Architectural Techniques for Interoperability and Coexistence

Tuesday, June 6, 2006

IEEE Computer Society
Twin Cities Chapter
Minneapolis, Minnesota

Robert Gezelter Software Consultant
35 – 20 167th Street, Suite 215
Flushing, New York 11358 – 1731
United States of America

+1 (718) 463 1079
gezelter@rlgsc.com
http://www.rlgsc.com

Enabling Coexistence and Interoperability

- Longevity
- Completeness
- Seamless Extensibility
  "The universe is stranger than we can imagine"

Achieving Longevity

- Bad – Inertia, Fiat
- Good – Conceptual Integrity

Aesthetics of Function

- Beauty of Function
- Similar but not identical to civil architecture
- Cause and Effect often reversed
- Aesthetics is the result, not the cause
- Consider a–>b is not the same as b–>a
- What “Looks Good” really means

Why Architecture? Consider an Example

- Computers are universal, aren't they?
- Basic Theory of Computation – Universal Turing Machine
- from geometry: Cartesian v. polar coordinates
- from mathematics: calculus

Why Architecture? – Time-Shared BASIC System

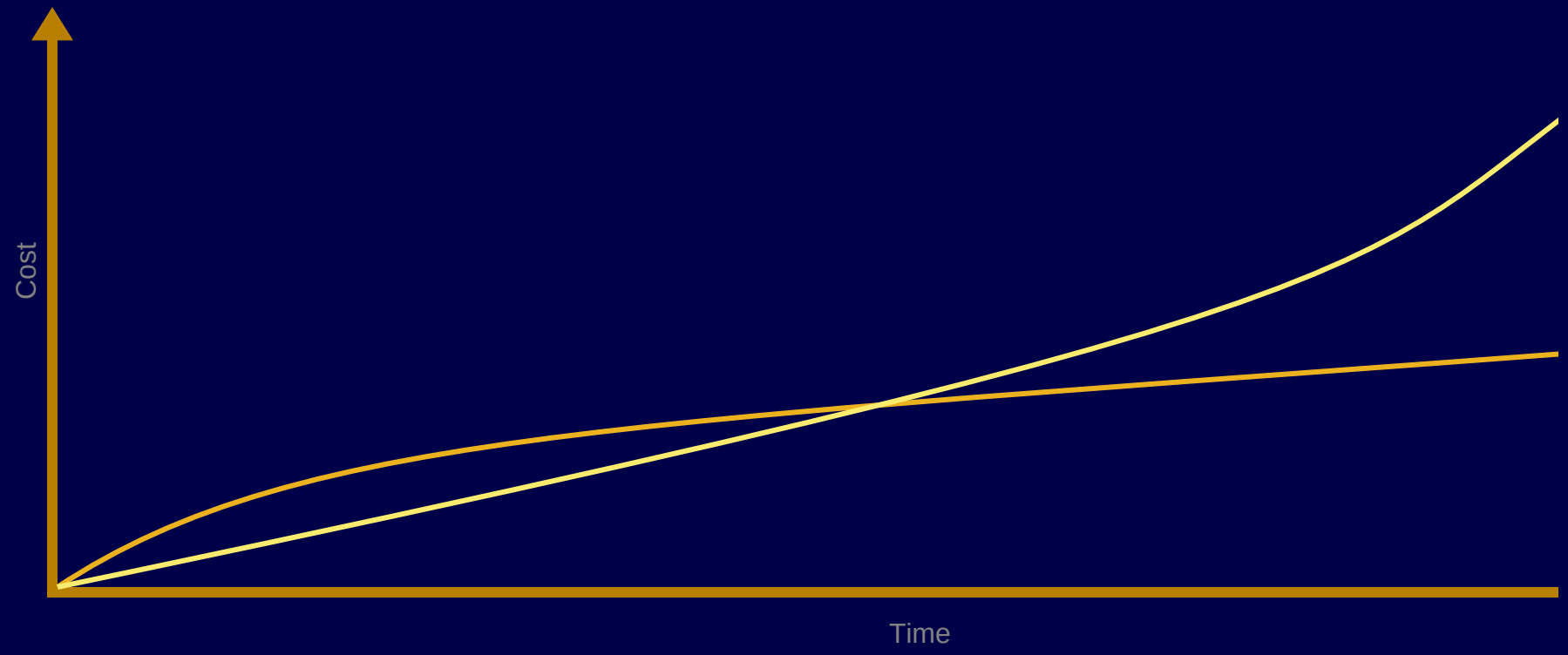
- "Time-shared BASIC" system
- multi-function: Payroll, Editing, A/R, A/P, MIS
- interwoven code gets complex quickly
- unrelated parts interact in unexpected ways

