**DOCUMENTACIÓN Y PRUEBAS EN EL DESARROLLO**

**TRADICIONAL DEL SOFTWARE**

La documentación se suele clasificar en función de las personas o grupos a los cuales está

Dirigida en:

Documentación para los desarrolladores

Documentación para los usuarios

Documentación para los administradores o soporte técnico

La documentación para desarrolladores es: aquélla que se utiliza para el propio desarrollo

del producto y, sobre todo, para su mantenimiento futuro. Se documenta para comunicar estructura y comportamiento del sistema o de sus partes, para visualizar y controlar la arquitectura del sistema, para comprender mejor el mismo y para controlar el riesgo, entre otras cosas. Todas las fases de un desarrollo deben documentarse: requerimientos, análisis, diseño, programación, pruebas, etc.. Una herramienta muy útil en este sentido es una notación estándar de modelado, de modo que mediante ciertos diagramas se puedan comunicar ideas entre grupos de trabajo. Un caso particular es el de UML. De todas maneras, los diagramas son muy útiles, pero siempre y cuando se mantengan actualizados. La documentación para desarrolladores a menudo es llamada **modelo**, pues es una simplificación de la realidad para comprender mejor el sistema como un todo.

La documentación para usuarios es: todo aquello que necesita el usuario para la

instalación, aprendizaje y uso del producto. Puede consistir en guías de instalación, guías del usuario, manuales de referencia y guías de mensajes.

En el caso de los usuarios que son programadores, verbigracia los clientes de nuestras

clases, esta documentación se debe acompañar con ejemplos de uso recomendados o de muestra y una reseña de efectos no evidentes de las bibliotecas. Buena parte de la documentación para los usuarios puede empezar a generarse desde que

comienza el estudio de requisitos del sistema. Esto está bastante en consonancia con las ideas de *extreme programming* y con metodologías basadas en casos de uso.

La documentación para administradores o soporte técnico: a veces llamada manual de

operaciones, contiene toda la información sobre el sistema terminado que no hace al uso por un usuario final. Es necesario que tenga una descripción de los errores posibles del sistema, así como los procedimientos de recuperación. Como esto no es algo estático, pues la aparición de nuevos errores, problemas de compatibilidad y demás nunca se puede descartar, en general el manual de operaciones es un documento que va engrosándose con el tiempo.

**LAS PRUEBAS EN EL DESARROLLO DE SOFTWARE**

***Calidad, errores y pruebas***

La calidad no es algo que se pueda agregar al software después de desarrollado si no se

hizo todo el desarrollo con la calidad en mente. Tiene que ver con la corrección, pero también con usabilidad, costo, consistencia, confiabilidad, compatibilidad, utilidad, eficiencia y apego a los estándares. Todos estos aspectos de la calidad pueden ser objeto de tests o pruebas que determinen el grado de calidad. Incluso la documentación para el usuario debe ser probada. Desde el punto de vista de la programación, nos interesa la ausencia de errores(corrección), la confiabilidad y la eficiencia. Un **error** es un comportamiento distinto del que espera un usuario razonable. Puede haber errores aunque se hayan seguido todos los pasos indicados en el análisis y en el diseño, y hasta en los requisitos aprobados por el usuario. De allí la importancia del desarrollo incremental, que permite ir viendo versiones incompletas del sistema. Por lo tanto, una primera fuente de errores ocurre antes de los requerimientos o en el propio proceso de análisis. Pero también hay errores que se introducen durante el proceso de desarrollo posterior. Así, puede haber errores de diseño y errores de implementación. Finalmente, puede haber incluso errores en la propia etapa de pruebas y depuración.

***Categorías de pruebas***

Según la naturaleza de lo que se esté controlando, las pruebas se pueden dividir en dos

categorías:

Pruebas centradas en la verificación

Pruebas centradas en la validación

Las primeras sirven para determinar la consistencia entre los requerimientos y el programa

terminado. Soporta metodologías formales de testeo, de mucho componente matemático. La verificación consiste en determinar si estamos construyendo el sistema correctamente, a partir de los requisitos.

La validación consiste en saber si estamos construyendo el sistema correcto. Las tareas de validación son más informales. Las pruebas suelen mostrar la presencia de errores, pero nunca demuestran su ausencia.

