Overlooked Aspects of COTS-Based Development

Marco Torchiano, *Norwegian University of Science and Technology*

Maurizio Morisio, *Politecnico di Torino*

Although developing with commercial-off-the-shelf components is gaining more attention from both research and industrial communities,[1] most literature on the topic doesn't clearly identify context variables such as the type of products, projects, and systems. In particular, the literature often lacks a definition of "COTS product," or, when a definition is present, it invariably disagrees with other studies (see the sidebar, "What COTS Product Do You Mean?"). A shared definition would improve discourse and enable researchers to meta-analyze published empirical data.

After a speculative effort to define and classify COTS products,[2] we decided to obtain this much-needed understanding from the bottom up. We asked people involved in industrial projects to name the key features in COTS-based development and to tell us what they think constitutes a COTS product. Here, we present the interview results in the form of six theses, which contradict widely accepted (or simply undisputed) ideas. We also present a definition of "COTS product" that captures the key features.

The sample of projects we used is limited and can't represent the whole IT industry. Instead, our study primarily considers Small-Medium Enterprises. While SMEs produce most software, they're usually neglected by research. Specifically, the COTS-development literature seems to focus on large, mostly governmental projects performed in major companies. Our goal is to empirically explore this key topic for the software industry, collect facts, and derive well-substantiated theses from these facts.

Studies on commercial-off-the-shelf-based development often disagree, lack product and project details, and are founded on uncritically accepted assumptions. This empirical study establishes key features of industrial COTS-based development and creates a definition of "COTS products" that captures these features.

The groundwork

From a methodological viewpoint, we conducted empirical research following two parallel threads that complement each other: a systematic literature investigation and an on-field exploratory study based on structured interviews. Because the industry doesn't have a common understanding of COTS products, we laid

What COTS Product Do You Mean?

Does the software industry have a common understanding of what a COTS (commercial off the shelf) product is? Based on the evidence, the answer is no. Many papers don't define the term, and those that do disagree on the definition. As part of our effort to better understand what types of products are considered COTS and to create a widely shared definition of the term, we conducted a literature survey of COTS papers.

We conducted our search for papers in January 2002 using the Web-based search facilities of the IEEE (IEEExplore) and the ACM (ACM Portal). We targeted journals, magazines, and transactions published between 1994 and 2001.

We selected papers in two phases. First, we searched using the keywords "COTS" and "off the shelf" and identified about 300 papers. Next, we screened these results to discard irrelevant papers (for example, those related to hardware COTS) and narrowed the candidates to 21.

Of these 21 papers, six didn't provide any definition for COTS products. The 15 remaining papers defined it in terms of different features. Table A lists the features and the number of papers mentioning them in the definition. Although some features recur several times, no two papers cite the same combination.

Because the definitions we found were unsatisfactory, we looked for examples of COTS products. Among the 21 papers, five gave no example of COTS products, four gave only examples of categories of products (for example, database or OS), and 12 gave examples of real products.

Table A

COTS products' key features found in a sampling of the literature

Features	Number of citations
No control over evolution	5
Black box (no source)	5
General purpose	4
API availability	2
No control over quality	2
No control over functionalities	2
Large installed base	2
Acquired elsewhere	1
Coarse grained	1
Copyright	1
No modification	1
Vendor seeking profit	1

the groundwork for these interviews by determining certain assumptions and definitions, in particular for "COTS product" and "COTS process." (In a moment, we'll provide a more detailed definition derived from the interviews.)

COTS-based development differs from the traditional approach in many respects.[3] COTS-based development has these distinguishing characteristics:

- Development essentially occurs through combining existing products.
- The marketplace exerts a strong influence on the evolution of COTS products.
- A continuous trade-off happens during development.

In conventional development, variations stem from only the requirements. Developers capture variations at the requirements-definition stage, and subsequent steps follow more or less mechanically. In COTS-based development, variations stem from either the requirements or the marketplace. Their accommodation involves a continuous trade-off among the requirements, the COTS products on the market, and the system's software architecture.

Vendors produce and distribute COTS products, integrators compose a set of products into a COTS-based system, customers acquire the system, and users use it. To illustrate the process, we've created a simplified COTS-based development chart identifying four main roles and two types of artifacts (see Figure 1).

