Process Improvement - The Way Forward

M M Lehman
Department of Computing
Imperial College of Science, Technology and Medicine
London SW7 2BZ

tel.:+44 (0)171 594 8214
fax: +44 (0)171 594 8215
email: mml@doc.ic.ac.uk

1 Introduction

Means for evolving, that is, developing, adapting and enhancing E-type[1] [LEH80] software have been significantly advanced over the years. Continuing efforts to improve the process of software evolution have produced numerous concepts, methods, techniques and tools. High level languages, structured programming, abstract data types, formal methods, non-procedural programming, CASE environments and object orientation exemplify innovations expected to overcome problems that have for so long [NAU69] frustrated consistent, cost effective, on-time development of functionally satisfactory and reliable software. Such innovation did indeed yield process-local benefit. Introduction of structured programming and high level languages, for example, greatly improved program design and coding. The study of programming methodology [GRI78] led to major advances in computer science and to the development of formal methods. These, in turn, provided opportunities for increasing individual and small group effectiveness by facilitating CASE based process mechanisation. Continuing technical innovation has played a significant role in major growth in size and functional complexity of computing applications and systems.

It has, however, not yielded a panacea, neither a silver bullet [BRO86] nor a philosophers' stone [TUR86]. Many problems still haunt industrial software development [GIB94]. Introduction of improved methods, techniques and tools into practice has not yielded a consistent capability for planned, on time, controlled-cost development of quality software. Nor has it resulted in major productivity growth, cost reduction or faster response to user needs. It has proven equally difficult to achieve major improvement in maintaining systems satisfactory as user needs and expectations change in evolving application and operational domains. Software evolution from concept to first installation and from release to release relies on processes still far from satisfactory [GIB94].

Despite the belief of major industrial organisations to the contrary [MAJ93] it may, of course, be unreasonable to expect order of magnitude improvement from individual innovations or in the overall process. Anticipation of benefit is, after all, not a proof that it is attainable. Software development is a creative, intellectual activity requiring human involvement, learning, judgment, decision and revision. It can be supported but not replaced by mechanisation. This by itself limits increases in global process quality, productivity and responsiveness. Moreover, since the operational

[1] *E*-type software is informally defined as software that implements an application or addresses a problem in the real world [LEH78]

environment undergoes continual change software products cannot, for long, remain fault free [LEH91]. The consequent need for continual change and evolution presents a further major obstacle to overall process effectiveness. There may well be others. Software development is intrinsically and will always remain a challenging and hazardous enterprise.

Nevertheless, the consistent failure of innovative ideas to yield major improvement in *E*-type processes despite their apparent effectiveness in *S*-type[2] [LEH80,95] programming suggests that there may be some common *cause* or *causes* constraining the former. If that is so, its identification should provide clues as to how more effective software processes might be achieved.

One possible cause is immediately apparent. Technical development is only one of many software process activities. Project management, user support, application analysis, marketing and enterprise management, for example, all contribute to system evolution, absorbing resources and impacting progress. All influence software process and product attributes. Individual innovations impact directly only a fraction of the total activity. Global impact of any single innovation must, therefore, be limited.

The slow rate of progress may also relate to the number or diversity of people and organisations involved in *E*-type application development, the complexity of the organisations and the processes executed and controlled or specific characteristics of *E*-type systems [LEH94B]. Each of these factors could explain the continuing difficulty in achieving major improvement.

2 Feedback and the Software Process

Recently a more basic constraint on process improvement has been suggested [LEH94]. It relates to the role of feedback in *S*-type and *E*-type processes respectively. By definition, the specification of an *S*-type program completely defines what is to be implemented. Its operational domain is bounded by a specification which is sacrosanct. Conformance to the specification completely determines, in a mathematical sense, *correctness* of the program and its parts. If, at any time and for whatever reason, the specification is considered unsatisfactory, if for example it does not fully address client needs, a *new* one must be generated and a *new* program to satisfy it developed. Each may be derivable from its predecessor. But technically both are new since specification *changes* are ruled out by definition. Validation of the specification and, when necessary, its revision is entirely separated from the process transforming the specification into its program implementation. Feedback, iteration, backtracking over process steps may be used to achieve convergence to a solution, to rectify errors or to escape from *blind alleys* but not to increase fitness for purpose by change of the specification. The global *S*-type process is, by definition, open-loop. In so far as changing perceptions, opportunities and needs require adaptation, enhancement or extension of an *S*-type system it evolves as a succession of new systems rather than by changes to its parts.

