Software Management of a Real-time, Critical System

Ivar-Andé Ihle
Rolls Royce Marine AS
Software management of a real-time, critical system
**Introduction**

**Analog PID (electrical)**

Op-amp integrator circuit:

**Software PID**

```python
previous_error = 0
integral = 0
start:
    error = setpoint - measured_value
    integral = integral + error*dt
    derivative = (error - previous_error)/dt
    output = Kp*error + Ki*integral + Kd* derivative
    previous_error = error
    wait(dt)
    goto start
```
Introduction & Motivation

Software - an important vessel component

Software management for DP during development

Share practices and experiences

Promote discussion about best practice
Industrial standards

DNV-GL Integrated Software Dependent Systems

ABS Integrated Software Quality Management

MTS TECHOP on Software testing
Software life cycle management

- Requirements
- High-level design
- Detailed specification
- Implementation
- Module testing
- System testing
- Maintenance
Rolls-Royce Product Management

Project Gates

Product/Service Development

- Concept Review
- Preliminary Design Review
- Critical Design Review
- Product Release Review
- Pre-service Review
- In-service Review

Internal Project

- Prospect Pursuit
- Project Launch
- Close-out Review

Project Management

Business Gates

- Bid/No Bid
- Tender Approval
- Proposal (Contract) Acceptance
- Takeover by Customer

Stages

- Preliminary Business Case Stage 1
- Business Case Stage 2
- Execution Stage 3
- In-Service Support Stage 4
- Serial Production Stage 5
- Disposal Stage 6

Capability Acquisition Process
Software challenges

Multiple developers

Continuous development phase

Testing

Safety management

Version control
The challenge with code

```plaintext
previous_error = 0 integral = 0
start:
    error = setpoint - measured_value
    integral = integral + error*dt
    d = (error - previous_error)/dt
    output = Kp*error + Ki*integral + Kd*d
    previous_error = error
    wait(dt)
goto start
```

```plaintext
axyz = 10; speed = input > 5 //testing only
start:
    error = setpoint - measured_value*0.3048
    integral = integral + error*dt
    d = (error - previous_error)/dt
    //intgrl = max(integral, axyz) //Check this
    intgrl = integral/2 // For vessel alfa
    if (speed)
        output = Ki*integral + Kd*d;
    else
        output = Kp*error + Ki* intgrl + Kd* d
    previous_error = error
    wait(dt)
goto start
```
Lessons

Focus on the final product
Consider robustness early.
Design for usability
Keep track of execution time.
Track issues
Transparent development phase

General code base
Organize software structure
Write readable and testable code
Test extensively. Test early
Verify all inputs.
How?
Requirements

Class rules
Business requirements
Customer inputs

Find actual problem and operator need
Analyze problem => find actual need
DP operators interview
Requirements: Usability

Technical coffe pot specification:
• Isolating handle
• Cover to keep content warm
• Precision spout
Icon DP software

Configurator + DP Software = DP Application

<>/ Icon DP code </>
General code base

- General interfaces.
- Generic modules
- Vessel-specific parameters

Diagram:
- Control System
- Thrust Allocation
- Signal Processing
- Vessel Model (Filtering & Estimation)
- Trajectory Generator
- Feedback and Feedforward Controller
- Commanded Thruster Forces
- Estimated Vessel Motion
- Power Limits
- Measurements
- Thrust Device Setpoints
- Operator Input

Configuration files
GUI Design Principles

«Less is more»
Pixel perfect
Consistent User Interaction
Operational flexibility
Modern
Common-look-and-feel
Software code

Understand pros and cons of code languages

Integrated Development Environment (IDE):
- What works best for your needs?
Readable Code base

Coding guidelines

Descriptive file names and methods
Single responsibility

SI units in code.

Mars Climate Orbiter
193 million USD
Disintegrated in the atmosphere.
Altitude miscalculation

Mix of US and SI units.
Code: floating point numbers

Ariane 5 steering failure: Convert sideways velocity from 64-bit to 16-bit format.

Redundant controllers. Similar failure.

Self-destruct before disintegration

Rocket and four (uninsured) research satellites lost
Robust code

Numerical robustness

```
if m > epsilon
  a = F/m

thrust = saturate(cmdThrust, minThrust, maxThrust)
```

Code robustness – input validation

```
functionA(waypointFile){
  ok = areWaypointsOK(waypointFile)
  if ok {
    followWaypoints(waypointFile)
  }
}
```
Product documentation

Configurator + DP Product manual base = Delivery specific manual

LATEX

Document

Sensors
Development tools: issue tracking

- Requirements
- Issue tracking software
- Release notes

- Sub-issues
- Code revision
- Software development
Development organization

Transparent development phase
Understand personal contribution

<table>
<thead>
<tr>
<th>TODO</th>
<th>ONGOING</th>
<th>DONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TASK 1</td>
<td>TASK 7</td>
<td>TASK 9</td>
</tr>
<tr>
<td>TASK 2</td>
<td>TASK 5b</td>
<td></td>
</tr>
<tr>
<td>TASK 3</td>
<td>TASK 8</td>
<td></td>
</tr>
<tr>
<td>TASK 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TASK 5a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TASK 6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Development organization

Distributed development: communication is important

Cooperation is recommended.

Programming in pairs.

Test-driven programming.
Testing during development

- **Unit tests**
  ```
  if(...
  )else{
    ...
  }
  ```

- **Module tests**

- **System tests**
Simulator driven development and testing

Internal "Hardware-in-the-loop" testing

Simulated environment

Failure mode simulations
Unit testing

Test software functions and components
Runs at compilation time
Independent of user inputs
Consistent test scope
Repeatable
No additional effort

```c
if{...
}else{
...
}
```
Module test

Extensive testing of a larger software module
Automatically or manual
Larger variation of user inputs
Different interactions
User interactive
System tests

Fully assembled system
Verify advanced features
Test interfaces
Verify sub-systems and external suppliers
Realistic input handling
Software version control

Software is a dynamic product

Continuous design, development, updates and deployment

Source code management: retrieve earlier versions

Software is never modified on specific installations.
Software safety management

- **Reported safety concern**
  - Communicated internally
    - Safety *alert* report
    - Investigation and analysis
  - Communicated externally
    - Safety *hazard* report
    - Implementation plans
- **Demonstrate safety**
Conclusions and lessons learned

Software safety depends on implementation and following established standards

Transparent software development process

Organize software structures and product

Readable, testable and flexible software

Trace issues