Carnegie Mellon University

COTS Impact On Software Development Life Cycles

Managing Software Development – Final Report

John Peabody
8/8/2010

Table of Contents

Executive Summary 3
1 Introduction 4
2 Motivation 4
3 COTS vs Open Source 5
3.1 COTS 5
3.2 Open Source 5
3.3 Trade Offs 5
4 COTS Influence on Development Activities 6
4.1 Volatile Nature of COTS 6
4.2 Art of Integration 6
4.3 Development Activities 7
4.3.1 System Context 7
4.3.2 Architecture and Design 8
4.3.3 Construction 8
4.3.4 Configuration Management 8
4.3.5 Deployment and Support 9
4.3.6 Risk Management 9
5 Evaluation Methods 9
5.1 EPIC 9
5.1.1 Inception Phase 10
5.1.2 Elaboration Phase 10
5.1.3 Construction Phase 10
5.1.4 Transition Phase 10
5.2 PECA 10
5.2.1 Planning the Evaluation 11
5.2.2 Establishing the Criteria 11
5.2.3 Collecting the Data 11
5.2.4 Analyzing the Data 12
5.3 CURE 12
6 Leveraging Evaluation Methods within Development Lifecycles 12

6.1.1 Iterative Life Cycle 12
6.1.2 Utilization of COTS Evaluation Methods 13
7 Summary 13
8 References: 14

Executive Summary

The utilization of commercial off-the-shelf (COTS) components is increasing within many organizations. The idea being that delivery dates, effort, cost, and the amount of custom development can all be reduced by integrating COTS components into a system. The issue however, is that many organizations lack a process oriented evaluation method for selecting off-the-shelf components they will benefit from. The result can be the selection of a component that increases development overhead rather than reducing it.

A handful of COTS component evaluation methods exist, but integrating these methods within an organizations development life cycle can be challenging. The paradigm shift from complete custom developed systems to COTS-based systems "requires new understandings about the COTS marketplace and how all engineering, business, and management activities must work together harmoniously to accommodate it." [Oberndorf 00] This report provides a high level overview of three existing COTS evaluation methods:

- The Evolutionary Process for Integrating COTS-Based Systems (EPIC)
- The PECA (Plan, Establish, Collect, Analyze) Process
- The COTS Usage Risk Evaluation (CURE) Method

It then provides suggestions on merging these evaluation techniques into an organizations development lifecycle. Any development life cycle that is supporting a COTS-based system must be iterative or cyclic in nature. This is driven by the following two key reasons:

- The COTS market is volatile in nature.
- The inter-dependencies between the system context, system architecture, and COTS component selection must be overcome.

COTS-based systems are increasing in popularity, yet methods to ensure the successful development of these systems are difficult to incorporate within the development life cycles of many organizations. While methods exist it is both the lack of knowledge as well as the lack of understanding that COTS-based systems require changes in process, culture, and techniques in order to mitigate the unique risks that arise when leveraging off-the-shelf components.

1 Introduction

In order to maintain a competitive edge in the technology driven, fast paced environment of today, many organizations are increasingly utilizing commercial off-the-shelf (COTS) components within their systems. The idea being that delivery dates, effort, cost, and the amount of custom development can all be reduced by integrating COTS components into a system. The issue however, is that many organizations lack a process oriented evaluation method for selecting off-the-shelf components they will benefit from. The result can be the selection of a component that increases development overhead rather than reducing it.

Though a handful of COTS component evaluation methods exist, integrating these methods within an organizations development life cycle can be challenging. The paradigm shift from complete custom developed systems to COTS-based systems "requires new understandings about the COTS marketplace and how all engineering, business, and management activities must work together harmoniously to accommodate it." [Oberndorf 00] This report explores a few of the existing COTS evaluation methods and provides suggestions on merging these evaluation techniques into an organizations development lifecycle.

The structure for the remainder of this paper is as follows:

- In Section 2, I provide the motivation for this report.
- In Section 3, I provide a definition of COTS and Open Source software to establish the differences, yet allow them to be equivalent in terms of the remaining sections.
- In Section 4, I describe the paradigm shift within development activities caused by utilizing COTS components within a system.
- In Section 5, I describe three COTS evaluation methods that currently exist.
- In Section 6, I make a recommendation for supporting COTS-based system design within an organizations software development life cycle.
- In Section 7, I summarize this report.

