

Preprints

## FEAST Workshop III

28 February - 1 March 1995

Department of Computing

Imperial College of Science, Technology and Medicine

180 Queen's Gate

London SW7 2BZ

+44 (0) 171 594 8214

fax +44 (0) 171 594 8215

## Contents

### Organisational Matters

General Information .....	2
Program .....	3
Attendees .....	4

### Briefing Papers

Summary of Post-Workshop II discussion Manny Lehman, Dewayne Perry, Wlad M Turski .....	6
A Somewhat Overdue Introduction Wlad M Turski .....	8

### Examples and Position Papers

A Hierarchy of Feedback Loops Tarek K Abdel-Hamid .....	11
The Role of Feedback in Knowledge Management Keith H Bennett .....	12
Feedback in the Spiral Model Barry Boehm .....	13
A View from BNR Dave Homan .....	14
An Example of Apparently Overlapping Control Loops Manny Lehman .....	15
Macroscopic Feedback Systems Nazim Madhavji .....	17
FEAST Modelling Language Suzanne Robinson .....	19
Fruits of an Initial Search for an Example Carolyn Story .....	21
A Survey and Comparison of some Research Areas Relevant to Software Process Modelling Terje Totland and Reider Conradi .....	22
Using Process Waiver Data to Improve a Design Process - Case Study of Feedback and Control Using the FEAST Model Lawrence G Votta and Mary L Zajac .....	25

### Proposal

The FEAST Project - Draft Generic Funding Proposal Manny Lehman .....	28
--	----

## General Information

- The workshop will be held on the 28 February and 1st March in the Department of Computing, Imperial College, 180 Queen's Gate, London, SW7 2PH.
- The nearest underground stations to the Huxley Building are South Kensington and Gloucester Rd, both on the District, Circle and Picadilly Lines.
- Sessions will be in room 418 on the 4th floor.
- Tea, coffee breaks and lunches in room 433 on the same floor.
- Workshop dinner on Tuesday night will be at 6 30 for 7 00 at 170 Queen's Gate, access from the campus or from Queen's Gate.
- Registration fee of £85, cash, UK cheque or traveller's cheque, payable to Imperial College.
- The workshop sessions will begin at 9 45 on the first day and 9 30 on the second in room 418 of the Huxley Building. Coffee will be available for one half hour before that time in room 433.
- Unless otherwise notified, hotel accomodation for those requesting it has been reserved at the Rembrandt Hotel, 11 Thurloe Place, SW7 2RS, tel. 0171 589 8100. Room charge £70 per night single.
- The nearest underground station to the hotel is South Kensington on the District, Circle and Picadilly Lines.
- Secretarial assistance will be available, in the first instance, from room 554, front corridor, 5th floor, (0171 59) 48213
- If that office is closed, assistance can be obtained from the General Office, room 437 on the 4th floor, (0171 59) 48298.
- Incoming telephone calls direct to +44 (0) 171 594 8214
- Departmental fax number is +44 (0) 171 581 8024
- email messages marked for your attention to [sred@doc.ic.ac.uk](mailto:sred@doc.ic.ac.uk).
- Outgoing email can be sent from the machine in 554 or from mml's machine in room 556.

## Program

### Tuesday, 28 February 1995

09 15 - 09 45	Registration and Coffee		433
	Morning Chair	Vic Stenning	
09 45 - 10 00	Opening remarks	Manny Lehman	418
10 00 - 11 00	Control and Feedback in the Software Process	Dewayne Perry	418
11 00 - 11 30	Coffee		433
11 30 - 13 00	Examples and other thoughts	Barry Boehm	418
13 00 - 14 00	Lunch		433
	Afternoon Chair	Mark Dowson	
14 00 - 15 30	More examples	Larry Votta	418
15 30 - 16 00	Tea		433
16 00 - 17 30	Still more examples	Berc Rustem (for TAK) Manny Lehman Bashar Nuseibah (for SR)	418
18 30 - 19 00	Cocktails		
19 00 - 21 30	Dinner		170 Queen's Gate

