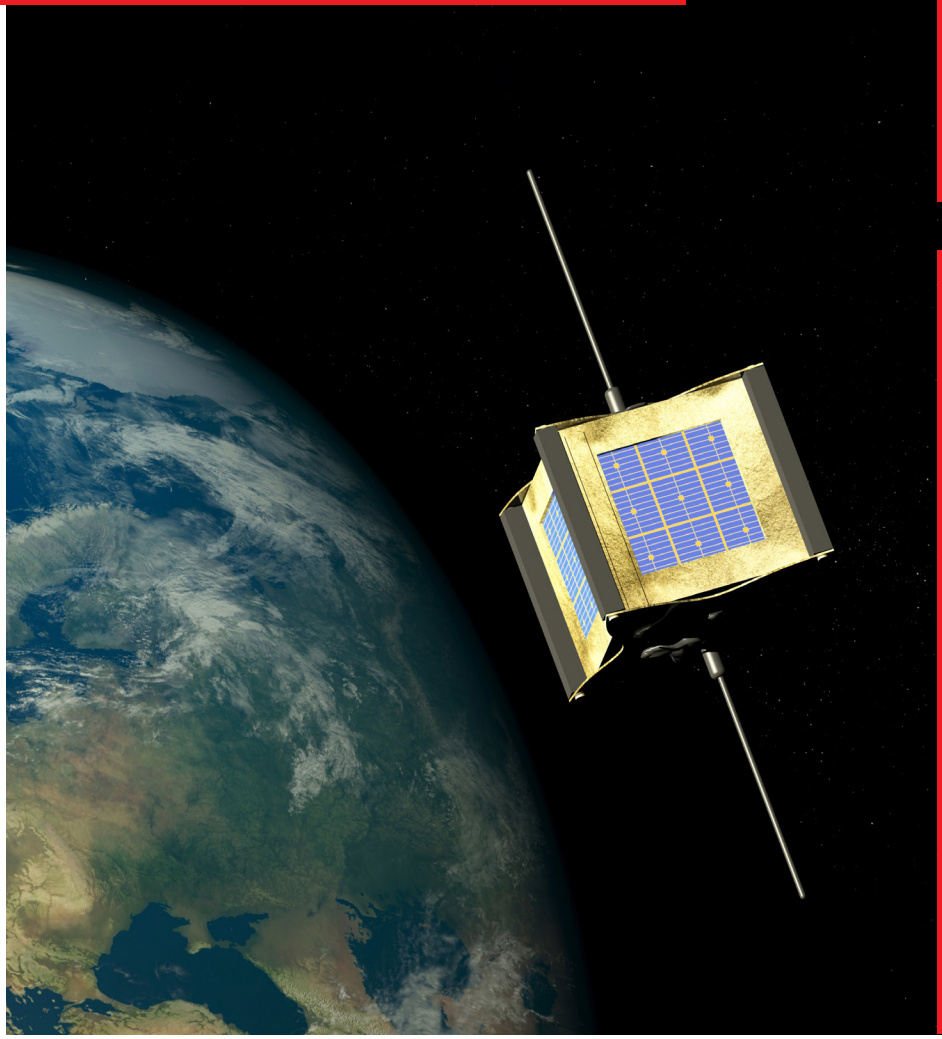
Fuzz Testing en el software de vuelo de un CubeSat:

Usando y analizando técnicas sistemáticas para mejorar la calidad de software en el ámbito aeroespacial bajo un contexto ágil



TAMARA GUTIÉRREZ

Magíster en Ciencias mención Computación por la Universidad de Chile. Actualmente se desempeña como investigadora científica en el Instituto para la Tecnología del Software del Centro Aeroespacial Alemán (DLR). Su línea de investigación está principalmente asociada al estudio y análisis de calidad de software de sistemas críticos dentro del área aeroespacial.

tamara.gutierrezrojo@dlr.de

ALEXANDRE BERGEL

Doctor en Ciencias de la Computación por la Universidad de Berna, Suiza. Actualmente se desempeña como científico en RelationalAI. Fue Profesor Asociado de la Universidad de Chile hasta 2022. Sus líneas de investigación son ingeniería de software e inteligencia artificial para sistemas críticos.

https://bergel.eu

RESUMEN. El éxito de las misiones de CubeSats depende de su rendimiento en un ambiente extremo. El software de vuelo es un componente crítico que maneja estas operaciones. Aunque en otras áreas se incluyen técnicas de testing de software avanzadas, las soluciones de software para CubeSats dependen mayoritariamente de técnicas costosas en tiempo, mientras que los requerimientos de este tipo de misiones necesitan añadir "agilidad" al desarrollo.