The *vendor* produces, sells, or distributes the COTS product and is the source for both the product and its documentation. Often, the vendor provides support services, which might

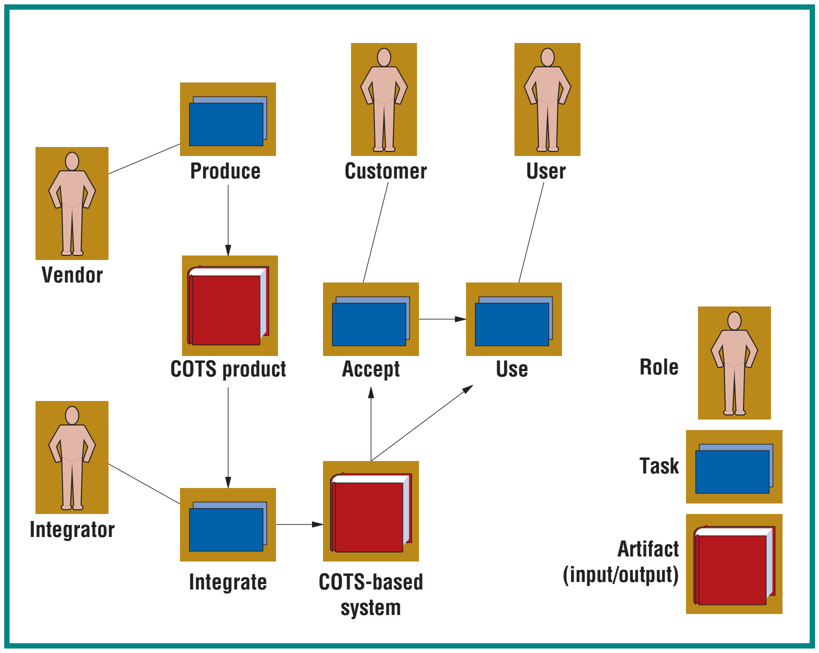

Figure 1. Simplified COTS-based development.

Table 1
Project descriptions

Project id	COTS-based system	Effort (person-months)	Staff	Elapsed time (months)	Country
A	A control framework for deploying telephony services based on policy	24.0	4	6	Italy
B	Integrated phone and Internet voice services with policy definition	108.0	8	18	Italy
C	A Web-based search service with a public Web interface and a special interface for companies	840.0 with maintenance	20	42	Norway
D	A digital broadcast system for satellite or broadband, such as TV on demand	90.0	9	10	Norway
E	A platform for integrated networks to deliver services as well as an initial call-center application	136.0	8	17	Norway
F	An intranet and extranet portal for an association of technology companies	10.0	4	2 5	Norway
G	A workflow-management system for service requests management	0.5	1	1	Italy

be free or sold separately.

Intuitively, a *COTS product* is software that isn't developed in the project. Rather, it's acquired from a vendor and used as-is or with minor modifications. To avoid bias, we adopted this wide definition in our empirical study.

The *integrator* builds the system using COTS products and, possibly, internally developed custom software (glueware) and components.

A *COTS-based system* (or *final system* or *system*) is the item built by the integrator and delivered to the customer. For simplicity's sake, we'll consider the CBS (or part thereof) built into a single project.

The *customer* acquires, pays for, and dictates the CBS requirements. The customer might also be the user.

The *user* is the set of people who use the CBS after the customer receives it from the integrator.

The interviews

Between January and May 2002, we conducted structured interviews in Norway and Italy with representatives from seven small- and medium-sized enterprises. One of us was the interviewer and the other was the scribe. The interviewer led with questions based on a four-part questionnaire, which aimed to

- Characterize the project
- Describe the software architecture of the project's CBS
- Characterize the COTS products used and describe any issues
- Describe the selection criteria for COTS products

The scribe recorded answers and interrupted only for clarification when necessary. To make the interviews more palatable, we kept the sessions from 45 to 75 minutes.

At the end of each interview, we wrote a transcript and sent it to the interviewee for comments and corrections. We also offered to provide interviewees with our investigation results.

