

FEAST Workshop

16 - 17 June 1994

Department of Computing

Imperial College of Science, Technology and Medicine

180 Queen's Gate

London SW7 2BZ

+44 (0)71 589 5111 ext. 5009

fax +44 (0)71 581 8024

Contents

Organisational Matters

General Information	2
Program	3
Attendees	4

Briefing Papers

Introduction to FEAST Manny Lehman, ICSTM	6
System Dynamics Modeling of Software Development Tarek Abdel-Hamid, Naval Post Graduate School	11
Issues in Process Architecture Dewayne Perry, AT & T Research	12
The Wider Process and its Support Bob Snowdon, ICL	15
Engineering, Managing, and Improving Software Processes Marc Kellner, SEI	18
Language Issues Jose Fiadeiro, Universities of Lisbon and ICSTM	18
Year 1 FEAST Objectives Manny M Lehman, ICSTM and Vic Stenning, Anshar Ltd. and ICSTM	19
Workshop Tasks Manny M Lehman, ICSTM and Vic Stenning, Anshar Ltd. and ICSTM	26

Background Papers

Software Engineering, the Software Process and their Support Manny M Lehman, ICSTM The Software Engineering Journal, September 1991, pps. 243 -258	
ISPW7, ISPW8 and EWSPT93 Position Papers Dewayne E Perry, AT&T Proceedings of the Respective Workshops	
Thinking in Circles Tarek K Abdel-Hamid American Programmer, May 1993, pps. 3 - 9	
Process Modelling Bill Curtis, Marc I Kellner and Jim Over, SEI Comm. ACM, Sept. 1992, pps. 75 - 90	

General Information

- The workshop will be held on the 16/17 June in the Department of Computing, Imperial College, 180 Queen's Gate, London, SW7 2PH
- Hotel accomodation for those requesting it has been reserved at the Vanderbilt Hotel, 76 Cromwell Rd., SW7. Room charge £65 per night single, £85 double.
- The nearest underground station to the hotel is Gloucester Rd., District, Circle and Picadilly Lines. The College is equidistant from that station and South Kensington.
- Sessions will be in room 418 on the 4th floor.
- Tea, coffee breaks and lunches in room 433 on the same floor.
- Dinner on Thursday evening will be at 6 30 for 7 00 at 170 Queen's Gate.
- Mrs B Claxton, room 554, front corridor, 5th floor, will be available to assist attendees as required.
- In her absence assistance can be obtained from the Department general office, room 437.
- Incoming telephone calls direct to +44 (0)71 589 5111 ext. 7537 (Mrs Claxton)
- Departmental fax number is +44 (0) 71 581 8024
- email messages marked for your attention to bpw@doc.ic.ac.uk.
- Outgoing email can be sent from Mrs Claxton's machine or from mml's in room 556.

Program

Thursday June 16, 1994

9 15 - 9 45	Registration and Coffee		433
9 45 - 10 00	Opening remarks	Tom Maibaum	418
10 00 - 11 20	Introduction to FEAST	Manny Lehman	418
11 20 - 11 40	Coffee		433
11 40 - 12 20	System Dynamics Modeling of Software Development	Tarek Abdel-Hamid	418
12 20 - 1 00	Issues in Process Architecture	Dewayne Perry	418
1 00 - 2 00	Lunch		433
2 00 - 2 40	The Wider Process and its Support	Bob Snowdon	418
2 40 - 3 20	Engineering, Managing and Supporting Software Processes	Marc Kellner	418 418
3 20 - 3 40	Tea		433
3 40 - 4 20	Language Issues	Jose Fiadeiro	418
4 20 - 5 00	Objectives and Tasks	Vic Stenning	418
6 15 - 7 00	Cocktails		170 Queen's Gate
7 00 - 9 30	Dinner		

Friday, June 17, 1994

9 15 - 9 45	Coffee		433
9 45 - 10 15	Opening of General Discussion	Wlad Turski	418
10 15 - 11 00	General Discussion	Chair - Valerie Downes	418
11 00 - 11 30	Coffee		433
11 30 - 1 00	General Discussion Continued	Chair - Valerie Downes	418
1 00 - 2 00	Lunch		433
2 00 - 3 00	Planning Discussion	Chair - Vic Stenning	418
3 00 - 3 30	Tea		433
3 30 - 5 00	Wind up Discussion	Chair - Manny Lehman	418

Attendees

IC is installing a new telephone system with direct inward dialling. The numbers below for IC attendees should be operative from 1 July 1994. Until then or if you have problems use +44 (0)71 589 5111 and ask for person by name, Fax +44 (0)71 581 8024.

Dr Tarek K Abdel-Hamid,
Administrative Sciences, Naval Postgraduate
School, Monterey, CA 93943
tel: +1 408 646 2686, fax: +1 408 646 3407
3991P@NAVPGS

Alan Ainsworth, British Airways
Viscount House Annex, P.O. Box 10
Heathrow Airport, Hounslow
Middlesex, TW6 2JA
tel: +44 (0)81 562 4690,
fax: +44 (0)81 562 4690

Dr. Bob Balzer, Information Sciences Inst
4676 Admiralty Way, Suite 1001,
Marina del Rey, CA 90292-6695
tel:+1 213 822 1511
bob.balzer@venera.isi.edu

Mr. Bob Bishop, 18 Aggisters Lane
Wokingham, Berks, RG11 4DN
tel: +44 (0)734 774017
fax: +44 (0)734 894 254
rb@gid.co.uk

Anthony Briggins, ICSTM,
180 Queens Gate, London, SW7 2BZ
tel: +44 (0)71 595 8248
fax: +44 (0)71 581 8024
awb@doc.ic.ac.uk

Valerie Downes, Ovum Ltd.,
1 Mortimer St., WIN 7RH
tel: +44 (0)71 255 2670
fax: +44 (0)71 255 1995
100350.1627@compuserve.com

Prof. Jose Fiadeiro, ICSTM,
180 Queens Gate, London, SW7 2BZ
tel: +44 (0)71 595 8245
fax: +44 (0)71 581 8024
llf@doc.ic.ac.uk

Dr Anthony Finkelstein, ICSTM,
180 Queens Gate, London, SW7 2BZ
tel: +44 (0)71 594 8261
acwf@doc.ic.ac.uk

David Freestone, BT, Section B81, 160PP27
Martlesham Heath, Ipswich IP5 7RE
tel: +44 (0)473 648070
fax: +44 (0)473 648409

Dr. Peter Gibbins, Sharp Labs of Europe Ltd.
Edmund Halley Road, Oxford Science Park
Oxford, OX4 4GA
tel: +44 (0)865 747711
fax: +44 (0)865 747717

Dave Homan, Northern Telecom Europe Ltd,
Oakleigh Road South, New Southgate
London, N11 1HB
tel: +44 (0)81 945 4000
fax: +44 (0)81 945 3454

Dr John Iliffe, 37 Western Road
London, N2 9JB
tel: +44 (0)81 883 0939
jki@doc.ic.ac.uk

Marc Kellner, Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
tel: +1 412 268 7700
mik@sei.cmu.edu

Prof. Peter Lee,
Department of Trade and Industry
151 Buckingham Palace Road,
London, SW1W 9SS
tel: +44 (0)71 215 1967
fax:+44 (0)71 215 2909

Prof. Meir M Lehman, ICSTM
180 Queens Gate, London, SW7 2BZ
tel: +44 (0)71 594 8214
fax:+44 (0)71 594 8215
mml@doc.ic.ac.uk

Mrs Z Levy
City University, Northampton Square,
London EC1V OHB

Prof. Tom Maibaum, ICSTM
180 Queens Gate, London, SW7 2BZ
tel: +44 (0)71 594 8283/84
fax:+44 (0)71 594 8285
tsem@doc.ic.ac.uk

Dr Jeanne Pierre Moularde, SEMA (France)
16, rue Barbes
F-92126 Montrouge
Cedex, France
Tel: +33-1-40 92 43 16
Fax: +33-1-46 56 96 53

