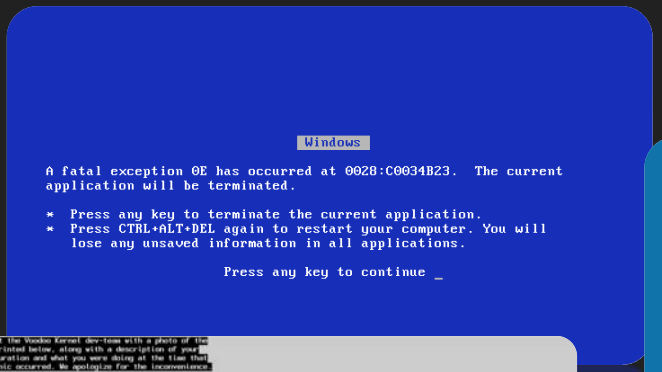
Herrar es *umano

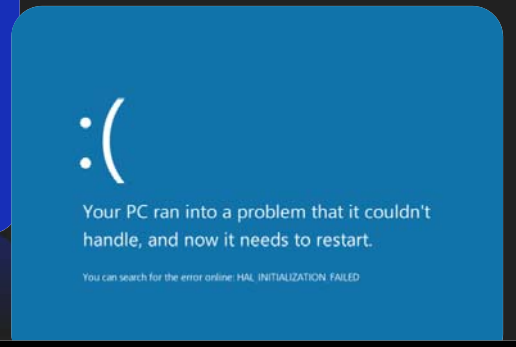
Sobre el error en informática

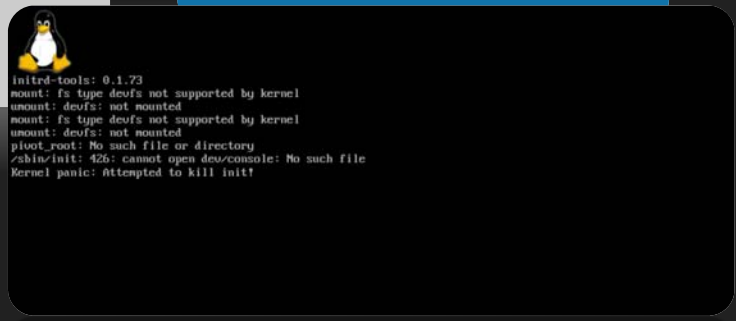
Ángel Luis Rubio García
Seminario de Informática “Mirian Andrés”
Universidad de La Rioja
7 de febrero de 2017

BSO y BSOD


Uh Oh
You're screwed.
Damn


Windows
A fatal exception 0E has occurred at 0028:C0034B23. The current application will be terminated.
* Press any key to terminate the current application.
* Press CTRL+ALT+DEL again to restart your computer. You will lose any unsaved information in all applications.
Press any key to continue _


:(
Your PC ran into a problem that it couldn't handle, and now it needs to restart.
You can search for the error online: HAL_INITIALIZATION_FAILED

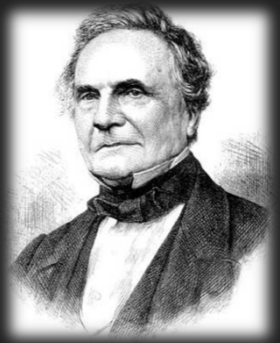

initrd-tools: 0.1.73
mount: fs type devfs not supported by kernel
umount: devfs: not mounted
mount: fs type devfs not supported by kernel
umount: devfs: not mounted
pivot_root: No such file or directory
/sbin/init: 426: cannot open dev/console: No such file
Kernel panic: Attempted to kill init!

