Why the Science of Computing is Important

Roland Backhouse

University of Nottingham

The hallmark of a science is the avoidance of error.

Perth to Kuala Lumpur, August, 2005

AGE, September 2005

Global alert after jet out of control

Steve Creedy
Aviation writer

PILOTS on a Boeing 777 from Perth to Kuala Lumpur battled to gain control of the plane last month after an unknown computer error caused the aircraft to pitch violently and brought it close to stalling.

A flight attendant dropped a tray of drinks and another began praying as the Malaysian Airlines pilots fought to counter false information being fed into the aircraft's autopilot system and primary flight display. The glitch prompted plane manufacturer Boeing to issue a global notice to all 777 operators alerting them to the problem.

Flight MH124 was about an hour out of Perth when the aircraft began behaving erratically. The incorrect data from a

rately tested the aircraft's two autopilot systems. But he was forced to keep flying manually when the plane banked to the right and the nose pitched down during both tests.

The pilot reported no difficulties flying the plane but noted that the automatic throttles remained armed.

As the aircraft was positioned to approach Perth, however, the flight display again gave a low airspeed warning and the auto-throttle responded by increasing thrust. The aircraft's warning system also indicated a dangerous windshear but the crew continued the approach and landed safely.

Shaken passengers remained in Perth overnight and were offered alternative flights the next day.

Investigations are focusing on faulty

- Malaysian Airline pilots fought to counter false information

- Malaysian Airline pilots fought to counter false information
- Auto-throttles remained armed

- Malaysian Airline pilots fought to counter false information
- Auto-throttles remained armed
- [When landing] low airspeed warning . . . auto-throttle increased thrust

- Malaysian Airline pilots fought to counter false information
- Auto-throttles remained armed
- [When landing] low airspeed warning . . . auto-throttle increased thrust
- After landing the autobrakes were not able to be cancelled

The ADIRU OPS versions up to and including version -07 contained a latent software error in the algorithm to manage the sensor set used for computing flight control outputs which, after the unit went through a power cycle, did not recognise that accelerometer number-5 was unserviceable. The status of the failed unit was recorded in the on-board maintenance computer memory, but that memory was not checked by the ADIRU software during the start-up initialisation sequence. The software error had not been detected during the original certification of the ADIRU and was present in all versions of the software. The effect of the error was suppressed by other software functions in OPS version -03. When the OPS version - 04 was released in December 1998, the software functions that suppressed the error were further revised to improve shop repair capability, re-exposing the undiscovered latent problem.

Therac-25 Disaster

Therac-25 Disaster

- Radiation Therapy machine
- Coding error

Therac-25 Disaster

- Radiation Therapy machine
- Coding error
- At least 6 accidents and 3 deaths attributed (1985-1987)
- More suspected

Therac-25 Disaster Root Causes

Therac-25 Disaster Root Causes

- Overflow error in non-initialised flag variable

Therac-25 Disaster Root Causes

- Overflow error in non-initialised flag variable
- Two modes: low and high power
- (Physical) Beam spreader used for high-power mode

Therac-25 Disaster Root Causes

- Overflow error in non-initialised flag variable
- Two modes: low and high power
- (Physical) Beam spreader used for high-power mode
- Switching between modes
- Improbable sequence of keystrokes within 8 seconds

2010

THE DAILY TELEGRAPH | SATURDAY, AUGUST 28, 2010

News

We're about to crash... oops my mistake

By Daily Telegraph Reporter

BRITISH Airways passengers thought they were going to die after an emergency message warning them that their plane may have to ditch into the sea was played in error.

Travellers flying from Heathrow to Hong Kong were told: "This is an emergency. We may shortly need to make an emergency landing on water."

But cabin crew on the Boeing 747 reassured them that the message was a mistake.

Michelle Lord, 32, of Preston, Lancs, said: "People were terrified. We all thought we were going to die." Another traveller said: "I can't think of anything worse than being told your plane's about to crash."

