On Board Software

November 2014
OBSW Characteristics

- On Board Software Properties
- On Board Software Environment
- On Board Software Dependability
- On Board Software Criticality
● System-software co-engineering: Software is part of the System. Software properties must be derived from System properties.

● End-to-end system response time will result into a → software schedulability property.

● System availability property will result into → software FDIR mechanisms that must have a particular behaviour.

● System performance property may result in a → software numerical accuracy property.
OBSW: Constraints

- **Embedded** Software
  - Cross Development Environment, bounded Memory Footprint and Processor Consumption

- **On Board** Software
  - Single Event Effects, (Upset, Latch Up), Memory Scrubbing

- **Real Time** Software
  - Processor load, scheduling issues, soft and hard deadlines

- **Deterministic** Software
  - No Dynamic thread creation or memory allocation, budget and schedulability

- **Remote** Software
  - Need for autonomy

- **Critical** Software (see further down)
  - Possible catastrophic consequences of failure

- **Dependable** Software (see further down)
  - Need for high Reliability, Availability, Maintainability and Safety
Achieving mission objectives and ultimate mission success relies on dependability of the space Systems.

### Analysis

- **FMECA:** Failure Mode Effects and Criticality Analysis
- **FTA:** Fault Tree Analysis
- **FHA:** Functional Hazard Analysis
- **HSIA:** Hardware Software Interaction Analysis
- **SCCFA:** Software Common Causes Failure Analysis
- **SWCA:** Software Criticality Analysis

### RAMS

- **Reliability:** Continuity of correct service.
- **Availability:** Readiness for usage.
- **Maintainability:** Easiness of repair/upgrade.
- **Safety:** Non-occurrence of catastrophic failure

### FDIR

- **Fault Detection**
  e.g. through monitoring
- **Fault Isolation**
  Determination of the cause
- **Fault Recovery**
  E.g. through redundancy

### FDIR

- **Certifiability**
  Ability to get stamp from certification body

### Troubles

- **Error:** A wrong or missing human action or thought
- **Fault:** An incorrect step, process or data definition in a program
- **Failure:** The inability of the software to perform its required functions

### Strategies

- **Fault Prevention**
  Avoidance and reduction of fault causes
- **Fault Tolerance**
  Avoidance and reduction of fault consequences
- **Fault Removal**
  Removal of fault occurrences
- **Fault Forecasting**
  Prediction of behaviour in presence of faults

As software plays more and more a prominent role in space systems, its contribution to the overall system dependability becomes a vital aspect of system development.
Software that if not executed or if not correctly executed or whose anomalous behavior could cause or contribute to a system failure resulting in:

A: Catastrophic Consequences
- Loss of Life, life threatening, personnel injuries, permanently disabling injury or occupational illness,
- Loss of an element of an interfacing manned flight system. Damage to other equipment. Loss of launch site facility facilities or loss of system. Severe detrimental environmental effects.

B: Critical Consequences
- Permanent or non-recoverable loss of the satellite’s capability to perform its planned mission
- Temporarily disabling but not life-threatening injury or occupational illness.
- Major damage to flight system or loss or major damage to ground facilities.
- Major damage to public or private property or major detrimental environmental effects.

C: Major Consequences
- Negligible or minor effect on the satellite’s mission and operability
- A detailed definition is left on a project by project basis and reported in its risk policy. Example is Mission Simulation Software

D: Minor Consequences
- A detailed definition is left on a project by project basis and reported in its risk policy. Example is Test Software

Note: In the DO-178-C (Software Considerations in Airborne Systems), the Design Assurance Level (DAL) is determined from the safety assessment process and hazard analysis by examining the effects of a failure condition in the system. The failure conditions are categorized by their effects on the aircraft, crew, and passengers (catastrophic, severe/hazardous, major, minor, no effect).
OBSW : Process

- On Board Software Phases
- On Board Software Development Lifecycle
- On Board Software Development Standard
- On Board Software Documentation
OBSW: Phases

**On Board Software Overview for ULg**

**Project Phases**
- Pre-A: Concept of Operations
- A: Concept & Technology Development
- B: Preliminary Design & Technology Completion
- C: Final Design & Fabrication
- D: System Assembly, Test, & Launch
- E: Operations & Sustainment
- F: Closeout

