Evaluation of Components for COTS-based Systems

GSAW Conference
March 2002
Definitions

- **COTS** - commercial off-the-shelf
  
  Came Out Too Soon
  
  Currently Offered, Totally Smoke

- **GOTS** - Government Off-The-Shelf
  
  GOne To Seed

- **NDI** - Non-Developmental Item
  
  Nobody Did It
  
  Next Design Iteration
The Problem

- More software-intensive systems
- Increasing development & maintenance costs
- Missed schedules & performance requirements
OTS Component-based Ideal

- Smaller proportion of developed SLOC
- Testing of software over a broader user base
- Faster system development
What’s Happening?

- Multiple vendor dependencies
- Version creep
- Increased integration costs
- Missed schedules & performance requirements
- High expectations/promises for “out-of-the-box”
Successful COTS

- Computer hardware
- Operating systems
- Relational database management systems
- GUI development tools/libraries
“Have some wine,” the March Hare said in an encouraging tone.

Alice looked all round the table, but there was nothing on it but tea. “I don't see any wine,” she remarked.

“There isn't any,” said the March Hare.

“Then it wasn't very civil of you to offer it,” said Alice angrily.

From Alice’s Adventures in Wonderland by Lewis Carroll
Managing Requirements (cont.)

- Identify unique system features
  - Unique features may drive development

- Prototype using selected components
  - Confirm true capabilities and performance
  - Learn strengths & weaknesses
  - Find bugs and workarounds

- Develop requirements iteratively
  - Prototyping aids in refining requirements
  - Requirements drive component selection

- Good communications between customer and COTS provider is rare and highly under-rated as a factor to success
Robust Communications of Requirements and Capabilities

- **Requirement** = stretching muscles
- **Capability** = physical strength (via energy bar)
- **Problem** = interpretation and implementation without communication
- **Result** = long walk home
Conventional vs. Frameworks

- Architecture must support adaptation and tailoring
- Conventional API approach vs. Framework Approach
- Conventional examples: Unix, IMSL
- Framework example: Rational ReqPro

![Diagram showing Conventional API and Framework]

- Custom Code
- COTS Component
- COTS Component
- Conventional API
- Custom Code
- Custom Code
- Framework
- COTS Component
Key Component Characteristics

- Existence - Off-the-shelf including key features
- User-extensible
  - Plug-ins (.DLL’s, etc.)
  - Scripting
  - Data-driven (configuration files, etc.)
  - Inheritance
- Accessible data
  - Defined interfaces / Industry standards
- Instrumentation - visibility into operation
- Vendor Support/Release History
- Common baseline across customers
Long-Term Maintenance

• “To freeze or not to freeze...”
  – that is the question
• 15 year programs / 5 year hardware
  – Lots of spares
    or
  – Periodic updates
• 15 year programs / 3 year software
  – Freeze software including OS
    or
  – Periodic updates including OS & HW
**Solaris™ Timeline Example**

Sun Solaris Release History

- First Release to Last Ship Date
- Vintage Support, Standard rate, no cosmetic patches, no enhancements
- Vintage Support, Premium Rate, no cosmetic patches, no enhancements

Conclusions

- Know your components and manage your requirements
  - Iterative requirements development is worth it
- Select components carefully
  - Evaluate the component not the feature list
  - Either the architecture or the component must support the needed tailoring and customization
- Plan for obsolescence of hardware & software
  - Program life vs. component lives
Rowing Together: Cooperation Between COTS Provider and Customers is Key