In strong contrast, the applications and operational domains realised by *E*-type programs are unbounded [LEH94B]. Knowledge about them cannot be absolute or complete. As a model of the application in its domain the bounded *E*-type system is an

[2] *S*-type software is required to be *correct* in the full mathematical sense with respect to a fixed specification [LEH78]

abstraction of *reality*. Behavioural judgments and pragmatic inputs about implementation resources and technology play a major role in setting the properties of the model. The gap between the reality of the application in its operational domain and the system model is bridged by assumptions [LEH91]. This gap must be maintained sufficiently narrow to ensure that, in usage, program behaves as required, that the system as a model reflects reality to the extent needed. The concept of *correctness* determining the acceptability of *S*-type programs is replaced by user *satisfaction* with the domain covered, system behaviour, program functionality and program execution [LEH91]. But experience, insight and understanding acquired during system evolution and usage generate new perceptions, needs and opportunities, changing expectations and criteria of satisfaction; and the external world also changes independently. Constant observation and a stream of information drive, guide and control system evolution to maintain user satisfaction as the application, its operational domain and user perception of both change. Bounds are continually redrawn during development and usage as feedback provides information and impetus for controlled change. The very nature [TUR81,LEH91,95] of real world applications and of the *E*-type software that models and implements them sets up continuing pressure for change and evolution based on observation, experience, learning, judgment and decision. Some of the information communicated serves to enlighten recipients. Other is used to control future execution of the activity that generated the information. It constitutes genuine feedback control [OXF81]. In contrast to *S*-type development the *E*-type process is inherently closed loop with iteration and backtracking guided and controlled by feedback from users, developers, managers and many others [LEH69,85]. The regular system dynamics that results determines many of the characteristics of the evolution process [BEL72, LEH85].

The significance of feedback in the software process and its role in determining the dynamics of that process has long been recognised. It was referred to in passing by several speakers at the Garmisch Conference [NAU69]. At about the same time it was briefly discussed in the 1969 *Programming Process* [LEH69] report. The first tangible evidence of an identifiable dynamics of evolution followed some years later [BEL72]. More detailed studies were reported in subsequent papers [BEL72, LEH85]. As an example brief mention may be made of the early identification of the feedback stabilised and controlled growth characteristics of OS/360 and other systems [LEH80] as illustrated in the growth plot reproduced in the figure below.

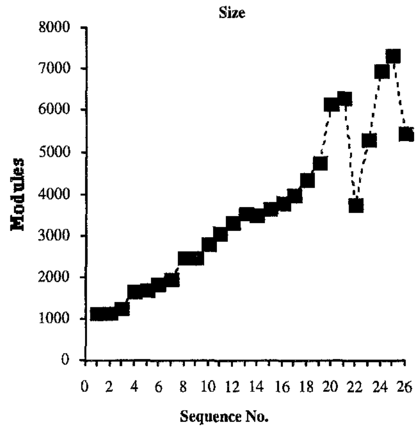

Fig. 1 The growth of OS/360

The cyclic pattern discernible in this plot is characteristic of feedback systems. As observed at the time [LEH72] "... 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 budgets varying, increasing numbers of users reporting faults or desiring new capability, varying management attitudes towards system enhancement, changing release intervals and improving methods....". The period of instability beyond release twenty representing OS/360 fission some months after the '72 paper was published is equally indicative of the feedback nature of the software release process. The oscillatory behaviour indicates a loss of control over system evolution. It is consistent with all known facts that this chaotic behaviour was triggered by over ambitious growth targets, that is, excessive positive feedback.

Further analysis of these and observations on a number of other systems [LEH80] led to identification of five laws of program evolution [LEH74,78,85]. These reflect human and organisational attributes and behaviour rather then software technology. From within the technology they must, therefore, be accepted as laws. More recent studies of organisational and managerial aspects of software process dynamics have developed techniques for the exploitation of that dynamics [ABD91]. Taken together the results of this work provide the basis for an emerging theory of software evolution [LEH85,91, ABD91].

3 A Consequence of Feedback Control

The 1969 - 72 studies from which the feedback nature of the software process was first inferred were restricted to release level evolution. But information generation and feedback play a major role at all levels of the process. Processes of *E*-type evolution constitute multi-level, multi-loop feedback systems. Loop characteristics and those of their mechanisms determine process dynamics. Such processes may therefore be expected to display the stable behaviour which is the hallmark of feedback systems in general [LEH94]. Despite changes in the characteristics of forward path elements and in the operational environment externally observable system properties are held relatively constant by negative feedback within specified limits over the operational range until, as a consequence of excessive positive feedback, instability sets in.