**Key Decision Points**
- A: Mission Concept Review
- B: Systems Requirements Review
- C: Mission/System Definition Review
- D: Preliminary Design Review
- E: Critical Design Review
- F: Systems Integration Review
- G: Operational Readiness Review
- H: Flight Readiness Review
- I: Post Launch Assessment Review
- J: Decommissioning Review

**Major Reviews**
- A: Mission Concept Review
- B: Systems Requirements Review
- C: Mission/System Definition Review
- D: Preliminary Design Review
- E: Critical Design Review
- F: Systems Integration Review
- G: Operational Readiness Review
- H: Flight Readiness Review
- I: Post Launch Assessment Review
- J: Decommissioning Review

**Major Project Reviews Precede Each Key Decision Point**
OBSW: Verification & Validation

Verification

- All along Development
- Methods
  - Reviews
    - Peer Review
    - Cross Reading
    - Independent Software Verification and Validation
  - Analysis
    - Static Analysis
    - Schedulability Analysis

- Tools
  - Software Engineering Tools (e.g. static code analyzer)

Validation

- At end of Development
- Methods
  - Test Campaign
    - Unit Testing (Code Coverage)
    - Integration Testing (Interface)
    - Validation Testing (Functionality)
      - Requirement baseline
      - Software Specification

- Tools
  - Software Validation Facilities
    - Hardware Models
    - Numerical Test Benches
    - Hybrid Facilities

Supported as far as possible by Formal Methods
OBSW: Standards

European Cooperation for Space Standardization

Consultative Committee for Space Data Systems

Standards:
- Management Standards → ECSS-M-XXX
- Product Assurance Standards → ECSS-Q-XXX
- Engineering Standards → ECSS-E-XXX

Reports:
- Blue: Recommended Standards
- Magenta: Recommended Practices
- Green: Informational Reports
- Orange: Experimental or ongoing research
- Yellow: Record, but not Historical
- Silver: Historical
OBSW Architecture

- **Interfaces and Data Flows**
- **Functional Architecture**
- **Static Architecture**
- **Dynamic Architecture**
- **Deployment Architecture**
Interfaces & Data Flows

On Board Software > Architecture > Interfaces

Telecommands
- High Priority commands → Hardware
- Nominal commands → Software
- Macro commands → Expanding
- Time tagged commands → Scheduling

Telemetries
- Housekeeping (Platform & Payload): Temperature, Pressure, Voltage and Current, Statuses
- Science Data (Payload): e.g. Raw or compressed images
Functional Architecture (1/2)

- Functional Architecture

definition of

- functional break down in
  - functional modules and
  - functional interfaces between these modules
On Board Software Overview for ULg

November 2014

On Board Software > Architecture > Functional Architecture > Variability

Functional Architecture (2/3)
### Functional Architecture (3/3)

On Board Software > Architecture > Functional Architecture > Functional Breakdown

Note: interactions between functions are not depicted

<table>
<thead>
<tr>
<th>TM/TC Telemetry &amp; Telecommand</th>
<th>HK Housekeeping</th>
<th>MON Monitoring</th>
<th>FDIR Fault Detection Isolation and Recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>space/groun communications or</td>
<td>Gathering, filtering and reporting of</td>
<td>detection of on board events based on</td>
<td>for the on board software and system dependability</td>
</tr>
<tr>
<td>communications between spacecraft</td>
<td>on board acquired data</td>
<td>ranges or thresholds or trends</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MMGT Mission Management</th>
<th>THERM Thermal Management</th>
<th>PWR Electrical Power Supply Management</th>
<th>GNC/AOCS/ACNS Attitude and Orbit Control Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>for the execution of the mission timeline</td>
<td>for the temperature control of the spacecraft e.g. through thermal heater lines,</td>
<td>distributes the power coming from the battery and the solar arrays and manages battery charge / discharge,</td>
<td>Computes the actuators commands from the sensors measurement in order to control the spacecraft attitude and position. Control the propulsion system.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OBT On-Board Time Management</th>
<th>SADSM Solar Array Drive Mechanisms Management</th>
<th>ACQ Data Acquisition</th>
<th>RDP/HRM Release, Deployment and Pyrotechnic Activation Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>For the synchronization of the on-board clock with ground time.</td>
<td>for the optimal alignment of a satellite’s solar panels towards the Sun.</td>
<td>Acquisition of on board data e.g. according to polling sequence table and depending on mode</td>
<td>for solar arrays/antenna deployment, for propulsion arming sequence before firing, or for specific needs like shield jettison or spacecrafts composites dispatching</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SM Storage Management</th>
<th>EQPT Equipment Management</th>
<th>RF Radio Frequency Management</th>
<th>PL Payload Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>for the storage of TM in case of Earth link unavailability</td>
<td>for the maintain of the equipment table as the reflect of the actual equipment status, and the management of equipment interface,</td>
<td>for the communications with ground or with other spacecrafts e.g. through S-Band or X-Band links.</td>
<td>for scientific payload or additional units (very mission specific)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CM Context Management</th>
<th>SADSM Solar Array Drive Mechanisms Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saves the context (failure history buffer, GNC states, OBSW patches, equipment table, mission timeline) in case of processor failure.</td>
<td>for the optimal alignment of a satellite’s solar panels towards the Sun.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ETC ... And Many Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>And many other functions depending on the platform bus and on the mission</td>
</tr>
</tbody>
</table>
Static Architecture (1/4)

