cada una de las fases se debe crear un Gantt de seguimiento que permita conocer la siguiente información:

Nombre de la tarea	Descripción	Fecha inicio	Fecha fin	Dependencias	Recursos

TABLA 1 EJEMPLO DE PLANIFICACIÓN DE TAREAS

Esta información permite gestionar y realizar un seguimiento más efectivo dentro del *proceso de pruebas*.

7. Definir riesgos y planes de contingencia

Estas son las actividades que se deben realizar para definir los riesgos y los planes de contingencia:

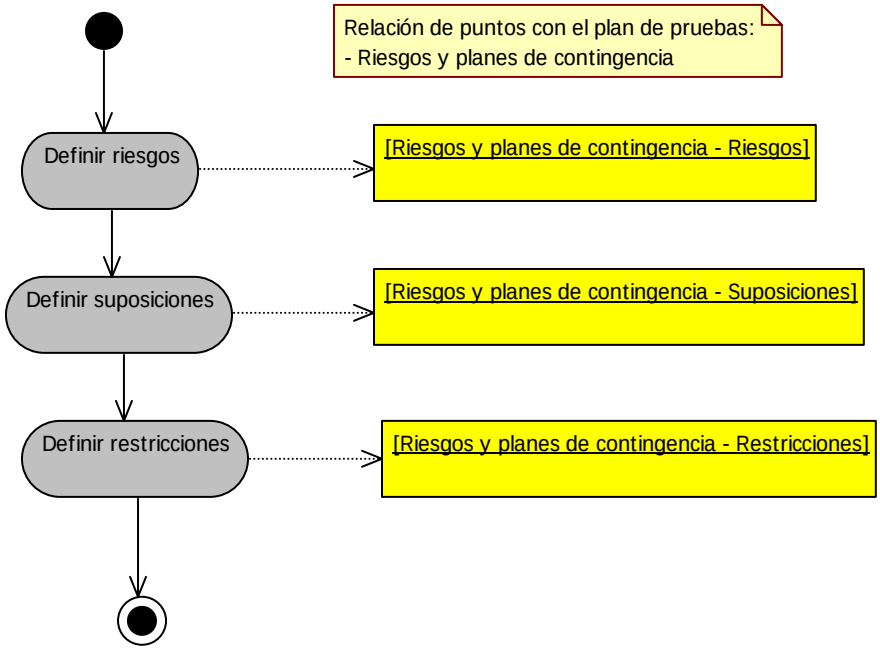


ILUSTRACIÓN 9 DIAGRAMA DE LA ACTIVIDAD DEFINIR RIESGOS Y PLANES DE CONTINGENCIA

Finalmente se definirán aquellas actividades que se deben realizar cuando la tarea no resulta como estaba prevista. Para ello se analizarán:

- **Los riesgos de las pruebas**: para cada riesgo identificado en el *proceso de pruebas* se indicará cual es la estrategia a seguir para poder controlarlo, y cuál es el plan de contingencia relacionado si este ocurriera.
- **Las suposiciones**: se indicarán cuáles son aquellas actividades que se supone correctas sin estar validadas antes. Se deberá indicar cuál es el impacto de que esta suposición falle, y cuál es la persona asignada para solucionarlo en caso de error .
- **Las restricciones**: para todos los objetos que tengan alguna limitación que pueda afectar al proceso, se deberá indicar cuál es el impacto de esta restricción en el resultado y quién es el responsable de la misma.

3.1.2. Tiempos aproximados

Estos tiempos pueden variar dependiendo de la envergadura del proyecto, pero esta fase no debería desviarse mucho de los siguientes tiempos:

Paso	Tiempo aprox.
Determinar alcance y objetivos	3 días
Determinar método de ensayo	2 días
Implementar política y/o estrategia	1 días
Determinar criterios de salida	1 día
Determinar recursos	2 días
Programar tareas	2 días
Definir riesgos y planes de contingencia	2 días

TABLA 2EJEMPLO DE TIEMPOS PARA LA FASE DE PLANIFICACIÓN

La mayor parte del tiempo de esta fase debería invertirse en reuniones, para determinar las necesidades de todas las partes y tomar decisiones sobre el proceso.

A continuación se puede ver un ejemplo de tiempos de planificación Es importante valorar que, debido al tamaño del proyecto y a la complejidad de la programación, a algunas tareas se les han asignado tiempos algo más amplios que los recomendados en la tabla anterior.

Planificación	**14 días**	**lun 19/07/10**	**jue 05/08/10**	
Determinar alcance y objetivos	**2,5 días**	**lun 19/07/10**	**mié 21/07/10**	
Determinar propósito y alcance	0,5 días	lun 19/07/10	lun 19/07/10	
Determinar información útil para la	0,5 días	lun 19/07/10	lun 19/07/10	4
Determinar fondo, objetivos y moti	0,5 días	mar 20/07/10	mar 20/07/10	5
Definir los documentos u objetos c	0,5 días	mar 20/07/10	mar 20/07/10	6
Definir esquema de inclusión y de	0,5 días	mié 21/07/10	mié 21/07/10	7
Determinar método de ensayo	**2 días**	**mié 21/07/10**	**vie 23/07/10**	3
Definir la metodogía y el proceso a	1,5 días	mié 21/07/10	jue 22/07/10	
Decidir los escenarios necesarios	0,5 días	vie 23/07/10	vie 23/07/10	10
Implementar política y estrategias	**2 días**	**vie 23/07/10**	**mar 27/07/10**	9
Consultar política y estrategia de l	1 día	vie 23/07/10	lun 26/07/10	
Definir política y estrategias del pr	1 día	lun 26/07/10	mar 27/07/10	13
Determinar los criterios de salida	**1 día**	**lun 26/07/10**	**mar 27/07/10**	13
Definir criterios de salida	0,25 días	lun 26/07/10	lun 26/07/10	
Definir criterios de entrada	0,25 días	lun 26/07/10	lun 26/07/10	16
Definir criterios de suspensión	0,25 días	mar 27/07/10	mar 27/07/10	17
Definri criterios de reanudación	0,25 días	mar 27/07/10	mar 27/07/10	18
Determinar recursos para las prueba	**3 días**	**mar 27/07/10**	**vie 30/07/10**	15
Determinar los entregables	0,5 días	mar 27/07/10	mar 27/07/10	
Definir las tareas	0,5 días	mié 28/07/10	mié 28/07/10	21
Escoger el entorno de prueba	0,5 días	mié 28/07/10	mié 28/07/10	22
Configurar el hardware	0,5 días	jue 29/07/10	jue 29/07/10	23
Configurar el software	0,5 días	jue 29/07/10	jue 29/07/10	24
Escoger los roles y personas impli	0,5 días	vie 30/07/10	vie 30/07/10	25
Programar tareas	**2,5 días**	**vie 30/07/10**	**mar 03/08/10**	20
Programar análisis	0,5 días	vie 30/07/10	vie 30/07/10	
Programar diseño	0,5 días	lun 02/08/10	lun 02/08/10	28
Programar implementación	0,5 días	lun 02/08/10	lun 02/08/10	29
Programar ejecución	0,5 días	mar 03/08/10	mar 03/08/10	30
Programar evaluación	0,5 días	mar 03/08/10	mar 03/08/10	31
Definir riesgos y planes de continge	**2 días**	**mié 04/08/10**	**jue 05/08/10**	27
Definir riesgos	1 día	mié 04/08/10	mié 04/08/10	
Definir suposiciones	0,5 días	jue 05/08/10	jue 05/08/10	34
Definir restricciones	0,5 días	jue 05/08/10	jue 05/08/10	35

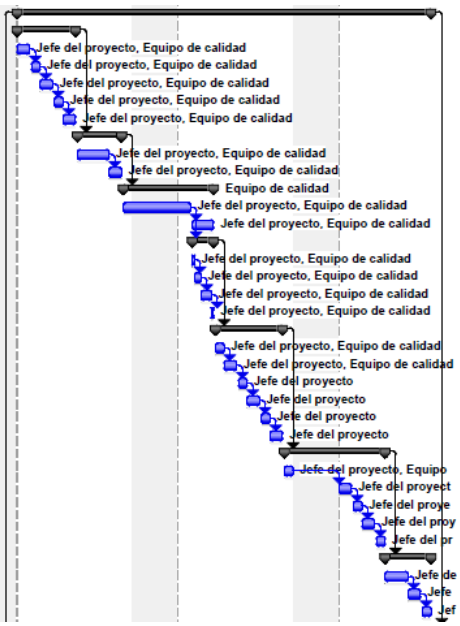


ILUSTRACIÓN 10 EJEMPLO DE TIEMPOS PARA LA FASE DE PLANIFICACIÓN

3.1.3. Recursos necesarios

Para cada tarea debe indicarse cuáles son los recursos necesarios. En la siguiente tabla se señalan los posibles roles para estas tareas:

Paso	Recursos
Determinar alcance y objetivos	Jefe de proyectos
	Equipo de calidad
Determinar método de ensayo	Jefe de proyectos
	Equipo de calidad
Implementar política y/o estrategia	Jefe de proyectos
	Equipo de calidad
Determinar criterios de salida	Jefe de proyectos
	Equipo de calidad
Determinar recursos	Jefe de proyectos
	Equipo de calidad
Programar tareas	Jefe de proyectos
	Equipo de calidad
Definir riesgos y planes de contingencia	Jefe de proyectos
	Equipo de calidad

TABLA 3 EJEMPLO DE RECURSOS PARA LA FASE DE PLANIFICACIÓN

Aquí se puede ver un ejemplo de recursos asignados a un proceso de planificación. Como la mayoría de tareas de esta fase implica tomar decisiones, generalmente se realizan reuniones para analizar la información de las pruebas y definir el proceso completo.

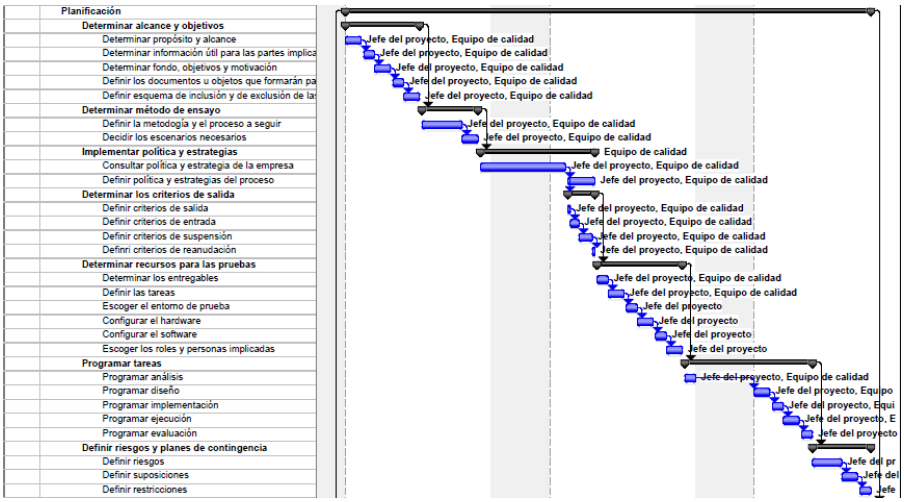


ILUSTRACIÓN 11 EJEMPLO DE RECURSOS PARA LA FASE DE PLANIFICACIÓN

3.1.4. Entregables

El entregable de esta fase es:

Plan de prueba (*test plan*): es un documento que describe el ámbito, el enfoque, los recursos y el programa de las actividades previstas para una prueba. Identifica, entre otros, los ítems de prueba, las características a ser probadas, las tareas, el grado de independencia del probador (*tester*), el entorno, las técnicas de diseño de pruebas y los criterios de entrada y salida a ser usados, así como las razones para ser escogidos y cualquier riesgo que requiera un plan de contingencia. Es el registro del proceso de planificación de pruebas (4.1*Plan de pruebas: documento de la IEEE std 829-1998*)

3.2. Fase: Control

Esta fase permite saber si el proceso está funcionando correctamente. Es transversal a todo el *proceso de pruebas* y permite conocer en todo momento el estado, las actividades que se están realizando y los resultados de las demás fases. Las actividades de control permiten saber si todas las tareas planificadas se están desarrollando de forma exitosa.

Control de prueba (*test control)*: es una tarea de gestión de la prueba que se encarga de desarrollar y aplicar el conjunto de acciones correctivas, para reconducir el proyecto de pruebas si la monitorización presenta desviaciones respecto a lo planeado

3.2.1. Actividades

El siguiente diagrama permite ver todas las actividades, entregables y decisiones que se deben tomar durante el proceso de control.

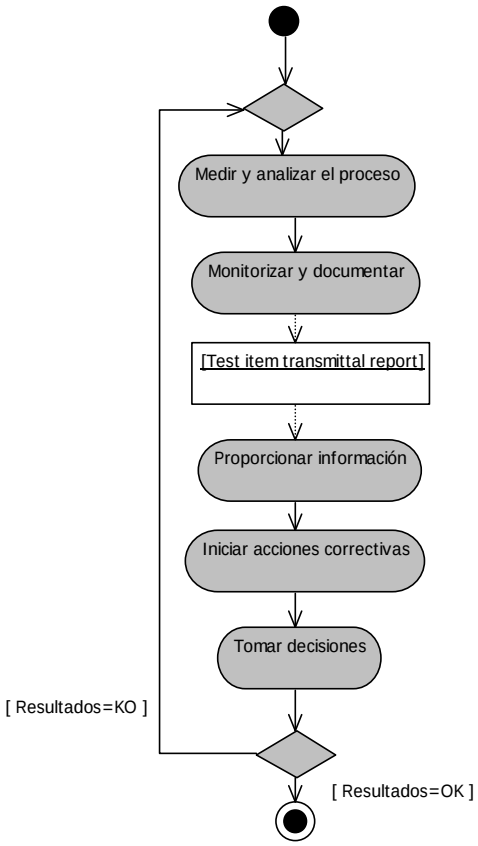


ILUSTRACIÓN 12 DIAGRAMA DE ACTIVIDADES PARA LA FASE DE CONTROL

Este proceso se inicia con la planificación, y continúa durante las siguientes fases. Es un proceso iterativo que se puede repetir varias veces en función de los resultados de las pruebas. A continuación se detallan cada una de las actividades del diagrama:

1. ***Medir y analizar***

   El primer paso, el más importante de esta fase, es definir cómo se medirá y se analizará todo el *proceso de pruebas*. Esta tarea permitirá conocer en todo momento el estado de las pruebas. Para poder realizar esta tarea es importante fijarse en:

   - **Cuántas pruebas se han hecho:** para conocer el % de cobertura.
   - **Cuántas pruebas se han superado**: para conocer el éxito de nuestro proceso.
   - **Cuántos defectos se han superado**: para poder documentarlos correctamente y no repetirlos en implementaciones futuras.
   - **De qué importancia son los defectos que se han hallado**: si los defectos encontrados son muy importantes, tal vez se deba detener el *proceso de pruebas* para revisar el diseño y la implementación del propio sistema.

2. ***Monitorizar y documentar***

   Monitorización de prueba (*test monitoring)*: es una tarea de gestión de la prueba que se encarga de las actividades relacionadas con la comprobación periódica del estado de un proyecto. Se preparan informes para comparar el estado actual con el planeado.

   Antes de poner en marcha las tareas de medición y análisis debe quedar bien definido como se monitorizará y se documentará esta información. Para ello hay que tener en cuenta los siguientes puntos:

   - **El progreso**: se debe tener claro cuál será el progreso del proceso, ya que éste permite definir en qué momentos se realizarán tareas de medición.
   - **La cobertura de pruebas:** conocer las partes del software que serán probadas permite saber el tipo de mediciones que se van a realizar y con qué herramientas se van a monitorizar.
   - **El criterio de salida**: estos valores servirán para realizar análisis de los resultados y poder presentar conclusiones sobre las pruebas realizadas. El proceso de control debe tener acceso a ellos en todo momento.

   La documentación debe permitir al equipo conocer en todo momento:

   - **Cuántas pruebas se han realizado**: es decir el % de cobertura.
   - **Cuáles son los resultados**: que pruebas han tenido un resultado exitoso y cuáles no han sido superadas.
   - **Qué conclusiones se han obtenido**: para poder conocer cuál es el estado de las pruebas.

- **Evaluación de riesgos extraídos**: para saber si las pruebas más importantes se están superando con éxito.

3. Proporcionar información sobre las pruebas

Para que la fase de control pueda ayudar al *proceso de pruebas*, la información obtenida debe compartirse. Esto debe permitir a otros agentes:

- Tomar decisiones bien documentadas y acordes al estado del proyecto.
- Utilizar la información para analizar las propias pruebas y mejorarlas durante el proceso y en futuras implementaciones.

4. Iniciar acciones correctivas.

Es importante adaptar las pruebas a los resultados que se van obteniendo, consiguiendo de esta forma mejorarlas. Algunas posibles mejoras son:

- Añadir casos que han aparecido al realizar alguna prueba y que no estaban contemplados al inicio del proyecto.
- Depurar los datos de entradas de las pruebas.
- Mejorar la dinámica del proceso.
- Modificar prioridades en función de los resultados obtenidos.

5. Tomar decisiones

Finalmente es necesario tomar decisiones basadas en toda esta información. Las decisiones las puede tomar el propio equipo que está realizando las pruebas o un equipo diferente. Para ello se debe tener clara la siguiente información:

- **Las medidas**: permitirán decidir si hay que iniciar acciones correctivas o si los resultados satisfacen los criterios de aceptación.
- **La información deducida de las pruebas**: es importante tener presente cuáles son las acciones correctivas que se deberían realizar para poder evaluar la envergadura y cómo pueden afectar al sistema y a las demás pruebas.
- **Cambios en la evaluación de riesgos**: es posible que a partir de los resultados de las pruebas, los riesgos más importantes del proceso deban ser revisados. También es importante conocer esta información para tomar decisiones en la fase de control.

3.2.2. Tiempos aproximados

En esta fase es imposible hacer un cálculo genérico de los tiempos que se tardará en realizar cada una de las tareas, ya que dependen de muchos factores. Es importante tener en cuenta el alcance, el tiempo y los recursos del proyecto. No es conveniente excederse en el número de pruebas, ya que dificultaría la determinación de los tiempos y la obtención de un proceso finito.

Paso	Tiempo aprox.
Medir y analizar	Igual que la planificación
Monitorizar y documentar	Depende de la duración del proyecto y los resultados de las pruebas
Proporcionar información	Depende de la duración del proyecto y los resultados de las pruebas
Iniciar acciones correctivas	Depende de la duración del proyecto y los resultados de las pruebas

TABLA 4 EJEMPLO DE TIEMPOS PARA LA FASE DE CONTROL

A continuación se muestra un ejemplo de tiempos de un proyecto real, de aproximadamente 3 meses de implementación. Se puede observar que la mayoría de tareas se realizan en paralelo, ya que, hipotéticamente, durarán lo mismo que durarán la implementación y la ejecución de las pruebas. Estos tiempos pueden alargarse si estas dos fases también lo hacen.

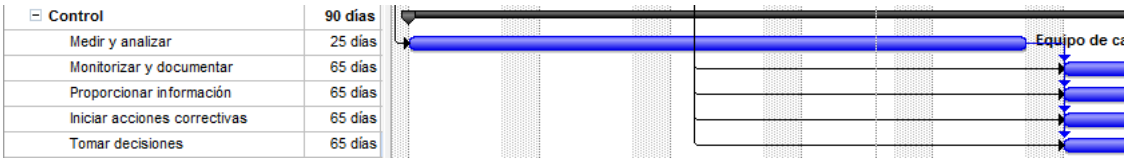


ILUSTRACIÓN 13 EJEMPLO DE TIEMPOS PARA LA FASE DE CONTROL

3.2.3. Recursos necesarios

Los recursos en esta fase variarán en función de los que posea la empresa, del conocimiento de las personas del equipo y de muchos otros factores. Generalmente será un jefe de proyecto quien se encargue del control del proceso, que puede ser el jefe del proyecto que se quiere verificar o un responsable asignado exclusivamente para el *proceso de pruebas*.

Paso	Recursos
Medir y analizar	Equipo de proyectos
	Equipo de calidad
Monitorizar y documentar	Jefe de proyectos
	Equipo de calidad
Proporcionar información	Jefe de proyectos
	Equipo de calidad
Iniciar acciones correctivas	Jefe de proyectos
	Jefe de proyectos de calidad
Tomar decisiones	Jefe de proyectos
	Jefe de proyectos de calidad

TABLA 5 EJEMPLO DE RECURSOS PARA LA FASE DE CONTROL

A continuación se puede ver un ejemplo de recursos asignados a un proyecto real. Se puede asignar más de una persona a esta tarea, ya que en algunos casos se necesita un perfil que ejecute la tarea, y otro que la controle y verifique los resultados obtenidos.

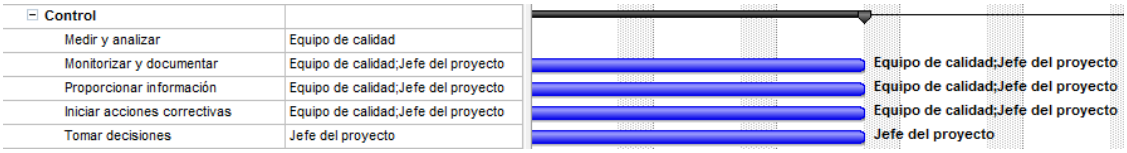


ILUSTRACIÓN 14 EJEMPLO DE RECURSOS PARA LA FASE CONTROL

3.2.4. Entregables

Test ítem transmittal report: Informe que describe la estructura de los documentos, los datos y la información del *proceso de pruebas*. Permite saber dónde se encuentra localizada toda la información referente al *proceso de pruebas*, conocer cuál es el estado de cada uno de los documentos y dónde se guardan los resultados del proceso.

3.3. Fase: Análisis y diseño

La fase de análisis y diseño de pruebas está compuesta por una serie de actividades en las que los objetivos de las pruebas se convertirán en condiciones tangibles y diseños de prueba. En esta parte del proceso se construirán los diseños y los procedimientos de las pruebas a partir de los objetivos identificados en la fase de planificación.

3.3.1. Actividades

En el siguiente diagrama se pueden observar todas las actividades, los entregables y las decisiones necesarias en esta fase.

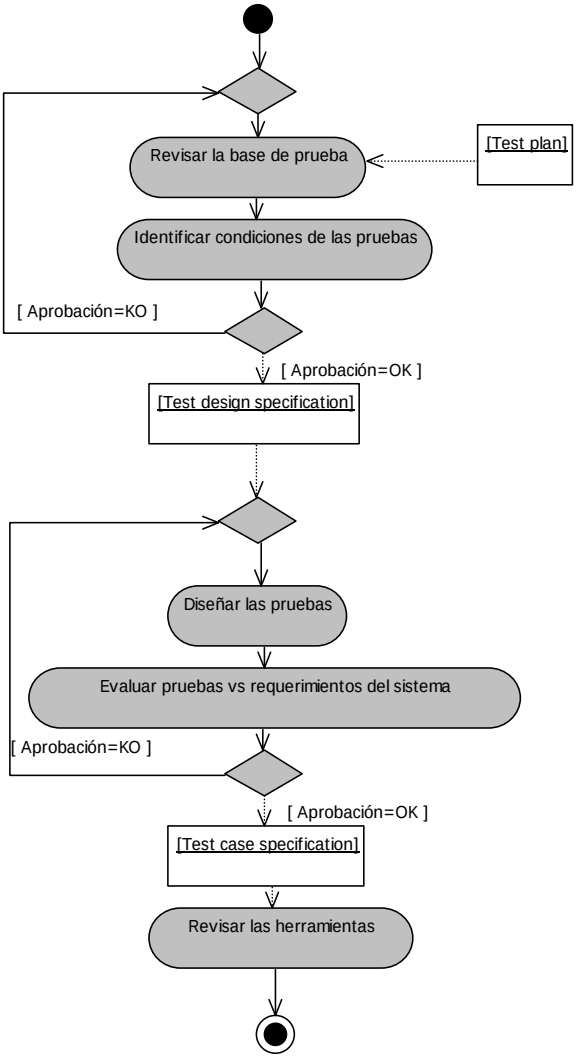


ILUSTRACIÓN 15 DIAGRAMA DE ACTIVIDADES PARA LA FASE DE ANÁLISIS Y DISEÑO

Se pueden observar dos iteraciones en el diagrama, la primera antes de entregar las especificaciones del diseño y la segunda antes de entregar los casos de prueba. Como

estos documentos son fundamentales para las siguientes fases es importante depurar su contenido y revisarlo con todas las partes implicadas. Antes de dar por finalizado un documento todas las partes implicadas debe aprobarlo para no arrastrar ambigüedades en los siguientes pasos.

1. ***Revisar la base de prueba (test basis)***

   Base de prueba (*test basis)*: son todos los documentos a partir de los cuales los requerimientos de un componente o sistema pueden ser inferidos. Es la documentación en la que se basan los casos de prueba. Si un documento puede ser modificado sólo a través de un proceso de modificación formal, la base de prueba se llama base de prueba helada (*frozen test basis*).

   Condición de prueba (*test condition)*: es un ítem o evento de un componente o sistema que puede ser verificado por uno o más conjuntos de pruebas.

   Durante esta fase se examinan las especificaciones del *software* que se va a probar y la documentación obtenida en los pasos anteriores. La base de prueba, ayuda a construir las pruebas puesto que definen lo que el sistema debe hacer una vez construido. Estudiando la base de prueba se identifican brechas y ambigüedades en la especificación.

2. ***Identificar condiciones de las prueba***

   Antes de empezar a diseñar propiamente, hay que crear un listado a alto nivel de aquello que se va a probar. Para ello se analizan los ítems detectados durante la fase de planificación y documentados en el *plan de pruebas*. Hay que fijarse en:

   - **Sus especificaciones**: saber exactamente qué debe cumplir ese ítem.
   - **Su comportamiento**: conocer qué actividades realizará y cuáles son los procesos en los que participará.
   - **Su estructura**: tener conocimiento de cómo va a implementarse cada uno de los ítems que serán validados.

3. Diseñar las pruebas

Llegados a este punto se empiezan a diseñar las pruebas. Es importante seleccionar las más representativas y relacionadas con partes del *software* que pueden suponer riesgos de mayor interés, basando esta decisión en las condiciones de prueba. Hay que tener en cuenta que el coste de probar todo el sistema generalmente es muy alto, por lo que se deben identificar las tareas más comunes y las más críticas.

Para cada una de las funciones se definirá:

- El caso de prueba.
- Los procedimientos que se deben seguir para ejecutar cada uno de los casos de prueba a alto nivel.

