

EPICS: Evolution Phenomenology in Component-intensive Software

M M Lehman J F Ramil
Department of Computing
Imperial College
180 Queen's Gate, London SW7 2BZ
tel. +44-20-7594 8214 fax. +44-20-7594 8215
{mml,ramil}@doc.ic.ac.uk

ABSTRACT

This paper briefly justifies and outlines initial plans for an empirical investigation into the evolution of component-intensive software systems. Questions to be addressed include whether the findings and observations of traditional software evolution studies garnered over the last 30 years or so, and most recently as part of the FEAST projects, apply to and are relevant in the context of component-intensive software. The goal of the suggested investigation is to contribute towards disciplined planning and management of long-term evolution of component-intensive software. In submitting this paper to the WESS 2001 workshop, the authors are presenting initial plans to obtain early exposure of the ideas and constructive criticisms from the empirical studies community.

Keywords

Component-Based Software Engineering, Component-Intensive Software, COTS, Empirical Studies, Evolution Planning and Management, Maintenance, Metrics of Software Evolution, Process Improvement, Reuse, System Dynamics.

1. INTRODUCTION

Software evolution processes, including those of software change, encompass all activities required to maintain stakeholder satisfaction over the operational life of a software system. The objectives of such activities include fixing, adaptation, and enhancement of the software. This all-inclusive view of software evolution encompasses what has otherwise been termed *maintenance*. The need for continuous evolution is, to a lesser or greater extent, relevant to *all* software that addresses real-world applications. In a world increasingly dependent on computers and, therefore its software, the nature, impact and management of evolution has great social and economic importance [leh85,01a].

The term *evolution* tends to be interpreted and studied in two separate and distinct ways. The more common approach sees the most important issues as those concerning the methods and means whereby evolution may be achieved. The focus of this approach is on the *how* of software evolution and includes both methods and tools to achieve the desired results in a systematic and controlled manner. Both *ab initio* development from conception to operational realisation and adaptation and extension of a system to be more satisfactory in a changing operational environment are considered, with far more attention being applied in research to generation rather than to change. This despite the fact that in industry more investment is made in change than in *ab initio* development. Work exemplifying this view has been presented in a series of international meetings on Principles of Software Evolution [ispse] and in a recent session on Formal Foundations of Software Evolution [ffse]. A complementary view has, however, also been taken. This, less frequently encountered but

nevertheless important, view is concerned with the *what* and the *why* of evolution. It examines the nature of the evolution phenomenon, its causes, drivers and impact. The approach is restricted to a much smaller community [e.g. leh74,leh85,94, kem99,raj00] and is based on the view that more insight into, and better understanding of evolution as a *phenomenon* will itself lead to improved methods for its planning, management and implementation. It will, for example, help identify the areas in which research effort is most likely to yield significant benefit. The FEAST/1 (1996-1998) and /2 (1999-March 2001) projects [leh96,98a] exemplify an investigation into the what and the why of evolution. In general, findings have strengthened the relevance of the view encapsulated in the FEAST hypothesis [leh94]. This states that *E*-type software evolution processes are multi-agent multi-level multi-loop feedback systems and must, therefore, be treated as such during planning and management of the process if sustained improvement of that process is to be achieved [leh94,feast].

The FEAST projects had as its broad objectives [leh96,98a] the empirical study of evolutionary attributes of products and processes based, *inter alia* on the construction of *black-box* [e.g. leh98b,ram01] and system dynamics [for61] *white-box* [e.g. kah00] models. These models were to reflect attributes of long-term software system evolution, including growth trends and evolution rates. Results strengthened the support for and understanding of the evolution phenomenology consolidated over the years [e.g. leh74,85,cho80]. That understanding led to the *SPE* program classification [leh80b,85,feast], the laws of software evolution [leh74,78,80a,b,85,97,feast] and a principle of software uncertainty [leh89,90]. Practical outcome of the project is exemplified by a set of rules and guidelines for software evolution planning and management [leh00a]. Together with the work of others in the area [e.g. kem99, raj01], the body of knowledge and the understanding achieved appears to offer the basis for the development of a formal theory of software evolution [leh00b]. A proposal for such a development is currently being prepared [leh00c] and, it is hoped, will run in parallel with the one [leh01b] for which preliminary plans are outlined in the present paper. The latter will aim at the empirical study of evolution in the component-intensive software and related domains, domains arousing widespread and increasing interest.

