Software crítico para sistemas espaciales

2024/25

Master Universitario en Enxeñaría Aeroespacial

Arno Formella
Miguel Ramón González Castro
Manuel Pérez Cota

Departamento de Informática
Universidade de Vigo

24/25

una cosa difícil

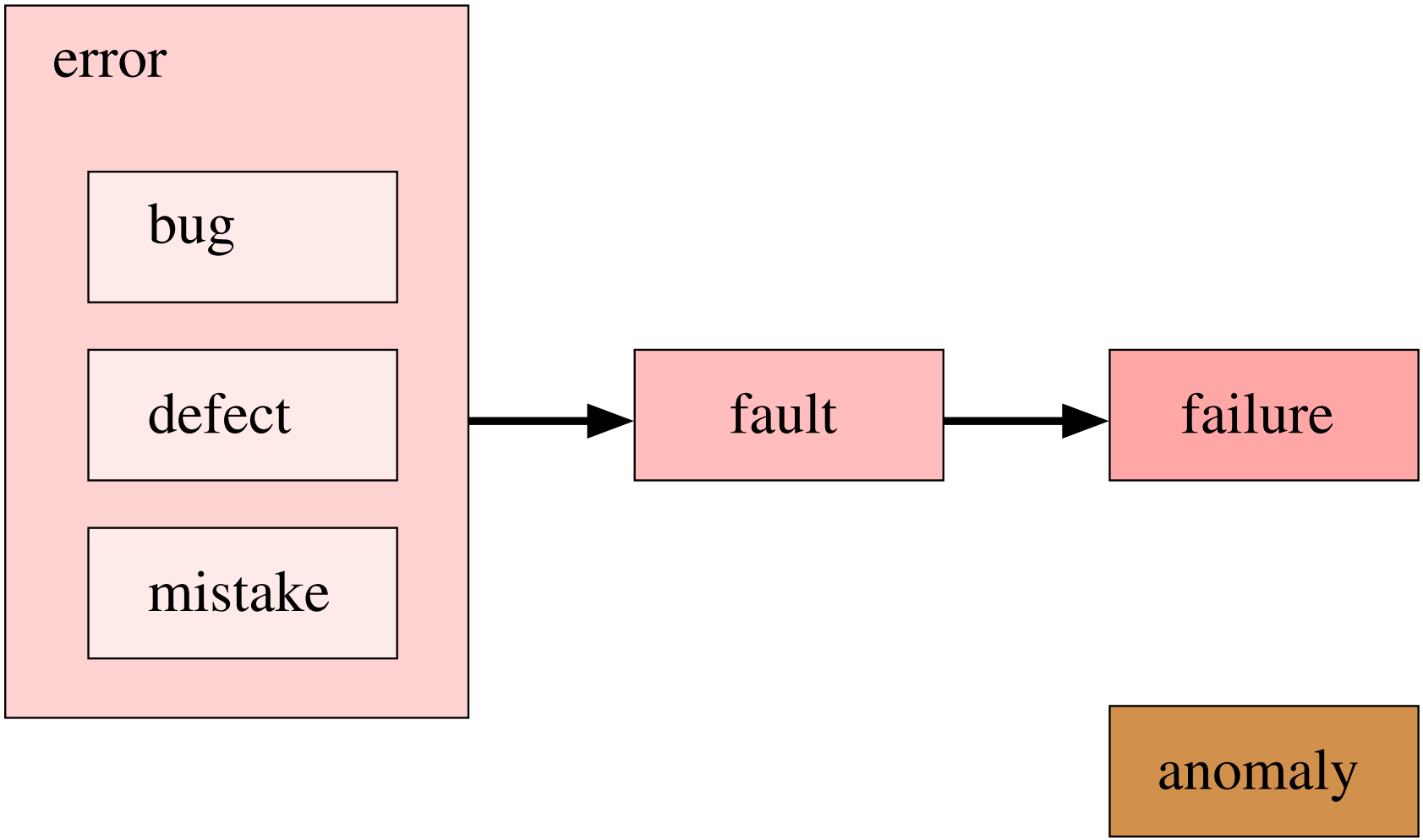
mirando muchas fuentes, parece difícil concretar el significado de las siguientes palabras (y sus traducciones a otros idiomas):

English	**castellano**	**Deutsch**
bug	bug	Bug
defect	defecto	Defekt/Mangel
mistake	malendentido	Irrtum
error	error	Fehler
fault	fallo	Versagen
failure	malfuncionamiento	Fehlverhalten
anomaly	anomalía	Anomalie

vamos a concretar la semántica para estas clases...

