Towards a Theory of Software Evolution
- And its Practical Impact

Working Paper, Ver. 1.0, 10 August 2000

M M Lehman J F Ramil
Department of Computing
Imperial College of Science, Technology and Medicine
180 Queen's Gate, London SW7 2BZ, UK
tel + 44 (0) 20 7594 8214; fax + 44 (0) 20 7594 8215
{mml,ramil}@doc.ic.ac.uk http://www-dse.doc.ic.ac.uk/~mml/feast

Abstract

After years of study of software evolution processes, most recently through the FEAST projects, it appears that now there is a sufficient body of knowledge that can serve as basis for the development of an axiomatic theory of software evolution. Such body is represented by, for example, patterns and behavioural invariants observed in attributes of industrially evolved systems. This paper discusses the need for such a theory, its practical impact, the underlying concepts and outlines an strategy for its development. As illustration, a set of statements (definitions, axioms, theorems) is given that leads to the principle of software uncertainty and to software evolution management guidelines.

1. Software Evolution

The phenomenon of software *evolution,* the consequence of an intrinsic need for continuing maintenance and further development of software embedded in real world domains, was first identified and studied in the early 70s [leh69,bel72]. Until recently, however, it did not arouse significant interest amongst academics or practitioners in the computer science, software engineering or software process communities. This situation has now changed. In recent years the interest in and study of the evolution phenomenon has spread as demonstrated by the recurrent appearance of the theme in events [e.g., pse98] and publications [e.g., pfl98,ben00]. Formulation of the FEAST hypothesis [leh94], three International FEAST Workshops [fea94,5], the results of the EPSRC funded FEAST/1 [leh96] and FEAST/2 [leh98] projects [feast], a recent (fourth) International FEAST 2000 Workshop with some 40 international participants and the publications that resulted [fea00a] have contributed to this development. Growing awareness is, in part, also due to the proliferation of desktop and personal computers, their increasing use by individuals, in industry, commerce, government and so on, and increasing use of the Internet. As users become ever more sophisticated and dependent on satisfactory system operation, the need for speedy, reliable, cost-effective evolution of their software through, for example, the provision of *management rules, tools and guidelines* [leh00a] has become increasingly apparent. It is forcing the continuing software evolution through upgrading or replacement that has become everyday experience for all serious computer users. Equally, increasing interest in business process improvement, resulting from the burgeoning integration of business processes and their internationalisation, has resulted in widespread recognition of the growing organisational dependence on software and, with it, the realisation of a need for business and software co-evolution [e.g., sebpc,soce].

In considering software evolution the FEAST group and a small number of other groups, e.g. [kem99], have viewed the word *evolution* as a noun [leh00c]. They are concerned with the *properties* of the evolution phenomenon, the *what* and the *why*. They seek to understand it by applying the scientific method of measuring, modelling, interpretation, hypothesis generation, as illustrated in figure 1. The primary goal has been to determine the underlying *causes*, *attributes* and practical *impact* of evolution on the software process and its products. Since the practical nature of the phenomenon as experienced in industry and by users was to be determined, examination of the evolution of a number of very different systems industry developed and supported systems was the first priority. As shown by its publications [feast], these studies have, and are still, yielded significant results including some that throw considerable light on the nature and attributes of software evolution. Figure 1 highlights the role of *theory,* the main theme of the present paper.