### Wednesday, 1 March 1995

09 00 - 09 30	Coffee		433
	Morning Chair	Manny Lehman	
09 30 - 11 00	Conceptual foundations of FEAST	Wlad Turski	418
11 00 - 11 30	Coffee		433
11 30 - 13 00	Positions	Carolyn Story Keith Bennett Dave Homan	418
13 00 - 14 00	Lunch		433
	Afternoon Chair	Bob Bishop	
14 00 - 15 30	Lessons learned (from the examples and discussion)	Bob Balzer	418
15 30 - 16 00	Tea		433
16 00 - 16 30	Summary points	Colin Tully	418
16 30 - 17 00	Commitments and Funding Proposal		418

**NOTE: The initial sequence of "example" presentations will be provisionally fixed just before the workshop and will be subject to change as the work of the workshop proceeds.**

As of the 21 Feb. the presenters, who will in some instances be discussing examples prepared by others, will include:

Barry Boehm, Bashar Nuseibeh (for Suzanne Robertson), Berc Rustem (for Tarek Abdul-Hamid), Carrie Story, Dave Homan, Keith Bennett, Manny Lehman, Reider Conradi (for Terje Totland and himself)

## Attendees

Dr. Bob Balzer  
Information Sciences Inst  
4676 Admiralty Way, Suite 1001  
Marina del Rey, CA 90292-6695  
tel+1 310 822 1511  
balzer@isi.edu

Dr Frank Belz  
TRW Def. Sys. Gr, 1 Space Park  
Redondo Beach, CA 90278  
belz@bigsur.sdd.TRW.COM

Professor Keith Bennett,  
Durham University  
Science Labs, South Road  
Durham, DH1 3LE  
tel 091 374 2632  
keith@uk.ac.dur.easby

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

Barry Boehm,  
CS Dept., University of Southern Cal.,  
LA CA 90089, USA  
tel 213 740 8163  
boehm@cs.usc.edu

Roger Browne,  
Roger Browne Design  
1 Woodston Grove, Solihull,  
West Midlands B91 3XH  
tel 021 709 0994  
fax 021 705 9187

Brian Chatters,  
ICL - STC, West Aveue, Kidsgrove,  
Stoke on Trent, ST7 1TL,  
tel 0782 794732,  
fax 0782 777009  
B.W.Chatters@uk03.wins.icl.co.uk

Professor Reider Conradi  
Norwegian Institute of Technology  
(NTH, Dept. of Computer Science  
N-7034 Trondheim, Norway  
tel 47 73 593444  
fax: 47 73 594466  
conradi@idt.unit.no

Mark Dowson  
Marleston Software Tech. Inc.,  
525K East Market Street 303  
Leesburg, VA 22075,  
USA tel +1 703338 3951  
fax +1 703 771 8413  
dowson@marlstone.com

Dr. José Fiadeiro,  
Dept of Informatics  
Faculty of Sciences  
University of Lisbon  
Campo Grande, 1700 Lisboa

Portugal  
tel/fax 351 1 7577831  
llf@di.fc.ul.pt

Professor Peter Henderson  
Dept. of Computer Science  
Southampton University  
University Road  
Highfield, SO17 1BJ  
Southampton University  
tel01703 595000  
P.Henderson@uk.ac.soton.ecs

Dave Homan  
Northern Telecom Europe Ltd.  
Oakleigh Road South  
London, N11 1HB  
tel +44 (0)81 945 3097  
fax +44 (0)81 945 3960  
homan@annsgy41.northern.co.uk

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

Professor Martin Loomes  
Hatfield University,  
PO Box 109, College Lane, Hatfield,  
Herts, AL10 9AB  
M.J.Loomes@herts.ac.uk

Professor Tom Maibaum  
Imperial College  
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 Sinisa Marin  
Swiss Bank, London  
sm16@doc.ic.ac.uk

Dr. Bashar Nuseibeh  
Imperial College  
180 Queens Gate, London, SW7 2BZ  
tel +44 (0)71 589 5111 ext 58286  
fax +44 (0)71 581 8024  
ban@doc.ic.ac.uk

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 +1 908 582 7550  
dep@research.att.com

Dr. Berc Rustem  
Imperial College  
180 Queens Gate,  
London, SW7 1BZ  
tel+44 (0) 71 594 8345

fax +44(0) 71 5890 8024  
br@doc.ic.ac.uk

Peter Salhofer,  
Graz, U. of Tech., Software Tech.  
Muenzgrabenstr. 11,  
A-8010 Graz, Austria  
tel 43 316 84 17 56 - 31,  
fax 43 316 84 17 56 6  
salhofer@ist.tu-graz.ac.at