BA said that the message was an automatic one that was triggered by a computer. A spokesman said: "We would like to apologise to passengers on board the flight for causing them undue distress."

Binary Search

Binary Search

"While the first binary search was published in 1946, the first binary search that works correctly for all values of n did not appear until 1962."

J.Bentley, Programming Pearls

Extract from a 2003 textbook on programming in Java

```
{  int hi = v.length ;
   int lo = 0 ;
   while (true)
   {
        int centre = (hi + lo) / 2 ;
        if (centre == lo)
        {  //
           //  Only two items left to test so it is either centre
           //  or centre+1 or it is not in.  This is an exit
           //  point of the infinite loop.
           //
           if (v[centre] == o)
              { return centre; }
           else if (v[centre+1] == o)
              { return centre+1; }
           else
              { return -1; }
         }
         if (v[centre] < o)
           { lo = centre ; }
         else if (o < v[centre])
           { hi = centre ; }
         else
           { return centre ; }
   }
}
```

$v.length = 2,\ v[0] = 10,\ v[1] = 20,\ o = 30$

```
{  int hi = 2 ;
   int lo = 0 ;
   while (true)
   {
        int centre = (hi + lo) / 2 ;
        if (centre == lo)
        {  //
           //  Only two items left to test so it is either centre
           //  or centre+1 or it is not in.  This is an exit
           //  point of the infinite loop.
           //
           if (<10,20>[centre] == 30)
              { return centre; }
           else if (<10,20>[centre+1] == 30)
              { return centre+1; }
           else
              { return -1; }
         }
         if (<10,20>[centre] < 30)
           { lo = centre ; }
         else if (30 < <10,20>[centre])
           { hi = centre ; }
         else
           { return centre ; }
   }
}
```

```
{  int hi = 2 ;
   int lo = 0 ;
   while (true)
   {
        int centre = (2 + 0) / 2 ;
        if (centre == 0)
        {  //
           //  Only two items left to test so it is either centre
           //  or centre+1 or it is not in.  This is an exit
           //  point of the infinite loop.
           //
           if (<10,20>[centre] == 30)
              { return centre; }
           else if (<10,20>[centre+1] == 30)
              { return centre+1; }
           else
              { return -1; }
         }
         if (<10,20>[centre] < 30)
           { lo = centre ; }
         else if (30 < <10,20>[centre])
           { hi = centre ; }
         else
           { return centre ; }
   }
}
```

"two items left" but test fails!

```
{  int hi = 2 ;
   int lo = 0 ;
   while (true)
   {
        int centre = (2 + 0) / 2 ;
        if (1 == 0)
        {  //
           //  Only two items left to test so it is either centre
           //  or centre+1 or it is not in.  This is an exit
           //  point of the infinite loop.
           //
           if (<10,20>[centre] == 30)
              { return centre; }
           else if (<10,20>[centre+1] == 30)
              { return centre+1; }
           else
              { return -1; }
         }
         if (<10,20>[centre] < 30)
           { lo = centre ; }
         else if (30 < <10,20>[centre])
           { hi = centre ; }
         else
           { return centre ; }
   }
}
```

```
{  int hi = 2 ;
   int lo = 0 ;
   while (true)
   {
       int centre = (2 + 0) / 2 ;
        if (1 == 0)
        {  //
           //  Only two items left to test so it is either centre
           //  or centre+1 or it is not in.  This is an exit
           //  point of the infinite loop.
           //
           if (<10,20>[centre] == 30)
              { return centre; }
           else if (<10,20>[centre+1] == 30)
              { return centre+1; }
           else
              { return -1; }
         }
         if (<10,20>[1] < 30)
           { lo = 1 ; }
         else if (30 < <10,20>[1])
           { hi = 1 ; }
         else
           { return 1 ; }
   }
}
```