En el trabajo que presentamos en este artículo, se desarrollaron y evaluaron técnicas de *fuzz testing* para facilitar las pruebas y mantener la robustez del software de vuelo de las misiones SUCHAI de la Universidad de Chile. Los resultados indican que *fuzz testing* mejoró la completitud de testing del software mediante la automatización y con muy poca interrupción en el desarrollo. Así, nuestras contribuciones muestran la diferencia en la forma en la que el software de vuelo es evaluado y las técnicas encontradas en la comunidad de ingeniería de software.

¿Qué es un CubeSat y para qué sirve?

Un CubeSat se define como un nanosatélite que tiene una forma cúbica y dimensiones estándares. La medida básica de un CubeSat, 1 unidad o "1U", consiste en un cubo de 10 centímetros por arista. También existen CubeSats de más unidades, por ejemplo de "2U" o "3U", dependiendo de la cantidad de subestructuras cúbicas que tenga el sistema. En la Figura 1 se muestra la imagen del CubeSat SUCHAI I, el primer satélite creado en Chile, cuya medida es de 1U.

El primer prototipo de un nanosatélite tipo CubeSat surgió hace veinte años aproximadamente. En un comienzo, los nanosatélites fueron creados con un propósito inicialmente educacional, gracias al cual los estudiantes son capaces de experimentar el desarrollo y la operación de un satélite durante el periodo que dura su carrera universitaria [1]. Hoy en día, los nanosatélites han abierto numerosas oportunidades no solo en el área educacional o científica, sino también en la industria debido, principalmente, a su bajo costo de desarrollo.

La importancia del software en un CubeSat y de su calidad

Para llevar a cabo una misión de CubeSats se requiere la participación de diversas áreas, principalmente, del campo ingenieril. El software es una de las partes fundamentales de una misión porque a través de sus instrucciones el sistema puede efectuar las distintas funciones para las que fue creado una vez en órbita. Uno de los componentes principales de un CubeSat es el computador a bordo, el cual ejecuta el software de vuelo. Este software permite al satélite realizar todas las operaciones necesarias de la misión y además realizar tareas para su mantención.

El uso y aplicaciones de los nanosatélites, y en especial, de los CubeSats, ha ido en aumento en los últimos años y aún se necesita superar múltiples desafíos para alcanzar su completo potencial [2]. Los nanosatélites cada vez requieren más atención a sus atributos de calidad para tener éxito en misiones más complejas. Específicamente, el software de vuelo de nanosatélites es un factor crítico para determinar la calidad de un satélite porque engloba toda la lógica del funcionamiento del sistema. La tasa de éxito de una misión depende altamente de la calidad de su software de vuelo [3].

¿Cuáles son las técnicas existentes para medir la calidad de software de un CubeSat?

En el campo espacial, varias técnicas de testing son usadas para asegurar la calidad del software de vuelo. Sin embargo, las técnicas más avanzadas solo son aplicables para misiones o sistemas más complejos, en términos de tiempo y costos, tales como los grandes satélites, rovers o misiones interplanetarias [4]. Dentro del estado del arte, las técnicas de testing aplicadas a software de vuelo de nanosatélites más mencionadas son *hardware in the loop simulation* (HILS) y *software in the loop simulation* (SILS) [5, 6]. Las metodologías de HILS y SILS pueden optimizar los costos totales del proceso de producción en algunas situaciones [7, 8]. Sin embargo, estas técnicas pueden ser difíciles de implementar y ejecutar, potencialmente riesgosas para el hardware cuando se ejecutan en modelos de vuelo o de ingeniería, y costosas en tiempo al preparar el ambiente de ejecución de las pruebas. Además, los casos de prueba deben estar predefinidos porque estas técnicas son

Figura 1. Prototipo del CubeSat SUCHAI I, cuya medida es de 1U.

