Approach to a Theory of Software Process and Software Evolution
- Position Paper -

FEAST 2000 Workshop
10 - 12 July 2000
Imperial College
London SW7 2BZ

M M Lehman
Department of Computing
Imperial College
London SW7 2PH
+44 (0)20 7594 8214
mml@doc.ic.ac.uk
http://www-dse. doc.ic.ac.uk/~mml/

Three FEAST workshops were held at Imperial College during 1994/5 [fea94/5] to explore the FEAST hypothesis, itself formulated in 1993 [leh94]. The FEAST/1 project (1996 - 8) [leh95] funded by EPSRC followed and led, in turn, to FEAST/2 (1999 - 2001) [leh98]. Many of the results of these studies have been published over the past few years. They may be found on the FEAST web site at http://www-dse. doc.ic.ac.uk/~mml/feast.

As part of their investigation, the projects obtained evolution data on a number of systems from the formal collaborators, ICL, Logica, Matra-BAE and MoD-DERA and BT (FEAST/2). Similar data was also received from Lucent Technologies through the good offices of Professor Dewayne Perry, who, together with Professor Wlad Turski, are EPSRC Senior Visiting Fellows to the projects. The release-based systems studied had each been evolved in a sequence of from 15 to 30 releases over some eight to twenty years. Models and analysis of the data and interpretation of the results revealed striking similarities in the evolutionary patterns and long term trends of these systems. Moreover the newly observed patterns and trends were strikingly similar to those of OS/360 and several other systems studied in the 70s [leh98b]. This despite the fact that the systems studied were developed and evolved by different organisations, addressed different application areas and implemented distinct architectures using different languages. Moreover the systems studied differed in their size by up to two orders of magnitude and in the number of persons involved in their evolution by even more. Since the day to day control of the evolution process was in the hands of humans, differences between the several systems in their short term evolutionary behaviour were to be expected. The similarity of their long term behaviour, however, would have come as a surprise had not the 70s and 80 interpretation of the initial OS/360 observations, their subsequent phenomenological interpretation and the encapsulation of the observations and their interpretations in a set of laws of software evolution [leh74,78,80,96] prepared the investigators for such commonality. Thus the FEAST/1 results were seen as further support for six of the eight laws and supported many of the other conclusions that had been reached. The new evidence did, however, suggest some minor changes to the wording of the laws [leh98b].

While, over a period of fifteen years, the laws were being developed no thought was given to any relationship between them. Each aspect of the observed behaviour was seen as characterising some attribute of industrial team development and maintenance of software systems and captured in a *law*. In particular, the signs of self-stabilisation in the evolutionary behaviour was regarded as symptomatic of the behaviour of a feedback system [bel72, leh78]. This conclusion was ultimately expressed in the formulation of the eighth - Feedback – law and, some time later, the FEAST hypothesis [leh94]. But even then the set of eight were still regarded just as that, a set of eight independent, behaviour based, statements derived from observation of the real world. It was only with formulation of the FEAST hypothesis that realisation struck: the set of eight laws were likely to prove inter-related with the first seven reflecting facts that the eighth appeared to abstract.

Over the years, the laws were subjected to a number of criticisms. In particular, it was felt that the statements did not include precise definitions or statements of assumptions. Moreover, presenting them as “laws” appeared questionable to some. The author’s view, on the other hand, saw them as relating to organisational and sociological factors that lay outside the realm of software engineering, outside the responsibility of software engineers. Hence, from the point of view of the latter, they must be regarded as laws. As time passed and, in particular, with the pursuit of the FEAST projects, continuing discussions between those involved led to increased understanding and insight of the process and of the evolution phenomenon. It became more and more evident that the laws share underlying concepts and assumptions. Thus an overall challenge arose. Can the accumulated knowledge and understanding be shown to be or can it be developed into, the basis for or a part of a *theory*? This would be consistent with, for example, the Oxford definition of the latter term as a “set of reasoned ideas intended to explain facts or events” [oxf89]. And this led to the further question, “is the role of feedback in the process a key to the development of a theory of software evolution?”