***Las pruebas y el desarrollo de software***

La etapa de pruebas es una de las fases del ciclo de vida de los proyectos. Se la podría

ubicar después del análisis, el diseño y la programación, pero dependiendo del proyecto en cuestión y del modelo de proceso elegido, su realización podría ser en forma paralela a las fases citadas o inclusive repetirse varias veces durante la duración del proyecto. La importancia de esta fase será mayor o menor según las características del sistema

desarrollado. Las pruebas no tienen el objeto de prevenir errores sino de detectarlos5. Se efectúan sobre el trabajo realizado y se deben encarar con la intención de descubrir la mayor cantidad de errores posible.

A veces se dan por terminadas las pruebas antes de tiempo. Las pruebas deben ser bien planificadas, con suficiente anticipación, y determinar desde el comienzo los resultados que se deben obtener. La idea de *extreme programming* es más radical: propone primero escribir los programas de prueba y después la aplicación, obligando a correr las pruebas siempre antes de una integración. Se basa en la idea bastante acertada de que los programas de prueba son la mejor descripción de los requerimientos.

***Tipos de pruebas***

Analizaremos 5 tipos de pruebas:

Revisiones de código

Pruebas unitarias

Pruebas de integración

Pruebas de sistema

Pruebas de aceptación

No son tipos de pruebas intercambiables, ya que testean cosas distintas.

Otra posible clasificación de las pruebas es:

De caja blanca o de código

De caja negra o de especificación

En las primeras se evalúa el contenido de los módulos, mientras en las segundas se trata

al módulo como una caja cerrada y se lo prueba con valores de entrada, evaluando los valores de salida. Vistas de este modo, las pruebas de caja negra sirven para verificar especificaciones.

Las pruebas unitarias suelen ser de caja blanca o de caja negra, mientras que las de

integración, sistema y aceptación son de caja negra. Las tareas de depuración luego de encontrar errores son más bien técnicas de caja blanca, así como las revisiones de código.

En todos los casos, uno de los mayores desafíos es encontrar los datos de prueba: hay que encontrar un subconjunto de todas las entradas que tengan alta probabilidad de detectar el mayor número de errores.

***Revisiones de código***

Las revisiones de código son las únicas que se podrían omitir de todos los tipos de

pruebas, pero tal vez sea buena idea por lo menos hacer alguna de ellas:

Pruebas de escritorio

Recorridos de código

Inspecciones de código

La prueba de escritorio rinde muy poco, tal vez menos de lo que cuesta, pero es una

costumbre difícil de desterrar. Es bueno concentrarse en buscar anomalías típicas, como variables u objetos no inicializados o que no se usan, ciclos infinitos y demás.

Los recorridos rinden mucho más. Son exposiciones del código escrito frente a pares. El

programador, exponiendo su código, encuentra muchos errores. Las llamadas inspecciones de código consisten en reuniones en conjunto entre los responsables de la programación y los responsables de la revisión. Tienen como objetivo revisar el

código escrito por los programadores para chequear que cumpla con las normas que se hayan fijado y para verificar la eficiencia del mismo.

***Pruebas unitarias***

Las pruebas unitarias se realizan para controlar el funcionamiento de pequeñas porciones

de código como ser subprogramas (en la programación estructurada) o métodos (en POO). Generalmente son realizadas por los mismos programadores puesto que al conocer con mayor detalle el código, se les simplifica la tarea de elaborar conjuntos de datos de prueba para testearlo. El tipo de prueba a la cual se someterá a cada uno de los módulos dependerá de su complejidad. (Los métodos de cobertura de caja blanca tratan de recorrer todos los caminos posibles por lo menos una vez, lo que no garantiza que no haya errores pero pretende encontrar la mayor parte.). Si se pretende realizar una prueba estructurada, se puede confeccionar un grafo de flujo con la lógica del código a probar.

***Pruebas de integración***

En el caso de las pruebas de integración y de sistema, dado que ya se han realizado las

pruebas unitarias, se tomará a cada uno de los módulos unitarios como una caja negra.

Si bien se realizan sobre módulos ya probados en forma individual, no es necesario que se terminen todas las pruebas unitarias para comenzar con las de integración.

Existen principalmente dos tipos de integración: La integración incremental y la no

incremental. La integración incremental consiste en combinar el conjunto de módulos ya probados (al principio será un conjunto vacío) con los siguientes módulos a probar. Luego se va incrementando progresivamente el número de módulos unidos hasta que se forma el sistema completo. En la integración no incremental o Big Bang se combinan todos los módulos de una vez. Por el hecho de poder ser llevada a cabo por distintos caminos, la integración incremental brinda una mayor flexibilidad en el uso de recursos.

