Framework de diseño de software crítico de control de vuelo tolerante a fallas aplicado a vehículos espaciales.

Por ***Arias Emmanuel***

Presentado ante la Universidad Nacional de La Matanza y la Unidad de Formación Superior de la CONAE como parte de los requerimientos para la obtención del grado de

MAGISTER EN DESARROLLOS INFORMATICOS DE APLICACION ESPACIAL

UNIVERSIDAD NACIONAL DEL CÓRDOBA

Junio, 2016

DIRECTOR
Gustavo Wiman
INVAP, Bariloche, Provincia de Rio Negro

Dedicado a
alguien

Abstract

Keywords:

Resumen

Esta tesis se trata de

Agradecimientos

Muchas gracias

Tabla de Contenidos

Índice de figuras

Índice de tablas

Lista de acrónimos

SW Software

FT Tolerancia a Fallas

FA Evitación de Fallas

FR Eliminación de Fallas

FF Predicción de Fallas

MTBF Tiempo Medio Entre Fallas

MTTF Tiempo Medio de Fallas

MTTR Tiempo Medio de Reperación

CMF Modo Común de fallas

FDIR Detección, Aislación y Recuperación de Fallas

Capítulo 1

Introducción

En este trabajo bla bla bla

1.1. Motivación

La Motivación de este trabajo es tal tal y tal

1.2. Objetivo del trabajo y preguntas de investigación

El objetivo de este trabjo es tal tal y tal

Y se busca reponder las siguientes preguntas

bla bla bla

Capítulo 2

Marco Teórico

2.1. Terminología

Existe una importante diferencia entre los signficados de las palabras falla, error y fracaso[1], que es importante destacar antes de comenzar con el desarrollo de este trabajo.

Un **fracaso** de sistema ocurre cuando el servicio prestado por el sistema ya no coincide con las especificaciones del mismo (Hanmer, 2007). Esto quiere decir que existe un problema que tiene una consecuencia negativa en el sistema completa, logrando que este ya no logre cumplir con sus especificaciones. Cuando el sistema no se comporta de la manera que es especificada, este ha fracasado. Esto signfica que lo que se espera de un sistema se encuentra descripto, comúnmente en especificaciones o requerimientos (Pullum, 2001).

Para la IEEE (1990) fracaso es "la inhabilitación de una sistema o componente a llevar a cabo las funciones requeridas en los requerimientos específicos de perfomance del mismo".

Hanmer (2007) ejemplifica fracasos de sistemas cuando: el sistema se bloquea y se para cuando no debería hacerlo, el sistema calcula un resultado incorrecto, el sistema no está disponible, el sistema es incapaz de responder a la interacción con el usuario. Cuando el sistema no hace lo que debe hacer, el sistema a fracasado. Los fracasos son detectados por los usuarios mientras usan el sistema.

Los fracasos son causados por los errores. Un **error** es una parte del estado del sistema que es susceptible de provocar un fracaso en el sistema, un error que afecta al servicio es una indicación que un fracaso se ha producido (Hanmer, 2007). Un error se puede propagar, es decir dar a lugar otros errores (Pullum, 2001).

IEEE (1990) define error como "la diferencia entre un valor computado, observado o medido, con el valor verdadero, especificado o el teóricamente verdadero".

Los errores se pueden clasificar en dos tipos: errores de tiempo y valores (Hanmer, 2007). Los errores de valores son aquellos que se manifiestan como valores discretos incorrectos o estados del sistema incorrecto. En cambio, los errores de tiempo pueden incluir errores de no cumplimiento de total las tareas.

Hanmer (2007) especifica los siguiente casos más comunes de errores:

- Timing: existe una falta de sincronización en la comunicación de los procesos.
- Bucles infinitos: ejecución de un bucle sin detenerse, esto consume memoria, y el fracaso del sistema.

[1] En inglés: fault, error y failure.

- Error de protocolo: errores en el flujo de comunicación ya que no coinciden los protocolos. Mensajes enviados en formato diferente, en tiempos diferentes, a lugares de sistemas incorrectos.
- Inconsistencia de datos: errores son diferentes en diferentes lugares.
- Sobrecarga de sistema: el sistema es incapaz de hacer frente a la sobrecarga de actividades a la que es expuesta.

La causa adjudicada o la hipótesis de un error es una **falla**, también llamado "bugs". Una falla activa es aquella que produce un error (Pullum, 2001). Una falla es un defecto que está presente en el sistema y que puede causar un error (Hanmer, 2007). Es la desviación actual de lo correcto Hanmer (2007).

Según IEEE (1990) una falla es "un defecto en un dispositivo de hardware o componente; como por ejemplo un corto circuito o un cable cortado". También realiza una segunda definción diciendo que falla es "un paso incorrecto, proceso, o definción de dato en un programa de computadora" IEEE (1990). Esta última definción es la que se usa en el ámbito de este trabajo.

Algunas fallas introducidas en el Software (SW) se detallan en Hanmer (2007), lo cual señala que pueden incluir:

- Especificaciones incorrecta de requerimientos
- Diseño incorrecto
- Errores de programación

Entonces, como lo indica Pullum (2001) con la tolerancia a fallas, lo que se busca es prevenir el fracaso mediante la "tolerancia" de fallas, las cuales son detectables cuando un error aparece. Las fallas son el motivo de errores y los errores son motivos de fracaso (Dubrova, 2013).