(hay más palabría: *mishap*, *incidence*, *glitch*, *flaw*, ...)

relación entre los términos



bug

- un bug es un error en el código del software
- un bug se puede encontrar y remediar/corregir
- un bug puede ser la consecuencia de un malfuncionamiento de una herramienta de generación de código automático
- un bug puede provocar grandes desastres
- si hay un bug, sufre el usuario, no el o la programadora.

defecto (*defect*)

- un defecto está localizado en el hardware (no en el software)
- un defecto puede existir desde el principio o por rotura durante el uso
- un defecto se puede reparar (por ejemplo, sustituyendo una parte defectuosa)
- existen defectos transientes, es decir, se manifiestan y desaparecen, o re-aparecen con cierta probabilidad
- existen defectos que se puede *remediar* en software
- defectos sueles ser raros en sistemas críticos asumiendo que componentes fiables y comprobados (incluso los procesos de montaje)
- un defecto se puede (muchas veces) simular en software
- no todos los defectos causan malfuncionamiento, especialmente si hay redundancia implementada, o se usa, por ejemplo, autosaneamiento implementado en hardware

equivocacion (*mistake*)

- un malendentido puede ser la consecuencia de una interpretación incorrecta de los requisitos
  (ejemplo: no se implementa una ordenación estable)
- un malendentido puede ser la malentendido del funcionamiento de un algoritmo
  (por ejemplo por falta de documentación completa)
- un malendentido de un requisito o una especificación por ser ellos imprecisos, incompletos, o ambíguos

error (*error*)

- un error es o bien un bug, o bien un defecto, o bien un malentendido
- un error se puede detectar mediante comprobaciones (*testing* o *debugging*)
- un análisis detallado de un error debe concluir si se trata de un bug, un defecto, o de un malendentido
- un análisis detallado de un error, y asumiendo que no haya malendentido ni bug, debe detectar la parte del sistema que está defectuoso

fallo (*fault*)

- un fallo es la consecuencia observada de un error, es decir, de un bug, de un malentendido o de un defecto
- un análisis de un fallo debe concluir con el error (o una lista de posibles errores)
- tal fallo debe documentarse en la revisión con una lista de de las observaciones y pruebas realizadas, y, si es posible, unas posibles causas para tal fallo
- la documentación del fallo debe contener una descripción como reproducir el fallo (incluyendo datos, configuraciones, y comandos etc.)
- en sistemas complejos que mantienen estados e historias un fallo puede aparecer solo en ciertas configuraciones
- fallos pueden estar ligados al hardware (y/o capas bajas) donde se ejecuta el software
- la documentación del fallo ayuda a depurar el software, con alguna estrategia de comprobaciones (con o sin modificaciones del código)

(divertidos) tipos de fallos (aún llamados bugs)

hay unos nombres divertidos derivados de personas famosas para caracterizar cierto comportamiento aparente de fallos:

- Bohrbug (*Niels Bohr*):
  un error que aparece siempre bajo criterios bien definidos y escenarios de condiciones conocidos, pues el clásico bug
- Heisenbug (*Werner Heisenberg*):
  un error del cual se sabe que está (se ha observado algún comportamiento erroneo) pasa si añadimos código para depurar/observar, el fallo desaparece o cambia comportamiento, quitando el código, aparece de nuevo en versión original
- Mandelbug (*Benoit Mandelbrot*):
  un error cuyas consecuencias parecen caóticas, a veces pasa una cosa otras veces otra, sin dedución lógica posible aparentemente

(divertidos) tipos de bugs (aún llamados bugs)

- Schrödinbug (*Erwin Schrödinger*):
  un error que solamente aparece observable cuando alguien depurando el código nota que en principio el programa no debe haber funcionado nunca, es decir, hasta que se observa por primera vez el fallo, todo funciona, a partir de ahi nunca jamás
- Hindenbug (*Paul von Hindenburg*):
  un error con consecuencias catastróficas
- Higgs-bugson (*Peter Higgs*):
  un error del cual se sabe que debe existe en el código pero parece casi imposible construir un caso de prueba para que se manifieste de forma concreta

malfuncionamiento (*failure*)

- un malfuncionamiento occure cuando el software (el sistema) no se comparta como exigido por los requisitos
- es decir, se observa un comportamiento diferente a lo requirido
- un malfuncionamiento puede tener consecuencias catastróficas con responsabilidades subyacentes
- un malfuncionamiento tiene como causa un fallo (observable si el software está intrumentado para generar tal información)
- para tal instrumentación se suele generar una versión de depuración o activar la generación de vitácoras más extendidas

anomalía (*anomaly*)

- una anomalía es un comportamiento observado que no concorda con los requisitos o especificaciones pero no directamente causado por un error sino indirectamente como efecto segundario observando el sistema en su entorno global
- algunas razones pueden ser: recolección de memoria no usada (*garbage collection*, carga actual del sistema (*system load*, procedimientos de mantenimiento (*mantainance procedure*, etc.
- incógnitas desconocidas pueden ser la causa de una anomalía
- una anomalía simplemente puede desaparecer una vez estando en un entorno controlado

