ECE750-Topic11:
Component-Based Software

COTS-Based Development and Its Cost Estimation

Ladan Tahvildari
Assistant Professor
Dept. of Elect. & Comp. Eng.
University of Waterloo
COTS Definition

- "Commercial Off the Shelf" Software
- Commercial Software Products
  - sold, leased, licensed at advertised prices
- Source Code Unavailable
  - generally an application program interface (API)
  - frequently tailoring options
- Usually periodic releases with feature growth, obsolescence
Related Terms

- **COTS** “Commercial Off the Shelf”
  - Black Box (internal modes not allowed)
  - White Box (internal modes permitted - we treat as NDI)

- **GFE** “Government Furnished Equipment”

- **GOTS** “Government Off the Shelf”

- **NDI** “Non-developmental Item/Not Developed In-house”

- **REUSE Code**
  - source code originally written for some other project
Rationale for Using COTS Products

- Significant change in s/w development practice over past 25 years:
  - building systems with pre-existing software to keep development & maintenance costs as low as possible
  - One such source: COTS

- Rationale for COTS based systems:
  - involve less development time by
    - taking advantage of existing, market proven, vendor supported products,
    - lowering overall development costs
## COTS Advantages and Disadvantages

### Advantages
- Available now; earlier payback
- Avoids expensive development & maintenance
- Predictable license costs & performance
- Rich in functionality
- Broadly used, mature technology
- Frequent upgrades often anticipate organization’s needs
- Dedicated support organization
- Hardware/software independence
- Tracks technology trends

### Disadvantages
- Licensing and intellectual property procurement delays
- Up front license fees
- Recurring maintenance fees
- Reliability often unknown/ inadequate; scale often difficult to change
- Unnecessary features compromise usability, performance
- No control over upgrades/maintenance
- Dependency on vendor
- Integration not always trivial; incompatibilities among vendors
- Synchronizing multiple-vendor upgrades
Caveat to Using COTS Products

- Two main characteristics of COTS:
  - source code not available to developer
  - evolution not under control of developer

- Results in trade-off:
  - development time can be reduced, but often at cost of increased s/w component integration work

- Unique risks associated with COTS:
  - cost of licensing and redistribution rights, royalties
  - effort needed to understand the COTS software
  - pre-integration assessment and evaluation,
  - post-integration certification of compliance with mission critical or safety critical requirements,
  - indemnification against faults or damage caused by vendor supplied components,
  - costs incurred due to incompatibilities with other needed software and/or hardware
When are COTS Products the “Right” Solution?
When are COTS Products the “Right” Solution?

- When they lie at the intersection of the three determinants of feasibility, \textit{and do so demonstrably better than could original code}:

  - \textbf{Technical}
    - ability to supply the desired functionality at the required level of reliability

  - \textbf{Economic}
    - ability to be incorporated and maintained in the new system within the available budget and schedule

  - \textbf{Strategic}
    - ability to meet needs of the system operating environment, including technical, political, and legal considerations
COTS Phenomena, Pitfalls and Practices

- You have no control over a COTS product’s functionality or performance
- Most COTS products are not designed to interoperate with each other
- You have no control over a COTS product’s evolution
- COTS vendor behavior varies widely
Selection of COTS Software Components

- **Functional Requirements**
  - capability offered

- **Performance Requirements**
  - timing & sizing constraints

- **Nonfunctional Requirements**
  - cost/installation/maintenance/reliability
COTS Software Integration Lifecycle

- Qualify COTS Product
- Perform System Requirements
- Administer COTS Software Acquisition
- Prototype the System Including COTS Software
- Fully Integrate COTS s/w and Interface Code
- Test Completed Prototype
COTS Software Integration
Cost Estimation Models

- Loral Federal Systems
  - CotsCost Model & Tool

- USC-CSE
  - COCOTS (Constructive COTS Software Integration Cost Model)
    - http://sunset.usc.edu/research/COCOTS/
COTS Integration Sources of Effort