Errors everywhere

Errors in all times

« If trials of three or four simple cases have been made, and are found to agree with the results given by the engine, it is scarcely possible that there can be any error »

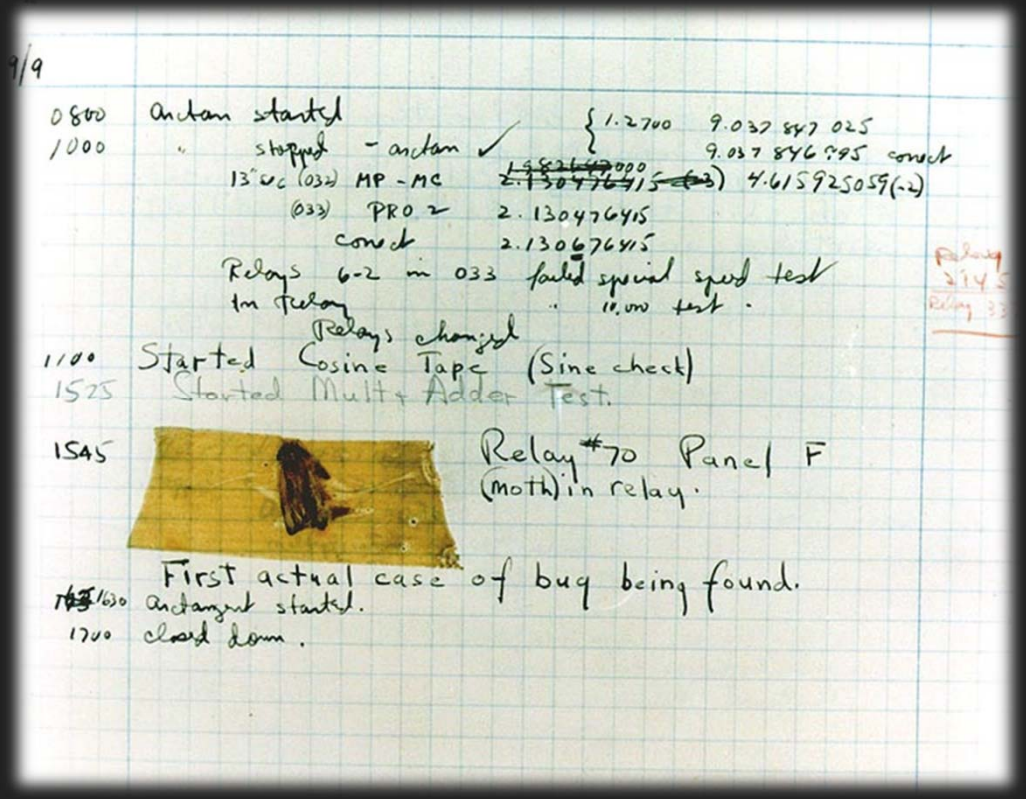
On the mathematical powers of the calculating engine
Charles Babbage, 1837



vía Tomas Petricek. *Miscomputation in software: learning to live with errors.*
http://tomasp.net/academic/drafts/failures/

vía Jonathan Heras (email)