Bashar Nuseibeh, ICSTM,
180 Queens Gate, London, SW7 2BZ
tel: +44 (0)71 595 8286
fax:+44 (0)71 581 8024
ban@doc.ic.ac.uk

Brian Oakley, 120 Reigate Road
Ewell, Epsom, Surrey, KT17 2BX
tel: +44 (0)81 393 4096
fax: +44 (0)81 393 9046

Dr Dewayne Perry, AT & T Bell Laboratories
Room 2B 431, 600 Mountain Avenue,
Murray Hill, New Jersey 07974, USA
tel: +1 908 582 2529, fax: +1 201 386 2133
dep@research.att.com

Suzanne Robertson,
Atlantic Systems Guild Inc.,
11 St. Marys Terrace, London, W2 1SU
tel: +44 (0) 262 3395
fax: +44 (0)71 262 1378
100065.2304@compuserve.com

Dr Berc Rustem, ICSTM,
180 Queens Gate, London, SW7 1BZ
tel: +44 (0)71 594 8345
fax: +44 (0)71 589 8024
br@doc.ic.ac.uk

Graham Samuel
39 Laurier Rd.
London NW5 1SH
+44 (0)71 485 7916

Gordon Scarrott, 34 Parkway,
Welwyn Garden City, Herts. AL8 6HQ
tel: +44 (0)707 323 073

Dave Smith, Northern Telecom Europe Ltd.,
Oakleigh Road South, New Southgate
London, N11 1HB
tel: +44 (0)81 945 4000
fax: +44 (0)81 945 3454

Dr Bob Snowdon, ICL -STC, West Avenue
Kidsgrove, Stoke on Trent, ST7 1TL
tel: +44 (0)782 771000
fax: +44 (0) 777009
ras@kid01pml.icl.co.uk

Carolyn Story, BNR Europe, London Road
Harlow, Essex, CM17 9NA
tel: +44 (0)279 403648
fax: +44 (0)279 437830
pmp@bnr.co.uk

Prof. Vic Stenning, Warilda, Soames Lane,
Ropley, Hants SO24 OER
tel/fax: +44 (0)962 772531
vs6@doc.ic.ac.uk

Colin Tully
6 Meadow Hill Rd
Tunbridge Wells
Kent TN1 1SQ
tel/fax +44 (0)892 539 125

Prof. Wlad M Turski, Institute of Informatics
University of Warsaw
Banacha 2,00-903 Warsaw 59, Poland
tel: +48 2 658 3522
fax: +48 2 658 3164
wmt@mimuw.edu.pl

Aidan Ward, Coherent Technical Ltd.,
Suite 10, Challenge House, Sherwood Drive
Bletchley, Milton Keynes, MK3 6DP
tel: +44 (0)908 274591
fax: +44 (0)908 274693
Aidan@CohTech.co.UK

Dr. Alan Whitfield, 163 Sycamore
Wilnecote, Tamworth,
Staffs, B77 5HF

**The following had to cancel their
attendance because of
circumstances beyond their control**

Professor Vic Basili,
Dept. of Computer Science,
University of Maryland, College Park,
MD 20742
tel: +1 301 405 2668,
fax +1 301 405 6707
basili@mimsy.umd.edu

Nazim Madhavji, McGill University
School of Computer Science
McConnell Engineering Bldg.,
3480 University St, Montreal, Quebec,
Ca. H3A 2A7
tel: 514 398 3740/514 284 6730
fax: 514 398 3883
madhavji@opus.cs.mcgill.ca

Shari Pfleeger, City University,
Northampton Sq,
London, EC1V 0HB
Tel: +44 (0)71 477 8426,
fax +44 (0)71 477 8

Introduction to FEAST

M M Lehman
Department of Computing
Imperial college of Science, Technology and Medicine
London SW7 2BZ
tel.:+44 (0)71 589 5111, ext. 5009
fax: +44 (0)71 581 8124
email: mml@doc.ic.ac.uk

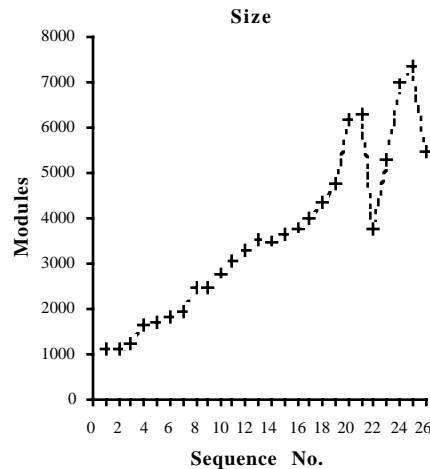
The means employed to develop *E*-type software (software that implements an application or addresses a problem in the real world [leh74]) have been significantly improved over the last thirty years. Numerous concepts, methods, techniques and tools have been developed and refined in an ongoing effort to achieve a more effective software development process. High level languages, structured programming, abstract data types, formal methods, non-procedural programming paradigms, object orientation, CASE, support environments, all these and many others were, each in turn, expected to overcome the problems that have frustrated the consistent achievement of cost effective, on time, development of functionally satisfactory and reliable software. Many, if not all, of the innovations have resulted in improvement in the local context of software development steps. The introduction of high level languages and of structured programming, for example, has increased coding effectiveness by several orders of magnitude. More generally, all the innovations have advanced individual and small group effectiveness on specific tasks. Individually and collectively they have enabled significant growth in the size and complexity of operational computing systems over the past thirty years. But none have proven to be the panacea, the silver bullet [bro86, tur86], anticipated by their proponents. It has proven equally difficult to improve the process that seeks to maintain a system satisfactory as its operational domain evolves with changes in user needs and expectations. Software *evolution*, whether from application concept to first implementation or from release to release still relies on processes that are far from satisfactory.

Reasons that explain the failure of individual improvements to yield anticipated levels of benefit are not difficult to find. The general failure, in the context of *E*-type evolution, of so many innovations raises the question whether there is not a common cause that has constrained them all? If this exists, identification could open the way to major process improvement opportunities.

An initial clue to the existence of a common cause emerges from the contrast between the significant advances that have been achieved in *S*-type programming (the development of programs required to be *correct* in the full mathematical sense with respect to a fixed specification) and the continuing problems haunting industrial software development and maintenance, the evolution of *E*-type software. The *S*-type approach is well represented by the work of the IFIP WG 2.3 group on programming methodology [gri78]. This recognises the need for iteration and backtracking over individual process steps, or even between a number of steps, to rectify errors or escape from *blind alleys*. Feedback from the output of the development or change process is, however, meaningless since, by definition, the *S*-type specification is fixed. Correctness of the final program relative to that specification is the *only* criterion of successful implementation. If or when for any reason the completed program, while technically correct, does not meet client needs a *new* specification, possibly derived from the previous one, must be generated and a *new* program to satisfy it must be developed. This *new* program too may be derived from the original, program. But technically it is a new program since it is defined by a new specification. Anticipating the conjecture presented below, one may observe that feedback from the *S*-type process output to its input has a long delay and low gain and does not result in an identifiable development dynamic.

In the context of *E*-type programs, on the other hand, *correctness* is meaningless. The real world application and its operational domain are potentially unbounded. Their descriptions cannot be absolutely or completely formalised to permit a correctness demonstration. Instead the criterion of *E*-type acceptability is *user satisfaction* with each program execution [leh91]. The limits of the application and the operational domain are dictated by pragmatic considerations, resources and technology for example, and are ultimately defined by the implemented program. Feedback expressing individual or group dissatisfaction of developers, users, marketeers or executives for example, will impact future development of the program. The very nature [tur81,leh91] of real world applications and of the *E*-type software that models and implements them sets up continuing pressure for change and system evolution. That is,

development and usage changes perceptions, leads to fresh insight and understanding, generates new needs and opportunities, changed expectations and criteria of satisfaction. The evolution process is a *learning process*, driven and controlled by information gathered, interpreted and used to direct subsequent activity or modify results of previous activity [leh69,85]. One aspect of that process was first studied in the early seventies and led to the identification of an evolution dynamics [bel72] and to the formulation of laws relating that dynamics to human and organisational behaviour [leh74,78,]. More recent work has examined and modelled organisational and managerial aspects of software evolution dynamics [abd91]. The resultant theory of software evolution [leh85,91, abd91] is supported by factual evidence. The growth of IBM operating system OS/360 from birth to its fission into VS1 and VS2 as shown in the figure, [leh80,85] is the earliest phenomenological evidence of the regular dynamic nature of software evolution subsequently confirmed in the systems of other manufacturers [leh80].