2. THE MOVE TOWARDS COMPONENT-INTENSIVE SOFTWARE

Software development by assembly of a set of mass produced components was already discussed in the sixties [nau69]. In advancing the concept the parallel with other engineering fields, where the concept of components is almost universally accepted and applied was frequently invoked. Nuts, bolts and a wide variety of electrical connectors and components exemplify internationally agreed standards. Such standard components can be obtained from numerous suppliers and used as building blocks

in different and unique products. It was, however, not until recently that the practical application has been widely explored in practice [bro98,dso99]. A recently coined term, *component-based software engineering* (CBSE) [bro98,iee00], reflects the current trend towards the use of components, developed either in-house or as commercial off-the-shelf software (COTS) [e.g. mck99,car00,mor00,myr99] units. CBSE is widely seen as a way of reducing some of the problems so often encountered, especially in *ab initio* software development. In particular, the use of COTS is seen, as a means to achieve increased productivity and reliability, and to decrease the delivery time for large-system implementation. To what extent these expectations are being fulfilled, in particular, in the context of long-term evolution, is an open question that, together with related issues, deserves empirical investigation.

The majority of empirical studies, including FEAST, have focused on software constructed and evolved using traditional paradigms, which did not make significant use of externally supplied components. But the evolution phenomenon is likely to re-appear as the newly emerging paradigms are applied. They must, for example, be expected to emerge, even dominate, in the areas of component-intensive software and their processes [leh00d]. Such systems are acquiring ever greater social and economic importance. Thus the question whether the findings of studies of traditional software domains are also relevant in the context of component-intensive software is of considerable, perhaps vital interest, as is the question of their extent. This paper presents preliminary plans for an investigation of this and related issues in a project to be called EPICS (*Evolution Phenomenology in Component-intensive Software*).

3. HYPOTHESES FOR EMPIRICAL INVESTIGATION

Eight *laws* of software evolution, as exemplified below, have been discussed in the literature [leh74,78,80a,b,85,97,feast]. Six of these have been generally supported (with minor modifications) by the study and interpretation of data gathered in FEAST. They provide phenomenological descriptors of evolutionary behaviour observed over the years in a number of software systems. The term *laws* was deliberately selected to indicate that they address and reflect forces rooted more in cognitive, organisational and societal mechanisms than in the specific software technology (e.g. languages, other tools, process models) being applied. Thus, from the point of view of individual stakeholders the phenomena reflected in the laws are outside their control. The phenomena appear to them as reflecting inescapable behaviour, hence the use of the term *laws*.

The laws provide qualitative descriptors that relate directly to the evolutionary behaviour of *E*-type software, that is, software used to solve a problem or address an application in a real world domain. Real world software, that is software of type *E*, is ultimately judged by stakeholder satisfaction with the results of its execution [leh80b,85]. Several papers [leh80b,85,feast] have discussed the fact that any *E*-type computer application must undergo continuing evolution if stakeholder satisfaction is to be maintained. The fact that the system implementing the application includes a significant number of mass-produced components does not affect this fundamental truth. Thus, one may expect the laws to be relevant in the context of component-based software engineering, though their statement, the phenomenology they reflect, may have to be refined in the light of experience and as supported and typified by empirical evidence.

A recent paper [leh00d] examines the most immediate implications of the laws in the context of component-intensive

systems, discusses their potential managerial impact and provides some preliminary recommendations. By so doing, the paper provides a reasoned set of questions and hypotheses to be further developed and empirically investigated. The proposed investigation is aimed at empirical investigation of the latter. As an example, one of these aspects is briefly discussed below.

3.1 Hypothesis: Complexity of the Software and its Impact

The second law of software evolution, "*Increasing Complexity*", [leh74] states that "As an *E*-type system evolves its complexity increases unless work is done to maintain or reduce it". It appears that, in principle, the introduction of components will apparently reduce the constraints implied by the second law. Why? The task of building a complex system is now virtually shared in a classical "divide-and-conquer" approach by component builders, who provide and evolve components, on the one hand, and integrators, on the other, with the market (and even specialised brokers¹) as mediator. The constraints implied by the 2nd law are expected, however, to re-emerge both at the level of the individual component and of the host system. The brief discussion below closely follows that in [leh00d]. The reader is referred to that paper for further details.

