Why Study Software Disasters?

Exam 2 out today, due at beginning of class Thursday.

It should not be a disaster!

---

Global Variables

// These macro define some default information of RTC
#define ORIGINYEAR 1980 // the begin year
#define MAXYEAR (ORIGINYEAR + 100) // the maxium year
#define JAN1WEEK 2 // Jan 1 1980 is a Tuesday


---

Function: IsLeapYear

// Local helper function checks if the year is a leap year

// Parameters:
//  Year: int
//  Returns: bool

static int IsLeapYear(int Year) {
    int Leap;
    if ((Year % 4) == 0) {
        Leap = 1;
        if ((Year % 100) == 0) {
            Leap = (Year % 400) ? 0 : 1;
        }
    }
    return (Leap);
}
We contacted a Microsoft spokesperson, who confirmed the issue with this official statement: "Early this morning we were alerted by our customers that there was a widespread issue affecting our 2006 model Zune 30GB devices (a large number of which are still actively being used). The technical team jumped on the problem immediately and isolated the issue: a bug in the internal clock driver related to the way the device handles a leap year. That being the case, the issue should be resolved over the next 24 hours as the time change moves to January 1, 2009. We expect the internal clock on the Zune 30GB devices will automatically reset tomorrow (noon, GMT). By tomorrow you should allow the battery to fully run out of power before the unit can restart successfully then simply ensure that your device is recharged, then turn it back on."

http://www.pcworld.com/article/156240/microsoft_says_leap_year_bug_caused_zune_failures.html

Questions about Bugs

Immediate
What is going wrong?
What is the bug?

Systemic
Is this bug a symptom of larger problems in the software design?
Why didn’t testing catch this?
Is this bug a symptom of larger problems in the development process, team, etc.?

In this case, the code came from Freescale, integrated into Microsoft Project (without going through MS development process)

Therac-25

Radiation Therapy Machine
Atomic Energy of Canada
1985-1987: gave six patients massive overdoses of radiation (3 died)

Nancy Levenson, Medical Devices: The Therac-25

Assumptions in AECL’s safety analysis:
1. Programming errors have been reduced by extensive testing on a hardware simulator and under field conditions on teletherapy units. Any residual software errors are not included in the analysis.
2. Program software does not degrade due to wear, fatigue, or reproduction process.
3. Computer execution errors are caused by faulty hardware components and by "soft" (random) errors induced by alpha particles and electromagnetic noise.

Nancy Levenson, Medical Devices: The Therac-25
Ariane 5 Movie

Ariane 5

- $500M rocket
devolved by European Space Agency
- June 4, 1996: first launch
  37s after ignition: lost guidance
  40s: exploded

Ariane 5 Inquiry Board Report (Jacques-Louis Lions):
http://esamultimedia.esa.int/docs/esa-x-1819eng.pdf

Flight Control System

Inertial Reference System (SRI)
- Calculates angles and velocities from on-rocket sensors (gyros, accelerometers)
- Data sent to On-Board Computer that executes flight program (controls booster nozzles, valves)

Redundancy in design to improve reliability
- Two separate computers running SRIs in parallel (same hardware and software) – one is “hot” stand-by used if OBC detects failure in “active” SRI

Design based on Ariane 4
- Software for SRI mostly reused from Ariane 4 implementation

Number Overflow Problems

- 16-bit signed integer
  - \(2^{16} = 65536\) different values
  - (-32768 – 32767)
- Alignment code converted the horizontal velocity (64-bit floating point value from sensors = up to \(\sim 10^{308}\)) to a 16-bit signed integer
- Overflow produces exception (Operand Error)

Defensive Programming

“The data conversion instructions were not protected from causing an Operand Error, although other conversions of comparable variables in the same place in the code were protected.”
It has been stated to the Board that not all the conversions were protected because a maximum workload target of 80% had been set for the SRI computer. To determine the vulnerability of unprotected code, an analysis was performed on every operation which could give rise to an exception, including an Operand Error. In particular, the conversion of floating point values to integers was analysed and operations involving seven variables were at risk of leading to an Operand Error. This led to protection being added to four of the variables, evidence of which appears in the Ada code. However, three of the variables were left unprotected. No reference to justification of this decision was found directly in the source code. Given the large amount of documentation associated with any industrial application, the assumption, although agreed, was essentially obscured, though not deliberately, from any external review.

The reason for the three remaining variables, including the one denoting horizontal bias, being unprotected was that further reasoning indicated that they were either physically limited or that there was a large margin of safety, a reasoning which in the case of the variable BH turned out to be faulty. It is important to note that the decision to protect certain variables but not others was taken jointly by project partners at several contractual levels.