- Static architecture deals with
  - Functional Properties
  - Static Decomposition
  - Software Component Model
- Software is broken down
  - in software components (aka modules, units or objects)
  - with clear interface between them
- Components are organized (see slide 2/4)
  - horizontally in variability layers with coherent abstraction level and
  - vertically in functional chains with coherent functionality
- Component are functional units (see slides 2-3-4/4)
  - That encapsulate functional services,
  - that expose these services to other components through well defined interfaces, and
  - that can be assembled together to build a software product
- Components are made up of (see slide 3/4)
  - Algorithms, State Machines
  - Data Structures
- Main design decision is (see slide 4/4)
  - Central Data Pool vs Distributed Data Flows
- Main programming paradigms are
  - Procedural programming vs Object Oriented programming (not used so far in on board software)
Static Architecture (2/4)

On Board Software > Architecture > Static Architecture > Generic Structure

- Platform Dependent
- Avionics Dependent
- Operation Dependent
- Mission Dependent

Coherent Abstraction Level

Coherent Functionality

INTERFACES

MODULES

FUNCTIONAL CHAINS

VARIABILITY LAYERS

- Thermal Mgr
- Power Mgr
- FDIR
- AOCS
- SC Mgr
- ...

Thermal Mgr
Power Mgr
FDIR
AOCS
SC Mgr
...
Static Architecture (3/4)

COMPONENT
  MODULE
  UNIT
  OBJECT

Provided Interface

Data Structures

State Machines

Algorithms

Functions
Procedures
Methods

Required Interface
Static Architecture (4/4)

On Board Software > Architecture > Static Architecture > Design Decision

Central Data Pool

Distributed Data Flow

VS
Component Model

STATIC ARCHITECTURE

Platform Independent Model

Design Level

Functional Concerns (algo)

Required Interface

Pairing up

Component

Connector

Component

Platform Specific Model

Implementation Level

Non Functional Concerns (tasking, timing, config, init, security, ...)

Provided Interface

Composition

Reusability

Replaceability

Execution Platform
Dynamic Architecture (1/3)

- Dynamic Architecture deals with
  - Non Functional Properties
  - Computational Model
  - Dynamic Behaviour

- Main concepts are (see slide 2/3)
  - Tasks, Threads and Interrupts
  - Scheduling, Processing Time and Priorities
  - Communication and Synchronisation

- Main design decision is (see slide 3/3)
  - Periodic/Cyclic/Synchronous/Time Driven/Polling vs
  - Aperiodic/Sporadic/Asynchronous/Event Driven/Interrupts
Dynamic Architecture: Concepts

Shared Resources (memory, inputs/outputs, ...)

Shared Executable (reentrance of code ...)

Execution Flow

Scheduling
  Periodicity
  Priorities, Preemption

Tasking
  Periodic Tasks

Tasking
  Sporadic Tasks

Communications
  Pipes, sockets
  Message queues, mailboxes, ...