También se suele utilizar el término anomalía en las operaciones de vehículos espaciales para referirse a comportamientos anómalos o no esperados del sistema (David M. Harland, 2005)

En Dubrova (2013) describe un ejemplo para diferenciar correctamente estos conceptos. Se considera el SW de una planta nuclear, en la cual existe una computadora que es responsable de controlar la temperatura, la presión y demás variables de interés para la seguridad del sistema. Se da el caso de que uno de los sensores detecta que la turbina principal se encuentra girando a una velocidad menor a la correcta. Esta falla hace que el sistema envíe una señal para aumentar su velocidad (error). Esto produce un exceso de velocidad en la turbina, lo cual tiene como consecuencia que la seguridad mecánica apague la turbina. En esta situación el sistema no está generando energía. Esto se considera un fracaso, porque el sistema no está entregando el servicio según lo establecido por los requerimientos. Pero es un fracaso salvable.

2.2. La fiabilidad en el software

El objetivo final de la Tolerancia a Fallas (FT), es el desarrollo de un sistema fiable (Dubrova, 2013). Teniendo en cuenta que SW que se encuentran dentro de las naves espaciales, como satélites, lanzadores, y sobre todo vehículos tripulados son críticos, ya que de ellos dependen el éxito o fracaso de una misión, se debe llevar a cabo un sistema fiable. La fiabilidad de un sistema es la capacidad del mismo de entregar a los usuarios un nivel deseado de servicio (Dubrova, 2013).

La fiabilidad también se la puede considerar como una propiedad global que permite justificar la confianza de los servicios de un sistema (Hitt y Mulcare, 2007). Por lo tanto, como lo indica Hitt y Mulcare (2007) la

fiabilidad es un término amplio y cualitativo que está relacionado con atributos no funcionales (o "-ilities"), que buscan generar un sistema "ideal", especialmente cuando su funcionamiento es crítico.

Como se muestra en la Figura 2.1 la consecuencia de la fiabilidad es la relación entre la evitación de fallos y la reducción de fallos, así como también la FT.

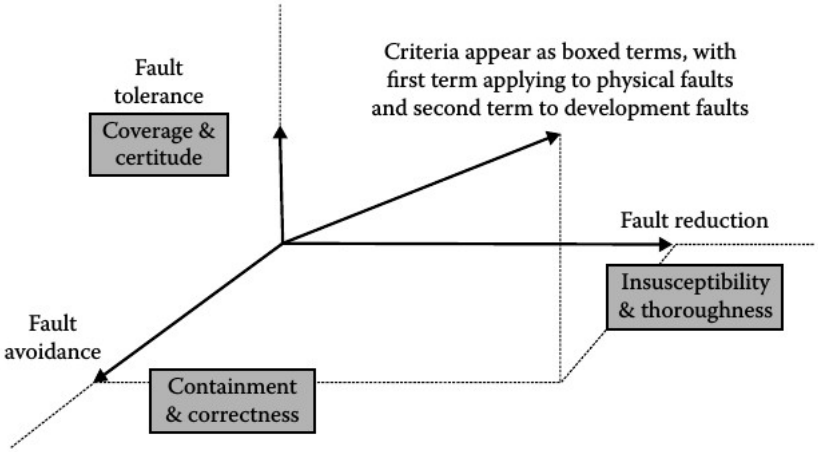

Figura 2.1: Fiabilidad (Hitt y Mulcare, 2007)

Según Pullum (2001) la fiabilidad puede ser clasificada en:

- Impedimentos: son aquellas cosas que se interponen en el camino de la fiabilidad. Son las fallas, errores y fracasos.
- Medios: los medios para lograr la fiabilidad, según el autor, se pueden dividir en dos grupos:
  1. Aquellos que son utilizados durante la construcción del SW (Evitación de Fallas (FA)[2] y FT).
  2. Aquellos que contribuyen con la validación del SW una vez desarrollado (Eliminación de Fallas (FR)[3] y Predicción de Fallas (FF)[4]).
- Atributos: describen las propiedades de la fiabilidad y proporcionan una forma de evaluar el logro de esas propiedades.

2.3. Impedimentos de la confiabilidad

El impedimiento de la confiabilidad o deteriodo de la confiabilidad es definido en términos de fallas, errores, y fracaso (Dubrova, 2013). Los mismos fueron desarrollados en la sección 2.1. Lo que tienen en común estos tres conceptos, es que avisan, o dan alerta cuando algo está mal (Dubrova, 2013). La diferencia radica, en que las fallas son a nivel físico; los errores se dan a nivel computacional; mientras que los fracaso se dan a nivel de sistema Dubrova (2013).

[2]En inglés, Fault Avoidance
[3]En inglés, Fault Removal
[4]En inglés, Fault Forecasting

2.3.1. Orígenes de la falla

Existen diversos orígenes de fallas. Estas pueden provenir desde terceros, en el caso de productos comprados, pueden deverse a una falta del conocimiento del problema, falta de tiempo, etc. Dubrova (2013) clasifica el orígen de las fallas en cuatro grupos: *especificación incorrecta, implementación incorrecta, defectos de fabricación y factores externos*

Las *especificaciones incorrectas* son aquellas que surgen debidas a una incorrecta especificación de requerimiento o un mal diseño de una arquitectura o de un algoritmo (Dubrova, 2013). Estos orígenes de fallas son bastante comúnes en el desarrollo de sistemas. Un ejemplo típico citado por Dubrova (2013), es el caso de requerimientos que ignoran aspectos del medio ambiente en el que opera el sistema. Una mala redacción de un requerimiento o el olvido de uno de ellos, puede traer graves problemas, atrasos y pérdida de dinero, en el diseño y producción de un sistema espacial.

Las *implementaciones incorrectas*, se refieren a las *fallas de diseño*, surgen cuando el sistema implementado no cumple con los requerimientos (Dubrova, 2013).

Otro orígen de falla son los *defectos de los componentes* (Dubrova, 2013).Estos pueden incluir defectos de fabricación, defectos aleatorios dados en los componentes, etc.