3.1.1 Individual Component Level

Individual components will be seen to follow the law since they must be adapted to satisfy a mix of changed requirements from a variety of sources with independent, possibly orthogonal, needs and assumptions. Unless work is done to control component complexity, the resultant increase will be reflected in a growing maintenance burden leading to a decline in maintenance productivity and response time, of the component supplier's organisation. Inevitably, knock on effects will impact on the integrator's organisation.

Complexity growth due to the volume and likely orthogonal nature of successive changes means that the component supplier may find it increasingly difficult and expensive to respond to component user needs in timely fashion with a high quality product. In particular, once a component is marketed, the supplier may not have sufficient incentives to invest significant effort in complexity reduction and other clean-up or re-engineering, even if the cost is shared by its customers. In the face of market and competitive pressures and with a focus on short-term profits, what will become the general accepted practice, remains to be seen. For example, in a market dominated by component suppliers, the latter may prefer to invest in refinement and extension of their products to attract new customers, rather than in cleaning-up existing products. Indeed, circumstances may force suppliers to withdraw support from individual components and offer a replacement which, despite a claim of upward compatibility, will inevitably involve new assumptions that may conflict with the host system. A whole series of new problems will emerge as both the component based, that is, the host system, and the components on which it relies are required to evolve.

3.1.2 Host System Level

It may be that the introduction of components rather than the use of traditional paradigms, will lead to faster and more effective preparation of the initial version of a host system. Thus, the introduction of components will appear to reduce the constraints implied by the second law. However, increased capability and growing expectations tend to go hand-by-hand. This, and other factors, will lead to an increase in the functionality and size of software systems. As an increasing number of components are used, each is effectively a primitive in a language defined by the

¹ Suggested in a recent presentation (EWSPT 8) by N. Madhavji.

full set of component units, with the interfaces providing the syntax of their use. Thus construction of the systems is based on a form of very high level programming with component characteristics as its primitives. Hence, one must expect many of the behavioural characteristics of the classical programming domains to reappear in the new domain. The behaviour and constraints implied by the second law will eventually reappear, though at a higher level of abstraction, the constraints implied by the law are difficult to avoid.

The preceding is a preliminary and incomplete analysis, presented here to suggest that there are issues related to the long term evolution of component-intensive systems that are worthy of investigation and to the resolution of which empirical work can make a significant contribution. Other hypotheses that have been advanced, for example, in [hyb97,leh00d] will be taken into account in conducting an investigation based on a refined version of the present preliminary plans. As in previous studies of software evolution the main goal of the proposed investigation will be to advance understanding and mastery of the phenomenon.

4. THE INVESTIGATION

The following activities are foreseen as part of the EPiCS investigation:

- (i) develop instruments and tools for the collection of qualitative and quantitative data in a number of industrial organisations to address the hypotheses put forward
- (ii) obtain empirical, qualitative and quantitative, data sets representative of the evolution of component-intensive industrially developed systems
- (iii) develop case studies of a number of evolving component-based applications, representing widely different domains. major events, such as system restructuring, process changes and team reorganisations, from inception, highlighting significant events and factors at play in the long term evolution
- (iv) in parallel, and as soon as appropriate data is available, apply conceptual and modelling methods analogous to those developed and successfully pursued in FEAST to the new data sets
- (v) calibrate the models built in the previous step and derive phenomenological predictions
- (vi) test the phenomenological hypothesis put forward in [hyb97,leh00d]
- (vii) document the results including observed characteristics, trends and patterns
- (viii) based on the above results derive a set of rules, tool proposals and guidelines for the evolution of component-intensive software [leh00a].

5. GENERAL APPROACH

The project will start with data collection from industrially evolved component-based applications in collaborator's organisations. Both, *quantitative* and *qualitative*² data, should be available [mor00]. The first will be empirical data reflecting a set of metrics of interest. Their selection will be inspired by the metric sets used in the study of evolution systems but adapted as necessary to what is available and to the need to address the

² The gathering of qualitative data, along with quantitative, in empirical studies of software evolution was one of the suggestions that emerged during a recent workshop [fesbp].

hypotheses of interest [leh00d]. Once collected, such data will provide a basis for both black box [e.g. cle93,han96] and white box modelling. As in FEAST, system dynamics process modelling [for61,sen90] will be applied to the latter wherever possible.

