

Feedback in the Software Process
Position Paper
SEA Workshop: Research Directions in Software Engineering-
Imperial College
London

April 14 -15th 1997

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

1 Early Work in the Software Process

The software process has been a topic of interest to individual investigators, for over four decades [ben56, leh69, roy70, boe76]. The earliest wider discussions on the topic was probably at the (first) International Process Workshop [spw84]. This subsequently became a regular event with the most recent in November 1996 (Proceedings of the series published by the Computer Society of the IEEE). Papers presented in the early workshops made reference to a wide variety of process models but interest in the process and its modelling was still relatively limited. The turning point came with Osterweil's keynote address to ICSE 9 [ost87], a revision of a paper he had presented at the third Software Process Workshop [ost86]. His ideas were received by many with great enthusiasm but others were more critical and even regarded his approach as potentially harmful [leh87, cur87, not88]. Despite these notes of caution, Osterweil's work triggered a host of activity in process studies focussing primarily on procedural process modelling and the use of process programs to control processes through the use of support environments [kad92]. These studies focussed largely on the technical aspects of software development. They paid little attention to and threw little light on the *behavioural characteristics* of the process, its evolutionary properties, or the influences of the development environment and the organisational framework on the process and its products. With hindsight it would seem that the doubters were correct in their analysis. Much of the resulting effort was sterile and unproductive, yielding little of long term value, little insight into the industrial process, little support for its improvement.

2 Process Improvement

More recently, the efforts of Watts Humphrey [hum89] and the Software Engineering Institute (SEI) have moved the focus away from modelling software processes for their own sake or for providing controls for environment based process guidance. More generally, *ad hoc* improvement of the software process through improvement of its parts is not accepted as the best approach or even sufficient. It is now recognised that improvement is evaluated in terms of the characteristics of the *total process* (that is, from conception through installation to use, adaptation and continuing extension) as observed from outside the process. With regards to process models the interest, particularly in industry, is focussed on the use of model based techniques for *disciplined* process improvement (SPI). These new modelling goals have brought with them new approaches. They have also resulted in a different view of the process. Earlier work had concentrated almost exclusively on the sequence of technical activities used to design and implement software. The SPI approach, culminating in the SEI CMM model [pau93], recognised key areas of activity, technical and non technical, in which changes might be expected to lead to perceptible improvement. The direct and major contribution of organisation and management to effective software development and maintenance, that is to software evolution, was also recognised and these areas off activity are also included in the SPI focus. A world wide process improvement network, SPIN, is now active in industry and relies heavily on the CMM philosophy. But CMM also has its doubters and critics [bol91, gra96].

3 Feedback in the Software Process

Some years ago, the present author pondered the question as to why the global industrial process still leaves much to be desired. This despite many significant innovations at the technical level of software development. Such innovation, includes more rigorous approaches to process design, newly disciplined process steps, new paradigms such as structured programming and object orientation, a host of new languages at ever higher levels and ever more sophisticated support tools.

The question having been asked, an answer immediately suggested itself. The solution might lie in a 70s observation [leh74,78] that *E*-type [leh80b,85] software applications and processes constitute a feedback system. It would then also be strongly related to process attributes encapsulated in the Laws of Software Evolution [leh74,78,80a,80b,85,96c,96d]. The observation and the laws were the outcome of a study of the growth of OS/360 as illustrated in full by the plot reproduced in figure 1. With respect to that plot note, however, that at the time when the first observation quoted below was made the data available extended only to the release corresponding to RSN 19.