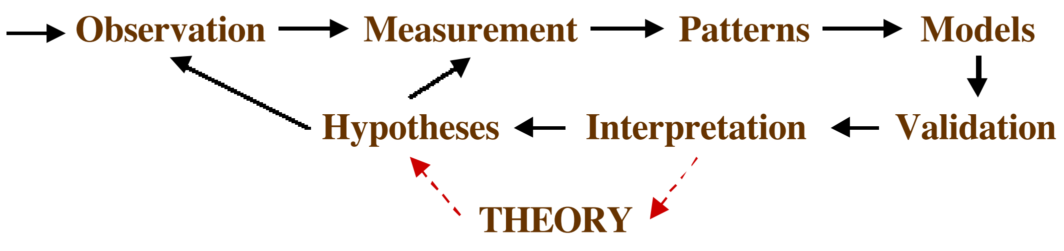

Fig. 1 - The Scientific Method

The overwhelming focus of software evolution studies is, however, on the *how* of evolution. The word *evolution* is interpreted as if it were a synonym of the verb *evolve* [e.g., gil81]. The concern has been and still is with improvement of the evolution process to achieve higher productivity, improved quality, faster development, greater adaptability and reliability and so on. The impetus for such investigation has come from the ubiquity of the computer, and hence of software, in all aspects of human activity. This has led to widespread, active, interest in process improvement in the context of software and, more generally, business processes, in a climate of expanding markets, increasing competition, advancing technology and, as evidenced by the EPSRC SEBPC initiative [sebpc], growing concern about business dependence on evolutionary adaptation and enhancement of its software; the need to achieve more effective co-evolution of a business, its organisation, processes, the marketplace, technology, and computer systems.

The noun and verbal views of evolution are complementary [leh00c]. The former is key to continuing improvement and for assessment of its potential practical value. Understanding repeatedly observed properties of evolution phenomena followed by development of a sound theory of evolution provides a base and framework for further improvement. But industry must constantly deal with immediate problems. It is, in the first instance, concerned with the *how* of evolution. However clearly and comprehensively properties of elements such as process triggers, process drivers and release sequence are recognised and understood, it is the discovery of ways and means to improve the process that brings direct benefit. This applies whatever the criteria for assessing and measuring the potential of individual improvements. And continued improvement is vital as society and individuals alike become ever more dependent on computers and on the software that gives them their functionality and power.

2. Background

One of the earliest studies of software evolution was triggered by a study of the IBM programming process [leh69]. One of the present authors (MML) has actively pursued the study of the topic ever since, supported at intervals by SERC, the US Army and, most recently, the EPSRC. Thirty year of observation and interpretation has produced results that include 8 laws of software evolution [leh74,85,96], the *SPE* program classification [leh85], a principle of uncertainty [leh89,90], the FEAST hypothesis [leh96] and FEAST/1 and /2 findings [feast]. There now exists deeper understanding of the software process, the nature and impact of feedback at both management and technical levels and the practical implications of the many observations and models [leh00b]. Major insight into the phenomena and relationships between them has followed. In particular, formulation of the feedback law, its extension to the FEAST hypothesis and observation of feedback-like behaviour in the systems studied suggest feedback as a basis for direct relationships between the laws. Common patterns of behaviour across a variety of systems, *behavioural invariants*, encourage the development of a software evolution theory.

3. The Role of Theory in Software Engineering

This paper introduces and illustrates plans for the development and formalisation of such a theory and indicates through examples how this has practical implications. As pointed out recently [for00] on the basis of extensive industrial experience, the determination of best practice, its transfer into industry and achieving widespread and willing acceptance requires one to overcome an inbred scepticism. Managers and practitioners must be convinced of its legitimacy and efficacy. The proposed theory, whose absence has been lamented by a number of authors over the years [leh85,be00], can serve this purpose.

Active interest over a period of more than thirty years in software process [leh69] has provided a body of knowledge yielding coherent, though incomplete, understanding of the observed behaviour. Seven years of intensive investigation immediately before, during and between the EPSRC supported FEAST projects [feast] has greatly strengthened the earlier insight, widened the domain of relevance and potential applicability of the concepts and observations and strengthened confidence in their validity. It is this *body of knowledge* that provides the foundation for the theory.

Note that development of a theory of software evolution is not, primarily, being proposed for the intellectual interest and challenge it presents. It is expected to have direct and immediate practical application and value. In connection with process improvement for example, it will provide a coherent framework that facilitates reasoning about the process and permits the derivation of qualitative and quantitative guidelines. The fragment of theory given below, for example, demonstrates that an assumption set is embedded in all real world software and that the set suffers inevitable and continuous invalidation. These observations lead to proof of the *principle of software uncertainty* [leh89,90] which identifies a major factor in software failure the world over. It follows that the capture, structured recording and regular review of assumptions, those explicitly stated, those implied by others and those implied by omission, must become an integral part of all software development and maintenance, that is of *software evolution*. Such is not established practice in industrial software development practice at the present time.