The second, qualitative data [sea99], will include results of interviewing key personnel, natural language descriptions of the evolution processes from their inception, etc. Such data will provide a set of *qualitative* descriptions of the phenomena and facilitate the identification of evolutionary mechanisms and of the major drivers underlying the evolutionary characteristics of the individual systems studied. Ideas inspired in the search for grounded theory [bar67] may inform such identification. The goal is to achieve a set of empirically based qualitative models [wol85] of the process. One aspect worthy of further attention is identification of stages, described in [raj00], in the evolution of software.

As the empirical data gathering progresses, quantitative (black box and white box) and qualitative process views will be developed, validated, checked for cross consistency and completeness and, when possible, merged. As in FEAST, models will be refined when possible following a top-down process inspired in [zur67]. This will necessarily proceed in an iterative fashion, prompt revision of the data sets and probably suggesting additional attributes. It is expected to complete more than one cycle during the investigation.

Results will be compared to the phenomenological invariants suggested by the *SPE* classification, the laws of software evolution and, in particular, with the hypotheses put forward in [leh00d]. The planned outcome will, hopefully, be a phenomenology of long term component-intensive software evolution and an empirically based set of rules and guidelines for the long-term evolution of such applications. Tool concepts may also emerge.

The concepts and methodological tools to be utilised in the suggested investigation are expected to be to a great extent analogous to those used and refined during the FEAST studies e.g. [leh96,98a,ram01], though differences between traditional and component-intensive domains must be taken into account. Modelling and characterisation of attributes beyond the immediate software process, such as those of *co-evolving* processes and domains [soce], e.g. business and organisational, will be given particular attention in the EPiCS study. Other expected differences are in the area of metric definitions, in particular with regards to the search for metrics that will usefully reflect evolutionary attributes of the process and the product. Much of the work in software metrics assumes that there is access to the software artefacts (e.g. the code) in full [e.g. fen97]. Since in many respects components are black boxes from the point of view of the component integrators, metrics that relate to them may be unobtainable.

Some challenges may already be identified. For example, it is unlikely, in the next several years, that metric data sets covering extensive periods of time will be available. So the small size of the data sets, already a challenge in FEAST, is likely to be an even greater challenge in EPiCS. This may not be so serious a limitation since one may expect faster and more frequent release cycles in the component-intensive process. This would compensate for the relatively recent adoption of the component-intensive paradigm. Another challenge emerges from the size of the integration teams. Since the teams involved in component-intensive processes tend to be smaller in size than those in traditional development, the role of the individuals may be more significant in determining evolutionary behaviour. Thus, one may expect that quantitative regularities in evolutionary attributes such

as growth trends and evolutionary rate (e.g. in counts of modules handled [leh85]) may exhibit a higher variance with, in general, less well defined trends than those where large teams are involved. This is one reason why collection, modelling and interpretation of qualitative data is crucial to the study.

6. RELATED WORK

Recent discussion of component-intensive software engineering [e.g. bro98,dso99,kon96,wu00] has focused on principles and techniques for the construction of re-usable components and component-based systems [e.g. hal97,jac97], on the processes followed in CBSE [mor00] and on the impact of its adoption on other organisational processes [mck99], all aspects of the *how*. The *what* and the *why* of the evolution of component-intensive software has received much less attention. Included in this omission are the evolutionary attributes of component-intensive products and processes. As suggested in the eighties and supported by the FEAST findings [feast], the evolution of software is inevitable, whatever the implementation paradigm. It is related to and driven by the evolution of the applications, that is, of the use to which the system is put [leh01a]. Application evolution may be achieved by changing the software to support the new functionality or performance requirements. Alternatively, these may be provided by throwing away the old system and providing a new one that has the desired characteristics [voa99]. In either event, the achievement of an empirically-based understanding the way in which component-intensive applications evolve is essential if effective, disciplined and timely evolution is to be achieved, and in particular, to facilitate planning, direction, management and control of component technology and application.