One may ask whether the time is ripe for the development of a theoretical base and framework for a theory of software evolution? Exploration of the FEAST hypothesis, determination of the structure and nature of the relationship between the laws and development of a theory as posited poses many challenges. Difficulties arise, for example from the non-linear nature of the software process, from the major role that humans play in defining, controlling and executing it and from the lack of accurate models of process behaviour. Nevertheless, based on the insights and understanding gained during pursuit of the FEAST investigation we believe that the answer is “yes” and such a development is now being initiated. In the first place a classical approach is being adopted. This identifies and states (in natural language) a series of definitions and axioms, from which theorems are being derived and proven. Statements not initially proven are retained as hypotheses until formally proven or rejected by means of a counter example or otherwise. Eventually, we seek to develop a fully formal representation of the theory.

This work has now begun and a first outline is being prepared. As an illustration of the approach, some initial axioms and theorems are stated below. As presented here, they do not, and are not intended to, constitute even an elementary theory. They are provided to generate wider interest; to trigger comments and an injection of ideas from the workshop participants. Further tentative results are available as “work in progress”, but the availability of a theory that is coherent, complete in some sense, and satisfying, is some way off.

Def. 1.: An *S*-type program (software system) is an *executable* model of a formal *specification* [leh85].

Note[1] 1: That is: the *specification* of an *S*-type program is a *formal theory* and its *implementation* is a *model* of that theory [tur81,87].

Note 2: Successful *verification* demonstrates that the program *satisfies* the specification, that it is *correct*.

Note 3: All *properties* (attributes) defined by such a specification are properties of the program.

Note 4: Properties not addressed in the specification of an *S*-type program are, by definition, of no concern and may or may not be reflected in the implementation.

Note 5: Interest in the program derives from the fact that it is believed that possession of the specified properties *guarantees* desired *behaviour* of the program in *execution*.

Def. 2 An *E*-type program (software system) is a model, (also termed an *implementation*) of a specification that, as an abstraction of the *real world*, is itself a further model of that real world.

[1]“Notes”, while presently informal and for clarification, will be formalised and become axioms, theorems, or corollaries as development proceeds

Note 6: For an *E*-type program, all properties defined by the specification are properties of the program.
Note 7: Properties not addressed in the specification of an *E*-type program are, by definition, of no concern and may or may not appear in the implementation.
Note 8: Interest in the program derives from the fact that it is believed that these properties will ensure the desired behaviour of the program in execution in a designated portion of the real world.
Note 9: Conceptually, the real world may be partitioned into different *domains*, each possessing, in general, an infinite number of *attributes*.
Axiom I: The abstraction of the real world that is the *defining model* of the specification of an *E*–type program has an infinite number of attributes.
Note 10: The real world is also a model of the specification [leh84,tur00].
Note 11: The defining model that abstracts the features of interest from the real world which are applied to the development of the specification must be shown to be *satisfactory* in relation to the real world. Its being a model of the specification is a means to achieve such satisfaction.
Axiom II The implementation (also a model of the specification) is finite.
Note 12: If the *E*-type program is to execute *satisfactorily* when software *executed* in the real world, its domain of execution (the *operational domain*) must remain consistent with the abstraction that is the defining model of the specification
Theorem[2] 1 Every *E*-type program is essentially incomplete in the sense that there will exist infinite sets of real world properties that are not reflected in the implementation.
Def. 3: The exclusion, conscious or otherwise, implicit or explicit, of an attribute of the real world from the specification reflects an *assumption*.
Def. 4: An assumption reflected in a specification is *invalid* if the *E*-type program derived from the specification is considered *unsatisfactory* by human observers for reasons associated with that assumption.
Note 13: The real world is dynamic, always changing.
Theorem 2 As the real world changes, assumptions as reflected in the specification may become invalid. This may cause the real world to no longer be a model of the specification.
Theorem 3 An implementation 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 is *unsatisfactory*.
Axiom III The real world is *dynamic* and undergoes continuing *change*.
Theorem 4 The rate of change of the real world is, in general, accelerated by *installation* and *use* (execution) of an *E*-type program.
Theorem 5 The behaviour of a program when it is executed is inherently uncertain, that is, it cannot be guaranteed to be satisfactory.
Note 14 Theorem 5 and related behaviour has previously been referred to as the Software Uncertainty Principle [89,90].
Etc., etc.

