

Impact of Commercial Off-The-Shelf (COTS) Software and Technology on Systems Engineering

**Dorothy McKinney
August, 2001**

Topics

- **Thoughts on Engineering Systems**
- **Using Commercial Off-The-Shelf (COTS) Technology**
- **Observations on Using COTS Software**
- **Impact of COTS on the Interface Between Systems Engineering and Software Engineering**
- **Conclusions**

Engineering Systems

- **The way we engineer systems needs to change to make cost-effective use of new products and technologies; for example:**
 - System requirements definition
 - Allocation of requirements to software
 - Selection of software technologies and off-the-shelf SW and HW
- **We want to leverage new technology, but use lessons learned the hard way over the last 20-30 years on how systems and software engineering can produce successful, supportable systems**
- **This talk addresses the impact of new technologies and the software marketplace on how systems engineering can and should be done**

Lessons Learned Engineering Systems

- **Our understanding of how systems engineering can increase the prospects of software engineering success has evolved, for example:**
 - **Need for all requirements to be defined at the start of the software effort has evolved to identification of areas of uncertainty**
 - spiral and incremental development
 - prototyping, rapid application development, etc.
 - **Software now drives requirements allocation in many software-intensive systems**
- **System architecting of unprecedented systems recognized as a major challenge (first software implementation is probably throw-away*)**

* As discussed in [The Mythical Man-Month](#)

Lessons Learned Engineering Systems (cont'd)

- **Having a Software Operational Concept (how the system will work) as well as a System Operational Concept (how the system will fulfill the customer's needs) significantly increases the chances that the software will work as intended**
- **Having multiple perspectives on the software/system (e.g. hierarchy of components and operational threads) increases likelihood of successful system integration and test**
- **Software involvement in system requirement definition, identification of derived requirements, and requirements allocation to hardware, software and personnel can reduce risks greatly**

Lessons Learned Engineering Systems (cont'd)

- **Plan work based on what you know, plus what you don't know but need to learn or discover**
- **Understand the extent of knowledge about:**
 - **who the stakeholders are**
 - **the customer's needs; needs of other stakeholders**
 - **the customer's constraints**
 - **the application domain**
 - **the system ops concept; the software ops concept**
 - **potentially applicable technology available in the marketplace**
 - **your organization's existing (software) products**
 - **software development environment, tools and methods**
- **Use data to confirm/illuminate this understanding**
- **View every system/software development as incremental**
- **Identify what needs to be discovered in each increment**

Commercial Off-The-Shelf Technology

- **How off-the-shelf software differs from hardware**
- **Software economics**
- **A bit of history**
- **Virtues of COTS**
- **Drawbacks of COTS software**
- **Models for using COTS software**
- **Challenges of using COTS software**
- **Human implications: skills required**
- **Conclusions on use of off-the-shelf software**

Difference Between HW & SW

- **Hardware used to be special-build too**
- **Standards for power consumption, interfaces, etc. were put in place**
- **Catalogs were developed to disseminate information on what parts are available**
- **Hardware distribution allows each step in the value chain to garner revenue**
- **Software does not have the same economics**
 - **Copies of hardware have value proportional to their cost**
 - **Copies of software have virtually no cost**

Software Economics

- **Software can be duplicated/distributed at costs so low that incentives to pay each member of the value chain are ineffective**
- **Make/buy comparisons are skewed by industry problems with inaccurate software cost estimation (optimism of developers)**
- **Economics of using COTS SW changes dramatically when a product's sales model changes (from hundreds of customers to thousands or millions)**
- **Rapid evolution of platforms and operating systems makes versions of a COTS package obsolete quickly**

Using Existing Software, Including Commercial Off-The Shelf SW

- **In the 1970s and 1980s, software reuse was identified as a key strategy for reducing software costs**
 - **Japanese “software factory” approach**
 - **Design (architecting) for reuse**
- **In the 1990s object-oriented software makes integration of separately developed software components more feasible**
- **The software profession has gotten reusable components, but they don’t meet industry criteria for appropriateness for reuse**
 - **Design for reuse “the right way” seems too expensive for commercial software developers, just as is was for system developers**

