Astra 2015 RCOS forum

Developing software for space: RAMS and introduction to TASTE framework

Jean-Loup Terraillon
Software systems engineering section head
The space context

- **ESA missions “must” be successful**
- Difficulty to test the software in its real **operational** environment (rockets, spacecrafts)
  - Low risk profile of engineering
  - Software engineering and PA standards
  - Software must be in full control
- **Minimize the risk** to activate a software design error (no software failure)
  - All behaviour tested, full coverage of code and of requirements
  - No dead code
  - No dynamic memory allocation (C++ malloc)
  - Real-time behaviour deterministic/predictable
- Secure all possible hazards (redundancies, safe mode “that must works”)
The RAMS requirements

**RAMS?**  
(RAM = Dependability)  
- Reliability (when the software works, it works well) (ECSS-Q-ST-30)  
- Availability (the software always replies to request, even if it is failing)  
- Maintainability (the software can be modified at reasonable effort)  
- Safety (a software error is not going to kill someone) (ECSS-Q-ST-40)

**Criticality**: If a software error can cause  
- the death of someone, the software is Category A  
- the loss of mission, the software is Category B  
- the loss of part of the system/mission, the software is Category C  
- nothing really important, the software is Category D

➡ **Category of a Rover?**
The RAMS requirements generate:

- The functional requirements of the **Fault Detection Isolation Recovery (FDIR)**

  ![Diagram showing problem, FDIR, feared event, ops/safe mode.]

- **Engineering and PA requirements**
  - good process and good product
  - Cat B = Cat A – code coverage at object level
  - Cat C = Cat B – ISVV, some code coverage, some verification, some models

FMECA, FTA
Good process and good product?

**Coverage** (by the test campaign):
- of the requirements (functional and non-functional coverage)
- of the source code (structural coverage)
- of the real-time behaviour

This is long and costly, therefore:

- replace testing by verification of design (schedulability analysis, *correct by construction*)
- *separating the concerns* in the architecture (real-time, communication, hardware access, etc) such as the test of each concern is easier.
- full visibility and understanding of the code (no dead code, no `malloc`)

- How to test the autonomy? (⇒ formal methods)
Modern software development anyhow...

- Software schedule is shorter and shorter
  - **FASTER** production
  - **LATER** modification
  - **SOFTER** industrial organisation
- ➔ automation ➔ model based
- ➔ reference architecture ➔ component based approach
- ➔ **software factory** (don’t reuse software, reuse a solution)

- However, the middleware must follow the space constraints
- Cannot embed Ubuntu and spawn tasks
- Need “High Integrity” middleware & operating system
The example of TASTE: an automatic integrator of components

Data Model Technology:
- Data type interface
- Hardware architecture
- Software architecture
- Real-time architecture
- Glue code
- Appli skeleton
- User code

Taste_Editor
- Informatics
- Robotics

buildsupport
- PolyOrb-HI generic
- PolyOrb-HI configured
- Ocarina

orchestrator
- Binary

Data type interface
- Hardware architecture
- Software architecture
- Real-time architecture
- Glue code
- Appli skeleton
- User code

PolyOrb-HI generic
- QGenC
- RTW-EC
- Matlab, Scade
- Opengeode
- C, Ada

Glue code
- C, Ada, SDL, Simulink

Tool product
- Generated
The example of TASTE: an automatic integrator of components

Data Model Technology:
- Data type interface
- Real-time architecture
- Glue code

Separation of concerns:
- Hardware architecture
- Software architecture

Buildsupport:
- Ocarna
- PolyOrb-HI generic
- PolyOrb-HI configured

Space products:
- Appli skeleton C, Ada, SDL, Simulink
- Matlab, Scade
- QGenC, RTW-EC
- User code

Correct by construction:
- Binary

Correct by construction:
- Rtems
- Xenomai
- Linux/Win

Tool products:
- Generated
Conclusion

- **Dependability and Safety** are important in Space

- **FDIR** and **Software Standards** protect the spacecraft

- Software code must be **characterized**, known, covered, validated

- **Software factories** help to master the complexity and embed **pre-qualified** building blocks.