4. Evaluar las pruebas vs. los requerimientos

Como se ha podido ver en los puntos anteriores las pruebas están muy relacionadas con los requerimientos del sistema, por lo que es importante asegurarse y validar que:

- Los requerimientos están bien definidos
- Los requerimientos tienen una única interpretación que queda reflejada en las pruebas relacionadas con cada uno de los requerimientos.

5. Revisar las herramientas

Las herramientas necesarias para el *proceso de pruebas* han sido definidas durante la fase de planificación. Se puede consultar esta información dentro del documento de *plan de pruebas*. Para finalizar la fase de diseño se debe revisar esta información, y además:

- **Diseñar la configuración del entorno de pruebas**. Para ejecutar las pruebas es posible que se necesiten varias configuraciones del entorno, por lo que es importante dejar documentado cómo será cada uno de ellos y qué información se debe tener en cuenta antes de ejecutar una nueva prueba.
- **Identificar cualquier infraestructura requerida y herramienta**s. Esto implica revisar todas las dependencias con las herramientas que serán necesarias. Es posible que para finalizar este punto se necesite la ayuda de un experto en sistemas.

3.3.2. Tiempos aproximados

Los tiempos de esta fase varían según el tamaño del proyecto pero no pueden variar mucho de los aquí indicados, ya que si se exceden podrían retrasar el resto del proceso. Si se dedica demasiado tiempo al diseño se puede generar un retraso importante sobre todo el proyecto. A tener en cuenta que estos tiempos pueden variar mucho si el proyecto es muy complejo o muy grande.

Paso	Tiempo aprox.
Revisar la base de las pruebas	3 días
Identificar las condiciones de las pruebas	2 días
Diseñar las pruebas	5 días
Evaluar las pruebas vs los requerimientos	2 días
Revisar las herramientas	3 días

TABLA 6 EJEMPLO DE TIEMPOS PARA LA FASE ANÁLISIS Y DISEÑO

La duración de la actividad de "diseñar las pruebas" puede variar mucho dependiendo de la cantidad de pruebas que se deban diseñar.

A continuación se muestran unos tiempos aproximados para un proyecto real.

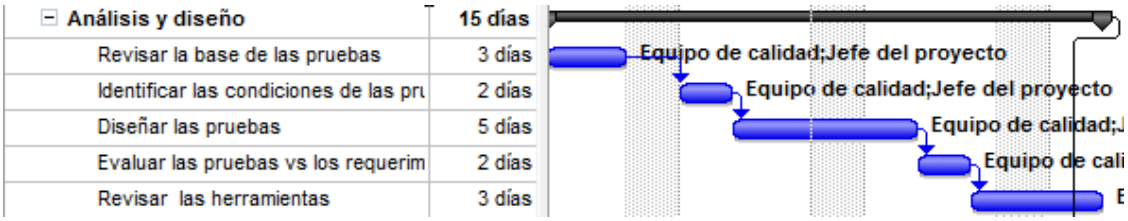


ILUSTRACIÓN 16 EJEMPLO DE TIEMPOS PARA LA FASE DE ANÁLISIS Y DISEÑO

3.3.3. Recursos necesarios

Los recursos para esta fase son los expertos en pruebas. Es muy importante realizar un buen análisis y un buen diseño, porque las decisiones tomadas en esta fase afectarán a todo el proceso, y puede resultar muy costoso rectificarlas.

Paso	Recursos
Revisar la base de las pruebas	Jefe de proyectos
	Equipo de calidad
Identificar las condiciones de las pruebas	Jefe de proyectos
	Equipo de calidad
Diseñar las pruebas	Jefe de proyectos
	Equipo de calidad
Evaluar las pruebas y los requerimientos	Jefe de proyectos
	Equipo de calidad
Revisar las herramientas	Equipo de desarrollo
	Equipo de calidad

TABLA 7EJEMPLO DE RECURSOS PARA LA FASE DE ANÁLISIS Y DISEÑO

A continuación se puede ver una estimación de recursos para un proyecto real. En este proyecto se ha querido implicar, en esta fase, la mayor cantidad de expertos, para garantizar un análisis y diseño muy bien definido, aunque pueda resultar más costoso de lo habitual debido a los perfiles de las personas implicadas. No es necesario implicar al jefe de proyectos si ya se tiene un equipo de calidad. Este podría simplemente generar una buena documentación del sistema sobre el cual se ejecutarán las pruebas, y así facilitar el trabajo al equipo de calidad y validar la documentación obtenida al final de la fase. Pero en este caso se ha decidido que participe en todo el proceso.

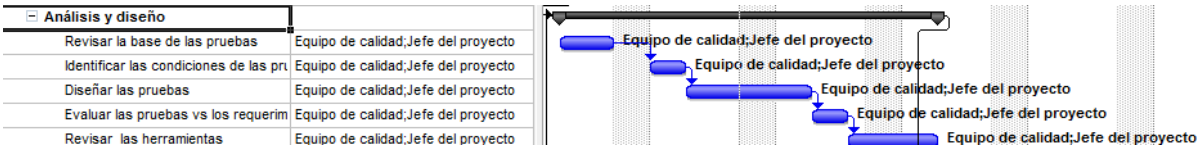


ILUSTRACIÓN 17 EJEMPLO DE RECURSOS PARA LA FASE DE ANÁLISIS Y DISEÑO

3.3.4. Entregables

Caso de prueba (*test case*): es un conjunto de valores de entrada, precondiciones de ejecución, resultados esperados y post-condiciones de ejecución, desarrollado para un objetivo particular o condición de prueba.

Especificación de diseño de prueba (*test design specification*): es un documento que especifica las condiciones de prueba (*coverage ítems*) para un ítem de prueba, el enfoque detallado de prueba y los casos de prueba de alto nivel asociados.

3.4. Fase: Implementación

La fase de implementación permite, a partir de las condiciones de las pruebas, crear los **conjuntos de pruebas** y el ***testware***. Durante esta fase también se configura el entrono donde se ejecutarán las diferentes pruebas y donde se construirán nuestros **datos de prueba**.

Conjunto de pruebas (*test suite*): es un conjunto de diferentes casos de prueba para un componente o sistema bajo prueba, donde la post-condición de una prueba es comúnmente usada como precondición para el siguiente.

***Testware*:** es todo el material producido durante el proceso de prueba requerido para la planificación, diseño y la ejecución de pruebas, tales como documentación, guiones, entradas, resultados esperados, procedimientos, archivos, bases de datos, entornos o cualquier *software* adicional o utilidad usada en el *proceso de pruebas*.

Datos de prueba (*test data*): son los datos que existen antes de que la prueba sea ejecutada, y que afectan o son afectados por el componente o sistema bajo pruebas.

3.4.1. Actividades

A continuación se muestran las actividades de la fase de implementación:

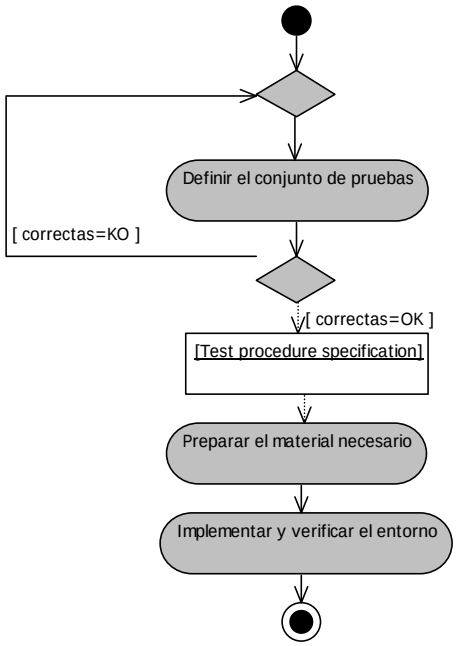


ILUSTRACIÓN 18 DIAGRAMA DE ACTIVIDADES PARA LA FASE DE IMPLEMENTACIÓN

1. ***Definir los conjuntos de pruebas***

   El primer paso es procesar la información que ha llegado de la fase anterior para definir los conjuntos de pruebas. Para ello se debe:

   - **Priorizar los conjuntos**: a partir de las prioridades del proyecto y de los riesgos definidos en la fase de planificación.
   - **Desarrollar los conjuntos:** agrupar aquellas pruebas que se puedan ejecutar a la vez, para optimizar tiempos y recursos.
   - **Crear datos de prueba para los conjuntos**: revisar todos los datos de las pruebas agrupadas en un mismo conjunto para generar datos de entrada genéricos. Este proceso puede ayudar a optimizar los datos de entrada.
   - **Describir los procedimientos de prueba**: se deben revisar las automatizaciones para poder ejecutar todas las pruebas del conjunto a la vez. También se debe documentar paso por paso el procedimiento a seguir, tanto para los conjuntos de pruebas como para las individuales.

2. ***Preparar el material necesario:***

   Antes de empezar la implementación de las pruebas, se debe revisar todo el material para adaptarlo a los cambios realizados en el punto anterior.

   - **Crear conjuntos de pruebas:** se deben revisar los casos obtenidos en la fase de diseño y adaptarlos a los conjuntos identificados en el paso anterior.
   - **Configurar un programa de ejecuciones** (*execution Schedule*): es recomendable automatizar las ejecuciones, por lo que, antes de empezar la implementación, es importante escoger un programa adecuado para las pruebas que se quieren realizar y ejecutarlo según las necesidades de las mismas.

3. ***Implementar y verificar el entorno.***

   Antes de empezar la ejecución hay que asegurar que el entorno está bien configurado, para poder ejecutar pruebas específicas sobre el mismo.

   Ejecución de la prueba (*test execution*): Es el proceso de ejecutar una prueba por parte del componente o sistema, produciendo los resultados actuales.

   Una vez está todo configurado correctamente, se puede empezar la implementación. Esta puede variar mucho en función del software que se vaya a utilizar y de la tipología de pruebas que se quieren realizar.

3.4.2. Tiempos aproximados

Los tiempos de esta fase pueden variar mucho dependiendo de los resultados obtenidos en las fases anteriores, del conocimiento del equipo en el entorno de las pruebas y de la complejidad de las automatizaciones.

También hay que tener en cuenta que, hasta no conocer el volumen de pruebas, es difícil estimar los tiempos de esta fase. Por eso mismo es posible que las estimaciones realizadas en la fase de planificación se vean modificadas al llegar a esta fase. Es recomendable realizar un proceso iterativo de pruebas. Esto permite adaptar el volumen de pruebas que se van a realizar a los tiempos y a los resultados obtenidos.

Paso	Tiempo aprox.
Definir conjuntos de pruebas	Depende de la cantidad de pruebas
Preparar el material necesario	Depende del proyecto
Implementar y verificar entorno	Depende del entorno del proyecto

TABLA 8 EJEMPLO DE TIEMPOS PARA LA FASE DE IMPLEMENTACIÓN

A continuación se pueden ver unos tiempos reales para una fase de ejecución. Se destinan un par de semanas aproximadamente a realizar todas las actividades necesarias para esta fase.

ILUSTRACIÓN 19 EJEMPLO DE TIEMPOS PARA LA FASE DE IMPLEMENTACIÓN

3.4.3. Recursos necesarios

Los recursos de estas fases pueden variar mucho según los recursos de la organización y el conocimiento de nuestros equipos. Es recomendable que la persona encargada de estas tareas sea experta en pruebas, y en el software que se va a usar para realizarlas. Si no es así debería formarse para optimizar esta fase.

Paso	Recursos
Definir conjuntos de pruebas	Equipo del proyectos
	Equipo de calidad
Preparar el material necesario	Equipo de sistemas
	Equipo de calidad
Implementar y verificar el entorno	Equipo de sistemas y del proyecto
	Equipo de calidad

TABLA 9 EJEMPLO DE RECURSOS PARA LA FASE DE IMPLEMENTACIÓN

Si se realizan pruebas estructurales sobre el sistema, las personas que vayan a implementar los módulos serán incluidos en la implementación de las mismas, porque conocen la estructura interna del programa mucho mejor que las personas que no han participado en el desarrollo. A continuación se muestra un ejemplo de planificación donde se implica al equipo de sistemas y de proyectos dentro de la fase de implementación de las pruebas.

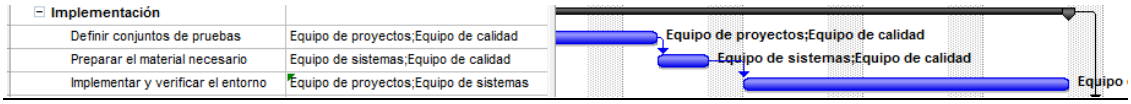


ILUSTRACIÓN 20 EJEMPLO DE RECURSOS PARA LA FASE DE IMPLEMENTACIÓN

3.4.4. Entregables

Especificación de procedimiento de prueba (*test procedure specification, test script, manual test script*): es un documento que especifica una secuencia de acciones para la ejecución de una prueba.

3.5. Fase: Ejecución

En esta fase se debe conocer toda la información generada en las fases anteriores. Para ello se puede consultar cada uno de los entregables que se han obtenido como salida de las mismas.

Esta información permite ejecutar las pruebas y registrar los resultados, que son las dos actividades básicas de la fase de ejecución. Una vez registrada toda la información se deberá analizar y tomar las decisiones adecuadas a los resultados obtenidos.

3.5.1. Actividades

El siguiente diagrama muestra todas las actividades y los entregables de esta fase. Hay que tener en cuenta que el *test log* debería crearse de forma automática durante la ejecución de cada una de las pruebas o conjunto de pruebas.

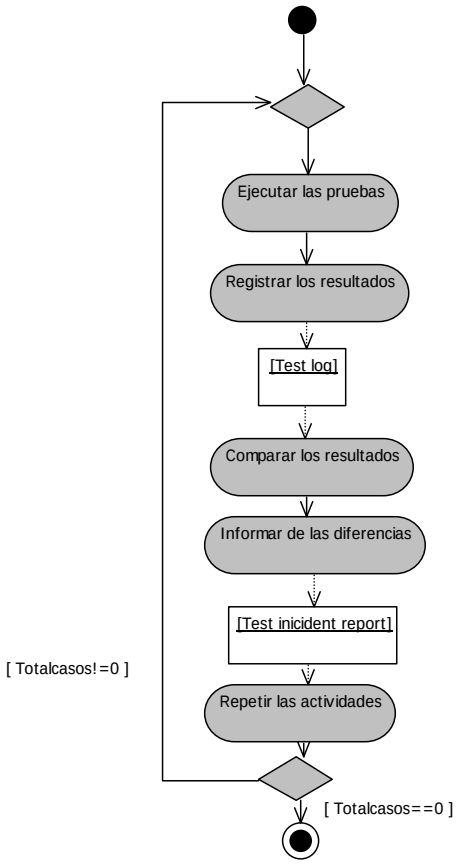


ILUSTRACIÓN 21 DIAGRAMA DE ACTIVIDADES DE LA FASE DE EJECUCIÓN

Este proceso se deberá realizar para cada uno de los casos definidos en los puntos anteriores. Es recomendable analizar criterios al final de cada ejecución, para poder

abortar el proceso lo más temprano posible y no arrastrar defectos en las siguientes ejecuciones.

1. ***Ejecutar las pruebas***
   Esta actividad es la base de todo el *proceso de pruebas*. Se deben consultar las prioridades de los casos que se han implementado y conocer los datos de entrada para cada uno de ellos. Se ejecutarán:
   - Los conjuntos de pruebas
   - Casos de prueba individuales

   Es recomendable ejecutarlos en este orden, ya que en los conjuntos se pueden encontrar más defectos que en los casos individuales. Además, en los conjuntos se encuentran normalmente defectos más genéricos que, si no se solucionan al principio, se arrastran en las siguientes ejecuciones.

2. ***Registrar los resultados***
   Es importante tener registros de todo lo que ocurre durante la prueba, ya que sin esta información puede resultar muy complicado resolver un defecto o identificar exactamente qué es lo no está funcionando correctamente. Para que los registros puedan resultar útiles se debe:

   - Registrar los resultados de la ejecución de la prueba
   - Grabar las identidades y versiones del *software* bajo prueba.

   Gracias a esta información se podrá:

   - Reportar defectos en versiones específicas
   - Reportar el registro que proveerá una pista de auditoría.

3. ***Comparar los resultados***
   Una vez ejecutadas las pruebas y obtenidos los resultados, éstos se deberán comparar con los esperados (qué pasó vs. qué tenía que pasar). Esta actividad será la que permitirá abortar el proceso en caso que los resultados no estén cumpliendo los criterios esperados.

4. ***Informar de las diferencias:***
   Si la prueba no se ha superado con éxito se debe recopilar la siguiente información:

   - Detectar dónde hay diferencias entre los resultados actuales y los esperados
   - Informar de las discrepancias como incidentes
   - Analizar para poder obtener más datos sobre el defecto.

   Esta información permitirá resolver los incidentes detectados.

5. ***Repetir las actividades***

Una vez solucionadas las incidencias detectadas durante la ejecución de un caso, se deben repetir todas las actividades de las pruebas para verificar una a una todas las soluciones realizadas sobre el sistema.

Es decir, volver a ejecutar las pruebas y conjuntos de pruebas que previamente habían fallado para confirmar la modificación o arreglo (*confirmation testing* o *re-testing*). Para ello:

- Se volverán a ejecutarán pruebas y conjuntos de pruebas si hay defectos.
- Se probará el *software* corregido de nuevo, para asegurar que el defecto ya había sido corregido correctamente (*confirmation test*).
- Se verificará que no se han introducido nuevos defectos en áreas no modificadas del software al corregir un defecto (*regression testing*).

Incidente (*incident*): Es cualquier evento que ocurra que requiera investigación.

Re-testing (*confirmation testing*): *Testing* que ejecuta conjuntos de pruebas que fallaron la última vez que fueron ejecutados, con la intención de verificar el éxito de las acciones correctivas.

Pruebas regresivas (*regressions testing*): Son pruebas de un programa previamente probado tras una modificación, para asegurar que no se han introducido o descubierto defectos en áreas no modificadas del software como resultado de los cambios hechos.

3.5.2. Tiempos aproximados

La estimación de tiempos de la fase de ejecución está muy relacionada con la cantidad de pruebas que se van a realizar y la complejidad de las mismas.

Paso	Tiempo aprox.
Ejecutar las pruebas	Depende de la cantidad de pruebas y de su complejidad
Registrar los resultados	Generalmente se automatiza y dura como el paso anterior
Comparar los resultados	Varía según la cantidad de pruebas
Informar de las diferencias	Varía en función de la cantidad de fallos encontrados
Repetir las actividades	Varía en función de los puntos anteriores

TABLA 10 EJEMPLO DE TIEMPOS PARA LA FASE DE EJECUCIÓN

Estos tiempos pueden reducirse si se utilizan programas para la validación del software que permitan automatizar las pruebas.

En el siguiente diagrama se puede ver un ejemplo de tiempos para una fase de ejecución. Pertenece a un proyecto que tiene aproximadamente 50 días de implementación. La ejecución de las pruebas empieza 20 días después del inicio de la implementación, y finaliza unos días después. Estos tiempos pueden variar, ya que el hecho de repetir las actividades supone una dilación en el tiempo que depende de la cantidad de defectos encontrados.

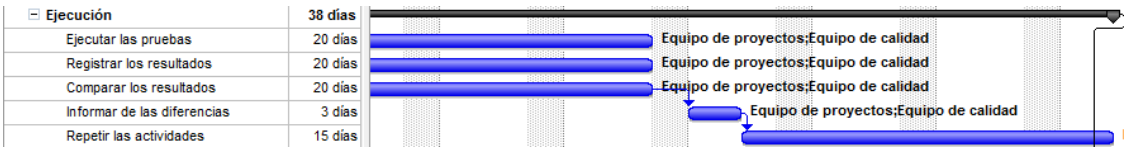


ILUSTRACIÓN 22EJEMPLO DE TIEMPOS PARA LA FASE DE EJECUCIÓN

3.5.3. Recursos necesarios

En esta fase los perfiles de los recursos pueden ser menos estratégicos, por lo que este tipo de tareas las realizará el propio equipo de desarrollo, en caso de no tener equipo de calidad, o, en el mejor de los casos, un equipo de calidad totalmente familiarizado con el software y el proyecto.

Paso	Recursos
Ejecutar las pruebas	Equipo de desarrollo
	Equipo de calidad
Registrar los resultados	Equipo de desarrollo
	Equipo de calidad
Comparar los resultados	Equipo de desarrollo
	Equipo de calidad
Informar de las diferencias	Equipo de desarrollo
	Equipo de calidad
Repetir las actividades	Equipo de desarrollo
	Equipo de calidad

TABLA 11 EJEMPLO DE LOS RECURSOS DE LA FASE DE EJECUCIÓN

En el siguiente cuadro se observa cómo los recursos que participan en cada actividad pueden pertenecer tanto al equipo del proyecto como al equipo de calidad. El equipo de calidad se encargará de ejecutar las pruebas funcionales relacionadas con los casos de uso del sistema, y el equipo de proyectos verificará las demás.

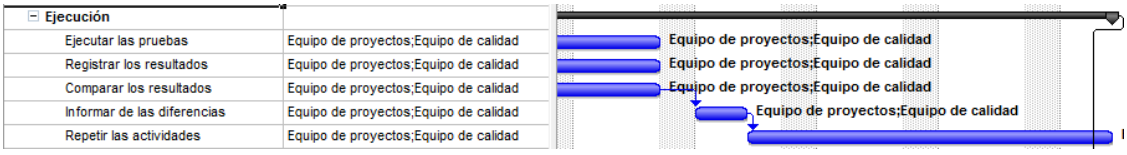


ILUSTRACIÓN 23EJEMPLO DE LOS RECURSOS DE LA FASE DE EJECUCIÓN

3.5.4. Entregables

Registro de prueba (*test log*): es un registro cronológico de los detalles relevantes sobre la ejecución de las pruebas.

***Test incident report*:** es el documento que recopila cualquier evento ocurrido durante la ejecución que requiera investigación.

3.6. Fase: Evaluación

En la fase de evaluación se verifican todos los criterios de salida. Se evalúan los resultados de las pruebas contra los objetos definidos en las fases anteriores a la ejecución. Esta actividad se debe realizar para cada nivel de prueba, puesto que se debe saber si se han hecho las pruebas suficientes en cada uno de los niveles incluidos en nuestras pruebas.

También se puede usar una combinación de criterios de cobertura y finalización, y utilizar criterios de aceptación. También se puede optar por trabajar simplemente con los criterios de salida del proceso.

Estos criterios son diferentes para cada proyecto. Deben estar muy bien especificados y no deben contener ambigüedades. Para evaluar criterios de salida se debe trabajar con métricas que permitan eliminar las ambigüedades. Existen muchas métricas, pensadas para casos muy diversos. Es muy importante tener definidas exactamente las que se usarán para cada uno de los casos.

3.6.1. Actividades

Estas son las actividades de la fase de evaluación:

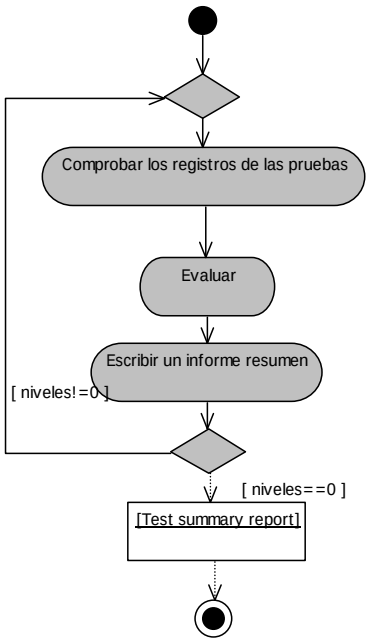


ILUSTRACIÓN 24 DIAGRAMA DE ACTIVIDADES DE LA FASE DE EVALUACIÓN

1. ***Comprobar los registros de las pruebas.***
   Al empezar esta fase se deben comparar los registros de las pruebas con los criterios de salida. Generalmente en la fase de ejecución ya se han descartado algunos defectos y se pueden haber abortado procesos. Por eso en esta fase los defectos serán de bajo riesgo. Al comparar los resultados se buscarán:
   - **Las evidencias de que las pruebas han sido ejecutadas y comprobadas correctamente.** Para justificar que realmente han superado las pruebas, se usarán métricas. Las métricas permiten verificar que los resultados obtenidos por nuestro sistema cumplen las expectativas definidas en los requerimientos.
   - **Qué defectos han surgido:** es importante conocer la tipología de defectos encontrados durante el proceso de prueba, para corregirlos en futuras implementaciones.
   - **Qué defectos se han arreglado**: para poder evaluar todo el proceso y no volver a repetir este tipo de incidencias.
   - **Si se les ha aplicado la prueba de regresión o bien están pendientes**: es importante saber si se ha verificado que las soluciones ejecutadas para arreglar defectos no han afectado a otras partes del sistema.

2. Evaluar

Un vez se conoce toda esta información se deben analizar los resultados y tomar decisiones. Durante la evaluación se debe decidir:

- **Si son necesarias más pruebas:** tal vez la experiencia o el resultado indican que pueden haber más defectos ocultos que no han sido identificados y que pueden tener alto riesgo.
- **Si el criterio de salida especificado debe ser cambiado:** es posible que se tengan que modificar los porcentajes de aceptación, o que no se hayan contemplado resultados de salida incorrectos que puedan afectar al sistema, o puede ser también que los casos de uso no tengan especificadas alternativas que puedan generar errores en la ejecución.
- **Si no se han realizado todas las pruebas diseñadas o alcanzado la cobertura esperada**: como se ha indicado anteriormente, debido a un exceso de pruebas no superadas o a una mala estimación de tiempo, es posible que no se hayan ejecutado todos los casos. Llegados a este punto se debe evaluar si las pruebas descartadas deben ser ejecutadas en una nueva iteración
- **Si los riesgos del proyecto han aumentado**: también es importante evaluar si han aparecido nuevos riesgos dentro del proyecto no contemplados al inicio. Si es así se deberán tomar decisiones que afectan a todo el proceso.

3. Escribir un informe resumen

Finalmente se debe realizar un informe para los agentes implicados. Es suficiente que los probadores (*testers*) sepan los resultados de las pruebas. Los interesados deben conocer que se han realizado las pruebas y sus resultados, para tomar decisiones informadas sobre el *software*.

3.6.2. Tiempos aproximados

Los tiempos de esta fase pueden variar según la cantidad de pruebas rechazadas o aceptadas durante el proceso. También pueden variar estos tiempos si muchas pruebas críticas son rechazas, ya que afectan a todo el conjunto del proyecto.