2 Motivation

The motivation for this report stems from my current employment at a Federally Funded Research and Development Center. This organization frequently integrates COTS components into legacy systems as well as systems currently under design and development. In addition to utilizing COTS components the organization develops an internal version of Government off-the-shelf (GOTS) software and hardware components that are integrated into systems throughout the organization as well as transitioned to industry. While COTS and GOTS components are frequently utilized I have never witnessed the use of any formal COTS evaluation techniques within our development life cycles, nor have I witnessed concern with generating artifacts necessary to support the evaluation of our internally developed GOTS components. It is this lack of knowledge regarding COTS evaluation that motivates the research within the report.

3 COTS vs Open Source

Today's off-the-shelf market place is flooded with software components for organizations to select from when developing a system. Understanding the fundamental differences between COTS and open-source components is the first step in selecting off-the-self components. By defining the two, the tradeoffs become apparent yet the decision on which is the right fit for an organization to leverage is based on business drivers within the organization. Either solution however, will need to be supported by an evaluation process that supports the organizations development life cycle.

3.1 COTS

According to "An Activity Framework for COTS-Based Systems" by Oberndorf, Brownsword, and Sledge, COTS is defined as a product [Oberndorf 00]:

- sold, leased, or licensed to the general public
- offered by a vender trying to profit from it
- supported and evolved by the vender, who retains the intellectual property rights
- available in multiple, identical copies
- used without modification of the internals

3.2 Open Source

At a high level, Open Source components are components in which the source code is made available for viewing and modifying and can distributed freely or for a fee [Hissam 01]. However the Open Source Initiative (OSI) provides a more in-depth definition that defines Open Source Software as something much more than components distributed with modifiable source code. According to OSI Open Source software (OSS) must allow developers to use, as is or modified, the OSS as a component within their system without paying the developer of the OSS. [OSI 10] In other words, use and ownership of the software is truly open and unrestricted.

3.3 Trade Offs

The tradeoff between COTS and OSS comes down to cost, support, and modifiability. Due to the fact the OSS is free; there is no support center that the organization can contact when issues arise. The time and cost to resolve these issues then falls back on the organization. However, if changes within the component are required such as: architectural, interface, or feature modifications; the source code is available and can be freely modified to meet the needs of the organization. COTS components have reduced modifiability because the source code is normally not provided however; the cost associated with COTS covers the support provided by the vendor. By having a vendor responsible for initial development and continued support, the organization has more resources available to focus on the intellectual property of their product, which in most cases, is the marketable component of the system being developed.

While choosing between COTS and OSS components is a business decision an organization will face if they decide to utilize off-the-shelf components, they both require a means of evaluation that can be incorporated into the organizations development life cycle.

4 COTS Influence on Development Activities

Traditional custom development can be thought of as a linear progression through the various phases of the development life cycle. In some development methods, even when iteration is introduced, it simply iterates over an individual phase until the given phase has reached completion. For COTS-based systems however, this linear progression is insufficient.

As seen in Figure 1 below, a shift in paradigm from the traditional linear development approach to a simultaneous approach is required to support COTS-based development. [Oberndorf 00] This paradigm shift is caused by the volatile nature of COTS components and the art of integration.

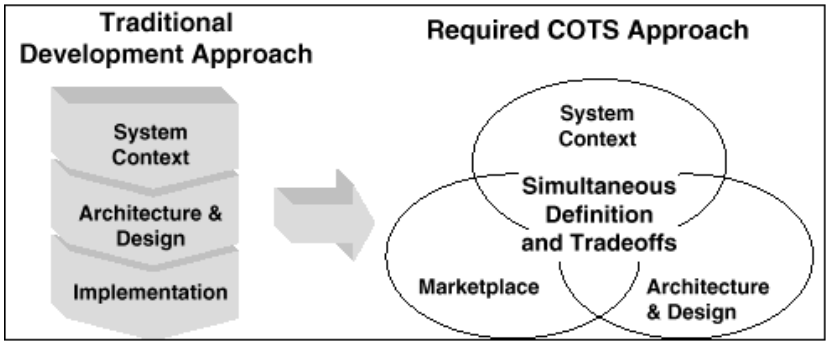

Figure 1: Traditional Versus COTS-Based Approach

4.1 Volatile Nature of COTS

An organization looking to compose a system of COTS components must quickly realize that the components are targeted at a market, not their specific system. COTS component developers are in business to turn a profit by developing a solution for a wide variety of customers. It is this market that influences the development of upgrades, the architectural layout, and release strategy for the COTS component; not the demands of the organization. This volatile nature is best described by eight inherent characteristics surrounding COTS development [Oberndorf 00]:

- COTS products and the marketplace change frequently and continuously.
- COTS products are driven by the marketplace, not one system's need.
- Products have built-in assumptions about how they will be used.
- Licensing and data rights are involved.
- Programs have limited control of the frequency or content of COTS releases.
- Programs have limited visibility into COTS product source code and behavior.
- Products are built on architectural assumptions that may vary across system components.
- COTS products will have interdependencies.

4.2 Art of Integration

Due to the volatile nature of COTS components, the development process that supports their utilization must account for this volatility. As seen in Figure 1 above, the selection of COTS components must be balanced with knowledge of the system architecture and design, but also with

knowledge of the system context and requirements. By understanding the system context and requirements, readily available COTS components that meet the requirements can be identified. However as the available COTS components are being identified, an architecture that supports their interaction must be conceptualized while meeting the system context and requirements. It is this "act of composition and reconciliation" that shifts COTS-based system development into an art of integration, rather than an "act of creation". [Oberndorf 00]

4.3 Development Activities

Figure 2 below depicts the numerous development activities affected by COTS integration. Many of these activities occur during the development of non COTS-based systems however, aspects of these activities must be modified in order to handle COTS specific issues. In addition, a few new activies are added to handle issues that are again specific to COTS integration. Activities of particular interest are those within the engineering activity area and COTS-based systems risk management. A few examples are presented in the following sections of how these activites are impacted by COTS component utilization. [Oberndorf 00]

Figure 2: COTS-Based Systems Activity Areas

4.3.1 System Context

System context is an example of an activity that is present in both custom system and COTS-based system development. In custom systems the system context contains the requirements of the system, business drivers, and any constraints that are being placed on the system. Utilizing the foundation established within the system context, custom system development uses this information to drive the remaining development activities; for COTS-based systems however, this does not entirely hold true. While the system context still provides the same critical information, it is no longer the single

building block driving development activities. Requirements provided by the system context must be met by available COTS components which in turn must be able to function together within the system architecture. Due to the dependencies, normal requirements elicitation activities must consider the architecture definition as well as the COTS component selection. [Oberndorf 00]

4.3.2 Architecture and Design

The goal of the architecture and design activity defines the composition of the system. This composition describes the components within the system, the interfaces between these components, as well as the structure and relationship between them. With custom system development the architecture and design can be completed prior to implementation and used as a roadmap for the implementation and integration of the system. In COTS-based systems, the architecture and design is again dependent on the system context and the selected COTS components.

In addition, the architecture and design of a COTS-based system is significantly impacted by the nature of the COTS market. The fact is that an organization has no control over changes made within COTS component releases, the scheduling of these releases, or the longevity of COTS components. To cope with this the system architecture must be flexible enough to support ever changing components while retaining the ability to meet the system requirements. Creating an architecture that can change overtime as the COTS market changes over time is a major factor in the success of COTS-based system development. [Oberndorf 00]

4.3.3 Construction

As described in section 4.2, COTS-based system development is more an art of integration than creativity. This becomes apparent in the construction activity. The focus shifts from generating custom code to meet the system requirements that conforms with the designed architecture to constructing an architecture that integrates the selected COTS components into a system that meets the requirements. [Oberndorf 00]

Other activities within the development cycle start to arise during the construction of COTS-based systems that occur outside construction of custom systems. For instance, maintenance activities may be required during construction due to unforeseen product updates for selected COTS components. Testing also needs to be adjusted to a "black-box" approach to support the closed nature of COTS development. [Oberndorf 00]

