Added Sources of Costs in Maintaining COTS-based Systems

Betsy Clark, Ph.D.
Brad Clark, Ph.D.
The American Society for Quality (ASQ)
Washington, DC & Maryland Metro Section
Software Special Interest Group (SSIG)
&
Society for Software Quality (SSQ)
Washington, DC Area Chapter
January 22, 2008
Background

• Data-collection interviews (with Dr. Chris Abts) for COCOTS
  – Twenty acquisition projects
  – Interviewed maintenance leads for about a third of these
• Most projects were viewed as reasonably successful in terms of their acquisition but less so in terms of maintainability.
• Unexpected thread through these interviews
  – Cost to maintain a COTS-based system exceeds that of a comparable custom system.
  – Difficulty communicating to management why maintenance is so expensive
• If COTS-based systems really are more costly to maintain:
  – What are these additional costs?
  – Are there strategies for managing or minimizing them?
Outline

Background
Ten Sources of Added Costs
Impact of the Number of COTS Components
Three Risk Mitigation Strategies
Conclusions
What is a COTS Component?

• Sold, leased, or licensed for a fee (which includes vendor support in fixing defects if they are found)
• The source code is unavailable
• The component evolves over time as the vendor provides periodic releases of the product (upgrades) containing fixes and new or enhanced functionality
• Any given version of a COTS component will reach eventual obsolescence or end of life in which it will no longer be supported by the vendor
What Makes Maintenance Different?

• For any given component, either
  – Do nothing
    • Eventually component will reach end of life
  – Upgrade to new version
    • Brings its own set of challenges
Different COTS System Compositions -1

- COTS-Solution Systems (CBS)
- COTS-Intensive Systems (CIS)

- COTS-Solution Systems are systems that are typical business or standard information technology (IT) systems
  - The major COTS component is essentially the system
  - Provides a user interface
  - Has it’s own architecture
  - Has internal business logic that must be followed to be used

Different COTS System Compositions -2

- COTS-Intensive Systems are comprised of many COTS components.
- These are more typically safety- and performance-critical systems
  - No one component is “king”
  - There may be many components that handle
    - The user interface
    - Data transmission, storage, manipulation and transformation
  - Interact with each other through custom-developed “glue code” using vendor-provided application program interfaces (APIs) and with custom-developed application code
  - Business logic is spread across components and is guided by the way the components are used
Surveyed Systems

- The systems that predominantly made up our sample are mission-critical systems with *high reliability and performance requirements* and would be classified as COTS-intensive
  - A number of our projects were air-traffic control systems
  - We also had ground control systems for missile launches and two ground control systems for satellites
  - These systems typically have a large amount of custom application code along with a large number of COTS products (10 to 50 was typical).
Sources of Added Costs

- We identified ten sources of added costs (compared to custom applications).

Licensing  Hardware Upgrades
Evaluation of New Releases  Disabling New Features
Defect Hunting  Early Maintenance
Vendor Support  Market Watch
Upgrade Ripple Effect  Continuous Funding
Licensing

• The most obvious additional cost burden is component licensing fees.
  – Fees can range from a one-time fee to yearly renewal
  – With one exception, licensing fees did not cause concern among the project members interviewed, presumably because this was an expected, known lifecycle cost.
    • The one exception occurred for a COTS-solution system that was used on a pilot basis at one location.
    – There is *effort required in tracking licensing requirements* to ensure that renewals are paid.
      • With different types of licensing and support agreements across different COTS components and vendors, this tracking can become an administrative burden.
• There are no comparable fees or effort for custom developed systems.
Evaluation of New Releases

• **A major source of costs** stems from COTS component volatility.

• In contrast to custom-developed code, a COTS software component is controlled by the vendor.
  – The timing and content of releases is at the discretion of the vendor.
  – Major effort may be required to evaluate and understand the implications of upgrading to a new component or perhaps switching to a whole new product entirely

• Evaluation activities require a test bed that can replicate all deployed system configurations of hardware and software.
  – For safety-critical systems, the amount of analysis can be large even though the ultimate decision may be to do nothing.

• The need for this ongoing black-box evaluation is unique to systems with COTS components.
Defect Hunting

• Defects appear to be more problematic for COTS-intensive systems than with custom code.
• Projects reported that it can be much more difficult with a COTS-based system to pinpoint the source of a problem.
  – It can be difficult to know whether a defect is coming from a COTS component or from other custom developed code.
  – We heard of finger-pointing situations in which a defect was in a COTS product but the vendor was unable to replicate it because they didn’t have the same hardware configuration.
  – Detective work results in added time and effort, translating into additional costs.
• With a custom system, one can see inside the box. Debugging can follow the path through the code without running into component boundaries. This eliminates finger pointing.
Vendor Support

• Support can range from 24/7 call service to dedicated on-site staffing
  – If the latest release of a COTS software component has new features or interfaces, a vendor’s support may be required to integrate the component into the current system

• Custom applications do not have this added expense
Upgrade Ripple Effect

• Due to the new, additional functionality in an upgraded component, the system may require changes to custom code, glue code between components, or tailoring of other COTS components.

• In custom developed code maintenance
  – only the fixes and enhancements that are needed are implemented, thus minimizing (but not eliminating) ripple effects.
Hardware Upgrades

• People found that upgrades to new software components sometimes required upgrades to new hardware as well
  – In one person’s words, “Vendors are constantly driven to add functionality. This puts more demands on the hardware. We haven’t been able to upgrade the hardware as quickly as we’d like.”