Paso	Tiempo aprox.
Comprobar registras de las pruebas	3 días
Evaluar	2 días
Escribir informe	5 días

TABLA 12EJEMPLO DE LOS TIEMPOS DE LA FASE DE EVALUACIÓN

En este ejemplo de tiempos sobre un proyecto real, se puede observar que el tiempo para comprobar los registros casi dobla al aconsejado en el punto anterior. Esto es debido a la cantidad de pruebas que se van a ejecutar. Hay que recordar que es una estimación previa, y es posible que durante el proceso este tiempo sea modificado en función de los resultados de la fase de ejecución.

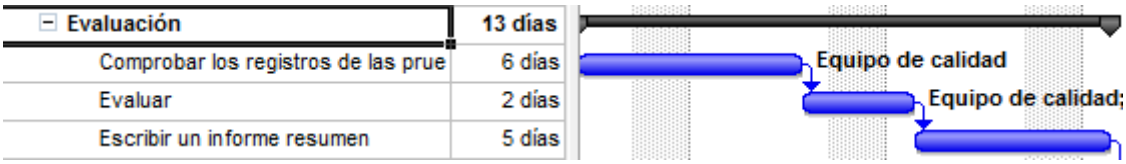


ILUSTRACIÓN 25 EJEMPLO DE LA FASE DE EVALUACIÓN

3.6.3. Recursos necesarios

Los recursos asignados a esta fase deben ser expertos tanto en el *proceso de pruebas* como en el mismo proyecto. Además es indispensable que hayan participado o supervisado todas las actividades del proceso. Por eso es recomendable que sean o el Jefe del proyecto o el equipo de calidad quienes realicen estas tareas.

Paso	Recursos
Comprobar registras de las pruebas	Equipo de desarrollo
	Equipo de calidad
Evaluar	Jefe de proyectos
	Equipo de calidad
Escribir informe	Jefe de proyectos
	Equipo de calidad

TABLA 13 EJEMPLO DE RECURSOS DE LA FASE DE EVALUACIÓN

En el ejemplo descrito a continuación se pueden observar dos perfiles asignados a las actividades: el equipo de calidad, que conoce el proceso al detalle, ha participado en todas las fases y tiene el conocimiento suficiente para evaluar los resultados; y el jefe de proyecto que es la persona que mejor conoce el sistema y las necesidades que se han evaluado. Además, el jefe del proyecto conoce cualquier modificación dentro de la implementación y especificación del proyecto, y puede saber si se han contemplado los cambios en el *proceso de pruebas*.

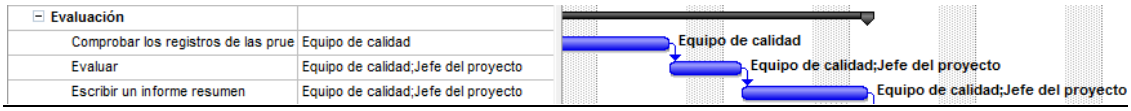


ILUSTRACIÓN 26 EJEMPLO DE RECURSOS DE LA FASE DE EVALUACIÓN

3.6.4. Entregables

Test summary report: es el documento final donde se recopilan todos los resultados y conclusiones de las pruebas.

3.7. Fase: Cierre

En la fase de cierre se pretende organizar toda la información relacionada con las pruebas, para consolidar la experiencia del proceso y del equipo que ha participado en el proceso. Para ello se recopila toda información obtenida en las diferentes fases, tanto los documentos de definición, como las conclusiones obtenidas o los artefactos que han formado parte de todo el proceso.

Esta fase generalmente se activa una vez se ha entregado el software al cliente y éste lo ha aceptado, ya que toda la información puede ser modificada hasta el último día del proyecto. Muchas organizaciones se olvidan de realizar estas actividades, muy importantes para poder reutilizar todo el conocimiento obtenido en futuros procesos.

3.7.1. Actividades

A continuación se describen las actividades y los entregables que participan en la fase de cierre.

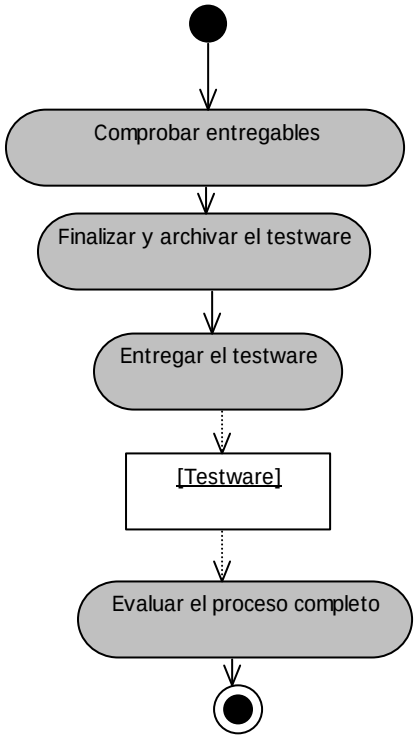


ILUSTRACIÓN 27 DIAGRAMA DE ACTIVIDADES DE LA FASE DE CIERRE

Es importante tener un repositorio de información relacionada con las pruebas donde se pueda consultar esta información. También es importante usar un software de gestión de incidencias que permita enriquecer este tipo de información y consultarla en un futuro.

1. ***Comprobar los entregables***

   Comprobar qué entregables planificados han sido entregados en cada una de las fases. Se verificará si:

   - Se tienen todos los entregables definidos
   - Si se posee la última versión de cada uno de ellos
   - Si han sido actualizados adaptándose a los cambios aparecidos durante el proceso

   También hay que asegurar que todos los informes de incidentes han sido resueltos hasta la reparación o el aplazamiento.

2. ***Finalizar y archivar el testware***

   Se deberá guardar toda la información relacionada con el *testware* y verificar, en caso que existan varias versiones del mismo, que toda la información es coherente y está actualizada.

3. ***Entregar el testware***

   Si el equipo encargado del mantenimiento no es el mismo que ha realizado la implementación, o forma parte de otra empresa, se le deberá entregar toda la información para que pueda mantener y modificar el sistema. Para ello es recomendable entregar algún tipo de documentación explicando el *proceso de pruebas* que se ha seguido.

4. ***Evaluar el proceso completo***

   Se deberá evaluar el resultado del *proceso de pruebas*, y analizar las lecciones aprendidas para futuros proyectos. Para ello se puede usar algún software que facilite esta información, como un gestor de incidencias, y realizar algún tipo de informe para que en futuros procesos se pueda acceder a la información más relevante de forma rápida y ágil.

3.7.2. Tiempos aproximados

Los tiempos de esta fase se basan en reuniones de análisis y de entrega de documentación, por lo que es complicado estimar cuál debe ser su duración exacta. Además, el principal problema de esta fase es que no se inicia hasta que el software ha sido entregado, y eso puede tener como consecuencia que se solape con otras fases de otros proyectos.

Paso	Tiempo aprox.
Comprobar entregables	X días
Finalizar y archivar el *testware*	X días
Entregar el *testware*	X días
Evaluar el proceso completo	X días

TABLA 14 EJEMPLO DE TIEMPOS DE LA FASE DE CIERRE

En el siguiente ejemplo se ha estimado un tiempo de aproximadamente 4 días para realizar el cierre. Pero este tiempo puede variar dependiendo de las tareas de las personas implicadas.

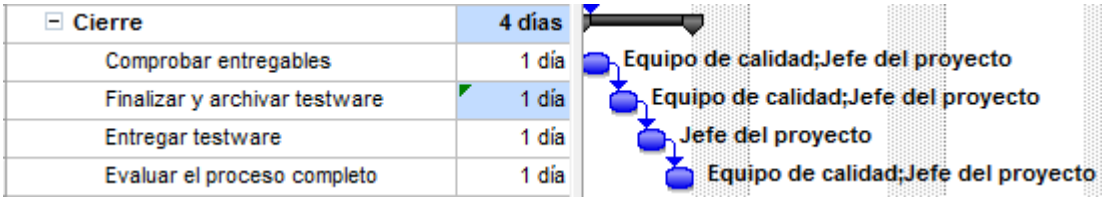


ILUSTRACIÓN 28 EJEMPLO DE TIEMPOS DE LA FASE DE CIERRE

3.7.3. Recursos necesarios

Lo interesante de esta fase es que pudieran participar todas las personas implicadas y aportar su conocimiento en cada una de las actividades. El problema generalmente es que los recursos, al llegar a esta fase, ya han sido asignados a nuevos proyectos y no pueden implicarse en estas tareas. A pesar de ello, es importante que alguien se responsabilice de cerrar el proyecto, pues de esta forma se podrá reutilizar el conocimiento obtenido.

Paso	Recursos
Comprobar entregables	Jefe de proyectos
	Equipo de calidad
Finalizar y archivar el *testware*	Jefe de proyectos
	Equipo de calidad
Entregar el *testware*	Jefe de proyectos
Evaluar el proceso completo	Jefe de proyectos
	Equipo de calidad

TABLA 15 EJEMPLO DE RECURSOS DE LA FASE DE CIERRE

A continuación se puede observar un ejemplo de recursos aplicados a las diferentes actividades de la fase. En este caso se ha decidido que el jefe del proyecto se encargue de cerrar toda la información relacionada con el mismo. Como el jefe del proyecto es generalmente el encargado de cerrar el proyecto en sí, es bueno que también cierre el *proceso de pruebas*, ya que dispone de más tiempo que los demás recursos al no tener, generalmente, tareas relacionadas con otros proyectos pendientes.

ILUSTRACIÓN 29 EJEMPLO DE RECURSOS DE LA FASE DE CIERRE

3.7.4. Entregables

***Testware*:** Es la recopilación de todos los artefactos producidos durante el proceso de prueba y requeridos para la planificación, diseño y ejecución de pruebas. Se guardará como resultado de todo el *proceso de pruebas*.

4. Documentación para el proceso de pruebas

En cada una de las fases del *proceso de pruebas* se ha indicado cuál es la documentación que se debe obtener como resultado de sus actividades. Esta documentación permite que todas las partes implicadas conozcan los resultados obtenidos en cada una de las fases.

Uno de estos documentos es especialmente importante: el *plan de pruebas.* Debe redactarse en todos los procesos de prueba que se realicen sobre cualquier producto software.

A continuación explicaremos cómo debe ser este documento, según la IEEE, y mostraremos cómo se ha aplicado en un proyecto real.

4.1. Plan de pruebas: documento de la IEEE std 829-1998

Propósito

Es un documento que describe el enfoque, los recursos y los tiempos de las actividades que se realizarán durante el *proceso de pruebas*. El documento permitirá identificar los ítems que serán probados, las características que se quieren validar, las tareas que se deben realizar, el personal responsable de cada tarea y los riegos asociados con este plan

Generalmente puede ser un documento muy estándar para todos los proyectos de una misma empresa, o para cada una de las tipologías de proyectos que se realizan.

A continuación se puede ver la estructura que debe tener este documento y la información que debe indicarse en cada uno de los puntos.

Durante la definición y explicación de la fase de planificación se han referenciado cada uno de los puntos de este documento, y se ha indicado en qué momento se debe rellenar cada uno de ellos.

Estructura

a. Identificador del documento

Se debe definir una nomenclatura para todos los documentos del *proceso de pruebas*. Cada uno de ellos debe tener un identificador único que permita conocer a qué proceso pertenece y qué tipo de documento es. También se debe definir una nomenclatura para las diferentes versiones del documento.

Un ejemplo de nomenclatura podría ser:

PL = Documento de tipo *plan de pruebas*

PR-XXX= identificador del proyecto

V.X.X=versión del documento

b. Introducción

En este apartado se describe información genérica sobre el proyecto como, por ejemplo, cuál es la intención del *plan de pruebas* o la información relacionada con el proyecto.

(a) Propósito

Este punto permite explicar la intención de este documento y de la información que se puede consultar en él. Puede ser un texto predefinido muy similar para todos los procesos de pruebas.

(b) Alcance

También describe a groso modo todo aquello que se validará. Se realizará un breve resumen de los niveles de las pruebas con los que trabajaremos, qué tipologías de pruebas se realizarán y qué características se validarán dentro del proceso. Este punto puede ser bastante genérico, ya en una misma empresa se realiza generalmente la misma tipología de pruebas porque las aplicaciones acostumbran a ser muy similares.

(c) Audiencia

Se debe indicar a qué público está dirigido este documento.

(d) Definiciones, Acrónimos y Terminología

Aquí se define y especifica información de gran interés para las partes implicadas en el proyecto, como explicación de las tipologías de pruebas que se realizarán.

(e) Referencias

Se puede hacer referencia a otros documentos que forman parte del proyecto, como:

- ERS
- Plan de riesgos
- Metodología usada en proyecto
- Plan de calidad
- Plan de desarrollo
- Etc.

(f) Fondo

Breve descripción del proyecto asociado al *plan de pruebas* y cuál es su intención. Se puede indicar el objetivo y la motivación para aportar más información al equipo de pruebas

(g) Misión de evaluación

En este punto se deben indicar cuáles son los objetivos de este *plan de pruebas*. Es decir, si lo que se busca es:

- Mostrar defectos
- Ajustar en los requerimientos
- Ajustar en el propósito
- Medir atributos
- Medir calidad

(h) Motivación para realizar las pruebas

Explicación de cuál es la razón por la que se realizan estas pruebas. Será un texto bastante genérico, dependiendo de la empresa y de su tipología de software.

c. Test ítems

Listado de todos los artículos u objetos que formarán parte de las pruebas. Estos pueden ser partes del software, documentos, dispositivos de almacenaje u otros artefactos relacionados con el proyecto.

d. Esquema de inclusión de pruebas

Descripción a alto nivel de aquellas pruebas que se incluirán y se realizarán en el proceso. Se puede describir a qué nivel se realizarán, qué tipología y qué funcionalidades. Es recomendable indicar las razones por las cuales estas pruebas están incluidas en el proceso.

También se puede indicar aquellas pruebas que inicialmente no se realizarán, pero que pueden ser incluidas en algunos casos.

e. Esquema de exclusión de pruebas

Descripción a alto nivel de aquellas pruebas que quedarán excluidas del proceso. Es importante indicar adecuadamente las razones por las cuales no serán incluidas.

f. Enfoque

Explicación de lo que se hará durante el *proceso de pruebas*, descripción a alto nivel de las actividades que ser realizará, y cómo se realizarán.

(a) Metodología

Descripción de la metodología y el proceso que se seguirá para alcanzar los objetivos.

(b) Escenarios

También es importante indicar los diferentes escenarios donde se ejecutarán las pruebas.

Nombre del escenario	Flujo Originario	Flujo alternativo

TABLA 16 RESUMEN DE ESCENARIOS

(c) Técnicas y tipos de pruebas

Explicación de cada una de las técnicas que se aplicarán. Indicando cuál es la intención de las pruebas y qué se intenta validar.

Ejemplo de tabla resumen con la información necesaria para cada una de las técnicas:

Objetivo de la técnica
Técnica
Estrategias
Herramientas requeridas
Criterios de salida
Consideraciones especiales

TABLA 17 RESUMEN DE TÉCNICAS Y TIPOS DE PRUEBAS

g. Criterios de entrada y salida

(a) Criterios de entrada al *plan de pruebas*

Generalmente se entrará en el *plan de pruebas* cuando se haya finalizado en su totalidad el proceso de análisis y diseño de las pruebas

(b) Criterios de salida del *plan de pruebas*

Se procederá cuando se hayan ejecutado todas las pruebas

(c) Criterios de suspensión o ruptura del plan de prueba

El criterio de suspensión se basará normalmente en un porcentaje de error, que indicará cuándo se debe abortar el *plan de pruebas.*

Criterios del ciclo de pruebas: si se trabaja con una metodología incremental, se definirán ciclos para las pruebas y para la implementación. Es importante establecer criterios para cada uno de los ciclos, y evitar así que la fase de pruebas de un incremento se haga demasiado larga.

(d) Criterios de entrada del ciclo de pruebas

Cuando esté finalizada todo la documentación necesaria para empezar la ejecución de las pruebas.

(e) Criterios de salida del ciclo de pruebas

Cuando hayan quedado registradas todas las ejecuciones de los casos de prueba necesarios.

(f) Criterios de suspensión o ruptura del ciclo de pruebas

El criterio de suspensión del ciclo de pruebas será un porcentaje. Generalmente se revisarán cuando el porcentaje de aciertos sea muy alto, lo que puede indicar que las pruebas no son las adecuadas.

h. Entregables

Los que se obtendrán como resultado del *proceso de pruebas*. Algunos ejemplos podrían ser:

- ***Test plan***
- ***Test design specification***
- ***Test case specification***
- ***Test procedure specification***
- ***Test ítem transmittal report***
- ***Test log***
- ***Test incident report***
- ***Test summary report***
- ***Testware***

i. Tareas

Explicación de las tareas que se realizarán durante el proceso.

Nombre de la tarea	Descripción	Tiempos

TABLA 18 RESUMEN DE TAREAS

j. Entorno

(a) Bases HW del sistema

Recurso	Cantidad	Tipo o descripción

TABLA 19 RESUMEN DE LAS BASES DEL HW

(b) Elementos de software

Nombre	Versión	Tipo o descripción

TABLA 20 RESUMEN DE LOS ELEMENTOS SOFTWARE

(c) Configuración del sistema

Nombre de la configuración	Descripción	Configuración implementada

TABLA 21 RESUMEN DE LA CONFIGURACIÓN DEL SISTEMA

k. Responsabilidades

Descripción de los roles necesarios para ejecutar el *plan de pruebas*

Nombre del rol	Descripción

TABLA 22 RESUMEN DE RESPONSABILIDADES

l. Personal y competencias

Relación entre los roles y las necesidades del proyecto

Nombre del rol	Núm. Mínimo de recursos	Responsabilidades específicas

TABLA 23 RESUMEN DE PERSONAL Y COMPETENCIAS

m. Programa

Relación entre las tareas y los recursos. Se puede aplicar la tabla anterior

Nombre de la tarea	Descripción	Fecha inicio	Fecha fin	Recursos

TABLA 24 RESUMEN DEL PROGRAMA

n. Riesgos y planes de contingencia

Resumen de los riesgos, suposiciones y restricciones más relevantes del *plan de pruebas*.

(a) Riesgos

Riesgo	Estrategia para controlarlo	Contingencia

TABLA 25 RESUMEN DE LOS RIESGOS

(b) Suposiciones

Suposición a verificar	Impacto si la suposición falla	Responsable

TABLA 26 RESUMEN DE LAS SUPOSICIONES

(c) Restricciones

Restricción sobre	Impacto de la restricción	Responsable

TABLA 27 RESUMEN SOBRE LAS RESTRICCIONES

o. Aprobación

Punto que deben firmar todas las personas implicadas en el proyecto, indicando que aceptan el *plan de pruebas*

4.2. Plan de pruebas: Ejemplo

Historial de revisiones

Fecha	Versión	Descripción	Autor
20/07/2010	1.0	Primera versión del *plan de pruebas* para PR-XXX	*Creador del documento*
16/08/2010	1.1	Versión revisada del *plan de pruebas* para PR-XXX	*Persona encargada de revisarlo*

TABLA 28 EJEMPLO DEL HISTORIAL DE REVISIONES

a. Identificador del documento

PL-PR-XXX = *Plan de pruebas* para el proyecto PR-XXX basado en las especificaciones documentadas en ERS-PR-XXX

b. Introducción

(a) Propósito

- Identificar las funcionalidades que serán sometidas al *proceso de pruebas*.
- Identificar la motivación para realizar las pruebas en determinadas áreas del sistema.
- Definir cómo se van a realizar las pruebas.
- Identificar las herramientas necesarias para realizar las pruebas de manera apropiada.
- Identificar los recursos necesarios para las tareas del *proceso de pruebas*.

(b) Alcance

Las pruebas se realizarán a nivel de integración y de sistema. Se realizarán pruebas de usabilidad y fiabilidad, para verificar que el sistema es de uso fácil y para garantizar que se manejan de forma segura y confiable los datos del sistema. También se realizarán pruebas de carga y de estrés, para verificar la eficiencia del sistema. Si es necesario, se pueden realizar pruebas de integridad.

(c) Audiencia

Este documento está dirigido al equipo de desarrolladores, para su uso durante el periodo/fase de pruebas. Debe consultarse durante la definición de las pruebas, y deben conocerlo todas las personas implicadas en el proceso de implementación. El cliente y los usuarios pueden consultarlo, si lo desean, para verificar los requerimientos.

(d) Definiciones, Acrónimos y Terminología

Usabilidad: es la capacidad de un sistema de ser entendible, aprendible y atractivo para un usuario, bajo unas condiciones específicas.

Fiabilidad: es la capacidad del software para mantener un específico nivel de rendimiento bajo unas condiciones específicas.

Pruebas de carga: determinan el funcionamiento de un sistema probado bajo distintas condiciones operacionales, tales como: el número de usuarios, el número de transacciones, un tiempo determinado, etc., bajo una configuración concreta del sistema.

Pruebas de estrés: determina el funcionamiento de lo que se está probando bajo unas condiciones extremas o poco habituales.

Integridad: determina la fiabilidad, la robustez y la resistencia de los bancos de pruebas en relación a los datos que contendrá el sistema.

Funcionalidad: capacidad del software para proveer funciones que satisfacen las necesidades expresadas o implícitas, cuando un sistema está bajo unas condiciones determinadas.

(e) Referencias

Para complementar este documento se pueden consultar otros documentados relacionados:

Documento	Versión	Fecha de la versión
ERS (ERS-PR-XXX)	2.0	15/07/2010
Plan de riesgos (PLR-PR-XXX)	1.2	12/07/2010
Plan de desarrollo del software	2.3	21/01/2009
Plan de calidad	1.5	01/04/2009

TABLA 29 EJEMPLO REFERENCIAS

(f) Fondo

El objetivo principal del proyecto "*Sin_Nombre*" es crear un sistema que permita a la empresa "*Empresa_Sin_Nombre*" tener nuevo canal más interactivo de comunicación con sus usuarios. Pero ello se propone implementar una aplicación web integrada dentro de su portal web, que abra nuevas vías de diálogo para sus clientes. Este espacio nuevo para los usuarios permitirá que opinen, voten, expresen sus necesidades y aporten a la empresa nueva información sobre sus gustos y preferencias. Se implementarán una serie de herramientas 2.0, diseñadas específicamente para ellos.

Es muy importante garantizar la seguridad en todos los procesos del sistema, ya que los usuarios podrán aportar información a casi todas las secciones de la aplicación. También se debe asegurar un fácil uso de la misma. Si los usuarios no son capaces de entender y manejar la aplicación de forma fácil, la herramienta será un fracaso.

Como se desconoce el número de usuarios que accederán al sistema, es importante realizar pruebas de eficiencia para un número elevado de participantes.

(g) Misión de evaluación

El objetivo básico de las pruebas es verificar que se cumplen las especificaciones del sistema y que éste implementa todas las funcionalidades descritas. Básicamente se quiere:

- Mostrar defectos: detectar errores en el código o en la ejecución del software.
- Ajustar en los requerimientos: garantizar que se cumplen los requerimientos del proyecto.
- Medir calidad: asegurar la calidad del producto que se está entregando.

(h) Motivación para realizar las pruebas

La motivación de estas pruebas es garantizar que se cumple un cierto porcentaje de aceptación del sistema y que se cumplen los requerimientos con el menor número de fallos posibles.

c. Test ítems

A continuación se indican aquellos objetos que serán probados:

- Diagramas de casos de uso
- Módulo de autenticación y registro
- Módulo de comentarios
- Módulo de votaciones
- Módulo de envío de recetas
- Módulo de envió de sugerencias
- Módulo de notificaciones
- Módulo de trazabilidad de las actividades
- Módulo de envío de correos electrónicos
- Intercambio de información entre sistema
- Base de datos

d. Esquema de inclusión de pruebas

Las pruebas que se incluirán en este proceso serán básicamente pruebas a nivel de componentes y de integración. Permitirán verificar el correcto funcionamiento de todos los casos de uso del sistema. Por eso se realizarán pruebas funcionales en los dos niveles.

También se realizarán a nivel de sistema: pruebas de carga que permitirán verificar el correcto funcionamiento bajo determinadas condiciones de uso; pruebas de tensión para verificar estabilidad bajo condiciones extremas; pruebas de seguridad para garantizar el acceso seguro a toda la información introducida dentro de la operación.

Aunque no son estrictamente necesarias, también es recomendable realizar pruebas de portabilidad y de mantenimiento, para evitar problemas futuros.

e. Esquema de exclusión de pruebas

Las pruebas de regresión quedarán excluidas del proceso, ya que se consideran demasiado costosas para un proyecto ad-hoc y de pequeña envergadura.

f. Enfoque

(a) Metodología

Se usará una metodología RUP definida por la propia empresa para implementar el sistema, por lo que las pruebas deberán adaptarse a ésta. Se diseñarán las pruebas de forma incremental y con iteraciones para adaptarse al análisis, diseño e implementación de la aplicación.

(b) Escenarios

Consultar documento ERS-PR-XXX. También se pueden consultar todos los casos de uso en este documento para poder definir los escenarios alternativos para cada prueba.

(c) Técnicas y tipos de pruebas

Explicación de cada una de las técnicas que se aplicarán, indicando cuál es la intención de la pruebas y qué se intenta validar con éstas.

(i) Pruebas de integridad de datos y Base de datos

Objetivo de la técnica	Demostrar que la BD no sufrirá errores y mantendrá la integridad de sus datos
Técnica	Se simularán datos erróneos en todas aquellas funciones que accedan a la base de datos (prueba de caja negra). Se verificará que todos los datos que se guarden son correctos y que no funcionan las técnicas de *SQL-Injection* y *Cross Site Scripting* en los formularios implementados
Estrategias	Es recomendable empezar validando aquellas *querys* más sencillas y aquellas que se usan más a menudo, ya que los errores serán muy similares en todas ellas. También se deben validar, si existen, aquellas que usen funciones complejas.
Herramientas requeridas	• Un programa que permita acceder directamente a la Base de datos, para ejecutar las *querys* directamente. • Todas las funciones del sistema que accedan a BD. • El sistema corriendo sobre el servidor o sobre un emulador. • Casos de prueba y las plantillas necesarias.
Criterios de salida	Para tener éxito en la prueba se deben soportar y manejar correctamente los datos erróneos, y debe ser impecable el uso de todas las funciones que accedan a la BD.
Consideraciones especiales	Es importante documentar todas las rutas y configuraciones del sistema necesarias para poder conectarse a la BD. También se debería crear un archivo aparte encriptado para que el sistema consulte toda esta información, evitando tocar el sistema si se modifica una ruta o configuración.

