Definition and classification of COTS: a proposal

Maurizio Morisio[1], Marco Torchiano[2]

[1]Dip.Automatica e Informatica, Politecnico di Torino, C.so Duca degli Abuzzi, 24, I-10129 Torino, Italy Fax: +39 011 564 7099 Phone: +39 011 564 7033
morisio@polito.it
[2]Computer and Information Science Department (IDI) Norwegian Univ of. Science and Technology (NTNU) O. S. Bragstads plass 2B, N-7491 Trondheim, Norway, Phone: +47 7359 4485 Fax: +47 7359 4466
Marco.Torchiano@idi.ntnu.no

Abstract. COTS based development impacts several issues in software development. New techniques have been proposed, or existing ones have been adapted. Several approaches have been proposed for effort and size estimation, product selection, and architectural mismatches identification. But a fundamental question must be clarified before: what is a COTS product? According to the literature a COTS seems to be anything from an operating system to a UI widget. It appears obvious that a finer level of granularity is required if we want to acquire a deeper insight in COTS related issues. This paper proposes a COTS classification scheme, which is as inclusive as possible. It is intended to provide both researchers and practitioners a tool to characterize more precisely their work. The next research step will be validating, by speculation first and empirically later, the influence of COTS classes on issues in COTS based development.

1 Introduction

In the last decade the use of Commercial Off-The-Shelf (COTS) products as parts of larger systems has grown steadily. The recent Open Source Software (OSS) tide adds an important new feature in the COTS market. Now it becomes more and more common to be able to find a COTS OSS product suitable for a project.

Using one or more COTS products has effects on nearly all activities and products of the software process: architecture and design, effort and cost estimation, validation and testing, and reliability. A growing body of research is dedicated to explore these areas. However, both in research papers and in informal discussions, one question remains un-answered: what is a COTS product? The definitions found in the literature are usually very broad, covering a large variety of products. As a result, researchers and practitioners use the same word with different meanings. Some of these definitions are discussed in section 2 Existing COTS definitions.

We argue that COTS has to remain a term with broad coverage. But inside this class of software products a number of subclasses have to be identified. A recent paper [2] confirms this view, stating that assessment and tailoring efforts vary significantly by COTS product classes.

From a syntactical perspective, the acronym COTS is an adjective, thus it should be used together with a noun; for the sake of readability we will sometimes use the acronym by itself as a noun, in these cases COTS should be read as COTS product.

This paper presents the findings of the first phase of our research. Our research plan is the following:

Phase 1 – General-purpose classification framework

- Identify key, recurring attributes of COTS products, using a broad definition of COTS (basically, product not developed by the developer of the final system). Here the research method is a literature search[21] and the use of any formal or informal knowledge available.
- Identify a set of attributes to describe COTS and structure them.
- Select a number of COTS products, characterize them under the attributes identified. Analyze the resulting clustering, define classes of COTS products.

Phase 2 – COTS definition

- Review, if needed, the definition of COTS to be used in the study.

Phase 3 – Links with the Software Process

- Using the COTS classes identified in Phase 1, and any available knowledge, state hypothesis of relationships between COTS classes and activities, products, attributes of COTS based software processes (e.g. cost models, selection methods, architectures, testing and validation techniques, etc.).

Phase 4

- Validate empirically the hypothesis defined in Phase 3

This paper presents initial results of Phase 1 of the research. It is organized as follows:

Section 2: survey of existing COTS definitions

Section 3: proposal of a characterization framework

Section 4. application of the characterization framework

2 Existing COTS definitions

In this section we present a survey of proposals available in the literature, directly or indirectly linked to the problem of classification and definition of COTS products.

We divided the literature related to COTS products into four parts: (1) definitions of COTS products, (2) classification of COTS-based systems, (3) attributes of COTS products, and (4) comparison of COTS products and components.

2.1 COTS Definitions

Oberndorf

In [11] the term COTS product is defined on the basis of the ‘Federal Acquisition Regulations’. It is defined as something that one can buy, ready-made, from some manufacturer's virtual store shelf (e.g., through a catalogue or from a price list). It carries with it a sense of getting, at a reasonable cost, something that already does the

job. The main characteristics of COTS are: (1) it exists a priori, (2) it is available to the general public or (3) it can be bought (or leased or licensed).