Why Architecture? – Classic Multiprogramming

- examples: OS/360 MxT, RT-11, RSX-11, OpenVMS, UNIX
- abstractions: user; process/task; files; protection
- restricted interactions between applications
- inverse exponential simplification of interactions

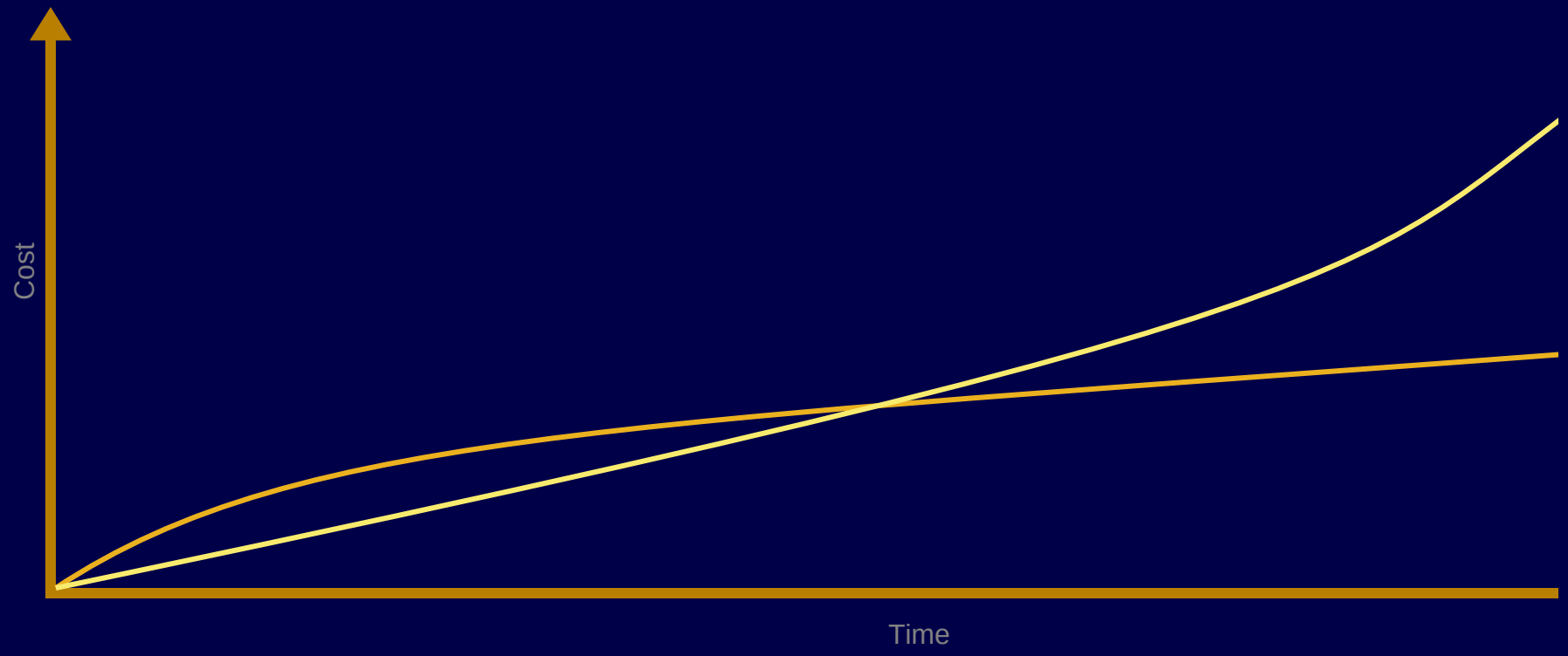
Aesthetic Costs over Time

- no redundant code – reduced line count
- alternate pathway bugs eliminated



Aesthetic Costs over Time

- reduced interaction effects
- assimilating increasing scope



The Requirements Trap

- Immediate Necessary is not e.e. Long Term Sufficiency
- Change is inevitable

Inertia – An Excuse

- low quality solution
- fear of change
- long term problem, entrenchment
- the longer inertia continues, the worse it gets

Positive – Conceptual Integrity

- conceptual integrity
- small base of good code
- little need for modification
- contract between implementation communities

Good Architecture is Good Architecture

- Architecture is Architecture
- Fads come and go; Style is timeless
- Critical examination of non-software architectures are well established and understood
- Software architectures are seductively malleable
- Exotica is more important than it appears

Good Architecture is Good Architecture

- intellectual precedents
  - Buildings
  - Ships
  - Aircraft

Software is not Different

- software and systems are deceptively malleable
- but, the malleability is not real
- code, once developed, is not malleable
- once systems are built to an interface, changes are expensive in effort, and schedule

Real Examples of Architecture:

- Y2K – The Most Costly Software Change in History
  - YYMMDDhhmmss
  - Too Short; should be YYYY
  - Printable Representation >= 14 bytes
  - Slow – Mixed Bases

Representation: Binary v. Printable

- Space – Mass Storage, Memory
- Binary – 8 binary bytes more than adequate
- Low break even on conversion

Slow Speed: Mixed Bases

- Years (YYYY) – Open
- Month (MM) – 12
- Day (DD) – 28 to 31