The above represents just one example of good practice that can be derived from a theory of software evolution. A recent paper [leh00a] collects together over 35 guidelines and rules for software evolution planning and management as derived from the FEAST observations and earlier work [lehweb,feast]. Many of the items on the lists that follow will appear intuitively self-evident. Others may seem like motherhood and without technical foundation or justification. What is new is the suggestions that they can be derived from and be based on a unified and coherent conceptual framework. It appears likely that the majority, if not all, of these recommendations, an abbreviated list of which follows, will be derived from the full theory when developed. Note also that in many case, individual items in this listing are compound statement (for compactness). In deriving them from a theory, relating them to the phenomenology it may well prove appropriate to split them into two or more statements.

Item	Paragraphs[1]	Description
1	3a	Prepare comprehensive specifications and maintain them updated
2	3b	Formalise specification (long term goal)
3	3c	Capture, document, structure and retain assumptions in specification
4	3d	Verify assumptions as part of verifying specification
5	3e,10c,13b	Continuing capture, updating and full documentation of assumptions during design, implementation, integration and validation work
6	3f, 10d, 13c, 13f	Develop and use tool support for specification work and recording of assumption throughout process in structured form (as database), classified by categories
7	l4a	Create and update comprehensive documentation to minimise impact of growing complexity
8	l4b	Apply conscious effort to control and reduce complexity and its growth
9	4c	Document assumptions and design and implementation rationale underlying a change
10	4d,13d	Periodically review assumption set
11	4e,8d	Plan for clean-up releases after major increments in functionality
		Observe safe change rate limits in planning and implementing change and evolution
12	4f	Constrain scope and size of a release increment based on past system successful incremental growth
13	4g,5d	Alternate major (functional enhancement, extension) and minor (clean-up, restructuring) releases
14	4h	Change validation must address change interaction with and impact on that part of the system that is not changed
		Change validation must include continued validity of assumptions
15	4i,10e	Assess domain and system volatility and take this into account in implementation
16	5a	Manage and control complexity in its many aspects
17	5b,10b	Pursue active complexity control (anti-regressive work)
18	5c	Determine and apply an appropriate level of anti-regressive work
19	6a,8a,c	Collect, plot and model historical data to determine patterns, trends and rates of change and growth
20	6b	Establish baselines of key measures over time and releases
21	6c	Use new data to assess, recalibrate and improve metric-based planning models
22	6d	Use '$m+2s$' criterion or similar to determine safe, risky or unsafe growth increments
23	6e	When required increment is large split into several releases as in evolutionary development
24	8b	Develop automatic tools to support data collection, modelling
25	10a	Follow established software engineering principles when implementing and evolving functionality (e.g., information hiding) to minimise interaction between system elements
26	11a	Consider, model, manage global process embedding technical process
27	11b	Consider, model, manage informal organisational links as part of global process
28	11c	Model dynamics of global process
29	11d	Use dynamic model to plan further work
30	11e	Use dynamic models to identify interactions, improve planning and control strategies
31	12a	Use dynamic models when seeking process improvement
32	13a	Estimate likelihood of change in areas of application domain and their impact on assumptions
33	13d	Periodic and event-triggered review and assessment of likelihood of change in assumption sets

[1] Paragraph numbers relate the present listing to that in [leh00a].

34	13g	Improve questioning of assumptions by, for example, using independent implementation and validation teams
35	13h	Provide ready access by evolution teams to all appropriate domain specialists

In association with literature references extending over a 30 year period, the paper [leh00a] provides an outline of the observations and phenomenological reasoning that has led to the above recommendations. Confidence in them, their widespread acceptability and, perhaps more importantly, their integration, extendibility and tool support would be greatly enhanced if they were shown to be the consequences of, related to, a coherent and convincing theory. A consistent theory will clearly make an important and systematic contribution to improvement of the software process.