TABLA 30 EJEMPLO DE TÉCNICAS DE INTEGRIDAD

(ii) Pruebas funcionales

Objetivo de la técnica	Analizar la funcionalidad del sistema. Es importante evaluar navegación, entradas y salidas, y el comportamiento del mismo
Técnica	Se deben consultar todos los casos de uso del sistema y crear varios casos de prueba para cada uno de ellos. Estos casos

	validarán el flujo básico del caso de la prueba, y todos los flujos alternativos que se hayan documentado. Se validará: • Que las funciones devuelven los resultados correctos para los datos correctos (caja negra) • Que los mensajes de error son correctos al introducir información errónea • Que el sistema realiza todas las funciones indicadas por el cliente, ni una más ni una menos.
Estrategias	Primero se ejecutará un caso de prueba para cada uno de los casos de uso del sistema, validando que se cumplen todos los requerimientos del sistema. Cuando se esté seguro de que el sistema se ajusta a las necesidades del usuario, se ejecutarán los casos de prueba de los flujos alternativos.
Herramientas requeridas	• El sistema corriendo sobre un servidor. • Un ordenador con internet para probar todas las funcionalidades desde el navegador. • Los casos de prueba y los casos de uso documentados para la aplicación.
Criterios de salida	• Que están implementados todos los casos de uso de forma correcta • Que todos los flujos alternativos están controlados y no generan fallos en el sistema.
Consideraciones especiales	La persona que ejecuta estas pruebas debe tener total conocimiento sobre el sistema, sus necesidades y sus requerimientos.

TABLA 31 EJEMPLO DE PRUEBAS FUNCIONALES

(iii) Pruebas de Interfaz con el usuario

Es importante validar que el usuario entiende y sabe usar a la perfección la aplicación. Para validarlo se pueden realizar una serie de pruebas con usuarios inexpertos, donde tengan que ser capaces de finalizar todos los procesos del sistema consultando simplemente las ayudas o documentación que recibirá el usuario final.

(iv) Pruebas de rendimiento

Objetivo de la técnica	Validar que si se realizan muchas consultas a la BD, o se llevan a cabo cambios en la misma, las ejecuciones tardan tiempos aceptables y no se producen retrasos perceptibles

	por parte de los usuarios.
Técnica	Abrir varias sesiones en varias máquinas y realizar consultas que requieran procesar mucha información.
Estrategias	Realizar una consulta desde una máquina y observar el tiempo de respuesta de ésta. Luego comparar con el máximo definido en las especificaciones. Si éste se cumple correctamente, realizar la misma consulta con varias máquinas a la vez y volver a verificar los tiempos observados.
Herramientas requeridas	• Sistema corriendo sobre el servidor. • Aplicación para medir los tiempos de la BD. • Varios ordenadores con navegador. • Documentación sobre los resultados esperados.
Criterios de salida	Para tener una prueba exitosa los tiempos, tanto con una sesión como con varias, deben ser similares y nunca superiores al máximo especificado.
Consideraciones especiales	Crear un script que pueda simular las sesiones de varios usuarios.

TABLA 32 EJEMPLO DE PRUEBAS DE RENDIMIENTO

(v) Pruebas de carga

Objetivo de la técnica	Validar que el tiempo de respuesta sea aceptable en un intervalo y que las consultas a la BD sean coherentes y específicas.
Técnica	Generar tráfico desde varias sesiones dentro de la aplicación, y que estas pidan datos a la base de datos y al servidor.
Estrategias	Enviar una consulta desde una sesión y validar que la respuesta sea la esperada. Una vez validada la información recibida, realizar la misma consulta desde varias sesiones a la vez y validar que los datos que se reciben son los correctos en todas ellas.
Herramientas requeridas	• Sistema corriendo sobre servidor real • Script para emular las sesiones • Varios ordenadores con navegador
Criterios de salida	Todas las consultas deben realizarse dentro del tiempo establecido, y debe mostrar toda la información

	correctamente.
Consideraciones especiales	Tener algún programa que permita monitorizar la BD

TABLA 33 EJEMPLO DE PRUEBAS DE CARGA

(vi) Pruebas de estrés

Objetivo de la técnica	Realizar ciertas pruebas en situaciones extremas para verificar cuándo el sistema falla. La intención es identificar: máxima capacidad de usuarios, múltiples usuarios realizando la misma transacción a la vez, sobrecarga de transacciones a la vez.
Técnica	Tener varios usuarios conectados y realizar todo tipo de transacciones sobre el sistema
Estrategias	Probar las funcionalidades que requieren mayor carga sobre el sistema con varios usuarios realizando el mayor número de transacciones posibles.
Herramientas requeridas	• Sistema corriendo sobre el servidor • Documentación del proyecto • Robots que realicen ciertas transacciones a la vez
Criterios de salida	Cuando se puedan documentar los niveles de tolerancia del sistema
Consideraciones especiales	Se deberán sincronizar varios clientes para realizar la misma funcionalidad al mismo tiempo.

TABLA 34 EJEMPLO DE PRUEBAS DE ESTRÉS

(vii) Pruebas de protección y control de acceso

Objetivo de la técnica	Validar que los usuarios sólo pueden acceder a aquellas zonas que tienen permisos para acceder
Técnica	Se debe verificar, en función del perfil de usuario que ha accedido al sistema, que solamente puede consultar y trabajar en zonas que su tipo de perfil le permita.
Estrategias	Se deben verificar todas las funcionalidades según el perfil del usuario. Se debe comprobar que no puede acceder a funcionalidades que no forman parte de su perfil. Cualquier irregularidad debe ser verificada para garantizar la

	integridad del sistema.
Herramientas requeridas	• Sistema corriendo sobre servidor real • Documentación de perfiles y casos de uso • Ordenador con un navegador
Criterios de salida	Los perfiles pueden realizar todos sus casos de uso y no pueden realizar casos de uso que no les pertenecen.
Consideraciones especiales	Los permisos deben ser validados por el cliente.

TABLA 35 EJEMPLO DE PRUEBAS DE CONTROL DE ACCESO

(viii) Pruebas de fiabilidad

Objetivo de la técnica	Verificar que tanto el SW como el HW son los apropiados para la aplicación
Técnica	Se deben abrir varias aplicaciones, además del sistema, de forma simultánea para validar su funcionamiento bajo condiciones de compartición de tiempos de procesador y de memoria.
Estrategias	Aumentar el número de usuarios que se conectan a las diversas aplicaciones, para verificar el funcionamiento bajo condiciones de alta concurrencia y solitudes de muchos usuarios
Herramientas requeridas	• El sistema corriendo sobre el servidor, con un navegador para poder realizar pruebas • Documentación sobre el sistema y el servidor
Criterios de salida	Que se cumplan todas las funcionalidades del sistema de forma correcta bajo estas condiciones
Consideraciones especiales	Si hay un exceso de proceso de datos en el servidor o se sobre-utilizan las máquinas, se puede producir errores importantes durante la ejecución del sistema

TABLA 36 EJEMPLO DE PRUEBAS DE FIABILIDAD

(ix) Pruebas de instalación

Objetivo de la técnica	Consiste en verificar que una vez terminado el sistema puede montarse de forma correcta en entorno real.
Técnica	Comprobar el correcto funcionamiento del sistema después de ser instalado en un entorno idéntico al entorno real.
Estrategias	Primero se comprueba que la instalación es la correcta; luego se verifica su comportamiento al realizar modificaciones y actualizaciones
Herramientas requeridas	• Sistema corriendo sobre el servidor. • Una herramienta para gestionar instalaciones
Criterios de salida	El sistema debe quedar correctamente instalado y no presentar problemas durante la ejecución.
Consideraciones especiales	Se debe contar con algún software o herramienta para instalación de sistemas.

TABLA 37 EJEMPLO DE PRUEBAS DE INSTALACIÓN

g. Criterios de entrada y salida

(a) Criterios de entrada al *plan de pruebas*

Para determinar cuándo se ejecutará el *plan de pruebas*, éste debe estar finalizado al 100% y se deben haber desarrollado la mayor parte de los casos de prueba.

(b) Criterios de salida del *plan de pruebas*

Cuando se hayan realizado al 100% las pruebas del sistema.

(c) Criterios de suspensión o ruptura del plan de prueba

Si mas del 30% de los casos ejecutados no son aceptables se deberá suspender el *plan de pruebas*.

(d) Criterios de entrada del ciclo de pruebas

Los casos de prueba se realizarán en función de los casos de uso implementados. Estos deben estar documentados al 100%, y deben estar diseñados todos los escenarios necesarios para estos casos de pruebas. Hay que tener disponibles todos los casos de pruebas para ese ciclo.

(e) Criterios de salida del ciclo de pruebas

Cuando hayan finalizados todas las pruebas de ese ciclo.

(f) Criterios de suspensión o ruptura del ciclo de pruebas

Si han pasado un 30% de las pruebas del sistema sin detectar errores, éstas deberán ser depuradas. Si se llega a un 25% de error en las pruebas también deberán revisarse.

h. Entregables

La documentación que se entregará en el *proceso de pruebas* será la siguiente:

- ***Test plan***
- ***Test design specification***
- ***Test case specification***
- ***Test procedure specification***
- ***Test ítem transmittal report***
- ***Test log***
- ***Test incident report***
- ***Test summary report***
- ***Testware***

Estos documentos, además de la información necesaria para diseñar y realizar las pruebas, deben documentar (si se aplica para ese proyecto):

- El resumen de la evaluación de las pruebas: porcentaje de aceptación de las funcionalidades del sistema
- Cobertura de las pruebas
- Calidad percibida
- Registro de incidencia
- Registro de cambios
- Pruebas de regresión
- Scripts de soporte
- Productos adicionales

i. Tareas

Tareas que se realizarán durante el proceso. Se apuntan estimaciones de tiempo, pero pueden variar en función del diseño de las pruebas. El proceso de implementación y ejecución no se puede determinar hasta tener todas las pruebas definidas.

Total		**93 días**
Análisis y diseño		**15 días**
	Revisar la base de las pruebas	3 días
	Identificar las condiciones de las pruebas	2 días
	Diseñar las pruebas	5 días
	Evaluar las pruebas vs los requerimientos	2 días
	Revisar las herramientas	3 días
Implementación		**23 días**
	Definir conjuntos de pruebas	10 días
	Preparar el material necesario	3 días
	Implementar y verificar el entorno	10 días
Ejecución		**38 días**
	Ejecutar las pruebas	20 días
	Registrar los resultados	20 días
	Comparar los resultados	20 días
	Informar de las diferencias	3 días
	Repetir las actividades	15 días
Evaluación		**10 días**
	Comprobar registros de prueba contra los criterios de salida	3 días
	Evaluar	2 días
	Escribir un informe resumen	5 días
Control		**90 días**

	Medir y analizar	25 días
	Monitorizar y documentar	65 días
	Proporcionar información	65 días
	Iniciar acciones correctivas	65 días
	Tomar decisiones	65 días
Cierre		**3 días**
	Comprobar entregables	1 días
	Finalizar y archivar *testware*	0,5 días
	Entregar *testware*	0,5 días
	Evaluar el proceso completo	1 día

TABLA 38 EJEMPLO DE TAREAS

j. Entorno

(a) Bases HW del sistema

Recurso	Cantidad	Tipo o descripción
Servidor	1	
Servidor de aplicaciones	1	Tomcat
Computador de prueba	1	Pentium 4
Gestor de base de datos	1	Oracle

TABLA 39 EJEMPLO DE BASES PARA EL HW

(b) Elementos de software

A continuación se muestran los elementos software básico para la implementación:

Nombre	Versión	Tipo o descripción
Windows	XP, Vista, 7	Sistema operativo
Linux	Debian, redhat	Sistema operativo
Internet Explorer	6.0 ó superior	Cliente web
Firefox	2.0 ó superior	Cliente web
Oracle	10 ó superior	Gestor de base de datos

TABLA 40EJEMPLO DE ELEMENTOS SOFTWARE

(c) Configuración del sistema

A continuación se muestran las configuraciones básicas del sistema:

Nombre de la configuración	Descripción	Configuración implementada
Perfil del sistema	Acceso a todas las herramientas necesarias para realizar las pruebas	Se deben instalar todas las herramientas necesarias para realizar las pruebas
Red	Proveedor de servicios de internet	Red interna o conexión a internet

TABLA 41 EJEMPLO DE CONFIGURACIÓN DEL SISTEMA

k. Responsabilidades

Descripción de los roles necesarios para ejecutar el *plan de pruebas*

Nombre del rol	Descripción
Jefe de calidad	Tiene una vista general de las pruebas y de las funcionalidades
Analista de pruebas	Identifica y define las pruebas especificas que se deben realizar
Diseñador de pruebas	Define el enfoque técnico para la implementación de los esfuerzos de las pruebas
Probador	Ejecuta las pruebas
Administrador de sistemas	Se encarga de que los ambientes de prueba estén configurados correctamente y sean los adecuados
Administrador de bases de datos	Se encarga de que la base de datos y el ambiente estén administrados y mantenidos correctamente
Diseñador	Identifica y define las operaciones, atributos y asociaciones de las clases implementadas para las pruebas
Implementador	Implementa y unifica las pruebas.

TABLA 42 EJEMPLO DE RESPONSABILIDADES

l. Personal y competencias

Relación entre los roles y las necesidades del proyecto

Nombre del rol	Núm.	Responsabilidades especificas
Jefe de calidad	1	• Planificar • Aceptar las pruebas • Identificar los objetivos • Decidir los recursos apropiados • Presentar todos los reportes • Evaluar la efectividad de las pruebas • Gestionar y seguir la evolución del proceso
Analista de	1	• Ayudar a identificar los objetivos de las pruebas • Definir los detalles de las pruebas

pruebas		• Definir los resultados de las pruebas • Documentar peticiones de cambios • Evaluar la calidad de los resultados y del producto • Definir la información de los reportes Si no hay diseñador de pruebas, también se encargará de: • Definir el enfoque de las pruebas • Definir las automatizaciones de las pruebas • Verificar las técnicas que se van a utilizar • Definir los elementos a probar • Estructurar la implementación de las pruebas
Probador	2	• Implementar las pruebas • Ejecutar las pruebas • Registrar los resultados • Analizar los fallos • Documentar los incidentes
Administrador de sistemas	1	• Administrar el sistema de pruebas • Instalar todas las herramientas necesarias para el proceso • Configurar el entorno Si no hay Administrador de bases de datos, también se encargará de: • Realizar el soporte de las bases de datos
Diseñador	1	• Define las clases de prueba requeridas para soportar los requerimientos definidos por el equipo de pruebas
Implementador	2	• Crea los componentes de pruebas necesarios para soportar los requerimientos de las mismas.

TABLA 43 EJEMPLO DE PERSONAL Y COMPETENCIAS

m. Programa

Relación entre las tareas y los recursos. Se pueden aplicar sobre la tabla anterior

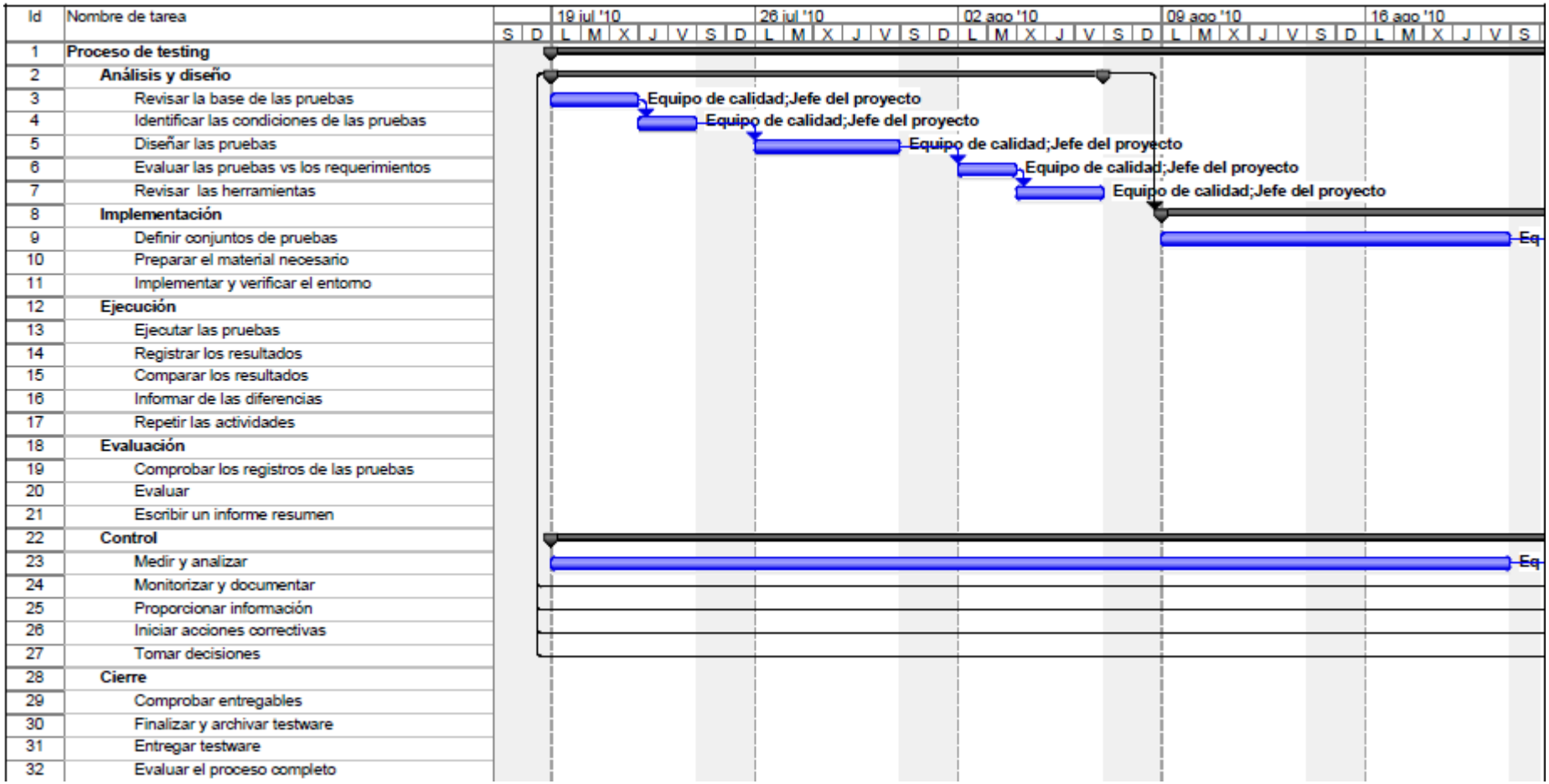
Tarea		Duración	Comienzo	Fin	Nombres de los recursos
Total		**93 días**	**19/07/2010 9:00**	**10/09/2010 19:00**	
Análisis y diseño		**15 días**	**04/08/2010 15:00**	**25/08/2010 13:00**	
	Revisar la base de las pruebas	3 días	04/08/2010 15:00	09/08/2010 13:00	Jefe del proyecto, Equipo de calidad
	Identificar las condiciones de las pruebas	2 días	09/08/2010 15:00	11/08/2010 13:00	Jefe del proyecto, Equipo de calidad
	Diseñar las pruebas	5 días	11/08/2010 15:00	18/08/2010 13:00	Jefe del proyecto, Equipo de calidad
	Evaluar las pruebas vs los requerimientos	2 días	18/08/2010 15:00	20/08/2010 13:00	Jefe del proyecto, Equipo de calidad
	Revisar las herramientas	3 días	20/08/2010 15:00	25/08/2010 13:00	Jefe del proyecto, Equipo de calidad
Implementación		**23 días**	**25/08/2010 13:00**	**25/08/2010 13:00**	
	Definir conjuntos de pruebas	10 días	25/08/2010 13:00	25/08/2010 13:00	Equipo de proyectos; Equipo de

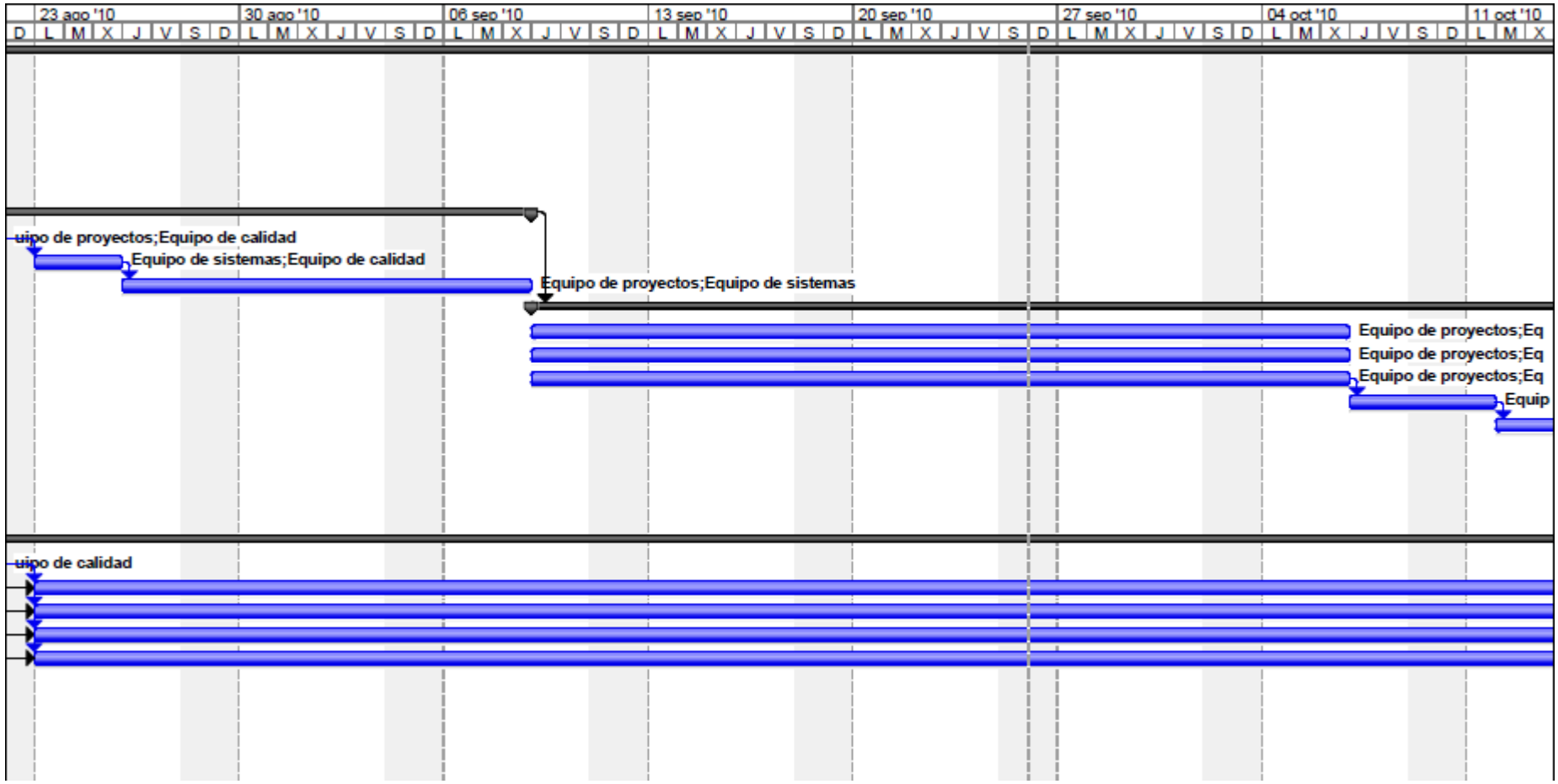
					calidad
	Preparar el material necesario	3 días	25/08/2010 13:00	25/08/2010 13:00	Equipo de sistemas; Equipo de calidad
	Implementar y verificar el entorno	10 días	25/08/2010 13:00	25/08/2010 13:00	Equipo de proyectos; Equipo de sistemas; Equipo de calidad
Ejecución		**38 días**	**25/08/2010 13:00**	**25/08/2010 13:00**	
	Ejecutar las pruebas	20 días	25/08/2010 13:00	25/08/2010 13:00	Equipo de proyectos; Equipo de calidad
	Registrar los resultados	20 días	25/08/2010 13:00	25/08/2010 13:00	Equipo de proyectos; Equipo de calidad
	Comparar los resultados	20 días	25/08/2010 13:00	25/08/2010 13:00	Equipo de proyectos; Equipo de calidad
	Informar de las diferencias	3 días	25/08/2010 13:00	25/08/2010 13:00	Equipo de proyectos; Equipo de calidad
	Repetir las actividades	15 días	25/08/2010 13:00	25/08/2010 13:00	Equipo de proyectos; Equipo de calidad
Evaluación		**10 días**	**25/08/2010 15:00**	**08/09/2010 13:00**	
	Comprobar registros de prueba contra los criterios de salida	3 días	25/08/2010 15:00	30/08/2010 13:00	Equipo del proyecto; Equipo de calidad

	Evaluar	2 días	30/08/2010 15:00	01/09/2010 13:00	Jefe del proyecto, Equipo de calidad
	Escribir un informe resumen	5 días	01/09/2010 15:00	08/09/2010 13:00	Jefe del proyecto, Equipo de calidad
Control		**90 días**	**19/07/2010 9:00**	**04/08/2010 15:00**	
	Medir y analizar	25 días	19/07/2010 9:00	04/08/2010 13:00	Equipo del proyecto; Equipo de calidad
	Monitorizar y documentar	65 días	04/08/2010 15:00	04/08/2010 15:00	Jefe del proyecto, Equipo de calidad
	Proporcionar información	65 días	04/08/2010 15:00	04/08/2010 15:00	Jefe del proyecto, Equipo de calidad
	Iniciar acciones correctivas	65 días	04/08/2010 15:00	04/08/2010 15:00	Jefe del proyecto, Equipo de calidad
	Tomar decisiones	65 días	04/08/2010 15:00	04/08/2010 15:00	Jefe del proyecto, Equipo de calidad
Cierre		3 días	08/09/2010 15:00	10/09/2010 19:00	
	Comprobar entregables	1 días	08/09/2010 15:00	08/09/2010 19:00	Jefe del proyecto, Equipo de calidad
	Finalizar y archivar *testware*	0,5 días	09/09/2010 9:00	09/09/2010 13:00	Jefe del proyecto, Equipo de

					calidad
	Entregar *testware*	0.5 días	09/09/2010 15:00	09/09/2010 19:00	Jefe del proyecto, Equipo de calidad
	Evaluar el proceso completo	1 día	10/09/2010 9:00	10/09/2010 19:00	Jefe del proyecto, Equipo de calidad