Y por último se tienen las fallas que son causadas por *factores externos*, los cuales provienen del medio ambiente, usuarios u operadores (Dubrova, 2013). Ejemplos de estos factores externos pueden ser, vibraciones, cargas electroestáticas, temperatura, radición electromagnética, envío incorrecto de comandos, etc.

2.3.2. Modos comúnes de fallas

Un *Modo Común de fallas (CMF)*[5] es una falla que ocurre simultaneamente en dos o más componentes redundantes (Dubrova, 2013).

Gangloff (1975) define los CMF como múltiples unidades de fracaso debido a una sola causa.

CMF son causados por fenómenos que crean dependencias entre unidades redundadas, lo que causa la falla de estas unidades simultaneamente (Dubrova, 2013).

Según como lo indica Dubrova (2013) el único enfoque para combatir los CMF, es mediante el diseño en diversidad. Diseño en diversidad es la implementación de más de una variante de la función en cuestión (Dubrova, 2013). Esto se puede lograr variando los algoritmos que se utilizan, diferentes equipos de trabajo realicen las mismas partes del sistema, de manera tal de tener redundancia en código, etc.

2.4. Medios de fiabilidad

Los medios de confiabilidad son métodos y técnicas que permiten el desarrollo de un sistema confiable (Dubrova, 2013). Los medios se pueden dividir en dos grandes grupos (Pullum, 2001):

1. Aquellos que son empleados durante el proceso de construcción del SW (Pullum, 2001),
2. y a quellos que ayudan en la validación del SW después que fue desarrollado (Pullum, 2001).

Dentro del primer grupo se tiene:

[5] En inglés, common-mode faults

- Evitación de Fallas
- Tolerancia a Fallas

Por otro lado, en el segundo grupo se puede mencionar los siguientes:

- Eliminación de Fallas
- Predicción de Fallas

2.4.1. Evitación de Fallas

FA son técnicas de mejoramiento de la fiabilidad utilizadas durante el desarrollo de SW para reducir el número de fallas introducidas durante la etapa mencionada (Pullum, 2001). Estas técnicas pueden estar presentes en las especificaciones y requerimientos del sistema, métodos de diseño de SW (Pullum, 2001).

Dubrova (2013) la denomina *prevensión de fallas*[6], y coincide con el autor anterior, definiendo FA como técnicas de control de calidad durante la especificación y fabricación de los procesos de diseño.

2.4.2. Tolerancia a Fallas

En la sección 2.6 (página 9) se discutirá con mayor detalle la FT. Esto es así ya que en este trabajo de tesis se tiene como principal punto de estudio la Tolerancia a Fallas.

2.4.3. Eliminación de Fallas

La FR hace referencia a las técnicas utilizadas para mejorar la fiabilidad empleadas durante la validación y verificación del SW (Pullum, 2001). Estas técnicas mejoran la fiabilidad del SW mediante la detección de fallas, usando métodos de verificación y la validación, y eliminando las fallas que se van detectando (Pullum, 2001).

Por otro lado Dubrova (2013) indica que el FR se lleva a cabo durante fases de desarollo de SW tanto como durante el ciclo de vida de un sistema. Durante la fase de desarrollo, FR consiste en tres pasos: *verificación, diagnóstico y correción* (Dubrova, 2013). FR durante la vida operacional de un sistema, consiste en el mantenimiento preventivo y correctivo del mismo (Dubrova, 2013).

2.4.4. Predicción de Fallas

La FF se realiza mediante la realización de una evaluación del comportamiento del sitema con respecto a la ocurrencia o la activación de una falla (Dubrova, 2013).Esta evaluación puede ser:

- Cualitativa: que tiene como objetivo clasificar los modos de fallas o combinaciones de eventos que llevan al sistema al fracaso (Dubrova, 2013).
- Cuantitativa: que tiene como objetivo evaluar en término de probabilidad, el grado en el cual los atributos de fiabilidad son satisfechos (Dubrova, 2013).

[6]En inglés, Fault prevention

FF incluye técnicas para aumentar la fiabilidad del sistema que son usados durante la validación del SW, con el objetivo de estimar la presencia de fallas y la ocurrencia o consecuencia de fracasos (Pullum, 2001)

2.5. Atributos de la fiabilidad

El objetivo final de la FT es desarrollar un sistema que sea fiable. Fiabilidad tiene muchas defiinciones, pero comúnmente es expresado como la probabilidad de **no fallar** (Hitt y Mulcare, 2007). "La fiabilidad es la probabilidad de que un sistema continúe funcionando correctamente durante un intervalo de tiempo particular" (Torres-Pomales, 2000).

La fiabilidad de un sistema SW puede ser descrita por una serie de atributos, los cuales son mencionados a continuación.

2.5.1. Confiabilidad

La confiabilidad es la probabilidad de que un sistema continua operando correctamente durante un intervalo de tiempo dado (Torres-Pomales, 2000).

Dubrova (2013) coincide que la confiabilidad *R(t)*[7] de un sistema es la probabilidad de que el sistema opere sin fracasos en el intervalo de tiempo *[0,t]*.

La confiabilidad es una medida de la entrega correcta del servicio que brinda un sistema (Dubrova, 2013).

En sistemas críticos como el SW de vehículo espacial, es sumamente necesario que tenga una alta tasa de confiabilidad, ya que por ejemplo, perder el contacto con la nave, podría representar la pérdida de la misión, o una gran cantidad de datos. Otro ejemplo que se puede mencionar es de un satélite geoestacionario de comunicación, la pérdida de este servicio debe ser baja, casi nula (idealmente).