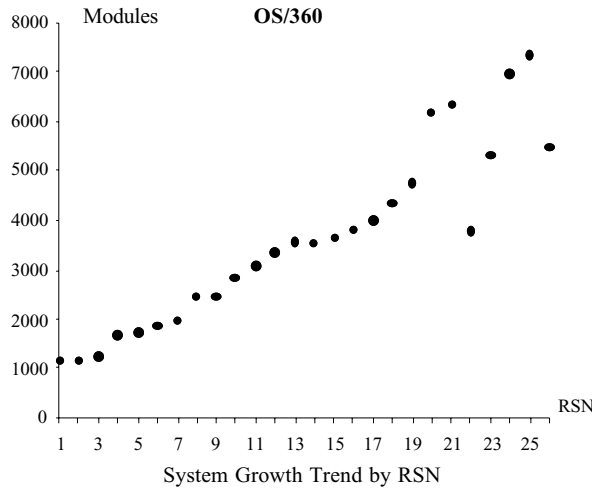


Figure 1. The Growth of OS/360 as a Function of Release Sequence Number

In commenting on an earlier version of this plot in the first publication of an area they then termed *software growth dynamics*, Belady and Lehman wrote: "The ripple ... is typical of a *self stabilising process* with positive and negative *feedback loops*. From a long-range point of view the rate of system growth is self-regulatory, despite the fact that many different causes control the selection of work implemented in each release, with varying budgets, increasing numbers of users reporting faults or desiring new capability, varying management attitudes towards system enhancement, changing release intervals and improving methods..."[bel72].

At a latter stage it was suggested that one should "... regard the organisation developing and maintaining a large program as a *system in the system theoretic sense*. ... Observation has shown that the system behaves as a self stabilising feedback system. ... The process leads to a *process dominated by feedback* ... with long range trends ... and invariances ..." [leh78].

The full implications of these observations do not appear to have been appreciated at the time by either the authors or any one else. Only now has the penny dropped and suggests that the intrinsic feedback nature of *E*-type software development and application may be the reason why major improvement of the total software process is so difficult to achieve. Note, however, that other reasons can also be identified, [leh94] and the question posed above must still be regarded as ripe for research.

Research in programming methodology and software engineering over the past decades has been mainly directed at improving the technical process of software development. The attention concentrated on better process and programming paradigms, better languages, extension of formalisation, more disciplined methods, the development of support tools and so on. Innovation in each of these areas was expected to improve the effectiveness of relevant process steps, to improve the quality, however defined, of the individual development activities to which it was applied.

As will be shown below, such advances may well lead to visible improvement at the global level when applied to relatively primitive processes with little or no disciplined control feedback. That is one may observe consequent benefit outside the total organisation involved in the process that transforms the original application concept or intent into an operational product in the user domain and maintains it satisfactory as usage and the domain evolve. However, as noted above more advanced processes that intentionally discipline and exploit feedback control become a feedback system, albeit one in which all the feedback mechanisms involve humans. The system is, therefore, essentially stochastic rather than determinate. Nevertheless, the

process must be expected to develop the characteristics property of other feedback systems. For such systems, changes to forward path mechanisms, and hence the impact of such changes on process characteristics, will be attenuated to a degree that is a complex function of the feedback loop characteristics. For a system with a single negative feedback control, for example, such attenuation will be approximately equal to the gain in the loop. For multiple loops providing both positive and negative feedback, the consequence will be more subtle. In any event the impact of local forward path changes alone will, in general, be much reduced.

It is, therefore to be expected that the introduction of forward path innovations such as those listed above into the software process, while causing local changes in process behaviour will not have major global impact. The effect on the total organisational process will be constrained *unless the feedback paths and mechanisms can be adjusted to compensate for, and even to exploit, the forward path changes*. Note that the more advanced the process, the more explicit or implicit use is made of feedback controls the smaller the effective global impact of forward path changes. The original introduction of, say, high level languages or structured programming did have a major impact because the processes of that time were relatively naive, undisciplined and not consciously feedback controlled. Similarly, even in our own time, local forward path improvements of, for example, CMM level one processes may have global impact since, by definition, such processes do not exploit disciplined feedback control. As one moves up the levels of process discipline, feedback is increasingly exploited and such changes become ever less effective of themselves. Process optimisation, or at least a move in that direction, can result only from the related changes to forward and feedback paths.

The difficulties encountered in sustaining software process improvement can, therefore, be explained in terms of the feedback phenomenon. It is of interest to ask whether support for this conclusion can be obtained from the success, or otherwise, of technical innovations such as inspection [fag76], rapid prototyping [pro82], incremental and evolutionary development [gil85], the use of software metrics and so on. These all represent feedback mechanisms, where measurement of the product of earlier work, of its impact or of its success is consciously applied in the first place to improvement of the product but, in more sophisticated environments, to improvement of the process. A demonstration that such feedback mechanisms alone, or in conjunction with forward path improvements, have consistently had more impact on the global process than have forward path changes alone, would constitute strong evidence in support of what is now termed the FEAST hypothesis.