Hasta el momento previo a la realización de este trabajo [...] las pruebas unitarias estaban basadas en las interfaces de los módulos principales del software, pero había cuatro pruebas diseñadas por módulo, a lo más.

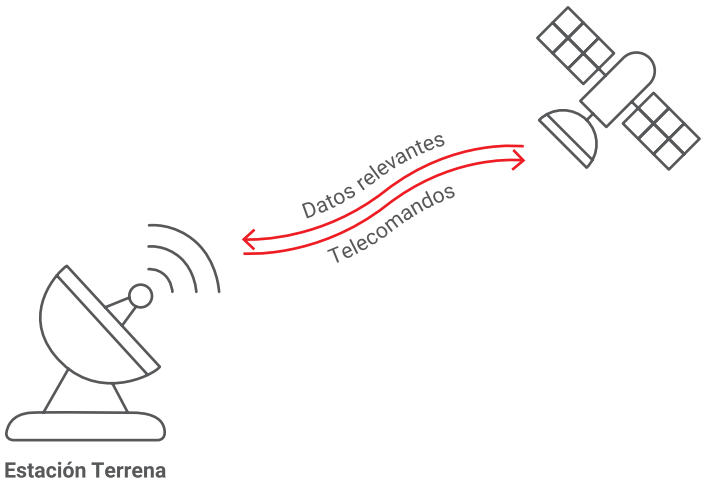


Figura 2. Operación de un nanosatélite. Envío de telecomandos desde una estación terrena hacia un CubeSat y envío de datos de telemetría de un CubeSat hacia una estación terrena.

complejas de automatizar [7]. Por otro lado, una revisión reciente de algunos frameworks relevantes de software de vuelo de nanosatélites muestra que solo tres de seis candidatos muestran el atributo de *confiabilidad*, el cual se refiere a la existencia de testing unitario con una significativa cobertura del código [9].

La habilidad de implementar diferentes técnicas de testing también recae en el diseño del software de vuelo. Los sistemas de manejo de comandos y datos están usualmente diseñados para recibir telecomandos enviados desde una estación terrena, ejecutar las acciones necesarias, y responder con datos obtenidos de la telemetría hacia la misma estación, tal y como se muestra en la Figura 2. Algunos diseños nuevos de software de vuelo explotan este concepto para implementar una arquitectura de software basada en comandos [10, 11]. Un diseño claro e interfaces bien documentadas pueden ayudar a implementar estrategias de testing que tratan al software de vuelo como una caja negra en lugar de intervenir el código con testing unitario o instrumentación.

Actualmente, las altas expectativas de los CubeSats están puestas en la posibilidad de desarrollar un gran número de satélites (megaconstelaciones) de una forma efectiva en costos [12, 13]. La efectividad del costo requiere que estas constelaciones puedan ser desarrolladas por pequeños grupos sin mucha experiencia (por ejemplo, startups) en ciclos cortos de desarrollo. Este hardware comercial tiene usualmente más capacidad de cómputo con menor consumo de energía, está más miniaturizado y actualizado con respecto a las necesidades tecnológicas. Sin embargo, generalmente el funcionamiento de este hardware y software no ha sido probado en el espacio con anterioridad, haciéndolos más riesgosos de utilizar en el espacio. La automatización de testing surge como la manera más efectiva en costo de mantener un desarrollo ágil mientras se asegura la calidad y robustez requerida en el sistema.

Las misiones SUCHAI y el software de vuelo SUCHAI Flight Software

El año 2017 fue lanzado desde el Centro Espacial Satish Dhawan, ubicado en India, SUCHAI I[1]. El satélite fue creado por un grupo de estudiantes, ingenieros e investigadores de distintas disciplinas en el Laboratorio de Exploración Espacial y Planetaria (SPEL) del Departamento de Ingeniería Eléctrica de la Universidad de Chile. Esta primera misión tenía como objetivo realizar una serie de experimentos y demostraciones tecnológicas.

