

# An Approach to a Theory of Software Evolution

M M Lehman     J F Ramil  
Dept. of Computing  
Imperial College  
180 Queen's Gate, London SW7 2BZ  
tel. +44 - 20 - 7594 8214, fax +44 - 20 - 794 8215  
{mml,ramil}@doc.ic.ac.uk

## ABSTRACT

This paper outlines plans for the proposed development of a theory of software evolution. Apart from its intrinsic value, such a theory will advance understanding of the attributes of this phenomenon, its drivers and its practical impact on the software process and its products. If achieved, such a theory will provide means to identify and justify best practice in a world increasingly dependent on computers, where continuous software process improvement is of major, universal concern.

## Keywords

Best practice, management guidelines, software evolution, software evolution theory.

## 1. Introduction

The *software evolution phenomenon* [leh01b] was first identified as such in the early 70s [leh69, bel72]. It is reflected in an intrinsic need for continuing maintenance and development of software used to address an application or solve a problem in real world domains. Until recently, however, it did not arouse general interest. This has now changed as demonstrated by events such as IWPSE 2001. Growing awareness of the evolution phenomenon is, in part, due to the pervasiveness of computers, their growing use in industry, commerce, government and so on, and increasing utilisation of the Internet. All lead to growing societal dependence on software; artefacts that must remain satisfactory as the real world and, hence, the operational domain within which they are used change. As users become ever more sophisticated and dependent on satisfactory system operation, the need for speedy, reliable, cost-effective evolution of their software has become ever more apparent. Drivers such as competition, advancing technology, new opportunities, ambition are forcing continuing software system evolution through upgrading or replacement. Software *change* is an everyday experience for all serious computer users. Moreover, growing organisational dependence on software has resulted in widespread recognition of a need for continuing and effective business and software *co-evolution* [e.g. sebpc, soce2000].

In considering software evolution, the present authors and other groups [e.g., pf198, kem99, ben00, feast2000], have been primarily concerned with the properties of the phenomenon, the *what* and *why* of evolution [leh00a]. The goal has been to achieve understanding by identifying the attributes and practical impact of evolution on the software process and its products together with underlying drivers. 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 different industrially developed and supported systems was a first priority. These

studies have, and are still, providing results that throw light on the nature and attributes of software evolution [e.g., www]. These include elements that lead to *empirical generalisations* which, in turn, provide significant inputs for development of a *theory* of the phenomenon. This position paper outlines plans for the development of such a theory [leh00d].

In contrast to the *what* and the *why* view of evolution, the more general focus of software evolution studies is on the *how* of evolution [e.g., ffse01]. The concern has been, and still is, to find effective abstractions, formalisms, procedures, methods and tools, for example, for performing and improving the evolution process so as to increase productivity, reliability, dependability, adaptability and predictability, to improve quality, to decrease development time and so on. Understandably, this has led to widespread interest in process improvement as evidenced respectively by, for example, the SEI CMM model [pau93] and a recent EPSRC initiative [sebpc]. The two views of evolution are complementary [leh00a]. Both are required. In particular, development of the envisaged theory offers potential for well founded improvement and for assessment of the potential practical value of such improvement. Understanding the properties of the software evolution phenomenon and encapsulation of that understanding in a theory has the potential to provide a base and framework for further improvement. It is increasingly recognised as essential for further systematic control and improvement of the software process [e.g., ben00].

## 2. Precursor Investigations

One of the earliest investigations of software evolution was triggered by a study of the IBM programming process [leh69]. It has been actively pursued ever since. Thirty years of observation and interpretation has produced results that include eight *laws* of software evolution [e.g., leh74, 85, 97], the *SPE* program classification [leh85], a principle of software uncertainty [leh89, 90], the FEAST hypothesis [feast1994/5, leh94] and, more recently, the findings of the FEAST projects [leh96, 98, feastwww]. In particular, formulation of the eighth, feedback-system, law, its extension to the FEAST hypothesis and observation of feedback-like behaviour in several, otherwise very different, systems suggests feedback as a basis for relationships between the laws. Together with other contributions such as [wir71, gil81, kem99, kit99, ben00] they appear to provide significant understanding of the software evolution process, of the nature and impact of feedback at both management and technical levels and of the practical implications of these observations and models.