En la **integración incremental ascendente** se comienza integrando primero los módulos

de más bajo nivel. El proceso deberá seguir los siguientes pasos:

Elegir los módulos de bajo nivel que se van a probar.

Escribir un módulo impulsor para la entrada de datos de prueba a los módulos y para

la visualización de los resultados.

Probar la integración de los módulos.

Eliminar los módulos impulsores y juntar los módulos ya probados con los módulos

de niveles superiores, para continuar con las pruebas.

La **integración incremental descendente** parte del módulo de control principal (de mayor

nivel) para luego ir incorporando los módulos subordinados progresivamente. No hay un

procedimiento considerado óptimo para seleccionar el siguiente módulo a incorporar. La única regal es que el módulo incorporado tenga todos los módulos que lo invocan previamente probados.

El método de **integración incremental sándwich** combina los métodos

ascendente y descendente. Consiste en integrar una parte del sistema en forma ascendente y la restante en forma descendente, provocando la unión de ambas partes en algún punto intermedio. La principal ventaja es que nos da mayor libertad para elegir el orden de integración de los módulos según las características específicas del sistema en cuestión.

La **integración no incremental** puede ser beneficiosa para la prueba de sistemas de

pequeñísima envergadura cuya cantidad de módulos sea muy limitada y la interfaz entre los mismos sea clara y sencilla. Consiste en integrar todos los módulos del sistema a la vez e ingresar los valores de prueba para testear todas las interfaces.

***Pruebas de sistema***

Las pruebas de sistema se realizan una vez integrados todos los componentes. Su objetivo es ver la respuesta del sistema en su conjunto, frente a distintas situaciones. Se simulan varias alternativas que podrían darse con el sistema implantado y en base a ellas se prueba la eficacia y eficiencia de la respuesta que se obtiene. Se pueden distinguir varios tipos de prueba distintos, por ejemplo:

Pruebas negativas: se trata de que el sistema falle para ver sus debilidades.

Pruebas de recuperación: se simulan fallas de software y/o hardware para verificar la

eficacia del proceso de recuperación.

Pruebas de rendimiento: tiene como objeto evaluar el rendimiento del sistema

integrado en condiciones de uso habitual.

Pruebas de instalación: verifican que el sistema puede ser instalado

satisfactoriamente en el equipo del cliente, incluyendo todas las plataformas y

configuraciones de hardware necesarias.

***Pruebas de aceptación***

Las pruebas de aceptación, al igual que las de sistema, se realizan sobre el producto

terminado e integrado; pero a diferencia de aquellas, están concebidas para que sea un usuario final quien detecte los posibles errores.

Se clasifican en dos tipos: pruebas Alfa y pruebas Beta.

Las **pruebas Alfa** se realizan por un cliente en un entorno controlado por el equipo de

desarrollo.

las **pruebas Beta** se realizan en las instalaciones propias de los clientes. Para que tengan lugar, se deben distribuir copias del sistema para que cada cliente lo instale en sus oficinas, dependencias y/o sucursales, según sea el caso.

**REACCIÓN ANTE LOS RESULTADOS DE LAS PRUEBAS**

Las pruebas nos llevan a descubrir errores, que en la mayoría de los casos son de tipo

funcional, es decir, del tipo: “el sistema debería hacer tal cosa y hace tal otra”

***Depuración***

La depuración es la corrección de errores que sólo afectan a la programación, porque no

provienen de errores previos en el análisis o en el diseño. A veces la depuración se hace luego de la entrega del sistema al cliente y es parte del mantenimiento.

Existen herramientas de depuración (en inglés, *debugging*) de los propios ambientes de desarrollo que facilitan esta tarea.

Las pruebas de integración, de sistema y de aceptación también pueden llevar a que sea

necesaria una depuración, aunque aquí es más difícil encontrar el lugar exacto del error. Por eso a menudo se utilizan *bitácoras* (*logs*, en inglés), que nos permiten evaluar las condiciones que se fueron dando antes de un error mediante el análisis de un historial de uso del sistema que queda registrado en medios de almacenamiento permanente.

La depuración se hace en cuatro pasos:

Reproducir el error.

Diagnosticar la causa.

Corregirla.

Verificar la corrección.