- Hours (hh) – 24
- Minutes (mm) – 60
- Seconds (ss) – 60

Feasibility Proof: OpenVMS TOY QuadWord

- from initial design: 64-bit binary value
- origin date: 17 November 1858 (Smithsonian)
- unit: uFortnight (100 us)
- single base
- library provided for conversion
- plus/minus dates straightforward
- Y2K problems limited to ported code and knick-knacks

What an architecture does:

- balance needs of different constituencies
- contract between communities
- aesthetics of utility
- form follows function

Computer v. Real World

- changeability is deceptive and illusory
- traditional architecture is better guide
- good architecture seems effortless
- good architecture minimizes interminable changes and scaffolding
- flexibility is not same as changeability

Positive Examples:

- IBM System 360/370/…
- “undefined means undefined”
  – System/360 Principles of Operation, 196x
- RFC 821/822, & revisions – SMTP

IBM System/360/370/…:

- "undefined means undefined"
  – System/360 Principles of Operation, 196x
- empirical checks outside architecture prohibited

RFC 821/822, & successors – SMTP:

- address only specified (by RFC 822) to the extent of the "@" separator and the resolution of the right-hand domain name
- handling of left-hand side left to destination agent
- Good: suggested line length of 80
- Bad: limit not required, often ignored; arbitrarily long lines permitted and a problem for non-stream file systems

Common Features of Positive Exemplars

- Demur on issues that are unneeded
- Documented ways to extend the architecture
- Specify what is needed – no more
- Avoid hubris
- Special cases are indicia of weakness

Negative Example:

- MS Windows 3rd Party
  - Stores GUID of CD-ROM Installation device
  - Doesn’t deal with multiple CD devices
  - GUID unique to Manufacturer/Model of drive
  - Other common alternative – drive letter
  - Search for volume label would be trivially different yet dramatically increases robustness