El software de vuelo del nanosatélite, llamado SUCHAI Flight Software, fue especialmente programado para la misión. SUCHAI Flight Software fue diseñado para ser altamente modular y flexible, y posee una arquitectura basada en comandos. De esta forma, los comandos pueden ser ejecutados automáticamente desde ciertos módulos del mismo software, o pueden ser enviados hacia la estación terrena para ejecutarse allí, tal como se muestra en la Figura 3.

Continuamente se han ido incorporando herramientas para mejorar la calidad

1 https://www.uchile.cl/noticias/133697/suchai-i-hacia-un-programa-espacial.

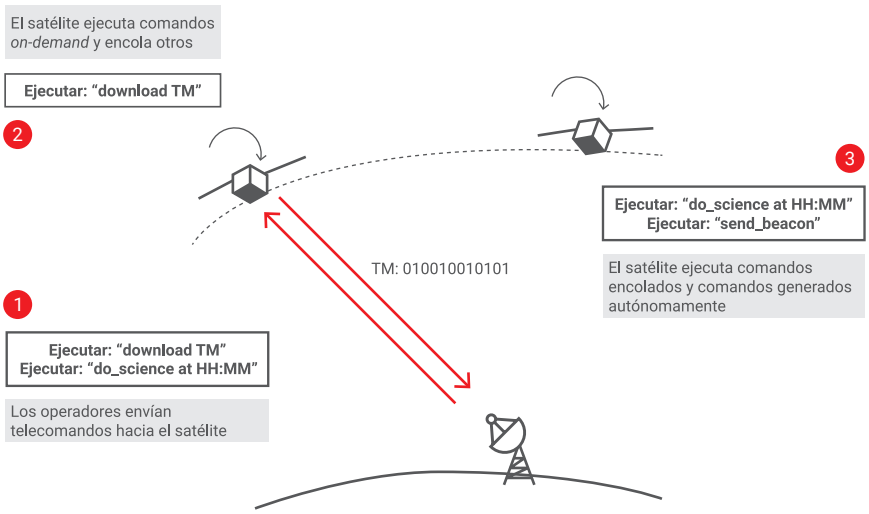


Figura 3. Operación del software de vuelo de los CubeSats SUCHAI bajo una arquitectura basada en comandos. Estos comandos pueden ser enviados directamente desde una estación terrena o ejecutarse autónomamente. Imagen obtenida de [10].

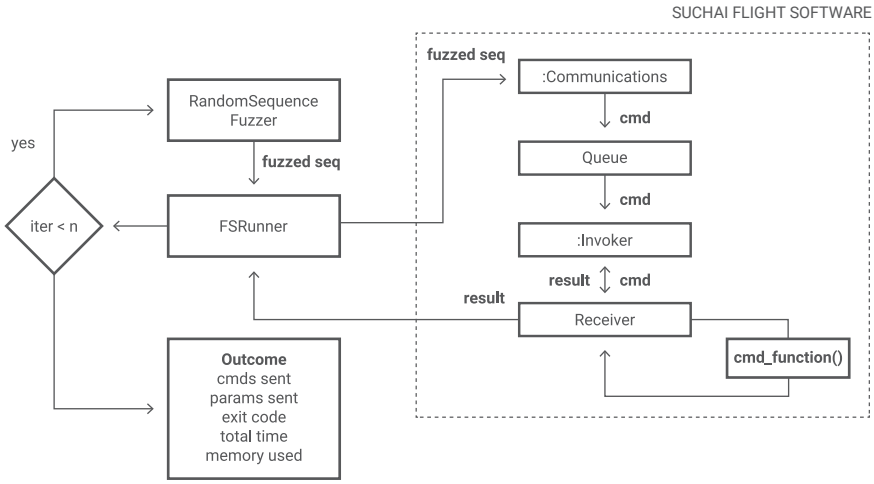


Figura 4. Diagrama lógico de la implementación de *fuzz testing* en el software de vuelo de las misiones SUCHAI. La aplicación consta de dos clases principales. *RandomSequenceFuzzer* representa el generador de secuencias y *FSRunner* ejecuta una instancia del software de vuelo y le envía las secuencias de comandos generadas. Los resultados principales a obtener tienen relación con el código de retorno del programa, el tiempo de ejecución y la memoria utilizada.