The meaning of the term "commercial" is a product customarily used for general purposes and has been sold, leased, or licensed (or offered for sale, lease or license) to the general public. As for the term "off-the-shelf", it can mean that the item is not to be developed by the user, but already exists.

Vidger

The work of Vidger and colleagues, presented in [14] and [15], provides a different definition of COTS products. They are pre-existing software products; sold in many copies with minimal changes; whose customers have no control over specification, schedule and evolution; access to source code as well as internal documentation is usually unavailable; complete and correct behavioral specifications are not available

SEI

According to the perspective of the SEI, presented in a recent work [5], a COTS product is: sold, leased, or licensed to the general public; offered by a vendor trying to profit from it; supported and evolved by the vendor, who retains the intellectual property rights; available in multiple, identical copies; and used without source code modification.

Basili and Boehm

Recently Basili and Boehm [2] proposed another definition of COTS. According to their definition, COTS software has the following characteristics: (1) the buyer has no access to the source code, (2) the vendor controls its development, and (3) it has a nontrivial installed base (that is, more than one customer; more than a few copies). This definition does not include some kind of products like special purpose software, special version of commercial software, and open source software.

The category of products addressed by such definition presents some specific non-technical problems, related to the quick turnaround (every 8-9 month) [2] of product releases. In addition, marketplace consideration adds further variability: in the COTS products market there are no widely agreed upon standards[16] mainly due to marketing strategies aimed at obtaining vendor lock-in. Variability and marketing strategies suggest that there will never be a single unified marketplace of standardized COTS products [18].

2.2 COTS-based Systems

Carney

In [6] Carney takes the point of view of the delivered system, instead of the part: he identifies three types of COTS systems as a function of the number of COTS used and their influence on the final system: turnkey systems are built around a (suite of) commercial product(s); intermediate systems are built around one COTS but integrate other components; integrated systems are built by integrating several COTS, all on the same level of importance.

Wallnau et al.

A similar classification of COTS-based systems is proposed in [17], with the concepts of COTS-solution systems (one substantial product (suite) is tailored to

provide a “turnkey” solution) and COTS-intensive systems (many integrated products provide system functionality).

2.3 COTS products attributes

Carney and Long

Carney and Long [7] propose two attributes to characterize COTS, origin and modifiability, and provide some examples.

There is no discussion of cost and property issues, which seems to be sometimes mixed with the origin axis, while in our opinion it should be discussed separately. No distinction can be found between what needs to be modified in order to make a product work and what can be modified in order to better integrate it into the delivered system.

COCOTS

A classification of COTS products could be derived from the COCOTS models [1]. Some cost drivers could be used to identify COTS products categories: product maturity, supplier willingness to extend product, product interface complexity, supplier product support, supplier provided training and documentation. Most of these attributes are related to the supplier and market conditions and not to technology.

Yakimovich

Several researches addressed the integration problem of COTS products; in particular the work by Yakimovich et al. [20] proposes a set of criteria for classifying software architectures in order to estimate the integration effort. The same characteristics are used to classify both the components and the systems.

Egyed et al.

A methodology for evaluating the architectural impact of software components is proposed in [8]. Such a method allows the selection of both the components and of a suitable architectural style. The key point is the identification of architectural mismatches.

2.4 Components

Component is a term now widely used, and probably as ambiguous as COTS. The relationship with COTS is strong, but COTS and components should be considered as two different concepts.

A lot of definitions of component can be found in the literature. A simple and compact definition is the following: “binary units of independent production, acquisition and deployment” [13]. But also looser definitions can be found: “a physical, replaceable part of a system that packages implementation and provides the realization of a set of interfaces. A component represents a physical piece of implementation of a system, including software code (source, binary or executable) or equivalents such as scripts or command files” [12].

In summary we can say that COTS products and components are two sets with a non-empty intersection but both need a neater definition.

3 COTS characterization framework

In Table 1 we propose a characterization framework for COTS. We propose a number of attributes and possible values to characterize a COTS product. A COTS product is described by a single value on each attribute. We designed the attribute framework so that each attribute has only one value. Multiple values or "don't care" are not possible. Different COTS products with the same set of values belong to the same class.