Projects

Our location was the main factor in choosing projects. We selected projects whose representatives were available for interviewing when we were in their country. In all cases, the interviewees played the role of integrator during COTS-based development.

Each interview focused on a single project using one or more COTS products. Table 1 features some key attributes of the projects.

To characterize project size, we used "project effort" rather than "size of the delivered product." Aside from the difficulty of measuring product size, integrating COTS products usually results in large delivered products even when the effort is low.

The projects used a wide variety of products that can be classified according to the architectural level—that is, their intrinsic role in a generic layered computing architecture:[2]

- *Operating systems:* Linux, Solaris, QNX, MS NT, and FreeBSD
- *Middleware:* Visibroker, BEA Weblogic, and OpenORB
- *Databases:* MySQL, MS Access, Oracle DB, MS SQL, and Hypersonic SQL
- *Support software:* PHP, Apache, Tomcat, IBM JRE, WebMacro, Log4J, Xerces, Agent++, Intel COPS Client, Telia BER Coder, and XMLDB

How the literature compares

To establish the interview's context and limitations, we began each interview by stating our motivation and presenting our basic definitions, which didn't include a definition for COTS products. We rigorously avoided suggesting any hypotheses during the interviews. Moreover, to avoid any bias on our part during the interviews, we saved analysis for a later stage of the study.

At the end of the interview, we asked the interviewee to enumerate the key features of COTS products. Six participants cited "not produced here." "Not modified" and "not a commodity" were both cited three times. Two participants said a COTS product is a "general purpose" piece of software.

We compared these features to those found in the literature (see Table A in the sidebar) and focused on the three features cited most frequently in each. The only common trait is that a COTS product is a general-purpose piece of software; it's not developed with a specific project in mind.

The most striking difference is that the interviewees emphasized the "not produced here" feature, which wasn't particularly important in the literature. The definitions in the literature might have considered this feature implied in the COTS acronym. However, the word "commercial" remains ambiguous when considering government off-the-shelf products, open source products, and products or components developed by other sites or business units in large corporations. "Not produced here" stresses that the project using the component didn't develop it, but it doesn't set any constraint on the property (of a copyright license, of the source code, of a copyleft license, or whatever else).

Most interviewees distinguished between COTS products and commodity software. The latter category includes all the software usually bundled with the operating system or part of the platform. On the other hand, interviewees didn't clearly distinguish COTS products integrated in the final CBS from those not integrated—that is, tools used to produce the final system. Six of the seven interviewees listed as COTS those products to be integrated; only one interviewee specified tools.

A definition of COTS products

Using information derived from our interviews, we can summarize COTS products' distinguishing traits in a more detailed, empirically based definition:

> *A COTS product is a commercially available or open source piece of software that other software projects can reuse and integrate into their own products.*

In particular, a COTS product

- Isn't produced in or exclusively for the project.
- Can be closed source or open source. If it's open source software, it's usually treated as if it were closed.
- Isn't a commodity. It's not shipped with the operating system, provided with the development environment, or generally included in any preexisting platform.
- Is integrated into the final delivered system. It's not a development tool.
- Isn't controllable, in terms of provided features and their evolution.

Table 2

Theses of COTS-based development vs. popular opinion

Our theses	Popular opinion
T1. Open source software is often used as closed source.	Open source software is quite different from closed-source software.[4]
T2. Integration problems result from lack of standard compliance; architectural mismatches constitute a secondary issue.	Architectural mismatches are the main hurdles in composing systems from existing components.[5,6]
T3. Custom code mainly provides additional functionalities.	Custom code plays mainly an integration role.[1,7,8]
T4. Developers seldom use formal selection procedures. Familiarity with either the product or the generic architecture is the key factor in selection.	COTS products are selected (or should be) through a formal procedure.[9]
T5. Architecture is more important than requirements for product selection.	Requirements drive all renegotiation processes involving COTS products.[9,10]
T6. Integrators tend to influence the vendor on product evolution whenever possible.	Integrators must passively accept lack of control over the vendor.[1,8]

Our theses

Besides defining COTS products, we derived from the interviews a deep insight into COTS-based development. Interestingly enough, the theses we developed contrasted with statements or implicit assumptions frequently found in the literature (see Table 2).

