

# Implications of Laws of Software Evolution on Continuing Successful Use of COTS Software

M M 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  
www-dse.doc.ic.ac.uk/~mml/

J F Ramil

Department of Computing  
Imperial College  
London SW7 2BZ  
tel: +44 (0)171 594 8216  
fax: +44 (0)171 594 8215  
jcf1@doc.ic.ac.uk  
www-dse.doc.ic.ac.uk/~jcf1/

## Abstract

However completely specified, integration of COTS software into real world system makes it of type *E* even though, were it to be fully and absolutely specified, it would satisfy the definition of an *S*-type system. Thus, the laws of software evolution that apply to *E*-type systems are also relevant in the COTS context. This paper examines the wider implications of this fact and in particular that such systems must undergo continuing evolution. Managerial implications of the laws of software evolution in the context of COTS are also briefly highlighted.

**Keywords:** Laws of Software Evolution, COTS, *E*- and *S*-type Systems, Specification, Assumptions

## 1 The Move to COTS and Reuse

The use of *commercial off the shelf software* (COTS) is increasingly being seen as a way of reducing the problems so often encountered in *ab initio* software development. A parallel may be drawn from other engineering fields in which the concept of components is universally accepted, standards are agreed on an international basis. These are exemplified by nuts and bolts in mechanics and a wide variety of electrical components and connectors in electronics, which can be obtained from different manufacturers in the market and then be used as building blocks of a unique and purposeful final product.

A complementary trend has considered software *reuse*, that is the reuse in some system of a software component originally developed for some other system or systems. While recognising that many of the arguments presented in this paper with regards to the use of COTS apply equally to the reuse of software components, the remain of this paper concentrates only on the first. The evolution of the latter are in some respects simpler to manage, in others they present new problems that are beyond the scope of the present analysis.

## 2 The Current Status of Laws of Software Evolution

Over the last several months the FEAST/1 project has been accumulating metrics data from four software systems currently being evolved by their respective organisations, ICL, Logica, Lucent Technologies and Matra - British Aerospace Dynamics. As stated in the original project proposal [leh96c], the investigation seeks to identify the presence of feedback mechanisms and controls in the global processes associated with each system, to evaluate the impact of such feedback, to improve process planning and control and to propose process improvements suggested by analysis of the observations and interpretations. As already reported in part [leh96d,97,8 tur96] (more comprehensive reports in preparation) the analysis has produced significant evidence of the presence of feedback control [leh96b] in each of the processes and a system dynamics [leh98b]. The latter appears to control the long term trends and short term fluctuations in their respective individual growth trajectories. Furthermore, the data and process models derived from it and interpretation of the latter are consistent with the laws of software evolution, as stated in various publications over the last twenty four years [leh74,78,80,96a]. Results from the FEAST/1 project provide increasing support and strengthen confidence in their overall validity [leh98d], though some change in detail and nuance is required to take into account the insights and understanding that has developed during the investigation.

### 3 Laws, COTS and their Interaction

#### 3.1 The Laws and COTS

Having previously been discussed in the literature as referenced above, the laws, models of software evolution and the evidence that the latter support the former will not be examined in detail in the present paper. The intent here is to examine the implications of the laws, if valid, on the evolution of COTS [kon96, sul97] software. The issue of the laws in the context of COTS has also been raised elsewhere [hyb97] and it is concluded there that further investigation is required. The current analysis was developed without knowledge of that source.

When first formulated the laws related to *Large Systems*, as variously defined [leh78,80a]. Subsequently they were seen to apply to all *E*-type (real world) systems [leh80b] irrespective of their size, their functional variety or the nature of the organisation and management structure within which they were developed. The precise form that the laws take is not of major consequence for this discussion. It should, however, be mentioned that the laws reflect and encapsulate observed behaviour that is organisational and sociological in nature and, so, exogenous to the technology being applied in the evolution process. It is this fact that suggested the term *laws* when they were first formulated.