4. Theory Development

The proposed staged development (for which support is now being sought) is to be rooted in the methods of *axiomatic theory*. Its inputs are a phenomenology based on 30 years observation and measurement of industrial software processes and interpretation of the results. Observations include patterns of behaviour and numerical measures of such behaviours. Some of these will constitute behavioural invariants, which may or may not be directly interpretable in terms of convincing phenomenology. Observations restricted to a subset of systems may also be considered for inclusion in the theory because of an interpreted relationship to characteristics of the domains to which they relate. Initially, the accumulated observations, knowledge and understanding must be assembled and organised in some meaningful structure. The elements of that structure will be expressed in a series of natural language definitions, axioms, theorems, corollaries and proofs. The axioms reflect fundamental insights and invariants about the real worlds of computer application and software development and maintenance. Informal notes of clarification and explanation will also be included. Many of these may eventually be recognised as a consequence of the basic axioms. They will prove to be theorems that follow from the axioms. Preliminary work suggests that the accumulated knowledge and understanding is sufficient to define a valid, viable and relevant theory. The underlying challenge is to state and prove theorems reflecting the axioms and then to derive practical implications. This would become the major aim of a project whose proposal its in preparation [leh00d]. The undertaking of such project will depend on the award and funding of the proposal.

5. The Proposed Initial Approach Exemplified

There follows a set of statements to exemplify the planned approach. The set has some interest in itself since, when filled out, it provides a proof of the principle of uncertainty. As presented here, proofs are limited to outlines since formal proofs can only emerge as the development proceeds. The selection represents a theory fragment, with many details of even the present restricted context not addressed. For example, the term *real world* is not defined. This is, in part, a matter of space. It must, however, be recognised that to determine a meaningful and appropriate definition involve issues whose resolution is closely bound up with the remainder of the developing theory. It would have to account the fact that the operational software becomes part of the real word that specifies it, that the system contains an implicit model of itself and that its development, installation and operation changes the real world (in its conventional sense) [leh77,84,85,tur00]. The real world is truly dynamic, always changing and the rate of change is increased by the development, installation and use of the system [leh85]. Other relevant issues are similarly glossed over in this example. The material here presented is tentative; must be expected change as the work proceeds, many outstanding issues are resolved, more of the observed phenomena are captured and represented.

Definition[2] 1	*E*-type programs are computer programs that solve a problem or address an application or supports an activity, in short that *address a problem*, in the real world.
Note[3] 1	Interest in *E*-type programs implies a belief that the specified properties are sufficient to ensure satisfaction when program is executed in specified domain.
Definition 2	A specification is a statement of the properties of a program believed to be necessary and sufficient to ensure that the program will be satisfactory (acceptable) to its stakeholders.
Note 2	The terms *satisfactory* and *acceptable* are here used informally and may be interpreted in the conventional sense. An adequate definition involves many issues whose resolution is an integral part of the development of the theory.
Definition 3	An *E*-type specification is an abstraction of a bounded real world domain and of a problem to be solved, such that the theory it constitutes can be modelled by a program that solves the problem.
Definition 4	Properties not addressed in the specification of an *E*-type program are declared to be of no concern. The program may or may not possess them.
Definition 5	The exclusion, explicit or implicit, conscious or unconscious, of an attribute of the real world from a specification represents an *assumption* that the property may be excluded from the *E*-type program it defines (without rendering the program unsatisfactory).
Note 3	Exclusion of a real world property from a specification is a property of the specification.

[2] Terms with an accepted mathematical definition, are to be so understood in the present context. See, for example, [tur81,87].

[3] 'Notes' are informal and for clarification. They may be become definitions, axioms, theorems or obsolete as development proceeds.