The "Snowball" Effect

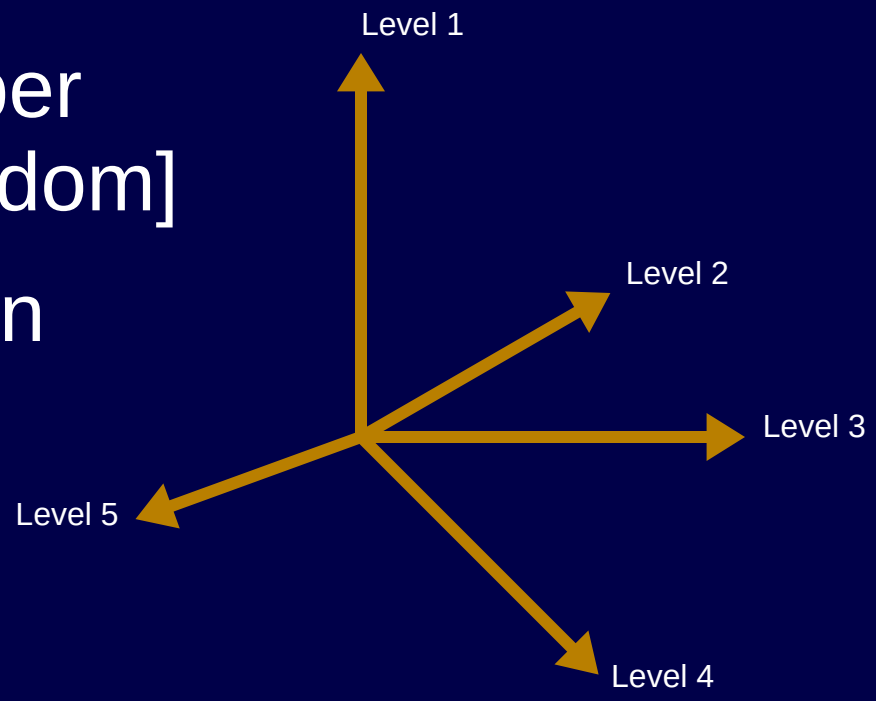
- Good begets better!
- Bad just gets worse –
  "The gift that keeps on giving. PAIN!"

Tools and Diagraming Drive Designs

- Classic Stack –
  ISO Open Systems Interconnect model
- One dimension – obvious
- Two dimension – difficult to visualize
- Multi-dimensional – almost impossible to visualize or discuss

Vector "Degrees of Freedom" Diagram

- ok, my nomenclature (Gezelter, 2004)
- each level is independent
- a full implementation needs >= 1 element per vector [degree of freedom]
- freedom of substitution



Vector “Degrees of Freedom” Diagram (cont’d)

- based on “Programming by Menu” concept (1992)

Scope of Architect's Work

- after requirements analysis/document
- define range and scope of architecture
- decouple wherever possible
- define successive minimal subsets

Decoupling is (EE) Agility

- agility is the ability to assimilate new roles without requiring unrelated changes
- agility is achieved by decoupling

Design and Specify Interfaces

- per System/360 PrOp – don't experiment; read the specification
- architectural interpretation process
- review board

What makes for a good, long-lived architecture?

- embracive
- expressive
- efficient for all players
- for simple user: straightforward
- for sophisticated users: expressive and unrestrictive

What happens if an architecture is not embracive?

- inevitable diverging extensions
- no back-compatibility
- Balkinization
- exponential increases in bugs, maintenance, TCO

What happens if an architecture is not expressive?

- “off-books” work arounds
- non-uniform interface
- examine UNIX read/write/get/put/control/select
- contrast with OpenVMS $QIO, uniform portal interface
- think “aviation English”; not “Shakespeare”

Reduce Pyramid of Complexity

Conclusions

- Good architecture is vital to leverage
- Flexibility is greatly enhanced by proper architecture
- Proper architecture is not a straight-jacket
- The most successful architectures are tremendously enabling
- Interoperability is dependent upon well-designed architectures

Questions?

Robert Gezelter Software Consultant
35 – 20 167th Street, Suite 215
Flushing, New York 11358 – 1731
United States of America

+1 (718) 463 1079
gezelter@rlgsc.com
http://www.rlgsc.com

Session Notes & Materials:
http://www.rlgsc.com/ieee/Minneapolis/2006-06/index.html