# 2- Fiabilidad y tolerancia a fallos

## 1. Fiabilidad, fallos y defectos

Los sistemas están formados generalmente por componentes que son, a su vez, otros sistemas que dan lugar a defectos por especificaciones inadecuadas, errores de diseño en los componentes software, fallo en componentes del procesador o por interferencia transitoria o permanente en el subsistema de comunicación.


Hay que distinguir los siguientes conceptos:

- **Fiabilidad**: es la medida del éxito con la que el sistema se ajusta a alguna especificación definitiva de su comportamiento (consistente, comprensible y no ambigua). En un STR deben tenerse en cuenta los tiempos de respuesta como parte de la especificación.
- **Fallo**: se producen cuando el comportamiento de un sistema se desvía de su especificación. Son el resultado de problemas internos no esperados que el sistema manifiesta eventualmente en su comportamiento externo
- **Errores**: son la manifestación externa de un problema interno del sistema. Suceden cuando el comportamiento del sistema no se ajusta a la especificación dada.
- **Defectos**: son las causas mecánicas o algorítmicas por las que se produce un error.

**Tipos de fallos**

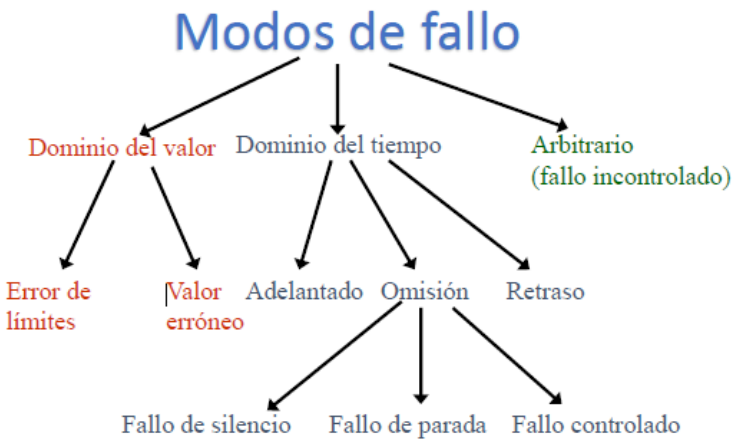
- **Transitorios**: se producen sólo durante un intervalo de tiempo (por ejemplo, fallo en componentes hardware debidos a interferencias de elementos externos).
- **Permanentes**: comienzan en un instante de tiempo y permanecen en el sistema hasta que son reparados (un cable roto, un error en el diseño del software, etc.).
- **Intermitentes**: fallos transitorios que ocurren de vez en cuando (por ejemplo, un fallo que se produce en un componente ante ciertas circunstancias periódicas).

En el caso del software, éste no se deteriora con el tiempo y los fallos (bugs) aparecen ante determinadas condiciones específicas reproducibles. En general, los **fallos en el software** pueden permanecer dormidos durante mucho tiempo.

## 2. Modos de fallo

Un sistema proporciona servicios y, por tanto, puede fallar de maneras diferentes. Se pueden clasificar los modos de fallo de un sistema de acuerdo al impacto que tendrá en los servicios que proporciona:

- **Fallos de valor**: el valor asociado al servicio es erróneo (**valor erróneo**). Los fallos de valor pueden derivar en un **error de límites**, ya que podría estar aún dentro del rango correcto de valores o fuera de ese rango esperado para ese servicio.
- **Fallos de tiempo**: el servicio se completa a destiempo:
  - Demasiado pronto (**adelantado**).
  - Demasiado tarde (**retraso**): deriva en un error de prestaciones.
  - Infinitamente tarde (**omisión**): puede derivar en un fallo de silencio (el sistema servicios correctos hasta que falla, junto con los sistemas siguientes), de parada (el fallo es detectado por otros sistemas relacionados con él) o controlado (falla de una forma específica).
- **Arbitrarios**: son fallos incontrolados combinación de fallos de valor y de tiempo.




## 3. Prevención de fallos y tolerancia a fallos

La **prevención de fallos** tiene como objetivo eliminar cualquier posibilidad de ocurrencia de fallos antes de que el sistema entre en funcionamiento. En contraparte, la **tolerancia a fallos** permite que un sistema siga funcionando, aunque exista presencia de fallos. Ambos enfoques pretenden producir sistemas fiables con modos de fallo bien definidos.

