CSE 435
Nov 16, 2015
Testing
Orly Airport in Paris shutdown due to bug in system running on Windows 3.1

http://www.itpro.co.uk/security/25597/planes-grounded-at-paris-orly-airport-thanks-to-windows-31-error
8.1 Software Faults and Failures
Why Does Software Fail?

- Wrong requirement: not what the customer wants
- Missing requirement
- Requirement impossible to implement
- Faulty design
- Faulty code
- Improperly implemented design
Patriot Missile

- Built by Raytheon
- originally designed to detect Soviet Cruise missiles
- Used in Gulf War in 1991 to detect incoming Iraqi Scud missiles
- designed to be mobile and operate for only a few hours at a time (to avoid detection)
To detect and destroy incoming missiles:

- search for airborne objects on radar
- calculate where to look next for incoming object (i.e., track it as it approaches)
- when incoming object in range, launch Patriot’s missile to destroy it
Patriot Missile, cont.

- predict location of incoming target based on its velocity and time of last radar detection
- velocity stored as a whole number and a decimal
- time as an integer representing tenths of a second (grows larger as time goes on)
Patriot Missile, cont.

- algorithm to compute next location requires real numbers
- computer system has 24 bit registers
- no exact representation for 1/10
- precision inaccuracy grows as time value increases
- inaccuracy directly proportional to target object’s velocity
Software Reuse

assumptions made in design of the original system:

- Soviet cruise missiles and high altitude aircraft operate at speed around MACH 2
- system will operate only a few hours at a time
In Operation Desert Storm

- Patriot missiles used as static defense (i.e., operating continuously)
- Scud missiles velocity around MACH 5
- Israeli forces discovered the bug, reported that targeting inaccuracies were 20% after 8 continuous hours
- Army presumed systems were not running more than 8 hours at a time; recommended rebooting regularly (reset the clock to 0)
- devised and distributed a bug fix
Patriot Missile, cont.

- Feb 25, 1991, a Scud missile launched at an airfield in Saudi Arabia
- Patriot system had been operating 100 continuous hours; resulting inaccuracy about .34 seconds
- could not track the incoming Scud
- hit an army barracks, killing 28 US soldiers, wounding 98.
Objective of testing: discover faults

A test is successful only when a fault is discovered

- Fault identification is the process of determining what fault caused the failure
- Fault correction is the process of making changes to the system so that the faults are removed
8.1 Software Faults and Failures

Types of Faults

- Algorithmic fault
- Computation and precision fault
  - a formula’s implementation is wrong
- Documentation fault
  - Documentation doesn’t match what program does
- Capacity or boundary faults
  - System’s performance not acceptable when certain limits are reached
- Timing or coordination faults
- Performance faults
  - System does not perform at the speed prescribed
- Standard and procedure faults
Typical Algorithmic Faults

- An algorithmic fault occurs when a component’s algorithm or logic does not produce proper output
  - Branching too soon
  - Branching too late
  - Testing for the wrong condition
  - Forgetting to initialize variable or set loop invariants
  - Forgetting to test for a particular condition
  - Comparing variables of inappropriate data types
- Syntax faults
Mariner 1

- 1962: Intended as a Fly-by of Venus
- guidance system responded improperly to commands from the ground
- destroy command sent 6 seconds before separation
- FORTRAN
  - implicit variable declaration
  - ignore whitespace
DO 30 I=1,100
  ...adjust...
  ...route...
  ...to...
  ... Venus...
30 CONTINUE

DO 30 I=1,100
  ...adjust...
  ...route...
  ...to...
  ... Venus...
30 CONTINUE
8.1 Software Faults and Failures

Sidebar 8.1 Hewlett-Packard’s Fault Classification
8.1 Software Faults and Failures
Sidebar 8.1 Faults for one Hewlett-Packard Division

[Diagram showing different categories and their percentages]
- Logic: 32%
- Computation: 18%
- Other code: 11%
- Data handling: 6%
- Documentation: 19%
- Requirements: 5%
- Hardware: 4%
- Process/interprocess: 5%
8.2 Testing Issues

Testing Organization

- Module testing, component testing, or unit testing
- Integration testing
- Function testing
- Performance testing
- Acceptance testing
- Installation testing
8.2 Testing Issues
Testing Organization Illustrated

Diagram showing the testing organization with stages such as Unit test, Integration test, Function test, Performance test, Acceptance test, and Installation test.
8.2 Testing Issues
Attitude Toward Testing

• Remember, goal of testing is not to demonstrate that the program is correct
• Purpose is to FIND faults. Successful testing means finding flaws

• Egoless programming: programs are viewed as components of a larger system, not as the property of those who wrote them
• don’t be defensive, get hurt feelings
8.2 Testing Issues
Who Performs the Test?

- Independent test team
  - avoid conflict
  - improve objectivity
  - allow testing and coding concurrently
8.2 Testing Issues

Views of the Test Objects

• Closed box or black box: functionality of the test objects
• Clear box or white box: structure of the test objects
8.2 Testing Issues
Clear Box

- Example of logic structure
8.2 Testing Issues
Factors Affecting the Choice of Test Philosophy

• The number of possible logical paths
• The nature of the input data
• The amount of computation involved
• The complexity of algorithms

Q: What is the difference between a branch and a path?
In class activity: Take out a sheet of paper
8.3 Unit Testing

Code Review

- Code walkthrough
- Code inspection
Each developer submits a list of modules “completed” each week to the manager.

Manager forms groups of 3-4 people, assigns modules from those people to the group.

Individuals go through and review code assigned to them, “red line” it.
Code walkthroughs at Litton, cont.

- meetings on Fridays to go over all the red line versions
- code could be:
  - accepted as is
  - accepted with modification
  - rejected (to be rewritten)
- was specification code, not final target language code
### 8.3 Unit Testing
#### Typical Inspection Preparation and Meeting Times

<table>
<thead>
<tr>
<th>Development Artifact</th>
<th>Preparation Time</th>
<th>Meeting Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement Document</td>
<td>25 pages per hour</td>
<td>12 pages per hour</td>
</tr>
<tr>
<td>Functional specification</td>
<td>45 pages per hour</td>
<td>15 pages per hour</td>
</tr>
<tr>
<td>Logic specification</td>
<td>50 pages per hour</td>
<td>20 pages per hour</td>
</tr>
<tr>
<td>Source code</td>
<td>150 lines of code per hour</td>
<td>75 lines of code per hour</td>
</tr>
<tr>
<td>User documents</td>
<td>35 pages per hour</td>
<td>20 pages per hour</td>
</tr>
</tbody>
</table>
So, is it worth it?
### 8.3 Unit Testing

#### Fault Discovery Rate

<table>
<thead>
<tr>
<th>Discovery Activity</th>
<th>Fault Found per Thousand Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements review</td>
<td>2.5</td>
</tr>
<tr>
<td>Design Review</td>
<td>5.0</td>
</tr>
<tr>
<td>Code inspection</td>
<td>10.0</td>
</tr>
<tr>
<td>Integration test</td>
<td>3.0</td>
</tr>
<tr>
<td>Acceptance test</td>
<td>2.0</td>
</tr>
</tbody>
</table>