Command and Control Software Development Lessons Learned

Lt Col Michael D. Sarchet
Deputy Director,
Space Systems Command and Control Division
Agenda

- Two real world case studies
- Lessons learned
  - Management
  - Technical
- Summary
Two Real World Case Studies

Case A
- First OO development
- No OO reuse
- Period of performance ’94 – ’03
- ~ 1.2M LOC

Case B
- Follow-on to Case A
- Reused Case A and other OO software
- Period of performance ’97 – ’03
- ~ 600K LOC
Management Lessons Learned

- Execution of OO development
- Getting and keeping good resources
- Incremental development
- Maintenance of multiple baselines
- Software reuse
Execution of 00 Development

Program A

At the time – no known successful OO development model
  + Received Academia support
  + Identified early training needs
  - No OO s/w processes defined and in place
  - Spent too much timing refining 00 software development processes
  - Underestimated productivity – new software methodology, tools, and processes
  - Down staffed system engineers with domain expertise too early

C++ language needs to be mastered
  - Code reviews and examples needed
  - Code reuse must be architectured from the start

Program B

Incorporated Lessons Learned from Program A and sub-contractor experience
  + Effective use of 00 mentors
  + Maintained system engineers through system delivery
  + System engineers maintained architecture change control authority

Incorporated reviews
  + Design completeness and consistency reviews
  + Peer code reviews
Getting and Keeping Good Resources

Program A
- OO development attracted the best and brightest programmers
- Encouraged existing programmers to upgrade skills
- Lost resources and talent to competitive job market
  - Lost training investment
- Needed more domain knowledge from legacy program

Program B
- Decoupled roll-off dependency from program A
  - Program B was initially dependent on resources rolling off Program A
- Developed incentive programs to keep personnel
- Hired and maintained system engineers with domain experience

Tie training investment to employment commitment
**Incremental Development**

<table>
<thead>
<tr>
<th>Inc 1</th>
<th>Inc 2</th>
<th>Inc 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>Code</td>
<td>Subsystem I&amp;T</td>
</tr>
<tr>
<td>Design</td>
<td>Code</td>
<td>Subsystem I&amp;T</td>
</tr>
<tr>
<td>Design</td>
<td>Code</td>
<td>Subsystem I&amp;T</td>
</tr>
</tbody>
</table>

**Overlapped incremental development**

- Large schedule risk
- Pushes out start of subsequent increments
- Creates resource conflicts; hardware and personnel

*Need separate teams and dedicated resources*
Maintenance of Multiple Baselines

Maintaining multiple baselines complicates:

- Schedule management
- Resource availability (hardware, personnel, facilities)
- Configuration management
- Regression testing (threads and requirements)
- Requirements sell off / verification

Decrease number of baselines
Software Reuse

- Software *NOT* designed for reuse
  - Scrutinize productivity estimation
    Realized 20% increase

- Software designed for reuse
  - Supports successful implementation
    Met schedule and cost goals

- To mitigate software reuse dependencies
  - Identify dependencies early
  - Identify gates and pass criteria
    Functionality and need dates
  - Identify contingency plans

*Design for reuse!*
Technical Lessons Learned

- Prototyping of high risk areas
- COTS products
- Software integration and test
- Requirements verification
Prototyping

- COTS risk mitigation efforts partially successful
  + Effective in the determination of COTS viability
    - Failed to reveal system performance implications

- Prototyping of inter-communications needed between prime and subcontractor software
  + Enabled prime and subcontractor to be highly successful in their reuse of different legacy OO software

Early prototyping is a major component in risk mitigation
COTS

- Select vendors with eyes wide open
  - Health / economics
  - Maintenance philosophy

- Vendor products need to adhere to standards
  - Evolving standards make development difficult
    - Caused rework and schedule impacts
  - Proprietary solutions are not solutions
    - Caused external interface incompatibility

- COTS product / functionality eliminated in subsequent upgrades
  - Forced custom software solution
COTS (cont.)

• COTS rapid evolutions
  • Developments cannot keep up
    • Need to maintain baseline stability beyond COTS upgrade releases
  • COTS products highly coupled
    • Vendors not on same release / upgrade timelines
    • Upgrading one product has a domino effect
    • COTS timelines drive program upgrade schedule
  • Forces external interfaces to adhere to version numbers
  • Every upgrade impacts cost and schedule

• Discontinued maintenance support drives
  • Life cycle cost
  • Custom solutions

A balance between COTS and proprietary solutions needs to be achieved
System Integration and Test

- Protect system integration period
  - Realistic integration period needed
  - Rolling deliveries cause inefficient integration
  - Inefficient integration causes re-runs of tests and extends test period

- System testing
  - Pay early attention to schedule and resources
  - Review productivity estimation from one increment to next
  - Define ASAP test philosophy, processes, and guidelines
  - Review resources (hardware, personnel) against schedule
  - Avoid maintenance of multiple baselines
Requirements Verification

Incremental requirements verification

- Test majority of requirements during incremental testing
- Retest requirements and functionality in final increment
  - Software is built upon or changed in each increment
- Maintaining partial verification matrix is a nightmare
  - Program A: no partial requirements verification
  - Program B: 90% requirements verified in last increment

Don’t expect requirements sell-off in early increments
Do test requirements at each increment
Requirements sell-off viable only in last increment
Summary

- Need a good model to execute an OO development
- Tie training investment to employment commitment
- Overlapped incremental development needs separate and dedicated resources
- Decrease number of baselines
- Design for reuse
- Early prototyping is a major component in risk mitigation
- A balance between COTS products and proprietary solutions needs to be achieved
- Pay early attention to Integration and Test philosophy, schedule and resources
- Requirements sell-off viable only in last increment