En la prevención de fallos existen dos fases:

- **Evitación de fallos**: limita la introducción de componentes potencialmente defectuosos durante la construcción del sistema. Para conseguirlo se realiza:
  - Utilización de los componentes más fiables dentro de las restricciones de coste y prestaciones dadas.
  - Utilización de técnicas exhaustivamente refinadas para la interconexión de componentes y el ensamblado de subsistemas.
  - Aislamiento del hardware para protegerlo frente a interferencias.
  - Especificación de requerimientos rigurosos y si no, formales.
  - Utilización de metodologías de diseño que hayan sido probadas.
  - Utilización de lenguajes que faciliten la abstracción de datos y modularidad.
  - Uso de herramientas de ingeniería de software para ayudar en la manipulación de los componentes software y en la gestión de la complejidad.
- **Eliminación de fallos**: consiste en la puesta en marcha de procedimientos para encontrar y eliminar las causas de los errores, tanto hardware como software.

Las pruebas de ejecución para prevenir fallos nunca pueden ser lo suficientemente exhaustivas y, por tanto, no eliminan todos los potenciales fallos, ya que solo se pueden utilizar para demostrar la presencia de fallos, no su ausencia. Además, resulta imposible realizar pruebas bajo condiciones reales. La mayoría de las pruebas son realizadas con el sistema en modo simulación, y es difícil garantizar que esta sea precias. Los errores introducidos en la etapa de requisitos del sistema puede que no se manifiesten hasta que el sistema esté operativo.

A pesar de todas las técnicas de prueba y verificación, como los componentes hardware pueden fallar la aproximación basada en la prevención de fallos resultará inapropiada:

- Cuando la frecuencia o la duración de los tiempos de reparación resulten inaceptables.
- Cuando no se pueda acceder al sistema para actividades de mantenimiento y reparación.

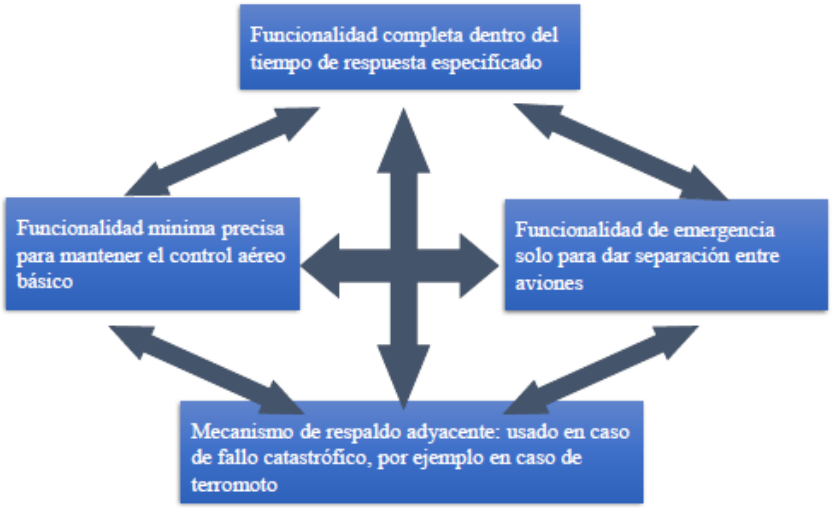
**Tolerancia a Fallos**

La tolerancia a fallos permite que un sistema siga funcionando incluso ante la presencia de fallos. Existen diferentes niveles de tolerancia a fallos:

- **Tolerancia total**: el sistema continúa operativo durante un tiempo limitado ante la presencia de fallos sin que éste tenga una pérdida significativa de funcionalidad o prestaciones.
- **Degradación controlada**: el sistema continúa operativo ante la presencia de fallos donde, además, se acepta una degradación parcial de la funcionalidad o de las prestaciones hasta que se produzca la recuperación o reparación.

- **Fallo seguro**: el sistema cuida de su integridad durante el fallo aceptando una parada temporal de su funcionamiento.

El nivel requerido en cada caso dependerá de la aplicación. Los sistemas más críticos si exigirán una tolerancia total frente a fallos.




Hay que tener en cuenta una serie de consideraciones previas de los sistemas tolerantes a fallos:

- Los algoritmos del sistema han sido diseñados correctamente
- Se conocen todos los posibles modos de fallos de los componentes
- Se han tenido en cuenta todas las posibles interacciones entre el sistema y el entorno

Además, la creciente complejidad del software y hardware hace necesario considerar tanto los fallos previstos como imprevistos, incluyendo los fallos de diseño software y hardware.

**Redundancia**

Todas las técnicas utilizadas para conseguir tolerancia a fallos se basan en añadir elementos extra al sistema para que detecte y se recupere de los fallos. Estos componentes son redundantes, ya que no son necesarios para el normal funcionamiento del sistema (**redundancia protectora**).