### 3. An Approach to Theory Formation

As its major input, the proposed development will exploit a body of codified knowledge, considered sufficient to permit disciplined exploration and refinement of candidate theories. Observations gathered during many years' of measurement and interpretation of industrial software processes provide the primary inputs to theory formation. They include *qualitative* and *quantitative* observations of *behaviour*. Some may even reflect behavioural *invariants*. Others will be restricted to a subset of systems, appear, therefore, to reflect common domain characteristics and so also lead, perhaps, to a lower level of empirical generalisations or *observational* laws [car66]. The accumulated observations, knowledge and understanding appear to be sufficient to start assembling and organisation into a theory. As this evolves, other observations become explainable in terms of it. Observations predicted from that theory should reflect fundamental insights and behavioural invariants, consistent behaviour across organisations and systems in the real worlds of computer application, software development and evolution. The figure below illustrates the basis of this procedure.

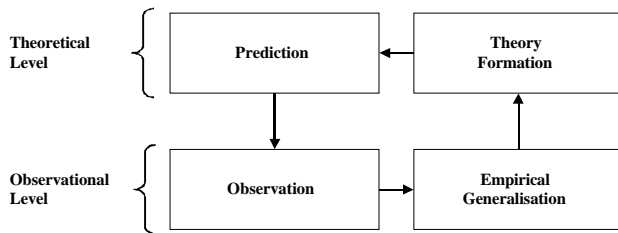


Figure 1 - Theory Development<sup>1</sup>

Once existing observations and empirical generalisations have been structured, theory formation starts. Procedures, such as the ones described in [shr90] should be of aid here. Moreover, the conclusion that the software process is a feedback system suggests that theory formation may be informed by control theoretic ideas. The discovery of which theory best explains the empirical observations and what its attributes are will provide clues to the key characteristics of the evolution process. Clearly, some phenomena will initially have to be ignored, approximations made to reality as in Physics or Engineering when friction is sometimes ignored. Whichever is adopted, results obtained will yield predictions to be validated against existing and new observations. Successful validation will strengthen confidence in the emerging theory, may extend its domain of relevance. Assessment of explanatory coherence and accuracy of predictions will guide the search for refinement and investigation of 2nd order phenomena.

### 4. Elements for Theory Formation

Given the current state of the study of software evolution, it is natural to form empirical generalisations, in the first instance, from elements such as observations, models and interpretations of existing data of software evolution [e.g. feastwww]. As these are assembled they provide openings for extension by reasoning along lines *illustrated* in the text box on the next column. Formation of the formal theory can follow.

The domain of the proposed theory and of the fragment introduced here relates to *E-type* software evolution. Such software includes "all programs that, when executed in a specified real world domain (the execution domain), solve a problem (or set of problems) defined in and part of that domain". Both domain and programs are models of (an explicit or implicit) specification that is itself an abstraction of the real world domain of interest that includes the problem to be solved. By definition, the requirement specification reflects all those properties of the execution domain that are required for acceptable solutions of the problem. That domain and the program will have additional properties not addressed by the specification. By omission or commission, such properties are declared to be of no concern in relation to an acceptable solution. An incompatibility between such additional properties, one from each domain is, therefore, of no concern, as long as the real world does not change. Change is, however, inevitable sooner or later. A property of either the domain or program previously accepted as of no concern may then block achievement of an acceptable solution. Worse still, what was previously regarded as an acceptable solution may no longer be acceptable. Thus, as the real world changes, one or more domain properties may become incompatible with the specification rendering the abstraction invalid. This inference leads into another, "if, as a result of changes in the real world execution domain, a specification is no longer a valid abstraction of that domain, the *E-type* program that models the specification may be unacceptable". It follows that "the behaviour of *E-type* programs when executed is inherently uncertain, cannot be guaranteed to be acceptable"; a restatement of the *principle of software uncertainty* [leh89,90].