Ahti Salo  
Nokia, P.O.Box 45, (Heikkiläntie7)  
FIN-00211, Helsinki, Finland  
tel +358 0 437 6594  
fax 358 0 455 2091

Warren Smith  
Caledonian University  
Cowcaddens Rd, Glasgow, G4 0BA  
tel 041 331 3280  
fax 041 331 3277  
wsm@gcal.ac.uk

Dr Bob Snowdon,  
ICL, STC., West Avenue  
Kidsgrove.,Stoke on Trent, ST7 1TL  
tel +44 (0) 782 771000  
fax +44 (0) 782 776667  
ras@kid01pml.icl.co.uk

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

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

Colin Tully,  
Colin Tully Associates  
6 Meadow Hill Road  
Tunbridge Wells, Kent. TN1 1SQ  
tel and fax 44 892 539125  
C.Tully@herts.ac.uk

Professor Wlad M Turcki,  
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

Dr Larry Votta,  
AT&T Bell Laboratories  
Napierville, IL  
votta@research.att.com

Dr Aidan Ward  
Antelope Consultants  
0181-693-5463  
100446.24@compuserve.com

**The following are unable to attend but maintain their interest in FEAST**

Dr Tarek K Abdel-Hamid,  
Administrative Sciences,  
Naval Postgraduate School,  
Monterey, CA 93943  
tel +1 408 656 2686,  
fax +1 408 656 3407  
3991P@NAVPGS.EARN

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

Jarek Bochinski,  
AT&T, Telfa, Poland,  
Research & Dev.,  
Software Management  
intgp1!jbochin  
tel +48 523 0001  
jbochin@intgp1.att.com

Dr Bill Curtis,  
TeraQuest Metrics & SEI  
3644 Ranch Creek  
Austin, Texas 78730-3701, USA  
tel 1 512 346 2435 :  
fax 1 512 346 4677  
curtis@acm.org

Larry Druffel,  
Software Engineering Inst. (SEI)  
Carnegie Mellon U.,  
Pittsburgh, PA 15213  
druffel@sei.cmu.edu

Professor Carlo Ghezzi,  
Dipartimento di Elettronica e  
Informazione, Politecnico di Milano  
Piazza L. da Vinci, 32  
20133 Milano, Italia  
tel 010 39 2 239 93411  
ghezzi@elet.polimi.it

Andy Greener,  
1 Captains Gorse  
Upper Basildon, Reading  
Berks RG8 8SZ  
tel/fax 0491 6719644  
andy@gid.co.uk

Volkmar Haase,  
Graz, U. of Tech., Software Tech.  
Muenzgrabenstr. 11,  
A-8010 Graz, Austria  
tel 43 316 84 17 56 - 31,  
fax 43 316 84 17 56 6

Watts Humphrey,  
34 Rockwell Lane  
Sarasota, Fl 34242, USA  
Tel: 813 349 2643  
watts@sei.cmu.edu

Katsuro Inoue  
Osaka University, Osaka, Japan  
inoue@ics.es.osaka-u.ac

Simon Kaplan,  
Department of Comp. Sc.  
Human-Computer Interaction  
Univ. Illinois at Urbana-Champaign,  
1304 W. Springfield Ave,  
Urbana Illinois 6180 Laboratory  
tel 217 244 0392,  
fax 217 333 3501  
smk@uiuc.edu

Dr. Marc Kellner,  
Software Engineering Institute  
Carnegie Mellon University  
Pittsburgh, PA 15213, USA  
tel +1 412 268 7721  
fax +1 412 268 5758  
mik@sei.cmu.edu

Professor 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

Mrs Z Levy,  
48 Haparsa St,  
Tel-Aviv 69085, Israel  
tel + 972 3 6471 373  
fax + 972 3 566 1667

Dr. Pertti Lounamaa,  
Nokia, P.O.Box 45, (Heikkiläntie7)  
FIN-00211, Helsinki, Finland  
tel +358 0 437 6594  
fax 358 0 455 2091  
lounamaa@research.nokia.fi