El objetivo de la tolerancia de fallos en este sentido es minimizar la redundancia maximizando la fiabilidad proporcionada, siempre bajo las restricciones de coste y tamaño del sistema, ya que los elementos añadidos incrementan la complejidad resultante del sistema y puede conducir a sistema menos fiables. Por ello, resulta aconsejable separar los componentes tolerantes a fallos del resto del sistema.

**Tolerancia a fallos hardware**

Abarca aspectos de redundancia estática y dinámica:

- **Estática** los componentes redundantes son utilizados dentro de un sistema (o subsistema) para ocultar los efectos de los fallos. Un ejemplo es la Redundancia Triple Modular (TMR) que consiste en tres subcomponentes idénticos y en circuitos de votación por mayoría, asumiendo que el fallo no es común y que es de naturaleza transitoria o por deterioro. Para ocultar fallos en más de un componente se requiere una redundancia N modular (NMR).
- **Dinámica** redundancia proporcionada dentro de un mismo componente que indica si la salida es errónea a través de un mecanismo de detección de errores. La recuperación se proporciona entonces por otro componente.

**Tolerancia a fallos software**

Los sistemas deben reconfigurarse, habitualmente haciendo funcionar algún tipo de componente redundante que pueda sustituir al componente causante del fallo, por lo que debe conseguirse que el sistema sea tolerante a fallos. Para conseguir dicha tolerancia, se suele recurrir a los enfoques:

- **Estático**: programación de N versiones

- **Dinámico**:
    - Detección y recuperación
    - Bloques de recuperación (recuperación hacia atrás).
    - Excepciones (recuperación hacia delante).

**Tratamiento de fallos y servicio continuado**

Un error es una manifestación de un defecto. Aunque los procesos de recuperación lleven al sistema a un estado consistente, el error puede volver a producirse. Así, la fase final de la tolerancia a fallos es erradicar el fallo del sistema. El tratamiento automático de los fallos resulta difícil de implementar y tiende a ser específico de cada sistema. Algunos sistemas no disponen de tratamiento de fallos, asumiendo que todos los fallos son transitorios. Otros suponen que las técnicas de recuperación bastan para recuperase de los fallos recurrentes.

Hay dos fases del tratamiento de fallos, la **localización del fallo** y la **reparación del sistema**. Las técnicas de detección de errores pueden ayudar a realizar un seguimiento del fallo de un componente (los componentes hardware pueden ser reemplazados); pero los fallos de software pueden ser eliminados en una nueva versión del código. Además, en las aplicaciones que no se pueden detener, es necesario modificar el software mientras se está ejecutando.

## 4. Programación de N versiones

El esquema de programación de N-versiones consta de N programas funcionalmente equivalentes a partir de la misma especificación inicial y un mecanismo de votación. Todos los N programas alternativos, desarrollados sin interacciones entre los grupos de diseño, se ejecutan simultáneamente y de forma independiente con las mismas entradas, tal que sus resultados son comparados a través de un **programa director**. Los resultados deberían ser idénticos, pero puede haber diferencias, escogiéndose una salida correcta por consenso.


Para comparar resultados se utiliza procesamiento con texto o con números enteros (buscando producir resultados idénticos) o procesamiento de números reales (lo que puede producir resultados diferentes). Cuando se producen resultados diferentes, se usan técnicas de votación inexacta. La **votación inexacta** incluye técnicas utilizadas en la comparación de los tipos de resultados que producen diferentes versiones. Una técnica es la comparación de rangos, utilizando una estimación previa, o utilizar un valor promedio de los N resultados. Es difícil encontrar un enfoque general para la votación inexacta.

Algunos aspectos a tener en cuenta en la programación de N versiones son:

- **Especificación inicial**: la mayoría de los fallos en el software provienen de una especificación inadecuada. Un error de especificación se manifestará en todas las N versiones de la implementación.
- **Independencia en el diseño**: una especificación compleja puede conducir a problemas de interpretación de los requerimientos. Si se contemplan datos de entrada poco frecuentes, pueden no detectarse todos los errores en la fase de pruebas.
- **Presupuesto inadecuado**: en la mayoría de los sistemas embebidos, el principal coste se debe al software, por lo que un sistema de N versiones multiplicaría el presupuesto, así como problemas de rendimiento. En un entorno competitivo, es raro que un cliente potencial proponga la técnica de la programación de N-Versiones (a menos que sea obligatorio). Además, no está claro que se pueda producir un sistema más fiable si los recursos potencialmente disponibles para construir N versiones fueran dedicados a construir una única versión.