del software de vuelo [10], tales como una herramienta de visualización para evaluar su arquitectura y un sistema de integración continua, donde se incluyen diferentes pruebas que se ejecutan cada vez que el software es modificado en el sistema de control de versiones donde se almacena.

Hasta el momento previo a la realización de este trabajo, las técnicas principales de testing aplicadas a SUCHAI Flight Software durante su desarrollo eran testing unitario, testing de integración y de simulación de hardware (*hardware in the loop simulation*). Las pruebas unitarias estaban basadas en las interfaces de los módulos principales del software, pero había cuatro pruebas diseñadas por módulo, a lo más. El sistema de testing de integración, que prueba la interacción entre los distintos módulos del sistema, consistía en ejecutar el software con una configuración específica y con casos de prueba conocidos por los operadores del satélite y desarrolladores del software. Por otro lado, las pruebas de simulación de hardware son costosas en general por el tiempo de preparación del ambiente de pruebas. Encima, estas estaban siendo ejecutadas en el mismo computador a bordo que se utiliza para la misión, lo cual requiere aún más cuidado en el diseño de las pruebas y el ambiente.

A partir de todo lo anterior, y también considerando los costos generales del proyecto para este tipo de misiones, que están basados en el tiempo y los recursos humanos principalmente, nace una necesidad de agilizar el proceso de verificación de calidad del software de vuelo para las siguientes misiones SUCHAI II, SUCHAI III y PlantSat. Estos nanosatélites fueron recientemente lanzados en abril de este año.

Fuzz testing en el software de vuelo de los nanosatélites SUCHAI

Fuzz testing es una técnica automatizada de testing que consiste en generar *inputs* aleatorios automáticamente para encontrar vulnerabilidades en un software [14]. En esta necesidad de encontrar una técnica de testing de software ágil y automatizable, estudiamos el uso

En total ejecutamos 25.760 secuencias en el software de vuelo de las misiones SUCHAI II, SUCHAI III y PlantSat.

de *fuzz testing* en el software de vuelo que se ocupa para las misiones de los nanosatélites SUCHAI [10]. Gracias a su diseño y aprovechando la arquitectura del software, este puede ser intervenido enviando comandos y observando su comportamiento. Así, la técnica fue implementada generando un conjunto de comandos aleatorios y parámetros. La aleatoriedad del número de comandos, el número de parámetros, la composición de los caracteres de los comandos, y la composición de los caracteres de los parámetros da lugar a cuatro estrategias propuestas dentro de este trabajo. En la Figura 4 se muestra y describe la lógica de esta implementación.

La ejecución de esta técnica de testing se llevó a cabo bajo un proceso sistemático que consistió en 8 *sprints*[2] con una duración de 1 a 2 horas cada sesión. Durante estas sesiones se entregó un reporte al equipo de desarrollo con las secuencias de comandos y parámetros que hacían que el software de vuelo terminara su ejecución debido a una falla. A partir de estos reportes, al comienzo de las sesiones se hizo una reproducción manual de las secuencias a estudiar para verificar el término del programa. De esta forma, cada una de las secuencias que producían fallas fueron reportadas en el repositorio de trabajo del equipo. Una vez que una falla era reportada, se procedía a identificar el problema específico y a resolverlo. Todas las fallas fueron caracterizadas a partir de un seguimiento a los cambios realizados al código en el sistema de control de versiones y un breve cuestionario que se le hizo al equipo de desarrollo. El cuestionario consistió en las tres preguntas base, las cuales se enumeran a continuación:

1. ¿Cuán importante es la falla?
2. ¿Cuán difícil es la falla de encontrar?
3. ¿Cuán difícil es la falla de arreglar?