7. REUSE

A trend complementary to the use of components relates to *software reuse*, that is the use in some system of a software component originally developed for some other system or systems [jac97]. Its application and, in particular, its increasing adoption, is considered by some [hal97] to be more effective in the long term, at least in some respects, than development of new code. However, extensive reliance on reuse introduces new problems. Some of these are similar to the problems that arise in component-intensive software and are implicitly covered in EPICS. Those which are exclusive of reuse are beyond the scope of the envisaged investigation.

8. FINAL REMARKS

The present paper discusses plans for an empirical investigation into the *what* and *why* of the evolution of component-intensive software. The main goal of the suggested investigation is to contribute towards disciplined planning and management of long-term evolution of component-intensive software. A funding proposal to carry out the investigation is currently being drafted. By submitting this paper, the authors are presenting their initial plans to the WESS 2001 workshop, seeking the early exposure of these ideas to the empirical studies community and possible partners. They welcome comments, suggestions and pointers to related work, completed or in progress.

9. REFERENCES³

[bar67] BARNEY, G.G. and STRAUSS, A.L.: 'The Discovery of Grounded Theory: Strategies for Qualitative Research', Aldine de Gruyter, 1967

- [bro98] BROWN, A. and WALLNAU, K.: 'The Current State of Component-Based Software Engineering (CBSE)', IEEE Software, September 1998, pp. 37-47
- [car00] CARNEY, D., HISSAM, S.A. and PLAKOSH, D.: 'Complex COTS-based Software Systems: Practical Steps for Their Maintenance', J. of Softw. Maintenance: Res. and Pract., vol. 12, 2000, pp. 357 - 376
- [cho80] CHONG HOK YUEN, C. K. S.: 'Phenomenology of Program Maintenance and Evolution', PhD thesis, Dept. of Comp., Imperial College, 1981
- [cle93] CLEVELAND, W.S.: 'Visualizing Data', Hobart Press, Summit, NJ, 1993, pp. 360
- [dso99] D'SOUZA, D. and WILLS, A.C.: 'Objects, Components, and Frameworks with UML: The Catalysis Approach' (Addison-Wesley, Boston, Ma. 1999)
- [feast] FEAST: 'Feedback, Evolution And Software Technology' Web Site including selected pubs. <http://www-dse.doc.ic.ac.uk/~mml/feast> <as of Aug. 2001>
- [fen97] FENTON, N.E. and PFLEEGER, S.L.: 'Software Metrics: A Rigorous and Practical Approach', Int. Thomson Comp. Press, London, 1997, pp. 638
- [fesbp] FEAST 2000: 'Intl. Workshop on Feedback and Evolution in Business and Software Processes'. 10 - 12 July 2000, Imp. Col., London. Pre-prints available from <http://www.doc.ic.ac.uk/~mml/f2000>
- [ffse] FFSE 2001: 'Intl. Special Session on Formal Foundations of Software Evolution', 13 March 2001, Lisbon, <http://prog.vub.ac.be/poolresearch/FFSE/FFSE-Workshop.html> <as of August 2001>
- [for61] FORRESTER, J.W.: 'Industrial Dynamics' (MIT Press, Cambridge MA, 1961)
- [hal97] HALLSTEINSEN, S. and PACI, M.: 'Software Evolution and Reuse'. Springer Verlag, Berlin, 1997, 293 pp.
- [han96] HAND, D.J. and CROWDER, M.: 'Practical Longitudinal Data Analysis', Chapman & Hall, London, 1996, pp. 232
- [hyb97] HYBERTSON, D.W., ANH, D.T. and THOMAS, W.M.: 'Maintenance of COTS-intensive Software Systems,' Softw. Maint.: Res. and Pract., 1997, 9, pp. 203-216
- [iee00] IEE Proceedings - Software: 'Special Issue on Component Based Software Engineering', v. 147, n. 6, Dec. (2000).
- [ispse] ISPSE 2000. 'Proc. of the Intl. Symp. on Principles of Softw. Evolution', Nov. 1-2, 2000, Kanazawa, Japan
- [jab97] JACOBSON, I., GRISS, M. and JONSSON, P.: 'Software Reuse - Architecture, Process and Organisation for Business Success' (Addison-Wesley, 1997, 560 pps.)
- [kah00] KAHEN, G., LEHMAN, M.M., RAMIL, J.F. and WERNICK, P.D.: 'An Approach to System Dynamics Modelling in the Investigation of Policies for E-type Software Evolution.' ProSim 2000, 12-14 July 2000, Imperial College, London UK. A revised version to appear in J. of Syst. and Softw., vol. 15, 2001
- [kem99] KEMERER, C.F. and SLAUGHTER, S.: 'An Empirical Approach to Studying Software Evolution', IEEE Trans. on Softw. Eng., vol. 25, n. 4, July/August 1999, pp. 493 - 509
- [kon96] KONTIO, J.A.: 'Case Study in Applying a Systematic Method for COTS Selection'. Proc. ICSE 18, 25-29 March 1996, Berlin, pp. 201-209

