

Thoughts on Continuing Successful Use of COTS and the Laws of Software Evolution

MM Lehman

Department of Computing
Imperial College
London SW7 2BZ
tel: +44 (0)171 594 8214
fax: +44 (0)171 594 8215
mml@doc.ic.ac.uk

COTS is *E*-type

Complete and formally specified COTS components satisfy the definition of *S*-type systems [1]. Their properties are defined by and only by their specifications. In execution, they satisfy the latter absolutely. All other potential properties are implicitly defined as *don't care*. Behaviour under external conditions sensitive to the latter is not predictable and no assumptions can or should be made about unit behaviour if one is encountered in execution. Provided the specified interface definitions are completely adhered to, the unit will continue to behave as specified, when built (integrated) into some other system. However, unless the host system is assembled entirely from *S*-type units and does not interact in execution with the real world, these components rapidly acquire *E*-type characteristics. Bricks built into a wall acquire or reveal properties not addressed in their specification by virtue of the building process and its association with other wall and environment elements.

In practice software addressing a real world domain or executed in the real world cannot be precise or complete. As a special case of a general principle fully discussed elsewhere [2, 3] the numbers of attributes that would have to be covered is, at least, countably infinite. It cannot be contained within a finite specification. Such software is, therefore, of type *E* [1]. This also applies to software components that support a real world *application* in a real world operational domain. In isolation, as long as it is *on the shelf* the component may behave as an *S*-type program. Once integrated into a real world application system, however, as part of the system, they inherit the attributes of *E*-type systems. Their life cycle and other behavioural characteristics will conform to that encapsulated in the laws of software evolution [1,4, 5]. COTS evolves. This position paper summarises the implications of this assertion. The key issues of concern follow from the fact that the host software in which COTS components operate must undergo continuing evolution.

Laws of Software Evolution - Current Status

The FEAST/1 project has been accumulating and modelling metric data from and the process dynamics of five software systems currently being evolved [4] since 1996.

Inter alia, the investigation has sought to identify the presence of feedback mechanisms and controls in the global processes associated with software evolution and to evaluate their impact [6]. The study has yielded significant evidence for the presence of feedback and dynamics in all the systems [4]. The accumulated evidence is consistent with and supportive of the laws of software evolution, strengthening confidence in their overall validity [5].

Laws, COTS and their Interaction

The laws encapsulate observed behaviour that is organisational and sociological in nature, exogenous to the technologies being applied in the evolution process, hence their designation as laws. Their implications for the use of COTS, briefly reviewed here, have previously been independently discussed in two sources [7, 8]. Both conclude that more investigation is required.

COTS integration into other systems, soft or hard, covers a spectrum of forms and degrees of coupling. It may be used, as are word processors and spreadsheets, within an application system environment to provide specific support facilities, stand alone with a many way interface to other systems, people and the machines that are involved in the activity addressed. Operational integration of the COTS is here a consequence of human or other activity or intervention. At the other extreme, one has COTS *wired in* to the application system by procedure calls or shared access to common data or devices, for example. An intermediate case, weak coupling, is illustrated by input-output data linkage where the application and COTS software communicate via buffers that permit each to view the other as devices. The degree to which the observations that follow are relevant over this wide spectrum needs further elucidation. In what follows, references to COTS should, generally, be taken to refer with certainty to the physically integrated, tightly coupled situation.

Types of Implications

In considering the impact of the laws three questions are raised here. What is their impact on COTS and on its use? What are the implications of such use on system maintenance and its continued satisfactory performance within changing application and system domains? Does

the use of COTS invalidate the laws completely or require their substantial modification? We can only touch on these questions here.

Embedded Assumptions and Uncertainty

It has been shown that as finite models of countably infinite domains, all *E*-type software contains explicit and implicit embedded *assumptions* related to the state of the real world and activities in it. But the real world domain, including computer applications are always changing at a rate accelerated by installation and operation of the system. Thus, even supposing all assumptions were initially valid, more and more become invalid or lead to unsatisfactory behaviour as time goes by. This phenomenon may be seen as the ultimate source of the universally experienced need for unending software maintenance and as posing a crucial challenge to the use of COTS.