Coincidiendo Pressman (2001), define la confiabilidad como la "probabilidad de tener operaciones libre de fallas de un programa de computadora, en un ambiente específico para un tiempo específico". El mismo autor también indica que la confiabilidad es la Tiempo Medio Entre Fallas (MTBF)[8]. Donde:

$$MTBF = MTTF + MTTR$$

Tiempo Medio de Fallas (MTTF) es el promedio de tiempo desde que empieza la operación del sistema hasta el tiempo que se produce la primera falla. Tiempo Medio de Reperación (MTTR) es el promedio de tiempo que se requiere para recuperarse, después de un fracaso, al correcto funcionamiento (Hanmer, 2007). MTBF es similar a MTTF, lo único que los diferencia es que MTBF es la suma de MTTF y MTTR. Según Hanmer (2007) MTBF es utilizado para aquellos sistemas que son reperables. Para el caso contrario se utiliza MTTF.

La IEEE (1990) define confiabilidad como "La capacidad del sistema o componente de realizar sus funciones requeridas bajo las condiciones establecidas durante un período de tiempo especificado".

[7] En inlgés, Reliability

[8] Del inglés, mean-time-between-failure

2.5.2. Disponibilidad

Es la probabilidad de que el sistema esté operando correctamente en un determinado instante de tiempo (Torres-Pomales, 2000). La disponibilidad *A(t)* de un sistema en el instante de tiempo *t* es la probabilidad que el sistema esté funcionando correctamente en el instante *t* (Dubrova, 2013).

Dubrova (2013) realiza un definción matemática de *A(t)*, llamandola también como, *punto de disponibilidad* o *disponibilidad instantanea*. Y la define como:

$$A(T) = \frac{1}{T}\int_0^T A(t)\,\mathrm{d}x$$

Para Hanmer (2007) la disponibilidad del sistema es el porcentaje de tiempo en el que es capaz de llevar a cabo una función determinada.

Para el caso de los sistemas que no pueden ser reperados el punto de disponibilidad es igual a la confiabilidad del sistema (Dubrova, 2013).

Los estados de disponibilidad pueden ser representados en términos de fuera de servicio por año. En la Tabla 2.1 que expone Dubrova (2013) se puede observar esta relación.

Disponibilidad	Fuera de servicio
90 %	36.5 días/año
99 %	3.65 días/año
99.9 %	8.76 horas/año
99.99 %	52 minutos/año
99.999 %	5 minutos/año
99.9999 %	31 segundos/año

Tabla 2.1: Disponibilidad en relación con su baja de servicio por año. Tabla modificada de Dubrova (2013)

La IEEE (1990) define la disponibilidad como "El grado en el cual un sistema o componente se encuentra operativo y accesible cuando se requiere su uso. También es expresado en términos de probabilidad"

En satélites de órbita baja (LEO[9]), es necesario que el satélite se encuentre disponible al momento de su pasada por las estaciones terreras para poder descargar los datos que se fueron almacenando. Del mismo modo, el satélite debe estar disponible para poder realizar las funciones necesarias para poder cumplir con su misión (como por ejemplo la registración de imágenes en una determinada zona terrestre).

Diferente es el caso para los satélites geoestacionarios, ya que estos deberían estar disponible la mayor parte del tiempo, ya que en la mayoría de los casos son de comunicación.

2.5.3. Seguridad

La seguridad se considera como una extensión de la confiabilidad (Dubrova, 2013). Seguridad *S(t)* es definidad como la probabilidad que el sistema sea capaz de realizar su función correctamente o discontinuar su función en una manera a prueba de fallas (Dubrova, 2013).

Según Torres-Pomales (2000) la seguridad es la probabilidad de que el sistema levará a cabo sus tareas de una manera no peligrosa. Un peligro se lo puede definir como "un estado o condición de un sistema, que juntos

[9]Del inglés, Low Earth Orbit

con otras condiciones ambientales de el sistema, conducirá inevitablemente a un accidente" (Torres-Pomales, 2000).

La seguridad es requerida para aquellas aplicaciones de seguridad crítica donde un fracaso puede resultar en lesiones humanas, pérdidas de vida o desastres ambientales (Dubrova, 2013).

Para satélites es importante que se tenga una alta seguridad, ya que una pérdida de una misión representa grandes cantidades de dinero perdido.

2.6. Tolerancia a falla

En sistemas críticos, como el de una planta nuclear, sistema médico, el sistema de vuelo de un avión, o el de un satélite, el SW (ni el hardware) deben fallar, ya que esto daría como resultado la pérdida de muchas vidas. Para el caso particular, del vehículo espacial (satélite, transbordador, lanzador), la falla del SW podría tener como consecuencia la pérdida de una misión, y/o una gran cantidad de dinero, y hasta vidas en algunos casos (vuelos tripulados). La principal diferencia entre el SW de una misión satelital, con la de un avión o una planta nuclear, o un sistéma médico, es que ante alguna falla o error, se torna complicado llegar hasta el satélite para realizar una actualización o cargar un parche de SW.

La IEEE (1990) define como SW crítico a "aquel cuyo fracaso puede tener un impacto en la seguridad, o puede causar grandes pérdidas financieras o sociales". El SW de estos sistemas críticos deben tener la capacidad de seguir funcionando, aún en la presencia de fallas, o errores. Imaginese el caso, de un avión comercial, con pasajeros a bordo, y de repente ocurre un problema debido al mal diseño del SW (por ejemplo un overflow de memoria). En esta situación es impensable que el SW se congele y que el piloto reinicie el sistema, esperar que se reestablezca al estado en el cual se encontraba antes del problema, para seguir funcionando. Lo mismo ocurre con el SW de naves espaciales, hay situaciones en la que no se puede esperar y es preferible que el sistema siga funcionando aún en la presencia de fallas.