The attributes we propose are either already defined in literature, or new. The contribution we want to make is both in organizing the existing attributes in a consistent framework, and in proposing new ones. The attributes are grouped into four categories:

- Source: where the product comes from
- Customization: how much the product can or should be customized
- Bundle: in what form the component is delivered, both to the integrator and to the customer of the system
- Role: what is the intrinsic role the product can assume in the final system

All of the attributes we propose are of ordinal type, except those in the role category, which are of nominal type.

Table 1. COTS characterization attributes.

Category	**Attribute**	**Possible Values**
Source	Origin (from [7])	In-house < existing external < externally developed < special version of commercial < independent commercial
	Cost & Property	Acquisition < license < free
Customization	Required Modification (from [7])	Minimal < parameterization < customization < internal revision < extensive rework
	Possible Modification	None or minimal < parameterization < customization < programming < source code
	Interface	None < documentation < API < OO interface < contract with protocol
Bundle	Packaging	Source code < static library < dynamic library < binary component < stand-alone program
	Delivered	Non delivered < partly < totally
	Size	Small < medium < large < huge
Role	Functionality	Horizontal, vertical
	Architectural level	OS, middleware, support, core, UI

3.1 Source

Origin. We adopt here the definitions proposed in [7]. The possible values we propose for this attribute are: *in-house, existing external, externally developed, special version of commercial, independent commercial*. We consider as commercial a product that is generally available to the public. So are open source and free software products.

Cost and property. The COTS can be obtained for a price or free. Obtaining the COTS could mean acquiring the source code or the executable code. The possible values we propose for this attribute are: *acquisition*, ownership of the product (including source code) is transferred to the buyer; *license*, a use license fee is required to use the product; *free*, no fee is required. Related legal / commercial issues are liability for defects contained in the COTS, responsibility for maintenance, and export restrictions.

3.2 Customization

Carney and Long [6] consider the modifiability attribute. We have split it into required and possible modification.

Required Modification. This attribute corresponds to the modifiability dimension proposed in [7]. It has five possible values: *extensive reworking, internal code revision, customization, parameterization, minimal*. The first two of them assume access to code, the second two imply some mechanism built into the COTS to modify its functionality, the last indicate almost no modification.

Possible Modification. This attribute refers to the internal possible customization of the COTS product. Such kind of modification is not required by the COTS to deliver its basic functionality. As an example, the open source web server Apache typically requires only simple parameterization, although its source code is accessible making any in-depth modification possible. The possible values of this attribute are: *source code*, code is available and can be modified; *programming*, a complete set of API or interfaces is provided possibly together with a scripting language; *customization*, it is possible to define macros or configuration files; *parameterization*, parameters can be defined for the product; *none or minimal*, the product cannot be modified.

Interface. An important factor, which impacts integration and glueware is represented by the interface provided by the COTS product. The possible values for this attribute are: *none*, no documented interface is provided and reverse engineering could be required; *documentation*, there is some documentation of the interfaces provided (e.g. syntax of the configuration files or protocols); *API*, a function level APIs are provided; *interface*, an object-oriented interface is formally defined by means of some standard IDL; *contract*, a contract is defined, that is both a set of interfaces and a protocol for using such interfaces.

This attribute could be very useful also in providing a better definition of component; putting a threshold on this attribute (e.g. interface) you can say if a product is a component or not.

3.3 Bundle

Packaging. The COTS can be packaged in different ways. Possible values for this attribute are: *source code, statically linkable binary library, dynamically linkable library, binary component, stand-alone executable program.* Packaging is the form in which the COTS product is used. A standalone program does not preclude access to the source code.

Delivered. Considering the product delivered to a customer or user, a COTS product can be integrated in it or not. If we consider a project developed in C, the C compiler is not part of the delivered system. However, some tools usually associated with the C compiler (e.g. the library of I/O functions) are probably integrated in the final product. Possible values for this attribute are: *non-delivered, partly, completely*.

