A Philosophy of Software Architecture

Preliminary Thoughts From the Perspective of Transformational Communications MILSATCOM (TCM)

Presented at:

Architecture-Centric Evolution and Evaluation (ACE2)

Sercel, Gat, and Hutchison

TCM Program

Topics

- Modern aerospace software
- Proposed definition of *Architecture*
- Some possible benefits of this definition
- Application to change management
- Operationalizing this view

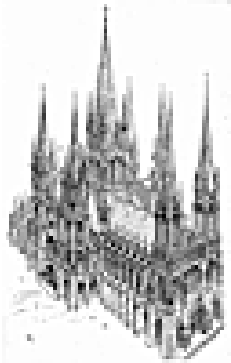
Modern Aerospace Software

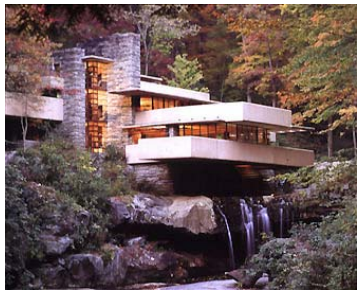
- Heterogeneous
- Multiplatform
- Distributed
- Large Scale
- Multi-contractor
- Evolutionary
- Backwards Compatible
- Standards Based (or not)

Architecture?

To (structural, naval, and other) architects…

is *a set of features shared by a set of designs.*





Why Bother?

- Proposed *Equivalent* Definition:
  - An architecture is *a set of constraints on design*

- Why This Definition?
  - *Good* constraints *can* guide designers towards *good* designs
    - Or at least away from bad ones.

Architecture May Aid Software Development and Engineering

- Interoperation
- Legibility
- Extensibility
- Change Management*
- Synchronization
- ….

* More on this.

TCM Software and Change Management

- TCM is a long-term program.
  - Changing requirements and technologies are inevitable.
- Without constraints, any change may require a complete redesign.
- Hence, these thoughts regarding architecture.

Two Classes of Constraints

- Compositional constraints
  - Decomposed into subsystems/components. (Often all that is meant by “architecture”.)
- Meta-Compositional constraints
  - Constraints on how components work, work together, or are designed
  - See examples/discussion

Two Types of *Constraints* That Aid Change Management

- Invariants
  - Guaranteed not to change
- Conditional invariants
  - Don’t change unless specific condition met.
  - Example: Design decisions placed under CM.

Examples of Such Constraint

- All interface data protocols must be (SDSI)
  - SD *(Self-Delimiting)*
    - Information on where data fields begin and end
  - SI (*Self-Identifying)*
    - The identity of fields is contained within the data stream itself
- Examples:
  - XML,
  - S-expressions

XML Could Be SDSI for Spacecraft Control

```
<command>
  <cmd_type>set_mode</cmd_type>
  <mode_id>standby</mode_id>
</command>
```

As opposed to:

```
00001001001100100000000000011011
```

plus a document that says:

CMD type (16 bits)	MODE (16 bits)

SDSI Protocols (cont'd)

- Easier to change than fixed-field protocols
  - No fixed-length fields
    - No code rewrite to accommodate larger data sizes
  - No ordering dependencies
    - No order-dependent bugs
  - Automatically backward-compatible
    - Unknown field types can simply be ignored
- But no free lunch
  - SDSI protocols are harder to parse, less efficient
  - Moore's law and shared code bases make this a good trade

Other Possible/Example System Constraints

- The system shall be evolutionary
  - Elements designed and configured to accommodate rapid and orderly modification.
  - May encompass response to existing or new requirements and technologies.
- The system shall support automated verification
  - Verify intended functionality of all changes to
    - operational software,
    - database, and
    - procedures

    in the intended operations environment.

Architectural *Constraints* Support: Software Maintainability & Extensibility

- Can be defined to ease the assembly and interactions among components of an architecture.
  - May be recursively decomposed into primitive system elements
- Embracing standards
  - Reduce engineering load on the program
  - Maximize shared code and/or COTS
- Facilitates software maintenance/evolution
  - Faster and cheaper upgrades and changes

Operationalizing These Thoughts

- Establish program level software IPT
  - All major software developers within program contribute
  - Tasks broader than S/W architecture
- Key Tasks/Products
  - Collaborative definition of program software architecture(s)
    - Two example views from this discussion include *syntactic* and *design*
  - Balance contractor/specific processes and constraints against program level view

Summary

- Effective *constraints* define effective *architecture*
- Necessary for managing change
  - Architecture (the concept, if not the term)
- Should be defined early in the product development life cycle and maintained as collaborative product of software IPT