Software Engineering for Satellites

Kathryn Anne Weiss

Software Engineering Research Laboratory
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
October 22, 2003
Topics of Discussion

- Background
  - Why is Software Engineering Hard?
  - Lifecycle
    - Cost
    - Requirements Specification
    - Approaches to Design
    - Implementation
    - Testing
    - Maintenance

- Why is Software Engineering Hard for Spacecraft?
- SERL Approach
- Component-Based Systems Engineering
  - SPHERES
- Conclusions
Background

Ariane 5

Mars Climate Orbiter

SOlar Heliospheric Observatory

Courtesy of Arianespace / ESA / CSG. Used with permission.
Background

- Why is Software Engineering Hard?
  - “Curse of flexibility”
    - “And they looked upon the software and saw that it was good. But they just had to add one other feature ...”
    - No physical constraints
  - Intangibility
  - Lack of historical usage information
  - Organized complexity
    - Too complex for complete analysis
    - Too organized for statistics
  - Large discrete state spaces
Background

◆ Software Lifecycle

- Feasibility Study
  - V & V
- Requirements
  - V & V
- Design
  - V & V
- Implementation
  - V & V
- Testing
  - V & V
- Maintenance
  - V & V
Background

- **Software Cost**

  - Coding
  - Requirements
  - Testing
  - Maintenance
Requirements Specification

- Most critical portion of the software lifecycle
- Majority of errors in software can be traced back to flaws in the requirements
- Many methods and types of requirements including:
  - Informal
    - English
    - UML
  - Formal
    - Zed
    - State Machines
    - Intent Specifications
Approaches to Design

- Software design grew out of the structured programming movement beginning in the 1960s
- Many approaches to design including:
  - Functional Decomposition
  - Object-Orientation (OO)
  - Event-based CBSE
  - Agent Architectures
- What approach to Software Design is appropriate for Satellite Engineering?
Implementation

- Only 10% of the software development effort!!!
  - Other 90% made up of planning and testing
- Issues include:
  - Programming Languages
  - COTS and Reuse
  - Interfaces
Testing

- Examining a program to see if it does not do what it is supposed to do is only half the battle – the other half is seeing whether the program does what it is not supposed to do!

![Graph showing the probability of finding more errors as the number of errors already found increases.](Image)
Maintenance

◆ Comprises approximately 70% of the software lifecycle cost and time
◆ Issues include:
  ★ Deployment and Training
  ★ Code Changes
    • Additional functionality
    • Fixing bugs
  ★ Diagnosis and Troubleshooting
  ★ Job Turnover – understanding someone else’s code
Why is Software Engineering Hard for Spacecraft?

- Spacecraft Software Structure and a Lack of Autonomy
- Loss of Domain Knowledge
- Miscommunication Among Multi-disciplinary Engineering Teams

Proposed Solution:
- Component-Based Systems Engineering
SERL Approach

- Intent Specifications
  - Why? instead of merely What? and How?
  - Design Rationale
- SpecTRM
  - Specification Toolkit and Requirements Methodology
- SpecTRM-RL
  - SpecTRM-Requirements Language
SERL Approach (Cont.)

- Level 3 – SpecTRM-RL
  - Easily Readable and Reviewable
  - Unambiguous and uses simple semantics

- Complete
  - Can specify everything need to specify

- Analyzable
  - Executable
  - Formal (mathematical) foundation
  - Assists in finding incompleteness
Component-Based System Engineering

- Functional Decomposition
  - Spacecraft Level
    - Command and Data Handling Computer
  - Subsystem Level
    - Attitude Determination and Control
    - Power
    - Thermal
    - Communications
    - Guidance and Navigation
    - Propulsion
Component-Based System Engineering (Cont.)

- Top-Down Decomposition
  - Component Level
    - Ex) NEAR’s Attitude Determination and Control Subsystem
      - Sun Sensors
      - Star Trackers
      - Inertial Measurement Units
      - Reaction Control Systems
      - Reaction Wheels
Component-Based System Engineering (Cont.)

- Construct software and hardware intent specifications from the component level to the system level
- Specification Toolkit and Requirements Methodology – Generic Spacecraft Component (SpecTRM-GSC)
  - Fully Encapsulated
  - Well-defined Interfaces
  - Generic
  - Component-level Fault Protection
Component-Based Systems Engineering (Cont.)

- Instead of performing CBSE, engineers can perform Component-Based Systems Engineering, in which the entire process of development (from the component-level to the system-level) is reused

- Benefits:
  - Provides the benefits of Component-Based Software Engineering without the detrimental effects of improper implementation of reuse
  - Supports the principles of systems engineering:
    - Common means of communication
    - Placing the component in context within the larger system
Component-Based System Engineering (Cont.)

- The development is performed in a systems engineering development environment (SpecTRM)

- Benefits:
  - Helps capture domain knowledge through recording rationale
  - Abstracts away the details of design
  - Provides various analyses
    - Simulate design alternatives
    - Nothing has been implemented at this point
  - Easy to incorporate changes to the software
  - Visualizations provide different perspectives on the same system
SPHERES

- Synchronized Position
  Hold Engage Reorient
  Experimental Satellites

Why SPHERES?

1. Autonomous
2. Highly modular
3. Test technique on a real system
SPHERES (Cont.)

[Diagram showing the structure of SPHERES, including Guest Scientist Program, Sphere Controller, Electrical Subsystem, Structural Subsystem, Propulsion Subsystem, PADS, Communication Subsystem, Firing Thrusters, Beacons, and SPHERES Laptop.]
Two Guest Scientist Programs were modeled to illustrate:

- The feasibility/scalability of the technique
- The ease with which the components can be reused
- The process of building a new spacecraft configuration from already existing components
SPHERES (Cont.)

- Rate Damper
  - One Sphere Configuration
  - Nullifies any angular rate experienced by the Sphere
- Leader/Follower (Rate Matcher)
  - Two Sphere Configuration
  - Follower Sphere matches the angular rate experienced by the Leader Sphere
- Demonstration
### Event Log

<table>
<thead>
<tr>
<th>Time</th>
<th>Model</th>
<th>Element</th>
<th>Array Index</th>
<th>Value</th>
<th>Event Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200 millisecs</td>
<td>RateDamp</td>
<td>AngularRateInput</td>
<td>-0.0104</td>
<td>Received</td>
<td></td>
</tr>
<tr>
<td>1200 millisecs</td>
<td>RateDamp</td>
<td>AngularRateInput</td>
<td>-0.0033</td>
<td>Received</td>
<td></td>
</tr>
<tr>
<td>1200 millisecs</td>
<td>RateDamp</td>
<td>AngularRateInput</td>
<td>-0.0104</td>
<td>Changed</td>
<td></td>
</tr>
<tr>
<td>1200 millisecs</td>
<td>RateDamp</td>
<td>AngularRateInput</td>
<td>-0.0033</td>
<td>Changed</td>
<td></td>
</tr>
<tr>
<td>1200 millisecs</td>
<td>RateDamp</td>
<td>GuestIdentModeInput</td>
<td>AccelerometerGyro</td>
<td>Changed</td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

- The research on and the test case application of Component-Based Systems Engineering show its potential for use in developing the next generation of spacecraft.
- The benefits of using the technique span not only the engineering issues faced by today’s spacecraft development teams but also the difficulties inherent in the aerospace industry.
Questions and Comments