The cyclic pattern discernible in the plot is characteristic of self stabilising feedback systems. As observed at the time [leh72] "... the plot ripple is typical of a self stabilising process with positive and negative feedback loops. From a long-range point of view the rate of system growth is self-regulatory, despite the fact that many different causes control the selection of work implemented in each release, with varying budgets, increasing numbers of users reporting faults or desiring new capability, varying management attitudes towards system enhancement, changing release intervals and improving methods...". The period of instability beyond release twenty representing OS/360 fission into VS1 and VS2 some months after the '72 paper was published is equally indicative of the feedback nature of the software release process. The oscillatory behaviour indicates a loss of control over system evolution. It is consistent with all known facts that this chaotic behaviour was triggered by over ambitious growth targets, stemming from excessive positive feedback.

The 1969 - 72 observations from which the feedback nature of the software process was first inferred were restricted to evolution at the release level. Observation may, however be extended to other levels of the software process. Learning and feedback play a major role in determining many, if not all, of the dynamic aspects of *E-type* software evolution at all levels. The *E-type* evolution *process* constitutes a multi-loop system, with both positive and negative feedback. The characteristics of the feedback loops and mechanisms determine process dynamics. The process may therefore be expected to display the stable behaviour that is the hallmark of feedback systems in general. Externally observable system properties and behaviour remain relatively constant within specified limits over the operational range until instability sets in despite changes in the characteristics of forward path elements, the process environment and the operational environment.

These observations may have been interpreted as restricted to the transformation process that refines a computer application concept into a solution system. After all the 1970s investigation concentrated on technical development. Their relevance is, however, much wider. Business management, marketing, customer support, project and process management all apply feedback as, for example, in the use of monitoring and reporting mechanisms as checks and balances. Similarly, organisational processes use feedback procedure to ensure, for example, steady business and organisational growth with disciplined product evolution as user experience and changing client needs are fed back to development organisations.

Similar mechanisms moderate evolution of the industrial software process. In general therefore, changes to forward path elements of the software process cannot be expected to produce major global improvement unless accompanied by commensurate change to related feedback mechanisms. Software process improvement must therefore be pursued in the context of the total process domain; business management, budget control, marketing, customer support, personnel, project, process and information management, technical development, process improvement and monitoring of all these. Such a broad focus, treating the process as a feedback system provides a realistic framework already exploited [abd91] for the study of process dynamics. The dynamics is also implicit in laws of software evolution [leh74,78,85] which, while inferred from technical data, were interpreted in terms of human and organisational behaviour. It was, in fact, this interpretation that justified their description as *Laws* rather than as *observed behaviour*.

These observations reveal the *common cause* suggested above. The innovations listed there were all forward path mechanisms. Yet their introduction into practice did not, in general, include a feedback review process. Thus though that may have changed local process properties, given the feedback stability property it should not come as a surprise that the wider impact was far less than expected. It is conjectured that the common factor inhibiting major software process improvement is undue concentration on forward path innovation. After all as a feedback system, global process changes are, in general, unlikely to be achieved without explicit attention to feedback characteristics. The potential of feedback, even on its own, to produce significant benefit is illustrated by innovations such as *inspection, reviews, prototyping, incremental and evolutionary development* and software process *measurement*. These specifically feedback based techniques have, even in limited use, shown enough promise to constitute positive evidence for the conjecture.

Note that the conjecture consists of two separate and distinct assertions with the second arising if and only if the first is valid

Assertion 1. The *software evolution process* for E-type systems, which includes both the development of software *ab initio* and its subsequent maintenance, that is evolution, constitutes a complex feedback learning system.

Comment. This assertion is undeniable. Hence, to obtain full benefit from a forward path change to the software process will generally require feedback paths and mechanisms to be added, removed or modified. This is well known. The question to be asked and answered is whether this process can be systematised; whether software process feedback design can be disciplined. The problem arises because of the human role in the process. Humans observe, interpret, verbalise, transform, communicate, assess, decide, control and apply the forward and feedback process information. They manipulate the information fed over the feedback paths that link the activities, organisations, spaces [ben93] and people involved in and responsible for the evolution process. Humans participate in and observe the software in execution, use the results of that execution and feed back their observations and experience to the development organisation. Is their involvement so ingrained, so extensive, so individual, so judgmental and so creative that meaningful and exploitable formalisation and modelling with optimised design and integration of the feedback mechanisms is beyond reach, at least for the moment? Concern is not with the validity of the assertion but with its relevance, its significance and with development of means for its exploitation.

A crucial tool in providing means for its exploitation is the ability to model the process. Modelling techniques employed in current software process modelling activity do not, generally, provide the necessary facilities since they have not sought to reflect detailed feedback properties. But relevant formalisms and methods have been developed in other areas. Comprehensive formal theory and methods for automatic feedback systems design and control for both continuous and stochastic systems is embodied in, for example, control and dynamic systems theories. In the last thirty years or so both have been extended and more or less successfully applied to systems involving humans; economic systems and organisational dynamics for example. There have also been some studies of the application of control theory to software development [woo79, leh85, abd91]. There exists, therefore, a *prima facie* case indicating that despite intimate human involvement formal models fully reflecting feedback mechanisms may be successfully developed and applied to the design and improvement of software processes. The assertion implies that such models are essential for major process improvement; that one must seek a representation or representations that permit integration of process models, as required, of *all* aspects of the process.

The process may be observed, described modelled, analysed and changed in many spaces and at many levels of detail. Feedback occurs when information originating at the output of some process element is injected after some delay to the input of that or an earlier element. At lower process levels where individual action in the software process is relatively isolated feedback refers to the transfer of a single, in some sense spontaneous and unpredictable, package of information for use as seen fit by the recipient. Where, as at higher levels of the process, a continuing stream of (discrete) information is involved, *feedback* in the full engineering sense of the term, is established. The two usages do not really differ. They simply assume different situations. Rigorous representation of the former case is difficult and prediction of its consequences may be impossible. The analysability of the latter despite the non determinacy of human involvement, stems from the fact that the total information flow in the process system, generally involves many decisions and many decisors. The information fed back is a composite of many inputs which, whilst not totally predictable or independent, is amenable to meaningful statistical representation and analysis. The consequent system behaviour has been shown to display statistically *normal* properties [cho81]. It is therefore reasonable to expect that at these levels of the software process, control theoretic and statistical process models reflecting system dynamics can be developed for use on their own or in conjunction with modelling techniques currently in vogue. When statistical representation is not possible new formalisms permitting representation of the feedback mechanisms will have to supplement current process modelling techniques. At this level simulation techniques would likely constitute an important element of the design and evaluation process.

Assertion 2. The feedback nature of the evolution process *explains*, at least in part, the failure of forward path innovations such as those introduced over the last decades to produce impact of the order of magnitude anticipated at the global process level.

Comment. This assertion does not deny that many, if not all, of the innovations yielded significant benefit at the local level, improving the effectiveness of individual process steps or activities significantly. High level languages, for example, have certainly increased the quality, productivity and predictability of program code development and of code changes by an order of magnitude. The assertion merely states that because of the constraining effect of feedback mechanisms such benefit did not extend as fully as expected to the global level. This general failure need, of course, not be due to this common cause. Instead it may be due to reasons specific to each innovation. In view of the number of such failures a common cause must, however, be suspected. The feedback hypothesis provides an explanation that is consistent with an established property of other feedback systems. It is, therefore, of interest to examine the innovations individually in the context of processes within which they have been employed to determine whether limitations to global improvement achieved can be attributed to the feedback nature of the software evolution process or can be overcome by modifying or providing additional feedback mechanisms. Failure to succeed in either would not prove the conjecture invalid. It would cast doubt on its practical relevance.

