Dedicado a

mi familia

Agradecimientos

Muchas gracias

Resumen

Esta tesis se trata de

Índice general

Índice de figuras

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

Software tolerante a fallas

2.1. Tolerancia a fallas

En sistemas críticos, como el de una planta nuclear, sistema médico, el sistema de vuelo de un avión, o el de un satélite, ni el software (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.

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 introducen entre un 50 y 65 porciento de errores del total de errores que se introducen 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", y existe un solo tipo de fallas y es introducido durante el desarrollo del SW (Torres-Pomales, 2000). Las fallas en el SW son el principal motivo de que todo un sistema falle.

Según Torres-Pomales (2000) existen cuatro maneras para hacer frente a las fallas de SW:

- Prevención
- Eliminación
- Tolerancia a Fallas (FT)
- Correcciones temporales en la secuencia de entrada

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[1].

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.

Se dice que un sistema es erroneo cuando deja de cumplir sus función. *Failure* puede ser el cese total del funcionamiento.

Redundancia es la capacidad de proveer capacidades funcionales que sería innecesario para enternos libres de fallos. Esto significa tener hardware adicionales, check bits en una cadena de datos, o algunas líneas de código que verifica el correcto resultado del SW.

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

Referencias

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

David M. Harland, D. R. D. L. a. (2005). *Space systems failures: Disasters and rescues of satellites, rockets and space probes* (1.ª ed.). Praxis.

Douglas Isbell and Don Savage. (1999). *Mars Climate Orbiter Failure Report - NASA.*

Dubrova, E. (2013). *Fault-Tolerant Design: an introduce* (1.ª ed.; C. R. Spitzer, Ed.). New York, U.S.A.: Springer-Verlag New York.

Ducard, G. J. J. (2007). *Fault-Tolerant Flight Control and Guidance Systems for a Small Unmanned Aerial Vehicle* (Tesis Doctoral no publicada). Swiss Federal Institue of Technology Zurich.

Edwards, C., Lobaerts, T., y Smaili, H. (2010). *Fault Tolerant Flight Control. A benchmarck challenge.* Springer-Verlag.

Friedenthal, S., Moore, A., y Steiner, R. (2008). *A Practical Guide to SysML: Systems Modeling Language.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Hanmer, R. S. (2007). *Patterns for Fault Tolerant Software.* England: John Wiley and Sons Ltd.

Hitt, E. F., y Mulcare, D. (2007). Fault-Tolerant Avionics. En C. R. Spitzer (Ed.), *Avionics Development and Implementation Second Edition* (cap. 8). Williamsburg, Virginia, U.S.A.: CRC Press.

Holt, J., y Perry, S. (2008). *SysML for systems engineering.* London, United Kingdom: The Institution of Engineering and Technology.

Lyu, M. R. (1995). *Software Fault Tolerance.* Chichester, New York, USA: John Wiley and Sons Ltd.

MARTE. (3 de Junio de 2016). *http://www.omgmarte.org/.*

Papyrus. (3 de Junio de 2016). *http://eclipse.org/papyrus/.*

Pressman, R. S. (2001). *Software Engineering, A Practioner's Approach* (fifth edition ed.). McGraw-Hill.

SysML. (2 de Junio de 2016). *http://sysml.org/.*

Torres-Pomales, W. (2000, October). *Software Fault Tolerance: A tutorial* (Inf. Téc. n.º NASA/TM-2000-210616). Hampton, Virginia: National Aeronautics and Space Administration Langley Research Center.