• In a comparable custom maintenance upgrade
  – With only the required features implemented, minimal impact to hardware performance can be preserved.
Disabling New Features

• There may be new features that need to be disabled for security or performance reasons.
• The added cost is in the form of additional tailoring of the COTS component.
• This may require discovering how to disable new features or custom code written to hide or disable the new features.
• In custom developed code maintenance
  – Disabling a new feature is not characteristic of custom systems.
Early Maintenance

• Because COTS components continue to evolve in the marketplace, it is possible that upgrades may begin before the system is deployed, particularly if the development spans several years.
• If the components are not upgraded, it is possible that much of the system may have reached end of life before the system is even delivered.
  – This was the case for one of the projects interviewed; this system had an application base totaling more than one million lines of custom code plus a total of 45 COTS components.
  – Almost half of these components were obsolete by the time the system was deployed.
Market Watch

• Because COTS vendors can go out of business, a number of those interviewed suggested that a “market watch” be established as a risk mitigation strategy to handle such an event.

• If a vendor goes out of business, either the component source code or a different component can be purchased.

• In custom developed code maintenance:
  – This activity is not required
Continuous Funding

• Systems with COTS components require a more stable funding base.
  – The consequences of delaying funding with a COTS-based system is that licenses may lapse, bug fixes and upgrades become unavailable or vendors go out of business with no resources to exercise the risk mitigation identified in a Market Watch.
  – *maintenance vs. sustainment*
    • More like hardware in that system will, in effect, degrade without ongoing dollars and effort
• In custom developed code maintenance:
  – Enhancements can be delayed until funding is obtained.
Number of Components versus Maintenance Costs

- One of the major “themes” emerging from the interviews is that maintenance costs increase steeply as the number of components increase
Number of Components versus Maintenance Costs

Cost of Maintenance
Fn (synchronization, complexity of system, no. planned upgrades, etc.)

$ n+x

Volatility effects dominate increased integration experience

n+3

n+2

n+1

Increased integration experience dominates volatility effects

Volatility effects just cancel increased integration experience

No. of COTS in system

1

2

3

4

5

Retire

Maintain

Source: Abts 2002
Number of Components versus Maintenance Costs

• As the number of components increase:
  – COTS *licensing* is much more complex with more components
  – *Evaluating the impact of upgrades* is considerably more burdensome if there are a lot of components
    • The number of possible interactions between components increases exponentially as the number of components increases.
  – When *trying to hunt down defects*, the complex interactions of many components make the task even more difficult.
  – *Configuration management* becomes more complex when many components and configurations exist in a system.
  – The possibility of a *ripple effect* is higher with the impact of component upgrades
  – The *market watch* becomes a large-scale activity.
Three Risk Mitigation Strategies to Deal with COTS Maintenance

• Across the projects interviewed, three strategies for dealing with COTS maintenance were observed:
  – Revert back to source code
  – Divide and conquer
  – Design for change
Revert Back to Source Code

• Applied to “critical” COTS components.
  – In one case, the product (an operating system) was allowed to reach end of life and the project purchased the source code from the vendor. Avoided the necessity for (special-purpose) hardware upgrades.
  – Several other projects replaced critical COTS components with their own custom-developed software.

• Advantage: Places control for fixing problems back in the hands of the maintenance organization. Removes the risk of being unable to fix future problems.

• Disadvantage: Additional expense in purchasing the source code or developing it from scratch.
Divide and Conquer

• Divides the COTS software components into two categories: non-critical and critical.
  – The non-critical COTS components are not upgraded. Resources are focused on the set of critical components.
  – For critical COTS components, there are market watch and evaluation activities. The decision to upgrade is made individually for each critical component.

• This strategy is driven by the need to balance the ongoing costs required for maintaining a COTS-intensive system with limited resources.

• Advantage: It saves money by ignoring a subset of components.

• Disadvantage: A portion of the system remains stagnant and unsupported.
Design for Change

• Uses “information hiding” in the form of wrappers to protect the system from unintended negative impacts of multiple component upgrades
  – As one person said in describing this strategy, “we wanted to be able to replace a product without damage to the rest of the system. As an example, we have a wrapper around the database. It could be a flat file or relational database – the custom application doesn’t care.”

• The project in our sample that used this strategy had a strong project sponsor right from the beginning who argued successfully for additional resources to design for change from the beginning.
  – This was a project that was planned for a long life with safety-critical requirements.

• Advantage: more assurance against unintended ripple effects
• Disadvantage: the necessity for resources early in system development
Conclusions

• We don’t want to leave the impression that we are against the use of COTS components
  – Given the complexity of many of today’s systems, total custom development is no longer feasible
  – Allows system developers to take advantage of the best that the marketplace has to offer
  – Removes the unnecessary “reinvention of the wheel” seen prior to the widespread use of COTS components

• Our objective is to help people anticipate and manage the added sources of costs in maintaining COTS-intensive systems.
  – Projects should understand the life-cycle implications of integrating a large number of COTS components
  – More thought should be given early to:
    – the impact of upgrades on the entire system
    – the reliance on vendors to fix problems
    – the strategies that will be used in dealing with multiple products, each evolving at the discretion of the vendor.
References

Presentation Source:

Abts 2002:
Related Papers and Contact Information


Available from:
Betsy Clark
Brad Clark
(703) 754-0115

Betsy@software metrics.com
Brad@software-metrics.com