TABLA 44 EJEMPLO DE PROGRAMA


Id
Nombre de tarea
19 jul '10
26 jul '10
02 ago '10
09 ago '10
16 ago '10
S D L M X J V S D L M X J V S D L M X J V S D L M X J V S D L M X J V S
1 Proceso de testing
2 Análisis y diseño
3 Revisar la base de las pruebas
4 Identificar las condiciones de las pruebas
5 Diseñar las pruebas
6 Evaluar las pruebas vs los requerimientos
7 Revisar las herramientas
8 Implementación
9 Definir conjuntos de pruebas
10 Preparar el material necesario
11 Implementar y verificar el entorno
12 Ejecución
13 Ejecutar las pruebas
14 Registrar los resultados
15 Comparar los resultados
16 Informar de las diferencias
17 Repetir las actividades
18 Evaluación
19 Comprobar los registros de las pruebas
20 Evaluar
21 Escribir un informe resumen
22 Control
23 Medir y analizar
24 Monitorizar y documentar
25 Proporcionar información
26 Iniciar acciones correctivas
27 Tomar decisiones
28 Cierre
29 Comprobar entregables
30 Finalizar y archivar testware
31 Entregar testware
32 Evaluar el proceso completo
Equipo de calidad;Jefe del proyecto
Equipo de calidad;Jefe del proyecto
Equipo de calidad;Jefe del proyecto
Equipo de calidad;Jefe del proyecto
Equipo de calidad;Jefe del proyecto
Eq
Eq


23 ago '10
30 ago '10
06 sep '10
13 sep '10
20 sep '10
27 sep '10
04 oct '10
11 oct '10
D L M X J V S D L M X J V S D L M X J V S D L M X J V S D L M X J V S D L M X J V S D L M X J V S D L M X
uipo de proyectos;Equipo de calidad
Equipo de sistemas;Equipo de calidad
Equipo de proyectos;Equipo de sistemas
Equipo de proyectos;Eq
Equipo de proyectos;Eq
Equipo de proyectos;Eq
Equip
uipo de calidad

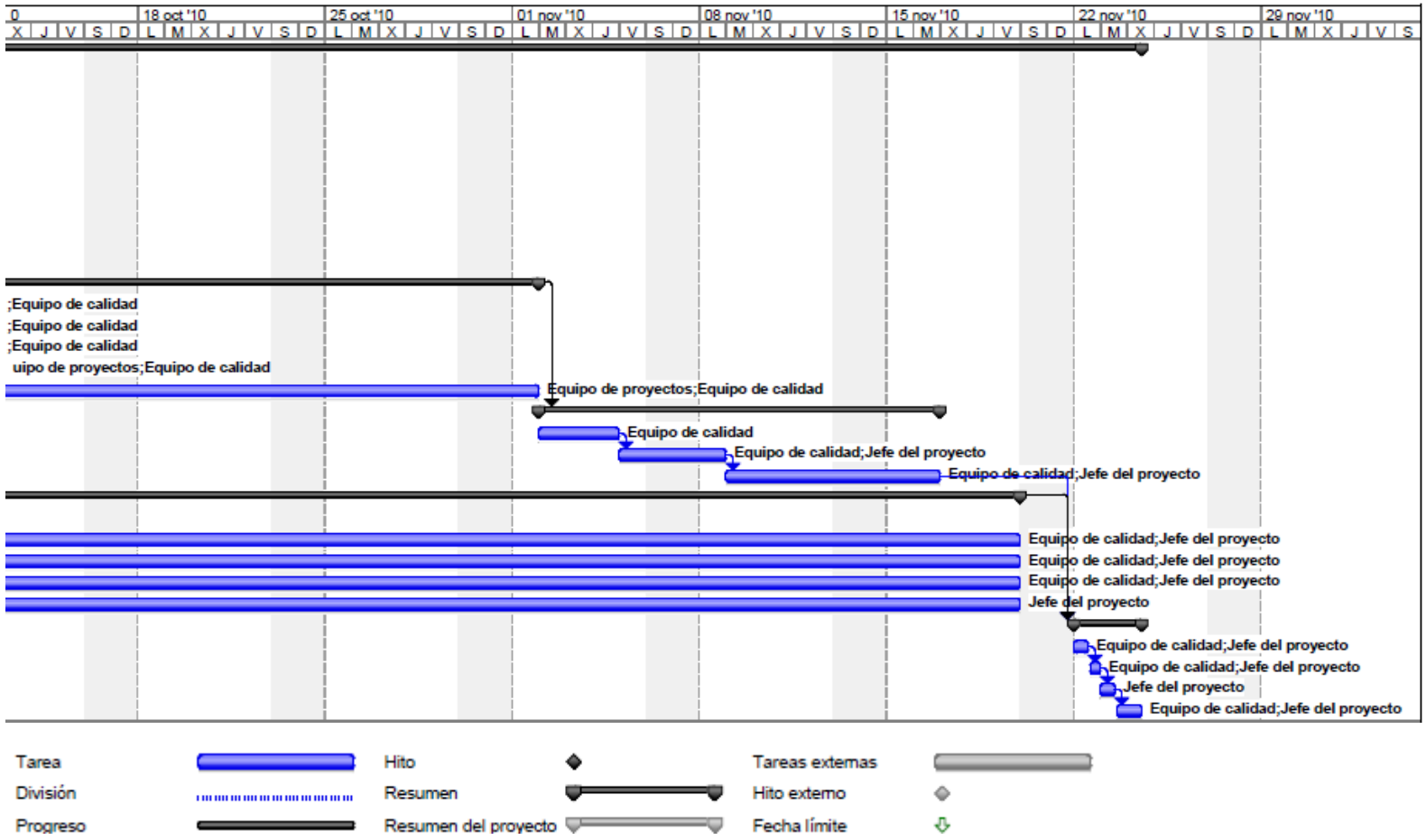


ILUSTRACIÓN 30 EJEMPLO DE PROGRAMA

n. Riesgos y planes de contingencia

Resumen de los riesgos, suposiciones y restricciones más relevantes del *plan de pruebas*.

(a) Riesgos

Riesgo	Estrategia para controlarlo	Contingencia
Número alto de casos de prueba	Ordenarlos por prioridades, basándose en los riesgos del proyecto	Revisar los casos de prueba y eliminar aquellos que tienen prioridad baja
Inexperiencia de los recursos en las herramientas de pruebas	Escoger herramientas sencillas y con mucha documentación accesible para las personas que deberán usarla	Revisar las pruebas para realizarlas de forma manual.
Reducción de los recursos	Recalcular los tiempos en función de los recursos destinados	Reasignar las tareas a otros recursos ya asignados al proyecto que tengan el conocimiento suficiente para realizarlas
Reducción del tiempo	Revisar todas las pruebas, calcular prioridades	Realizar solamente aquellas pruebas que validen procesos críticos

TABLA 45 EJEMPLO DE RIESGOS

(b) Suposiciones

Suposición a verificar	Impacto si la suposición falla	Responsable
Los tiempos definidos son aceptables	El proceso no se podrá finalizar	Jefe del proyecto
Los tiempos necesarios para aprender a usar las herramientas son aceptables	El proceso no se finalizará	Jefe del proyecto
Los casos de uso están completo	No se probarán todas las funcionalidades	Jefe del proyecto

Los requerimientos están bien definidos	No se probarán correctamente los requerimientos	Jefe del proyecto
El gestor de base de datos es el correcto	Tiempos de respuesta incorrectos, resultado ambiguo	Jefe del proyecto

TABLA 46 EJEMPLO DE SUPOSICIONES

(c) Restricciones

Restricción sobre	Impacto de la restricción	Responsable
Servidor de aplicaciones	No se podrán realizar las pruebas correctamente	Administrador del sistema
Herramientas para realizar las pruebas	No se podrán realizar las pruebas correctamente	Administrador del sistema

TABLA 47 EJEMPLO DE RESTRICCIONES

o. Aprobación

Punto que deben firmar todas las personas implicadas en el proyecto, indicando que aceptan el *plan de pruebas*

4.3. Otros documentos interesantes

- ***Test design specification:*** este documento es resultado de la fase de diseño y análisis, especifica las condiciones de prueba para un ítem.
- ***Test case specification:*** es resultado de la fase de diseño y especificación, en este documento se puede consultar los valores de entrada, resultados esperados y demás información necesaria para ejecutar una prueba.
- ***Test procedure specification***: este documento es el resultado de la fase de implementación e indica la secuencia de acciones para la ejecución de una prueba.
- ***Test ítem transmittal report***: describe la estructura de documentos, datos e información del *proceso de pruebas*
- ***Test log*** es el registro cronológico de detalles relevantes durante la ejecución de las pruebas.
- ***Test incident report:*** es el documento que recopila cualquier evento ocurrido durante la ejecución y que requiera investigación.
- ***Test summary report***

5. La calidad del Software

Uno de los principales objetivos del *proceso de pruebas* es validar la *calidad del software*. Pero ¿qué significa eso exactamente?

A continuación explicaremos qué significa *calidad del software*, cómo podemos validarla y en qué momento debe validarse.

5.1. ¿Qué es la calidad?

Según la RAE la calidad es la propiedad o conjunto de propiedades inherentes a algo, que permiten juzgar su valor. Si concretamos, la *calidad del software* es el grado en que un componente, sistema o proceso satisface los requerimientos específicos y/o las necesidades y expectativas del usuario/cliente.

Para poder validad la *calidad del SW* es necesario definir primero cuáles son las características que debe cumplir y bajo qué condiciones.

Se deben fijar valores medibles que permitan validar que la calidad se está cumpliendo.

Según la tipología del proyecto será más importante garantizar unas características que otras.

5.2. ¿Cómo validar la calidad?

La ISO 9126 define que se deben cumplir las siguientes características para garantizar la calidad en todos los proyectos Software:

- **Funcionalidad (*funcionality*):** es la capacidad del SW de proveer funciones que satisfacen las necesidades expresadas o implícitas cuando está bajo unas condiciones determinadas. Para garantizar la funcionalidad se deben verificar:
    - Idoneidad (*suitability*): es la capacidad del *software* de proveer el conjunto de funciones apropiado para realizar unas tareas y unos objetivos concretos para los usuarios.
    - Exactitud (*accuracy*): es la capacidad del SW para proveer de aquello que es lo correcto, o cumplir los resultados o efectos con el grado de precisión necesario.
    - Interoperabilidad (*interoperability*): es la capacidad del SW de interactuar con uno o varios conjuntos del propio sistema, o de otros sistemas.

  - Seguridad (*security*): es la capacidad del SW de proteger información y datos, de forma que personas o sistemas no autorizados no pueda leer ni modificarlo, y las personas autorizadas sí puedan hacerlo.
- **Fiabilidad (*reliability*):** es la capacidad del SW para mantener un específico nivel de rendimiento bajo unas condiciones específicas. Para garantizar la fiabilidad se debe garantizar:
  - Madurez (*maturity*): es la capacidad del SW para recuperar un fallo como resultado de un defecto en sí mismo.
  - Tolerancia a fallo *(fault tolarence*): es la capacidad del SW para mantener un nivel específico de rendimiento en caso de fallo o infracción.
  - Recuperación (*recoverability*): es la capacidad para resestablecerse a un nivel concreto de rendimiento, y recuperar los datos directamente afectados en caso de error.
- **Usabilidad (*usability*):** es la capacidad del SW para ser entendible, aprendible, usado y atractivo para el usuario bajo unas condiciones específicas. Para garantizar la usabilidad se debe verificar:
  - Comprensibilidad (*understandability*): es la capacidad del SW para permitir al usuario entender si es el adecuado, y cómo se debe usar para tareas particulares y ciertas condiciones de uso.
  - Facilidad de aprendizaje (*learnability*): es la capacidad del SW para permitir al usuario aprender sus aplicaciones.
  - Operatividad (*operability*): es la capacidad del SW para permitir al usuario operar y controlar.
  - Atractivo (*attractiveness*): es la capacidad del SW de ser atractivo para el usuario.
- **Eficiencia (*efficiency*):** es la capacidad del SW para proveer el rendimiento relativo a la cantidad de recursos usados. Para garantizar la eficiencia se debe verificar:
  - Comportamiento en el tiempo (*time beahvior*): es la capacidad del SW para adecuar respuestas, tiempos de proceso y tasa de rendimiento cuando ejecuta sus funciones.
  - Utilización de recursos (*resource utilisation)*: es la capacidad del SW para el uso apropiado de cantidades y tipos de recursos cuando ejecuta sus funciones.
- **Mantenibilidad (*maintainability)*:** es la capacidad del SW para ser modificado. Las modificaciones pueden incluir correcciones, mejoras o adaptaciones del SW. Para garantizar la mantenibilidad se debe verificar:
  - Analizabilidad (*analysability*): es la capacidad del SW para ser diagnosticado por deficiencias o causas de fallo en el propio SW, o en las partes, para ser modificado o identificado.
  - Variabilidad (*changeability*): es la capacidad del SW para ser especificado en una modificación o en una implementación.
  - Estabilidad (*stability*): es la capacidad del SW para evitar efectos inesperados en modificaciones del propio SW.

    - Capacidad de prueba (*testability*): es la capacidad del SW de ser modificado y validado.
- **Portabilidad (*portability*):** es la capacidad del SW de ser transferido de un entorno a otro. Para garantizar la portabilidad se debe verificar:
    - Adaptabilidad (*adaptability*): es la capacidad del SW para adaptarse a diferentes entornos específicos sin aplicar acciones ni otros medios que aquellos para los cuales tiene propósito el SW.
    - Capacidad de instalación (*installability*): es la capacidad del SW para ser instalado en un entorno específico.
    - Co-existencia (*co-existence):* capacidad del SW para co-existir con otro SW independiente, compartiendo entorno y recursos.

5.3. Métricas

Se entiende por métrica, todo aquello que corresponda o está vinculado al metro o a su sistema de medida.

En software, y según la ISO-1925, las métricas definen un método de medición y una escala.

Las métricas pueden incluir métodos para categorizar datos cualitativos. Pueden ser internas o externas, y directas o indirectas.

Métricas internas: son una escala cuantitativa y un método de medición. Pueden ser utilizadas para la medición de un atributo o de una característica de un producto de software. Las principales características de las métricas internas son:

- Se aplican sobre un producto no ejecutado.
- Se aplican durante las etapas de diseño e implementación.
- Permiten medir la calidad de los entregables intermedios.
- Permiten predecir la calidad del producto final.
- Permiten al usuario iniciar acciones correctivas temprano en el ciclo de desarrollo.

Métricas externas: son una escala cuantitativa y un método de medición. Pueden ser utilizadas para la medición de un atributo o característica de un producto de software, derivado del comportamiento del sistema del que forma parte. Las principales características de las métricas externas son:

- Se aplican sobre un producto ejecutado.
- Se aplican durante las etapas de pruebas y las etapas posteriores al desarrollo.
- Permiten medir la calidad de los entregables intermedios y completos.
- Permiten validar la calidad del producto final.
- Permiten al usuario realizar acciones correctivas.

Métricas directas: son aquellas que se realizan aplicando un aparato o sistema informático para medir una magnitud. Por ejemplo:

- Coste
- Esfuerzo
- Líneas de Código
- Velocidad de Ejecución
- Memoria
- Número de Errores

Métricas indirectas: calculan el valor de la medida mediante una fórmula (expresión matemática), previo cálculo de las magnitudes que intervienen en la misma por métricas directas. Por ejemplo:

- Funcionalidad
- Complejidad
- Eficiencia
- Fiabilidad
- Manteanibilidad

5.4. Atributos básicos para garantizar la calidad

De todos los atributos que se deben verificar para garantizar la *calidad del software*, hay un grupo que son condición indispensable en todos los proyectos. Son los atributos relacionados con la funcionalidad. A continuación se explicará de forma detallada cada uno de ellos, cómo verificarlos y en qué casos tienen especial importancia.

5.4.1. Funcionalidad (*Funcionality*)

Definición

La funcionalidad es la capacidad del SW de proveer funciones que satisfacen las necesidades expresadas o implícitas cuando un SW esta bajo unas condiciones determinadas.

Pretende valorar el conjunto de capacidades del programa, la generalidad de las funciones y la seguridad global.

En qué casos se utiliza

Según la ISO 9126 cualquier sistema debe cumplir las propiedades relacionadas con la funcionalidad: Idoneidad (*Suitability*), Exactitud (*Accuracy*), Interoperabilidad (*Interoperability*), Seguridad (*Security*)

Pasos a seguir

A continuación mostraremos un diagrama con la relación entre las actividades necesarias para validar el Sotfware y las actividades del ciclo de vida del software.

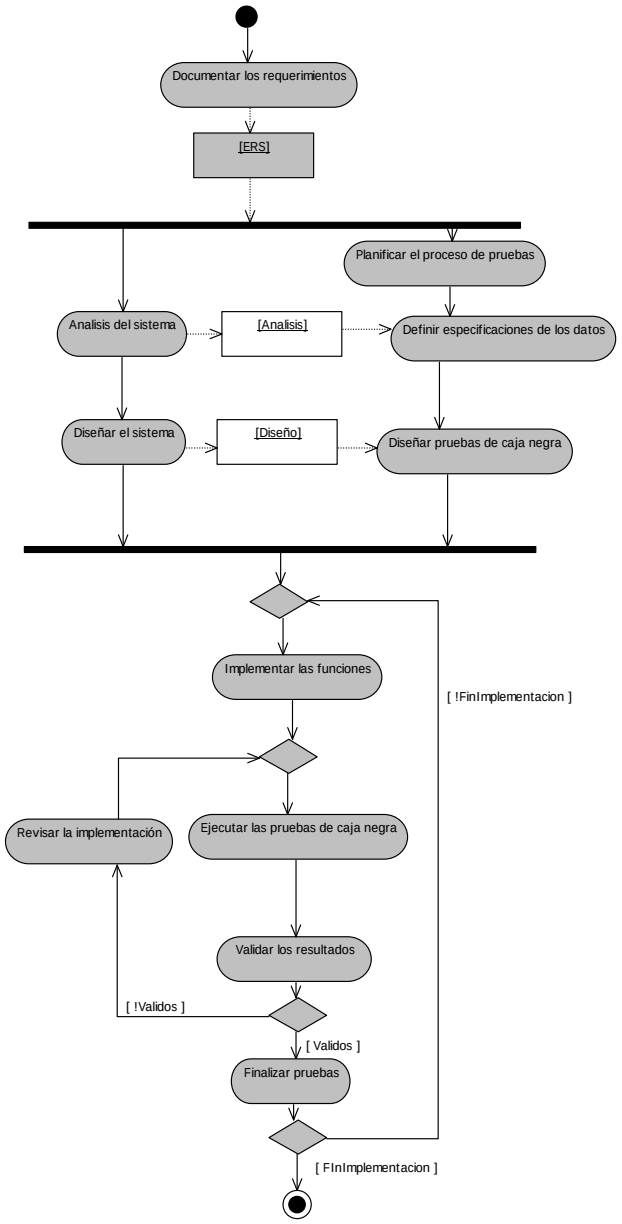


ILUSTRACIÓN 31 DIAGRAMA DE ACTIVIDADES PARA VALIDAR LA FUNCIONALIDAD

5.4.2. Idoneidad (*Suitability*)

Definición

La idoneidad es la capacidad del *software* de proveer el conjunto de funciones apropiado para realizar unas tareas y unos objetivos concretos para los usuarios.

Para medir la idoneidad de un sistema, se valida lo correcta que es una especificación y se comprueba si la implementación se adapta a éste.

Cómo documentar

Para poder validar la idoneidad necesitamos determinar cuáles son las funcionalidades que debe cumplir el sistema. Esta información se puede especificar de la siguiente forma:

- Utilizando casos de usos
- Generado un listado con todas las funcionalidades
- Dibujando *story boards*

Técnicas relacionadas

Para comprobar la idoneidad de un sistema se pueden utilizar técnicas dinámicas basadas en las especificaciones como:

- *Use case testing*
- *State transtion*
- *Decision tables*

Para este tipo de validaciones es necesario tener una buena documentación de las funcionalidades y de los cambios en el sistema, así que también es importante utilizar técnicas estáticas que permitan validar la documentación como:

- *Walkthorughts*
- *Tecnical reviews*
- *Inspection*

Este tipo de técnicas pueden aplicarse a nivel de componente, a nivel de integración y a nivel de sistema. Aunque realmente son pruebas que permitirán validar el nivel de aceptación de nuestro sistema. No se debería aceptar ningún sistema que no cumpliera todos los requerimientos, a menos que el cliente así lo haya decidido.

En qué casos se utiliza

Es recomendable validar siempre la idoneidad de un sistema, ya que nos permite saber si las funciones implementadas se ajustan a las necesidades especificadas en los requerimientos.

Para saber si un programa es correcto es imprescindible comprobar que permite realizar al usuario todas las funcionalidades indicadas al inicio del programa.

Métricas relacionadas

A continuación se muestran algunas de las métricas que se utilizan para validar la idoneidad. Pueden usarse como métricas externas o como métricas internas, aunque generalmente se usan como externas e indirectas.

Las métricas diseñadas para validar la idoneidad indican una serie de atributos que se deben usar para evaluar de forma explícita las funciones implementadas. Permiten evaluar si son adecuadas para cumplir su misión.

a) Functional Adequacy

¿Cuándo se usa?		Cuando se quiere saber la cantidad de funciones que cumplen los requerimientos del sistema implementadas hasta el momento.	
¿Qué evalúa?		El número de funciones que son correctas para llevar a cabo la tares especificadas, y se compara con el número de funciones que se deben evaluar.	
Técnicas relacionadas		*Use case testing* *State transition*	
Documentación necesaria	ERS Informe de evaluación	**Público objetivo**	Programador ***Software Quality Assurance***
Fórmula	X=1-A/B A=número de funciones erróneas B=número de funciones evaluadas		
Valores esperados	0.0 < X < 1.0	**Valores correctos**	Cercanos al 1.0

TABLA 48 RESUMEN DE LA MÉTRICA *FUNCTIONAL ADEQUACY*

b) Functional especification stability

¿Cuándo se usa?		Cuando se quiere validar la estabilidad de la especificación. Muchas veces las especificaciones no se ajustan realmente a las necesidades del sistema. Esta métrica permite validar si se han tenido que realizar muchos cambios en el sistema después de la puesta en marcha.	
¿Qué evalúa?		El número de funciones que han sido modificadas después que el sistema está en marcha. Esta información se va recopilando durante la fase de mantenimiento del producto.	
Técnicas relacionadas		Se anotan la cantidad de cambios realizados durante el mantenimiento	
Documentación necesaria	ERS Informe de evaluación	**Público objetivo**	Encargado del mantenimiento ***Software Quality Assurance***
Fórmula	X=1-A/B A= número de funciones que se han cambiado B= número de funciones descritas en los requerimientos		
Valores esperados	0.0 < X < 1.0	**Valores correctos**	Cercanos al 1.0

TABLA 49 RESUMEN DE LA MÉTRICA *FUNCTIONAL ESPECIFICATION STABILITY*

c) Functional Implementation Completeness

<table>
<tr><td colspan="2">¿Cuándo se usa?</td><td colspan="2">Cuando se quiere conocer la estabilidad de la especificación. A veces, durante la implementación se detectan errores en la especificación que implican cambios en la implementación. Información incoherente, campos omitidos, funcionalidades no descritas pero que son necesarias para cumplir los requerimientos, etc. Esta métrica permite comprobar si el resultado se está desviando mucho de los requerimientos.</td></tr>
<tr><td colspan="2">¿Qué evalúa?</td><td colspan="2">El número de funciones que quedan por implementar vs. el número de funciones descritas en los requerimientos.</td></tr>
<tr><td colspan="2">Técnicas relacionadas</td><td colspan="2">Se anotan las modificaciones introducidas durante la fase de diseño, implementación y pruebas del sistema. Y se añaden en forma de anexo a los requerimientos.</td></tr>
<tr><td>Documentación necesaria</td><td>ERS
Informe de evaluación</td><td>Publico objetivo</td><td>Programador
Software Quality Assurance</td></tr>
<tr><td>Fórmula</td><td colspan="3">X=1-A/B
A= número de funciones que faltan
B= número de funciones descritas en los requerimientos</td></tr>
<tr><td>Valores esperados</td><td>0.0 < X < 1.0</td><td>Valores correctos</td><td>Cercanos al 1.0</td></tr>
<tr><td colspan="2">Métricas similares</td><td colspan="2">Functional Implementation completeness: funciona igual pero, además de las funciones que faltan, cuenta también las funciones incorrectas</td></tr>
</table>

TABLA 50 RESUMEN DE LA MÉTRICA *FUNCTIONAL IMPLEMENTATION COMPLETENESS*

Pasos a seguir

Para validar la idoneidad es recomendable realizar las siguientes tareas:

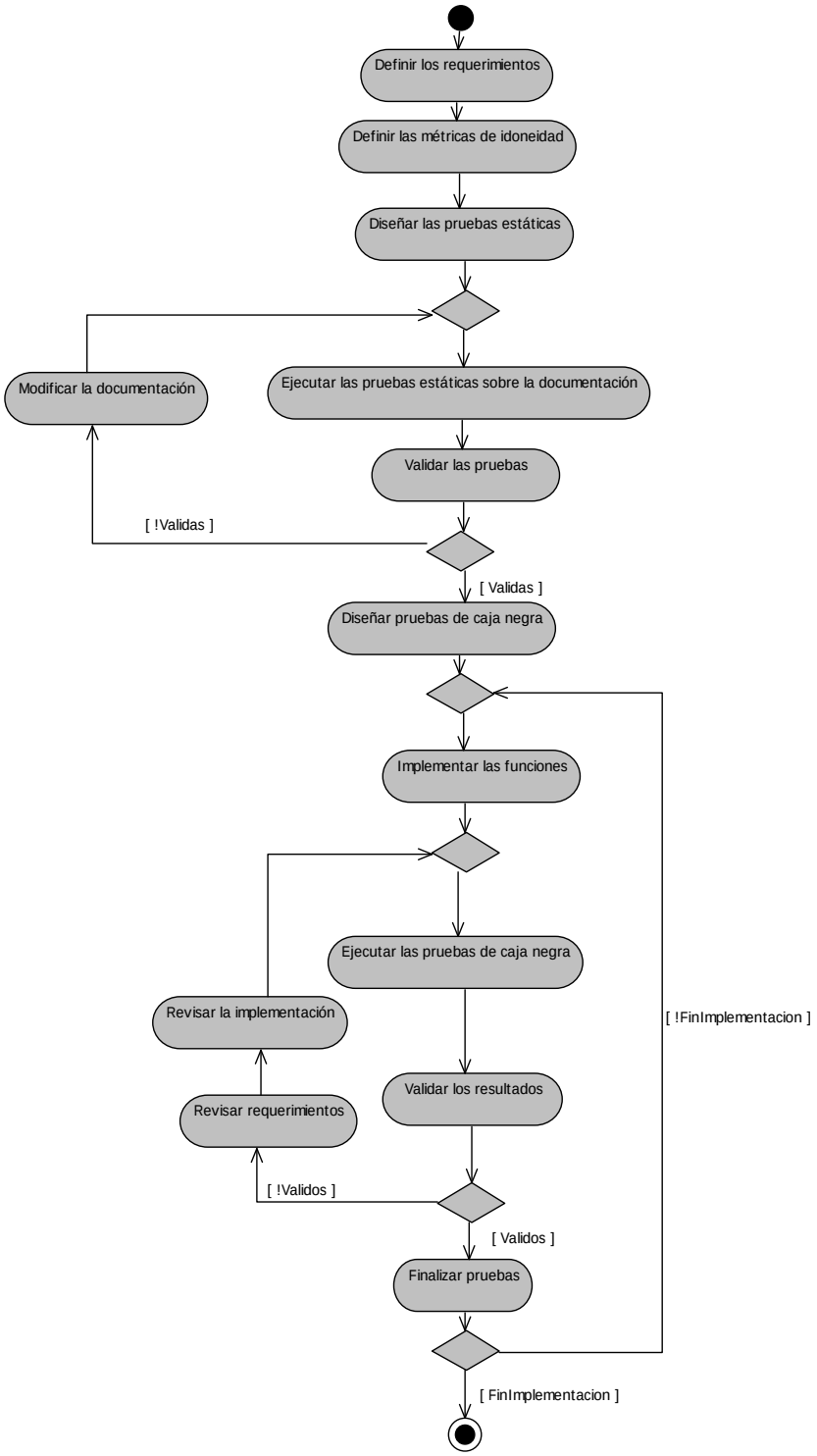


ILUSTRACIÓN 32 DIAGRAMA DE ACTIVIDADES PARA VALIDAR LA IDONEIDA

Descripción de los pasos:

- **Definir los requerimientos:** para poder validar la idoneidad es necesario que ante todo se documenten de forma correcta los requerimientos del sistema. Esta documentación será la que se usará para definir las pruebas que validarán este atributo del software.
- **Definir las métricas que se van a usar:** una vez conocemos todos los requerimientos, se debe decir que métricas se usarán para validar que el sistema cumple correctamente todas las funcionalidades necesarias para el usuario. Por ejemplo si se escoge la métrica *functional adequacy* se podrá saber si las funcionalidades implementadas hasta el momento son las correctas, o si se escoge la métrica *functional especification stability* podremos saber si la especificación es correcta.
- **Diseñar las pruebas estáticas**: si las especificaciones del sistema no son claras, verificar la idoneidad puede llegar a ser muy complicado. Las pruebas estáticas permiten comprobar que la documentación que especifica las funcionalidades del sistema es adecuada y está aceptada por todas las partes implicadas en el proyecto.
- **Ejecutar las pruebas estáticas sobre documentación:** una vez se tiene toda la documentación necesaria para realizar las pruebas, se ejecutan las pruebas estáticas sobre la misma.
- **Validar las pruebas**: una vez ejecutadas se recogen todos los comentarios y cambios facilitados por las partes implicadas.
- **Modificar la documentación**: se realizan las modificaciones que son necesarias.
- **Volver a ejecutar las pruebas estáticas:** se vuelve a validar toda la documentación facilitada hasta que todas las partes implicadas la acepten.
- **Diseñar las pruebas de caja negra:** para validar la idoneidad normalmente se utilizan pruebas de caja negra, que permiten saber si la funcionalidad del sistema es la correcta. En este punto se diseñarán todas las pruebas necesarias para poder trabajar con las métricas descritas en el punto anterior, utilizando los requerimientos ya validados con las pruebas estáticas. Las técnicas más utilizadas para validar la idoneidad son *Use case testing* y *State transtion.*
- **Implementar las funciones**: una vez la documentación esta correctamente verificada y las pruebas de caja negra diseñadas, se puede pasar a la implementación del sistema. Es recomendable ir validando el software durante la implementación, por lo que en este punto se implementarán un conjunto de funciones en cada iteración.
- **Ejecutar las pruebas de caja negra:** ejecutar las pruebas para esas funciones.
- **Validar los resultados con las métricas:** comparar los resultados obtenidos con los esperados, y validarlos con las métricas.
- **Revisar los requerimientos:** si hay diferencias, validarlas con los requerimientos para verificar que éstos son correctos.
- **Revisar la implementación:** realizar los cambios necesarios.
- **Volver a ejecutar las pruebas**: volver a ejecutar las pruebas que no funcionaron correctamente.

- **Volver a validar las métricas:** volver a validar que los resultados son correctos y aplicar las métricas.
- **Finalizar las pruebas:** si todas las pruebas se han superado satisfactoriamente y todas las funcionalidades han sido superadas, se finalizan las pruebas.

Si la mayoría de implementaciones que realizará la empresa van a requerir validar la idoneidad de los entregables, es recomendable añadir esta información a la estrategia y a las políticas de pruebas de la empresa.

Ejemplo
Se tiene un caso de uso y queremos saber si se ajusta a nuestra implementación. Se requiere que el usuario pueda introducir recetas dentro del sistema, y éstas se guarden en la base de datos para mostrarse más tarde desde la administración. Se quiere validar que se guardan correctamente todos los datos de la receta en la base de datos y que se visualizan de forma adecuada.

Para ello se crean una serie de pruebas que introducirán datos dentro del formulario de forma automática. Una vez introducidos se comparan los valores introducidos con los valores visualizados. Se debe mirar que el número de campos, la tipología y el valor de los mismos sea el correcto. Una vez esto sea correcto se comparará con los campos especificados en los requerimientos. Para ello se usarán unas métricas tipo *Functional Adequacy*.

En caso que no sean los campos del formulario los mismos campos que los definidos en los requerimientos, se deberá consultar el anexo a la especificación. Puede ser que los requerimientos se hayan modificado durante la implementación. Si es así se pueden utilizar unas métricas tipo *Functional Implementation Completeness*, que nos permitirá ir conociendo lo buena que era la especificación inicial del sistema.

Se deberá repetir esta operación para todos los formularios del sistema.

5.4.3. Exactitud (*Accuracy*)

Definición

La exactitud es la capacidad del SW para proveer de aquello que es lo correcto, o cumplir los resultados o efectos con el grado de precisión necesario.

Para medir la exactitud es importante evaluar si el sistema se muestra en el formato adecuado para el usuario y opera de forma correcta los valores. Se debe observar que los valores tengan los formatos definidos, los límites establecidos, estén contemplados los valores erróneos, etc.

Cómo documentar

Para poder validar la exactitud se debe especificar qué significa un resultado exacto para el sistema. Para ello se debe tener en cuenta la siguiente información:

- Tipología de datos de entrada y salida de cada funcionalidad.
- Límites superiores e inferiores de los posibles valores de entrada y salida.
- Qué tipo de datos se deben tratar como valores incorrectos.
- Cantidad de decimales de los valores numéricos
- Codificación necesaria para los datos alfa-numéricos

Esta información se puede documentar de la siguiente forma:

- **Definir un listado de propiedades genérica**: por ejemplo la codificación y los decimales son características comunes a casi todos los casos
- **Crear una tabla con las consideraciones específicas para cada funcionalidad o caso de uso**: esto permitirá especificar modificaciones en las propiedades genéricas (en los casos que sea necesario), definir los resultados esperados para esa funcionalidad en concreto, especificar los límites de los valores que se usarán en esas funciones, o cuál es la tipología de datos para esa función.
- **Especificar los casos de error para cada funcionalidad**: si se trabaja con casos de usos la descripción textual de los mismos nos va a permitir definir los caminos alternativos o los caminos erróneos. Si no se usan estos diagramas, se puede crear un listado o diagrama de flujos erróneos relacionado con cada funcionalidad.

Técnicas relacionadas

Para comprobar la exactitud de los datos de un sistema se pueden utilizar técnicas dinámicas basadas en las especificaciones o técnicas de caja negra como:

- *Use case testing*
- *State transtion*
- *Decision tables*
- *Boundary value analysis*
- *Equivalence partitioning*

Estas técnicas se aplicarán a nivel de componente y de integración.

En qué casos se utiliza

Validar la exactitud de un sistema es importante en cualquier implementación, pero se debe realizar un análisis más amplio cuando el programa trabaja con datos muy precisos o realiza operaciones matemáticas complejas. También es muy importante validar la exactitud cuando se trabaja con valores monetarios o se realizan operaciones económicas. En estos casos un pequeño error en los resultados puede tener consecuencias muy importantes.

Métricas relacionadas

A continuación se muestran algunas de las métricas que se utilizan para validar la exactitud. Pueden usarse como métricas externas o como métricas internas, variando en función de si se aplican en ejecución o no.

Las métricas diseñadas para validar la exactitud indican un conjunto de atributos para evaluar la capacidad del producto de software, comprobando si los resultados son correctos y/o convenientes.

a) Accuracy Expectation

¿Cuándo se usa?		Cuando se buscan diferencias entre el resultado obtenido y el esperado. Esta métrica es utilizada para comprobar que los valores de salida obtenidos son los especificados.	
¿Qué evalúa?		Se hacen pruebas con los inputs y los outputs y se compara con los resultados esperados. Se contabilizan aquellos casos en los que los resultados son inaceptables.	
Técnicas relacionadas		*Decision tables* *Boundary value analysis* *Equivalence partitioning*	
Documentación necesaria	ERS Manual de usuario Opinión de los usuarios Informe de las pruebas	**Publico objetivo**	Programador ***Usuario***
Fórmula	X=A/T A= número de casos con diferencias más allá de lo permitido T=tiempo de operación		
Valores esperados	X >=0	**Valores correctos**	cercanos al 0

TABLA 51 RESUMEN DE LA MÉTRICA *ACCURACY EXPECTATION*

b) Precision

¿Cuándo se usa?		Cuando se quiere conocer la frecuencia con la que los usuarios encuentran resultados de precisión inadecuada. Esta métrica es interesante cuando una función trabaja con un número de decimales muy elevado o realiza operaciones complejas.	
¿Qué evalúa?		El número de resultados con precisión inadecuada. Se ejecutan pruebas funcionales y se anotan los resultados para comprobar que los resultados tienen la precisión necesaria.	
Técnicas relacionadas		*Decision tables* *Boundary value analysis* *Equivalence partitioning* También se deben documentar los resultados inadecuados reportados por los usuarios, o encontrados aplicando pruebas	
Documentación necesaria	ERS Informe de las pruebas	**Público objetivo**	Programador Usuario
Fórmula	X=A/T A= número de resultados con precisión inadecuada T=tiempo de operación		
Valores esperados	X >= 0	**Valores correctos**	cercanos al 0
Métricas relacionadas		*Computational Accuracy:* en lugar de la precisión inadecuada, evalúa la cantidad de operaciones inexactas obtenidas. Sirve para conocer la frecuencia con la que el usuario encuentra resultados no adecuados	

TABLA 52 RESUMEN DE LA MÉTRICA *PRECISION*

Pasos a seguir

Para validar la exactitud es recomendable realizar las siguientes tareas:

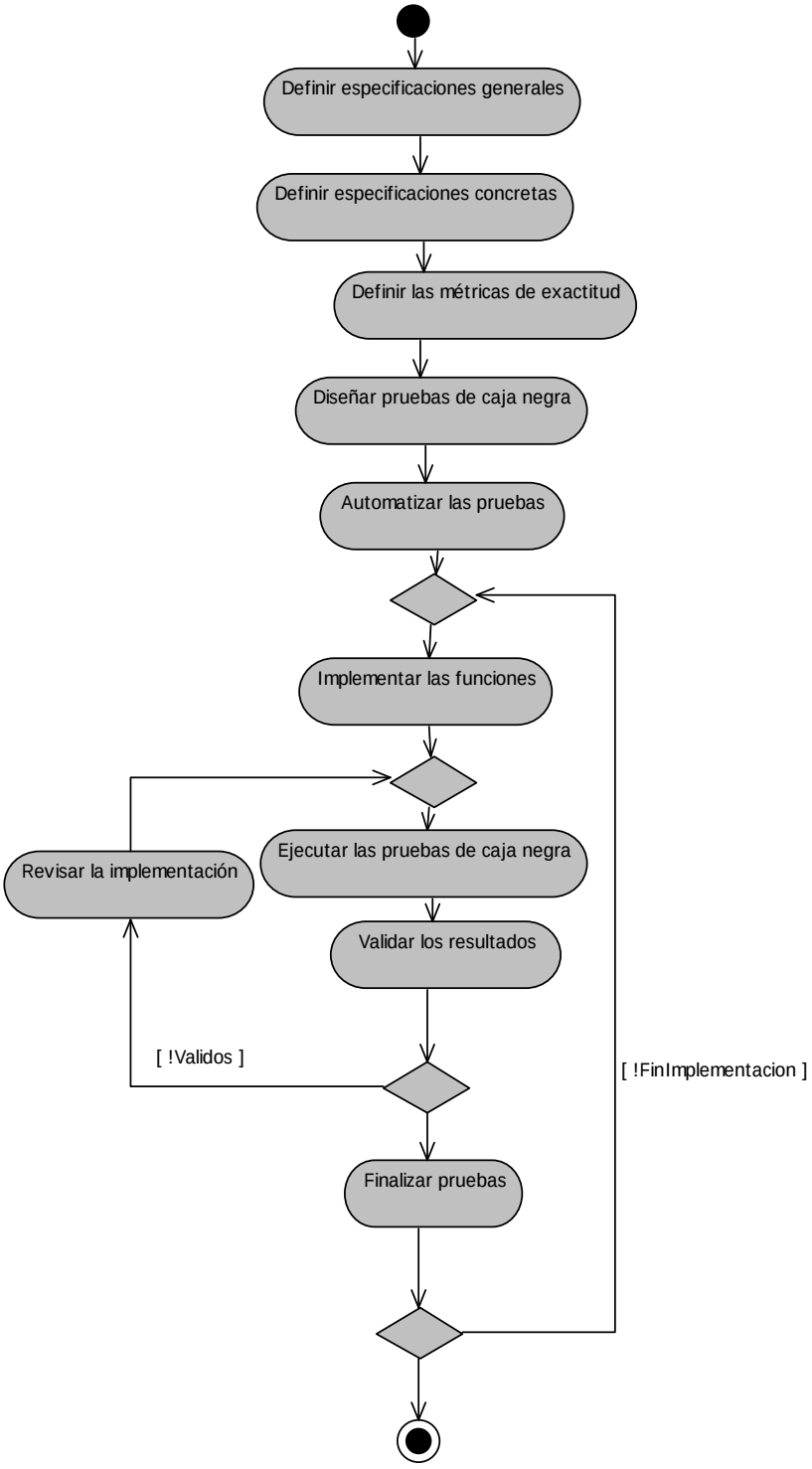


ILUSTRACIÓN 33 DIAGRAMA DE ACTIVIDADES PARA VALIDAR LA EXACTITUD

Descripción de los pasos:

- **Definir especificaciones generales:** primero de todo se debe documentar aquellas especificaciones de datos que serán comunes a todas las pruebas del sistema.
- **Definir especificaciones específicas:** una vez documentadas todas las comunes se procederá a documentar aquellas especificaciones de datos que se deben validar en funciones en concreto. Por ejemplo: los límites de un *array* en concreto que solamente se utiliza en una función.
- **Definir las métricas de exactitud:** una vez conocida la tipología de datos que se quiere validar, se deben escoger las métricas a utilizar. Si se quiere validar la precisión de los decimales de unos números en concreto se puede utilizar una métrica como *Precision*; si queremos validar si los datos que se obtienen como resultado son los correctos será apropiada una métrica como *Accuracy Expectation.*
- **Definir los datos que se quieren validar:** una vez conocidas las métricas y el tipo de especificaciones que se van a validar se debe seleccionar sobre qué partes del sistema se realizarán las pruebas, ya que la cantidad de datos que se utilizan pueden llegar a ser muy grande y la cantidad de validaciones a realizar demasiado costosa.
- **Diseñar las pruebas de caja negra:** para validar la exactitud normalmente se utilizan pruebas de caja negra que permiten verificar los resultados de cada una de las funcionalidades, sin necesidad de conocer como están implementadas. En este punto se diseñarán todas las pruebas necesarias para poder trabajar con las métricas descritas en el punto anterior, basándose en las especificaciones que se quieren cumplir. *Decision tables, Boundary value analysis y Equivalence partitioning* serán las técnicas más utilizadas para validar la exactitud.
- **Automatizar las pruebas:** en este tipo de pruebas se puede utilizar un software que permita automatizar las pruebas y ejecutar de formar secuencial todos los valores que se quieren introducir como entradas de cada una de las funciones.
- **Implementar las funciones**: una vez las pruebas de caja negra están diseñadas, se puede pasar a la implementación del sistema. Es recomendable ir validando el software durante la implementación, por eso mismo en este punto se codificarán un conjunto de funciones en cada iteración.
- **Ejecutar las pruebas de caja negra:** ejecutar las pruebas para esas funciones.
- **Validar los resultados con las métricas:** comparar los resultados obtenidos con los esperados y aplicar las métricas.
- **Revisar la implementación:** realizar los cambios necesarios.
- **Volver a ejecutar las pruebas**: volver a ejecutar las pruebas que no funcionaron correctamente.
- **Volver a validar las métricas:** volver a validar que los resultados son correctos y aplicar las métricas.
- **Finalizar las pruebas:** si todas las pruebas se han superado satisfactoriamente y todas las funcionalidades han sido superadas, se finalizan las pruebas.

Si la mayoría de implementaciones que realizará la empresa van a requerir validar la exactitud de los valores de los datos, es recomendable añadir esta información a la estrategia y a las políticas de pruebas de la empresa

Ejemplo
Se quiere validar un programa creado para gestionar subvenciones en una empresa que imparte formación a trabajadores en paro. La herramienta va a permitir introducir alumnos, cursos y subvenciones, y gestionar toda la información relativa a estos datos.

Dentro de las diferentes funcionalidades que se han implementado hay una, más compleja que las demás, que se encarga de calcular el dinero de una subvención en concreto. Este dinero es distribuido de diferente forma en función de los cursos realizados, asociados a la subvención, y de la cantidad de alumnos que los han finalizado. Es muy importante que este cálculo sea correcto, ya que si la empresa se pasa de alumnos no recibe dinero por la formación de éstos, y si no llega al número establecido pierden dinero ya que debe devolverlo a la entidad que la subvenciona.

Para poder validar estos cálculos, la empresa que imparte los cursos ha entregado los cálculos realizados de forma manual en ejercicios anteriores.

Estos documentos permiten crear una serie de tablas con la información necesaria para este cálculo. El dinero de la subvención, el coste de la formación de un alumno (material, coste de profesor por alumno, mantenimiento de salas) y el coste indirecto asociado a la formación. También se han creado una serie de pruebas de caja negra para poder validar esta función.

Con toda esta información se ha ejecutado una prueba para cada uno de los ejercicios del cliente. Se han introducido los datos necesarios para realizar los cálculos en función de la información facilitada y se han comparado con los resultados esperados utilizando la métrica *Accuracy Expectation*. Como se han detectado algunas desviaciones en los resultados de algunos de los ejercicios se ha verificado la precisión de cada una de las operaciones usando la métrica *Precision* ya que la pérdida de un decimal en algunas operaciones puede causar una pérdida de dinero considerable, al ser éstas numerosas.

5.4.4. Interoperabilidad (*Interoperability*)

Definición

Es la capacidad del SW de interactuar con uno o varios conjuntos del propio sistema, o de otros sistemas. Esta propiedad es importante cuando el sistema necesita operar con otros sistemas, o enviar información a algún sistema externo.

Cómo documentar

Para poder validar la exactitud debemos conocer cuáles son los sistemas externos con los que debe operar.

- Diagrama de comunicación (UML), que dé una idea visual de todos las interfaces externas con las que se debe interactuar.
- Listado especificando la información básica de las interfaces con las que se quiere interactuar.
- Tabla con las configuraciones necesarias para poder enviar y recibir los datos necesarios.
- Tabla con el listado y la información relativa a hardware adicional, si éste es necesario para realizar las comunicaciones.
- Tabla con la especificación de la información que se quiere enviar y recibir.
- Referencias a todos los documentos relacionados con los sistemas externos.

Técnicas relacionadas

Para comprobar la interoperabilidad de un sistema se pueden utilizar técnicas dinámicas basadas en las especificaciones, o técnicas de caja negra como:

- *Use case testing*
- *State transtion*
- *Decision tables*
- *Boundary value analysis*
- *Equivalence partitioning*

A diferencia de los casos anteriores estas pruebas se realizarán a nivel de integración y de sistema, ya que a nivel de componente no tienen mucho sentido.

Será necesario crear una serie de *Drivers* y *Sturbs* que permitirán simular las interfaces externas durante el *proceso de pruebas* del sistema.

En qué casos se utiliza

La interoperabilidad debe validarse en todos los sistemas implementados, ya que todos tendrán siempre entradas y salidas asociadas. Las entradas básicas siempre serán desde teclado o ratón, y las salidas serán por pantalla o a través de la escritura de fichero. Es recomendable tener unas pruebas básicas diseñadas para validarla.

La importancia de comprobar la interoperabilidad aparece cuando los sistemas deben comunicarse a través de entradas o salidas diferentes a las básicas. Por ejemplo recibir o enviar datos a un sistema externo, enviar información a través de la red, publicar información a través de sistemas de sindicalización de datos, etc. En estos casos se

deberán crear pruebas específicas para validar el intercambio de información y comprobar que están controlados todos los posibles casos de error detectados.

Métricas relacionadas

A continuación se muestran algunas de las métricas que se utilizan para validar la interoperabilidad. Pueden usarse como métricas externas o como métricas internas, aunque generalmente se usarán como métricas externas y se calcularán de forma indirecta.

Las métricas diseñadas para validar la interoperabilidad indican un conjunto de atributos que permiten evaluar la capacidad de interacción del producto de software con otros sistemas reconocidos.

a) Data Exchangeability

¿Cuándo se usa?		Cuando se quiere saber lo correcto que es el intercambio de información entre funciones de diferentes interfaces durante la puesta en marcha. Esta métrica permite saber si las comunicaciones funcionan de forma aceptable o si se producen errores demasiado a menudo. Para ello se valida que todos los formatos que se pueden intercambiar	
¿Qué evalúa?		La información que se envía como salida del sistema, basándose en las especificaciones. Se cuenta el número de formatos de datos que se intercambian de forma correcta y se comparan con el total de formatos que se deben intercambiar.	
Técnicas relacionadas		*Decision tables* *Boundary value analysis* *Equivalence partitioning*	
Documentación necesaria	ERS Informe de las pruebas	**Público objetivo**	Programador
Fórmula	X=A/B A=número de formatos aprobados B=número total de formatos que se deben intercambiar		
Valores esperados	0 < X <1	**Valores correctos**	cercanos al 1.0

TABLA 53 RESUMEN DE LA MÉTRICA *DATA EXCHANGEABILITY*

b) Data Exchangeability (User's success attempt based)

¿Cuándo se usa?		Cuando se quiere evaluar la cantidad de fallos que sufre el usuario, y si el sistema funciona correctamente la mayoría de veces que se intercambian datos	
¿Qué evalúa?		La cantidad de veces que se usa la función y la cantidad de veces que falla	
Técnicas relacionadas		*Use case testing* *State transition*	
Documentación necesaria	ERS Informe de las pruebas	**Público objetivo**	Encargado del mantenimiento
Fórmula	X=1 – A/B Y= A/T A = número de fallos B = veces que se usa T = período total		
Valores esperados	Y >=0 0 =< X =< 1	**Valores correctos**	Y cercana a 0 X cercana a 1

TABLA 54 RESUMEN DE LA MÉTRICA *DATA EXCHANGEABILITY (USER'S SUCCESS ATTEMPT BASED)*

Pasos a seguir

Para validar la interoperabilidad es recomendable realizar las siguientes tareas:

ILUSTRACIÓN 34 DIAGRAMA DE ACTIVIDADES PARA VALIDAR LA INTEROPERABILIDAD

Descripción de los pasos:

- **Crear diagrama de comunicación**: ante todo se deberán documentar las comunicaciones que realizará el sistema.
- **Definir especificaciones de la comunicación:** una vez conocemos las comunicaciones se especificará la tipología de datos que éstas envían.
- **Especificar las configuraciones necesarias:** también es importante documentar cómo se configuran las comunicaciones, ya que la configuración puede afectar a la implementación y a las pruebas.
- **Definir las métricas de interoperabilidad:** después de conocer como funcionarán las comunicaciones se deberá escoger qué métricas de *Data Exchangeabilitity* se utilizarán.
- **Diseñar pruebas a nivel de integración:** se diseñarán pruebas de caja negra a nivel de integración, para poder ir validando que las comunicaciones se realizan correctamente de forma incremental. En estas pruebas se utilizarán *Drivers* y *Stubs*.
- **Diseñar pruebas a nivel de sistema:** para validar que todas las comunicaciones funcionan correctamente se diseñarán pruebas a nivel de sistema que permitirán validar la funcionalidad con otras interfaces.
- **Automatizar las pruebas:** en este tipo de pruebas se puede utilizar un software que permita automatizar las pruebas. De esta forma se pueden ejecutar de formar secuencial todos los valores que se quieren introducir como entradas de cada una de las funciones que requieren comunicación con alguna interface externa.
- **Implementar las funciones**: una vez las pruebas de caja negra están diseñadas, tanto a nivel de integración como a nivel de sistema, se puede pasar a la implementación del sistema. Es recomendable ir validando el software durante la implementación, por lo que en este punto se implementarán un conjunto de funciones en cada iteración.
- **Integrar funciones relacionadas:** antes de empezar a ejecutar las pruebas se deben integrar las funciones que deben probarse juntas, y añadir *drivers* y *stubs* si son necesarios.
- **Ejecutar las pruebas a nivel de integración:** ejecutar las pruebas de integración para esas funciones.
- **Validar los resultados:** comparar los resultados obtenidos con los esperados, y aplicar las métricas.
- **Revisar la implementación:** realizar los cambios necesarios.
- **Volver a ejecutar las pruebas**: volver a ejecutar las pruebas de integración que no funcionaron correctamente y aquellas que pueden estar relacionadas
- **Volver a validar las métricas:** volver a validar que los resultados son correctos y aplicar las métricas.
- **Finalizar pruebas de integración:** si todas las pruebas se han superado satisfactoriamente y todas las funcionalidades han sido superadas, se finalizan las pruebas.
- **Ejecutar pruebas a nivel de sistema:** ejecutar las pruebas del sistema.