4 The FEAST Hypothesis and FEAST/1

The FEAST (Feedback, Evolution And Software Technology) hypothesis was first expressed [fea94, leh94] as: *As complex feedback systems, E-type [leh80b,85] software processes evolve strong system dynamics and with it the global stability characteristics of other feedback systems. Consequent stabilisation effects are likely to constrain efforts at process improvement*. More recently [leh96b] it was restated as: *As for other complex feedback systems, the dynamics of real world software development and evolution processes will possess a degree of autonomy and exhibit a degree of global stability*.

An international FEAST workshop [fea94] was organised to explore the hypothesis and its implications. The interest aroused led to two further workshops [fea94,5] and, subsequently to a successful research proposal FEAST/1 [leh96b]. The stated objectives of that project, underway since October 1996, are to:

- (a) provide objective evidence that feedback phenomena and the consequent system dynamics have substantial impact in the software process
- (b) demonstrate that the phenomena can be exploited in both managing and improving industrial processes
- (c) produce justification for a more substantial study based on the feedback perspective

More specific objectives are to:

- (i) provide models of the long term (multi-release) behaviour of industrial software processes and their products that are more extensive and precise than any hitherto available
- (ii) provide calibrated and validated SD models of development processes for individual releases
- (iii) use these models to expose feedback phenomena within such processes
- (iv) demonstrate the predictive capability of the models
- (v) identify process improvements suggested by the models and assess their impact in practice
- (vi) demonstrate the need for further studies and define their nature

5 Process Research

In the context of the SEA workshop on Research Directions in Software Engineering it is noted that these objectives represent, at best, foundational investigations. They will, it is hoped, confirm the relevance of the

FEAST hypothesis to full mastery of the software process and the contribution on process improvement of process models that include feedback mechanisms. As stated the hypothesis implies three assertions:

- I Software evolution processes for *E*-type systems constitute complex feedback systems
- II Such feedback is likely to constrain benefit derived from forward path changes, however promising in themselves
- III **Major** process improvement requires attention to feedback mechanisms to address constraints arising from the process dynamics

It also appears plausible to infer from these assertions, that:

- Slow progress in process improvement may be due, at least in part, to lack of attention to feedback phenomena

Research to explore the hypothesis must begin by examining evidence that supports or contradicts these assertions. If, as we expect, they are upheld, foundations for their wider exploitation must then be established. These are likely to employ the whole gamut of software engineering theory and practice. Even if only confirmed for certain classes of development domains, a whole new dimension is added to the study of the software process and to process improvement R & D. Investigations starting with the feedback premise will benefit from work in the software process area to date, but set new research directions and objectives, offering the potential for important new benefits.

Study of the area is not straight forward. In the first place much of the feedback control activity is not planned or even conscious. Moreover, feedback related phenomena and the techniques required for their analysis and control are not normally part of the background of those active in software engineering research. Even for professionals in the field of control engineering the problems posed by feedback mechanisms that include human observation, interpretation, decision and action (or inaction) as an integral part of their *modus operandus* raises quite new problems. Some of these may, however, have already be tackled in fields such as economic modelling [bar92, bec94]. There is also existing work in system dynamics applied in the software engineering domain [abd89,91, mad96].

6 An Overview of Initial FEAST/1 Results and their Research Implications

Initial results from FEAST/1 [tur96, leh96] are most encouraging in their support of earlier conclusions and the FEAST hypothesis in particular. As an example consider figure 2 which reproduces the growth curve of the Logica FW system. The limited data provided by Logica so far has permitted only preliminary analyses. The size of FW is shown here plotted against its release sequence number (RSN). It will be apparent how similar this response is to the OS/360 growth curve of figure 1, displaying in particular, the ripple phenomenon.

