An Approach to a Theory of Software Evolution

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

J F Ramil
Computing Dept., Faculty of Maths and Computing
The Open University
Milton Keynes MK7 6AA, U.K.
tel. +44 - 1908 - 65 4088, fax +44 - 1908 - 65 2140
j.f.ramil@open.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 the software evolution 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, empirical generalisations, FEAST, laws of software evolution, management guidelines, process improvement, software engineering.

1. INTRODUCTION

The *software evolution phenomenon* [leh01b] was first identified as such in the early 70s [bel72,leh69]. It represents an intrinsic need for continuing maintenance and development of software used in real world applications or to solve problems in a real world domain. Until recently, however, it did not arouse general interest. Events such as the sequence of IWPSE workshops [e.g. this volume] demonstrate that this has now changed. Growing awareness of the evolution phenomenon is due, amongst other factors, to the pervasiveness of computers, their growing use in industry, commerce, and government, increasing exploitation of the Internet and so on. 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 integrated, sophisticated and dependent on satisfactory system operation, the need for speedy, reliable, cost-effective evolution of their software has become ever more intense. Competition, advancing technology, new opportunities, ambition and so on are driving continued software system evolution through progressive enhancement, upgrading or even 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].

IWPSE 2001 Vienna Austria

In considering software evolution, the present authors and other groups [e.g., pfl98,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 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., feastwww]. These include elements that lead to *empirical generalisations* [car66] 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]. Together, understanding the properties of the software evolution phenomenon and encapsulation of that understanding in a theory provide a base and framework for further improvement. The former is increasingly recognised as essential for further systematic control and improvement of the process [e.g., ben00].

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., this volume]. 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, for example, by the SEI CMM model [pau93] and a recent EPSRC initiative [sebpc].

The two views of evolution, the *how* and the *what/why*, are complementary [leh00a]. Both are required. Together, and in association with the envisaged theory, they increase the potential for well founded improvement and for assessment of the practical value it can deliver.

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 [leh89,90,02], the FEAST hypothesis [feast94/5,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. They and other contributions [e.g., wir71, gil81,kem99,kit99,ben00], 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. 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 appearing, 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 two-level procedure.

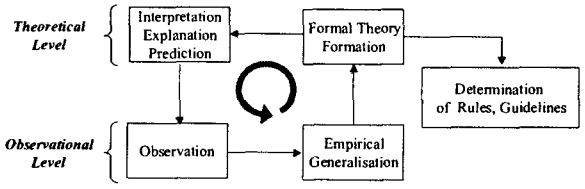

Figure 1 - Theory Development[1]

Various implementations of the two-level approach to theory formation exist. One example is provided by the *Carnapian* approach [car66], inspired by the terminology and reasoning of the experimental sciences. Another, inspired by abstract theories such as Geometry, is here termed *Euclidean*. These approaches are complementary and believed to have the potential to yield, not only a theory of evolution, but also evolution management rules, tools and guidelines. In principle, both approaches (and others?) could be pursued in parallel if the proposed project [leh00d] is launched. Consideration of others, their evaluation and the balance amongst them must be one of the initial tasks of any such project.

[1] Prof. Tom Maibaum. Private communication.

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 theory. Formation of candidate theories, identification 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 and approximations made to reality as in Physics or Engineering where, for example, friction is sometimes ignored.

Whichever approach 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 its explanatory and predictive power will guide the search for refinement and investigation of second order phenomena.

4. INPUTS TO THEORY FORMATION

Given the current state of the study of software evolution, it is, in the first instance, natural to form empirical generalisations from elements such as observations, models and interpretations of existing evolution data [e.g. feastwww]. As this is assembled it can provide openings for extension by reasoning along lines *illustrated* in the text below. 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 specification reflects all properties of the execution domain required for acceptable solutions of the problem. That domain and the program will have additional properties not addressed by the specification. By specific mention or by omission, such properties are declared to be of *no concern* in relation to an acceptable solution. Incompatibility between such additional properties, one or more from each domain is, therefore, of no concern, as long as the real world does not change. Change is, however, inevitable sooner 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* [leh89,90,02].

5. PROPOSED DEVELOPMENT

A brief example included as part of the ISPSE 2000 keynote lecture [leh00b] illustrated a possible transition between levels. Even though some of the statements have since then been revised, this contribution can still provide an example of what can be achieved and a flavour of what is intended. For the sake of brevity, the revised set of *definitions*, *observations* and *inferences* (theorem precursors) is not included here. It is available in the charts of IWPSE 2001 and of more recent presentations [feastwww]. Even the refined version constitutes an initial formulation requiring refinement. It is expected that this will be achieved through expansion of the scope of the theory and, for example, its formalisation.