Virtues of Commercial Off The Shelf

- **Costs are lower than custom development, because product development costs are shared over many users**
- **Many others participate in finding bugs and limitations to the product (and the producer may actually fix these bugs)**
- **You can incorporate new technology more quickly because you use products containing it without having to learn all about it yourself**
- **Development time and risk are avoided when the COTS product provides all the features you need**

Drawbacks of COTS Software

- **Harmonization of a COTS package with**
 - **Platform-specific operating system variants and peripheral drivers**
 - **Operating system version(s)**
 - **Companion COTS SW packages**
- **Feature availability and timing (vaporware)**
- **Bug fixes often only available in later releases**
- **Difficult features dropped in later releases**
- **Features glut can swell resource requirements**
- **Long-term support of COTS software**
 - **Company survival (escrowing source code)**
 - **Evolution of features**

Models of COTS SW Usage

- **Subroutine library model**
- **One “anchor” COTS application model**
 - **Macro language to write your application**
 - **Other applications with built-in bindings to the anchor application**
- **Islands of COTS SW packages with minimal interfaces**
- **Disparate COTS packages as building blocks; glue code used to integrate them**

Challenges of Using COTS SW

- **Using a SW package for what it is worth requires adopting the conceptual framework of the COTS SW package**
 - **May not match either user or developer's ideas about what the system should do or how it should do it**
 - **Documentation often does not explicitly describe this conceptual framework; some developers infer it from installation and operational manuals**
- **This conceptual framework governs:**
 - **How tasks are decomposed in the software**
 - **Interfaces (to user and other software/OS)**
- **Multiple COTS SW packages with different conceptual frameworks may be tough to integrate even if they interface well**

Which Comes First, the Chicken (Requirements) or the Egg (COTS SW)?

- **Conventional system development principles include a purposeful sequence:**
 - **Need identification (problem/opportunity identification)**
 - **Needs analysis**
 - **System requirements definition**
 - **Allocation of functions to hardware, software and personnel**
 - **Derivation of software requirements**
 - **Selection of applicable existing/COTS software**
- **Effective use of COTS software generally requires COTS SW evaluation/assessment starting during needs analysis**
 - **Drives many system requirement definition specifics**
 - **Dictates functional allocation**
 - **Determines derived requirements allocated to COTS software**

Key Skills in Using COTS SW

- Different skills are needed for using COTS SW effectively
- Most COTS SW packages could be designed/implemented better by your good people
- Designing and writing code are more fun for most software engineers than integrating the (sub-optimal) packages available off-the-shelf
- Integration skills are different than development skills
- Problem-resolution is different when you must work around limitations/characteristics of a COTS SW package

Conclusions on Using COTS SW

- In the future, most software development efforts won't be able to afford not to use off-the-shelf software
- Used inappropriately, off-the-shelf software can cost more to use than developing needed software functionality from scratch
- Understanding what it takes to use COTS SW effectively is very important, so you can help your enterprise make the right business decisions

Impact of COTS on the Interface Between Systems Engineering and Software Engineering

- **Key drivers determining the system engineering process**
 - Cost effect of using COTS SW => COTS SW selection precedes requirements definition
 - COTS use => Requirements allocated to COTS are only what COTS products have demonstrated they can do
- **Software-intensive systems versus hardware-intensive systems**
 - Interface definition
 - Hardware design drivers and constraints
 - Selecting/designing software or hardware first
- **Personnel skills needed and division of work**

Conclusions

- **Many new technologies and tools with promise can be found in the marketplace**
- **Changes are needed in our systems engineering approach to requirements analysis and design to take full advantage of available technology & products**
- **Selection for actual use is harder than ever**
 - **Ensure short-term support is sufficient**
 - **Balance long-term concerns**
- **Deliberate selection of your strategy for using externally developed technology and software**
 - **Ensure your staff understands and implements the strategy (and your incentives support its implementation)**
 - **Occasionally review of the effectiveness of your approach**