Synchronisation
  Semaphores
  Mutexes
  Critical Sections

Interrupts
  Hardware Interrupts
  Software Traps

Several interrupts and periodic and sporadic tasks with different periods and priorities leading to possibly complex schedulability issues

November 2014

On Board Software Overview for ULg
Dynamic Architecture: Computational Model

Computational Models

- **RMA**: Rate Monotonic Algorithm
  Static priority preemptive scheduling. Applies to cyclic jobs. Shorter cycle get higher priority

- **DMA**: Deadline Monotonic Algorithm
  Static priority preemptive scheduling. Applies also to sporadic jobs. Shorter deadline get higher priority

- **EDF**: Earliest Deadline First Algorithm
  Dynamic priority preemptive scheduling. Process closest to its deadline get highest priority

- **RCM**: Ravenscar Computational Model
  Fixed-priority preemptive system with tight restrictions on tasking and synchronisation such as priority ceiling that optimally bound priority inversion, to achieve lock-free mutual exclusion and to avoid deadlocks. Warrants static analysability of the source code and predictability of execution

- **TSP**: Time and Space Partitioning
  Hierarchical superposition of two computational models:
  Round robin scheduling of partitions and
  Fixed priority scheduling of tasks within partitions

See also Schedulability Analysis
Dynamic Architecture: Schedulability

- Proof of software schedulability
  - Run Time Testing (how to prove exhaustivity?)
  - Static Analysis (superior to testing)
    - Selected Computational Model (see dynamic architecture)
      - Implementation is assumed to comply to model
      - Mathematical schedulability criteria
    - Software Budget Report (estimated or measured)
      - Processor utilization
      - Worst Case Execution Times
Dynamic Architecture: Schedulability

Compliance to a selected Computational Model allows for Static Analysis and Formal Check of Schedulability Conditions, based on corresponding Mathematical Model fed by actual Measurement from execution of realistic scenarios.

To this respect, Static analysis is superior to testing, which faces exhaustivity issue in real-scale systems.
Deployment Architecture (1/2)

- Deployment Architecture deals with the Mapping
  - of a logical architecture (software components)
  - to a physical environment (hardware resources).

- Main concepts are
  - Distribution
  - Communication

- Main design decision is
  - Centralised Architecture vs
  - Distributed Architecture

Consider also
- Multi Processor
- Multi Core
- Time and Space Partitioning
A deployment architecture depicts the mapping of a logical architecture (software components) to a physical environment (hardware resources).

The physical environment includes the computing nodes, processors, memory, storage devices, and other hardware and communication devices.
Reference Architecture

Design Pattern
(« a repeatable solution ... »)

Middleware
(« a connectivity service ... »)

Common Services
(« shared services ... »)

Lighweight
On Board
Application Framework

Applications

AOCS  POWER  THERMAL  ...  PAYLOADS

On Board File System

Core Services

PUS services

DHS Support Package

BSW & RTOS I/F

Coms I/F

Generic Application Services

Data Acquisition

Mission Specific Applications

Generic Component

Platform Specific Components

Layered and Modular Architecture

Mission Specific Applications

Generic Component

Platform Specific Components

Layered and Modular Architecture
Building Blocks

- PUS (being revisited)
- SOIS (is this really new?)
- OBCP (new standard)
- FDIR (on going study)
- CFDP
- FS
- ...

On Board Software > Architecture > Reference Architecture > Building Blocks
OBSW Environment

- Execution Environment
  - On Board Processors
  - Real Time Operating Systems
- Development Environment
  - Modeling Environment
  - Programming Language
  - Production Environment
  - Validation Environment

On Board Software Overview for ULg
OBSW Environment : Processors

- **Hardened Space Qualified processors**
  - 1990: 1750 – 16 Bits – 2 MIPS
  - 2010: LEON – SPARC V8 - 32 Bits – Cache – Pipeline -100 Mhz – 84 MIPS
  - Future: NGMP – Quad Core LEON

- **Commercial of the Shelf**
  - e.g. Power PC 1600 MIPS but sensitive to SEEs

See Avionics Overview
OBSW: Operating System

- **RTOS**: Real Time Operating Systems
  - VxWorks (Commercial)
  - RTEMS (Open Source)
  - Linux RT (Not widely used ... in OBSW)

- **TSP**: Time and Space Partitionning
  - Hypervisor
  - \( \mu \)Kernel
OBSW Environment: Modelling

- Paradigm Shift
  - From Programming to Modelling
  - Model Based Software Engineering

- Modeling Domains
  - Architecture Modeling,
  - Data Modeling,
  - Behaviour Modeling

- Modelling Languages
  - UML, SYSML, AADL, AAML, SDL, ...