Some brief comments on the intellectual process that led to the ISPSE 2000 example follow. The starting point was input such as the one provided in the text at the end of section 4. Based on it, sets of intuitive definitions and observations were identified. Individual observations should be eventually linked to empirical generalisations or, if they appear to hold on their own –as, for example, when based in common experience– proposed as axioms or theorems in the formal theory. Another element in the ISPSE outline is represented by inferences, derived from observations and their interpretations. The informal set, as such, suggests the basis for a proof of the software uncertainty principle [leh89,90,02] and suggests a number of practical management guidelines.

As shown in figure 1, the theory formation process requires iteration. This involves formalisation, refinement and validation of candidate sets of definitions, observations and inferences, leading to identification of a first one and eventually a succession of satisfactory and useful sets.

6. 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 such a development, if successful is likely to provide a rationale for some elements of *best practice* and, in doing so, help justify any 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]. It should support best practice by providing a unifying framework that encapsulates empirical generalisations and behavioural environments together with an explanation of why they occur. Moreover, a theory should act as a catalyst for further empirical work by providing, for example, a source of hypotheses for empirical testing.

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 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.

7. 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 on *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. 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 envisaged is more akin to the theories of the physical sciences [e.g., car66, tha92].

8. FINAL REMARKS

The research hypothesis presented in this paper is that the evolution process may be described by a formal scientific theory. Furthermore, the presence and strength of feedback in driving and steering system evolution suggests that control theoretic concepts should find application in this theory. That such a theory can be achieved in practice remains to be determined. It is considered that the FEAST and other studies provide sufficient conceptual foundations, empirical data and generalisations 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 briefly explored in this position paper. A first attempt to exploit aspects of the theory to provide 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,hae98,01] 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.

Theories are not developed over night but evolve over many years with contributions from many quarters. The duration and staffing of a project [leh00d] must permit those involved to assimilate the existing body of knowledge, master the skills required and then, systematically and progressively, evolve the theory at its various levels, following the process illustrated by the figure 1 or an alternative process. A project following this route will deliver intermediate outcomes such as hypotheses to be tested, empirical generalisations, implications and, hence, the axioms and theorems of candidate(s) formal theory. The individual results will be interesting and significant in their own right but do not in themselves constitute "a theory". The research must continue over a reasonably long period of time and at a sufficient level of activity, so that one achieves a critical mass that can reasonably be termed a theory. That is, a coherent and comprehensive set of relations, theorems and practical implications for testing in industry and for further exploration, validation or rejection, binding together and extension. More detailed objectives and intermediate outcomes can be identified with figure 1 providing a framework for their identification. Practical outcomes relate to industrial application of the theory as illustrated by the paper "Rules and Tools of Software Evolution Planning Management and Control" [leh01a].

If theory formation from observations and behavioural invariants is successful, it will make a significant contribution to software engineering technology. It also has potential to provide foundations and a framework for further progress in technology improvement and to make a contribution to the development of software architectures for effective and reliable evolvable software. All these are crucial for a world ever more reliant on software. Within that context it also has important implications for general business process improvement.

9. ACKNOWLEDGEMENTS

Our thanks are due to all who, anonymously or otherwise, commented on the drafts of a proposal [leh00d] upon which this contribution is based, and, in particular, to Professors Chris Hankin, Tom Maibaum, Dewayne Perry and Wlad Turski. Their participation in discussions and their willingness to share insights has contributed significantly to the development of these ideas.

10. REFERENCES[2]

[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, pp. 73-87

[car66] Carnap R, *Philosophical Foundations of Physics*, Basic Books Inc. 1966

[feast94/5] *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 <as of Jan 2002>

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

[hae98] Haeberer 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] Haeberer 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

[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

[2] References identified with an '*' are reprinted in [leh85].

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 [feast2000] and with Ramil JF as: *Towards a Theory of Software Evolution - And Its Practical Impact*, invited lecture, in Katayama T et al. (eds.) Proc. ISPSE 2000, Kanazawa, Japan, 1-2 Nov. 2000, pp. 2 - 11, IEEE CS Pr.

[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: Case for Support Part 2*, EPSRC project proposal, DoC, Imp. Col., Dec. 2000, rev. Sept 2001

[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,in Annals of Softw. Eng., vol. 11, special issue of Softw. Management, 2001, pp. 15-44

[leh01b] Lehman MM and Ramil JF, *Evolution in Software and Related Areas*, in this volume

[leh02] *id, Software Uncertainty*, Soft-Ware 2002, 1st Intl. Conference on Computing in an Imperfect World, Belfast, North Ireland, 8-10 April 2002, forthcoming

[list] Publication listings available from links at http://wwwdoc.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 of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO, 1969, 231pp, http://www.cs.ncl.ac.uk/people/brian.randell/home.formal/NATO/

[pau93] Paulk MC *et al*, *Capability Maturity Model*, Version 1.1, IEEE Software, v.10, n.4, 1993, pp. 18-27

[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.

[soce00] *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