The above is intended to do no more than to provide a preliminary introduction, extracted from work in progress and intended to illustrate an approach currently under development. If it can be successfully and convincingly completed, the resultant should make a significant contribution to providing software engineering technology with the theoretical foundations and framework needed to support further process and technology improvement. Expressing the theory in an appropriate formalism will represent a further advance. The present development is a first, essential, step to achieve this outcome.

Acknowledgements
My sincere thanks are due to my colleagues Juan Ramil, Dr Goel Kahen and Siew Lim for their support, questioning and constructive criticism and to Professor Wlad M Turski for many hours of discussion and much enlightenment.

[2] Proofs of theorems are not included in the present paper

References

[bel72] Belady LA and Lehman MM, *An Introduction to Growth Dynamics*, Proc. Conf. on Statistical Computer Performance Evaluation, Brown Univ., 1971, Academic Press, 1972, W Freiberger (ed.), pp. 503 -511

[leh74] Programs, Cities, Students, Limits to Growth?, Inaugural Lecture, May 1974. Publ. in Imp. Col of Sc. Tech. Inaugural Lect. Se., v. 9, 1970, 1974, pp. 211 - 229. Also in Programming Methodology, (D Gries ed.), Springer Verlag, 1978, pp. 42 - 62

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

[leh78] Lehman MM, *Laws of Program Evolution - Rules and Tools for Programming Management*, Proc. Infotech State of the Art Conf., Why Software Projects Fail, - April 9-11 1978, pp. 11/1 - 11/25

[leh80a] Lehman MM, *On Understanding Laws, Evolution and Conservation in the Large Program Life Cycle*, J. of Sys. and Softw., v. 1, n. 3, 1980, pp. 213 - 221

[leh80b] Lehman MM, *Programs, Life Cycles and Laws of Software Evolution*, Proc. IEEE Spec.. Issue on Softw. Eng., v. 68, n. 9, Sept. 1980, pp. 1060 - 1076

[leh84] *id.*, *A Further Model of Coherent Programming Models*, in, Proce*eding 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

[leh89] *id.*, 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] *id.*, Uncertainty in Computer Application, Technical Letter, CACM, vol. 33, no. 5, pp. 584, May 1990

[leh94] id., Feedback in the Software Evolution Process, Keynote Address, CSR Eleventh Annual Workshop on Software Evolution: Models and Metrics. Dublin, 7 - 9th Sept. 1994, Workshop Proc., Information and Software Technology, spec. iss. on Software Maintenance, v. 38, n. 11, 1996, Elsevier, 1996, pp. 681 – 686

[leh95] Lehman MM and Stenning V, FEAST/1 - *Feedback, Evolution And Software Technology; Case for Support*, EPSRC Research Proposal, parts 1-3, Dept. of Comp., ICSTM, March 1996

[leh96] Lehman MM, *Laws of Software Evolution Revisited*, Position Paper, EWSPT96, Oct. 1996, LNCS 1149, Springer Verlag, 19

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

[tur00] Turski WM, *An Essay on Software Engineering at the Turn of the Century*, Invited Talk, ETAPS, Berlin, March, 200097, pp. 108 - 124

[leh98a] *id.*, *FEAST/2 - Case for Support*, EPSRC Research Proposal, *parts 1-3*, Dept. of Comp., ICSTM, Jul. 1998

[leh98b] Lehman MM and Ramil JF, *Feedback, Evolution And Software Technology*, Keynote Lect., Proc. Int. Conf. on Softw. Eng. and its Applications, Session 1, Paris, 8 -10 Dec. 1998, pp. 1 - 12

[oxf89] Oxford Advanced Learner's Dictionary, Oxford University Press, 1989

[tur81] Turski W M, *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