The FEAST Project

A recently launched project, FEAST, (*Feedback, Evolution And Software Technology*) is exploring the feedback conjecture to discover ways in which it may be exploited. As observed above, the basic fact that the software evolution process constitutes a learning based feedback system is self evident. What must be investigated and demonstrated is the extent to which feedback phenomena have constrained the benefit derived from the introduction of innovative concepts, methods, tools and techniques in forward path mechanisms. Ideally one should be able to identify specific feedback paths that inhibited or damped the benefit obtained from individual innovations and to identify changes to the feedback structure and mechanisms that would produce additional benefit. At the same time it could be most rewarding to exploit the insight expressed in the conjecture; to develop methods, techniques and tools whereby the process, including its feedback mechanisms, may be modelled, evaluated and implemented or changed to exploit the potential benefits of each innovation to the fullest extent.

As a first step it is intended to model the process and its properties using appropriate techniques to expose the role and impact of feedback in software evolution. Models will be derived from process theory and from observation, measurement and analysis of industrial processes. Detailed examination of the role and contribution of people in such mechanisms will be included. A necessary precursor to this modelling activity is the adoption of formalism that permits adequate representation, at various levels of detail, of software processes with their feedback loops and mechanisms. Exploration of suitable techniques and representations must include control theoretic and system dynamics approaches as well as the formal languages such

as those currently used in process modelling. Nevertheless, the proposed modelling activity will differ radically from that currently in vogue. The latter tends to divert attention, even hide, global process activities and properties. Project FEAST will focus on them.

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

References

- [abd91] Abdel-Hamid T and Madnick S E, *Software Project Dynamics - An Integrated Approach*, Prentice Hall, Englewood Cliffs, NJ 07632, 263 p.
- [bel72] Belady L A and Lehman M M., *An Introduction to Program Growth Dynamics*, in *Statistical Computer Performance Evaluation*, W Freiburger (ed), Academic Press, New York, 1972, pp. 503 - 511
- [ben93] Benford S and Fahlen L, *A Spatial Model of Interaction in Large Virtual Environments*, Proc. Third Europ. C. on Comp. Supp. Coop. Work - ECSCW '93, Milan, 1993, Michelis, Simona, Schmidt (eds), Kluwer Acad. Pubs., pps. 109 - 124
- [bro86] Brookes F P, *No Silver Bullet - Essence and Accidents of Software Engineering*, Information Processing 86, Proc. IFIP Congress 1986, Dublin, Sept. 1-5, Elsevier Science Publishers (BV), (North Holland), pp. 1069 - 1076
- [cho81] Chong Hok Yuen C K S, *Phenomenology of Program Maintenance and Evolution*, PhD Thesis, Dept. of Comp., Imp. Col. 1981
- [gri78] Gries D, *Programming Methodology - A Collection of Articles by Members of IFIP WG2.3*, Springer Verlag, New York, 1978, p. 437
- [leh74] Lehman M M, *Programs, Cities, Students - Limits to Growth*, Imperial College. Inaugural Lecture Series, vol. 9, 1970 - 1974, also. in [GRI78], pp. 42-69 and in [leh85], pp. 133 - 163
- [leh78] *Laws of Program Evolution - Rules and Tools for Programming Management*, Proc. Infotech State of the Art Conf., *Why Software Projects Fail*, - April 9-11 1978, pp. 11/1 - 11/25
- [leh80] Lehman M M, *Programs, Life Cycles and Laws of Software Evolution*, Proc. IEEE Special Issue on Software Engineering, vol. 68, no. 9, Sept. 1980, pp. 1060 - 1076
- [leh85] Lehman M M and Belady L A, *Program Evolution*, - Processes of Software Change, Academic Press, London, 1985
- [leh91] Lehman M M, *Software Engineering, the Software Process and Their Support*, IEE Softw. Eng. J. Spec. Iss. on Software Environments and Factories, Sept. 1991, vol. 6, no. 5, pp. 243 - 258
- [tur81] Turski W M., *Specification as a Theory with Models in the Computer World and in the Real World*, Infotech State of the Art Report, se. 9, no. 6, 1981, pp 363 - 377
- [tur86] Turski W M *And No Philosophers Stone Either*, Information Processing 86, Proc. IFIP Congr., Dublin, Sept. 1 - 5, 1986, Elsevier Sci. Pubs, London, pp. 1077 - 1080
- [woo79] Woodside C M, *A Mathematical Model for the Evolution of Software*, ICST CCD Res. Rep. 79/55. Also in J. of Sys. and Softw. vol 1, no. 4, Oct. 1980, pp. 337 - 345 and in [leh85], pp. 339 - 354

System Dynamics Modeling of Software Development

Tarek Abdel-Hamid
Naval Post Graduate School
Monterey
CA

The objective of this talk is to present a paradigm for the study of software project management that is grounded in the feedback principles of system dynamics. System Dynamics is the application of feedback control systems principles to model, analyze, and understand the dynamic behavior of complex systems. Its origins trace back to the pioneering work of Jay W. Forrester.

The proposed approach has three unique features. First, the system dynamics model integrates the multiple functions of the software development process, including both the management-type functions (e.g., planning, control, and staffing) as well as the software production-type activities (e.g., design, coding, reviewing, and testing). A major deficiency in much of the research to date on software management has been its inability to integrate our knowledge of the micro components of the software development process such as scheduling, progress measurement, programming, and testing to derive implications about the behavior of the total socio-technical system in which the micro components are embedded. Clearly, this micro-oriented research work does provide a useful contribution to our understanding of the software development activity. However, before we can say that we have a complete understanding, it is necessary to show that our knowledge of the individual components can be put together in a total system.

The basic argument for this is that interactions and interdependencies are common in all social systems. The management system is a conglomerate of interrelated and interdependent functions. No one management subsystem can perform effectively without the others. Action taken by one subsystem can be traced throughout the entire management system and throughout the complex environment in which the management system exists. As a result, the behavior of an individual subsystem in isolation may be very different from its behavior when it interacts with other subsystems.

The second unique feature of the proposed modeling approach is the use of the feedback principles of system dynamics to structure and clarify the complex web of dynamically interacting variables. The software development process, like most organizational systems, is characterized by a complex conglomerate of interconnected feedback loops. This is perhaps most vividly captured by the classic waterfall model of the software development life cycle, where work units (whether analysis, design, or coding) typically re-circulate several times before leaving as a finished product.

The third feature, is the use of computer simulation. Computer simulation is utilized for two reasons. The first, is to handle dynamic complexity. The behavior of systems of interconnected feedback loops is often not intuitively obvious, even though the dynamic implications of isolated loops may be reasonably obvious. Second, computer-based simulation models are effective tools for controlled experimentation. The effects of different assumptions and environmental factors can be tested. In the model system, unlike the real systems, the effect of changing one factor can be observed while all other factors are held unchanged. Such experimentation yields new insights into the characteristics of the system that the model represents.

Introduction

The fact we refer to the software development process and provide a process support environment is a measure of the (im)maturity of much of process research. We do not have a software development process. We have a large set of processes. We have multiple, interdependent processes in much the same way that software products have multiple, interdependent components.

Because of this component structure of our development processes, we have architectural considerations that are both similar to and different from product architectures. The similarities stem from the fact that a process architecture is made up of data, processing and connecting elements [PW92] in much the same way that a product architecture is. The differences stem from the facts that 1) people are the primary (essential, dominant) processing agents that manipulate the data and perform the process activities and 2) the connecting elements are often organizational and social rather than just technological.

In the discussion that follows, I will consider these multiple, interdependent processes first within their architectural context and second within their organizational and social context.

The Architectural Context

While software development processes are clearly a domain-specific set of business processes, the actual set of elements in this domain-specific process architecture varies widely along several dimensions. The actual set of elements in a development process architecture will vary dependent on the size of the project (1-5, 20, 200, 2000 people, etc), the type of product developed (safety critical, prototype, etc), the type of approach (entrepreneurial, waterfall, spiral, evolutionary, etc), or the level of process maturity. These dimensions partition the process architectures into different process architectural styles.