T1. Open source software is often used as closed source. Many projects used open-source or free- software products. Although the source code was easily available, the projects didn't look at the code. This was primarily because the projects didn't need to see or modify it or they lacked the knowledge, skill, or resources to do so. The projects considered the source code's availability and the presence of a large developer community as a guarantee in case any technical problems arose. Open source products are usually considered to be completely different from closed-source prod-

Formalizing the selection process for COTS products has sparked great interest in the past few years.

ucts.[4] However, we found that, in practice, the projects used and treated them both in the same way. So, we argue that the COTS product debate should include open source software products.

T2. Integration problems result from lack of compliance with standards; architectural mismatches constitute a secondary issue. Most projects adopted at least one well-established generic architecture and technology, such as the Linux-Apache-MySQL-PHP (LAMP) architecture from the business-developer community.

These solutions are based on interactions among components enabled through standard protocols such as Corba, COM (Component Object Model), EJB (Enterprise JavaBeans), and SQL (Structured Query Language). Problems arise when the architecture's components don't fully or correctly support the standards or when different components support different and incompatible versions of a standard. In one project, two COTS products supported slightly different versions of Corba, making it impossible for them to interact.

The literature cites architectural mismatches as one of the primary sources of integration problems;[5,6] however, we found no instances of this. On the other hand, reusing a well-established architecture, such as LAMP, was a success factor for the projects.

T3. Custom code mainly provides additional functionalities. In several projects, the selected COTS product lacked only a few required functionalities. One of the most common remedial solutions among interviewees in such a case was to add custom code to the components to provide the missing functionalities. We call this type of code "addware."

The literature greatly emphasizes the integration aspects of custom code (often called "glueware").[1,7,8] However, based on the interviews, we've concluded that projects use custom code more often as addware. This is largely due to the emergence of sophisticated component systems, such as Enterprise JavaBeans and .NET, which include development environments that generate most of the glue-code automatically. Also, in relation to Thesis 2, if developers use an established architecture, the project will likely require little effort for integration.

T4. Developers seldom use formal selection procedures. Familiarity with either the product or the generic architecture is the leading factor in selection. Formalizing the selection process for COTS products has sparked great interest in the past few years. Researchers have proposed several structured, formal, or semiformal selection procedures.[9] However, no projects in our survey used any of them. These procedures suggest taking into account a variety of attributes, but we found that familiarity is often the only attribute considered. Several projects chose both the system architecture and the COTS products in this manner. This practice makes sense when you consider that adopting a new product might require a long period of familiarization before effective and efficient use. And, when resources and time are limited, it's not an attractive option. For instance, in Project D, the developers chose one relational database over another because they used the product before and knew its administration interface.

T5. Architecture is more important than requirements for product selection. In some projects, the developers first identified the product providing the system's core functionalities and then adopted its (possibly implicit) architecture. In other projects, the developers decided on the architecture before detailing the requirements. In both cases, once the architecture was chosen, it placed constraints on product selection.

For COTS products selection, some researchers propose a process based largely on requirements and their negotiation.[9,10] Although requirements are important, we noticed that in the projects in our study, the architecture really drove the selection process and was the fulcrum of all trade-off activities.

T6. Integrators tend to influence the vendor on product evolution whenever possible. In a few projects, the integrators tried to drive the COTS products' evolution toward their own needs. An integrator can increase control by

- Acquiring the firm that owns the COTS product. This occurred in one project.
- Providing resources for open source development of the product to include the required features. This occurred in two projects.
- Leveraging a *monopsonistic* condition (a

market situation in which the only buyer exerts a disproportionate influence on the market) for a niche market. This occurred in one project.

Published studies generally describe the lack of control over the features and evolution of COTS products as an immutable condition that integrators must tolerate.[1,8] In our experience, integrators act to modify this condition—something especially true with open source products.

Far from being a comprehensive study of COTS, this article offers insight into an area usually neglected in other published studies: COTS-based development from the perspective of Small-Medium Enterprises. On the basis of our interviews, we suggest further research into five key aspects of COTS-based development.