4.3.4 Configuration Management

While configuration management can be a challenge in custom development system, the challenge is only increased by factors associated with COTS components. Configuration management for COTS components does not start at the development phase, but as soon as evaluation of COTS components begin. Performing configuration management at evaluation time is necessary for COTS components to ensure information like version numbers, applied patches, and other component dependencies are managed. This level of configuration management should be maintained

throughout the life of a system to ensure compatibility between COTS components and overall system functionality. [Oberndorf 00]

4.3.5 Deployment and Support

The deployment and support of a COTS-based system is composed of the activities required to support and maintain a system post deployment. As mentioned in Section 4.3.3 however, it is common for COTS components to require patches and updates prior to the system even reaching the construction phase. There are also instances were construction activities will be required after a system has been deployed, normally the support phase. For example, if a COTS component reaches the end of life and is no longer supported, a newly identified component may need to be integrated into the system to extend the life of the system. This crossover between the two activities blurs the lines between construction and support activities, which are normally clearly two independent activities. [Oberndorf 00]

4.3.6 Risk Management

While the process for performing risk management within a COTS-based system remains the same as a custom developed system, it is worth noting that COTS-based systems face different risks. New risks within COTS-based systems arise from issues such as vender relationships, marketplace developments, licensing and component releases, etc. [Oberndorf 00] The COTS Usage Risk Evaluation was developed to assist in identifying risks associated with COTS-based systems and will be described in further details in Section 5.3.

5 Evaluation Methods

A number of various methods have been established to aid in the development of COTS-based systems. These methods are aimed at specific activities such as integration of COTS-based systems, product evaluation, and risk evaluation. This section provides a high level overview of three such methods.

5.1 EPIC

The Evolutionary Process for Integrating COTS-Based Systems (EPIC) is a process "to help organizations build, field, and support solutions based on COTS and other pre-existing components." [Albert 02] EPIC is an iterative process in which, similar to the Rational Unified Process, is composed of four phases:

- Inception
- Elaboration
- Construction
- Transition

A quick overview of these four phases is provided in the following sections, with more details available within "Evolutionary Process for Integrating COTS-Based Systems (EPIC): An Overview." [Albert 02]

5.1.1 Inception Phase

The purpose of the inception phase is to define the system context. The system requirements, constraints, and expectations are elicited from the system stakeholders and used to perform a quick evaluation of available COTS components that meet these requirements. In addition, the cost and schedule for the system are defined. The results of the inception phase should be at least one feasible, solution to evaluate. A summary of the possible solutions is created in which solutions that merit further examination are identified. [Albert 02]

5.1.2 Elaboration Phase

Within the elaboration phase, the goal is to expand the level of knowledge associated with the feasible solutions identified in the inception phase. Through further definition of the system context, experimentation, and prototyping the system architecture begins to take shape as well as the integration of COTS components. By the end of the elaboration phase a single system solution should be selected as the baseline for the construction phase. [Albert 02]

5.1.3 Construction Phase

The construction phase aims at creating a version of the system solution that is of production quality. This includes the generation of custom components, interfaces between components, and the definition of any tests that need to be performed. Also created during the construction phase is any documentation needed to support the COTS-based system. By creating this production level release, system stakeholders can verify that the system meets the requirements and is ready for deployment to the customers. [Albert 02]

5.1.4 Transition Phase

The transition phase moves the system to the user community. The user community is made aware that the system has been released and customer support begins. Bug fixing, patches, and feature updates are to be expected as users begin to utilize the system. An important aspect of the transition phase is continuous monitoring of the market place. As COTS components change over time so too must the system. The organization must be aware of these changes and be able to support these changes within the deployed systems. [Albert 02]

5.2 PECA

While the EPIC method provides a complete COTS-based system integration process, the PECA or Plan, Establish, Collect, Analyze process provides a process that "helps organizations make carefully reasoned and sound product decisions." [Comella-Dorda 03] The PECA process also is composed of four activities:

- Planning the evaluation
- Establishing the criteria
- Collecting the data
- Analyzing the data