Tal lo como indica Pressman (2001) las fallas de SW implica problemas cualitativos que son descubiertos después de que el SW es llevado a los usuarios y probados por ellos. Una gran cantidad de estudios indican que en las actividades de diseño se introducen entre un 50 y 65 porciento de errores del total de errores que se dan durante el proceso del SW (Pressman, 2001). Esto no debe ocurrir en el ámbito espacial, ya que una vez que el sistema es utilizado, es muy dificil corregir los errores que surgen

Cabe aclarar que el SW al no ser un componente físico, no puede ser tratado de la misma manera que un componenete hardware. Como ejemplifica Torres-Pomales (2000), las fallas que surgen a nivel de bit, como por ejemplo en un disco duro, son fallas del dispositivo de almacenamiento y pueden ser mitigadas con la aplicación de técnicas de redundancias. Esto no es así para el SW. Por lo tanto evitar los errores en el a nivel de SW no es tan trivial como en el hardware.

A nivel de SW las fallas son llamadas "bugs" (tal como se indica en la sección 2.1 en la página 2), y existe un solo tipo de fallas que es introducido durante el desarrollo del SW (Torres-Pomales, 2000). Las fallas en el SW son el principal motivo de que todo un sistema fracase.

La FT, puede ser utilizada como una capa más de protección (Torres-Pomales, 2000). Esta aplicada al SW se refiere al uso de técnicas que permiten seguir brindando el servicio en un nivel aceptable de perfomance y seguridad después que una falla de diseño ocurra.

Debe hacerse una diferencia entre FT y calidad. Hanmer (2007) lo define de la siguiente manera: "FT es la capacidad del sistema a ejecutarse apropiadamente a pesar de la presencia de fallas. FT ocurre en tiempo de ejecución". Cuando se habla que un sistema es tolerante a fallas, signfica que fue diseñado de tal manera, que puede seguir funcionando correctamente aún en la presencia de errores de sistemas (Hanmer, 2007).

En cambio calidad, tal como lo define Hanmer (2007), “se refiere a cuán libre de fallas está el sistema. Técnicas de calidad que indican cómo el SW es creado. Si el sistema fue testeado.”

Un sistema de alta calidad tendrá menor número de fallas, que esto representa menor número de fallas en tiempo de ejecución. La reducción del número de fallas no implica que los resultados de los defectos son menos severos (Hanmer, 2007). El sistema debe tomar medidas para reducir el impacto de los errores y fallas, y es allí donde surge la FT.

Un sistema tolerante a fallas provee una continua y segura operación, aún durante la presencia de fallas. Un sistema tolerante a fallas, es un elemento crítico para una arquitectura de vuelo, lo cual incluye hardware, SW, timing, sensores y sus interfaces, actuadores, elementos y datos de comunicación con los diferenes elementos (Hitt y Mulcare, 2007).

Este tipo de sistemas debería detectar los errores causados por fallas, evaluar los daños producidos por la falla, aislar a la misma y por último recueperarse, en ese caso se habla de aquitectura o sistemas FDIR[10].

FT es la capacidad de un sistema a continuar funcionando a pesar de la ocurrencia de fallas (Dubrova, 2013). Un sistema tolerante a fallas debe ser capaz de manejar fallas tanto de hardware como de SW. La FT es necesaria debido a que es imposible construir un sistema perfecto.

El objetivo de la FT es el desarrollo de sistemas los cuales funcionen correctamente en presencia de fallas (Dubrova, 2013). La FT es alcanzada mediate la utilización de algunos tipos de redundancias (Dubrova, 2013). *Redundancia* es la provisión de capacidades funcionales que sería innecesario para enternos libres de fallos (Dubrova, 2013). Esto significa tener hardware adicionales, check bits en una cadena de datos, o algunas líneas de código que verifica el correcto resultado del SW. La redundancia permite enmascarar una falla, o detectarla, para luego localizarla, contenerla y recuperarse de esta (Dubrova, 2013). Las técnicas de tolerancia de fallas se emplean durante la adquisición, o desarrollo del SW. Permite al SW tolerar fallas después que este haya sido desarrollado (Pullum, 2001). Cuando una falla se da, las técnicas de FT proveen mecanismos al sistema de SW para prevenir el fracaso del sistema (Pullum, 2001).

2.7. Clasificación de un sistema de control tolerante a fallas

2.8. Redundancia en el software

A pesar de lo comentado anteriormente, sobre la importancia del SW, todavía existe una creencia, de que el SW aparece por arte de magia, y que los programadores no son nunca, lo suficientemente capaces, de hacer un SW libre de errores. Salvo aquellas empresas u organizaciones que tienen un proceso maduro de desarrollo de SW, el resto cae en el error de pensamiento mencionado anteriomente.

Dubrova (2013) explica que la FT aplicado en el SW no está tan entendido, ni maduro, como es en el caso de la FT aplicada en hardware. Si una falla exisitiera en el SW, esta se haría “visible”, solo cuando las condiciones relevantes ocurran (Dubrova, 2013). Y muchas veces por tiempo o costo, no se realizan los tests cubriendo todos los posibles ambientes reales, lo cual tiene consecuencias desastrosas, tal como se expone en la sección 1.1 (página 1).

Para sistemas complejos o grandes, donde existe una gran cantidad de estados, implica que solo una pequeña porción del SW puede ser verifcada correctamente (Dubrova, 2013). Los tests tradicionales y métodos de depuración actuales, no alcanzan para grandes sistemas (Dubrova, 2013). La utilización de métodos formales para describir las característica requeridas por el comportamiento del SW, exigen gran complejidad computacional,

[10] FDIR, del inglés: Failure detect, isolate and recover

y solo son aplicables en ciertas situaciones (Dubrova, 2013).

Las técnicas de FT pueden dividir en dos grupos:

- técnicas de una sola versión, se utilizan cuando existe una sola versión del SW en el sistema.
- técnicas multi-versión, se utilizan cuando se desarrollan varias versiones de una misma función.

