Systems Integration

Case Study - From Hierarchical to Layered Systems

Pere Palà

iTIC http://itic.cat

v1.1 October 2015

Source: A significant part is from Mark W. Maier and Eberhardt Rechtin's *The Art of Systems Engineering 3rd Ed*

Introduction

First fact

- Software is naturally constructed as a Layered System
    - In contrast to classic systems engineering: Hierarchical Systems

Case Study: MedInfo

- Makes medical imaging systems: x-ray, computed tomography (CT), magnetic resonance imaging (MRI)
- Clients: hospitals and clinics worldwide
- They are integrated into the user's technical infrastructure (so far as possible)
- Starting point: each system is designed, manufactured, sold and operated as a stand-alone system
- Business progression
    - Upgrades to systems and introduction of new imaging systems
    - Each product has own product manager
    - Each product has its chain of suppliers

Motivation for Change

- Incremental improvement is feasible with current structure. But...
- Software cost
  - Hardware-dominated cost to software-dominated cost. Now it is 70%
  - Hardware: commodity. Available through subcontracting
  - Competitive differentiation comes from software
  - User demands: processing algorithms, display, customization
  - Need for interconnection and integration
- User demand for interconnection and integration
  - Radiologists need different imaging technologies during one day
  - Different computers? File transfers?
  - Simple integration: A single viewer platform, move data to common platform
  - Complex integration: combine, overlay or jointly process images from different systems

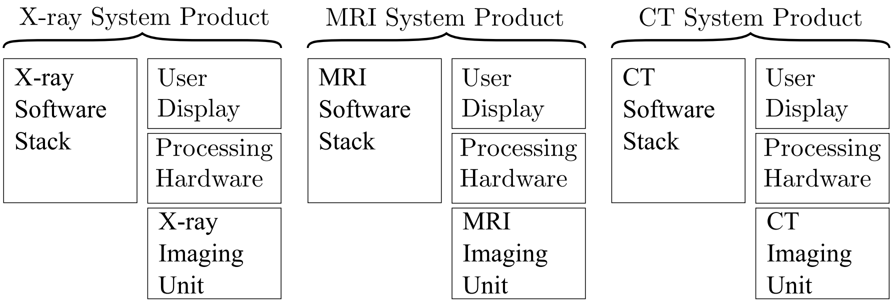
Motivation for Change /2

- Shorter product cycles
  - Competition makes new products faster
  - Need to match expectations
- Lateral and vertical product space expansion
  - Pressure to grow
  - To be integrated into medical information systems...
  - ...means:
  - Try to expand your boundaries
  - Or others may expand them
  - Integrated system markets may become “winner take all” markets

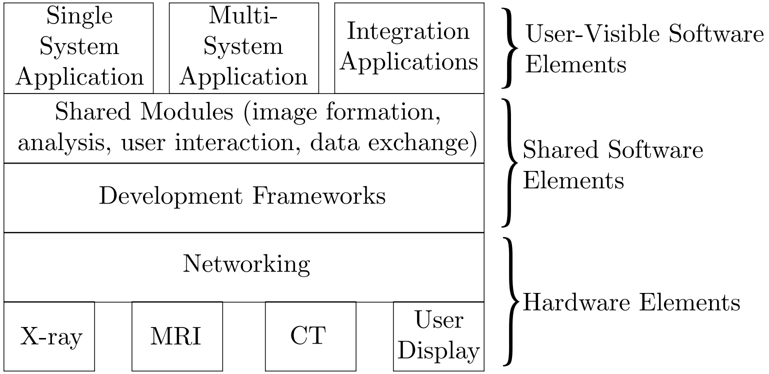
Layered Alternative

- Becoming software-dominated (in cost) means that different products share software
    - Networking
    - Data storage
    - User interface
- Integration means protocol sharing (code sharing)
- Build layers isolating parts that change from each other
- Hierarchical system decomposition and end-to-end product managers make it difficult to discover and manage shared code

Original View



Layered View



Layered Systems

- Products may (or not) look as before
- Client wishes stand-alone system: ok
- Client wishes integrated system: ok
- Hierarchical: a lower-level element is *part of* a higher-level element
- Layered: a lower-layer element *provides services* to a higher-layer element

- Idea borrowed from ISO's Open System Interconnect

Transition

- End-to-end management responsibility changes
  - In stovepipe organizations an individual is responsible for the product
  - Problem $\rightarrow$ Fix for the product
- When something goes wrong, who is responsible for the fix?
- Product manager has no control over all elements
- Problems have to be solved at a level lower than CEO
- Financial decisions are at CEO level
- Quality management?
  - Some quality thresholds may be different for different products
  - How to enforce standards when they do not relate to customer perceived quality (but have cost)?

The quality requirements on the components of a shared layer are likely to be much more demanding than when those components are not shared

Transition /2

- Development of automated software tests
- Shared libraries with assertions: Predicates indicating that something has to be true
- Subcontracting / Outsourcing
  - Specification of a layer is different than specification of a box
  - Is expertise in specifying in-house?
  - Test and integration. How? Each subcontractor buys licenses? etc
  - What if subcontractor goes out of business / drops support / releases a poor version?

Conclusions

- Layered architecture can drop total lines of code
- But: overhead of a new development environment
- Can allow integration
- But: can be a long way
- If layers isolate areas of change faster product evolution may happen. Choose good invariants (e.g. TCP/IP)
- Transition will be painful (related to the human rather than the technical side)