It must be noted that COTS components integration into a unique software system can take many forms and covers a spectrum of degrees of coupling. At one extreme one may consider software that is used within some application environment to provide specific facilities to support the application. In operation it is stand alone with a many way interface between it, one or more application systems, the people and the machines that are active in pursuit of the application. Examples are provided by word processors and spreadsheet software. The integration of the COTS, while real, is a consequence only of human activity or intervention. At the other extreme one has the case where the COTS is "wired in" to the application system by means of, for example procedure calls or shared access to common data or storage devices. 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. Determination of the relevance to the implication on the use of COTS of the laws of software engineering, of the degree of coupling as exemplified by the above three situations is not, as yet clear or self-evident. In what follows, references to COTS and COTS systems should, generally, be taken to refer with certainty to the fully physically integrated, tightly coupled situation. The degree to which they apply to, say, information flow linkages or to coupling via human action requires further elucidation.

If its behaviour were completely defined, COTS *per se* would satisfy the definition of *S*-type systems [leh80b]. In that event, the laws would not apply to it as long as it is "on the shelf". Once physically integrated into a real world application system, however, they would, as part of the system, certainly have the attributes of *E*-type systems. It is on this observation that the present analysis is based.

#### 3.2 Types of Interaction

To facilitate the discussion, the laws as recently reformulated, are individually stated and their implications outlined. Before doing so, however, one further remark is in order. Three factors must be considered. What is the impact of the laws on the use of COTS software from the point of view of that software? What, if any, is the impact of the use of COTS software on maintaining the whole system satisfactory within a continually changing application and system - hard, soft, human - domain within which the software system and its COTS elements operate? Finally, is the use of COTS likely to so materially effect the validity of the laws as formulated, to invalidate or require their substantial modification?

#### 3.3 The Principle of Uncertainty [leh89,90]

One further fact plays a vital role in the analysis that follows. The software component of any general purpose computer system implementing some application is a model of that application in its operational domain. This implies that to be complete that software (or the system of which it is a part) contains a model of itself since, once installed, the system becomes an intrinsic part of the application in its real world domain. The real world application and operational domain are each, intrinsically, unbounded. However many observations or properties one identifies and associates with either, one can always add another. The system that is the model is clearly finite. One has, for example, only finite time and resources to create it, there exists only finite storage capacity to hold it, human knowledge and understanding of the application and its domain is limited. All these limitations cause the computer system to be incomplete. A further source of incompleteness arises from the fact that a finite model cannot contain a precise model of itself. As a model, every *E*-type computer system is therefore multiply

incomplete. The resultant gap between the system and its operational domain is bridged by *assumptions*, explicit and implicit, that, one way or another, are embedded in the system. But the real world domain and the application itself are dynamic, always changing, change accelerated by installation and operation of the system. Thus even supposing all assumptions were initially valid in the domains as visualised, more and more become invalid, or lead to unsatisfactory operational functionality or behaviour as time goes by. This phenomenon may be seen as the ultimate source of the unending need for software maintenance that has been universally experienced since the beginning of the computer age.

## 4 The Laws and their Impact

### 4.1 Statement of the Laws

In quoting the laws their most recent formulation is used though, as already stated, their precise formulation is likely to be refined as further insight is gained through the study of a number of systems developed by different organisations for a variety of applications in significantly different development domains.

### 4.2 The First and Sixth Law

**Continuing Change (I):** *E-type systems must be continually adapted else they become progressively less satisfactory.*

**Continuing Growth (VI):** *The functional content of E-type systems must be continually increased to maintain user satisfaction over their lifetime.*

The term user in the sixth law applies in the context of the present paper to both COTS integrators and end-users. Note that these statements do not include or refer to the fact that the operational domain of any *E-type* software also undergoes continuing change. Such change is partly driven by installation and operation of the system, partly by exogenous forces. This observation appears to deserve formulation as a ninth law, representing as it does a fundamental and inescapable fact with profound implications on computer usage, design, implementation and maintenance. For the purpose of this paper, however, it is sufficient to recognise that it lies directly at the root of the first law [leh98a]. The sixth law is more related to the fact that finiteness of the implemented software implies that its properties are bounded relative to those of the application and its domain. Properties excluded by the bounds eventually become a source of performance limitations, irritation and error to be eliminated.

Thus, the two laws are due to different causes, reflecting distinct phenomena. It is often difficult to determine which changes or additions to a system are related to law I and which to law VI. Treating them jointly appears to be appropriate for the purpose of this paper. They may, conveniently, be considered together.