Text including Elements for Theory Formation

### 5. Potential Benefits

The need and the contribution that such a theory of software evolution could make to the advancement of software technology has been recognised over the years [e.g., nat69,ben00,]. Such a development is now being seriously considered and planned. It is, however, not being proposed simply because of the intellectual interest and challenge it presents. Rather it is recognised that the proposed development may help to identify *best practice* and so help justify the additional cost of its deployment. Determination of best practice, its transfer to industry and achieving widespread and willing acceptance requires one to overcome inbred scepticism. Managers and practitioners must be convinced of its legitimacy and efficacy. To date little, if any, effort has been invested in the formation of process theory to demonstrate such legitimacy, despite several expressions of need for such a theory [e.g., ben00]. A theory should provide a unifying framework that encapsulates empirical generalisations and behavioural environments together with an explanation of why they occur, so providing a basis for justification of best practice. Moreover, a theory may act as a catalyst for further empirical work by providing, for example, a source of coherent hypotheses for empirical testing.

<sup>1</sup> Thanks are due to Prof. Tom Maibaum for bringing this approach to our attention

A theory can also have direct and immediate practical application and value. Current interest in software architectures [sha96], the search for reuse, pressures for moves to component and COTS based systems [leh00c], all reflect the fact that increasing human dependence on computers requires that software must, simultaneously, be made cheaper, more reliable, of higher quality and more *evolvable*. An explanatory theory that identifies sources of evolutionary pressures, the controls and constraints that stabilise the resultant evolutionary behaviour and the attributes of that behaviour, significantly advance the ability to architect and design systems for faster, more reliable and timely evolution.

From the point of view of process improvement, as widely understood and pursued, an explanatory theory provides a coherent framework, facilitating reasoning about the process and permitting derivation of qualitative and quantitative management and implementation guidelines. For example, theory already proposed [leh00b] demonstrates that an increasing number of elements of the *assumption* set embedded in all real world software will inevitably become invalid as time elapses. It follows that the capture, recording and regular review of the set, whether explicitly stated or implied by commission or omission, must become an integral part of all software development and maintenance, that is of software evolution. Such is not established practice even in such as sensitive areas as safety or business critical software.

The above provides just one simple example of good practice emerging from theory based reasoning. Many more such rules and guidelines for software evolution planning and management have been derived from FEAST observations and earlier work [feastwww,list]. They are discussed at greater length in a paper that outlines observations and reasoning that leads to specific practical recommendations [leh01a]. Confidence in them, their acceptability, integration, extendibility and tool support would be greatly enhanced if they were shown to be part of a coherent, explanatory theory. The latter would make a contribution to software process improvement, providing a conceptually sound rationale for best practice.

## 6. Related Work

The need for a theory of the software process as such has been discussed in the writings of one of the authors (mml) 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 thoughts recently outlined [leh00b], we are not aware of any existing work in *theory level* [car66] theory formation in the sense proposed here, whether in the wider arena of software engineering or of constituents such as software process, evolution and maintenance. Informal elements at the observational level can be found in *system dynamics* [e.g. feastwww] and other process models. Ontology and taxonomy work have been pursued [e.g. kit99] and may provide inputs to the proposed study. Mathematical theory relating to formal representation, 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. Despite differences due to human involvement in software evolution, as an observational descriptor the theory contemplated is more akin to the theories of the physical sciences [e.g., hem65,tha92].

## 7. Final Remarks

The research hypothesis is that the evolution process may be described by a formal scientific theory. Furthermore, the strength of feedback in driving and steering system evolution suggests that control theoretic concepts should find application in the proposed theory. How this is to be achieved remains to be determined. As a result of the FEAST and other studies, it is believed that sufficient conceptual foundations, empirical data and generalisations are available to start exploration of both hypotheses. Issues and challenges that arise, the selection of appropriate applicable formalisms and the application of the approaches to theory formation have been touched upon in this position paper. A first attempt to explore the further challenge relating to exploitation of the theory as a source and justification for rules, guidelines and tools [leh01a] for software evolution, has been suggested [leh00b]. The application of formal methods and of the many representations and logics in computer science [e.g., tur87, hae98,hae01] is also very relevant here. The proposed study will require access to the necessary knowledge and understanding of and experience in these approaches. The proposal is clearly a task for an interdisciplinary team and the involvement of others interested in taking up this approach is welcome. The development of satisfactory definitions and formalisation are amongst the first challenges that face the project. But if the theory formation from observations and behavioural invariants is successful, it will make a significant contribution to software engineering technology. It will provide foundations and a framework for further progress in technology improvement. It will also make a fundamental contribution to the development of software architectures for effective and reliable evolvable software, crucial for a world ever more reliant on software. Equally, it has important implications for general business process improvement.