## 5. Comparación entre la programación de N versiones y los bloques de recuperación

Tanto la programación N versiones como los bloques de recuperación comparten aspectos en su filosofía básica, pero presentan diferencias:

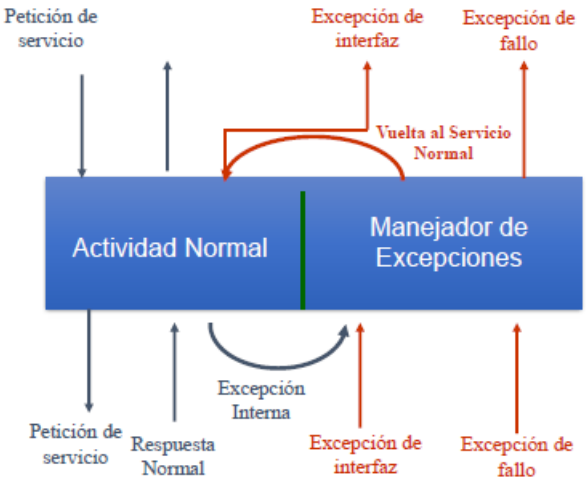
- **Redundancia estática frente a dinámica**: la programación N versiones se basa en la redundancia estática (todas las versiones corren en paralelo, independientemente de si se producen o no fallos), mientras que los bloques de recuperación son dinámicos (los módulos alternativos sólo se ejecutan cuando ha sido detectado algún error).
- **Sobrecargas asociadas**:
  - Mayor coste de desarrollo, al requerir ambos la creación de algoritmos alternativos. La programación N versiones requiere diseñar el proceso conductor y los bloques de recuperación el test de aceptación.
  - Procesamiento extra en tiempo de ejecución. La programación N-Versiones necesita N veces los recursos de una única versión. Aunque los bloques de recuperación sólo necesitan un conjunto de recursos en un instante de tiempo, requiere gestionar los puntos de recuperación y la restauración del estado de la ejecución.
- **Diversidad en el diseño**: ambas aproximaciones exploran la diversidad en el diseño para conseguir la tolerancia de errores no previstos, siendo ambas susceptibles de contener errores originados en la especificación de requisitos.
- **Detección de errores**: la programación N versiones utiliza la comparación de votos para detectar los errores, mientras que en los bloques de recuperación se utiliza un test de aceptación.
- **Atomicidad**: la programación N versiones elude el problema de la recuperación de errores hacia atrás (cuando no se pueden deshacer los errores que se hayan podido producir en el entorno) ya que, se supone, las versiones no interfieren entre ellas (son atómicas). L os bloques de recuperación deben estructurarse para producir una salida sólo cuando se pase el test de aceptación.

## 6. Redundancia dinámica y excepciones

Una **excepción** es la ocurrencia de un error. Cuando ocurre el hecho o condición de la excepción invocado por la operación que ha causado dicha excepción, se **genera (o lanza) la excepción**. Así, se **maneja la excepción** mediante un método de recuperación hacia delante, ya que no tiene vuelta a un estado anterior, en el que el control se transfiere a un manejador que se encarga de lanzar los procedimientos de recuperación.

Las excepciones y la gestión de excepciones se pueden utilizar como un mecanismo de gestión de errores de propósito general:

- Enfrentarse a condiciones anormales que surgen en el entorno.
- Tolerar los defectos en el diseño del programa.
- Proporcionar capacidades de propósito general para la detección y la recuperación de errores.



## 7. Medida y predicción de la fiabilidad del software

El software no se deteriora con el uso, por lo que su fiabilidad se determina mediante la probabilidad de que un programa dado funcione correctamente en un entorno concreto durante un determinado periodo de tiempo. Para ello, se utilizan diferentes modelos para estimar la fiabilidad del software:

- Modelos de crecimiento de la fiabilidad del software estimar la fiabilidad de un programa basándose en su historial de errores.
- Modelos estadísticos estimación de la fiabilidad de un programa mediante la realización de casos de prueba aleatorios.

## 8. Seguridad, Fiabilidad y Confiabilidad

- **Seguridad**: es la ausencia de condiciones que puedan causar muerte, lesión, enfermedad, daños o pérdidas de equipos o aspectos nocivos para el medio ambiente. La seguridad es así la probabilidad de que no se den las condiciones que conducen a un percance, independientemente de si se realiza la función prevista. Para garantizar los requisitos de seguridad de un sistema, el análisis de seguridad del sistema debe realizarse en todas las etapas del desarrollo de su ciclo de vida.
- **Fiabilidad**: es la medida del éxito con la que un sistema se ajusta a la especificación de su comportamiento. Se expresa habitualmente también en términos de probabilidad.
- **Confiabilidad**: es la propiedad del sistema que permite calificar justificadamente como es de fiable el servicio que proporciona.