Size. An important factor is the size of the COTS, we propose a simple classification in 4 groups ranging from small, like UI widgets, to huge, like the Oracle 8 DBMS or the Windows NT operating system. The approach we adopt is based essentially on the size of the COTS product (the figures in MB are indicative): *small* means less than 0.5 MB, *medium* means from 0.5MB to 2MB, *large* means from 2MB to 20MB, huge means more than 20MB.

An alternative measure is based on the number of use cases supported by the product[9]. While this method has the potentiality to become a good measure of the computational size of software products, it has several drawbacks. The size and complexity of each use case may vary greatly and thus the measure of the size could be inaccurate. Besides, use cases for COTS products usually are not available.

3.4 Role

Type of functionality. COTS offer a variety of functions, however they can be classified in two broad categories. *horizontal*, the functionality is not specific to a domain, but can be reused across many different application domains (e.g DBMSs, GUIs, networking protocols, web browsers); *vertical*, the functionality is specific to a domain, and can be reused only in the domain (e.g. financial applications, accounting, Enterprise Resource Planning, manufacturing, health care management, and satellite control). Horizontal COTS have been available on the market for a long time, experience and know how about them are usually widely available. As a result, using horizontal COTS is usually less risky and more common than using vertical COTS.

Architectural Level. This attribute is somewhat similar to the previous one, but it refers to a generic layered computing architecture. The levels we propose are: *Operating System*; *Middleware*, software which enable communication and

integration; *Support*, elements that cover a well defined and standardized role in the architecture but do not provide vertical functionality; *Core*, products which provide domain specific functionalities; *User Interface*, highly reusable user interface components.

4 Application

We have defined the attributes in **Table 1** with an exhaustive approach, including all attributes that, by speculation, could be relevant to characterize and distinguish COTS products. **Table 2** shows that the proposed attributes are able to discriminate products that any practitioner considers as COTS, but also as very different from one another, not only in terms of the functionality offered, such as operating system, file sharing utility, or user interface widget. For lack of space we limit the list to three products.

However, some attributes could be useless to characterize certain COTS products, or the number of attributes could be too high for any practical use. We need to discriminate the necessary and sufficient attributes.

Table 2. Attribute values for two COTS products.

	COTS Product		
Attribute	**MS Windows NT**	**Samba**	**MS Chart Control**
Origin	Indep. Comm.	Indep. Comm	Indep. Comm.
Cost & Property	License	Free	License
Required Modification	Parameterization	Parameterization	Minimal
Possible Modification	Programming	Source code	Programming
Interface	API	API	Contract
Packaging	Standalone	Standalone	Binary Component
Delivered	Completely	Completely	Completely
Size	Huge	Large	Small
Functionality	Horizontal	Horizontal	Horizontal
Architectural Level	OS	Middleware	UI

4.1 Hypotheses

We identified a set of hypotheses about the possible impact of the attribute values on the development process of the delivered system. An overview of these hypotheses is presented in Table 3.

Table 3. Attribute impact.

Attribute		**Impact on the process**
Source	Origin	ease of change, availability of certification, control on product customization, marketplace competition
	Cost & Property	acquisition and maintenance costs

Customization	Required Modification	customization cost, comprehension effort, integration effort
	Possible Modification	adaptability, ease of integration
	Interface	ease of integration, language/middleware lock-in, architectural contstraints and mismatches
Bundle	Packaging	porting and adaptation effort, configuration management, platform constraints
	Delivered	redistribution issues (both legal and commercial)
	Size	learnability, setup effort
Role	Functionality	reusability across projects, availability of required functionality
	Architectural level	the choice of the product can be dictated by external factors, different integration problems

5 Conclusions

Based on the attributes we presented and the definition found in the literature we propose the following definition of COTS products:

- Origin >= special version of commercial
- Cost & Property >= license
- Required modification =< customization
- Possible modification >= parameterization
- Interface >= API
- Packaging >= static library
- Delivered = totally
- Size >= medium
- Functionality = vertical
- Architectural level in { support, core }

The main contributions of this work are:

- a survey of current COTS definitions
- the proposal of a new COTS definition: as a result COTS remains a broad term but we identified a set of attributes that can discriminate different COTS products
- a set of hypotheses about the impact of the defined attributes on the COTS based development process

Both the attributes we identified and the impact hypotheses we formulated are not definitive. They are to be considered as statements to set an initial framework and stimulate discussion.