## 8. ACKNOWLEDGEMENTS

Many thanks are due to our colleagues in industry and academia and, in particular, to Professor Tom Maibaum. By participating in discussions and in the revision of an early project proposal [leh00d], from which this position paper has developed, they have contributed significantly to development of these ideas.

## 9. REFERENCES<sup>2</sup>

- [bel72]\* Belady LA and Lehman MM, An Introduction to Program Growth Dynamics, in Statistical Computer Performance Evaluation, W. Freiburger (ed.), Academic 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, Limerick, Ireland, Jun. 2000, ACM ord. no. 592000-1, pp. 73-87

---

<sup>2</sup> References identified with an '\*' are reprinted in [leh85].

- [car66] Carnap R, Philosophical Foundations of Physics, Basic Books Inc. 1966
- [feast1994/5] Preprints of the three FEAST Workshops, Lehman MM (ed.), Dept. of Comp., Imp. Col., 1994/5
- [feast2000] Preprints of FEAST 2000 International Workshop on Feedback and Evolution in Softw. and Business Processes, Ramil JF (ed.), Dept. of Comp., Imp. Col., London, 10-12 Jul. 2000, 124 pps. Available via links at <http://www.doc.ic.ac.uk/~mml/f2000> <July 2001>
- [feastwww] FEAST Projects Web Site, Dept. of Comp., Imp. Col., <http://www.doc.ic.ac.uk/~mml/feast> <July 2001>
- [gil81] Gilb T, Evolutionary Development, ACM Softw. Eng. Notes, Apr. 1981
- [hae98] Haerberer AM, Maibaum TSE, The very Idea of Software Development Environments: A Conceptual Architecture for the ARTS Environment Paradigm, ASE'98, Redmiles D and Nuseibeh B, eds, IEEE Comp. Sc. Press, 1998
- [hae01] Haerberer AM and Maibaum TSE, Scientific Rigour, an Answer to a Pragmatic Question: a Linguistic Framework for Engineering, ICSE 2001, Toronto, Canada, May 12-19, 2001
- [hem65] Hempel C, Aspects of Scientific Explanation, Free Press, NY, 1965
- [kem99] Kemerer CF and Slaughter S, An Empirical Approach to Studying Software Evolution, IEEE Trans. Softw. Eng., v. 25, n. 4, Jul./Aug. 99, pp. 493-509
- [kit99] Kitchenham B *et al*, Towards an Ontology of Software Maintenance, J. of Softw. Maint., v. 11, 1999, pp 365-389
- [leh69]\* Lehman MM, The Programming Process, IBM Res. Rep. RC 2722, IBM Res. Centre, Yorktown Heights, NY, Sept. 1969.
- [leh74]\* *id.*, Programs, Cities, Students, Limits to Growth?, Inaug. Lect., May 1974, Imperial College of Science Technology Inaugural Lect. Series, v. 9, 1970-74, pp. 211-229. Also in Gries D., (ed.), Programming Methodology, Springer, 1978, pp. 42-62
- [leh85] Lehman MM and Belady LA, Program Evolution - Processes of Software Change, Academic Press, London, 1985
- [leh89] Lehman MM, Uncertainty in Computer Application and its Control through the Engineering of Software, J. of Software Maintenance, Research and Practice, v. 1, 1 Sept. 1989, pp. 3-27
- [leh90] *id.*, Uncertainty in Computer Application, Technical Letter, Comm. ACM, v. 33, n. 5, pp. 584, May 1990
- [leh94] *id.*, Feedback in the Software Evolution Process, Keynote Address, Proc. CSR Eleventh Annual Wrksh. on Softw. Ev. - Models and Metrics. Dublin, 7-9th Sep. 1994, Also in Info. and Softw. Tech., spec. iss. on Software Maint., v. 38, n. 11, 1996, Elsevier, 1996, pp. 681-686