More fundamental than the determination of architectural elements is the problem of defining what an architectural element is. In building software, we have a relatively good understanding what the various component granularities are. In defining processes, however, we do not yet understand and agree upon the various granularities needed to create a process architecture and we do not yet agree on the principles of process element decomposition [DP94]. We have a small vocabulary to describe these components, but no well-defined meanings or well-understand sizings. For example, the term process is used to mean anything from a small activity that may or may not be automated to the entire set of activities that is done in the development of a product.

In the context of a very large development project and organization that I have been working with, the term *process* is used to denote a large set of (perhaps concurrent) activities. Some of these activities include the use of tools but many are often done by people independent of a supporting technological context. In fact, even in the most tool intensive of these activities - for example, doing a system build [WR93] - the human element is critical and dominant over the technological. Certainly the large-grained processes are not of the automated or even automatable type. The critical difference between process processing elements and product processing elements is that they are entities that are performed by people - that is, humans, not tools are the processing agents. They may be formally modeled [P91], but much of the processing that is done by people is done informally and outside the context of technological support. Moreover, much of what is done even in the context of technological support is actually independent of that support.

Most of the data elements in process architectures are unlike those in product architectures in a way similar to the way that process processing elements are unlike their product counterpart. These data elements are generally informal documents. At best, various instances of the same kind of data element have the same general structure (often derived from the use of document

templates). While these data elements can be formally modeled [P91], they are not automatically manipulatable by process elements as they are in product architectures. Again, as with the process elements, these data elements are manipulated by people as the processing agents.

We are on firmer ground, however, in considerations of architectural form. Here, at least at one level of abstraction, there is an analogy between product architecture and process architecture. This level of abstraction is that of the basic relationships between architectural elements [CDP94]. As in product architectures, we have process elements that are independent of each other - they do not depend on each other in any way. We have process elements that are input dependent - they depend on another element for some form of input necessary to successful processing. We have process elements that function much the same way that subroutines do - they are nestable within another element. For example, the bug reporting process is performed within the context of some other process element (often coding or testing) causing the suspension of that element while it is performed and returning to the suspension point when it has been completed.

It is with elements that are both concurrent and dependent on each other where the analogy begins to break down. We have some concurrent dependencies that function along the lines of cooperating processes with well defined points of interactions, or even that function like co-routines. However, we also have instances of concurrent dependencies that are much more unstructured. For example, it is often the case, especially in large projects, that the design and coding processes interact in arbitrary ways - dependent on the circumstances rather than on well defined modes of interaction that we have in inspections, etc. It is in this informal and unstructured form of interaction that we need further work on descriptive notation.

We find one other analogue relationship in product architecture - namely, that of one element monitoring another. The most appropriate case of the kind of relationship is that of the progress monitor in a database system. Its function is to monitor the transactions and determine when progress cannot be made for some subset of the transactions because of resource contentions. Its job is then to reallocate resources so that at least some transactions can progress. The project management process element functions analogously within the context of a process architecture. Note, however, that we do not have the analogous supporting infrastructure that we have in database architectures. Neither do we have supporting notations to describe this kind of relationship between process elements.

The Organisational and Social Context

The analogy between process and product architecture breaks down even further when we consider connecting elements. Where processing and data elements exhibit some automated aspects, connecting elements exhibit even less automation. Practically all of the connecting elements are organizational and social in structure - and certainly informal and non-automated.

At the fine-grain level of connecting elements are various forms of communication between people such as electronic mail, phone-mail, phone conversations and face-to-face interchanges. We found in one of our process experiments [PSV94] that the automated forms of communications (specifically electronic mail) were used only for broadcast messages. Technical interchanges were invariably done either in person or over the telephone.

At the large-grain level of connecting elements are various forms of meetings, document handoffs, test laboratories, etc. I will consider several such elements and indicate some interesting experiments with these connecting elements that have illustrated significant improvements in both their structure and their performance.

Probably the most common process architecture connecting element is the document handoff. This element is as prevalent in process architectures as procedure call are in product architectures. Unlike a procedure call, however, it is a very lossy communication channel. The resulting documents represent but a pale shadow of the knowledge gained in its construction. This uncommunicated knowledge is needed in understanding the documents properly. Without it, the documents become a source of error injection. In one process experiment [PV93], the project was organized so as to minimize these handoffs and thus to minimize the error insertion aspects of the element. This was done by using interdisciplinary teams to maintain continuity of knowledge while handing off document from one process element to the next. In other words, the underlying structure of the process and connecting elements was changed organizationally to incorporate previously local data elements into the combined structure. Both the performance and quality of the resulting process architecture improved dramatically.

Another common connecting element is that of relying on management for technical decisions. This is a two-way connecting element where a request for a decision is sent up the hierarchy to the appropriate level of responsibility and eventually the response comes back down the hierarchy. The typical problem with this kind of connecting element is that it suffers from a wide degree of variance in response time. In another process experiment [PV93], the project was organized in such a way to minimize the time factor of this connecting element. This was done, again, by using interdisciplinary teams, with the added factor of decision empowerment. The time variance factors associated with the connecting element were removed. Again, both the performance and the quality of the resulting process architecture improved dramatically.

One of the problems that we have uncovered in our visualization and analysis experiments [CDP94] is the large amount of complexity introduced into the process architecture because of the lack of appropriate connecting elements. In particular, project management requires the production of a large number of artifacts (data elements) to be produced by the managed processes for them only. This increases the number of data elements significantly and the number of connections between the project management process element and other process elements. What is needed is a monitoring capability to serve as a connector between project management and various process and data elements as well as the underlying process state.

Yet another problem arises with respect to the constraints on the process elements. We found the following classes of process elements in our analysis of our current process architecture: processes, organizations and roles. While some roles and organizations are necessary, the primary problem is that they represent undefined process elements. Clearly, we must refer to customers as part of our process architecture. However, the primary class of process element should be processes. They in turn are decomposed into various subprocesses and tasks where the actual consumption and production of the process artifacts takes place. These processes are performed by organizations and their internal task steps are performed by people in various roles. Thus the distribution of the various data elements via the connecting elements should be between processes. Organizations (as groupings of people) and roles (as performed by people) are the processing agents of the process elements, not appropriate architecture elements.

Summary

Process architecture differs from product architecture in a number of important ways. First, where product architecture assumes computational processing agents for the various architectural elements, process architecture has in addition two radically different type of processing agents - namely, people and organizations. Second, where product architecture assumes components that have computationally well-defined semantics, process architecture usually has components that, while they may be modeled formally, are at heart informal with loosely defined semantics and unenforceable execution.

References

- [CDP94] David C. Carr, Ashok Dandekar and Dewayne E. Perry. *Experiments in Process Description, Visualization and Analysis*, April 1994.
- [DP94] Ashok Dandekar and Dewayne E. Perry. *Experience Report: Barriers to an Effective Process Architecture*, April 1994.
- [P91] Dewayne E. Perry. *Policy-Directed Coordination and Cooperation*, 7th International Software Process Workshop, Yountville CA, October 1991.
- [PSV94] Dewayne E. Perry, Nancy A. Staudenmayer and Lawrence G. Votta. *Understanding Software Development Processes, Organizations and Technologies*, March '94. Sub. for publ.
- [PV93] Dewayne E. Perry and Lawrence G. Votta. *A Tale of Two Projects*, July 1993. Technical Memorandum.
- [PW92] Dewayne E. Perry and Alexander L. Wolf. *Foundations for the Study of Software Architecture*, ACM SIGSOFT Software Engineering Notes 17:4 (October 1992), pp 40-52.
- [WR93] Alexander L. Wolf and David S. Rosenblum. *Process-Centered Environments (Only) Support Environment-Centered Processes*, 8th Int. Softw. Proc. Workshop, Dagstuhl Germany, March 1993.