Summary of Conclusions

From the above it follows that when considering or adopting COTS, organisations *must* negotiate an appropriate relationship with the vendor. For in the maintenance context the host maintenance organisation and process become hostages to the COTS and its vendor.

In response to market forces, and the feedback induced, for example, by the use of COTS, the vendor will evolve COTS components. Before a host organisation adopts a new version of a component comprehensive checking of all known assumptions, implicit or explicit, in the host system, in the change implementation and in both the original and replacement components is required.

If new versions are not adopted the host organisation will eventually lose vendor support. Either way significant problems will arise. A minimal condition for COTS adoption is expressed willingness of the vendor to support host maintenance as required, access to COTS source code or its deposit in escrow.

As host organisations develop, maintain and evolve their systems they must consider not only the declared specification of each COTS component but also the impact of the assumptions embedded in their own system and in the COTS components. This clearly involves a significant and difficult challenge. In particular, the vendors will likely be unwilling, probably unable, to give comprehensive data on the assumption set underlying the component other than published in the specification. Clearly, they will be unable to provide information on implicit assumptions. As a result, the use of COTS introduces a new level of complexity and uncertainty [3] in development and, more so, in system maintenance and evolution.

The third major issue considered here relates to the continuing validity of the laws and their implications on practical software evolution and management. This com-

plex issue can clearly not be fully addressed here. In summary, it may be said that, initially, the introduction of COTS will possibly reduce behaviour and constraints implied by the laws. However, as an increasing number of such components are used in a system, one may usefully think of each as a *primitive* in a language defined by the full set of COTS units, with the interfaces providing the syntax of their use. Thus one may expect that the implied behaviour and constraints will reappear, perhaps in a somewhat more complex form, because of the disparate *language* levels used in the system construction.

Some Recommendations

A number of recommendations can now be made. The degree to which they are essential, offer potential for improving the process of usage or reduce the Trojan Horse effect, cannot be discussed here. Specific recommendations to COTS developers must also be excluded.

1. Capture, record, structure and annotate and periodically review assumptions made throughout the entire development and maintenance process.
2. Use only COTS components for which a formal specification is provided, against which the component has been validated and which includes a comprehensive list of assumptions and *don't cares*.
3. Receive assurances that these will be maintained as the COTS units themselves evolve and that users will receive the appropriate documentation and support.
4. Introduce process steps that address issues and impact of embedded assumptions, both before adoption of COTS units and whenever changes are made.

References¹

- [1] Lehman MM and Belady LA, Program Evolution, - Processes of Software Change, Acad. Press, London, 1985
- [2] Lehman MM, Feedback, Evolution and Software Technology - The Human Dimension, Worksh. on Human Dimensions in Successful Softw. Dev., ICSE 20, April 21, 1998, Kyoto, Japan
- [3] *id.*, Uncertainty in Computer Application and its Control Through the Engineering of Software, J. of Softw. Maint.: Res. and Pract., v. 1, n. 1, Sept. 1989
- [4] <http://www-dse.doc.ic.ac.uk/~mml/feast1/>
- [5] Lehman MM, Perry DE, Ramil JF, On Evidence Supporting the FEAST Hypothesis and the Laws of Software Evolution, to appear, Proc. Metrics'98, Bethesda, Maryland, Nov. 20-21, 1998
- [6] Lehman MM and Stenning V, FEAST/1: Case for Support, Dept. of Comp., Imp. Col., March 1996
- [7] Lehman MM and Ramil JF, Implications of Laws of Software Evolution on Continuing Successful Use of COTS Software, Dept. of Comp, Imp. Col, Tech. Rep. 98/8
- [8] Hybertson, DW, Anh DT and Thomas WM, Maintenance of COTS-Intensive Software Systems, Soft. Maint: Res. and Pract, v.. 9, 1997

¹ Additional references and material in [4]