- Model Verification
  - Strong Syntax, Formal Verification

- Automatic Generation

ECLIPSE Integrated Development Environment (IDE) and Eclipse Modelling Framework (EMF)
Languages

- **C**: procedural language, widely used, poor expressivness but good control, well suited to system programming, efficient code but error prone

- **Ada**: strong typing, well suited to embedded and real-time programming, mainly used in launchers

- **C++**: object oriented, based on C, less efficient due to object orientation, not used or poorly used on board so far

- **Java**: object oriented, interpreted language, under investigation, possibly for OBCP

- **Assembler**: low level language, for specific usage
OBSW Environment: Production

Centralised Database

- Contains ...
  On-board software configuration, Constant values, and Parameters useful for ground
  tasks, message queues, observable parameters, events, actions associated with events, default housekeeping, default monitoring, patch areas, Drivers commands, TC function ID’s (PUS 8, 1), Memory partitions, File partitions, On-board stores, Default on-board storage allocation, PUS services and sub-services, Types of the parameters in the PUS TC/TM, Error codes, Application ID ...

- Ensure coherency between ... the different outputs

Production Tools

- Support Automatic generation of ... Configuration Files and Documentation

- Allow for easy configuration
OBSW Environment: Validation

On Board Software > Environment > Validation

Simulation Controllers
- Graphical User Interface
- Script Language Interpreter
- Integrated Symbolic Debugger
- Distributed Simulation Interface

Multi Controllers Support

Drivers
- TTM
- SIM
- DAM

Applications
- AOCS
- FDIR
- SYS
- MGT

Data Handling Software
- DHS
- PUS
- DHS
- CORE

Simulation Core
- Instruction Set Simulator Engine
- Processor Registers Simulation
- Memory Simulation
- Marker/Breakpoints/Coverage
- Discrete Event & Time Simulator Engine

Communications
- TTM
- SIM
- DAM

Equipments
- ST
- GPS
- MM
- MT
- RW

Instrum
- REC
- MPM

Test Conducting Script

Ground
- RWM

Environment
- OBSW

Generic SVF Component
- Platform dependent SVF Component

Copyright © 2014 by SPACEBEL - All rights reserved
OBSW Modeling

- Modeling Exemples
- Modeling Definition
- Modeling Objectives
- Modeling Characteristics
- Modeling Methods
Software has the rare property that it allows us to directly evolve models into fully-fledged implementations without changing the engineering medium, tools, or methods.

The software model may evolve into the system it was modeling in a seamless process.
What is a Model?

- A model is a description of (part of) a system written in a well defined language. A well defined language is a language with a well defined form (syntax), and meaning (semantic), which is suitable for an automatic interpretation by a computer. (Anneke Kleppe et al. « MDA Explained »)

- A formal representation of a function, behaviour and structure of the system we are considering. Expressed in an unambiguous language. (Chris. Raistrick et. Al. « Model Driven Architecture and Executable UML »)

- Models are an abstraction of the reality captured in a specific representation format i.e. diagram or language.

- A model is a simplification of something so we can view, manipulate, and reason about it, and so help us understand the complexity inherent in the subject under study. (Steve Mellor et. All « UML Distilled »)

- A simplification of a system built with an intended goal in mind: the model should be able to answer questions in place of the actual system. (J. Bézivin & O. Gerbé. « Towards a precise définition of the OMG/MDA Framework »)

- A model is a complex structure that represents a design artifact such as a relational schema, and interface definition (API), and XML schema, a semantic network, a UML model or an hypermedia document (Phil. Bernstein. « A Vision of Management of Complex Systems »)

- A model captures a view of a physical system. It is an abstraction of a physical system with a certain purpose; This purpose determines what is included in the model and what is relevant. Thus the model completely describes those aspects of the physical system that are relevant to the purpose of the model, at the appropriate level of detail. (OMG. « UML Superstructure »)

- A functional specification of the function, structure and/or behaviour of an application or system. (OMG. « MDA Guide »)
Objectives of Software Modeling