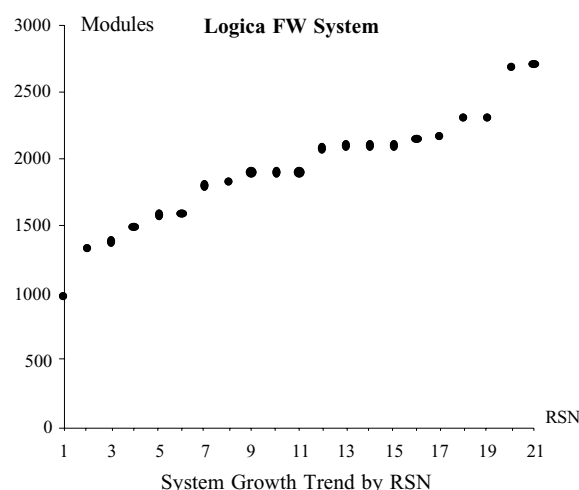


Figure 2. The Growth of Logica FW system as a Function of Release Sequence Number

The 1970s study of the OS/360 evolution data [leh69,72,80,85] led to observations that were encapsulated in a series of laws of program evolution. These were later confirmed, or rather not contradicted by, more limited studies of other systems [leh80b]. The first three laws were published in the mid seventies [leh74], a further

two some four years later [leh78] and three more in recent years. While the analysis is in no sense complete and further evidence from more systems is required if wider confidence in the laws is to be achieved, the initial examination data leads to conclusions that may be interpreted as supporting the laws [leh96c] or, to be more scientific, not contradicting them. One example is provided by the incremental growth of FW per RSN [leh96c]]. This too is strikingly similar to that of OS/360. It was this growth pattern in the latter system that led to the formulation of the fifth law of software evolution and the same general behaviour is now observed in this quite different system.

An even more striking result is reported by Turski [tur96]. He showed that the FW growth trend is best fitted by an inverse square model with a single parameter "E". In the first instance he estimated E using all 21 releases for which data was then available. He then successively recalculated E using all subsets of data pairs always starting with the first two releases RSN 1 and RSN 2. Figure 3 plots the *absolute error* and *standard deviation* of the size predicted by the model relative to the actual size, as a function of the number of data pairs used for estimating E. The plot provides strong evidence of that an internal controlling dynamics develops over the early releases. The figure suggests that data from the first six releases suffice to determine the value of E to yield a model that provides a growth trend predictor accurate to better than 5%. Thereafter, the internal dynamics dominates further growth. This is a remarkable result [tur96] that greatly increases confidence in the validity of the FEAST hypothesis.

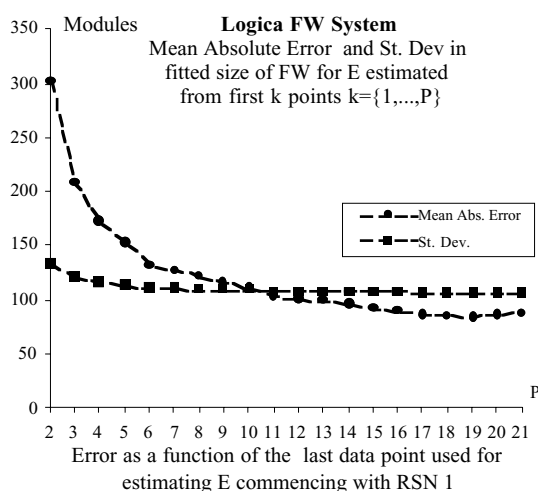


Figure 3. 1990s System - Logica FW

7 Business Process Improvement

One further brief observation must be made before the implications of the above review on the workshop topic, Research Directions for Software Engineering are briefly considered. In general, *E*-type software systems are not developed for their own sake. They are required to address a need in some domain as effectively as possible. Past experience has shown the significance of this observation. For example, in seeking to use computers in the late fifties and early sixties US banks and insurance companies, the first business organisations to attempt major, if not total, automation, soon recognised that it was inappropriate to install computers as electronic clerk-replacements. For the introduction of computer based systems and processes into a business operation one needed to address the question "How shall we *conduct our business* now that computers are available?".