5.2.1 Planning the Evaluation

In planning for the evaluation process there are a handful of key decisions that must be made. An evaluation team must first be selected. The PECA process recommends a team that represents a diverse makeup of the organization. This includes a balance of power, as well as representation from different departments. The hope in creating a diverse evaluation team is that the opportunity for bias is reduced by the balance created by the diversity. Once an evaluation team has been selected a charter must be formed; establishing the goals, scope, and factors that are to be used during the evaluation. Without a formalized charter the evaluation process may proceed without guidance and become unmanageable. The other key decision that must be made in planning the evaluation is the selection of the approach to leverage during the evaluation. Proper criteria must be established according to the criticality of the component being selected [Comella-Dorda 03]:

- depth of the evaluation
- first fit vs. best fit
- Component filtering

5.2.2 Establishing the Criteria

Establishing the criteria effectively maps the systems requirements to a prioritized set of evaluation criteria. This mapping occurs by performing the following steps [Comella-Dorda 03]:

- Define the evaluation requirements.
- Define the evaluation criteria.
- Prioritize the criteria.

Evaluation requirements will stem from both system requirements and system context. Non-functional requires such as security and performance requirements are system requirements that will help guide the evaluation while context requirements like architecture, operational environment, and programming constraints are system context requirements that will feed into the evaluation.

Defining the evaluation criteria creates a means by which to verify that a COTS component can meet an evaluation requirement. "A good criterion needs both the capability statement and a measurement method." [Comella-Dorda 03] In other words, a metric is established that can be used to verify that a COTS component performs to the specifications described by the evaluation requirements and therefore, will meet the need of the system. These evaluation criteria are then prioritized based in the impact they have on the system as a whole.

5.2.3 Collecting the Data

The motivation behind collecting evaluation data on a particular COTS component is to verify it meets expectations, advertised capabilities, and system requirements. A variety of methods exist for performing this data collection from vender appraisals and literature research to constructing complete test beds and prototyping. Independent of the data collection method the goal is the same; collect the metrics that verify the COTS component meets the criteria and requirements needed to provide functionality to the system. [Comella-Dorda 03]

5.2.4 Analyzing the Data

After collecting data from various COTS components this data must be analyzed so that a selection can be made. PECA provides various techniques for both the consolidation and analysis of the collected data. Upon consolidating and analyzing the data recommendations can be made. What the analysis provides is the tradeoff within the recommendations. This trade space may span both the system requirements and criteria, but will provide adequate information to make a selection. [Comella-Dorda 03]

5.3 CURE

The COTS Usage Risk Evaluation (CURE) method is a risk evaluation method aimed at identifying risks related specifically related to COTS component. Following the basic method of the Software Risk Evaluation (SRE) technique, CURE gathers data utilizing a questionnaire followed by conducting an interview based on a discussion document. The questionnaire provides the evaluation team with insight into the system being developed that allows the team to focus the discussion topics to enable more effective risk identification. The discussion document contains fifteen chapters of discussion topics that cover everything from the system description to the maintenance and sustainment of the system. The selection of topics to cover can be modified by the evaluation team based on feedback from the questionnaire, providing system specific data that should reveal potential risk areas. [Carney 03]

This data is then analyzed to identify risk factors and risk conditions that complete condition-consequence statement templates. These condition-consequence statements identify a set of risks that are specific to COTS utilization and could have a significant impact within the system. The result of this analysis are areas in which the development team must be caution of potential risks that have yet to be identified as well as risk areas where the development team has already been made aware of the potential risks and possibly has mitigated these risks.[Carney 03]

6 Leveraging Evaluation Methods within Development Lifecycles

As described in the previous sections COTS-based systems are faced with unique challenges when compared to the traditional development methods of custom systems. In order to overcome these challenges development teams must integrate COTS evaluation, integration, and risk management techniques into their development life cycles. The following sections are my recommendations for integrating these activities into an organizations development life cycle.

6.1.1 Iterative Life Cycle

Any development life cycle that is supporting a COTS-based system must be iterative or cyclic in nature. This is driven by the following two key reasons:

- The COTS market is volatile in nature.
- The inter-dependencies between the system context, system architecture, and COTS component selection must be overcome.

Due to the fact that an organization has no control over any aspect of a COTS component, they are at the mercy of vender. Only the vender can implement bug fixes and develop patches when issues are found within a component. The vender also controls when these fixes are released as well as when updates are released. Responding to changes related to currently utilized COTS components can only be supported by an iterative development life cycle.