³ An "*" indicates that the paper has been reprinted in [leh85].

- [leh74] *LEHMAN, M.M.: 'Programs, Cities, Students, Limits to Growth?' Inaugural Lecture, May 1974. Publ. in Imp. Col. of Sc. Tech. Inaug. Lect. Ser., 9, 1970 - 1974, pp. 211-229. Also in Programming Methodology, D Gries (ed.), Springer Verlag, 1978, 42-62
- [leh78] *LEHMAN, M.M.: 'Laws of Program Evolution - Rules and Tools for Program Management'. Proc. Infotech State of the Art Conf., Why Software Projects Fail, - April 1978, 11/1-11/25
- [leh80a] *LEHMAN, M.M.: 'On Understanding Laws, Evolution and Conservation in the Large Program Life Cycle', J. of Sys. and Software, 1980, 1, (3), pp. 213-221
- [leh80b] *LEHMAN, M.M.: 'Programs, Life Cycles and Laws of Software Evolution'. Proc. IEEE Special Issue on Softw. Eng., 68, (9), Sept. 1980, pp. 1060-1076
- [leh85] LEHMAN, M.M. and BELADY, L.A.: 'Program Evolution, - Processes of Software Change', (Acad. Press, London, 1985)
- [leh89] LEHMAN, M.M.: 'Uncertainty in Computer Application and its Control Through the Engineering of Software', J. of Softw. Maintenance: Res. and Practice, 1, (1), September 1989, pp. 3-27
- [leh90] LEHMAN, M.M.: 'Uncertainty in Computer Application'. Tech. Let., CACM, 33, (5), May 1990, pp. 584-586
- [leh94] LEHMAN, M.M.: 'Feedback in the Software Process', Keynote Address, CSR Eleventh Annual Wrksh. on Softw. Ev. - Models and Metrics. Dublin, 7-9th Sep. 1994. Also in Info. and Softw. Tech., spec. iss. on Softw. Maint., v. 38, n. 11, 1996, Elsevier, 1996, pp. 681 - 686
- [leh96] LEHMAN, M.M. and STENNING, V.: 'FEAST/1: Case for Support.' Department of Computing. Imperial College, March 1996, available from FEAST web page, see ref. [feast]
- [leh97] LEHMAN, M.M.: 'Laws of Software Evolution Revisited'. Proc. EWSPT'96, Nancy, 9-11 October 1996, LNCS 1149, Springer Verlag, 1997, pp. 108-124
- [leh98a] LEHMAN, M.M.: 'FEAST/2: Case for Support.' Department of Computing, Imperial College, July 1998, from FEAST web page ref. [feast]
- [leh98b] LEHMAN, M.M., PERRY, D.E. and RAMIL, J.F.: 'On Evidence Supporting the FEAST Hypothesis and the Laws of Software Evolution'. Proc. Metrics'98, Bethesda, Maryland, 20-21 November 1998, pp. 84 - 88
- [leh00a] LEHMAN, M.M.: 'Rules and Tools for Software Evolution Planning and Management', in [fesbp]; rev. version with J.F. Ramil in Annals of Softw. Eng., spec. iss. on Softw. Managmt., v. 11, 2001
- [leh00b] LEHMAN, M.M. and RAMIL, J.F.: 'Towards a Theory of Software Evolution and its Practical Impact'. Invited Talk, Proceedings Intl. Symposium on Principles of Softw. Evolution, ISPSE 2000, 1-2 Nov, Kanazawa, Japan, pp. 2 - 11
- [leh00c] LEHMAN, M.M.: 'TheSE- An Approach to a Theory of Software Evolution', project proposal, Dept. of Computing, Imperial College, Dec. 2000, pp. 9
- [leh00d] LEHMAN, M.M. and RAMIL, J.F.: 'Software Evolution in an Era of Component Based Software Engineering', IEE Proceedings - Software, v. 147, n. 6, Dec. (2000), pp. 249 - 255, earlier version as Technical Report 98/8, Imperial College, London, Jun. 1998