Estas se explican en las siguientes secciones.

2.8.1. Técnicas single version

Estas técnicas son utilizadas para tolerar parcialmente las fallas del diseño de SW (Pullum, 2001). Técnicas single-version de FT se basa en el uso de redundancia aplicada a una única versión de una pieza de SW para detectar y recuperarse de fallas (Torres-Pomales, 2000).

Estas técnicas a los software que cuentan con una sola versión, un número capacidades funcionales que no serían necesarias dentro de un ambiente libre de fallas (Dubrova, 2013).

2.8.1.1. Estructuras de software

En Torres-Pomales (2000) se mencionan dos técnicas de estructuración del SW que son muy buenas a la hora de mantener FT en el SW.

La defínción de una arquitectura en el software es de suma importancia ya que proveen las bases para la implementación de FT (Torres-Pomales, 2000). Una de las técnicas utilizadas en el desarollo del software es la modularización. Esta consiste en descomponer el problema en componentes manejables. Esto tiene como resultado que sea más eficiente la aplicación de la FT en el diseño de un sistema (Torres-Pomales, 2000).

El particionado es otra técnica mencionada en Torres-Pomales (2000), lo cual provee aislación entre módulos independentes del sistema. Esta técnica permite descomponer al problema en partes separadas (Pressman, 2001). El particionado puede ser horizontal u vertical. En el primero se descomponen el problema moviendose en forma horizontal en la jerarquía, mientras que el segundo se parte de lo más general hasta llegar a lo detallado, moviendose verticalmente en la jerarquía (Pressman, 2001).

Sistema de cierre es un principio de FT, en el cual ninguna acción es permitible sin una autorización expresa (Torres-Pomales, 2000). Siguiendo este principio ninguna de las funciones que componen al sistema deberían tener más capacidad de la necesaria (Torres-Pomales, 2000). Las ventajas de desarrollar un sistema bajo este pricipio, es que es sencillo el manejo de errores, y evitar la propagación de fallas si ocurriesen.

2.8.2. Técnicas multi-version

Las técnicas de multi-version utilizan dos o más versiones diferentes del mismo módulo de SW (Dubrova, 2013) (Torres-Pomales, 2000), lo cual satisface el requerimiento de diversidad.

El objetivo de utilizar diferentes versiones de SW es que es construido de diferentes maneras, por lo tanto fallarían de diferente maneras (Torres-Pomales, 2000).

2.8.3. Técnicas de detección de fallas

Para los SW tolerantes a fallas, de una sola versión, se suelen utilizar varios tests de "aceptación" para detectar fallas (Dubrova, 2013). Es necesario que estos SW cuenten con dos propiedades: auto protección[11] y auto check[12] (Torres-Pomales, 2000). La auto protección significa que los componentes de sistema tienen la capacidad de protegerse así mismo mediante la detección de errores (Torres-Pomales, 2000). La propiedad de auto check significa que los componente son capaces de detectar fallas internas y tomar las acciones necesarias para evitar la propagación del error.

El resultado del sistema depende del resultado de los tests. Si el resultado pasa exitosamente el test, este es el correcto, caso contrario significa la presencia de fallas (Dubrova, 2013). Un test es más efectivo si se puede calcular de una manera simple (Dubrova, 2013).

Las técnicas utilizadas son las siguientes:

- *Timing checks:* se agrega a los sistemas una restricción de tiempo. Basado en esa restricción se puede deducir si el comportamiento del sistema se desvió (Dubrova, 2013). Los más utilizado es el *watchdog timer*, este es un contador, actualizado con un *timer* que detecta si un módulo de SW se bloqueó o congeló, entonces se reincia ese módulo o el sistema.
- *Coding checks:* se utiliza en los sistemas donde los datos se codifican usando técnicas de redundancia de datos (Dubrova, 2013).
- *Reversal checks:* son aquellos donde se toma los valores de salida, y con ellos se busca encontrar cuáles fueron los datos de entrada. Si los datos de entrada reales coinciden con los calculados (para una misma salida), este se encuentra libre de fallas (Dubrova, 2013).
- *Reasonableness checks:* usa propiedades semánticas en los datos para detectar fallas (Dubrova, 2013).
- *Structural checks:* se basa en el conocimiento de las propiedades de la estructura de datos (Dubrova, 2013).
- *Replication checks:* se basa en la comparación de resultados de varios componentes (Torres-Pomales, 2000).

Se suelen utilizar árboles de fallas, como una técnica auxiliar en el desarrollo de sistemas para la detección de fallas (Torres-Pomales, 2000). El árbol de falla permite obtener un enfoque top-down de las diferentes fallas que se pueden dar. El árbol no cubre todas las fallas que puedan darse, pero si ayudan en un alto grado en el desarrollo de SW tolerante a fallas (Torres-Pomales, 2000).

2.8.4. Técnicas de recuperación de fallas

Una vez que la falla es detectada, el sistema debe proceder a recuperarse de aquella, y volver a un estado operacional normal (Dubrova, 2013). Si los mecanismos de detecciín y contención de fallas fueron desarrollados correctamente, esta es contenida dentro de un set de módulos en el momento de la detección (Dubrova, 2013).

[11] En inglés, self-protection
[12] En inglés, self-checking

2.8.4.1. Manejo de excepciones

En muchos SW y lenguajes de programación, se logra recuperarse mediante el manejo de excepciones. El manejo de excepciones es la interrupción del funcionamiento normal para responder a un funcionamiento anormal del sistema (Torres-Pomales, 2000). Los posibles eventos que pueden lanzar una excepción son:

1. Excepciones de interfaces, son lanzadas por un módulo cuando se da una solicitud inválida de algún servicio (Dubrova, 2013).
2. Excepciones locales, son lanzadas por algún módulo cuando sus propios mecanismos de detección de fallas encuentran un problema interno (Dubrova, 2013).
3. Excepciones de fracaso, son lanzadas cuando un mecanismos de detección encuentra una falla, per es imposible recueperarse de esa falta (Dubrova, 2013).