The above observations may have been interpreted in the context of the transformation processes applied to refine computer application concepts into solution systems. After all, the 1970s investigation concentrated on technical development. Their relevance is, however, much wider. Management, customer support, quality assurance, process engineering and so on all apply feedback controls derived from monitoring and reporting mechanisms, checks and balances. More feedback and control comes from the organisational (business) environment. Technical developers and their management seek to meet project goals. The software process is changed as participants and software process engineers observe the effectiveness and appropriateness of the current process, as technology advances. Organisational processes use feedback procedures to ensure steady business and organisational growth with disciplined product evolution as, for example, user experience and changing client needs are reported and economic circumstances change.

There can be no doubt that feedback based control plays a significant role in software development processes and in the improvement of such processes. In accordance with the stability property of feedback controlled systems, changes to forward path elements of such processes cannot, therefore, be expected to produce major global improvement unless accompanied by commensurate changes to related feedback mechanisms [LEH94]. Software process improvement must be pursued in the context of the total process domain and the feedback controls that regulate its behaviour. That domain includes, amongst others, users of all types, corporate management, marketing, customer and user support, project, process and information management, technical development, quality assurance, process engineering, interaction with related processes, process improvement and monitoring of all these. Such a broad focus provides a realistic framework for the study of process effectiveness, process dynamics and changes in both.

The innovations listed in the opening paragraph were all forward path mechanisms. Yet their introduction into practice did not, in general, include a comprehensive review of total process-domain and its feedback controls. Thus though their adoption may have changed local process properties it should not come as a surprise that the wider impact was far less than expected. It is suggested that a common factor constraining major software process improvement has been a lack of attention to the impact of feedback on forward path innovation. Support for this conclusion is provided by the positive contributions arising from the introduction of innovative techniques such as *inspection*, *reviews*, *prototyping*, *incremental* and *evolutionary development* and the emerging *metrics technology*. These techniques all include a strong negative feedback control component. Their potential for major impact on global process effectiveness provides further support for the feedback hypothesis.

4 The FEAST Conjecture

The above above observations have been formalised in the following conjecture:

As a multi loop feedback system the E-type software process will display global invariance characteristics

This conjecture includes three separate and distinct assertions

I The *software evolution process* for *E*-type systems constitutes a complex feedback system

II Process feedback is likely to limit the benefit derived from individual forward path changes

III Major process improvement requires that changes to individual steps are accompanied by adjustment of feedback paths and/or mechanisms

The first assertion is undeniable. The others follow from the global stability characteristic property of other feedback systems. If, as seems likely, the software process as a feedback system also possesses this property, improvement resulting from changes to one of its forward path mechanisms will be constrained by pre-existing negative feedback. To remove such constaints requires examination, probable modification, possible removal of at least some of the feedback controls that are almost certain to be in place. But can examination and adjustment of process feedback can be systematised? Can software process feedback design be disciplined? The general theory and practice of analysis, control and design of feedback systems is advanced and well understood. A question arises in the case of the software process because of the major, independent and creative role of the many individuals involved in the process, the varied roles they fullfill, the unpredictable nature of their influence on feedback information. Humans observe and participate in the process and in the operation of its product. They manipulate and control information fed over paths that link organisations, activities, *spaces* [BEN93] and people involved in system and process evolution. They *observe*, *interpret*, *verbalise*, *transform*, *communicate*, *assess*, *decide*, *control* and *apply* both forward and feedback information. They feed back their interpretation to other units involved in the evolution. Each of these acts imposes a personal stamp on the information. One must, therefore, ask whether human involvement is so extensive, so ingrained, so individual, so judgmental, so creative that meaningful and exploitable formalisation and modelling with optimised design and integration of the feedback mechanisms is, at least for the moment, beyond reach, cannot be disciplined? The issue is not the validity of the assertions but their practical implications.

