2ND INTERNATIONAL WORKSHOP ON SOFTWARE HEALTH MANAGEMENT

(SHM 2011)

SMC-IT 2011

Gabor Karsai
Software Failure: Malaysian Air (Boeing 777) in-flight upset (2005)

- Airplane’s autopilot experienced excessive acceleration values.
- Autopilot pitched nose-up to 17.6 degree and climbed at a vertical speed of 10,650 fpm.
- Airspeed reduced to 241 knots and aircraft descended 4,000 ft.
- Re-engagement of autopilot followed by another climb of 2,000 ft.

Contributing Factors: “An anomaly existed in the component software hierarchy that allowed inputs from a known faulty accelerometer to be processed by the air data inertial reference unit (ADIRU) and used by the primary flight computer, autopilot and other aircraft systems.”

Qantas 72

Qantas 72 - Oct 7, 2008 – A330 (Australia) – ATSB Report

At 1240:28, while the aircraft was cruising at 37,000 ft, the autopilot disconnected. From about the same time there were various aircraft system failure indications. At 1242:27, while the crew was evaluating the situation, the aircraft abruptly pitched nose-down. The aircraft reached a maximum pitch angle of about 8.4 degrees nose-down, and descended 650 ft during the event. After returning the aircraft to 37,000 ft, the crew commenced actions to deal with multiple failure messages. At 1245:08, the aircraft commenced a second uncommanded pitch-down event. The aircraft reached a maximum pitch angle of about 3.5 degrees nose-down, and descended about 400 ft during this second event. At 1249, the crew made a PAN urgency broadcast to air traffic control, and requested a clearance to divert to and track direct to Learmonth. At 1254, after receiving advice from the cabin of several serious injuries, the crew declared a MAYDAY. The aircraft subsequently landed at Learmonth at 1350.

The investigation to date has identified two significant safety factors related to the pitch-down movements. Firstly, immediately prior to the autopilot disconnect, one of the air data inertial reference units (ADIRUs) started providing erroneous data (spikes) on many parameters to other aircraft systems. The other two ADIRUs continued to function correctly. Secondly, some of the spikes in angle of attack data were not filtered by the flight control computers, and the computers subsequently commanded the pitch-down movements.

Explore software health management in the context of system level dependability cases

  http://www7.nationalacademies.org/CSTB/pub_dependable.html
- Explicit claims of system (and subsystem) requirements including assumptions about the application domain and environment in which the system is to operate
- Evidence that software satisfies these explicit claims under the stated domain assumptions
- Architectural principles, enforced by hardware mechanisms, that ensure that software behavior dependencies are traceable; and
- Mechanisms for correctly composing software systems from trusted components within the constraints imposed by the architectural principles
Software Health Management

Why ‘Software Health Management’ and why now?

• Complexity of systems necessitates an additional layer ‘above’ software fault tolerance that manages the ‘Software Health’
• Embedded software ....
  • is a crucial ingredient in aerospace systems
  • is a method for implementing functionality
  • is the ‘universal system integrator’
  • could exhibit faults that lead to system failures
  • complexity has progressed to the point that zero-defect systems (containing both hardware and software) are very difficult to build
• Systems Health Management is an emerging field that addresses precisely this problem: How to manage systems’ health in case of faults ?
• ‘Software Health Management’ is not …
  • A replacement for existing and robust engineering processes and standards (DO-178B)
  • A substitute for hardware- and software fault tolerance
  • An ‘ultimate’ solution for fault tolerance
Software Health Management: Key Points

- Software is an extremely complex engineering artifact
- Software can have latent faults due to this complexity
- Faults appear during operation when unforeseen modes or interactions arise
- Traditional techniques like voting and self-C-checking pairs have shortcomings
  - Common mode faults
  - Fault cascades

- SHM is the extension of FDIR techniques of physical systems to software systems
<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>10:30AM-10:45AM</td>
<td>Introductory Remarks</td>
</tr>
<tr>
<td>10:45 AM-11:30AM</td>
<td>Invited Talk: On the reliability of monitored systems</td>
</tr>
<tr>
<td></td>
<td>Authors: John Rushby and Natarajan Shankar</td>
</tr>
<tr>
<td>11:30 AM–12:15PM</td>
<td>Smart Health Management for Swarm-based Space Exploration Systems</td>
</tr>
<tr>
<td></td>
<td>Authors: Mike Hinchey and Emil Vassev</td>
</tr>
<tr>
<td>12:15 PM - 1:30PM</td>
<td>Lunch</td>
</tr>
<tr>
<td>1:30 PM - 2:15 PM</td>
<td>Software Health Management with Bayesian Networks</td>
</tr>
<tr>
<td></td>
<td>Authors: Ole Mengshoel and Johann Schumann</td>
</tr>
<tr>
<td>2:15 PM - 3:00 PM</td>
<td>Automated Failure-modes-and-effects Analysis of Embedded Software</td>
</tr>
<tr>
<td></td>
<td>Author: Peter Struss</td>
</tr>
<tr>
<td>3:00 PM – 3:30 PM</td>
<td>Break</td>
</tr>
<tr>
<td>3:30 PM – 3:45 PM</td>
<td>Monitoring Safety Critical Systems</td>
</tr>
<tr>
<td></td>
<td>Author: Alwyn Goodloe, NASA</td>
</tr>
<tr>
<td>3:45 PM – 5:00 PM</td>
<td>Discussion: Challenges and Solutions for Software Health Management</td>
</tr>
<tr>
<td>5:00 PM - 5:05 PM</td>
<td>Closing Remarks</td>
</tr>
</tbody>
</table>
DISCUSSION
Challenges and Solutions
The Road Ahead
Challenges to Software Health Management

- **Detection challenge**: How to detect software anomalies?

- **Isolation (diagnostics) challenge**: How to isolate the source of the anomaly? What is a ‘source’, in any case?

- **Mitigation challenge**: How to choose and execute a mitigation action that preserves safety and function?
Challenges to Software Health Management – Contributing factors

- Lack of analytical model
- Unlike faults in physical systems, software faults arise abruptly and exhibit unexpected behaviors
- Development of complex embedded software is still often an “art” – rather than science
- Several software metrics exist, but it is hard to quantify what good software is
- Typically, software health managers are also written by hand, increasing the chance that they too have bugs.
The Road Ahead

• Better solutions for the D/I/M challenges?
• Better systems engineering?
• Connections to hardware health management?
• Pragmatics: Challenge Problems?