Software criticality classification and reduction

D. Escorial Rico
Workshop of Critical Software (WOCS) - Tokyo
27/09/2012
Context

Software more and more used in the space domain in important functions

Classification of software components

Implications of criticality classification of Software

Software safety and dependability discipline: software related failures
More context – ESA and ECSS

Terms and Requirements from ECSS – European Cooperation for Space Standardisation

Evolution of applicable standards for ESA projects – European Space Agency

Study for software criticality classification and reduction
Software Criticality Analysis – Process

System level dependability and safety analysis
Software products specifications
Hardware-Software interaction analysis

Software Criticality Analysis

Classification of software components
Measures for handling of Critical software
Design recommendations for criticality reduction
Software Criticality Analysis – Process cont’d

1. Software components functional analysis
2. Software dependability analysis
3. Criticality reduction
Software Criticality Analysis –
Functional analysis

Software products specifications
Hardware-Software Interaction analysis

Software components functional analysis

Software components
Functions implemented by software components
Software components interactions
Software Criticality Analysis – Software dependability analysis

Software components
Functions implemented by software components
Software components interactions

Software dependability analysis:
Software Failure Modes, Effects and Criticality Analysis
Software Fault-Tree Analysis

Failure mitigation measures
Design recommendations
Classification of software components
Measures for handling critical software
Software Criticality Analysis – Criticality reduction

Failure mitigation measures
Design Recommendations
Classification of software components

Criticality Reduction

Design recommendations for criticality reduction
How Software is classified

Main ECSS criteria to classify software:
Depends on the consequences of a software caused failure

Software malfunction:
Software not executed when expected, or not correctly executed, or with anomalous behaviour or contributing to a system failure
<table>
<thead>
<tr>
<th>Category</th>
<th>Definition</th>
</tr>
</thead>
</table>
| A        | Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in:  
  - **Catastrophic consequences** |
| B        | Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in:  
  - **Critical consequences** |
| C        | Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in:  
  - **Major consequences** |
| D        | Software that if not executed, or if not correctly executed, or whose anomalous behaviour could cause or contribute to a system failure resulting in:  
  - **Minor or Negligible consequences** |
<table>
<thead>
<tr>
<th>Severity</th>
<th>Dependability Consequences</th>
<th>Safety consequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catastrophic</td>
<td>Failures propagation</td>
<td>Loss of life</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Severe detrimental environmental effects</td>
</tr>
<tr>
<td>Critical</td>
<td>Loss of mission</td>
<td>Temporarily disabling but not life-threatening injury</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Major detrimental environmental effects</td>
</tr>
<tr>
<td>Major</td>
<td>Major mission degradation</td>
<td></td>
</tr>
<tr>
<td>Minor or Negligible</td>
<td>Minor mission degradation or any other effect</td>
<td></td>
</tr>
</tbody>
</table>
Measures for handling critical software

Depending on the Software Criticality: Project applies more or less strict engineering and Product Assurance requirements – Tailoring

Critical software presents a risk for the mission, mitigated by more severe and costly requirements during whole software life cycle.
Different requirements for different categories of software – Examples

Test coverage targets

<table>
<thead>
<tr>
<th>Code coverage versus criticality category</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source code statement coverage</td>
<td>100%</td>
<td>100%</td>
<td>AM</td>
<td>AM</td>
</tr>
<tr>
<td>Source code decision coverage</td>
<td>100%</td>
<td>100%</td>
<td>AM</td>
<td>AM</td>
</tr>
<tr>
<td>Source code modified condition and decision coverage</td>
<td>100%</td>
<td>AM</td>
<td>AM</td>
<td>AM</td>
</tr>
</tbody>
</table>

AM: Agree & Measure

Independent Software Verification and Validation performed by a third party

Process Assessments and Improvement
The criticality of SW is assigned considering the overall system design:

Compensating provisions that can prevent or mitigate failure consequences (e.g. inhibits, monitors, operational procedures)

One or more independent back-ups exist for the function implemented by the SW

*Please note that the approach for SW criticality reduction presented is not part of the ECSS standards*
Conditions are established for acceptable compensating provisions in the SW criticality assignment, i.e.

- **Effectiveness** of compensating provisions must be demonstrated in all situations
- **No common cause of failure** between software and compensating provisions
- Sufficient **time to intervene** in all situations
- **Probabilistic assessment cannot be used** as a criterion for SW criticality reduction
The SW criticality category must be assigned based on the severity of the residual failure modes, i.e. the ones remaining after the compensating provisions become effective (including loss of function).

If monitors or inhibits are used as compensating provisions, any SW contained in them must be classified at the criticality corresponding to the most severe failure they compensate.
Verification of effectiveness and timeliness of compensating provisions (implemented in software, hardware, operations or a combination thereof) against potential software errors
Possibility that low critical software causes the failure of high critical software:

- **Prevention of failure propagation**
- **Identification of shared resources**
Research study on going

Propagation from Lower Criticality to Higher Criticality:

- Interactions of SW components
- Types of failures that might propagate
- Prevention mechanisms
# Prevention of Failure Propagation – Some examples

<table>
<thead>
<tr>
<th>Interaction</th>
<th>Failure type in low criticality</th>
<th>Impact in higher criticality</th>
<th>Prevention Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous</td>
<td>Aborted</td>
<td>Abortion of calling thread</td>
<td>Watchdog on return from call</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>Timing failure</td>
<td>Tolerated or causes deadline miss or timeout</td>
<td>Timeouts on return</td>
</tr>
<tr>
<td>Resource sharing</td>
<td>Resource violation</td>
<td>Arbitrary (including system crash)</td>
<td>Firewalling</td>
</tr>
</tbody>
</table>
**Supervisor**

To *monitor* the behaviour of the system

Checking a given set of properties (values, time constraints, ...)

The supervised computer performs safe (harmless), not necessarily correct actions
Firewall
To protect different parts of a software system by isolating them from each other

Partitioning
To isolate different software components
Memory partitioning
Processor time partitioning
Conclusions

Improvement of the Software Criticality Analysis process

Critically reduction measures framework

Strict conditions to reduce criticality

Study on prevention of failure propagation

* Please note that the approach for SW criticality reduction presented is not part of the ECSS standards

D. Escorial Rico | Workshop of Critical Software (WOCs) - Tokyo | 27/09/2012 | D/TEC | Slide 23
ECSS references

ECSS-Q-ST-80C – Space product assurance – Software product assurance
ECSS-Q-HB-80-03A – Space product assurance – Software dependability and safety
ECSS-E-ST-40C – Space engineering – Software Engineering
ECSS-Q-ST-30C – Space product assurance - Dependability
ECSS-Q-ST-40C – Space product assurance – Safety
ECSS-Q-ST-30-02C – Space product assurance – Failure modes, effects and criticality analysis (FMECA/FMEA)
ECSS-Q-ST-30-09C – Space product assurance – Availability analysis
ECSS-Q-ST-40-21C – Space product assurance – Fault tree analysis – Adoption notice ECSS/IEC 61025
Dates and Venue: 12-13 June 2013 at ESA/ESTEC, The Netherlands

"Intensifying dialogue on Software Product Assurance matters and obtain feedback from Industry"

Topics:

- Software Dependability and Safety
- Software Product Service History and Reuse
- Software Product Evaluation for Conformity
- Software Engineering for evolving technologies
- Research and Development activities in the field of SW Product Assurance