In selecting COTS software for integration into a system one must be clearly concerned with the adequacy of its behaviour and functioning after its integration into the operational system. This will be assessed on the basis of a functional specifications, substantiated during the various process steps and finally confirmed during system validation. Precise data on interface, representational and linguistic characteristics will be of major concern in designing and implementing the system within which the COTS is embedded. Other functional and non-functional factors too will be of concern. Here we focus on the set of assumptions, explicit and implicit, embedded in the COTS software.

As the application, the system and the domain change, as per the first and sixth laws, adaptation of the COTS software to the new operational environment will often be required. This may be achieved by changing the COTS software locally, by implementing more difficult, complex and obscure changes to the embedding system, the software, hardware or application, by requesting the COTS developer<sup>1</sup> to implement the necessary changes, by obtaining a new COTS from an alternative COTS developer or by in-house implementation of the component that was originally a COTS. We will not attempt a detailed examination of all these choices and will comment only in those ones which appear to be more likely. Local adaptation increases the maintenance burden beyond that which would be required were COTS not used from the start on. It also means that responsibility for its future maintenance moves to the COTS integrator. The link with the COTS developer, the support provided by the developer and the link with other COTS integrators using the same system will be weakened or severed. This represents a major negation of the justification that led to selection of the COTS software in the first place. When

---

<sup>1</sup> In this paper the term COTS developer refers to the individuals developing and maintaining, that is the evolution of the COTS product and vendor to the organisation marketing it. COTS integrator refers to individuals and organisations that obtain COTS for their use as components as part of a larger software system, and, finally, end-users refer to users of the latter.

the COTS software is upgraded by the development organisation serious delays may occur between identification of a need for modification or addition and its implementation. Moreover, there are likely to be conflicts between the needs and priorities of different COTS integrators. Individual organisations will still have to implement local patches to the COTS or the embedding system software, a self defeating exercise both short term and long term and one likely to lead to the introduction of new faults. And if and when the COTS developer does upgrade the software at source in response to multiple COTS integrator requests (or for its own reasons), the upgrade may not be installed into host systems without undoing or redoing patches and carefully revalidation. In practice, if the benefits of using COTS are to be preserved, the most reasonable choice is to stay with the original COTS code. The first and sixth laws suggest that that is not feasible. Thus, the introduction of COTS is likely to raise the maintenance burden and to introduce serious quality, reliability and maintenance issues. This is so even when no changes to upgraded and reinstalled COTS software appear, initially, to be required, since evolution of the embedding system will require careful examination of all embedded COTS to verify this.

All of these problem sources reflect one common phenomenon, the invalidation of assumptions embedded in the application system and its embedded COTS software (of whatever proportion); the phenomenon that underlies the Principle of Uncertainty. If this is true for locally produced, non COTS, software as demonstrated by incidents exemplified by of sinking of HMS Sheffield, Airbus 320 crashes, the destruction of Ariane 501, the imminence of Sept 9th, 1999, current concern with the Year 2000 and so on<sup>2</sup>, how much more so for COTS. COTS developers cannot possibly be aware of assumptions to be made *in the future* by integrators. Nor can the latter be aware of all assumptions reflected in a COTS item unless these are fully documented, and that is impossible since assumptions are relative to the operational domain in which the item is used. Thus the use of COTS, while initially attractive, is likely, in the long run, to prove burdensome, expensive, unreliable and a security risk.

### 4.3 The Second Law

**Increasing Complexity (II):** *As an E-type system evolves its complexity increases unless work is done to maintain or reduce it.*

As already stated, if a COTS element is completely specified, it may, in isolation, be regarded as of type S. Each new release, developed in response to COTS integrator requests and/or COTS developer self interest, would then be regarded as a new program. A series of such releases sequentially integrated into customer systems will however be of type E and will conform to the law since they result from a mix of requirements from a variety of sources with independent, possibly orthogonal, needs. The consequent increase in complexity will be reflected in a growing maintenance burden as discussed in section 4.2.

The increase in complexity due to the volume and orthogonal nature of customer requests means that the COTS developer will find it increasingly difficult and expensive to respond to COTS integrator needs in timely fashion with a high quality product. The implications of this to the integrators are self evident.