2.8.4.2. Checkpoint y Restart

Para los software de una sola versión existen pocos mecanismos de recuperación. Checkpoin y restart es uno de ellos. También es conocido como *backward error recovery* (Dubrova, 2013). La mayoría de las fallas que se dan en los SW son debido a fallas que provienen del diseño, tal como se mencionó anteriomente. Estas fallas son activadas por entradas al sistema (Dubrova, 2013).

Este mecanismo cuenta con el módulo principal que se encuentra en ejecución combinado con un bloque que realiza tests de aceptación. Si se detecta una falla, en el bloque de testeo, se envía una señal de "reincio", para que el módulo principal vuelva al estado anterior, es decir, antes de producirse el error. Este estado anterior se encuentra almacenado en una memoria checkpoint (Dubrova, 2013). En la figura 2.2[13], se muestra la representación de este mecanismo.

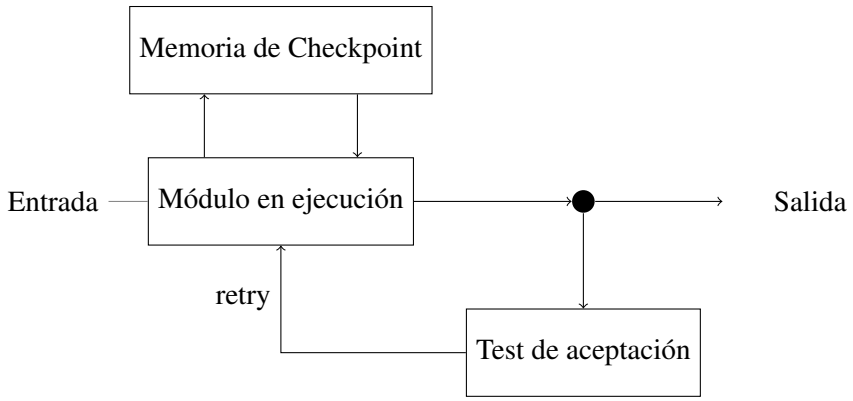

Figura 2.2: Representación de checkpoint y restart

Existe dos tipos de checkpoints, estáticos y dinámicos. Los checkpoints dinámicos toman una "fotografía" del estado del sistema antes de comenzar la ejecución del SW y lo guarda en memoria (Dubrova, 2013). Si se detecta una falla, el sistema regresa a ese estado y comienza de nuevo su ejecución (Dubrova, 2013). Los checkpoints estáticos se basan en regresar el módulo a un estado predeterminado (Torres-Pomales, 2000). Se puede regresar a un estado inicial o a un set de estados predeterminados (Torres-Pomales, 2000).

Por otro lado se encuentran los checkpoints dinámicos. Estos usan checkpoints creados dinámicamente. Estas son imágenes del estado del sistema en varios puntos durante la ejecución (Torres-Pomales, 2000).

[13] Basado en Dubrova (2013) y Torres-Pomales (2000)

Hay tres formas de crear los checkpoints dinámicamente:

1. Equidistantes, en el cual los intervalos que se crean los checkpoints son siempre iguales, los intervalos se elijen teniendo en cuenta el rate de falla (Dubrova, 2013).
2. Modular, en el cual los checkpoints se crean al principio o al final de la ejecución de un módulo.
3. Random, los checkpoints se crean aleatoriamente en el tiempo.

2.8.4.3. Procesos pares

Los procesos pare utilizan dos versiones identicas de un proceso de SW que corre en procesadores separados (Dubrova, 2013) (Torres-Pomales, 2000). El mecanismo de recuperación que se utiliza es el de checkpoint y restart (Torres-Pomales, 2000).

Como se puede observar en la figura 2.4[14] el primer procesador se encuentra activo. Este envía un checkpoint al segundo procesador. Si una falla se detecta, el primer procesador se apaga y se cambia al segundo procesador. El segundo procesador carga el checkpoint y continua con la operación.Toma el rol del primer procesador (Torres-Pomales, 2000). Luego el primer procesador realiza un auto test para verificar si el problema continua. Si se encuentra que este procesador sigue teniendo problema, se continúa trabajando con el segundo procesador (Dubrova, 2013).

La principal ventaja que brinda este mecanismo según Dubrova (2013) es que permite entregar el servicio ininterrumpidamente.

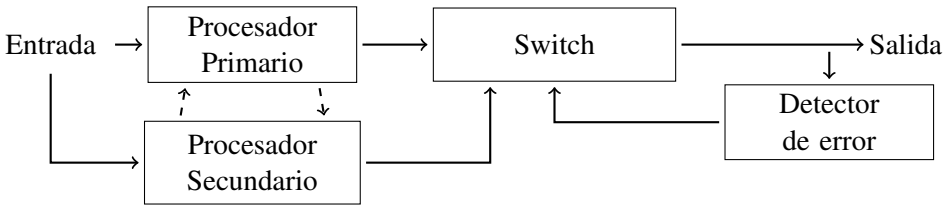

Figura 2.3: Representación del proceso pares

2.8.4.4. Diversidad de datos

La diversidad es una técnica utilizada para mejorar la eficiencia en los checkpoint y restart, usando diferentes entradas por cada reinicio (Dubrova, 2013). Esto se basa en que las fallas en el SW son dependientes de las entradas (Dubrova, 2013). Es poco probable que la misma falla se de con la misma secuencia de entrada (Dubrova, 2013).

