# Systems Integration

## Software and Information Technology Systems

Pere Palà

iTIC http://itic.cat

v1.0 October 2013

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

# Introduction: The Status of Software Architecting

## Introduction

- Software is becoming the centerpiece of complex system design
- Developers of products are developing software
- Why?
  - Ability to create intelligent behavior
  - Ability to accomodate changing trends (technical, economical)
- Other fields may be more stable (physical architecture of aircraft, spacecraft, cars)

## Anecdotal Evidences

- From 70% Hard 30% Soft to 30% Hard 70% Soft
- Complete Systems-on-a-Chip (SOC). The differentiation comes through Software
- From hardware designer to hardware integrator and software developer

# Information Technology

- Integration of computers and communications
- Trend: IT practice is not developing applications but integrating preexisting applications
- Well-architected software can evolve
  - Evolution of software is more rapid than evolution of hardware (cost). Regular full replacement is feasible
- Software is flexible
  - Good medium to implement system intelligence
- General purpose hardware
  - Economies of scale: hardware is cheap
- General purpose software
  - Operating systems, web applications
  - Open-source and reusable code
- Software architecture is important!

# Software Architecture and Trends

## Software Architecture

- Structure of a software system: components and interfaces
- Plus: behavior, constraints and applications

## Trends

- Software: from support role to centerpiece
- Hardware selection: depending on the ability to support software (and not the converse!). AVR vs PIC
- BUT: The system (and not the software) is THE end product. Client acquires the system, not the software!

# Software as a System Component

- System architecture and software architecture
- Software provides abstractions for creating system behavior (software layers)
- Software allows evolutionary delivery
- Software must be integrated into a hardware system
- There seems to be no successor to software as a tool to implement behaviorally complex systems

# Software for Modern Systems

## Characteristics of modern systems

- Storage of large volumes of information and its semiautonomous and intelligent interpretation
- Responsive human interfaces. Mask the underlying machine. Operate in metaphor
- Semiautonomous adaptation to the behavior of the environment and individual users
- Real-time control of hardware (faster than human) with complex functionality
- Constructed from mass-produced computing components and unique software (customizable)
- System coevolutions with customer. Experience changes perceptions of what is possible

# Software for Modern Systems /2

- High-level languages plus general-purpose computers $\rightarrow$ complex, evolutionary systems at reasonable cost
- Achieving the same with hardware is orders of magnitude more expensive. Evolution is more difficult.

- Software layers allow more behavioral complexity
- Trend: machine language, assembler, general-purpose (C, Ada...), domain-specific (MATLAB, SQL...)
- Language models become closer to application
- Computational abstractions

# Systems, Software and Process Models

- Challenge: integration needs of hardware and software
- Hardware is best developed with few iterations
- Software can and should evolve through iterations
- Hardware: well-planned design and production cycle. Large-scale production deferred to the final delivery
- Software: requires access to the targeted hardware platform
- Software distribution costs are low. Except if certification is required
- Hardware changes?

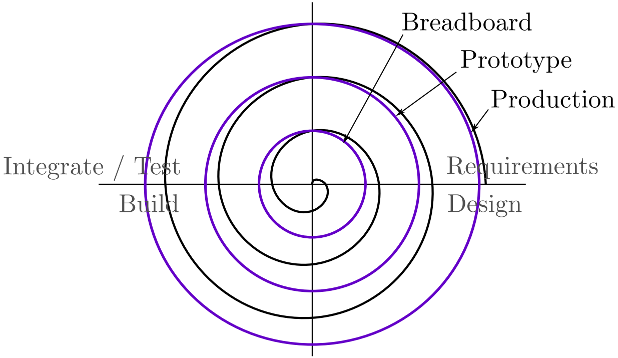
# Waterfalls for Software?

- Hardware: process chosen is usually waterfall. Iterations are local to each phase
- Software can use a waterfall model: design, coding, test and delivery
- Spiral model is the usual choice. All successful software systems are iteratively developed and delivered
- Spiral may help fixing problems discovered in the field
- Waterfall tries to avoid them with good requirements.
  - Communication protocols may be not precise enough, not fully implemented
  - Therac 25

# Spirals for Hardware?

- Spiral for hardware means frequent prototyping
- For systems which are one-of-a-kind, a prototype is a full system!
- Prototype parts of the system
- Build scale models
- For mass-produced systems prototype cost may be reasonable
- Have to be built on production lines similar to the final one. Also expensive

# Integration: Spirals and Circles



# Iterations

- Stable hardware forms should appear early
- Software iteration: aiming at release 1.0 for production hardware
- Hardware-software codesign: make physical prototypes unnecessary. A log way to go!
- Software: Do the hard parts first

# Hierarchy

- Systems can be viewed in hierarchies
  - System composed of subsystems composed of small units
  - A system may be embedded in higher-level systems becoming a component
  - Decomposition in design, integration in reverse
- This view may not match software development
  - Object-oriented abstractions
  - Layers
  - Infrastructure objects (databases, operating systems)

# Objects

- Collection of functions and data
- Objects run concurrently with other objects
- May run on different machines
- Number of objects may be unknown. Determined at runtime
- May work in a network, in arbitrary numbers, depending on events

# Layers

- Are a form of hierarchy
- But lower-level elements do not appear at the higher level!
- Within a layer, objects are peers: not contained one in another

- Middleware layer: objects
- Some middleware services provided by OS. Other by external software units
- Software hierarchy becomes disconnected of hardware hierarchy. (This is THE objective)

# Infrastructure Objects

- Large objects: operating systems and databases
- Millions of lines of code. Rich functionality
- Architect has to adapt to these components (or develop his own??)

# Hierarchies Reconciled

- Software is biggest part of cost $\rightarrow$ Adopt software view. Not necessarily!
- Much software is not object-oriented but procedurally structured
- Hierarchical view and layered view are alternate views –not exclusive
- Each partitioning has advantages and disadvantages
  - Autonomous pieces are (sometimes) attractive: Have their own software, may be independently developed...

# The Role of the Architect

- Architect is the user's advocate
- Responsible for what-it-does and how-it-does. With a limit: up to the system concept.
- Look at the software and the underlying hardware as an integrated whole that delivers value to the user
- Make the system evolvable by paying attention to the interfaces

# Programming languages

- Languages influence, guide and restrict our thoughts
- Some problems decompose easily in one language and with difficulty in others
- Each language encapsulates lower-level languages
- Statements in C, statements in Octave, SQL statements...
- Programmers deliver the same number of lines of code per day, regardless of the language they are writing in
- Use languages that require few lines of code

# Architectures

- Architecture: Macintosh's desktop
  - Defines type of information, much of the processing
  - Operation is human-centered
  - Software is built around a main event loop
- Metaphor of desktop: its elements should behave as real elements on a desktop

# Heuristics

- Choose components so that each can be implemented independently of the internal implementations of all others
- Programmer productivity in lines of code per day is largely independent of language
- The number of defects remaining undiscovered after a test is proportional to the number of defects found in the test. The constant of proportionality is rarely less than 0.5
- Low delivered defect rate can be only achieved by low defect insertion rate and by layered defect discovery
- Software should be grown or evolved, not built
- The cost of removing a defect rises exponentially with the number of development phases since it was inserted
- Do not fix bugs later; fix them now
- Personnel skill dominates all other factors in productivity and quality