Professor 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

Jean Pierre Moularde,  
SEMA (France), 16, rue Barbès  
F-92126 Montrouge Cedex,  
France  
tel +33-1-40 92 43 16  
fax +33-1-46 56 96 53  
moularde@metra.fr

Dr Shari Lawrence Pfleeger,  
2 Riverside Close  
Kingston Upon Thames,  
KT1 2JF  
tel (071) 477 8426  
shari@csr.city.ac.uk

Bill Riddle,  
sda, rmise, 1113 Spruce Street  
Boulder, Colorado, 80302  
tel +1 303 499 4782  
fax +1 303 938 5005  
icspl@sda.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 Bob Rockwell,  
Softlab GmbH.,  
120, D-81677 Munich,  
tel +49 89 93 00 1475,  
fax +49 89 938 281  
roc@softlab.de

Graham Samuel,  
39 Laurier Road, London, NW5 1SH  
tel 071 485 7916  
graham@livfoss.demon.co.uk

Colston Sanger  
69 Kings Road  
Haslemere, Surrey GU27 2QG  
colston@gid.co.uk

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

Ian Sommerville,  
Lancaster University,  
Lancaster LA1 4YR  
is@comp.lancs.ac.uk

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

## Summary of Post-Workshop II Discussion (Manny, Wlad, Dewayne) 26 October 1994

### Preliminary Remark:

These notes are minutes of a fast flowing discussion involving the three of us. The record kept at the time has been mildly edited. They must be treated as an attempt to capture the essence of the discussion and not as a polished, considered and up to the minute text. We plan a fuller paper to reflect these musing and further thought. Meanwhile in the interest of clarity the notes first published in the preprints of the Second FEAST workshop are repeated

### Start of Manifesto

#### The Feast Project

This *Manifesto* lays the technical foundations for what we believe could be an interesting, worthwhile, multi-year funded project. Following from the results of the first workshop, it provides the basis from which, among other things, a project definition and workplan should be developed for a realistic and manageable startup project. The subsequent direction of the project is dependent on the initial results.

#### General Goals:

1. To produce specific recommendations, guidelines, methods and tools for software evolution process improvement.
2. To contribute to a science of software evolution.

#### Postulates:

The Feast project is limited to consideration of process systems that govern the evolution of software systems and that satisfy the following postulates:

1. They have rich networks of feedback.
2. Some of the feedbacks stabilize characteristics in these systems.
3. Some feedbacks are controllable.

[Note added by mml. We believe that there exist real software processes that satisfy these postulates. It will be an objective of the FEAST project to identify and model such processes or to demonstrate that they do not (cannot?) hold in practice. It will also be of interest to explore how they might be changed or supplemented to increase their applicability and/or value. **End of note**]

#### Process Unit Model:

Consider a process element (pe) which applies resources to transform inputs into outputs. If one of the destinations of the output is a controller (c), where output is fed back into the process element, we obtain a general controlled feedback loop (as in the diagram below). We term this total configuration a process unit (pu). Process elements can contain process units. As a first approach we assume that control units (Cs) are themselves not, and do not contain, pus, ie are not, themselves, controlled by feedback.

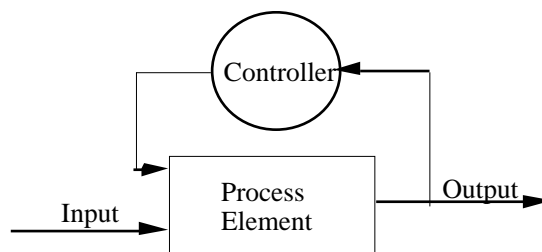


Figure 1: The basic process element

#### Hypothesis:

A process that satisfies the postulates above can be usefully decomposed into a manageable number of process units.

#### End of Manifesto

#### Some observations on the model:

1. In the *elementary building block* as previously described the arrows are, as yet, undefined. We may wish or need to attach precise semantics to some arrow in the future and recognise that this will be difficult as each arrow will have to be treated individually. We start with the simplest models and shall add embellishments if and when they are shown to be desirable and helpful.
2. Nor do the boxes have much semantics until the functions are defined. These functions will not usually be represented by a formula, but more likely by scenarios for the controller in response to the observed state. Depending on the observations, the controller can do various things.
3. The controller is not a person but a function. A person may perform several functions simultaneously and may even combine several functions simultaneously. The basic control elements must be kept as simple as possible.
4. The introduction, sooner or later, of either state or time information cannot be avoided. Where ever time is involved, there will also be delays. The magnitude of the latter is a major determinant of system behaviour. In control theory delays lead to rates of change and to differential equations, etc. Consideration of states requires the *next state* operator. This takes us into Turing machines.