Finalmente, para prevenir la ocurrencia de la misma fallas en nuevas versiones del software, las secuencias identificadas sirvieron como casos de prueba y se agregaron como tests unitarios a un servidor de integración continua utilizado para hacer pruebas automáticas.

Para cada estrategia ejecutamos secuencias de diferentes tamaños. Cada secuencia podía contener 5, 10, 50 o 100 comandos. En total ejecutamos 25.760 secuencias en el software de vuelo de las misiones SUCHAI II, SUCHAI III y PlantSat. Inicialmente, esta ejecución duró aproximadamente tres días. En una réplica de este experimento con las mismas secuencias, la ejecución duró aproximadamente dos días. La réplica de este experimento fue llevada a cabo para analizar el tiempo de ejecución en un computador con otras características, incluyendo una mayor capacidad de almacenamiento y procesamiento.

Resultados y principales conclusiones

A partir de la ejecución de *fuzz testing* en el software de vuelo de SUCHAI encontramos 12 secuencias que hacían que el programa fallara en su ejecución. Cada una de estas secuencias hacía fallar al programa debido a la ejecución de un comando específico. El equipo de desarrollo identificó 10 comandos en particular que producían fallas. La Figura 5 muestra la frecuencia de ocurrencia de los comandos que aparecen en las secuencias que hicieron que fallara la ejecución del software, clasificados por cada uno de los módulos del programa, los cuales están representados por un cierto color. Si un comando está escrito en color rojo en el eje X, significa que se identificó una falla con ese comando. De la misma figura se pueden identificar los comandos que hicieron fallar la ejecución del software de vuelo. Siete de los 10 comandos identificados aparecían con más frecuencia en las secuencias.

Las principales conclusiones de la caracterización de fallas indican que 11 de las 12 fallas fueron consideradas fáciles de encontrar por el equipo de desarrolladores y que 8 de las 12 fallas fueron consideradas fáciles de solucionar. Sin embargo, 8 de los 10 comandos encontrados fueron considerados críticos para el software porque se ejecutan en el computador a bordo directamente. El máximo número de líneas de código y de funciones modificadas para arreglar una falla fue 375 y 10, respectivamente.

El trabajo que realizamos [15] presentó el impacto de utilizar *fuzz testing* para verificar la correcta operación de un software de vuelo de nanosatélites. La evaluación fue llevada a cabo en una serie de tres nanosatélites que en ese momento estaban siendo desarrollados en la Universidad de Chile (SUCHAI II, SUCHAI III y PlantSat). De esta forma se presenta una metodología que hemos desarrollado para aplicar *fuzz testing* al software de vuelo de nanosatélites como parte de una proceso ágil de desarrollo de software de vuelo de CubeSats. Además se destaca y discute los desafíos que enfrentamos y describe los principales requerimientos para implementar esta técnica en proyectos similares, y presenta un estudio de caso convincente aplicando técnicas modernas de testing a un software embebido crítico lo cual, creemos, abre un nicho en

2 Periodos definidos donde un equipo trabaja para completar una cierta cantidad de trabajo. Corresponde a la metodología ágil Scrum.