Definition 6	An assumption reflected in a specification is invalid if the *E*-type program derived from the specification is unsatisfactory for reasons associated with that assumption.
Definition 7	*Change* of the real world occurs when at least one of its properties changes
Axiom 1	The real world may be partitioned in an infinite number of ways into domains that, in general, each possess an infinite number of properties.
Axiom 2	The real world is dynamic, always changing.
Axiom 3	*E*-type programs as such (as distinct from such programs in execution) are finite.
Theorem 1	As the real world changes assumptions as reflected in the specification may become invalid, causing the specification to no longer be a model of the real world.
Outline proof	Definition 7, definition of *model*
Theorem 2	The real world domain abstracted into a specification is a model of that specification.
Outline proof	Definition 3, definition of model
Theorem 3	An *E*-type program is a model of, at least, the bounded real world domain identified by its specification.
Outline proof	Definition 3, theorem 1
Theorem 4	The abstraction of the real world from which the specification of an *E*–type program is derived may have an infinite number of properties.
Outline proof	Axiom 1, definition 4
Theorem 5	An *E*–type program reflects a finite number of properties of the specification.
Outline proof	Axiom 3, definition 2, 3. A finite program cannot implement an infinite number of distinct properties.
Theorem 6	To guarantee satisfaction in execution of an *E*-type program, its domain of execution must be consistent with the real world domain that is the source model of its specification.
Outline proof	Follows from definition (see note 2) of satisfactory
Theorem 7	As a model of its real world operational domain, an *E*-type program is essentially incomplete in the sense that there exist an infinite number of real world properties not reflected in it.
Outline proof	Theorem 4, definition 5, note 3
Corollary	An infinite number of assumptions are embedded in every *E*-type program.
Outline proof	Axiom 1, theorem 4, theorem 6
Theorem 8	A program which is a model of a specification that does not have the real world, as it is at the time of execution, as a model may be unsatisfactory even if previously satisfactory.
Outline proof	Axiom 2, theorem 1, theorem 5, theorem 8, definition 1 with note 1
Theorem 9	The behaviour of a program when executed is inherently uncertain, that is, cannot be guaranteed to be satisfactory.
Note 4	Previously referred to as the software uncertainty principle
Outline proof	Axiom 2, theorem 9
Guideline 1	Perform work in (evolve) specification and program to maintain model consistency under changes in the real world
Basis	Axiom 2, theorem 6
Guideline 2	Perform continuing capture, documentation, review of validity and impact analysis of sets of assumptions (underlying specification, program)
Basis	Theorem 7, theorem 9

The above statements briefly illustrate the proposed development including also two examples of the guidelines and their relationship to the emerging theory. They reflect a restricted set of observations and must be extended to yield a complete theory of software evolution. Even this example is not trivial and appears sufficient to prove the software uncertainty principle [leh89,90]. A full proof may require further axioms or revised definitions. When available, such a theory and its constituent statements abstract insights, interpretations and understanding of evolution as developed over the years, most recently, in FEAST. A theory so derived from observed phenomenology and behavioural invariants, its successful and convincing formalisation and extension by theoretical reasoning and with its theorems proven, will make a significant contribution to software engineering technology, providing foundations and a framework for further progress in technology improvement. As an integrated and coherent base and framework with direct implications on software process technology it also has implications for general business process improvement.

6. Project Outline

The concepts presented in the paper have been maturing for some years, triggered by the consistent results obtained over the years in our studies of software evolution. Based on the results now available and on the ever wider interest in software evolution the time is now ripe for systematic investigation and development. In the context of software engineering it represents a pioneering effort in uncharted waters. The development is by no means straightforward and requires widespread exposure and discussion; the involvement and contribution of as many researchers as possible. Hence our submission to ISPSE. A proposal for an active research investigation is currently being prepared [leh00d]. Initiation will depend on successful funding. As envisaged at the present time, the major objective of the project will be the development and formalisation a theory of software evolution and the derivation of implications in terms of practical rules, tools and guidelines for software evolution. The initial task of such project could be to expand the example presented in the previous section until a sufficient base established. Once this is accomplished, development will be greatly facilitated by switching to a formal representation. Modal action logic[4] has been suggested as a candidate for this purpose. This phase of the project presupposes availability of a sufficient body of codified knowledge to permit reasoned exploration and refinement to extract new forms, further relationships, theorems and interpretations to enrich the theory. It could begin some six month after project initiation and, thereafter, continue in parallel with axiomatic extension by interpretation of the phenomenology. Together the two techniques will facilitate reasoned exploration, interpretation, refinement and extraction of new forms with additional relationships and theorems to enrich the theory and provide pointers to its practical application.