y además

- veremos para todos los conceptos unos ejemplos más en adelante
- hay que ser pacientes con los compañeros y compañeras en un proyecto, no insistir demasiado en una semántica concreta de las palabras, sino consensuar en el equipo teniendo en cuenta el estándar empleado
- a veces se usa también la palabra síntoma (*symptom*) que podemos concretar como una parte de la descripción del fallo una vez haber observado detectado el error
- a veces se usa también la palabra causa fundamental (*root cause*) que podemos concretar como el error principal que inicia toda la cadena de causa-efecto observada

problemática de errores sin malfuncionamiento

- no todo los fallos necesariamente producen un malfuncionamiento
- a veces los efectos son menores como pérdida de rendimiento o precisión
- por la existencia de redundancia, la observación del efecto de un error puede quedar sin advertencia
  (de redundancia hableremos más en adelante de nuevo)
- un reinicio o reconfiguración del sistema puede enmascarar el problema subyacente

incógnitas desconocidas (*unknown unknowns*

> *There are known knowns. These are things we know that we know. There are known unknowns. That is to say, there are things that we know we don't know. But there are also unknown unknowns the ones we don't know we don't know. (Donald Rumsfeld, 2002)*

(*unknown knowns* serían secretos (conocimientos) que conocen algunos pero no los conoce (entiende) la persona que habla, típica situación en un examen :-)

Incógnitas desconocidas se convierten en conocimiento de posible causa de un malfuncionamiento una vez haberles observado y analizado por primera vez

FDIR (*fault detection, isolation, and recovery*)

es buena idea que se usa mucho en sistemas críticos durante la operación del dispositivo:

- tener un modelo de posibles fallos (especialmente pensando en el uso y en el entorno)
- tener un plan que hacer si tal fallo ocurre
- simular el comportamiento del sistema (por ejemplo con un gemelo digital) en tal caso, es decir, con inyección del fallo en la simulación
- si se detecta un fallo, se intenta que no tenga más efecto en el resto del sistema, es decir, se aísla el causante (si es posible)
- o bien se intenta remediar el problema, o bien se intenta seguir con el sistema sin está funcionalidad fallida
- del FDIR hablaremos otra vez más en adelante

razones del malfuncionamiento

- exceso de confianza en digitalización
  - se cree que software es correcto hasta que se comprueba el contrario, pero ES al revés: software es incorrecto hasta que se comprueba el contrario
- falta de endender los riesgos asociados con software
  - muchos errores se basan en fallos en la especificación de los requisitos menos en el propio código, los errores en software son muy diferentes a defectos en sistemas mecánicos o electrónicos
- confusión entre fiabilidad y seguridad
  - el problema no es que falle una componente, sino que fallen las interacciones entre los componentes en un sistema complejo

razones del malfuncionamiento

- las interfaces entre humanos y máquinas pueden ser causa de una interpretación errónea de la situación
  - distracción, demasiada información, presentación de datos en formato que confunde un humano etc.
- especificaciones incorrectas e incompletas
  - falta de especificar lo que no debe ocurrir
  - ojo: hay gente/normas que incluso recomendan no hacer lo
- proceso de revisión incompleto y superficial
  - la semántica de texto escrito (en especificaciones) es difícil de relacionar con el software implementado, por eso, disciplina y experiencia son imprecidibles

razones del malfuncionamiento

- ingeniería de la seguridad del sistema inadecuado o no continuado
  - muchos posibles problema se detectan a lo largo del diseño, y se tiene que atender de nuevo desde el principio (validación del sistema)
- ignorar reglas de diseño de software crítico
  - eliminación de funcionalidades no requeridos, separación en módulos, codificación de errores y datos, control de entrada/salida de componentes, etc
- reutilización de software
  - incorporar otro software requiere un análisis total respecto a todos los (nuevos) requisitos

razones del malfuncionamiento

- demasiada complejidad en el software
  - para estar preparada por cualquier futuro requisito o cambio de parámetro se aumenta la complejidad de software una tendencia entre ingenieros de software, *keep it simple*
- uso de gemelos digitales en entornos no realistas
  - no se puede detectar problemas si no se simula cerca de la realidad, *fly what you test, test like you fly*
- falta de recolectar información del estado del sistema especialmente relacionada con la seguridad
  - a veces no hacen falta protocolos específicos (*sophisticated incidence tracker software*), sino comunicación simple entre participantes con evidencias almacenadas, p.ej. email

razones del malfuncionamiento

- sobre-creencia en la automatización digital
- confundir seguridad con fiabilidad
  - puede ser que los requisitos que cumple el software sean incompletos, erroneos, o inseguros)
- sobrestimar la redundancia
- subestimar el riesgo a lo largo del tiempo
  - si algo no ha fallado durante mucho tiempo, no significa que no falle cuando las circunstancias se presten;
  - normalmente el riesgo aumenta con el tiempo

razones del malfuncionamiento

- ignorar alertas tempranas
- ingeniería inadecuado del factor humano y sus procesos cognitivos
  - humanos suelen *intentar* actuar correctamente en situaciones críticas,
  - pero quizá con un modelo mental incorrecto o inadecuado
- no-revisar adecuadamente todos los componentes en un sistema y los planes por ejecutar durante el proyecto