- [leh01a] LEHMAN, M.M. and RAMIL, J.F.: 'Software Evolution', inv. keynote lect., IWPSE 2001, Vienna, Sept. 10-11, a revised and extended version of an article to appear in Marciniak J. (ed.), Encyclopedia of Software Engineering, 2nd. Ed., Wiley, 2002
- [leh01b] LEHMAN, M.M. and RAMIL, J.F.: 'EPiCS: Evolution Phenomenology in Component-intensive Software', proposal draft, Dept. of Comp., Imp. Col., Aug. 2001
- [mck99] MCKINNEY, D.: 'Impact of Commercial Off-The-Shelf (COTS) Software on the Interface Between Systems and Software Engineering.' Invited Talk, Panel on COTS Integration, Proc. ICSE'99, 16-22 May 1999, Los Angeles, CA, pp. 627-8
- [mor00] MORISIO, M., *et al*: 'Investigating and Improving a COTS-Based Software Development Process', Proc. ICSE 22, 4-11 June 2000, Limerick, Ireland, pp. 32 - 41
- [myr99] MYRTVEIT, I. and STENSRUD, C.: 'Benchmarking COTS Projects Using Data Envelope Analysis', Metrics 99, Boca Raton, FL., pp. 269 - 278
- [nau69] NAUR, P. and RANDELL, B. (eds.): 'Software Engineering'. report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 October 1968, January 1969, 231 pps.
- [nus97] NUSEIBEH, B.: 'ARIANNE 5 Who Dunnit?', IEEE Software, May/June 1997, pp. 15-16
- [prosim] PROSIM 2000: 'Int. Workshop on Software Process Simulation Modelling', 12 - 14 July 2000, Imp. Col., London
- [ram01] RAMIL, J.F., LEHMAN, M.M., and SANDLER, U.: 'An Approach to Modelling Long-Term Growth Trends in Large Software Systems', submitted to ICSM 2001, 6-10 November, Florence, Italy
- [raj00] RAJLICH, V.T. and BENNETT, K.H.: 'A Staged Model for the Software Life Cycle', Computer, July 2000, pp. 66 - 71
- [sea99] SEAMAN, C.: 'Qualitative Methods in Empirical Studies of Software Engineering', IEEE Trans. on Softw. Eng. vol. 25, n. 4, July/Aug. 1999, pp. 557 - 572
- [sen90] SENGE, P.: 'The Fifth Discipline - The Art & Practice of The Learning Organisation', Currency/Doubleday, NY, 1990, pp. 423
- [soce] SOCE 2000: 'Workshop on Software and Organisation Co-evolution'. 12-13 July 2000. Imp. Col., London
- [tur87] TURSKI, W.M. and MAIBAUM, T.: 'The Specification of Computer Programs', Addison Wesley, London, 1987, p. 278)
- [tur96] TURSKI, W.M.: 'Reference Model for Smooth Growth of Software Systems,' IEEE Trans. on Soft. Eng. 22, (8), August 1996, pp. 599-600
- [voa99] VOAS, J.M.: 'Disposable Information Systems: The Future of Software Maintenance?,' Journal of Softw. Maint: Res. Pract., 1999, 11, pp. 143-150
- [wol85] WOLSTENHOLME, E.F.: 'A Methodology for Qualitative System Dynamics', in The 1985 Int. Conf. of the System Dynamics Society, 1985
- [wu00] WU, Y., PAN, D. and CHEN, M.H.: 'Techniques of Maintaining Evolving Component-based Software', ICSM 2000, 11-14 Oct., San Jose, CA, pp. 236 - 246
- [zur68] ZURCHER, F.W. and RANDELL, B.: 'Iterative Multi-Level Modelling - A Methodology for Computer System Design', Info. Proc. 67, Proc. IFIP Congr. 1968, Edinburgh, Aug. 1968, pp. D138 - 142