As currently visualised, the project objectives must include:

- development of a formal theory of software evolution based on the phenomenological invariants and observed evolutionary patterns of industrially developed, marketed and maintained systems
- representation of the theory in both axiomatic form and by means of a formal symbolism
- derivation and proofs of theorems that follow from the selected axiom set
- interpretation of the theorems, their validate against real world data and determination of practical implications and potential for exploitation [leh00a]
- fully documentation and dissemination of the results of the project to encourage critical examination, adjustment of results where appropriate, extension of the theory and, above all, wide take up of the approach
- exploration of industrial exploitation.

To realise the objectives we propose the following steps:

- identification, classification and structuring of process patterns, invariants and other observations to provide a basis for theory development
- derivation and development of an initial set of definitions, axioms and theorems from the observations of previous activity
- develop proofs of these theorems.
- iterate from the identification step to extend theory and, in parallel, initiate formal development
- select formalism(s) and initial axiom/theorem set to test viability of formalism for interpretation, theorem proving and evaluate utility of selected formalism
- if unsatisfactory repeat else continue development of theory, taking into account desirable extensions to the axiom set and the models and interpretations under the previous step.

7. Related Work

The need for a theory, formal or otherwise, of the software process as such has been discussed in the author's writings and in the FEAST group for some time. There are also scattered references elsewhere to the absence of a theoretical basis and framework for software engineering and to the role that such a theory could play. For example, in a recent overview of the field Bennett and Rajlich state "...A major challenge for the research community is to develop a good theoretical understanding and underpinning for maintenance and evolution, which scales to industrial applications..." [ben00]. However, other than initial thoughts outlined in a recent workshop paper [leh00b], we are not aware of any existing work in the development of a theory, whether of the wider arena of software engineering or of constituents such as software process, evolution and maintenance. Mathematical theory that relates to formal representations, formal methods and programming languages does exist and may prove important in supporting the proposed study. But such theory is qualitatively different to that being proposed here. As a phenomenological descriptor the latter is more akin to the theories of the physical sciences [shr90].

The basis for axiomatic theory development as epitomised by Euclidean geometry and used for many centuries in the mathematical and physical sciences is well established. So is the application of *formal methods* and of the many representations and logics in computer science [tur87]. Success in the investigation will require access to the appropriate knowledge, understanding and experience to all these approaches through the co-ordinated work of an international team with expertise in the various related fields.

8. Final Remarks

The research hypothesis is that the software evolution phenomenon and the associated global process that is the sum total of all involvement in the evolution process, may be described by a formal theory, despite the role played by humans decision taking, execution and feedback in that process. As a result of the FEAST and earlier studies, it is believed that the necessary facts are available to demonstrate that this is so. In addressing this issue by construction, the other main issues that will arise relate to the selection of an appropriate axiom

[4] Private communication from Professor Jose Fiadeiro, University of Lisbon.

set and the application of the logics, methods and modelling techniques, briefly discussed above, to a domain where, it is believed, they have not previously been applied.

The long term benefit of the development of the theory addressed here is self evident. It is clear to many that its absence is a genuine obstacle to the advancement of software technology, the mastery of feedback in the software process and to major software process improvement, currently a matter of major concern, and expenditure, to industry.