### 4.4 The Third, Fourth and Seventh Laws

**Self Regulation (III):** *E-type system evolution processes are self regulating.*

**Conservation of Organisational Stability** (invariant work rate) (IV): *The average achievable evolution activity rate in an evolving E-type system tends to be non increasing over product lifetime, unless feedback control mechanisms are appropriately adjusted.*

**Declining Quality** (VII): *The quality of E-type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes.*

These three laws apply specifically and directly only to the COTS development organisation. They, nevertheless, also impact COTS integrators and their maintenance efforts since, as in section 4.2, they will directly experience the increasing difficulties and costs that the COTS supplier faces in responding to customer needs in timely fashion with a high quality product. The COTS vendor may well become ever more reluctant to respond to specific proposals or requests.

---

<sup>2</sup> It is, unfortunately, not possible to go into detail here but underlying all these "incidents", past and future, lie assumptions, initially justified, that ultimately became or will become invalid.

## 4.5 The Fifth Law

**Conservation of Familiarity (V):** *As an E-type system evolves all associated with it must maintain mastery of its content and behaviour to achieve satisfactory usage and evolution. Excessive growth diminishes that mastery and leads to a transient reduction in growth rate or even shrinkage. Therefore, the mean incremental growth remains constant or declines.*

Although the very first statement of this law [leh80a] claimed that the average incremental growth remained constant, phenomenological analysis suggests that it might increase as a consequence of increasing familiarity with the application and the system, improved technology and improving skills. It might, equally, decline as a consequence of the introduction of fresh and relatively less skilled personnel, their unfamiliar with the system, its concepts and the assumptions that underlie it, increasing complexity and distance (orthogonality of) changes from the system concepts and architecture. Current results from FEAST/1 suggest that it probably declines steadily as the system evolves. Thus it appears that the latter effects dominate. Results also seem to support the original observation [leh80a] that increments which infringe some threshold tend to be followed by a small or even negative increment(s) which is a direct consequence of the fifth law [leh98c].

This law does not appear to have direct implications for COTS integrators other than that it limits the rate at which a vendor can respond to their needs and requests. It also suggests to them that excessive demands in terms of needs or the timescale within which they must be met will, if apparently satisfied, lead to a decline in quality, an increase in fault rates and delays in satisfying subsequent requests.

## 4.6 The Eighth Law

**Feedback System (VIII):** *E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement for other than the most primitive processes.*

This law provides a statement of fact. Its implications and its potential for process improvement are currently being explored in the FEAST/1 project [leh96]. It identifies the nature of the phenomena and mechanisms that give rise to the other seven laws. It does, however, have a direct implications on the use of COTS in terms of the design and improvement of software processes which involve, or are intended to involve, COTS software.

An essential property of a complex feedback system is that it develops a dynamics of its own [leh98b,c, wer98]. Once COTS has been integrated into some larger application system involving but not limited to computers, the COTS developer organisation becomes a part of the larger system. Its dynamics may influence that of the overall system and impacts the COTS integrator and end-users. How? The COTS integrator may be forced to accept releases with unrequired or even undesirable changes. Required changes may be delayed or even denied due to the COTS developers' preoccupation with work for other COTS integrators or the risk of their disappearance from the market place. Changes in technology [hyb97] or limitation in the COTS developers organisation will feed through to the COTS integrator. Process designers and managers must take the feedback nature of the process and its dynamics into account in planning, managing and attempting to improve the development and evolution of their software systems. Failure to do so will lead to the serious long term quality and cost problems associated above with the individual laws.

## 5 Conclusion

The wider use of COTS, and the reuse of locally developed software components, is widely regarded as one of the answers to the current need for never-ending maintenance of installed software. As argued elsewhere [leh69,85,98a], the reasoning applied here suggests that the use of COTS may provide short term effectiveness but introduce long term problems, intensifying the very phenomenon it was intended to control. COTS functional and interface definition is not enough. At the very least application and domain bounds and all other assumptions that can be captured must be fully and freely available to all COTS integrators and end-users, individual and corporate.

Recognition of the global process as a feedback system suggests that when adopting the use of COTS components, organisations *must* concern themselves with the total system involved in the development. This recognises that in the context of software systems using COTS, the COTS development and evolution organisation must be treated as integral part of the development organisation. The implications of this and methods by which this can be achieved are under study in the FEAST/1 project with tools being applied,

developed or in the concept stage. As testified by the references below, and by results already obtained but not yet published, the phenomena reflected by the laws are real and probably universal. Partial, at least, solutions are still some way off.