5 Exploiting Process Feedback

To exploit feedback one must be able to model the process and its dynamics. Techniques currently employed in process modelling do not, in general, provide the necessary facilities since they have not sought to reflect detailed feedback properties. But relevant formalisms and methods have been developed in other areas. Comprehensive techniques for feedback design and control of continuous and of stochastic systems is embodied, for example, in control theeory and in dynamic systems theory. Both have been extended and applied, though admittedly with limited success, to systems involving humans; economic systems, organisational dynamics and the application of control theory to software development [WOO79, LEH85, ABD91], for example. There exists, therefore, a *prima facie* case suggesting that models reflecting feedback mechanisms may be successfully developed and applied to the design and improvement of software processes at least at levels of detail where statistical abstraction of people activity has meaning. Such models are an essential tool if the above conjecture is to be exploitable for the process of process improvement. Modelling techniques that will permit the representation and evaluation of *all* aspects of the process are an urgent necessity.

In the many spaces in which the process operates and at the many levels of detail at which it occurs feedback may take one of two forms. *Control feedback* describes the situation in which information derived from information originating at an output of some process elemen is injected, after some delay, to an input of that or an earlier element to effect some form of control. It is this meaning of feedback that is studied in, for example, control theory where it refers to a control signal derived directly from a mechanism, from the change in value of some output variable relative to a previously observed value, from the rate of change of a variable and so on. But the term *feedback* is also used colloquially to refer to information flow without any indication as to how, where or when, if at all, that information is to be used. It provides *enlightenment*, advances human *understanding*, facilitates *learning*. Since no control information is derived, such feedback can have no analysable impact on the processes from which it stems or which it reaches.

In the context of the software process both usages are relevant. At levels, where interest and concern focus on the work of individuals, isolated, and in some sense spontaneous and unpredictable, items of information are fed for use as seen fit by the recipient. That recipient may choose not to act directly in response to the information though it may, nevertheless, effect further action as a consequence of its impact on *understanding*, *viewpoint*, *attitude* and so on. But any changes in the latter are all internal to the individual or individuals concerned. As such, they cannot be reflected in process models or formal descriptions of the system dynamic, at most, as noise or randomised variations on unit inputs. Decision to take action (or to take no action), on the other hand, leads to a control action. This situation is an instance of the first, the normal, engineering usage of feedback. Whether these low level aspects of feedback loops and mechanisms in the software process and their impact can be modelled and exploited, requires further investigation .

At higher levels of the process there will be many continuing streams of (discrete) information. The same distinction must, nevertheless be made. Where information is simply absorbed its receipt will not directly impact the process. Where the information is assessed for possible action a control signal is derived, though if the decision is not to act (for the moment) that control signal may be a null signal. Here too the term *feedback* is being used in its normal engineering connotation. Nevertheless, because of the non determinacy of human involvement its modelling and management poses difficult technical and managerial problems.

The belief that systematic techniques for the observation, measurement, modelling and management of feedback can be developed stems from the fact that the total information flow in the process generally involves many decisions. The information fed back and, more significantly, the resultant action is a composite of many inputs. These, whilst not absolutely predictable or independent, are amenable to meaningful statistical representation and analysis [LEH80]. Consequent system behaviour has been shown to display statistically *normal* properties [CHO81]. It is therefore reasonable to expect that at these levels of the software process, control theoretic and statistical process models reflecting the system dynamics can be developed for use on their own or in conjunction with modelling techniques currently in vogue. When statistical representation is not meaningful new formalisms will have to supplement current process modelling techniques. At this level simulation techniques would likely constitute an important element of the design and evaluation process.

6 Feedback as the Constraint on Process Improvement

Reasoning as outlined above has suggested that the *common cause* referred to in section 1 is related to the feedback nature of the evolution process. Whether it *explains* the failure of process innovations such as those identified in the opening paragraph to produce impact of the order of magnitude anticipated at the global level remains to be determined. In truth many, if not all, of the innovations yielded significant benefit at the local level, improving the effectiveness of individual process steps or activities significantly. As an example consider the conception and introduction of high level languages. This certainly increased the quality, productivity and predictability of program code development and its changeability by an order of magnitude. What is now suggested is that the constraining effect of feedback has prevented the full potential of such languages from being experienced at the global level. And so for other innovations.

The feedback hypothesis provides an explanation that is consistent with an established property of other feedback systems. But that observation by itself does not *prove* that the lack of major advances in process improvement is due to this common cause. It could still be primarily due to reasons specific to each innovation. In view of the number of such failures a common cause must, however, be suspected. It is, therefore, of interest to examine innovations individually in the context of processes within which they have been employed to determine whether their limited impact at the global level can be attributed to the constraining effect of feedback control. If it can, it can be overcome by modifying the feedback structure. Failure would not prove the conjecture invalid. It *would* cast doubt on its practical relevance.