1. COTS Assessment (pre- and post- commitment)
   - Functionality, performance, interoperability, etc.

2. COTS Tailoring and Tuning
   - Effects of platform, other COTS products

3. Glue Code Development
   - Similar to other COCOMO II estimation

4. Application Volatility Due to COTS
   - COTS volatility, shortfalls, learning curve

5. Added Application V&V Effort
   - COTS option and stress testing
   - Debugging complications, incorrect fixes
COTS Assessment

Initial Filtering Effort

Total Effort = \left( \frac{\text{# COTS Candidates}}{\text{Average Filtering Effort}} \right)

Final Selection Effort

Total Effort = \sum \left( \frac{\text{# COTS Candidates}}{\text{Average Assessment Effort for Attribute in Given Domain}} \right)

Effort/candidate is project-dependent, within domain guidelines
COTS Tailoring and Tuning

\[
\text{Total Effort} = \sum \left( \frac{\text{# COTS Candidates Tailored at Complexity Level}}{\text{Tailoring Complexity Levels}} \right) \left( \frac{\text{Average Effort at Tailoring Complexity Level in Domain}}{i} \right)
\]

- Five tailoring effort complexity levels:
  - Very Low, Low, Nominal, High, Very High
- Differentiated based on number tailored parameters,
  - Difficulty of needed scripts, API iterations, etc.
Glue Code Development and Test

Total Effort = \( A \cdot [(\text{size})(1+\text{breakage})]^B \cdot \Pi \) (effort multipliers)

- **A** - a linear scaling constant
- **Size** - of the glue code in SLOC or FP
- **Breakage** - of the glue code due to change in requirements and/or COTS volatility
- **Effort Multipliers** - 13 parameters, each with settings ranging VL to VH
- **B** - an architectural scale factor with settings VL to VH
Glue Code Development and Test: Glue Code Cost Drivers

Personnel Drivers

1) ACIEP - COTS Integrator Experience with Product
2) ACIPC - COTS Integrator Personnel Capability
3) AXCIP - Integrator Experience with COTS Integration Processes
4) APCON - Integrator Personnel Continuity

COTS Component Drivers

5) ACPMT - COTS Product Maturity
6) ACSEW - COTS Supplier Product Extension Willingness
7) APCPX - COTS Product Interface Complexity
8) ACPPS - COTS Supplier Product Support
9) ACPTD - COTS Supplier Provided Training and Documentation

Application/System Drivers

10) ACREL - Constraints on Application System/Subsystem Reliability
11) AACPX - Application Interface Complexity
12) ACPER - Constraints on COTS Technical Performance
13) ASPRT - Application System Portability

Nonlinear Scale Factor

1) AAREN - Application Architectural Engineering
Increased Application Effort Due to COTS Volatility

Approximate Model:

Total Effort = (Application Effort) \cdot \left[ \frac{BRAK\ COTS}{100} \right] \cdot (EAF)_{COTS}

Detailed Model with COCOMO II Parameters:

Total Effort = (Application Effort) \cdot \left[ \left(1 + \frac{BRAK\ COTS}{1+BRAK} \right)^{1.01 + \sum_{-1}^{1}} \right] \cdot (EAF)_{COTS}

**BRAK COTS**: % application code breakage due to COTS volatility

**BRAK**: % application code breakage otherwise

**S**: COCOMO II scale factor

**EAF**: Effort Adjustment Factor (product of effort multipliers)
Total COTS Integration Cost Estimate

Total Integration Effort (in Person-Months) = Assessment Effort + Tailoring Effort + Glue Code Effort + Volatility Effort

where

Assessment Effort = Filtering Effort + Final Selection Effort

Total integration Cost = (Total Integration Effort) • ($$/Person-Month)
COCOTS: Most Important Aspect

- COCOTS is completely open.

- Regardless of whatever estimates it provides, the descriptions of the elements that have gone into the model help highlight the most important factors that should be of concern to managers and developers of software systems using COTS software components.

- It’s the very essence of a "constructive" cost model:
  - one that helps an estimator better understand the complexities of a given software job to be done
  - by being open permits the estimator to know exactly why a model gives the estimate it does