2.8.4.5. Bloques de recuperación

Esta técnica combina las bases de la técnica de checkpoints y restart enfocada con múltiples versiones de un componente de SW en el sentido de que una versión de SW diferente es lanzada cada vez que se encuentra una falla (Torres-Pomales, 2000). Los checkpoints son creados antes de que una versión de SW se ejecuta (Torres-Pomales, 2000). La ejecución de las múltiples versiones pueden ser secuencial o paralelas dependiendo de la disponibilidad de la capacidad de procesamiento y perfomance requerida (Torres-Pomales, 2000).

[14] Basada en Dubrova (2013) y Torres-Pomales (2000)

La representación de esta técnica se puede observar en el figura [AGREGAR IMAGEN]. Las versiones son diferentes implementaciones de un mimo programa. Solo uno de estas versiones provee la salida del sistema. Si un error es detectador por el test de aceptación, se vuelve hacia atrás, se retoma el último checkpoint, y se vuelve a ejecutar el módulo de SW pero con una versión diferente a la que se ejecutó anteriomente (Dubrova, 2013).

Los checks del test de aceptación deben mantenerse simples para mantener la velocidad de la ejecución (Dubrova, 2013).

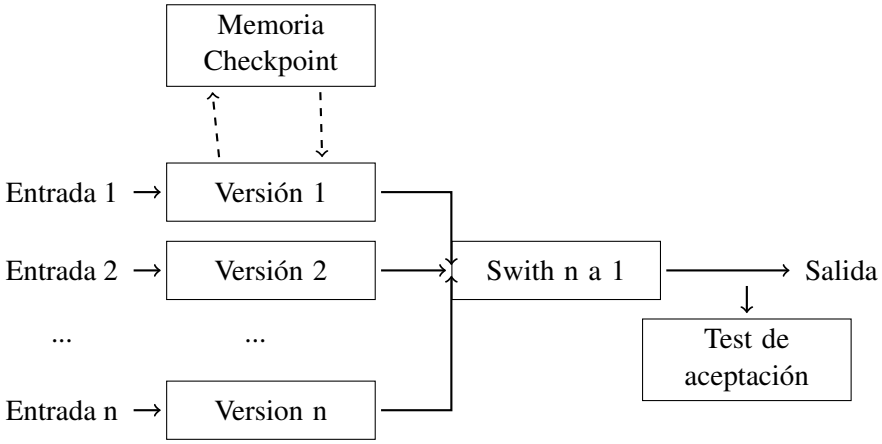

Figura 2.4: Configuración de bloques de recuperación

2.8.4.6. Programación N-version

2.8.4.7. Programación N-Auto Checking

Bibliografía

Thomas Anderson y John C. Knight. A Framework for Software Fault Tolerance in Real-Time Systems. *IEEE Trans. Software Eng.*, (3):355–364, 1983.

Dr Ralph D. Lorenz (auth.) David M. Harland. *Space Systems Failures: Disasters and Rescues of Satellites, Rockets and Space Probes*. Springer Praxis Books. Praxis, 1 edition, 2005. ISBN 978-0-387-21519-8,978-0-387-27961-9.

Douglas Isbell and Don Savage. Mars Climate Orbiter Failure Report - NASA, 1999.

Elena Dubrova. *Fault-Tolerant Design: an introduce*. Springer-Verlag New York, New York, U.S.A., 1 edition, 2013.

Guillaume Jacques Joseph Ducard. *Fault-Tolerant Flight Control and Guidance Systems for a Small Unmanned Aerial Vehicle*. Tesis de Doctorado, Swiss Federal Institue of Technology Zurich, 2007.

Cristopher Edwards, Thomas Lobaerts, y Hafid Smaili. *Fault Tolerant Flight Control. A benchmarck challenge*. Springer-Verlag, 2010. ISBN 978-3-642-11689-6.

Sanford Friedenthal, Alan Moore, y Rick Steiner. *A Practical Guide to SysML: Systems Modeling Language*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

W. C. Gangloff. Common mode failure analysis. *IEEE Transactions on Power Apparatus and Systems*, 94(1): 27–30, Jan 1975. ISSN 0018-9510. doi: 10.1109/T-PAS.1975.31820.

Robert S. Hanmer. *Patterns for Fault Tolerant Software*. John Wiley and Sons Ltd, England, 2007. ISBN 978-0-470-31979-6.

Ellis F. Hitt y Dennis Mulcare. Fault-Tolerant Avionics. En Cary R. Spitzer, editor, *Avionics Development and Implementation Second Edition*, capítulo 8. CRC Press, Williamsburg, Virginia, U.S.A., 2007.

Jon Holt y Simon Perry. *SysML for systems engineering.* The Institution of Engineering and Technology, London, United Kingdom, 2008. ISBN 978086341825.

IEEE. IEEE Standard Glossary of Software Engineering Terminology. Std. 610.12-1990, IEEE, 1990.

Michael R. Lyu. *Software Fault Tolerance*. John Wiley and Sons Ltd, Chichester, New York, USA, 1995. ISBN 9780471950684.

MARTE. `http://www.omgmarte.org/`, 3 de Junio de 2016.

Papyrus. `http://eclipse.org/papyrus/`, 3 de Junio de 2016.

Roger S. Pressman. *Software Engineering, A Practioner's Approach*. McGraw-Hill, fifth edition edition, 2001.

Laura L. Pullum. *Software fault tolerance techniques and implementation*. Artech House, USA, 2001. ISBN 1-58053-137-7.

SysML. `http://sysml.org/`, 2 de Junio de 2016.

Wilfredo Torres-Pomales. Software Fault Tolerance: A tutorial. Technical Report NASA/TM-2000-210616, National Aeronautics and Space Administration Langley Research Center, Hampton, Virginia, October 2000.