However, the most significant risk organizations face that is the result of the volatility of the COTS market is a component reaching end of life or a better component being released. The flexibility provided by an iterative development life cycle supports the evaluation of emerging technologies while a system is deployed such that when a component reaches the end of life a replacement solution should have previously been identified.

6.1.2 Utilization of COTS Evaluation Methods

Much like other methods and processes for software development the COTS evaluation methods have situations in which they are most effective. This section provides my recommendations of when to utilize the methods described in Section 5.

The EPIC method provides a complete process for integrating multiple COTS components together with existing components in order to develop and deploy a functional system. Due to the fact that EPIC is a complete life cycle process, it should be utilized in larger scale systems in which multiple COTS components must be integrated. Given the overhead involved it does not seem effective to utilize EPIC when developing a system when one or two COTS components will be utilized.

The PECA method provides an evaluation method that can be tailored to the size of the development system and its flexibility conforms to an iterative life cycle. I recommend utilizing this method on any development effort that plans to utilize COTS components. Without properly evaluating COTS components for conformance to system requirements and criteria, a COTS-based system is on track for failure.

In order for system development projects to be successful risk management must be performed to mitigate the negative effects of the unknown. These negative effects are only amplified within COTS-based systems again due to the volatile nature for COTS components. It is because of this that CURE, or some customized version be utilized when developing COTS-based systems.

7 Summary

In summary, COTS-based systems are increasing in popularity, yet methods to ensure the successful development of these systems are difficult to incorporate within the development life cycles of many organizations. While methods exist it is both the lack of knowledge as well as the lack of understanding that COTS-based systems require changes in process, culture, and techniques in order to mitigate the unique risks that arise when leveraging off-the-shelf components.

8 References:

[Albert 02] Albert, Cecillia; & Brownsword, Lisa. **Evolutionary Process for Integrating COTS-Based Systems (EPIC): An Overview** (CMU/SEI-2002-TR-009). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.

[Anderson 07] Anderson, William; Morris, Ed; Smith, Dennis; & Ward, Mary C. **COTS and Reusable Software Management Planning: A Template for Life-Cycle Management** (CMU/SEI-2007-TR-011). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2007.

[Carney 03] Carney, David J.; Morris, Edwin J.; Place, Patrick R. H. **Identifying Commercial Off-the-Shelf (COTS) Product Risks: The COTS Usage Risk Evaluation** (CMU/SEI-2003-TR-023). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.

[Clapp 98] Clapp, Judith A.; & Taub, Audrey E. **A Management Guide to Software Maintenance in COTS-Based Systems**. Bedford, MA: Center for Air Force C2 Systems, MITRE, 1998.

[Comella-Dorda 03] Comella-Dorda, Santiago; Dean, John; Lewis, Grace; Morris, Edwin; Oberndorf, Patricia; & Harper, Erin. **A Process for COTS Software Product Evaluation** (CMU/SEI-2003-TR-017, ADA443491). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.

[Hissam 01] Hissam, Scott; Weinstock, Charles B.; Plakosh, Daniel; Asundi, Jayatirtha. **Perspectives on Open Source Software** (CMU/SEI-2001-TR-019). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.

[Linger 02] Linger, Richard C.; Lipson, Howard F.; McHugh, John; Mead, Nancy R.; Sledge, Carol A. **Life-Cycle Models for Survivable Systems** (CMU/SEI-2002-TR-026). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.

[Lipson 01] Lipson, Howard F.; Mead, Nancy R.; & Moore, Andrew P. **Can We Ever Build Survivable Systems from COTS Components?** (CMU/SEI-2001-TN-030). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.

[Miller 00] Miller, Ann. **COTS Software Supplier Identification and Evaluation**. Rolla, MO: Department of Electrical and Computer Engineering, University of Missouri – Rolla, 2000

[Oberndorf 00] Oberndorf, Trivia; Brownsword, Lisa; & Sledge, Carol A. **An Activity Framework for COTS-Based Systems** (CMU/SEI-2000-TR-010, ADA383836). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000.

[OSI 10] Open Source Initiative. "The Open Source Definition Version 1.9" [online]. http://www.opensource.org/osd.html (2010).