As already indicated, some of the direct benefits accruing to the FEAST/1 and /2 industrial collaborators have been brought together in a report entitled Rules and Tools for Software Evolution Planning and Management [leh00a] which will be distributed to the collaborators shortly for use by their technical personnel. A shortened version, limited to a listing of the rules and guidelines and excluding the explanatory material and conceptual justification, is being prepared for management use. Once identified and put into context, much of the material appeals to common sense but practitioners may still question the validity or utility of the material as a body. As regards the latter there should, in fact be no question. The availability of a theory that supports, rationalises, integrates and extends the "rules" [leh00a] and, more generally, the results of the FEAST studies should play an important role in creating the confidence that is required to achieve the advances in software quality; increases in productivity and reductions in response time that are needed in an age and a society relying ever more on the computer and on software in all spheres of activity. The development of a theory in the forms proposed should facilitate reasoning about evolution and the process of evolution so leading to new theorems. When interpreted, these will in turn lead to further rules, guidelines, methods, procedures and tools. Even in the medium term, such a theory has significant potential for contributions to software process improvement and, more generally, for business and organisational process improvement.

9. Acknowledgements

Grateful thanks are due to our colleagues at Imperial College for their support, questioning and constructive criticism and to Professor Wlad M Turski for many hours of discussion and much enlightenment. Comments on the ideas presented in this paper by Professor Jose Fiadeiro are very much appreciated. Financial support from the UK EPSRC, grants FEAST/1 - GR/K86008 96/98, SVFs - GR/L07437 96/7 and GR/L96561 98/9, FEAST/2 - GR/M44101, SVFs - GR/N02412 00/01, is grateful acknowledged.

10. **References**[5] - for a full bibliography and copies of some of the most recent FEAST papers see links at [feast]

[bel72]* Belady LA and Lehman MM, *An Introduction to Program Growth Dynamics*, in Statistical Computer Performance Evaluation, W. Freiburger (ed.), Acad. Press, NY, 1972, pp. 503 - 511.

[ben00] Bennett KH and Rajlich VT, *Software Maintenance and Evolution: A Roadmap*, in Finkelstein, A. (ed.), The Future of Software Engineering, 22nd ICSE, June 2000, Limerick, Ireland, pp. 73 - 87

[fea94,5] *Preprints of the three FEAST Workshops*, Lehman MM (ed.), Dept. of Comp., ICSTM, 1994/5

[fea00a] *Preprints of FEAST 2000 International Workshop on Feedback and Evolution in Softw. and Business Processes*, Ramil JF (ed.), Dept. of Comp., Imperial College, London, 10-12 July 2000, 124 pp. http://www-dse.doc.ic.ac.uk/~mml/f2000

[feast] *FEAST Projects Web Site*, Dept. of Comp., Imp. Col., London, UK, Aug. 2000 http://www-dse.doc.ic.ac.uk/~mml/feast/

[for00] Fordham RG, *Software Development Challenges for the 2000's*, Keynote Address, Proc. PROFES'2000 2nd International Conf. on Product Focused Softw. Process Improvement, Oulu, Finland, 20 - 22 Jun. 2000, in Frank Bomarius and Markku Oivo (eds.) LNCS 1840, Springer Verlag, Berlin, 2000, p. 3

[gil81] Gilb T, *Evolutionary Development*, ACM Softw. Eng. Notes, April 1981

[kem99] Kemerer CF and Slaughter S, *An Empirical Approach to Studying Software Evolution*, Trans on Softw. Eng., v. 25, n. 4, Jul./Aug. 99, pp. 493 - 509

[leh69]* Lehman MM, *The Programming Process*, IBM Research Report RC 2722, IBM Research Centre, Yorktown Heights, N.Y., Sept. 1969.

[leh74]* Lehman MM, *Programs, Cities, Students, Limits to Growth?*, Inaugural Lecture, May 1974. Publ. in Imp. Col of Sc. Tech. Inaugural Lect. Series, v. 9, 1970 - 74, pp. 211 - 229. Also in Programming Methodology, (D

[5] Papers indicated with a '*' have been reprinted in [leh85].

