

Key Ground System Architectural Issues

Data Accountability

- * Unique data product identifier that travels round-trip
 - Originates with user request
 - Accompanies data request via uplink
 - Preserved through data generator(s)
 - Accompanies data returned via downlink
 - Preserved throughout ground data processing pipeline
 - Final product matched to request

- * Cooperating databases of accountability status to answer queries
 - "Where's my data?"
 - "Why does it look like this?"
 - "Is that all there is?"
 - "Can the gaps in my image be filled?"
 - "What were the processing parameters used on my data?"

Key Ground System Architectural Issues

Flight-Ground coordination

- * Flight and Ground Systems are cooperating parts of a single Flight-Ground System
- * Derive both flight and ground software from the same database
 - Command parameters
 - Telemetry channels
 - Flight/Ground SW Configuration parameters
- * Flight and Ground use each other's tools to develop software
- * Test as you fly
 - Use Ground System to test Flight System pre-launch, and vice versa
 - Requires an integrated test schedule (difficult across multiple organizations)
- * CM both Flight and Ground software together
 - Rapid turnaround is necessary during development

Key Ground System Architectural Issues

Integrating with legacy systems

- * Decide what to keep and what to replace
 - Keep: well-maintained, highly valued by users
 - Discard or Replace: bloated, obsolete, arcane, ugly
- * Understand architecture of legacy systems
 - Avoids unnecessary re-invention
 - Helps to know how to cleave along fault lines
- * Multiple development organizations adds to lead time
 - Up to 1.5 years from release of new OS to deployment of new GDS
 - OS → 3rd Party COTS → Multi-Mission Infrastructure → Mission Specific Adaptation → User Configuration
 - Complicates scheduling of dev/integrate/test/install cycles
- * "Software reuse without contracts is sheer folly" - Bertrand Meyer

Thoughts on “Architecture”

- * Best viewed in hindsight
 - ”Architecture is what you wish you had, but never have time for”
- * System Architecture: includes people, holistic, globally optimized
 - Any system is also a subsystem, any subsystem is also a system
 - Ground System is part of a Flight-Ground System
 - Flight-Ground System is part of a Multi-Mission System
- * Software Architecture: design patterns, frameworks, ...
 - Good substitute for a system architecture
 - S/W can accommodate any architectural concept (n-tiered, m-layered,..)
- * Scale-dependent
 - Small = limited use, limited time frame, simple single-point design
 - Medium = single-purpose, multi-user, design for change
 - Large = multi-purpose, multi-decade, user-configurable
- * Best if an architect owns it

Success Factors

A Good Architecture ...

- accommodates or redefines the management structure
- has an architect
- is scaled appropriately for its context
- has the Ground as part of a Flight-Ground architecture
- strikes a balance between automation and operations
- optimizes across hardware, software, people, cost, risk
- contributes to overall mission success
- has longevity, can evolve
- survives a "good enough" implementation

A Good Architect ...

- "feels the pain" of the developers, testers, and users
- maintains a constant vision
- communicates to developers and users in their language
- refines architecture with top-down & bottom-up iterations
- is a generalist with expertise that is both broad and deep
- understands the importance of system engineering
- knows that "can be done" does not mean "should be done"

A Good Engineer...

- understands the importance of the architectural plan
- can overcome a flawed architecture

A Bad Architecture ...

- ignores or is driven by the existing management structure
- is developed by committee, or happens by accident
- is too simple or too grand for the real world
- treats the Ground as the only "System"
- uses AI where humans would be better, or vice versa
- focuses on software only (or people, or hardware, or...)
- focuses on one phase of mission (e.g. getting to launch)
- becomes obsolete quickly, too rigid
- requires an "all-or-nothing" implementation (monolithic)

A Bad Architect ...

- lives in an ivory tower
- has no vision, or often changes it
- cannot translate the vision into vernacular
- fails to adjust for practicalities
- is a specialist in one discipline or paradigm
- thinks that having a good architecture is sufficient
- insists on using the newest fad just because

A Bad Engineer...

- equates "system engineering" to "system architecture"
- cannot fill in the missing pieces during implementation