- **Validar los resultados:** comparar los resultados obtenidos con los esperados, y aplicar las métricas.
- **Realizar modificaciones necesarias:** realizar los cambios necesarios.
- **Volver a ejecutar las pruebas**: volver a ejecutar las pruebas de sistema que no funcionaron correctamente y aquellas que pueden estar relacionadas
- **Volver a validar las métricas:** volver a validar que los resultados son correctos y aplicar las métricas.
- **Finalizar las pruebas:** si todas las pruebas se han superado satisfactoriamente, se finalizan las pruebas.

Es recomendable añadir a la estrategia y a las políticas de pruebas de la empresa las pruebas de interoperabilidad para las entradas y salidas básicas de todo el sistema que se implemente de forma habitual.

Ejemplo

Tenemos un sistema que queremos sindicalizar con una red social. Para ello debemos permitir que ésta consulte datos de nuestro sistema, por lo que hemos programado un código dentro de la red social, que realice peticiones a nuestra base de datos. Para explicar el funcionamiento de esta comunicación tenemos una documentación que nos permite saber todos los accesos que se harán a la base de datos y qué tipo de datos se consultarán.

Se han diseñado una serie de pruebas de caja negra desde la red social que realizarán consultas y modificaciones dentro de la base de datos. Esta registrará en un log todas las operaciones realizadas. Finalmente se comparará el número de peticiones realizadas con el número de peticiones registradas, usando una métrica de *Data Exchangeability* que permitirá saber el número de errores que se cometen.

5.4.5. Seguridad (Security)

Definición

Capacidad del SW de proteger información y datos, de forma que personas o sistemas no autorizados no pueda leerlo ni modificarlo, y personas autorizadas sí puedan hacerlo.

Cómo documentar

Para poder validar la seguridad de un sistema se debe primero definir qué significa seguridad para ese sistema en concreto

- Diagramas Entidad-Relación
- Diagramas de clases (UML)
- Diagrama de despliegue (UML)
- Documentar permisos de los roles
- Documentar los posibles accesos al sistema
- Documentar funciones que crean, usan, modifican o destruyen datos
- LOPD (Ley Orgánica de protección de datos)

Técnicas relacionadas

Para comprobar la seguridad de un sistema se pueden utilizar técnicas dinámicas basadas en las especificaciones, o técnicas basadas en "prueba/error" como:

- *Error guessing*
- *Exploratory testing*

Si el sistema es Web, también se deben tener en cuenta aquellas técnicas que se utilizan para vulnerar este tipo de sistemas, como por ejemplo:

- *SQL-Injection*
- *Cross-Site Scripting*
- *XPATH injection*

Los errores de seguridad también pueden variar en función de la dimensión de los datos que se intercambian. Por ejemplo: una técnica para vulnerar un sistema puede consistir en enviar una cantidad de datos muy grandes hasta conseguir que éste caiga. O, en otros casos, una simple sentencia de base de datos maliciosa no controlada que puede afectar a todos los datos que tenemos guardados.

La seguridad debe validarse a nivel de sistema, aunque también puede ser útil a nivel de integración.

En qué casos se utiliza

La seguridad se debe validar en todas las implementaciones, aunque es realmente importante tenerla en cuenta cuando los datos almacenados tienen un nivel de privacidad muy alto. En estos casos se debe asegurar que no se pierden datos, que no se realizan errores en las operaciones y que el sistema tiene bien controladas todas las técnicas maliciosas que pueden afectar al sistema.

También hay que controlar la seguridad del sistema durante toda la fase de mantenimiento del mismo, ya que aparecen nuevas técnicas maliciosas continuamente.

Métricas relacionadas

A continuación se muestran algunas de las métricas que se utilizan para validar la seguridad. Pueden usarse como métricas externas o como métricas internas, aunque normalmente serán de tipo externas e indirectas.

Las métricas diseñadas para validar la seguridad indican un conjunto de atributos para evaluar la capacidad del producto software, para evitar el acceso ilegal al sistema y / o a los datos.

a) Access auditability

<table>
<tr><td colspan="2">¿Cuándo se usa?</td><td colspan="2">Cuando se quiere conocer si el acceso a los datos de una aplicación se realiza de forma segura. Se deben validar todas las operaciones: consultas, escritura, borrados o modificaciones. Generalmente estos accesos son a Bases de Datos o a ficheros.</td></tr>
<tr><td colspan="2">¿Qué evalúa?</td><td colspan="2">La cantidad de accesos a los datos que quedan grabados, para poder saber el grado de fiabilidad de las operaciones y garantizar así la seguridad de las operaciones</td></tr>
<tr><td colspan="2">Técnicas relacionadas</td><td colspan="2">Error guessing

Exploratory testing</td></tr>
<tr><td>Documentación necesaria</td><td>Especificación de las pruebas

Informe de las pruebas</td><td>Público objetivo</td><td>Programador</td></tr>
<tr><td>Fórmula</td><td colspan="3">X=A/B

A=número de accesos grabados

B=número total de accesos realizados</td></tr>
<tr><td>Valores esperados</td><td>0 < X <1</td><td>Valores correctos</td><td>Cercanos al 1.0</td></tr>
</table>

TABLA 55 RESUMEN DE LA MÉTRICA *ACCESS AUDITABILITY*

b) Access controllability

¿Cuándo se usa?		Cuando se quiere evaluar cómo son de controlables los accesos al sistema, y si se puede *hackear* algún tipo de información almacenada dentro del mismo. Esto incluye accesos desde fuera de la aplicación y accesos con permisos incorrectos.	
¿Qué evalúa?		Evalúa el número de accesos ilegales de las operaciones, y los compara con el número de accesos ilegales especificados. Es importante definir qué es un acceso ilegal dentro del sistema.	
Técnicas relacionadas		*SQL-Injection* *Error guessing* *Exploratory testing*	
Documentación necesaria	Especificación de las pruebas Informe de las pruebas Informe de operaciones	**Público objetivo**	Programador
Fórmula	X=A/B A = número de accesos ilegales contemplados B = número de accesos ilegales especificados		
Valores esperados	0 =< X =< 1	**Valores correctos**	X cercana a 1

TABLA 56 RESUMEN DE LA MÉTRICA *ACCESS CONTROLLABILITY*

c) Data corruption prevention

¿Cuándo se usa?		Cuando se quiere conocer la frecuencia de las acciones corruptas una vez el sistema ya ha sido puesto en marcha. Es importante tener un control de la seguridad del sistema durante el mantenimiento, ya que continuamente aparecen nuevas técnicas que pueden dañar los datos.	
¿Qué evalúa?		El número de acciones corruptas para grandes y pequeñas cantidades de datos. Es muy importante considerar este tipo de errores sobre diferentes dimensiones de datos, ya que muchas veces los errores de seguridad pueden variar en función de la cantidad de datos que se intercambian	
Técnicas relacionadas		*Error guessing* *Exploratory testing* Cualquier prueba nueva de seguridad que aparezca	
Documentación necesaria	Especificación de las pruebas Informe de las pruebas Informe de operaciones	**Público objetivo**	Programador Encargado de mantenimiento
Fórmula	X=1-A/N A= número de veces que los datos grandes fallan N= número de pruebas ejecutadas para probar errores Y=1-B/N B= número de veces que los datos pequeños fallan Z=A/T o B/T T= periodo de tiempo de las pruebas		
Valores esperados	0 =< X =< 1 0 =< Y =< 1 0 =< Z	**Valores correctos**	X cercana a 1 Y cercana a 1 Z cercana a 0

TABLA 57 RESUMEN DE LA MÉTRICA *DATA CORRUPTION PREVENTION*

Pasos a seguir

Para validar la seguridad es recomendable realizar las siguientes tareas:

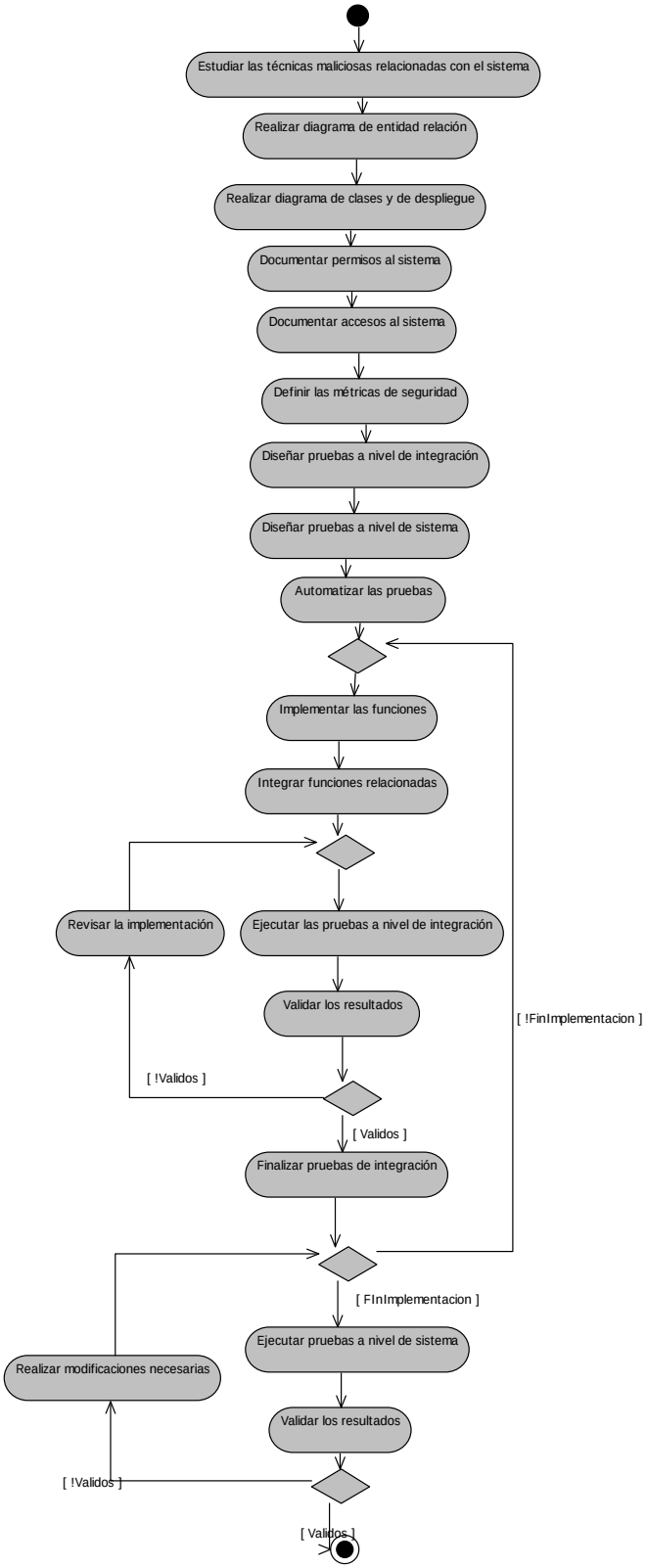


ILUSTRACIÓN 35 DIAGRAMA DE ACTIVIDADES PARA VALIDAD LA SEGURIDAD

Descripción de los pasos:

- **Estudiar las técnicas maliciosas relacionadas con el sistema:** antes de empezar a trabajar con la seguridad de un sistema, debemos documentar qué significa seguridad para ese sistema en concreto.
- **Realizar diagrama de entidad relación:** los diagramas E/R definirán cómo son las bases de datos, y las relaciones que hay entre las diferentes tablas. Es muy importante verificar que éstas son seguras, por lo que se debe documentar correctamente su estructura.
- **Realizar diagrama de clases y de despliegue:** definir como estará instalado nuestro sistema, y cuáles son las clases que se implementarán, nos ayudará, entre otras cosas, a conocer qué partes del sistema son más vulnerables.
- **Documentar permisos al sistema:** es importante documentar cuáles son los permisos de las diferentes funcionalidades del sistema, para garantizar que sólo las personas especificadas podrán ejecutarlas. Las pruebas de seguridad nos permitirán verificar si este punto funciona correctamente.
- **Documentar accesos al sistema:** también deben estar documentadas las diferentes formas de acceder al sistema, para poder verificar que se realizan de forma segura.
- **Definir las métricas de seguridad:** después conocer toda la información relacionada con la seguridad del sistema, se escogerán aquellas métricas que se deben utilizar. Es recomendable trabajar con alguna métrica que valide los accesos al sistema, como *Access controllability*, y una métrica que controle si los datos se pueden corromper, como *Data corruption prevention.*
- **Diseñar pruebas a nivel de integración:** se diseñarán pruebas de caja negra a nivel de integración para poder ir validando que la seguridad está controlada correctamente de forma incremental. En estas pruebas se utilizarán *Drivers* y *Stubs*.
- **Diseñar pruebas a nivel de sistema:** para validar que la seguridad está controlada, se diseñarán pruebas a nivel de sistema que permitirán validar ataques sobre toda la aplicación.
- **Automatizar las pruebas:** en este tipo de pruebas se puede utilizar un software que permita automatizarlas. De esta forma se pueden ejecutar de forma secuencial todo tipo de sentencias que pueden resultar dañinas para el sistema, y así validar que no funcionan.
- **Implementar las funciones**: una vez las pruebas de caja negra estén diseñadas, tanto a nivel de integración como a nivel de sistema, se puede pasar a la implementación del sistema. Es recomendable ir validando el software durante la implementación, por lo que en este punto se implementarán un conjunto de funciones en cada iteración.
- **Integrar funciones relacionadas:** antes de empezar a ejecutar las pruebas se deben integrar las funciones que deben probarse juntas, y añadir *drivers* y *stubs* si son necesarios.
- **Validar los resultados:** comparar los resultados obtenidos con los esperados, y aplicar las métricas.

- **Revisar la implementación:** realizar los cambios necesarios.
- **Volver a ejecutar las pruebas**: volver a ejecutar las pruebas de integración que no funcionaron correctamente, y aquéllas que puedan estar relacionadas
- **Volver a validar las métricas:** volver a validar que los resultados son correctos y aplicar las métricas.
- **Finalizar pruebas de integración:** si todas las pruebas se han superado satisfactoriamente, y todas las funcionalidades han sido superadas, se finalizan las pruebas.
- **Ejecutar pruebas a nivel de sistema:** ejecutar las pruebas del sistema.
- **Validar los resultados:** comparar los resultados obtenidos con los esperados y aplicar las métricas.
- **Realizar modificaciones necesarias:** realizar los cambios necesarios.
- **Volver a ejecutar las pruebas**: volver a ejecutar las pruebas de sistema que no funcionaron correctamente, y aquellas que puedan estar relacionadas
- **Volver a validar las métricas:** volver a validar que los resultados son correctos y aplicar las métricas.
- **Finalizar las pruebas:** si todas las pruebas se han superado satisfactoriamente, se finalizan las pruebas.

Si la mayoría de implementaciones que realizará la empresa van a requerir validar la seguridad del sistema, es recomendable añadir esta información a la estrategia y a las políticas de pruebas de la empresa.

Ejemplo

Tenemos un sistema que requiere que los usuarios faciliten una serie de datos personales a través de un formulario. Debemos validar que estos accesos al formulario no pueden verse afectados por técnicas maliciosas, como el *SQL-Injection*. Para ello se ha generado un listado de sentencias que podrían resultar peligrosas para los datos. También se han introducido datos falsos en la Base de Datos. Una vez realizadas estas tareas, se programa un código que se encargará de ejecutar todas estas pruebas y que registrará los resultados de las mismas.

Mediremos la cantidad de resultados dañinos con una métrica de *Access controllability*. Esto nos permitirá conocer el nivel de seguridad de nuestro sistema y cuáles son los puntos débiles. Después se investigará cómo solucionar estas vulnerabilidades.

6. Planificación

Estos son los tiempos planificados para realizar el proyecto. Las tareas están contabilizadas en días y no en horas, ya que se parte de la premisa que el recurso no dedicará las jornadas enteras a realizar estas tareas. El proyecto se implementará en paralelo junto con otras actividades.

Se pueden identificar una serie de tareas que nos han permitido definir hacia dónde se dirige el proyecto. Aunque estas tareas se sitúen al inicio del proyecto, se ha recurrido a ellas en varias ocasiones durante la implementación, ya que se han ido realizando modificaciones en la documentación utilizada en función de las necesidades de cada punto.

Hay que tener en cuenta que la planificación se ha ido modificando en función del conocimiento obtenido.

Tareas			Duración	Inicio	Fin
Pruebas	Escoger documentación		2 días	01/01/2010 9:00	04/01/2010 19:00
	Guardar documentación interesante		1 día	05/01/2010 9:00	05/01/2010 19:00
	Revisar toda la información		10 días	06/01/2010 9:00	19/01/2010 19:00
Decir puntos a tratar	Escoger información interesante		10 días	01/01/2010 9:00	14/01/2010 19:00
	Descartar información		2 días	15/01/2010 9:00	18/01/2010 19:00
Contenido del proyecto	Fases fundamentales del proceso de pruebas	Buscar documentación	2 días	20/01/2010 9:00	21/01/2010 19:00
		Leer documentación	6 días	22/01/2010 9:00	29/01/2010 19:00
		Diseñar estructura a seguir	2 días	01/02/2010 9:00	02/02/2010 19:00
		Definir pasos necesarios	5 días	03/02/2010 9:00	09/02/2010 19:00
		Describir y adaptar todos los pasos	15 días	10/02/2010 9:00	02/03/2010 19:00
		Buscar información relacionada	5 días	03/03/2010 9:00	09/03/2010 19:00
	Documentación para el proceso de pruebas	Buscar documentación	1 día	10/03/2010 9:00	10/03/2010 19:00
		Describir y estudiar el plan de pruebas	3 días	11/03/2010 9:00	15/03/2010 19:00
		Implementar plan de pruebas	10 días	16/03/2010 9:00	29/03/2010 19:00

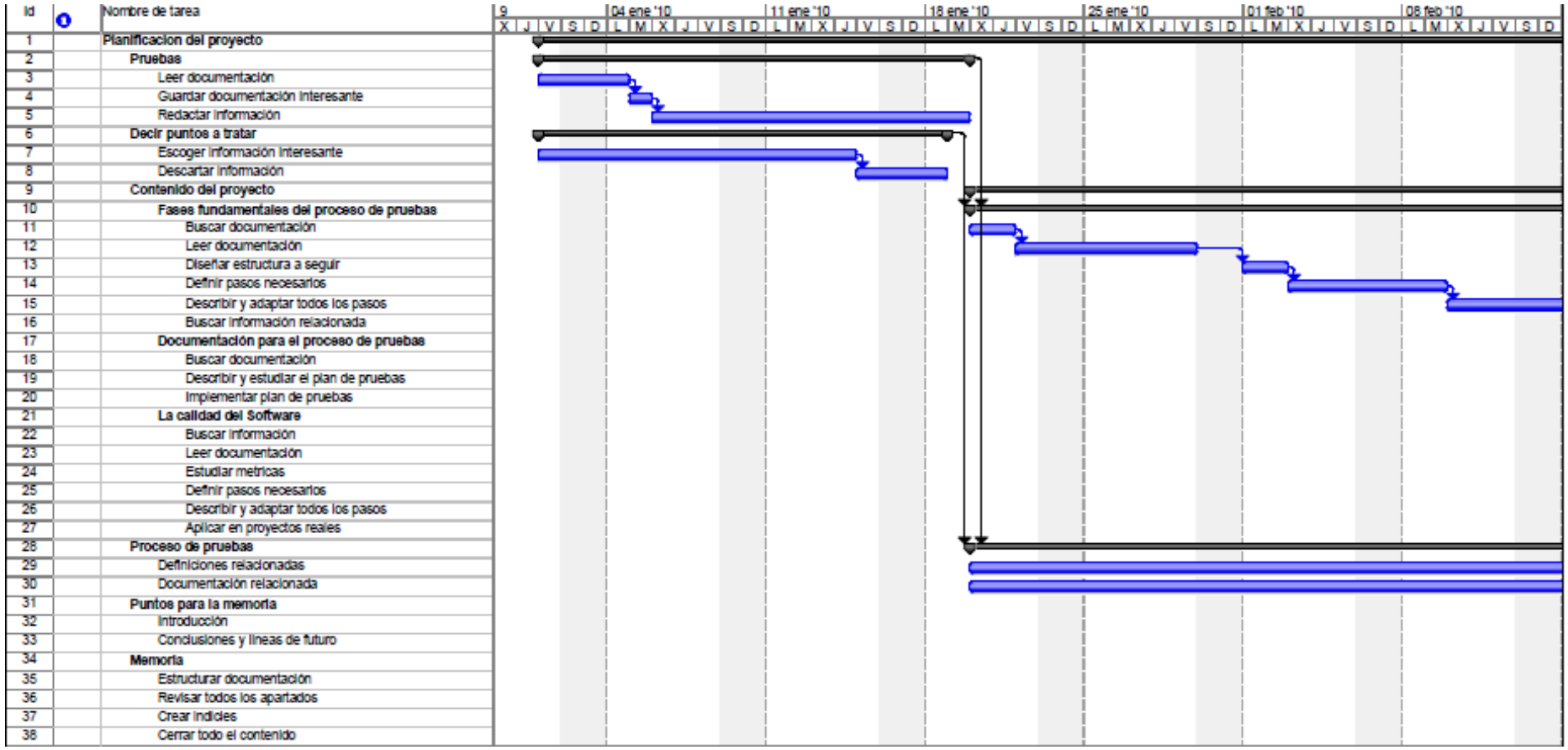
	La calidad del Software	Buscar información	2 días	30/03/2010 9:00	31/03/2010 19:00
		Leer documentación	4 días	01/04/2010 9:00	06/04/2010 19:00
		Estudiar métricas	10 días	07/04/2010 9:00	20/04/2010 19:00
		Definir pasos necesarios	4 días	21/04/2010 9:00	26/04/2010 19:00
		Describir y adaptar todos los pasos	10 días	27/04/2010 9:00	10/05/2010 19:00
		Aplicar en proyectos reales	6 días	11/05/2010 9:00	18/05/2010 19:00
Proceso de pruebas	Definiciones relacionadas		85 días	20/01/2010 9:00	18/05/2010 19:00
	Documentación relacionada		85 días	20/01/2010 9:00	18/05/2010 19:00
Puntos para la memoria	Introducción		2 días	19/05/2010 9:00	20/05/2010 19:00
	Conclusiones y líneas de futuro		2 días	21/05/2010 9:00	24/05/2010 19:00
Memoria	Estructurar documentación		3 días	25/05/2010 9:00	27/05/2010 19:00
	Revisar todos los apartados		10 días	28/05/2010 9:00	10/06/2010 19:00
	Crear indicies		2 días	11/06/2010 9:00	14/06/2010 19:00
	Cerrar todo el contenido		10 días	15/06/2010 9:00	28/06/2010 19:00

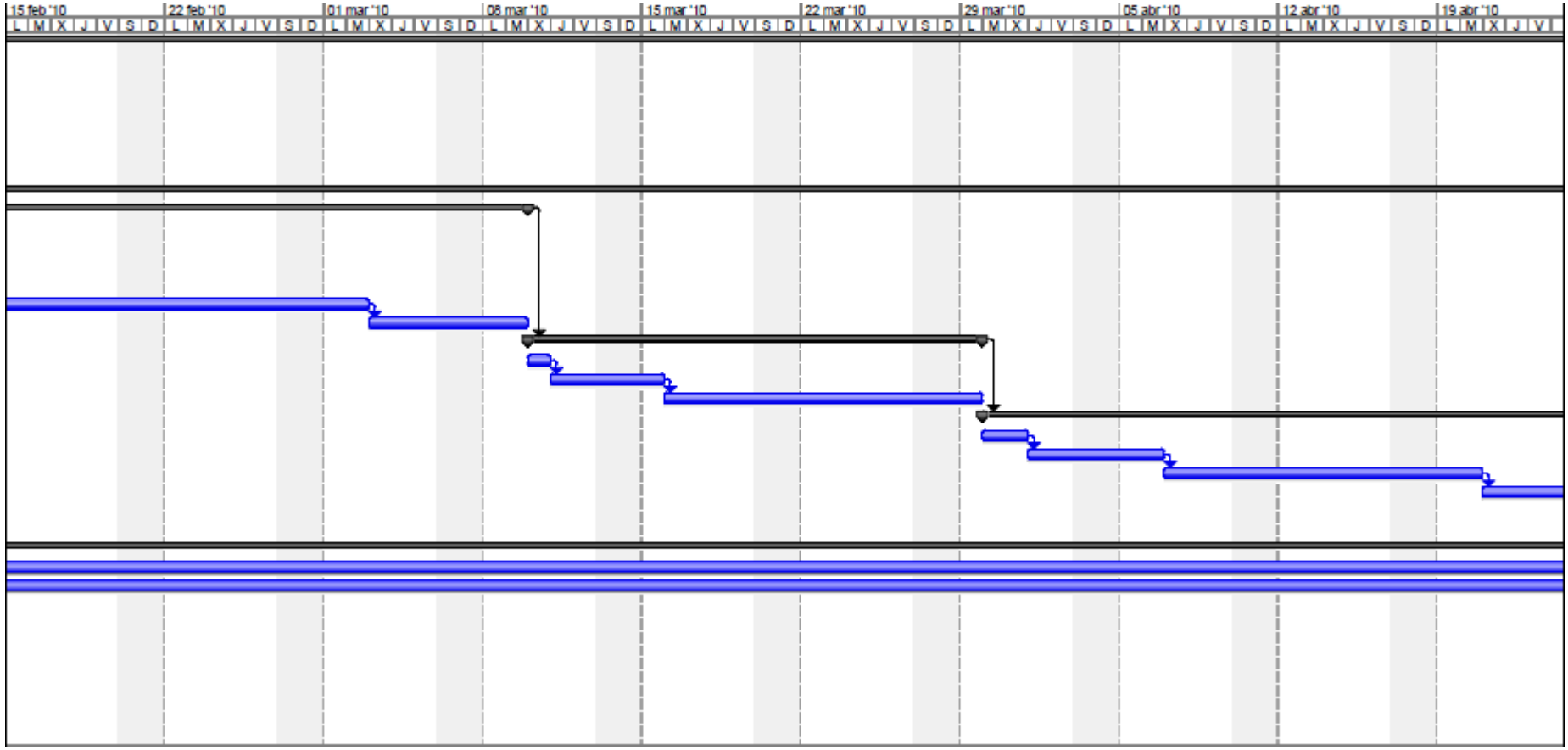
TABLA 58 PLANIFICACIÓN DEL PROYECTO

Se puede observar que durante todo el proyecto se han realizado una serie de tareas en paralelo que han permitido completar la información necesaria para cada uno de los capítulos.