Only recently has this awareness spread more widely; have enlightened organisations begun an organised, disciplined and integrated search for overall business process improvement. Such activity also includes computer supported processes. There are, in fact, few business processes that do not involve, or depend on, computers in some fashion. In addressing the process improvement challenge it was soon discovered that the existence of legacy computer systems severely constrained the freedom to change and improve other business processes unless the role of the computer system and the manner of its integration into the operation is well understood. Moreover, the software which plays a major role in determining the characteristics of the entire process must be reliably, responsively and economically adaptable. To achieve this requires, in general, the improvement of both the business and software suppliers software processes. Clearly, modelling and improving these to achieve such adaptation must be done in the context of both businesses. More specifically, improvement of software industry processes, processes that include *ab initio* development, fault fixing, adaptation and extension, must consider also the nature of their interactions with clients. Process

improvement cannot be achieved as a self contained exercise.

8 Implications for Software Engineering Research Directions

The preceding discussion has identified and outlined three *facts of life*.

The first indicates that the principal industrial and commercial interest in the software process and its modelling is in its relationship to process improvement. In this context it has been recognised that the concern must be with the organisation, the activities, the management and the control of *all* concerned *in any way* with the software evolution process.

The second is based on recognition of the fact that both the purely technical process and the total process followed to develop and evolve *E*-type systems is, at least in part, triggered, guided and controlled by feedback loops, formal and informal. If forward path changes, improvements in any aspect of the technology for example, are to achieve externally observable and significant impact they must be accompanied by appropriate modification of, at least, the control feedback mechanisms. Adjustment of these will be facilitated by quantitative models of the system in general and the controlled subsystems in particular.

Finally it is now recognised that there is little point in improving the technical software development process in itself. What must be improved is the business processes with and within which the computer system is integrated.

All this impacts the software engineering fraternity in two ways. In the wider community, computers are playing an ever more crucial role. These computers depend totally on their software for functionality and for the contribution they make towards organisational goals. It is fundamental to *E*-type systems that they require to be continually adapted to support and remain compatible with changing goals, a changing application in its changing operational environment [leh80b,85,91]. Modelling the technical software development process in ever greater detail yields minimal payback to this wider community whether in economic or societal terms. The future focus in business process modelling *must* be on the total process in its operational environment. This also holds true for the software industry. Consideration of the software process must involve a far wider range of activities and responsibilities than those considered in most past software process modelling. In addition, the evidence suggests that process studies, metrics and development work must include the feedback mechanisms at many levels of the process. These must be identified, modelled and mastered. And then we must learn to exploit them.

Just deciding which organisations and activities are relevant is, itself, often problematical. The most visible are those directly involved in the technicalities of software development and evolution. But corporate executives, management, the sales and user support organisations and, of course, the user community also strongly influence many aspects of the process. Indeed that influence may turn out to be dominant.

The conclusion is that software engineering R & D must turn from the more traditional forms of process modelling and assessment. It must address and seek to understand and improve the more complex software evolution domains being posited [leh97]. And this implies that these domains be modelled at many levels of detail using appropriate techniques. In view of the major widening of the process domain and the transition from open to feedback system representation this is a significant change in direction. It involves new approaches, even new disciplines. It is likely to lead to a long term, multi-disciplinary investigation which could benefit from the involvement of mathematicians, statisticians, control engineers, organisation theorists, management and behavioural scientists and so on. But we software engineers bear the primary responsibility. Only we can recognise the true nature of the problem. Only we are aware of the technologies that are available or that can be developed. Only we can comprehend the totality of processes and activities that must be employed.

9 Acknowledgements

Responsibility for claims made in this paper rests entirely with the present author. The major roles played by Dr Dewayne Perry and Professors Vic Stenning and Wlad Turski (the FEAST core group) in developing and exploring some of the concepts and results presented here deserve full and grateful acknowledgement. More recently the two FEAST/1 Research Associates, Dr P Wernick and Juan F Ramil have joined the team of investigators. Finally acknowledgement is due to the EPSRC for grants numbers GR/K86008 and GR/LO7437 that are supporting most aspects of this investigation and are permitting the continuing involvement of Professor Turski and Dr Perry.