La leyenda del bug

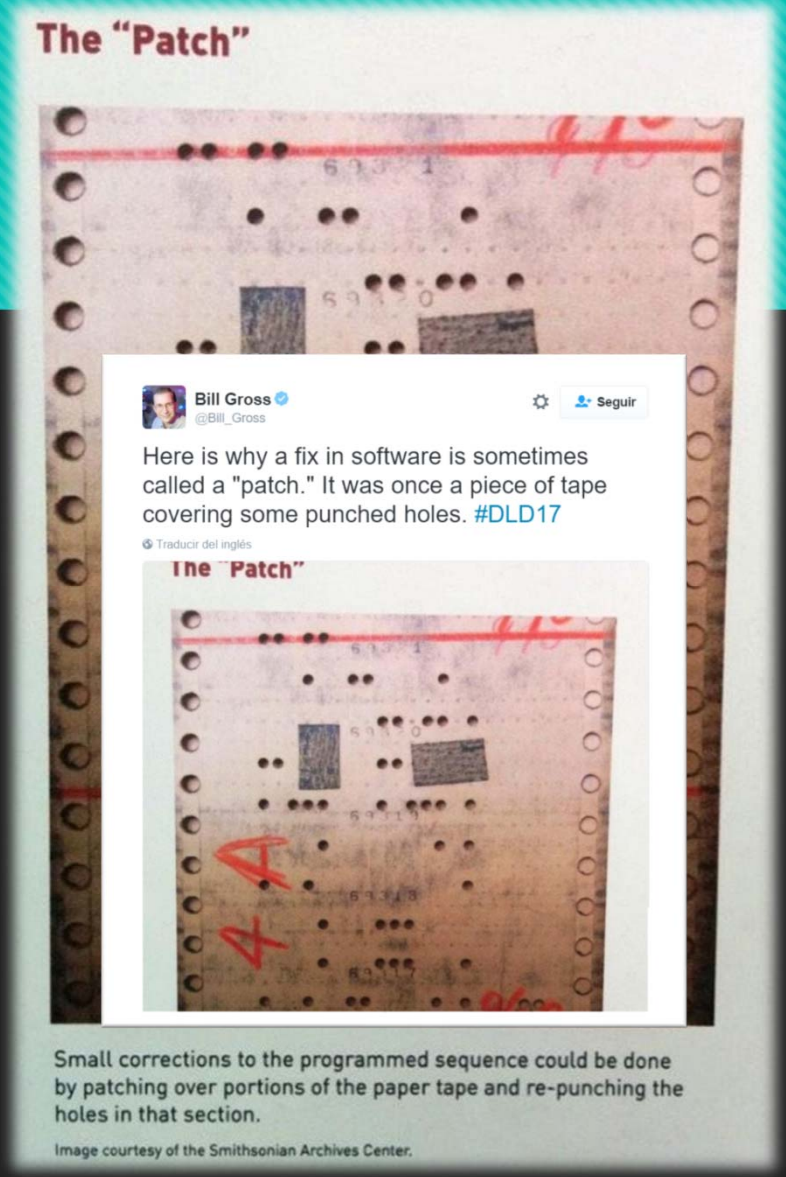


Grace Murray Hopper (1906-1992)

Doodle
9/12/2013

El origen del parche

https://twitter.com/Bill_Gross/status/820650555626389509



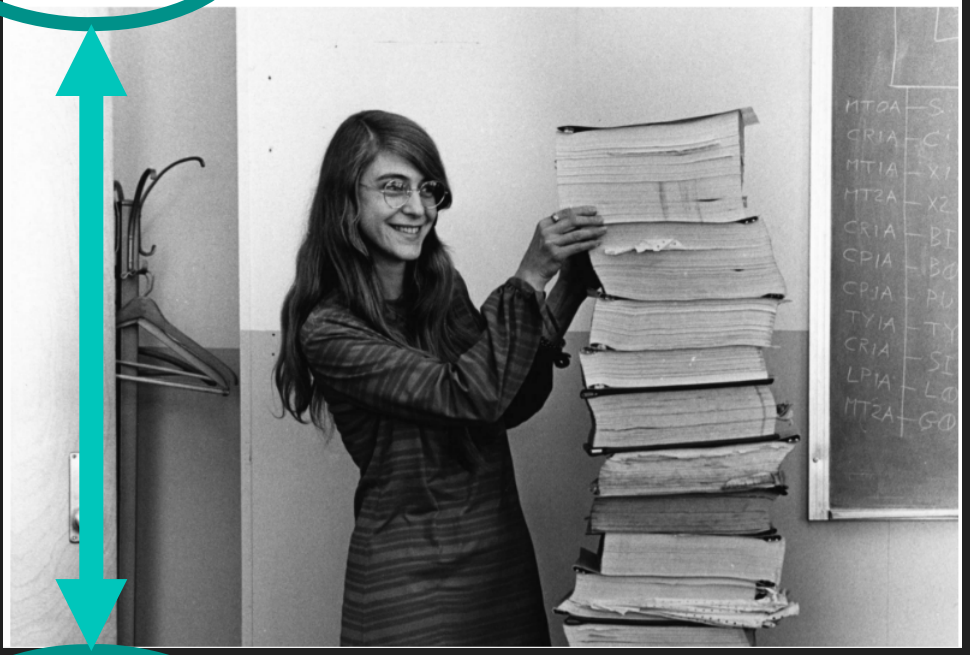
EL PAÍS TECNOLOGÍA

Margaret Hamilton, la mujer que nos llevó a la Luna

Su visión de cada misión como un sistema en el que interactuaban muchos factores fue fundamental para el correcto desarrollo del software de a bordo de las misiones Apolo