- [leh96] Lehman MM and Stenning V, FEAST/1: Case for Support Part 2, Dept. of Comp., Imp. Col., London, UK, Mar. 1996. Available via links at [feastwww]
- [leh97] Lehman MM, Laws of Software Evolution Revisited, EWSPT96, Oct. 1996, LNCS 1149, 1997, pp. 108-124
- [leh98] *id.*, FEAST/2: Case for Support Part 2, Dept. of Comp., Imp. Col., London, UK, Jul. 1998. Available from [feastwww]
- [leh00a] Lehman MM *et al*, 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 [feastwww].
- [leh00b] Lehman MM, Approach to a Theory of Software Process and Software Evolution, FEAST 2000 Pre-prints, Imp. Col., London, 10-12 Jul. 2000. Available via links at [12] and with Ramil JF as Towards a Theory of Software Evolution - And Its Practical Impact, invited lecture, in Katayama T *et al.* (eds.) Proceedings ISPSE 2000, Intl. Symp. on the Principles of Software Evolution, Kanazawa, Japan, 1-2 Nov. 2000, pp. 2 - 11, IEEE CS Press
- [leh00c] Lehman MM and Ramil JF, Software Evolution Phenomenology and Component Based Software Engineering, IEE Proc. Softw., sp. issue on Component Based Software Engineering, v. 147, n. 6, Dec. 2000, pp. 249 - 255, earlier version as Tech. Rep. 98/8, Imperial College, London, Jun. 1998
- [leh00d] Lehman MM, An Approach to a Theory of Software Evolution - TheSE, EPSRC project proposal, DoC, Imperial College, Dec. 2000
- [leh01a] Lehman MM, Rules and Tools for Software Evolution Planning and Management, FEAST 2000 Pre-prints, Imp. Col., London, 10-12 Jul. 2000. A revised version with Ramil JF to appear in Annals of Softw. Eng., vol. 11, special issue of Softw. Management, 2001
- [leh01b] Lehman MM (with Ramil JF), Software Evolution, keynote lecture, in this volume
- [list] Publication listings available from links at <http://www.doc.ic.ac.uk/~mml>
- [mai00] Maibaum TSE, Mathematical Foundations of Software Engineering: a Roadmap, in A. Finkelstein (ed.), The Future of Software Engineering , ICSE 2000, June 4-11 Limerick, Ireland, ACM ord. no. 592000-1, pp. 161-172
- [nato69] Naur P and Randell B (eds.), Software Engineering, Report on a conference sponsored by the NATO Science Committee, Garmisch, 7-11 Oct. 1968, Jan. 1969, 231 pps.
- [pfl98] Pfleeger SL, The Nature of System Change, IEEE-Softw. v.15, n.3; May-Jun. 1998; pp. 87-90
- [sebpc] SEBPC Systems Engineering for Business Process Change, EPSRC Managed Research Programme, <http://www.ecs.soton.ac.uk/~ph/sebpc/> <Nov. 2000>
- [sha96] Shaw M and Garlan D, Software Architecture: Perspectives on an Emerging Discipline, Prentice-Hall, 1996
- [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.
- [soce2000] SOCE 2000 Workshop on Software and Organisation Co-evolution, Imp. Col., London, 12-13 Jul. 2000
- [tha92] Thagard P., Conceptual Revolutions, Princeton Univ. Press, Princeton NJ, 1992 pp. 285
- [tur87] Turski WM and Maibaum T, The Specification of Computer Programs, Addison Wesley, UK, 1987, 278 pps.
- [wir71] Wirth N, Program Development by Step-wise Refinement, Comm. ACM, v. 14, n. 4, Apr. 1971, pp. 221-227