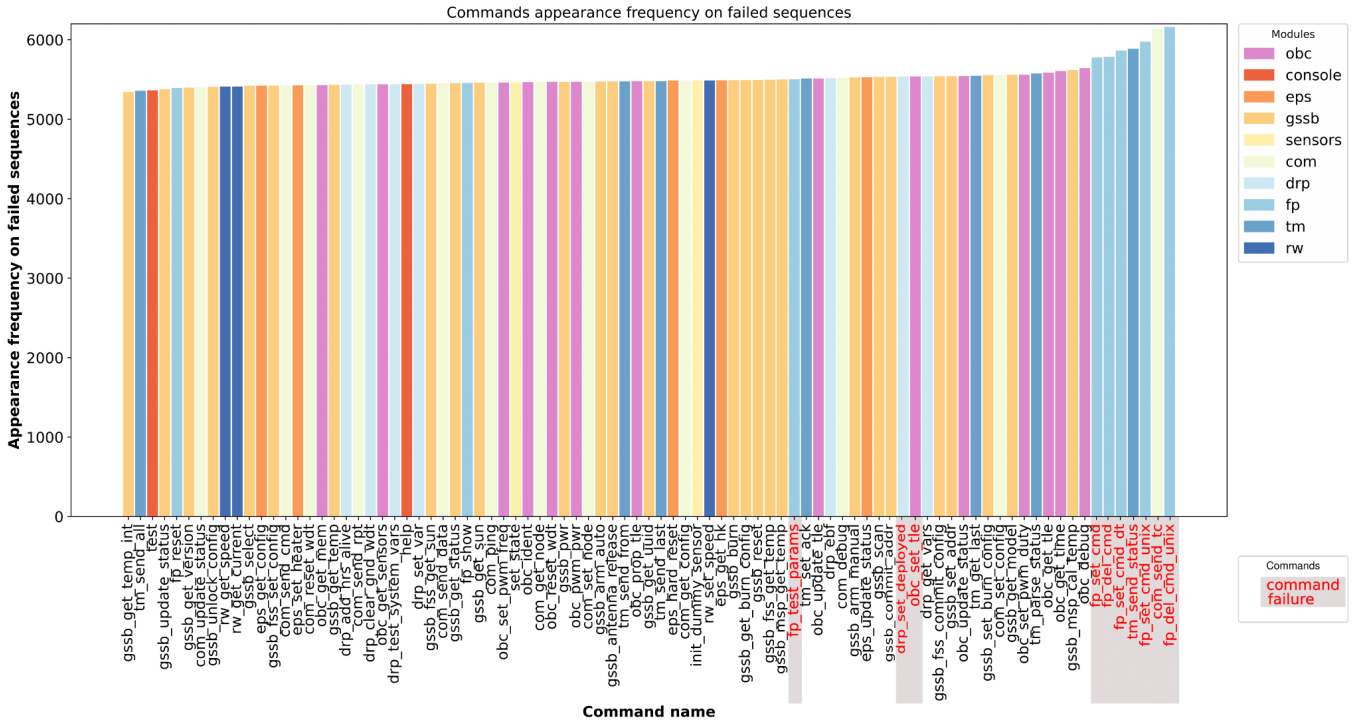


Figura 5. Frecuencia de aparición de los comandos en las secuencia que hicieron fallar la ejecución del software, clasificados por tipo de comando (módulo). Los comandos que hicieron fallar la ejecución del software de vuelo de las misiones SUCHAI están escritos en rojo en el eje X (*identified command failure*).

el campo de testing de software de vuelo de nanosatélites.

La experiencia de este trabajo nos indica que preparar el software de vuelo para líneas de montaje más grandes puede ser desafiante y requiere agilidad y automatización con respecto al testing para alcanzar la robustez deseada. Sin embargo, el testing de software de vuelo es aún un área incipiente. Actualmente, solo han sido reportadas experiencias esporádicas, y, hasta el momento, se han propuesto prácticas de testing de bajo costo no dedicadas. Un software de vuelo es un componente altamente valioso, y las técnicas para mejorar su robustez merecen ser cuidadosamente estudiadas y diseminadas.

Mientras el área de ingeniería de software ha producido muchas técnicas, incluyendo *fuzz testing*, no hay un reporte público de su uso en software de vuelo de CubeSats. En ese sentido, nuestras contribuciones mejoran las prácticas de testing de software de vuelo, las cuales actualmente son llevadas a cabo de una forma no automatizada. Nuestras observaciones desde diferentes agencias de investigación que desarrollan software de vuelo dejan ver una brecha entre la forma en que el software de vuelo es desarrollado y las técnicas propuestas por la comunidad de ingeniería de software.