2nd execution of loop body. $\texttt{hi} = 2$, $\texttt{lo} = 1$

```
{  int hi = 2 ;
   int lo = 0 ;
   while (true)
   {
       int centre = (2 + 1) / 2 ;
        if (centre == 1)
        {  //
           //  Only two items left to test so it is either centre
           //  or centre+1 or it is not in.  This is an exit
           //  point of the infinite loop.
           //
           if (<10,20>[centre] == 30)
              { return centre; }
           else if (<10,20>[centre+1] == 30)
              { return centre+1; }
           else
              { return -1; }
         }
         if (<10,20>[centre] < 30)
           { lo = centre ; }
         else if (30 < <10,20>[centre])
           { hi = centre ; }
         else
           { return centre ; }
   }
}
```

2nd test for termination. $\mathtt{hi} = 2$, $\mathtt{lo} = 1$, $\mathtt{centre} = 1$

```
{  int hi = 2 ;
   int lo = 0 ;
   while (true)
   {
        int centre = (2 + 1) / 2 ;
        if (1 == 1)
        {  //
           //  Only two items left to test so it is either centre
           //  or centre+1 or it is not in.  This is an exit
           //  point of the infinite loop.
           //
           if (<10,20>[1] == 30)
              { return 1; }
           else if (<10,20>[1+1] == 30)
              { return 1+1; }
           else
              { return -1; }
         }
         if (<10,20>[centre] < 30)
           { lo = centre ; }
         else if (30 < <10,20>[centre])
           { hi = centre ; }
         else
           { return centre ; }
   }
}
```

```
{  int hi = 2 ;
   int lo = 0 ;
   while (true)
   {
       int centre = (2 + 1) / 2 ;
       if (1 == 1)
        {  //
           //  Only two items left to test so it is either centre
           //  or centre+1 or it is not in.  This is an exit
           //  point of the infinite loop.
           //
           if (<10,20>[1] == 30)
              { return 1; }
           else if (<10,20>[1+1] == 30)
              { return 1+1; }
           else
              { return -1; }
        }
        if (<10,20>[centre] < 30)
          { lo = centre ; }
        else if (30 < <10,20>[centre])
          { hi = centre ; }
        else
          { return centre ; }
   }
}
```

```
{  int hi = 2 ;
   int lo = 0 ;
   while (true)
   {
       int centre = (2 + 1) / 2 ;
       if (1 == 1)
        {  //
           //  Only two items left to test so it is either centre
           //  or centre+1 or it is not in.  This is an exit
           //  point of the infinite loop.
           //
           if (<10,20>[1] == 30)
              { return 1; }
           else if (<10,20>[1+1] == 30)
              { return 1+1; }
           else
              { return -1; }
         }
         if (<10,20>[centre] < 30)
           { lo = centre ; }
         else if (30 < <10,20>[centre])
           { hi = centre ; }
         else
           { return centre ; }
   }
}
```

"or centre+1"

```
{  int hi = 2 ;
   int lo = 0 ;
   while (true)
   {
       int centre = (2 + 1) / 2 ;
       if (1 == 1)
        {  //
           //  Only two items left to test so it is either centre
           //  or centre+1 or it is not in.  This is an exit
           //  point of the infinite loop.
           //
           if (<10,20>[1] == 30)
              { return 1; }
           else if (<10,20>[1+1] == 30)
              { return 1+1; }
           else
              { return -1; }
         }
         if (<10,20>[centre] < 30)
           { lo = centre ; }
         else if (30 < <10,20>[centre])
           { hi = centre ; }
         else
           { return centre ; }
   }
}
```

Designing Algorithms to Meet Specifications

Designing Algorithms to Meet Specifications

Given an array V and array indices M and N such that

$$M < N \quad \wedge \quad V[M] \leqslant 0 < V[N]$$

design an algorithm that will return an index i such that

$$M \leqslant i < N \quad \wedge \quad V[i] \leqslant 0 < V[i+1]$$

Pre- and Post-conditions

$M < N \quad \wedge \quad V[M] \leqslant 0 < V[N]$

```
{  int  i;
```

$M \leqslant i < N \quad \wedge \quad V[i] \leqslant 0 < V[i+1]$

```
}
```

Invariant

$M < N \quad \wedge \quad V[M] \leqslant 0 < V[N]$

```
{  int  i;
   int  j;
```