Although the source of the Operand Error has been identified, this in itself did not cause the mission to fail. The specification of the exception-handling mechanism also contributed to the failure. In the event of any kind of exception, the system specification stated that the failure should be indicated on the databus, the failure context should be stored in an EEPROM memory (which was recovered and read out for Ariane 501), and finally, the SRI processor should be shut down.

It was the decision to cease the processor operation which finally proved fatal. Restart is not feasible since attitude is too difficult to re-calculate after a processor shutdown; therefore the Inertial Reference System becomes useless. The reason behind this drastic action lies in the culture within the Ariane programme of only addressing random hardware failures. From this point of view exception - or error - handling mechanisms are designed for a random hardware failure which can quite rationally be handled by a backup system.

**Java Version**

```java
public class Overflow {
    public static void main (String args[]) {
        int x;
        double d = 5000000000.0;
        x = (int) d;
        System.out.println("d = " + d + "/" + "x = "+x);
    }
}
```

```
d = 5.0E9 / x = 2147483647
What is 2147483647 + 1? -2147483648
```

**Ada Programming Language**

- Developed by a 1970s US DoD effort to create a safe, high-level, modular programming language
- 1987-1997: All DoD software projects required to use Ada
- Still fairly widely used in safety-critical software
  - Boeing 777
  - SPARK/Ada (subset with verification)

**Ada Package Declaration**

```ada
package Rational_Numbers is
type Rational is record
    Numerator : Integer;
    Denominator: Positive;
end record;
function "="(X,Y : Rational) return Boolean;
function "/"(X,Y : Integer) return Rational;
function "+"(X,Y : Rational) return Rational;
function "-"(X,Y : Rational) return Rational;
function "."(X,Y : Rational) return Rational;
function "+*"(X,Y : Rational) return Rational;
function "/"(X,Y : Rational) return Rational;
end Rational_Numbers;
```


Type safety and information hiding are valuable: Ada code has $1/10^{th}$ as many bugs as C code, and cost $1/2$ as much to develop.
Ada Exception Handling

begin
  ... --- raises exception
end

exception
  when Exception: action

If exception raised in block B
If there is a handler, jumps to its action; if not, exception propagates to call site (and up)

Inertial Reference System

- Exception in alignment code for number conversion
- No handler in procedure
- Propagated up to top level
- SRI response to exception is to shutdown and put error on databus

Why was the alignment code still running?

The error occurred in a part of the software that only performs alignment of the strap-down inertial platform. This software module computes meaningful results only before lift-off. As soon as the launcher lifts off, this function serves no purpose.

Why didn’t testing find this?

The original requirement accounting for the continued operation of the alignment software after lift-off was brought forward more than 10 years ago for the earlier models of Ariane, in order to cope with the rather unlikely event of a hold in the count-down e.g. between -9 seconds, when flight mode starts in the SRI of Ariane 4, and -5 seconds when certain events are initiated in the launcher which take several hours to reset. The period selected for this continued alignment operation, 50 seconds after the start of flight mode, was based on the time needed for the ground equipment to resume full control of the launcher in the event of a hold. This special feature made it possible with the earlier versions of Ariane, to restart the count-down without waiting for normal alignment, which takes 45 minutes or more, so that a short launch window could still be used. In fact, this feature was used once, in 1989 on Flight 33.

The same requirement does not apply to Ariane 5, which has a different preparation sequence and it was maintained for commonality reasons, presumably based on the view that, unless proven necessary, it was not wise to make changes in software which worked well on Ariane 4.
What was the real problem?

What are the lessons?

Recommendations

**CORRECTIVE MEASURES**

- Correction of the problem in the SRI that led to the accident
- Reexamination of all software embedded in equipment
- Improvement of the representativeness (vis-à-vis the launcher) of the qualification testing environment
- Introduction of overlaps and deliberate redundancy between successive tests:
  - at equipment level
  - at stage level
  - at system level
- Improvement and systematisation of the two-way flow of Information:
  - up from equipment to system: nominal and failure-mode behaviour
  - down from system to equipment: use of equipment items in flight
Bertrand Meyer’s Analysis

“Reuse without a contract is sheer folly! Without a precise specification attached to each reusable component -- precondition, postcondition, invariant -- no one can trust a supposedly reusable component.”


Ken Garlington’s Critique

• Design contracts unlikely to solve this problem:
  – Specification would need to correctly identify precondition
  – Code review would need to correctly notice unsatisfied precondition
  – Or, run-time handler would need to recover correctly

Charge

• Avoid a software disaster for your projects
  – Coordinate with your team closely: all your code should be working together now
  – Make sure simple things work before implementing “fancy features”
• Subscribe to RISKS to get a regular reminder of software disasters: http://catless.ncl.ac.uk/Risks

http://home.flash.net/~kennieg/ariane.html

Exam 2 is out now, due at beginning of class Tuesday (it should not be a software disaster either!)