The first aspect is *assessing standards compliance* (for example, CORBA and SQL). Standards compliance is important for integrating commercial products. Proprietary variants jeopardize plug-and-play capabilities, thereby restricting the ability to choose the best products. Certifying COTS products' compliance to standards is urgently needed to avoid these problems.

The second is *COTS products' extensions*. Missing or incomplete features in COTS products are significant problems. Because developers create and glue extensions to products ad hoc, it's important to develop generic techniques that allow the seamless extension of existing products.

The third is *familiarity with COTS products*. The most important factor in COTS products selection is familiarity (either with the products or with the architectural style). Developers often choose a suboptimal product only because they already know it. So, developing strategies for mining COTS products and acquiring quick familiarity with them is an important goal. Selection techniques should also include familiarity as a main criterion.

The fourth is *architecture*. Software architecture is the main tool for carrying out the trade-off between CBS requirements and available product features. This means the need is growing for research on architecture renegotiation strategies and trade-off methodologies.

The final aspect is *techniques for influencing COTS products' evolution*. Integrators seek and apply techniques for influencing COTS products' evolution. For instance, participating in an open source project lets developers steer development to favor and prioritize the most wanted features. Other techniques based on market forces or communication strategies could be devised to support this consumer-producer relationship.

About the Authors

Marco Torchiano is an assistant professor at Politecnico di Torino. His research interests include component-based software engineering, COTS-based development, software architecture, software maintenance, and design patterns. He received his PhD in computer engineering from Politecnico di Torino. He's a member of the IEEE Computer Society. Contact him at the Dip. Automatica e Informatica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy; marco.torchiano@polito.it.

Maurizio Morisio is an assistant professor at Politecnico di Torino. His research interests include experimental software engineering, wireless Internet software engineering, software reuse metrics and models, agile methodologies, and COTS processes and integration. He received his PhD in software engineering from Politecnico di Torino. Contact him at Dip. Automatica e Informatica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy; maurizio.morisio@polito.it.

Acknowledgments

We thank Reidar Conradi and Letizia Jaccheri for supporting this study. A special acknowledgment goes to the interviewees for their kind collaboration. Reviewers' comments and suggestions greatly helped improve the article. The Norwegian IKT project INCO (*In*cremental *Co*mponent-based development; nr.140398/431) partly supported this work.

References

1. V. Basili and B. Boehm, "COTS-Based Systems Top 10 List," *Computer*, vol. 34, no. 5, 2001, pp. 91–93.
2. M. Morisio and M. Torchiano, "Definition and Classification of COTS: A Proposal," *Proc. 1st Int'l. Conf. COTS-Based Software Systems* (ICCBSS 2002), LNCS 2255, Springer-Verlag, 2002, pp. 165–175.
3. L. Brownsword, T. Oberndorf, and C.A. Sledge, "Developing New Processes for COTS-Based Systems," *IEEE Software*, vol. 17, no. 4, 2000, pp. 48–55.
4. G. Lawton, "Open Source Security: Opportunity or Oxymoron?" *Computer*, vol. 35, no. 3, 2002, pp. 18–21.
5. A. Egyed, N. Medvidovic, and C. Gacek, "Component-Based Perspective on Software Mismatch Detection," *IEE Proc.-Software*, vol. 147, no. 6, 2000, pp. 225–236.
6. D. Yakimovich, J. Bieman, and V. Basili, "Software Architecture Classification for Estimating the Cost of COTS Integration," *Proc. 21st Int'l. Conf. Software Eng.* (ICSE 99), ACM Press, 1999, pp. 296–302.
7. M. Morisio et al., "Investigating and Improving a COTS-Based Software Development Process," *Proc. 22nd Int'l. Conf. Software Eng.* (ICSE 2000), ACM Press, 2000, pp. 32–41.
8. B. Boehm and C. Abts, "COTS Integration: Plug and Pray?" *Computer*, vol. 32, no. 1, 1999, pp. 135–138.
9. P. Lawlis et al., "A Formal Process for Evaluating COTS Software Products," *Computer*, vol. 34, no. 5, 2001, pp. 58–63.
10. B. Boehm, "Requirements That Handle IKIWISI, COTS, and Rapid Change," *Computer*, vol. 33, no. 7, 2000, pp. 99–102.