En el trabajo de tesis “Systematic Fuzz Testing Techniques on a Nanosatellite for Agile Mission Development” [16] también se presenta un análisis de esta aplicación a otros software de vuelo, un caso de estudio para el software de vuelo “FPrime”, que fue desarrollado por la NASA, y el estudio de la ejecución de las mismas estrategias en otras arquitecturas de las misiones SUCHAI como una extensión de este trabajo. Los próximos pasos de esta investigación debieran incluir el estudio de estrategias más avanzadas con un enfoque en una identificación de fallas guiadas por la cobertura del código u otras métricas de análisis dinámico. ■

A partir de la ejecución de fuzz testing en el software de vuelo de SUCHAI encontramos 12 secuencias que hacían que el programa fallara en su ejecución.

REFERENCIAS

[1] T. Villela, C. A. Costa, A. M. Brandão, F. T. Bueno, and R. Leonardi, "Towards the thousandth cubesat: A statistical overview", International Journal of Aerospace Engineering, vol. 2019, 2019.

[2] E. National Academies of Sciences, Medicine, et al., Achieving science with CubeSats: Thinking inside the box. National Academies Press, 2016.

[3] D. L. Dvorak, "NASA Study on Flight Software Complexity", in AIAA Infotech@Aerospace Conference and AIAA Unmanned...Unlimited Conference, (Reston, Virgina), p. 264, American Institute of Aeronautics and Astronautics, 2009.

[4] J. Finnigan, "A scripting framework for automated flight sw testing: Van allen probes lessons learned", in 2014 IEEE Aerospace Conference, pp. 1–10, 2014.

[5] J. Kiesbye, D. Messmann, M. Preisinger, G. Reina, D. Nagy, F. Schummer, M. Mostad, T. Kale, and M. Langer, "Hardware-In-The-Loop and Software-In-The-Loop Testing of the MOVE-II CubeSat", Aerospace, vol. 6, p. 130, 2019.

[6] J. Schoolcraft, A. T. Klesh, and T. Werne, "MarCO: Interplanetary Mission Development On a CubeSat Scale", in SpaceOps 2016 Conference, SpaceOps Conferences, American Institute of Aeronautics and Astronautics, 2016.

[7] J. A. Ledin, "Hardware-in-the-loop simulation", Embedded Systems Programming, vol. 12, pp. 42–62, 1999.

[8] S. Jeong, Y. Kwak, and W. J. Lee, "Software-in-the-loop simulation for early-stage testing of autosar software component", in 2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN), pp. 59–63, IEEE, 2016.

[9] D. José Franzim Miranda, M. Ferreira, F. Kucinskis, and D. McComas, "A Comparative Survey on Flight Software Frameworks for 'New Space' Nanosatellite Missions", Journal of Aerospace Technology and Management, p. e4619, 2019.

[10] C. E. González, C. J. Rojas, A. Bergel, and M. A. Díaz, "An Architecture-Tracking Approach to Evaluate a Modular and Extensible Flight Software for CubeSat Nanosatellites", IEEE Access, vol. 7, pp. 126409–126429, 2019.

[11] S. Nakajima, J. Takisawa, S. Ikari, M. Tomooka, Y. Aoyanagi, R. Funase, and S. Nakasuka, "Command-centric architecture (c2a): Satellite software architecture with a flexible reconfiguration capability", Acta Astronautica, vol. 171, pp. 208–214, 2020.

[12] C. Boshuizen, J. Mason, P. Klupar, and S. Spanhake, "Results from the planet labs flock constellation", in Proc. AIAA/USU Conf. Small Satell., pp. 1–8, 2014.

[13] I. F. Akyildiz and A. Kak, "The Internet of Space Things/CubeSats", IEEE Netw., vol. 33, no. 5, pp. 212–218, 2019.

[14] P. Godefroid, "Fuzzing: hack, art, and science", Communications of the ACM, vol. 63, pp. 70–76, 2020.

[15] T. Gutiérrez, A. Bergel, C. E.González, C. J. Rojas, & M. A. Díaz. Systematic Fuzz Testing Techniques on a Nanosatellite Flight Software for Agile Mission Development. IEEE Access, 9, pp. 114008-114021, 2021.

[16] T. Gutiérrez Rojo, Systematic fuzz testing techniques on a nanosatellite flight software for agile mission development, 2022.