Finalmente se han reservado una serie de días para cerrar todo el contenido, completarlo y finalizar la estructura de este documento.

A continuación podemos observar el Gannt de tareas realizado para planificar este proyecto.




15 feb '10
22 feb '10
01 mar '10
08 mar '10
15 mar '10
22 mar '10
29 mar '10
05 abr '10
12 abr '10
19 abr '10
L M X J V S D

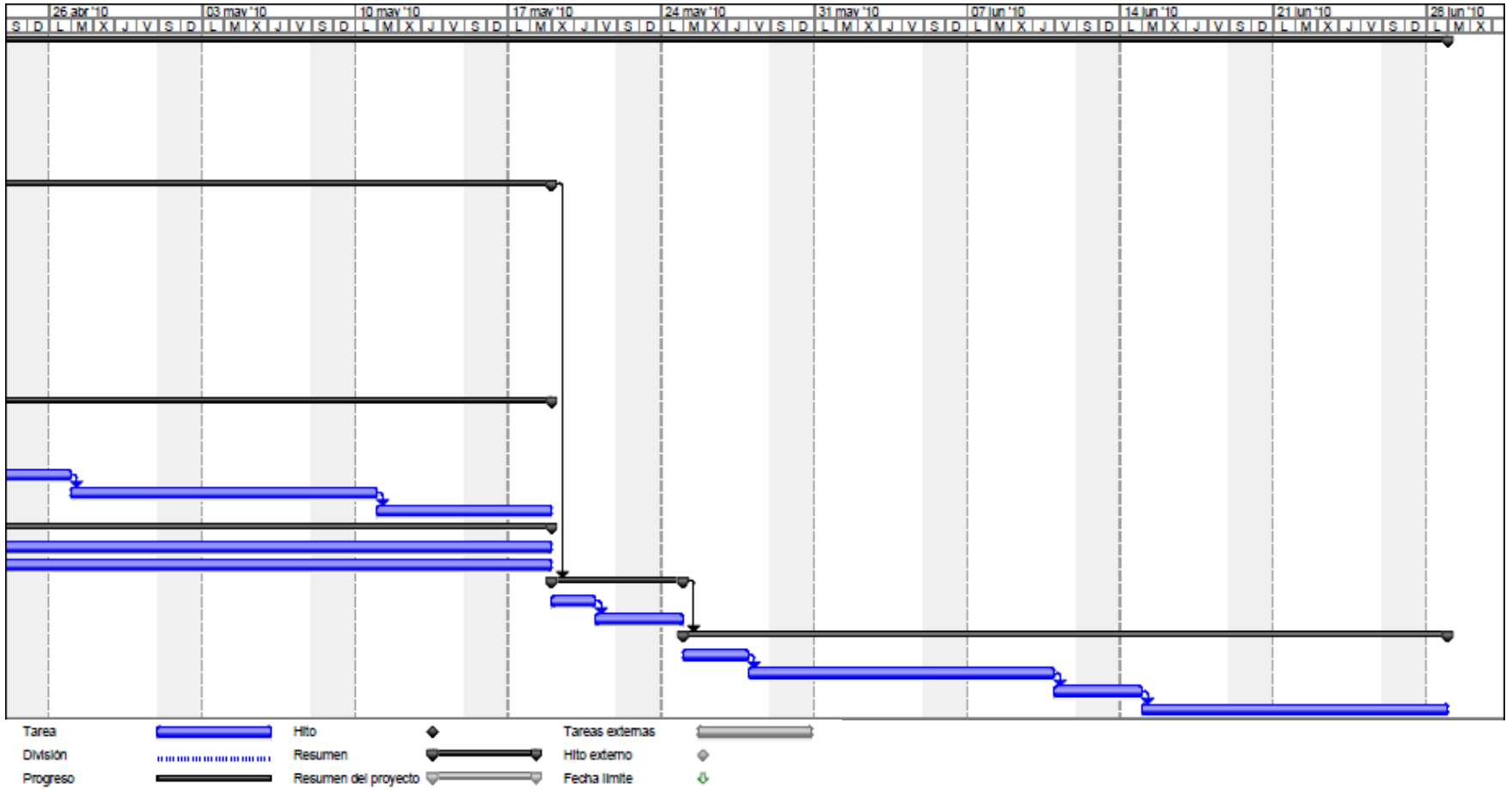


ILUSTRACIÓN 36 GRAFICA CON LA PLANIFICACIÓN DEL PROYECTO

7. Estudio económico

El estudio económico muestra una aproximación del esfuerzo humano que ha sido necesario para realizar el proyecto, se contabilizan las horas invertidas en cada uno de los apartados y se describen las actividades realizas.

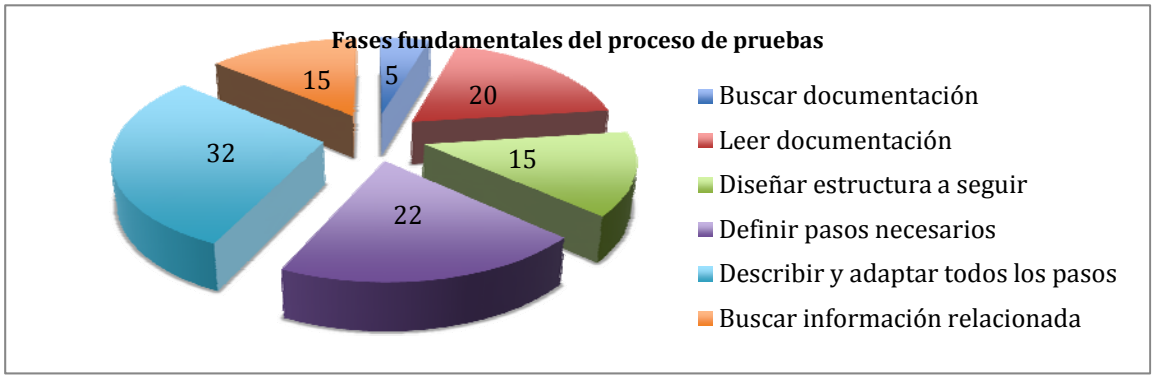
7.1. Fases fundamentales del proceso de pruebas

A continuación se muestran las tareas realizadas para completar el capítulo "Fases fundamentales del proceso de pruebas", se indican los tiempos utilizados y se describen cada una de las tareas.

Tareas	Horas	Descripción
Buscar documentación	5	Buscar libros, artículos y estándares relacionados con el proceso de *proceso de pruebas*.
Leer documentación	20	Leer aquellos que parecen más relevantes y apuntar toda la información que puede resultar interesante.
Diseñar estructura a seguir	15	A partir de la documentación recopilada definir cuál será a estructura del proceso.
Definir pasos necesarios	22	Definir todos los pasos de la estructura y crear los diagramas de actividad para cada uno de ellos.
Describir y adaptar todos los pasos	32	Describir cada una de las actividades que se realizarán e iterar otra vez los dos puntos anteriores, para buscar mejoras en el proceso.
Buscar información relacionada	15	Buscar definiciones e información interesante para complementar el *proceso de pruebas*.

TABLA 59 ESTUDIO ECONÓMICO DE APARTADO: FASES FUNDAMENTALES DEL PROCESO DE PRUEBAS

Podemos observar que la tarea más compleja ha sido describir y adaptar los pasos, que está relacionada con la creación de la estructura. Para ello se han trabajado estos puntos



de forma iterativa consiguiendo mejorar el proceso descrito.

ILUSTRACIÓN 37 GRAFICA DEL ESTUDIO ECONÓMICO DEL APARTADO: FASES FUNDAMENTALES DEL PROCESO DE PRUEBAS

7.2. Documentación para el proceso de pruebas

Las tareas realizadas para poder completar el capítulo "Documentación para el proceso de pruebas" y los tiempos, son los siguientes:

Tareas	Horas	Descripción
Buscar documentación	3	Leer los estándares de la IEEE, buscar documentación relacionada y documentarse sobre todo lo necesario.
Describir y estudiar el plan de pruebas	8	Revisar y estudiar todos los puntos del *plan de pruebas*, asociarlos con la fase de planificación y describirlos.
Implementar plan de pruebas	25	Aplicar el *plan de pruebas* en un proyecto real. Revisar cada uno de los puntos, descripciones y relaciones mientras se aplicaba.

TABLA 60 ESTUDIO ECONÓMICO DEL APARTADO: DOCUMENTACIÓN PARA EL PROCESO DE PRUEBAS

Se puede observar que todo el peso de este aparatado ha recaído sobre la implementación del *plan de pruebas*. Se ha realizado sobre un proyecto real y ha necesitado varias modificaciones hasta adaptarse correctamente a las necesidades del proyecto.

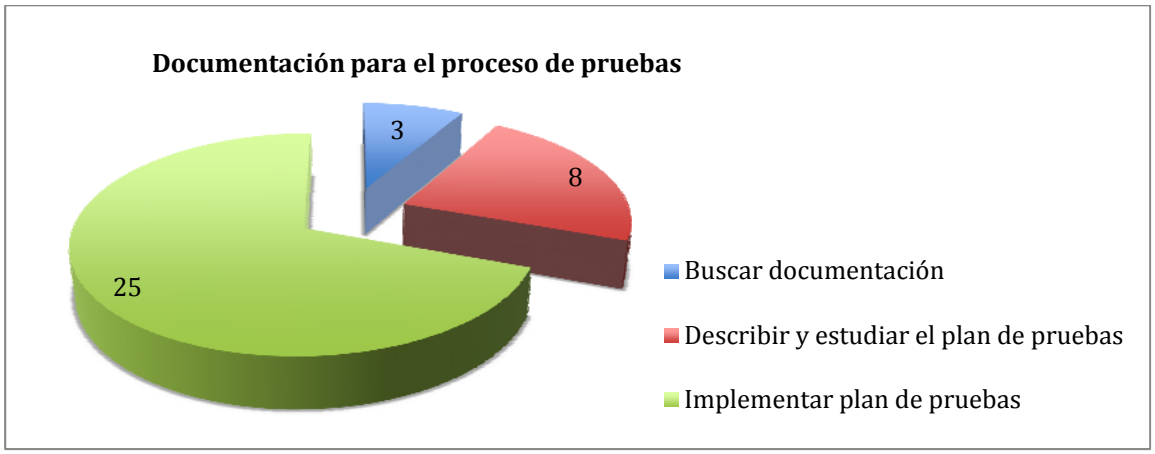


ILUSTRACIÓN 38 GRÁFICA DEL ESTUDIO ECONÓMICO DEL APARTADO: DOCUMENTACIÓN PARA EL PROCESO DE PRUEBAS

7.3. La calidad del Software

Las tareas realizadas para poder completar el capítulo "La calidad del SW" y el tiempo empleado, son:

Tareas	Horas	Descripción
Buscar información	5	ISO 9126 y otras lecturas complementarias.
Leer documentación	20	Leer toda la documentación relacionada con la ISO 9126, la calidad y las métricas.
Estudiar métricas	28	Leer, seleccionar, traducir y clasificar
Definir pasos necesarios	10	Definir cuándo es necesario validar cada una de las propiedades del software y relacionarlo con proyectos reales.
Describir y adaptar todos los pasos	21	Describir la documentación, las técnicas, las métricas y los casos
Aplicar en proyectos reales	15	Aplicar validaciones en proyectos reales

TABLA 61 ESTUDIO ECONÓMICO DEL APARTADO: LA CALIDAD DEL SOFTWARE

La tarea más difícil de este apartado ha sido la parte relacionada con las métricas. Realizar el estudio de todas las que recomienda la ISO 9126 y asociarlas con las pruebas, ha resultado complejo. Sintetizar toda la información y mostrarla de forma más útil ha requerido mucho tiempo, ya que se han realizado diversas iteraciones en cada uno de los puntos para mejorarlos.

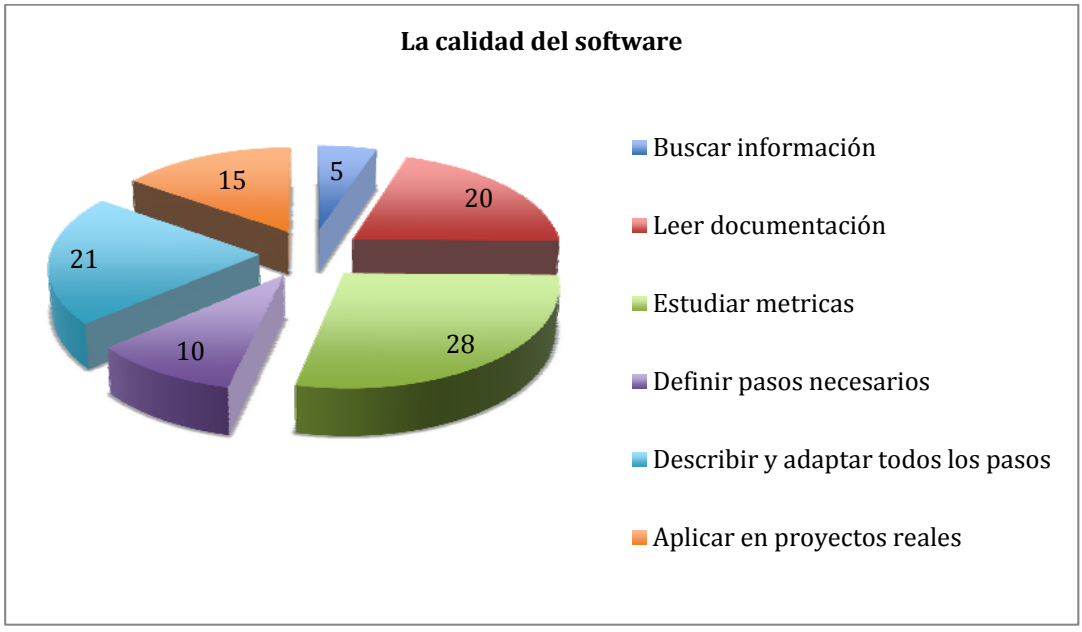


ILUSTRACIÓN 39 GRÁFICA DEL ESTUDIO ECONÓMICO DEL APARTADO: LA CALIDAD DEL SOFTWARE

7.4. Proceso de pruebas y otras tareas

Las tareas realizadas y los tiempos para poder completar los capítulos "Proceso de pruebas", "Planificación", "Conclusiones" y "Líneas de futuro" son los siguientes:

Tareas	Horas
Planificación	3
Definiciones relacionadas	12
Documentación relacionada	10
Introducción	8
Conclusiones y líneas de futuro	6

TABLA 62 ESTUDIO ECONÓMICO PARA LOS APARTADOS: PROCESO DE PRUEBAS Y PARA OTROS APARTADOS

Es difícil estimar exactamente los tiempos que han sido necesarios, porque estas tareas se han complementado todas las demás tareas del proceso.

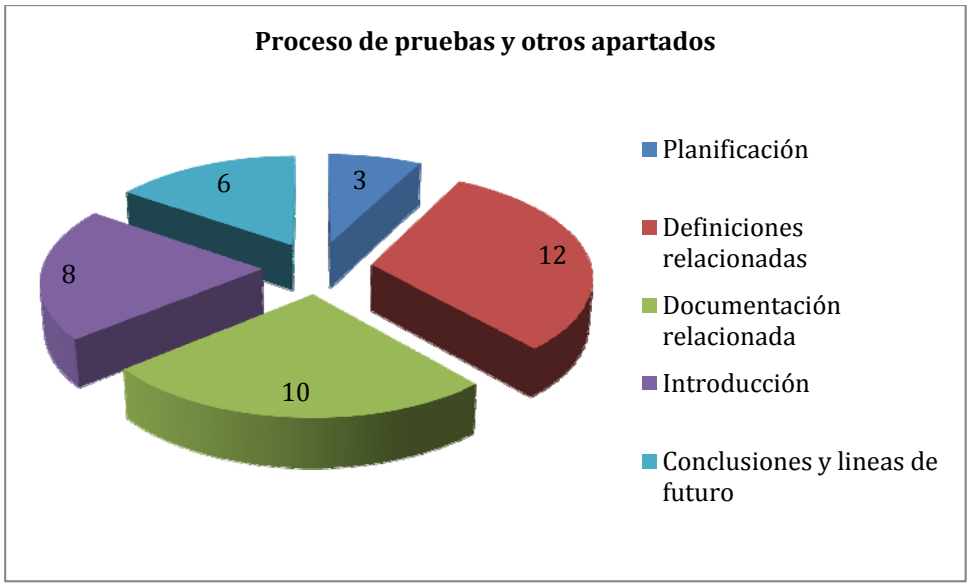


ILUSTRACIÓN 40 GRÁFICA DEL ESTUDIO ECONÓMICO PARA LOS APARTADOS: PROCESO DE PRUEBAS Y PARA OTROS APARTADOS

7.5. Total del proyecto

El resumen de todos los costes necesarios para completar el proyecto es el siguiente:

Tareas	Horas
Fases fundamentales del proceso de pruebas	109
Documentación para el proceso de pruebas	36
La calidad del SW	99
Proceso de pruebas y otros apartados	36
TOTAL	267

TABLA 63 ESTUDIO ECONÓMICO TOTAL DEL PROYECTO

Como se puede observar en el siguiente gráfico, las tareas que han necesitado una mayor inversión de horas han sido aquellas relacionadas con la *calidad del software* y las fases fundamentales del *proceso de pruebas*. Aunque la información de los otros apartados está directamente relacionada con estas tareas y han compartido muchas de las horas invertidas.

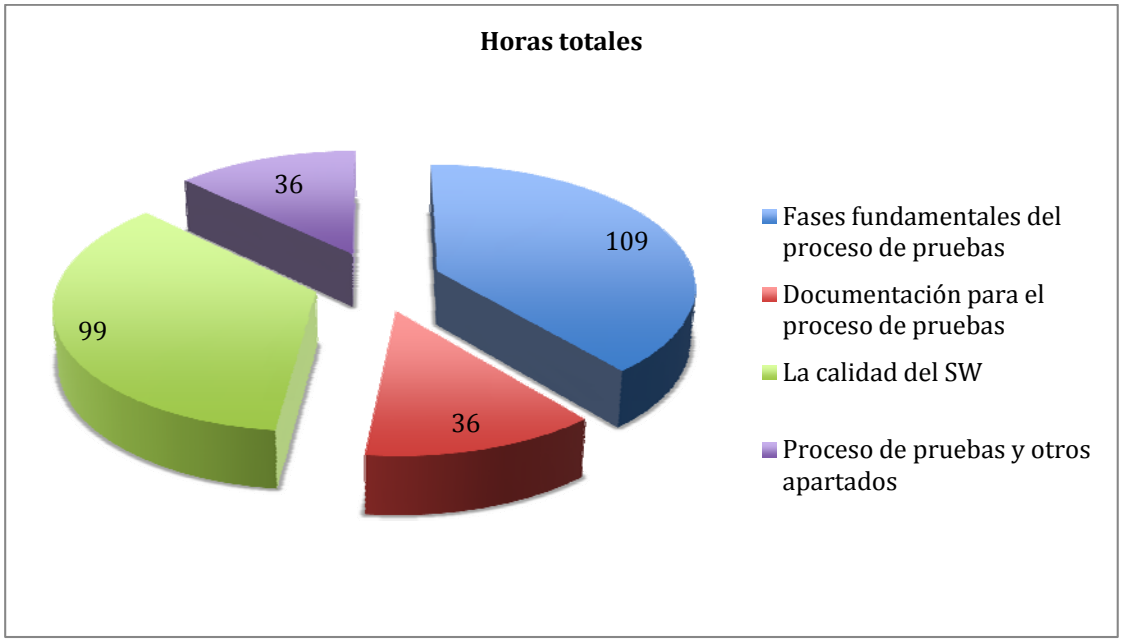


ILUSTRACIÓN 41 GRÁFICA DEL ESTUDIO ECONÓMICO TOTAL DEL PROYECTO

8. Conclusiones

La idea inicial de este proyecto era conocer más a fondo en qué consiste la *calidad del software*, y proponer un *proceso de pruebas* práctico y útil, aplicable en una empresa, que permitiera verificar la calidad en todas sus implementaciones.

La *calidad del software* es, en la actualidad, un mundo poco conocido. Por esta razón la mayoría de empresas no tienen procesos orientados a la calidad, o no los usan correctamente. Generalmente es difícil justificar esta fase ante un cliente, ya que no son conscientes de su necesidad y simplemente ven un incremento de costes y de tiempo. Además, la fase de pruebas, al ser el último paso del proyecto, se ve perjudicada por los retrasos en las demás fases, y se acaba reduciendo, o incluso eliminando, para cumplir los tiempos pactados.

Pero los defectos existen, y acaban apareciendo cuando el sistema se ha puesto en producción. Como consecuencia de ello, el usuario se encuentra con un sistema defectuoso y ha de reportar fallos al equipo de producción. El resultado final es un cliente insatisfecho y varias incidencias pendientes de solucionar.

Por este motivo son necesarios los *procesos de pruebas* dentro de un desarrollo. Éstos nos permiten mejorar la satisfacción del cliente, y conseguir un sistema robusto y útil para los usuarios.

El mundo de la calidad es amplio y desconocido, y al principio resulta complicado adaptarlo al funcionamiento de una empresa. Por eso es recomendable hacerlo de una forma incremental, permitiendo que las personas se adapten paulatinamente.

Además, encontrar información práctica sobre la *calidad del software* acostumbra a ser complicado, ya que casi toda la información que existe es teórica. Y es difícil ver cuáles son los primeros pasos que se deben realizar.

Cuando se busca documentación relacionada con la calidad, se encuentra un campo extenso y desconocido, con exceso de recomendaciones que generalmente no se adaptan realmente a las necesidades que se quieren cubrir. Por este motivo hay que leer mucho, y seleccionar aquellas prácticas que pueden adaptarse a los procesos ya definidos dentro de la empresa u organización.

Durante la realización de este proyecto me he dado cuenta de que existe mucha información interesante en la que me hubiera gustado profundizar, pero conseguir abarcarlo todo es un trabajo que necesita muchos años, mucho estudio y mucha experiencia.

Por ello se ha redactado un documento útil y concreto que permite implantar un *proceso de pruebas* dentro de una empresa. Además se ha conseguido llegar un poco más lejos gracias a la aportación de unas herramientas y unas técnicas, útiles y prácticas, que mejoraran la calidad de todo este proceso.

9. Líneas de futuro

Las líneas de futuro de este proyecto pueden ser muchas, pero creo que lo más interesante es ampliar y adaptar poco a poco la información introducida en los diferentes capítulos del trabajo a las necesidades de los proyectos.

Se puede empezar completando la documentación que se usa a lo largo de todo el *proceso del software*. Por ejemplo, crear plantillas para definir los casos de prueba y los procedimientos de los mismos, o trabajar la documentación para informar del estado del proceso.

Una vez definida esta documentación se debe revisar el *proceso de pruebas* aquí descrito para acabar de adaptarlo a la información que necesitan estos documentos.

A nivel de calidad se pueden estudiar y ampliar todos los atributos relacionados con la eficiencia, la fiabilidad, la usabilidad, la portabilidad y el mantenimiento del software, para ir mejorando la calidad de los sistemas en todos los aspectos.

Como ayuda al proceso, se pueden buscar herramientas que permitan mejorar aspectos como la gestión y el control de los procesos, la automatización de las pruebas, la validación y verificación de los atributos, etc. Actualmente hay infinidad de opciones en el mercado, algunas libres y otras propietarias. Sería interesante realizar un estudio para saber cuáles son las que mejor se adaptan a nuestras necesidades.

Finalmente se debe realizar una revisión de toda la información aquí descrita después de haberla aplicado en varios proyectos reales para poder mejorarla y adaptarla a las necesidades que vayan apareciendo.

10.Bibliografía

Libros/Artículos

International Organization for Standardization. *Software engineering – Product quality-Part 1: Quality model. ISO/IEC 9126-1.* Canada: 2001. Reference number: ISO/IEC 9126-1:2001(E)

International Organization for Standardization. *Software engineering – Product quality-Part 2: External metrics. ISO/IEC 9126-2.* Japan: 2002. Reference number: ISO/IEC 9126-2:2002(E)

International Organization for Standardization. *Software engineering – Product quality-Part 3: Internal metrics. ISO/IEC 9126-3.* Japan: 2002. Reference number: ISO/IEC 9126-3:2002(E)

Institute of Electrical and Electronics Engineers. *IEEE Standard for Software Quality Assurance Plans IEEE std 730-1998.* New York: Institute of Electrical and Electronics Engineers, 1998. ISBN: 0-7381-0328-4

Institute of Electrical and Electronics Engineers. *IEEE Standard for Software Test Documentation IEEE std 829-1998.* New York: Institute of Electrical and Electronics Engineers, 1998. ISBN: 0-7381-1443-X

Spillner, Andrea. Linz, Tilo. Schaefer, Hans. *Software Testing Fundation.* Santa Barbara: Rocky Nook Inc. 2007. ISBN: 978-1-933952-08-6

Oskarsson Ö, Glass R.L. *An ISO 9000 approach to building Quality Software*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc. 1995. ISBN: 0-13-228925-3

Documentos electrónicos

Real Academia Española. *Calidad* [en línea]. LA 22ª EDICIÓN. 2001
[Consulta: 30 marzo 2010] Disponible en:
<http://buscon.rae.es/draeI/SrvltConsulta?TIPO_BUS=3&LEMA=calidad>

Consejo superior de administración electrónica. *MÉTRICA. VERSIÓN 3 Metodología de Planificación, Desarrollo y Mantenimiento de sistemas de información* [en línea]. España: Ministerio de Administraciones Públicas. [Consulta: 21 de febrero 2010 y 19 junio 2010] Disponible en: <http://www.csae.map.es/csi/metrica3/index.html>

Rational. *Rational Unified Process Best Practices for Software Development Teams* [en línea]. Cupertino: Rational Software. 1998 [Consulta: 21 de febrero 2010 y 19 junio 2010] Disponible en:
<http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf>

11.Índice de ilustraciones

12.Índice de tablas