The Wider Process and its Support

Bob Snowdon
ICL
Kidsgrove
Stoke on Trent ST7 1TL
UK
22nd May 1994

Note:

This document is very much working material on this subject ahead of the FEAST workshop, June 16/17 1994.

The Process and the Life-cycle

The software development process and the software life cycle are not, of course, the same things. The former relates to some constructive process, the purpose of which is the development of something called software, whereas the latter relates to the idea of the changing properties and relationships of software as some kind of artifact.

Obviously the two ideas are not unrelated. The software development process, after all, is the means by which software is formed and thereby gains (some of) its properties and relationships.

Historically, the software development process has been given a rather limited definition, as a *process* whose prime objective is to *produce* software artifacts starting from a specification or a requirements statement or whatever. Recognition of the need to provide support for users of these artifacts in the way of maintenance or enhancement is often reflected only in a phase, often identified as *maintenance*, being added to the end of the other, constructive or defining, phases of the process.

Studies of the software lifecycle, on the other hand, include properties and relationships for *software* in a much broader sense.

Realisation that there is also a broader *process* associated with the software lifecycle is important. Whereas the narrow view of the software development process often reflected concerns of the developer (or even, in certain cases, only those of perhaps the programmer or designer or project manager), a broader understanding must take account of the many other *players* who have effect on this thing called software.

Complex Nature of Software

It was, perhaps, possible to ignore such considerations when software was *simple*. That is, when a computer program was merely a means of creating a specialised calculator of some sort. However, the software systems of interest now possess a number of complex properties, particularly the way in which they interact with, affect and are affected by the environments (or systems) of which they are a part. The systems in which software is used are affected by such usage, which itself is an agent for change in that software.

Software and Other Systems

The extent of the complexity and uncertainty of these relationships is unpredictable and therefore requires ongoing consideration if the software development process is to be constructive. If a software system is inflexible, or the process of enhancement is incapable of reacting to new understandings about the relationships of the software with other systems, then that software will *die*. Its lifecycle will be foreshortened.

Such an observation is not, of course, new! Much of the vast amount of reported waste in software development effort or of the inappropriateness or rejection of software systems by customers or users may be attributed to inadequacies such as these. The goal, of course, is to gain an understanding of the *larger* software development process so that the life cycle of software is long and productive to all parties concerned.

The Wider Perspective of the Software Development Process

A strong message from the above is that the software development process must be understood from a wider perspective than has, perhaps, been the case traditionally. The nature of software is that its life cycle is affected by many factors, but not in ways which can generally be predicted. This suggests that the software development process be considered part of a larger process, or at least in the context of something larger. That is that the process of software development is a component of the development or change of systems more generally.

An example can be taken from the retail sector.

Retail companies compete with each other to satisfy their customers' needs. They compete in various ways, more and more taking advantage of computer systems to offer either direct benefit to their customers in the form of lower prices or new or improved services, or indirectly as being a more effective organisation (perhaps because of improved arrangements with suppliers or banks, again maybe using computer technology). Being competitive is a process of change in the retail system. If computing systems are to enable or facilitate this change, then they must be flexible and capable of enhancement. This in turn reflects upon the suppliers/developers of these systems who must be able to provide the required capabilities to the retail company. Nowadays, computer systems are built from many *commodity* components, so the requirement for change is, in turn, passed to the suppliers of these components and so on. And, of course, each of these companies themselves needs to be effective in its own market, so there is a continual need, not only to be able to satisfy their customers' needs, but to do so in ever more effective ways.

Value Chains

We get, therefore, a *value chain* type of model of the development process, with different organisations playing particular roles in the chain and receiving feedback from other participants in the process. Thus the model is not really a chain, but a structure which includes all the relevant feedbacks to keep the whole thing going. Each organisation will, of course, be part of many such *value structures* and will succeed (or fail!) according to how well it can take advantage of commonality amongst them.

The Nature of Software

An important factor in all of this, of course, is the nature of software itself. Computer systems have been described above as components in a *larger* system. A computer system is formed from a number of different technologies, but it is the software that, in the end, provides its distinctive characteristics. If computer systems are to be able to be used effectively to give ongoing added value to systems further up the value chain, then it essential that they possess the necessary characteristics. In the end this comes down to the properties of such systems and thus to the nature of their characterising software.

The development process could be considered as taking the nature of software as a given. However, this is clearly not true. Some attention must be given to determining what general characteristics software ought to possess, if it is to be an effective component in the sort of environment being discussed. Examples include concepts such as architecture (how should software systems be structured) and more specific things such as programming language and models. All of these clearly have effects on the development process.

Support for the Wider Process

Finally there is the question of how the broader process might be supported. Traditionally, support for the development process has mirrored the narrow view of its subject matter. Thus we have various models of software production, and tools which automate or facilitate particular aspects. These include compilers, editors and testing systems, tools for measuring such things as the number of branches in code, through configuration management systems and automated build to the idea of project management and IPSE. A slightly broader interpretation might allow us to include diagnostic support systems, upgrade and set up tools and so on.

The *value chain* model provides a slightly different point of view of the process and thus of what support might imply. The process spans a set of organisations, each using computer systems to make it (more) effective. These computer systems themselves are, to a greater or

lesser extent, supporting the software development process. Thus, for example, there are elements of the retail organisation concerned with developing the retail computing system. This will involve interacting with suppliers of that system, or its components. Similarly, the suppliers of these systems will interact with their suppliers and so on. Support for the process must therefore span these various organisations and will involve appropriate tooling concerned with the interactions that must take place. Further, each organisation is actually a separate component in all of this behaviour, with its own peculiar needs. Further still, each organisation is just that, an organisation. It must be expected that there should therefore be some commonality in how each is supported by computer systems in performing its part. Process modelling and process support technology appear to be valuable elements in providing the appropriate systems (at the least at a framework level) in all of these cases. Taken as a whole there is a recurring picture which appears to relate the software development process to the processes of organisations which are the subject of that development process and thus to relate means of supporting the development process (software development tools) to means of supporting the subject organisations processes (i.e. the software products themselves).

Engineering, Managing, and Improving Software Processes

Marc Kellner
Software Engineering Institute
Carnegie Mellon University
Pittsburgh
PA

This talk will briefly cover the two topical areas of:

- (1) process management, CPI, and TQM,
- (2) process engineering.

As suggested, the talk will also relate these to the FEAST issues and focus, most notably feedback and evolution, in an implications section.

Language Issues

Jose Fiadeiro
University of Lisbon
and
Department of Computing
Imperial College
London SW7 2BZ
jose@doc.ic.ac.uk
with
M M Lehman
Department of Computing
Imperial college of Science
London SW7 2BZ
email: mml@doc.ic.ac.uk

Process models are crucial to the FEAST investigation as they are to process improvement in general. In recent years many techniques and formalisms have been employed to represent such models with the choice at least in part, a function of the purpose for which the models were being developed; analysis, evaluation, communication, simulation, enactment, process guidance and so on. In that respect the FEAST use of models is no different. The special problem to be faced arises from the fact that models representing properties as covered by current process modelling technology will have to be employed in conjunction with others representing human process involvement and for representation, determination, examination and modification of the dynamic properties of the process. The latter may well be based on control theoretic, system dynamics or, more generally, statistical techniques, using representations and techniques that are quite different from those currently employed.

This need presents a major challenge. The lecture will concentrate on some of the representational issues with particular emphasis on areas and possible solutions in which the principle author has had previous experience.

Year 1 Objectives for FEAST

M M Lehman
Imperial College of Science Technology and Medicine
and
V Stenning
ICSTM and Anshar Ltd.