JAVIER PEDREIRA 'WICHO' | MICROSIERVOS

16 ENE 2017 - 13:12 CET



http://tecnologia.elpais.com/tecnologia/2017/01/09/actualidad/1483980291_213857.html

Una de las cosas que defendía, por ejemplo, era la necesidad de diseñar los programas a prueba de errores de quienes los iban a usar, a pesar de la oposición de la misma NASA, que no veía tal necesidad, pues decía que los astronautas no se iban a equivocar nunca... Hasta que Jim Lovell consiguió borrar todos los datos de navegación del ordenador de a bordo durante la misión Apolo 8 al introducir un comando equivocado.

Las preguntas

- ¿Se pueden clasificar los errores?
- ¿Existen categorías de error?
- ¿Y niveles de error?

¿Es posible elaborar una ontología de los errores en informática?

La motivación

Encontrar esa ontología sería bueno porque…

1. Se facilitaría la enseñanza de (al menos una parte de) la informática
   - Si los errores que se pueden cometer estuvieran categorizados se podría estructurar su aprendizaje…
   - …para encontrar formas que evitas… …os

¿Qué es **"mejor"** software?

Un software *ideal*

- Un producto software es/debe ser una solución para un problema, para una necesidad
- Cuando se *finaliza el desarrollo*, hay que presuponer que esta solución es 100% correcta (sin errores ni fallos de ningún tipo)

¿Mejorar un software perfecto?

- **En esta situación teórica, *mejorar el software* solo puede significar *mejorar el proceso de desarrollo de software***
- Esta mejora siempre lo será para los desarrolladores
  - Desarrollar de la forma que les resulte más sencilla/cómoda
  - Desarrollar con menos costes (tiempo, dinero): esta mejora además se puede *trasladar a los clientes* (**ventaja competitiva**)

La gran verdad/engaño del software

- Se asume, sin sonrojo, que es *razonablemente posible* que una solución software no sea 100% correcta, y que pueda contener fallos
- ¿Es esto también cierto para otros *productos*?
  - Productos ‘con tara’
  - Averías ‘por el uso’

Usuario de software: cliente cautivo

- Cualquier producto tiene su garantía… menos el software
- El concepto relacionado *software assurance* es otra cosa:

  "Nivel de certeza en que el software está libre de vulnerabilidades, ya sea que hayan sido diseñadas intencionalmente en el software o insertada accidentalmente en cualquier fase de su ciclo de vida, además de que el software funcione como se tiene previsto"

  National Information Assurance Glossary. Instruction No. 4009 Committee on National Security Systems

  (vía https://es.wikipedia.org/wiki/Garantía_de_software)

Mejorar el software: detectar y reducir errores

- Asunción realista:
  Cualquier software (*potencialmente*) contiene fallos
- *Por tanto*, ***mejorar el software*** **tiene** necesariamente **que significar encontrar medios (durante el desarrollo o tras él) para detectar y reducir el número de fallos**

Enfoques de ingeniería

- Análisis (cuantitativo, estadístico-probabilista) de los fallos → **Fiabilidad (*reliability)*** de software
  - Relacionado en particular con el concepto de sistema tolerante a fallos (***fault-tolerant systems)***
- Detección de los fallos → **Prueba (*testing*)** de software
- Prevención de los fallos → **Control de calidad (*quality assurance*)** de software

Un acercamiento naif

- Todo lo anterior es muy posterior...
- ... porque yo lo que me pregunto no es cómo analizar o prevenir los fallos, sino algo previo, que es...
- **¿Qué tipos de errores hay/puede (potencialmente) haber?**

Clasificaciones ortogonales

- Según el **nivel**
  - *Sintáctico/gramatical*
  - *Semántico*
- Según la **gravedad**
  - *Crítico*
  - *Grave*
  - *Leve*
- Según la ***fase***
  - Error en/de *diseño*
  - Error en/de *codificación*
  - Error en/de *uso*