One major addition to current development practice can be suggested. Given the inevitable embedding in any system of a mass of assumptions about the real world application being addressed and the operational domain in which it is applied<sup>3</sup>, it follows that such assumptions should be captured, recorded in accessible fashion and reviewed whenever a change to the system, to the domain or to application practice is made. None of these actions are reflected in current industrial practice or in their process models. Taking the capture, recording and regular review of assumptions seriously must become a high priority. Procedures for monitoring and evolving assumptions must be developed. Such procedures must record a mapping between assumptions and software elements<sup>4</sup> so that the parts affected by the invalidation of any assumption may be rapidly identified, the consequences assessed and reliable corrective action implemented. Making developers more aware of the existence and implications of assumptions will bring immediate pay off. It is an activity that must be adequately supported.

Some evidence suggests that forces outside the technical evolution process drive and constrain software system evolution [leh98c] There are good reasons to believe that, for COTS-based software such external forces may be even stronger, introducing into the evolution situation such factors as technology change and cycles [hyb97]. This poses an extra challenge and load on those in charge of managing and performing evolution. Further studies are needed to fully understand these issues and to provide tools and methods appropriate to this new situation. Development of such methods and tools will follow but will prove neither trivial nor simple, if only because so many assumptions (if not most?) are implicit, or made on the fly by individuals. Such tools are, however, essential if mankind is to survive its ever growing dependence on computers, that is on software; if society, individually and collectively, is to retain control of its own destiny.

Overlooking or ignoring phenomena that unquestionably exist, that have been recognised and that are increasingly being unified in a body of knowledge and understanding about the software process can only be harmful. If the nature of the problem is fully understood, its consequences appreciated, appropriate technologies and methods developed and rigorously applied and adequate precautions taken, it would appear certain that the widespread use of COTS and the trend to reuse of in-house software will eventually prove advantageous; that the benefit resulting therefrom will exceed any risks or negative effects that such practice can bring with it. For the moment the message must be "Proceed with caution, forget the panacea".

## 6 References - references identified by an \* may be found in [leh85]

- [hyb97] Hybertson, DW, Anh DT and Thomas WM, *Maintenance of COTS-intensive Software Systems*, Soft. Maintenance: Research and Practice, Vol. 9, 203-216, 1997, pp. 203-216
- [kon96] Kontio, J. A, *Case Study in Applying a Systematic Method for COTS Selection*, Proc. ICSE 18, IEEE CS Press Order No PR07246, March 25-29, 1996, Berlin, Germany, pp. 201-209
- [leh74]\* Lehman MM, *Programs, Cities, Students, Limits to Growth?*, Inaugural Lecture, May 1974. Publ. in Imp. Col of Sc. Tech. Inaug. Lect. Ser., vol 9, 1970, 1974, pp. 211 - 229. Also in *Programming Methodology*, (D Gries ed.), Springer, Verlag, 1978, pp. 42 - 62
- [leh78]\* *id.*, *Laws of Program Evolution - Rules and Tools for Programming Management*, Proc. Infotech State of the Art Conf., Why Software Projects Fail, - Apr. 1978, pp. 11/1 11/25
- [leh80a]\* *id.*, *On Understanding Laws, Evolution and Conservation in the Large Program Life Cycle*, J. of Sys. and Software, v. 1, n. 3, 1980, pp. 213 - 221
- [leh80b]\* *id.*, *Programs, Life Cycles and Laws of Software Evolution*, Proc. IEEE Special Issue on Software Engineering, v. 68, n. 9, Sept. 1980, pp. 1060 - 1076
- [leh84] Lehman MM, Stenning V and Turski W M, *Another Look at Software Design Methodology*, ICST DoC Res. Rep. 83/13, June 1983. Also, *Software Engineering Notes*, v. 9, no 2, April 1984, pp. 38 - 53

---

<sup>3</sup> It is estimated that there is one such assumption for every ten or so lines of source code. No verification by actual count is known to the authors. But even if in error by a factor of two or more, the resultant number of embedded assumptions is such that for any system other than a trivial one, a significant amount of rework is needed to maintain validity of the assumption set and, therefore, user satisfaction.

<sup>4</sup> That is, the different models of the application, including requirements, specifications, high-level and low-level designs, code [leh84], support tools and CASE tools. All contain embedded assumptions which may affect the characteristics of the final delivered software.