Introduction

This working paper identifies and outlines possible objectives for the FEAST project over its first months of activity to 31 March 1995. It is intended to stimulate discussion at the first FEAST workshop and to result in the adoption of specific goals that should include a technical and funding proposal for a collaborative follow on project. The list is presented in pseudo-logical not priority order. The feasibility of achieving a sub set of goals from the list presented, in the time and with the resources available under present funding, will depend on which groups are willing and able to undertake work in this period and on their experience, expertise and resources. It is hoped and expected that the exchange of viewpoints, concepts and experience at the Workshop and the interest created will lead to a definitive selection of feasible short term goals and to commitments and action by the participants and organisations represented to ensure their being successfully met. Positive results are needed to demonstrate the potential value and feasibility of the proposed investigation and so to obtain funding that will permit launching of the wider long term collaborative multi disciplinary activity required to bring this investigation to a successful conclusion. Such a conclusion would include a theoretical framework, constructive proposals for improvement of ongoing and future industrial processes and methods and tools to achieve such improvement. It should result in both process improvement and in an improved process improvement process.

Clarification

The FEAST conjecture consists of two separate and distinct assertions with the second arising if and only if the first is valid

Assertion 1. The *software evolution process* for *E*-type systems, which includes both software development and its maintenance, constitutes a complex feedback learning system.

Comment. This assertion is undeniable. Hence, to obtain full benefit from a forward path change to the software process will generally require feedback paths and mechanisms to be added, removed or modified. This is well known. The question to be asked and answered is whether this process can be systematised; whether software process feedback design can be disciplined. The problem arises because of the human role in the process. Humans observe, interpret, verbalise, transform, communicate, assess, decide, control and apply the forward and feedback process information. They manipulate the information fed over the feedback paths that link the activities, organisations, spaces [ben93] and people involved in and responsible for the evolution process. Humans participate in and observe the software in execution, use the results of that execution and feed back their observations and experience to the development organisation. Is their involvement so ingrained, so extensive, so individual, so judgmental and so creative that meaningful and exploitable formalisation and modelling with optimised design and integration of the feedback mechanisms is beyond reach, at least for the moment? FEAST concern is not with the validity of the assertion but with its relevance, its significance and with development of means for its exploitation.

A crucial tool in providing means for its exploitation is the ability to model the process. Modelling techniques employed in current software process modelling activity do not, generally, provide the necessary facilities since they have not sought to reflect detailed feedback properties. But relevant formalisms and methods have been developed in other areas. Comprehensive formal theory and methods for automatic feedback systems design and control for both continuous and stochastic systems is embodied in, for example, control and dynamic systems theories. In the last thirty years or so both have been extended and more or less successfully applied to systems involving humans; economic systems and organisational dynamics for example. There have also been some studies of the application of control theory to software development [woo79, leh85, abd91]. There exists, therefore, a *prima facie* case indicating that despite intimate human involvement formal models fully reflecting feedback mechanisms may be successfully developed and applied to the design and improvement of

software processes. The assertion implies that such models are essential for major process improvement; that one must seek a representation or representations that permit integration of process models, as required, of *all* aspects of the process.

The process may be observed, described modelled, analysed and changed in many spaces and at many levels of detail. Feedback occurs when information originating at the output of some process element is injected after some delay to the input of that or an earlier element. At lower process levels where individual action in the software process is relatively isolated feedback refers to the transfer of a single, in some sense spontaneous and unpredictable, package of information for use as seen fit by the recipient. Where, as at higher levels of the process, a continuing stream of (discrete) information is involved, *feedback* in the full engineering sense of the term, is established. The two usages do not really differ. They simply assume different situations. Rigorous representation of the former case is difficult and prediction of its consequences may be impossible. The analysability of the latter despite the non determinacy of human involvement, stems from the fact that the total information flow in the process system, generally involves many decisions and many decisors. The information fed back is a composite of many inputs which, whilst not totally predictable or independent, is amenable to meaningful statistical representation and analysis. The consequent system behaviour has been shown to display statistically *normal* properties [cho81]. It is therefore reasonable to expect that at these levels of the software process, control theoretic and statistical process models reflecting system dynamics can be developed for use on their own or in conjunction with modelling techniques currently in vogue. When statistical representation is not possible new formalisms permitting representation of the feedback mechanisms will have to supplement current process modelling techniques. At this level simulation techniques would likely constitute an important element of the design and evaluation process.

Assertion 2. The feedback nature of the evolution process *explains*, at least in part, the failure of forward path innovations such as those introduced over the last decades to produce impact of the order of magnitude anticipated at the global process level.

Comment. This assertion does not deny that many, if not all, of the innovations yielded significant benefit at the local level, improving the effectiveness of individual process steps or activities significantly. High level languages, for example, have certainly increased the quality, productivity and predictability of program code development and of code changes by an order of magnitude. The assertion merely states that because of the constraining effect of feedback mechanisms such benefit did not extend as fully as expected to the global level. This general failure need not, of course, be due to this common cause. Instead may it be due to reasons specific to each innovation. In view of the number of such failures a common cause must, however, be suspected. The feedback hypothesis provides an explanation that is consistent with an established property of other feedback systems. It is, therefore, of interest to examine the innovations individually in the context of processes within which they have been employed to determine whether limitations to global improvement achieved can be attributed to the feedback nature of the software evolution process or can be overcome by modifying or providing additional feedback mechanisms. Failure to succeed in either would not prove the conjecture invalid. It would cast doubt on its practical relevance.

Global Objectives

The general objectives of the FEAST project:

- establish the relevance of the first assertion of the FEAST conjecture and determine if and how recognition of the feedback effect may be exploited for software process improvement
- establish the validity or otherwise of the second assertion of the FEAST conjecture
- explore and develop process improvement methods, techniques and tools based on exploitation of feedback properties of the software process

Broad objectives for year 1:

- explore and adopt preliminary means for modelling, reasoning, evaluating and communicating about total processes with their feedback paths and mechanisms
- obtain evidence that supports the conjecture or that indicates that feedback phenomena are not significant or not systematically exploitable
- if, as expected, examination of the evidence is encouraging, identify potential opportunities for exploiting the conjecture and its implications to promote process improvement

Given these broad objectives, specific immediate goals must be considered:

Objective 1

Establish a common process modelling capability to permit intra-project communication

Rationale

The ability to model processes is fundamental to the FEAST project. Clearly, the models must cover all spaces and properties of interest (including, for example, feedback loops and broader organisational structures). While current process modelling notations may be appropriate for certain aspects of the investigation additional or alternative facilities will be required to model, assess, evaluate and design feedback and total system mechanisms.

Approach

Identify the properties needed. Examine established notations for modelling system behaviour including those already being used for software process modelling, in control theory and for system dynamics studies. Merge and integrate a suitable subset of these, providing also any additional facilities identified. This is expected to be an ongoing activity.

Feasibility

It may initially be difficult to identify the representational properties. One may have to be satisfied with selection of a small number of methods and notations and then merge and supplement these as experience is gained and progress made.

Objective 2

Further explore and develop control theoretic and system dynamics modelling techniques in the light of the conjecture and FEAST objectives:

Rationale

There have already been some successes in the application of these techniques to a limited number of aspects of software evolution [woo79, leh85, abd91]. The application of these approaches must be widened and intensified to address the total process.

Approach

Elaborate phenomenological process models to include all relevant (and possibly relevant) feedback elements and transform such more complete models into dynamic models.

Feasibility

Will not be simple but, now that the potential has been recognised progress should be possible.

Objective 3

Examine individually a subset of process innovations that appear to have failed to yield the anticipated benefit. Identify individual causes and/or feedback mechanisms that might explain the failure. Seek additional feedback mechanisms that might increase improvement to be obtained from individual innovations.

Rationale

Assertion 2 suggests that feedback effects have constrained innovations such as high level languages, structured programming, formal methods, object orientation, CASE, support environments and so on from delivering the anticipated improvements to the software process. Identifying constraining loops in some of these or improving their contribution by provision of additional feedback mechanism(s) is one way of exploring the assertion's validity.

Approach

Select a candidate innovation to be investigated. Construct models of both conceptual and actual industrial processes that exploit the selected innovation and all identified feedback loops. Determine the likely/actual impact of the innovation, that of the various feedback mechanisms and search for additional feedback paths or mechanisms for further improvement. Such an analysis should be carried out on several of the listed or other candidate innovations.