Errores en la primera fase: diseño

- Es muy frecuente (oír) decir: “Es un error de diseño”
- Pero, ¿qué es eso? ¿Existen errores de diseño *intrínsecos?*
- *En mi opinión,* estos errores solo pueden ser una **mala interpretación/traslación de la especificación**
- Puede haber diseños *poco eficientes*, pero ¿erróneos?
- También se aplica al *diseño de algoritmos*

Errores en la segunda fase: codificación

- En primer lugar, podríamos considerar errores en los modelos/tipos de datos (tablas relacionales, esquemas XML, documentos JSON, etc.)...
- ... pero de nuevo son más bien una **mala interpretación/traslación, en este caso del diseño**

Errores en la segunda fase: codificación

- En segundo lugar, los errores de programación *no-sintácticos*
  - Los sintácticos deben ser detectados en edición o como tarde, en interpretación/compilación
- Son los que más fácilmente entendemos y estamos habituados a manejar
- Variables no inicializadas (objetos no creados), bucles infinitos, ‘cases’ no exhaustivos...

Excepciones y errores

- Las excepciones (y su manejo) son un sistema de control de errores (que parecen) “gallegos”
- Son “*errores*” puesto que manifiestan una situación anómala, o poco frecuente
- Pero “*no son errores*” en el sentido en el que los estamos intentando explicar, puesto que “*están previstos*” (ahora volvemos)

Excepciones y errores

« *While **try-catch-finally** is conceptually simple, it has the most complicated execution description in the language specification and requires four levels of nested "if's" in its official English description. In short, it **contains a large number of corner cases that programmers often overlook*** »

W. Weimer and G. C. Necula, "Exceptional Situations and Program Reliability"
ACM Transactions on Programming Languages and Systems, Vol. 30, No. 2, March 2008.

Errores en la tercera fase: uso

- Son los más 'divertidos' porque...
  NO DEBERÍAN PODER PRODUCIRSE
- Un software correcto/bien diseñado debería *prever* cualquier potencial error de usuario y tener establecidos métodos para *evitarlos*
  - Ejemplo típico: lista de opciones vs. entrada libre (muy relacionado con usabilidad)
- Pero en la práctica, es en uso cuando se detectan...

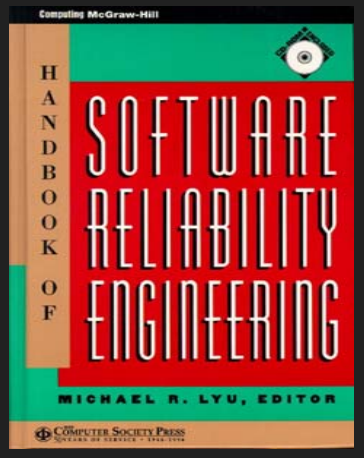
Vuelta atrás

- Un usuario percibe un error si el software *no se comporta como se/él espera*
  - Con matices en función de la propia experiencia/expertise del usuario
- Es necesario volver al principio de todo, es decir, a la *especificación* (o más atrás, a los *requisitos*)
- Y de nuevo, ¿tiene sentido hablar de errores en una especificación (o en los requisitos)?

Un guiño al grupo

- Se podría hablar de especificaciones *incompletas*
  - Por ejemplo, si la especificación no detalla que cosas “no debe hacer/permitir” el software (típico problema en seguridad)
- ¿La especificación formal al rescate?
  - No, porque incluso una especificación formal *correcta* puede ser *incompleta*
  - ¿Cómo detectar esa incompletitud? ¿Especificando la especificación? **No, sin caer en una recursividad infinita…**