- To deal with complexity of systems development through
  - **Abstraction:**
    Abstract a problem to focus on some particular points of interest and to improve understandability of a problem
  - **Iteration:**
    Iterative modeling may be expressed at different level of fidelity
  - **Separation of Concerns:**
    Possible set of nearly independent views of a model ("Aspect Oriented Modeling")
  - **Domain Specific Language:**
    To focus on specific domain expertise

- To minimize development risks through
  - Through **analysis and experimentation** performed earlier in the design cycle
  - Enable to investigate and compare **alternative solutions**

- To improve communication ...
  - to foster **information sharing** and reuse!
    A model is often best suited than a long speech!
Characteristics of Useful Models:

- **Abstract**
  - Emphasize important aspects while removing irrelevant ones
- **Understandable**
  - Expressed in a form that is readily understood by observers
- **Accurate**
  - Faithfully represents the modeled system
- **Predictive**
  - Can be used to answer questions about the modeled system
- **Inexpensive**
  - Much cheaper to construct and study than the modeled system

To be useful, engineering models must satisfy all of these characteristics!
In an attempt to formalize more and more the expression of the software documentation and production, the design, architecture and requirements have moved from simple text to drawings and from drawings to models.

With the emergence of new tools these model representations can be constructed, translated and exploited in different ways:

- **Analysis**: various types of model checking e.g. completeness and consistency analysis
- **Simulation**: execution of model, behavior can be simulated
- **Design**: decomposition of system in smaller components, establishing interfaces
- **Coding**: automatic generation of source code e.g. C or Ada (auto-coding)
- **Testing**: automatic generation of tests (auto-testing)
- **Proving**: formal verification or proof
Modeling: Methods

- **Modeling Methods**
  - **UML** (Unified Modelling Language)
    - Not a method (weak semantic) but a notation (well defined syntax)
    - Can be extended by profiles (e.g. SysML, MARTE, ...)
    - Can be complemented/supplemented (e.g. OCL)
  - **AADL** (Architecture Analysis and Design Language)
  - **SDL** (Specification and Description Language)
  - .... and many others (e.g. AAML ...)
  - See also Matlab/Simulink/Stateflow
Use Case Diagram

On Board Software > Modeling > UML > Use Case Diagram

CFDP manager

PUS services
configure, control and monitor CFDP transactions
perform high-level operations on the distributed CFDP file-system
transfer PUS messages to a remote SC

Communication system
send and receive PDUs to and from remote CFDP entities

onboard application
local CFDP entity
remote CFDP entity
remote file system
local file system
inter spacecraft link
space link
try to associate the received PDU with one of the existing transactions

In this example, there is still enough memory resources for cfdp_transaction and cfdp_receiver but not for the bit array.
OBSW: On Board Procedures

**OBCPs: On Board Control Procedures**

- **OBOPs:**
  - On Board Operation Procedures
  - Quite Simple
  - Can be considered as Ground Procedures uploaded on board
  - Limited interactions with FSW, through TC and TM

- **OBAPs:**
  - On Board Application Procedures
  - More Complex
  - Must be considered as Flight Software
  - Tight Interaction with FSW
OBSW: On Board Procedures

• Source Language
  • Instruction Set
  • Language Constructs

• Target Language
  • Interpreted
  • Pre-compiled
  • Compiled

• Production Chain
  • Preparation Environment
  • Execution Environment
  • Management Environment

• Fault Containment
OBSW : On Board Procedures

Production Environment
On Ground

User

OBCP Editor

Production Program

OBCP Compiler

OBCP

OBCP Syntax

Configuration Data Base

Debugging Environment
On Ground

OBCP Debugger

DHS Stubs

OBC Simulator

Validated OBCP

Execution Environment
On Board

OBCP Interpreter

OBCP Manager

Data Handling Software
OBSW: Trends

- **Functionality**: Improved on board autonomy, file based operations, on board data processing, formation flying technologies,

- **Distribution**, Multiprocessor, Multicore, Time and Space Partitionning, Integrated Modular Avionics,

- **Standard Architecture**
  Reference Architecture, Application Framework, Middleware, Building Blocks

- **Standard Components**
  Packet Utilisation Standard (PUS), Spacecraft On Board Interface Services (SOIS), On Board Control Procedures (OBCP),

- **Methods and Tools**: Component Based Software Engineering, Model Driven Engineering, Requirement Engineering, Automatic Code Generation and Testing, Formal Verification