10 References - items marked with an "*" are included in [leh85]

- [abd89] Abdel-Hamid T and Madnick S E, *Lessons Learned from Modelling the Dynamics of Software Development*, CACM, v. 32, n. 12, Dec. 1989, pp. 1426 - 1438
- [abd91] Abdel-Hamid T and Madnick S E, *Software Project Dynamics: An Integrated Approach*, Prentice Hall, Englewood Cliffs, 1990, NJ 07632, 264 p.
- [abd89] Abdel-Hamid T and Madnick S E, *Lessons Learned from Modelling the Dynamics of Software Development*, CACM, v. 32, n. 12, Dec. 1989, pp. 1426 - 1438
- [bar92] Barghouti N S and Kaiser G E, *Scaling Up Rule Based Development Environments*, Int. J. on Softw. Eng. and Knowledge Eng., v.2, n. 1, March 1992, pp. 59 - 78
- [bec94] Becker R S, Hall B, and Rustem E, *Robust Optimal Control of Stochastic Nonlinear Economic Systems*, J. of Economic Dynamics and Control, n. 18, 1994, pp. 125 - 148
- [bel72] Belady L A and Lehman M M., *An Introduction to Program Growth Dynamics*, in Statistical Computer Performance Evaluation, W Freiburger (ed), Academic Press, New York, 1972, pp. 503 - 511
- [ben56] Bennington H D, *Production of Large Computer Programs*, Proc. Symp. on Advanced Computer Programs for Digital Computers, sponsored by ONR, June 1956, Republ. in Annals of the History of Computing, Oct. 1983, 350 - 361
- [boe76] Boehm B W, *Software Engineering*, IEEE Trans. on Comp., v. C-5, n. 12, Dec. 1976, pp. 1226-1241
- [bol91] Bollinger T B and McGowan C, *A Critical Look at Software Capability Evaluations*, IEEE Software, July 1991
- [cur87] Curtis B, Krasner H, Shen V and Iscoe N, *On Building Software Process Models Under the Lamppost*, Proc. 9th Int. Conf. on Softw. Eng., Monterey, CA, 30 March 2 Apr. 1987, IEEE Comp. Soc. pub. n. 767, IEEE Cat. n. 87CH2432-3, pp. 96 - 103
- [fag76] Fagan M I, *Design and Code Inspections to Reduce Errors in Program Development*, IBM Sys. J., v.15, n. 3, 1976, pp. 182 - 211
- [fea94] *Preprints of the (first) FEAST Workshop*, M M Lehman (ed.), Dept. of Comp., ICSTM, June 1994
- [fea94,5] *Preprints of the three FEAST Workshops*, MM Lehman (ed.), Dept. of Comp., ICSTM, 1994/5
- [gil85] Gilb T, *Evolutionary Development vs. the Waterfall Model*, SE Notes, v. 10, n. 3, July 1985, pp. 49 -61
- [gra96] Gray E M and Smith W L, *On the Limitations of Software Process Assessment and the Recognition of a Required Reorientation for Process Improvement*, ESI, Bilbao, Spain; Proc., Sep., 1996
- [hum89] Humphrey W S, *Managing the Software Process*, Addison-Wesley, Reading, MA, 1989
- [kad92] Kadia R, *Issues Encountered in Building a Flexible Software Development Environment: Lessons Learnt from the Arcadia Project*, Softw. Eng. Nores v. 17, n. 5, Dec. 1992, pp. 169 - 180
- [leh69]* Lehman M M, *The Programming Process*, IBM Res. Rep. RC 2722, IBM Res. Centre, Yorktown Heights, NY 10594, Sept. 1969
- [leh74]* id., *Programs, Cities, Students, Limits to Growth?*, Inaugural Lecture, May 1974. Publ. in Imp. Col of Sc. Tech. Inaug.1 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
- [leh85] Lehman M M and Belady L A, *Program Evolution, - Processes of Software Change*, Academic Press, London, 1985, pp. 538
- [leh87] Lehman M M, *Process Models, Process Programs, Programming Support - Invited Response To A Keynote Address By Lee Osterweil*, Proc. 9th Int. Conf. on Softw. Eng., Monterey, CA, 30 March 2 Apr. 1987, IEEE Comp. Soc. pub. n. 767, IEEE Cat. n. 87CH2432-3, pp. 14 - 16