- [leh85] Lehman MM and Belady LA, *Program Evolution - Processes of Software Change*, Academic Press, London, 1985, 538 p.
- [leh89] Lehman MM, *Uncertainty in Computer Application and its Control Through the Engineering of Software*, J. of Software Maintenance: Research and Practice, v. 1, n. 1, Sept. 1989, pp. 3 - 27
- [leh90] *id*, *Uncertainty in Computer Application*, Tech. Let., CACM, v. 33, n. 5, May 1990, pp. 584 - 586
- [leh94] *id*, *Feedback in the Software Evolution Process*, Keynote Address, CSR Eleventh Annual Workshop on Software Evolution: Models and Metrics, Dublin, 7 - 9th Sept. 1994, Workshop Proc., Information and Software Technology, sp. is. on Software Maintenance, v. 38, n. 11, 1996, Elsevier, 1996, pp. 681 - 686
- [leh96a] *id*, *Laws of Software Evolution Revisited*, Proc. EWSPT'96, Nancy, 9 - 11 Oct. 1996
- [leh96b] Lehman MM, Perry DE and Turski WM, *Why is it so Hard to Find Feedback Control in Software Processes?*, Invited Talk, Proc. of the 19th Australasian Comp. Sc. Conf., Melbourne, 31 Jan - Feb 2 1996. pp. 107 - 115.
- [leh96c] Lehman MM and Stenning V, *FEAST/I: Case for Support*, DoC, Imperial College, London, March 1996, <http://www-dse.doc.ic.ac.uk/~mml/feast1>
- [leh96d] *id*, *Process Improvement - The Way Forward*, Invited Keynote Address, Proc. Brazilian Software Engineering Conference, 14 - 18 October 1996, pp. 23 - 35
- [leh97a] Lehman MM, Perry DE, Ramil JF, Turski WM and Wernick PD, *Metrics and Laws of Software Evolution - The Nineties View*, Proc. Metrics '97, Albuquerque, NM, 5 - 7 Nov., 1997. Also as *Process Improvement - The Way Forward*, in *Elements of Software Process Assessment and Improvement*, IEEE CS Press, 1998
- [leh97b] Lehman MM, *Process Models - Where Next?*, "Most Influential Paper of ICSE 9 Award", Proc. ICSE 19, Boston, 20 - 22 May 1997, pp. 549 - 552
- [leh98a] *id*, *Feedback, Evolution and Software Technology - The Human Dimension*, Proc. Workshop on Human Dimensions in Successful Software Development, ICSE 20, April 21, 1998, Kyoto, Japan
- [leh98b] Lehman MM and Wernick PD, *System Dynamics Models of Software Evolution Processes*, Proc. Int. Wrkshp. on the Principles of Software Evolution, ICSE 20, Kyoto, Japan, April 20 - 21, 1998
- [leh98c] Lehman MM and Ramil JF, *The Impact of Feedback in the Global Software Process*, ProSim'98, Proc. Int. Wrkshp. on Softw. Proc. Simulation Modelling, June 22 - 24, 1998, Silver Falls, Oregon, to appear, Journal of Systems and Software, 1999
- [leh98d] Lehman M M, Perry D E, Ramil J F, *On Evidence Supporting the FEAST Hypothesis and the Laws of Software Evolution*, to appear, Proc. Metrics'98, Bethesda, Maryland, Nov. 20-21, 1998
- [leh98e] Lehman M M, Perry D E and Ramil J F, *Implications of Evolution Metrics on Software Maintenance*, to appear, Proc. Int. Conf. on Soft. Maintenance (ICSM'98), Washington DC, Nov.16-24, 1998
- [sul97] Sullivan KJ, Cockrell J, Zhang S. and Coppit D., *Package-Oriented Programming of Engineering Tools*, Proc. ICSE 19, IEEE CS Press, Ord. no. PR07816, March 17-23, 1997, Boston, MA, pp. 616-617
- [tur96] Turski WM, *Reference Model for Smooth Growth of Software Systems*, IEEE Trans. on Soft. Eng. v.22, n.8, Aug. 1996
- [wer98] Wernick PD and Lehman MM, *Software Process White Box Modelling for FEAST/I*, ProSim'98, Proc. Int. Wrkshp. on Softw. Proc. Simulation Modelling, June 22 - 24, 1998, Silver Falls, Oregon, to appear, Journal of Systems and Software, 1999