Feasibility

Given investigators with the necessary experience and understanding, the proposed investigation appears feasible. It requires individual(s) thoroughly familiar with all technical and organisational aspects of the process of some organisation. They, must have access to and consult with others having detailed knowledge and understanding the various aspects of that process eg. analysts, developers, integrators, all levels of management, marketeers, users, etc. It will be necessary to ensure systematic and disciplined study of total organisational involvement in the process. The principal problem will be how to evaluate the effects of the feedbacks identified or proposed in terms of the constraint or benefit they provide/yield. Until evaluation criteria and models have been developed such evaluation may have to be intuitive and qualitative, based on phenomenological or theoretical reasoning.

Objective 4

To establish the extent to which past process innovations introduced their own changes to feedback mechanisms, processes they encompassed and the consequences of their introduction.

Rationale

It has been suggested that introduction of many, if not all, of the innovations being considered may have included the modification of feedback mechanisms. On the other hand such changes may have been local (confined to activities directly related to the innovative concept) or otherwise limited rather than global (linking other activities in the overall process/organisation). Given extreme findings, this investigation would either support or negate the validity of assertion 2. The more likely outcome is intermediate findings that provide insight as to the degree to which feedback design is already a part of process design and improvement.

Approach

Select innovations to be examined and identify associated feedback paths and mechanisms. Identify any seen as directly associated with the innovation. Determine also whether addition and/or modification of, for example, more global paths (as identified, for example, by work on objective 3) could have improved the impact of the innovation in any way.

Feasibility

The work is perfectly feasible but its positive contribution to the study is unclear. Its main potential is as a counter argument to any attack on the assumption implied by assertion 2.

Objective 5

Study in detail an existing industrial process to identify feedback mechanisms constraining performance and opportunities to improve performance through the modification of existing feedback mechanisms or the addition of new ones.

Rationale

This objective and the associated investigation is complementary to that of objective 3. Instead of focussing on individual innovations it focuses on an industrial process and its total model, identifies its feedback loops, including informal ones that are not part of the recognised process, but stem from regular human contacts. The investigation must characterise and model the individual loops and mechanisms, seek to identify their contribution to process characteristics and any constraining impact, explore opportunities for additional, correcting feedback paths and mechanisms and undertake a preliminary estimate of the benefit the introduction of such additional or alternative mechanisms might bring.

Approach

This study must be based on a sufficiently detailed model of an existing industrial process including user support, maintenance, marketing, organisational planning and other management activities. It must be led by an individual with significant industrial process experience and the authority and contacts to permit consultation and discussion with experts in all relevant areas.

Feasibility

Suitable starting point models are known to exist in a number of organisations. If the right individual can commit to this study and given the necessary organisational support such a study could make considerable progress in the time available and provide significant results, though it would be unlikely to be exhaustive. Once again, the principal problem will be how to evaluate the effects of the feedbacks identified or proposed in terms of the constraint or benefit they yield. Until evaluation criteria and models have been developed such evaluation may have to be intuitive, qualitative, based on phenomenological or theoretical reasoning.

Objective 6

To investigate existing metrics and experience databases for feedback related data and models that could throw light on the issues raised by this project. Also to identify and obtain data that relates to the FEAST investigation relating to the properties of identified feedback loops (eg. delays, rate of package transmission, size of information packages, number of humans involved in transmission etc.) and investigate how that data might be used for:

- the investigation
- process improvement

Rationale

Existing metrics databases, for example the SEL collection at the University of Maryland or data collected by ICL in its current process studies, may already contain information to support or refute the FEAST conjecture. Such databases could help to further the FEAST investigation by providing data for, for example, the construction, driving, validation and evaluation of dynamic, statistical and total phenomenological process models.

Approach

Identify accessible sources of appropriate data. For each such database:

- classify information held
- identify data relating to feedback phenomena/activity
- apply to existing models or develop appropriate models that can reflect feedback properties
- define new FEAST related data types that might be extractable from the existing databases or that is presently unobtainable but could be extracted from future measurement activity
- Suggest further measurements/ investigations that might be undertaken

Feasibility

It is expected that access to existing databases will provide an effective entry point for the quantitative aspects of FEAST. The result will depend on what those familiar with the databases can identify and extract. Additional work will be required to develop appropriate investigative techniques and models and to exploit those models in the context of the project.

Objective 7

From a broad understanding of the industrial software development and maintenance environment, search for, explore and evaluate ideas for new feedback mechanisms to improve the software process or modifications to existing ones.

Rationale

The challenge posed by FEAST in terms of the magnitude, difficulty and fundamental nature of the investigation cannot be met by a single group funded for one year. The initial investigation being planned at this workshop must extend insight into the feedback characteristics and

behaviour of the software process. It must initiate activity to develop constructive and practical improvement proposals that will provide an *a priori* case for further study and development. It must provide material input to support an application for funding for the extended study that will be required to fully investigate the improvement potential of feedback control of the process and to develop means for its exploitation.

Approach

Examine and probe existing high level process models that claim to be reasonably complete to search for feedback needs or opportunities.

Feasibility

A systematic investigation is speculative and difficult to discipline but not impossible for individuals with appropriate background. The initial approach might be to engage in a free wheeling or brain storming discussion amongst process knowledgeable people driven by some some reasonably complete process model together with critical examination of that model.

Objective 8

Prepare a proposal or proposals for submission to one or more funding bodies for continuation of the project beyond year one.

Rationale

The challenge posed by FEAST in terms of the magnitude, difficulty and fundamental nature of the investigation and the diversity of the potentially relevant technologies and skills required cannot be met by a single group funded for one year. A three year investigation by a mixed group of collaborators, at least, will be required to obtain reasonably complete results with practical application. This initial investigation must prepare the necessary material and proposal involving all potential collaborators able to make a contribution and wishing to be included.

Approach

Mutual consultations and joint preparation with one Proposer in Chief to drive the effort.

Feasibility

In view of the potential significance and practical importance of this investigation we should be confident of a positive response if the right proposal is presented to the right body or bodies by the right combination of collaborators in the right way at the right time.

Final Summary and Initial Proposal for Debate

How many of the candidate objectives proposed above can be undertaken and met will clearly depend on which organisations, groups and individuals are able to make commitments to devote time to FEAST over the coming months. Objective 8 must be undertaken and met if the investigation and development is to continue. Of the others 5 and 7 appear to be the potentially most rewarding but others should also be considered if at all feasible. The metrics aspects as reflected in objective 6 are particularly important and it is hoped that work in this area can be spearheaded by Vic Basili and the group at Maryland. Vic was, most unfortunately unable to participate in the workshop nor were various alternatives proposed able to attend. He is, however, already committed to working on the project and I trust that some metrics work will get under way despite the fact that this area will not have been represented at the workshop.

References

- [abd91] Abdel-Hamid T and Madnick S E, *Software Project Dynamics - An Integrated Approach*, Prentice Hall, Englewood Cliffs, NJ 07632, 263 p.
- [ben93] Benford S and Fahlen L, *A Spatial Model of Interaction in Large Virtual Environments*, Proc. Third European Conf. on Comp. Supported Cooperative Work - ECSCW '93, Milan, 1993, Michelis, Simona and S Achmidt (eds), Kluwer Acad. Publ., pps. 109 - 124

[leh85] Lehman M M and Belady L A, *Program Evolution*, - Processes of Software Change, Academic Press, London, 1985

[woo79] Woodside C M, *A Mathematical Model for the Evolution of Software*, ICST CCD Res. Rep. 79/55. Also in J. of Sys. and Softw. vol 1, no. 4, Oct. 1980, pp. 337 - 345 and in [leh85], pp. 339 - 354

Proposed Workshop Goals Day 2

M M Lehman and V Stenning

Morning - refine objectives

The FEAST conjecture - clarify, modify, refine

Overall FEAST objectives - clarify, modify, refine

Year 1 objectives - identify omissions, clarify, modify, refine, prioritise

Afternoon - Produce an outline plan for year 1

Identify groups and/or individuals willing and able to make commitments

Identify group objectives and approaches that might be taken

Identify means of interaction between groups and individuals

Discuss outline plan including possible further meetings

AOB