- [leh91] id., *Software Engineering, the Software Process and Their Support*, IEE Softw. Eng. J. Spec. Iss. on Software Environments and Factories, Sept. 1991, v. 6, n. 5, pp. 243 - 258
- [leh94] *Feedback in the Software Evolution Process*, Keynote Address, CSR Eleventh Annual Workshop on Software Evolution: Models and Metrics. Dublin, 7 - 9th Sept. 1994, in Information and Software Technology, sp. iss. on Software Maintenance, v. 38, n. 11, 1996, Elsevier, 1996, pp. 681 - 686
- [leh96] Lehman M M, Perry D E and Turski W M, *Why is it so hard to find Feedback Control in Software Processes?*, Invited Talk, Proc. of the 19th Australasian Comp. Sc. Conf., Melbourne, Australia, 31 Jan - Feb 2 1996. pp. 107-115.
- [leh96b] Lehman M M and Stenning V, *FEAST/1: Case for Support*, ICSTM, March 1996
- [leh96c] Lehman M M, *Laws of Software Evolution Revisited*, Proc. EWSPT'96, Nancy, 9 - 11 Oct. 1996
- [leh96d] id., *Process Improvement - The Way Forward*, Invited Keynote Address, Proc. Brazilian Software Engineering Conference, 14 - 18 October 1996, pp. 23 - 35
- [leh97] id., *Process Modelling - What Next?*, Proc. ICSE 97, Boston, 20 - 22 May 1997, to be published
- [mad96] Madachy R J, *System Dynamics Modelling of an Inspection Process*, 18th Int. Conf. on Softw. Eng., Berlin, 25 - 29 March. 1996, IEEE Comp. Soc. ord. n. PR07246, IEEE Cat. n. 96CB35918, pp. 376 - 386
- [not88] Notkin D, *Applying Software Process Models to the Full Life Cycle is Premature*, in Representing and Enacting the Software Process, Proc. the 4th Int. Process Workshop, 11 - 13 May 1988, Tully C (ed), Moretonhampstead, UK, ACM Softw. Eng. Notes, June 1989, ACM Press, IEEE cat. n. 88TH0211-3, pp. 116 - 117
- [ost86] Osterweil L, *Software Processes are Software Too, Iteration in the Software Process*, Proc. of the 3rd Int. Proc. Workshop, Breckenridge, CO, 17 - 19 Nov. 1986, IEEE cat. n. TH0184-2, IEEE Comp. Soc. order n. 709, 1987, pp. 79 - 80
- [ost87] Osterweil L, *Software Processes are Software Too*, Proc. 9th Int. Conf. on Softw. Eng., Monterey, CA, 30 March - 2 Apr. 1987, IEEE Comp. Soc. Pub. n. 767, IEEE Cat. n. 87CH2432-3, pp. 2 - 13
- [pau93] Paulk M C, Curtis B and Chrsi M B, *Capability Maturity Model for Software, Version 1.1*, Software Engineering Technical Report, CMU/SEI-93-TR, Feb. 24 1993
- [pro82] *Working Papers from the ACM SIGSOFT Rapid Prototyping Workshop*, Colombia, MD, 19 - 21 April, 1982, SEN, Dec. 1982
- [roy70] Royce W W, *Managing the Development of Large Software Systems*, IEEE Wescon, August 1970, pp. 1 - 9
- [spw84] Potts C (ed), *Proceeding of the Software Process Workshop*, Egham, Surrey, UK, Feb. 1984. IEEE, cat. n. 84CH2044-6, Comp. Soc., Washington D.C., ord. no. 587, pp. 27 -35
- [tur96] W M Turski, *Reference Model for Smooth Growth of Software Systems*, IEEE Trans. on Soft. Eng. v.22, n. 8, August 1996

mml567[papers]
7/3/97