Invariant: $M \leqslant i < j \leqslant N \quad \wedge \quad V[i] \leqslant 0 < V[j]$

$M \leqslant i < N \quad \wedge \quad V[i] \leqslant 0 < V[i+1]$

```
}
```

Termination Condition

$M < N \quad \wedge \quad V[M] \leqslant 0 < V[N]$

```
{  int  i;
   int  j;
```

Invariant: $M \leqslant i < j \leqslant N \quad \wedge \quad V[i] \leqslant 0 < V[j]$

```
   while  i + 1 ≠ j
          {
```

```
          }
```

$M \leqslant i < N \quad \wedge \quad V[i] \leqslant 0 < V[i+1]$

```
}
```

Establishing the Invariant

$M < N \quad \wedge \quad V[M] \leqslant 0 < V[N]$

```
{  int i;
   int j;
   i, j := M, N;
   Invariant:   M ⩽ i < j ⩽ N   ∧   V[i] ⩽ 0 < V[j]

   while  i + 1 ≠ j
          {




          }
```

$M \leqslant i < N \quad \wedge \quad V[i] \leqslant 0 < V[i+1]$

```
}
```

Measure of Progress

$M < N \quad \wedge \quad V[M] \leqslant 0 < V[N]$

```
{  int  i;
   int  j;
   i, j  :=  M, N;
```

Invariant: $M \leqslant i < j \leqslant N \quad \wedge \quad V[i] \leqslant 0 < V[j]$

Measure of progress: $j - i$

```
   while  i + 1 ≠ j
        {
```

```
        }
```

$M \leqslant i < N \quad \wedge \quad V[i] \leqslant 0 < V[i+1]$

```
}
```

Maintaining the Invariant whilst Making Progress. Step 1

$M < N \quad \wedge \quad V[M] \leqslant 0 < V[N]$

{ int i;
int j;
i, j := M, N;
Invariant: $M \leqslant i < j \leqslant N \quad \wedge \quad V[i] \leqslant 0 < V[j]$
Measure of progress: $j - i$
while $i + 1 \neq j$
{ $i < j \quad \wedge \quad i + 1 \neq j$
int $k := (i + j)/2$;
$i < k < j$

}

$M \leqslant i < N \quad \wedge \quad V[i] \leqslant 0 < V[i + 1]$
}

Maintaining the Invariant whilst Making Progress. Step 2

$M < N \quad \wedge \quad V[M] \leqslant 0 < V[N]$

```
{  int i;
   int j;
   i, j := M, N;
   Invariant:   M ⩽ i < j ⩽ N   ∧   V[i] ⩽ 0 < V[j]
   Measure of progress:   j − i
   while i + 1 ≠ j
        { i < j   ∧   i + 1 ≠ j
          int k := (i + j)/2;
          i < k < j
          if   V[k] ⩽ 0  →  i := k
          □   0 < V[k]  →  j := k
          fi
        }
M ⩽ i < N   ∧   V[i] ⩽ 0 < V[i + 1]
}
```

Conclusions

Conclusions

- Our livelihoods and sometimes our lives depend on the correct functioning of computer software

Conclusions

- Our livelihoods and sometimes our lives depend on the correct functioning of computer software
- The 90/10 rule is particularly important

Conclusions

- Our livelihoods and sometimes our lives depend on the correct functioning of computer software
- The 90/10 rule is particularly important
- The science of computing is not easy to learn

Conclusions

- Our livelihoods and sometimes our lives depend on the correct functioning of computer software
- The 90/10 rule is particularly important
- The science of computing is not easy to learn
- That's no excuse for not teaching it!