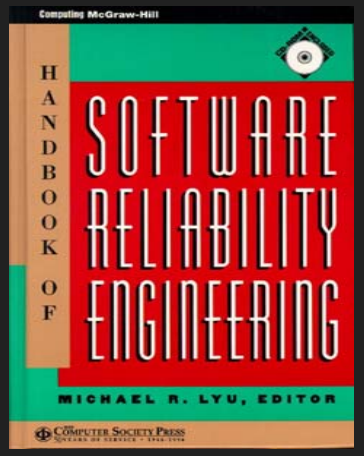
Alguna referencia... y definiciones



1995

- **Avería (*failure*)**: ocurre cuando el usuario percibe que el software deja de proporcionar el servicio esperado
- **Corte (*outage*)**: caso especial de avería dada por una pérdida o degradación del servicio durante un periodo de tiempo (normalmente asociado a fallos de hardware o alimentación)

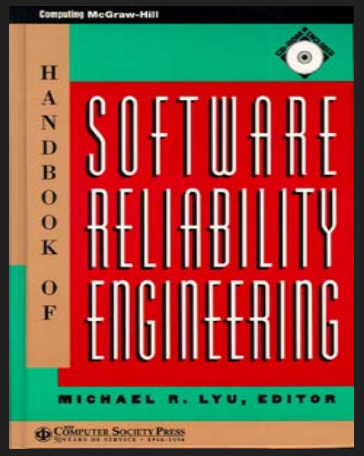
Fenomatizando



1995

- Según estas definiciones, tanto la **avería** como el **corte** son *efectos, elementos que se perciben*
- [En términos *fenomáticos* son **noemas**]
- Pero lo relevante son las causas [**noesis**]

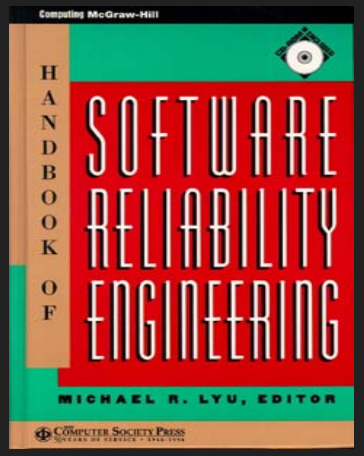
Averías vs. fallos



1995

- **Fallo (fault)**: causa de una avería (*failure*) del software o detección de un problema interno
- Según esta definición, un fallo (*fault*) puede estar *latente* en el software por tiempo indefinido (incluso sin llegar a manifestarse), si no se llega a producir la avería ni se detecta el problema

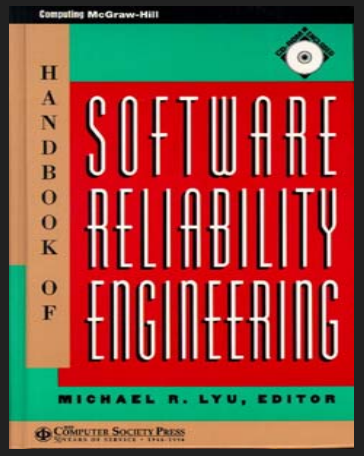
Defectos y errores



1995

- **Defecto (*defect*)**: término genérico para referirse tanto a averías (*failures; efectos; noemas*) como a fallos (*faults; causas; noesis*)
- **Error (*error*)**: discrepancia entre un valor calculado, observado o medido y el valor verdadero, especificado o teóricamente correcto

Lo humano es la equivocación



1995

- **Equivocación (mistake)**: acción humana que como resultado produce un software que contiene (al menos) un fallo

Lo mismo pero distinto

- IEEE Standard Classification for Software Anomalies (1044™ – 2009)
  https://standards.ieee.org/findstds/standard/1044-2009.html
- Definiciones para: *defect, error, failure, fault, problem*
  - **Error (*error*)**: acción humana que produce un resultado incorrecto
  - **Problema (*problem*)**: dificultad o incertidumbre experimentada por una o más personas, como resultado de un encuentro insatisfactorio al utilizar un sistema
- También proporciona una colección de atributos para *defect* y *failure*

¿Conclusiones?

- Existe terminología, aunque no está extendida
- La importante diferencia entre causa (fallo-fault) y efecto (avería-failure)
- En el fondo, solo hay dos grandes categorías de errores
  - Errores de interpretación/traslación (de la especificación al diseño, del diseño al código)
  - Errores semánticos de programación

No existen errores informáticos, sino errores **de los** informáticos