In summary, from the facts that feedback systems, in general, display global stability and resistance to change to a degree dependent on the detailed characteristics of their feedback mechanisms and that the software evolution process constitutes a feedback system, one must suspect that the benefits obtained from innovative changes to forward path methods techniques or tools in the software evolution process will be limited. The extent and degree of the constraining effect will depend on the characteristics of the many individual feedback paths and on the interactions between them. It may equally be anticipated that the global benefit obtained from improvements in forward path technology can, in general, be increased by attention to (adjustment of) the characteristics of relevant feedback mechanisms. It is thus tempting to suggest that the feedback phenomenon explains why, despite the many innovative concepts that have been introduced into forward path technology, it has proven so difficult to achieve major improvement in the global software process. Whether this is indeed so remains to be explored as does the question whether, if true, it can be systematically exploited.

An aside is appropriate at this point. In papers at IFIP Congress '86 Brooks [BRO86] and Turski [TUR86], respectively, pointed out that one must expect neither a silver bullet nor a philosophers' stone to solve the software engineering problem once and for all. The FEAST conjecture is not an exception. If it can be exploited, it may make a significant contribution to improvement of the software evolution process. It must be seen just as that, no more.

7 The FEAST Project- (Feedback, Evolution And Software Technology)

Feedback control and its role in software evolution, the software process and process evolution (improvement) are now being investigated with international collaboration, in a project, FEAST, supported for its first year by a grant from the UK Department of Trade and Industry. If successful, the project may be expected to have a profound impact on the software development and maintenance processes and on the process of process improvement The investigation will seek to verify the feedback conjecture and search for ways in which the feedback phenomenon may be exploited.

As already observed, the basic fact that the software evolution process constitutes a feedback system is self evident. But has feedback really constrained the benefit derived from the introduction of innovative concepts, methods, tools and techniques in forward path mechanisms? How may feedback control be exploited? Ideally one should be able to identify feedback paths that inhibited or damped the benefit obtained from individual innovations in current industrial processes and to explore beneficial changes to the feedback structure and mechanisms. This will require the development of methods, techniques and tools whereby the process, including its feedback mechanisms, may be modelled, evaluated and implemented or changed to maximise the global benefit, however defined in any circumstance, obtained from each innovation. Given success it this activity, exploitation means will follow.

As a first step it is intended to model the process and its properties using appropriate techniques and representations to expose the role and impact of feedback in software evolution. A preliminary model has already been derived from process theory. Models derived from observation, measurement and analysis of industrial processes will follow once the process is in full swing with industrial collaboration. Detailed examination of the role and contribution of people in such mechanisms must be included. A necessary precursor to extended modelling activity is the adoption of formalism that permits adequate representation, at various levels of detail, of software processes with their feedback loops and mechanisms. Exploration of suitable techniques and representations must also include control theoretic and system dynamics approaches as well as formal languages such as those currently used in process modelling. Nevertheless, the proposed modelling activity will differ radically from the process modelling currently in vogue. The latter tends to divert attention from, even hide, feedback and global process properties in general. Project FEAST will focus on them.

The insight and understanding developed in the early stages of this integrated analysis of current software process technology will lead to process evaluation and improvement in terms of both forward and loop properties. Exploitation of existing and emerging development and support technology must be enhanced to address and exploit feedback properties and thereby yield improved process attributes. Methods, techniques and tools for the design and evaluation of feedback control mechanisms must be developed. New and improved mechanisms exploiting the potential of feedback must be developed. Finally, lessons learned must be applied to the extension of process theory and the generation of principles and guidelines that will facilitate the transfer of results of the study to software engineers responsible for design, support and improvement of the software process, to software developers and to their managements in industry and elsewhere.

FEAST studies have now (March 1995) been underway for some nine months. During that time three workshops involving participants from industrial, academic and research organisations in Canada, Finland, France, Norway, Poland, Portugal, UK and USA have been held. The main focus so far has been on the identification and definition of basic concepts, the adoption of outline definitions, preliminary examination of project issues and objectives and consideration of how best and most profitably the investigation should proceed. Funding by the UK Department of Trade and Industry is about to end. The rate of progress from now (April 1995) will depend on the further funding obtained. Success in the the project will ensure, sustain and extend future advances in the software evolution (development *ab initio*, enhancement, extension) process, yielding methods, tools and metrics for the systemisation of process technology, effective evaluation techniques, support tools, further improvement of the process. By its very nature the study will also make a significant contribution to process theory and the development of a scientific base and framework for software process technology.