We plan to revise the attribute list and validate the hypotheses about its impact onto the development process.

6 References

[1] C.Abst, B.Boehm, E.Clark. “COCOTS: A COTS Software Integration Lifecycle Cost Model - Model Overview and Preliminary Data Collection Findings", Technical report USC-CSE-2000-501, USC Center for Software Engineering, 2000.

[2] V.Basili, B.Boehm. “COTS-Based Systems Top 10 List” IEEE Computer 34(5), May 2001, pp 91-93

[3] P.Brereton, D.Budgen. "Component-Based Systems: A Classification of Issues". IEEE Computer Vol. 33, No. 11; November 2000, pp. 54-62

[4] L.Brownsword, D.Carney, T.Oberndorf. “The Opportunities and Complexities of Applying Commercial-Off-the-Shelf Components” SEI Interactive, 6/98, 1998, avail. at http://interactive.sei.cmu.edu/Features/1998/June/Applying_COTS/Applying_COTS.htm

[5] L.Brownsword, T.Oberndorf, C.Sledge. "Developing New Processes for COTS-Based Systems". IEEE Software July/August 2000, pp. 48-55

[6] D.Carney. “Assembling Large Systems from COTS Components: Opportunities, Cautions, and Complexities”. SEI Monographs on Use of Commercial Software in Government Systems, Software Engineering Institute, Pittsburgh, USA, June 1997.

[7] D.Carney, F.Long. “What Do You Mean by COTS?”. IEEE Software, March/April 2000, pp. 83-86.

[8] A.Egyed, N.Medvidovic, C.Gacek. “Component-based perspective on software mismatch detection”. IEE Proceedings-Software, Volume: 147 Issue: 6, December 2000

[9] M.Tsagias, B.Kitchenham. “An Evaluation of the Business Object Approach to Software Development”. The Journal of Systems and Software 52, 2000, pp 149-156

[10] M.Morisio, C.Seaman, A.Parra, V.Basili, S.Kraft, S.Condon. “Investigating and Improving a COTS-Based Software Development Process”. 22nd International Conference on Software Engineering, Limerick, Ireland, June 2000.

[11] T.Oberndorf. “COTS and Open Systems - An Overview”. 1997, available at http://www.sei.cmu.edu/str/descriptions/cots.html#ndi

[12] OMG, “Unified Modeling Language Specification” version 1.3, June 1999.

[13] C.Szyperski. “Component Software Beyond Object Oriented Programming”. Addison-Wesley, 1998.

[14] M.Vigder, M.Gentleman, J.Dean. "COTS Software Integration: State of the Art". Technical Report NRC No. 39190, 1996.

[15] M.Vidger, J.Dean. “An Architectural Approach to Building Systems from COTS Software Components”. In Proceedings of the 1997 Center for Advanced Studies Conference (CASCON 97), Toronto, Ontario, 10-13 November, 1997, available at http://seg.iit.nrc.ca/English/abstracts/NRC40221abs.html

[16] J.Voas. "Faster, better, cheaper". IEEE Software, May/June 2001, pp. 96-97

[17] K.Wallnau, D.Carney, B.Pollak. “How COTS Software Affects the Design of COTS-Intensive Systems”. SEI Interactive, 6/98, 1998, available at http://interactive.sei.cmu.edu/Features/1998 /June/cots_software/Cots_Software.htm

[18] K.Wallnau. "On Software Components and Commercial ('COTS') Software". In Proceedings of 1999 International Workshop on Component-Based Software Engineering, Los Angeles, CA, USA, May 17-18, 1999

[19] H.Wang, C.Wang. "Open Source Software Adoption: A Status Report". IEEE Software, March/April, 2001

[20] D.Yakimovich, J.Bieman, V.Basili. “Software Architecture Classification for Estimating the Cost of COTS Integration”. Proceedings of the 21st International Conference on Software Engineering, Los Angeles, USA, 1999, pp. 296 –302.

[21] M.Zelkowitz, D.Wallace. "Experimental Models for Validating Technology" In IEEE Computer 31(5), May 1998, pp. 23-31