Gries, ed.), Springer, Verlag, 1978, pp. 42 - 62

[leh77]* Lehman MM, *Human Thought and Action as an Ingredient of System Behaviour*, in The Encyclopaedia of Ignorance, R Duncan and M Weston-Smith (eds.), Pergamon Press, London, 1977, pp. 347 - 354

[leh84] Lehman MM, *A Further Model of Coherent Programming Models*, in, Proceeding of the Software Process Workshop, Potts C (ed.), Egham, Surrey, UK, Feb. 1984. IEEE cat. no. 84CH2044-6, Comp. Soc., Washington D.C., order n. 587, Feb. 1984, pp. 27 -35

[leh85] Lehman MM, and Belady LA, *Program Evolution - Processes of Software Change*, Acad. Pr., London, 1985

[leh89] Lehman MM., *Uncertainty in Computer Application and its Control through the Engineering of Software*, J. of Software Maintenance, Research and Practice, vol. 1, 1 September 1989, pp. 3 - 27

[leh90] Lehman MM, *Uncertainty in Computer Application*, Technical Letter, Comm.ACM, vol. 33, no. 5, pp. 584, May 1990

[leh94] Lehman MM, *Feedback in the Software Evolution Process*, Keynote Address, CSR Eleventh Annual Wrksh. on Softw. Ev. - Models and Metrics. Dublin, 7-9th Sep. 1994, Proc., Info. and Softw. Tech., spec. iss. on Softw. Maint., v. 38, n. 11, 1996, Elsevier, 1996, pp. 681 - 686

[leh96] Lehman MM, and Stenning V, *FEAST/1: Case for Support Part 2*, Department of Computing, Imperial College, London, UK, Mar. 1996. Available from links at the FEAST project web site [feast]

[leh97] Lehman MM, *Laws of Software Evolution Revisited*, EWSPT96, Oct. 1996, LNCS 1149, Springer, 1997, pp. 108 - 124

[leh98] Lehman MM, *FEAST/2: Case for Support Part 2*, Department of Computing, Imperial College, London, UK, Jul. 1998. Available from links at the FEAST project web site [feast]

[leh00a] Lehman MM, *Rules and Tools for Software Evolution Planning and Management*, FEAST 2000 Pre-prints, Imperial College, London, 10-12 July 2000. Available via links at [fea00a]

[leh00b] Lehman MM, *Approach to a Theory of Software Process and Software Evolution*, FEAST 2000 Pre-prints, Imperial College, London, 10-12 July 2000. Available via links at [fea00a]

[leh00c] Lehman MM, *Evolution as a Noun and Evolution as a Verb*, SOCE 2000 Workshop on Software and Organisation Co-evolution, 12 - 13 Jul. 2000, Imperial College, London. Available via links at [feast].

[leh00d] Lehman MM, Proposal in Preparation, Department of Computing, Imperial College, London, UK, Aug. 2000.

[lehweb] Publication listings available from links at http://www-dse.doc.ic.ac.uk/~mml

[pfl98] Pfleeger SL, *The Nature of System Change*, IEEE-Softw. v.15, n.3; May-June 1998; pp. 87-90

[pse98] Proc. Wrkshp. on *Principles of Software Evolution*, in conjunction with ICSE '98, Apr. 1998, 20 - 21

[sebpc] SEBPC *Systems Engineering for Business Process Change*, EPSRC Managed Research Programme, http://www.ecs.soton.ac.uk/~ph/sebpc/

[shr90] Shrager J and Langley P (eds.), *Computational Models of Scientific Discovery and Theory Formation*, Morgan Kaufmann Publishers, Inc, San Mateo, CA, 1990, 498 pps.

[soce] SOCE 2000 *Workshop on Software and Organisation Co-evolution*, Imperial College, 12 - 13 July, 2000

[tur00] Turski WM, *An Essay on Software Engineering at the Turn of the Century*, Invited Talk, ETAPS, Berlin, March, 2000, pp. 108 - 124

[tur81] Turski WM, *Specification as a Theory with Models in the Computer World and in the Real World*, System Design, Infotech State of the Art Report, P. Henderson (ed.), se. 9, n. 6, 1981, pp 363 - 377

[tur87] Turski WM and Maibaum T, *The Specification of Computer Programs*, Addison Wesley, London, UK, 1987, p. 278