The project is challenging but feasible. First practical results should be available within two years from the availability of adequate support. But in view of the difficulty of the issues under study and the spectrum of disciplines involved the main body of results is likely to require 3 to 5 years to achieve. The degree of success and the rate at which it is achieved will clearly depend, in part, on the funding obtained. The calibre of people attracted to and participating in FEAST suggests that significant progress can be anticipated.

References

[ABD91] Abdel-Hamid T and Madnick S E, Software Project Dynamics - An Integrated Approach, Prentice Hall, Englewood Cliffs, NJ 07632, 263 p.

[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

[BEN93] Benford S and Fahlen L, A Spatial Model of Interaction in Large Virtual Environments, Proc. Third European Conf. on Comp. Supported Cooperative Work - ECSCW '93, Milan, 1993, Michelis, Simona and Schmidt (eds), Kluwer Acad. Publishers, pp. 109 - 124

[BRO86] Brooks F P, No Silver Bullet - Essence and Accidents of Software Engineering, Information Processing 86, Proc. IFIP Congress 1986, Dublin, Sept. 1-5, Elsevier Science Publishers (BV), (North Holland), pp. 1069 - 1076

[CHO81] Chong Hok Yuen C K S, Phenomenology of Program Maintenance and Evolution, PhD Thesis, Dept. of Comp., Imp. Col. 1981

[GIB94] Gibbs W W, Software's Chronic Crisis, Scientific American, Sept. 2994, pps. 72 - 80

[GRI78] Gries D, Programming Methodology - A Collection of Articles by Members of IFIP WG2.3, Springer Verlag, New York, 1978, 437 p.

[LEH69] Lehman M M, The Programming Process, IBM Research Report RC xxxx, also in [leh85]

[LEH74] Lehman M M, Programs, Cities, Students - Limits to Growth, Imperial College. Inaugural Lect. Series, vol. 9, 1970 - 1974, also. in [gri78], pp. 42 - 69 and [leh85], pp. 133 - 163

[LEH78] Laws of Program Evolution - Rules and Tools for Programming Management, Proc. Infotech State of the Art Conf., Why Software Projects Fail, - Apr. 9 - 11 1978, pp. 11/1 - 25

[LEH80] Lehman M M, Programs, Life Cycles and Laws of Software Evolution, Proc. IEEE Special Issue on Software Engineering, vol. 68, no. 9, Sept. 1980, pp. 1060 - 1076

[LEH84] Lehman M M, Stenning V and Turski W M, (1984). 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

[LEH85] Lehman M M and Belady L A, Program Evolution, - Processes of Software Change, Academic Press, London, 1985, 538 p.

[LEH91] Lehman M M, Software Engineering, the Software Process and Their Support, IEE Softw. Eng. J. Spec. Iss. on Software Environments and Factories, Sept. 1991, vol. 6, no. 5, pp. 243 - 258

[LEH94] Lehman M M, Feedback, Evolution and Software Technology, Preprints of the First FEAST Workshop, Imperial Col., June. 1994

[LEH94B] Lehman M M, The Characteristics of S- and E-Type Systems, Preprints of the Second FEAST Workshop, Imperial Col., Nov. 1994

[LEH95] Lehman M M, Feadback, Evolution and Software Technology, Software Process Newsletter, IEEE, Apr. 1995

[MAJ93] Major J, Keynote Address, ICSE15, Baltimore, 17 - 21 May 1993

[NAU69] Naur P and Randell B, Software Engineering - Report on a Conference, Sponsored by the NATO Science Committee, Garmisch, 1968, Scientific Affairs Division, NATO, Brussels 39, 1969

[OXF81] See, for example, The Concise Oxford Dictionary, Seventh Edition, Apr. 1981, p. 355

[TUR81] Turski W M., Specification as a Theory with Models in the Computer World and in the Real World, Infotech State of the Art Report, se. 9, no. 6, 1981, pp. 363 - 377

[TUR86] Turski W M And No Philosophers Stone Either, Information Processing 86, Proc. IFIP Congr., Dublin, Sept. 1 - 5, 1986, Elsevier Sci. Pubs, London, pp. 1077 - 1080

[WOO79] Woodside C M, A Mathematical Model for the Evolution of Software, ICST CCD Res. Rep. 79/55. Also in J. of Sys. and Softw. vol. 1, no. 4, Oct. 1980, pp. 337 - 345 and in [leh85], pp. 339 - 354