## Discussion about observed kinds of control:

1 Consider a case where control inhibits output – for example in a system release even when the "gatekeeper" at the output of a process unit says OK, the output can still be inhibited by external control.

*Claim:* a feedback controller cannot inhibit output it can merely change the state or behaviour of internal mechanisms. Thus there are at least two controlled release process units each with its own gatekeeper. The first achieves release from development, the second release to the user.

2) Analysis and modelling of project Management will be a challenge. It provides continuing supervision and has the ability to reallocate resources, start, terminate, or change processes. It acts like a combination of controllers and gatekeepers. Some outputs are generated especially for management (for example, progress reports) being produced and sampled over time.

*Claim:* need to separate (but not ignore) organisational structure and people from the process units model. Some aspects of supervision are part of work, some part of control. The model has the effect of blurring the distinction between management and workers (management usually objects to this blurring). Redistribution of resources is control and must be included in the study.

## Discussion on Technology:

Balzer's third hypothesis: The general expectation (by funding authorities?) is that drastic improvement will only be obtained through the development, transfer and application of (advanced) technology. In some sense this is orthogonal to the basic tenets of the project since the conjecture says that it is likely that advancing technology can produce major impact only if accompanied by review and possible modification of the feedback structure, the latter itself may in fact produce major benefit without being accompanied by advances in technology. A fundamental tenet and thesis of the project is that process cannot and should not be studied independently of either organization or technology. It is also relevant to Balzer's hypothesis to comment that, in the past, technology advance and its funding has tended to focus on software creation and, in particular, on improving the execution of direct software development activities as identified, for example, in the waterfall model. What is now needed is a new technology for evolving systems, since many installed and operational systems cannot easily or reliably be reproduced, shut down or replaced, they must be dynamically adapted and extended.

## Discussion of the Functional Unit Model - See also Wlad TurSKI's paper: *A Somewhat Overdue Introduction*

Basic decision: The model was postulated on the assumptions that an elementary building block, core model, must be 1. as simple as possible, 2. based on the minimum of assumptions, 3. amenable to rigorous representation and manipulation. Thus we deemed it correct to retain the model as defined in the *manifesto*, but augment it with discussion and examples. In particular, we need to distinguish between control and work (where work includes gatekeeping) and to separate the modelling of disciplined activity from the organizational and personal aspects.

We recognised that the question of *scope*, particularly of control, is an interesting research problem? Information may ripple up or go via direct communication to various levels in the hierarchy. Both communication and (feedback) control must be modelled. How this is implemented when both may span many levels of activity or control requires investigation and may depend on the goals of the modelling. One example of this far reaching kind of control is the allocation and reallocation of resources.

Note that there are information paths that feed forward in the process and are not feedback. Others feed backward in the activity path (but forward in time - as does all feedback) and produce enlightenment but not control that is directly attributable to the information fed back. We focus the initial FEAST studies on *feedback control*, its identification, representation, management and optimisation. We recognise that control which is not feedback control is also exercised in the process. Management, for example, may send a control signal to a design group saying (for whatever reason) "Use object-oriented implementation". Or there might be resource-allocation that has that has to do with corporate policy and nothing with feedback.

There is a similarity between the alternative meanings of each of *feedback* and *control* and that of initial settings and structures. Feedback control seeks to set a course that is optimal with current perceptions/experience. Based, via feedback, on past experience or results, it may modify current means or goals or establish an entirely new set.

It is in this context that improvement must be considered. Given the stability properties of systems with negative feedback control our global model and its constituents must be a vehicle for identifying feedback paths and controllers in the software systems process and their manipulations both locally and in concert with other controllers for global improvement. Global optimization depends on the overall goals and the global system.

## Project Descriptor - title

Our primary consideration is on improving the process of evolving software systems – that is, improving the process whereby software - really computer applications - evolves or is evolved. The model - down to its elementary building block - is a means to this goal. Wlad suggested a phrase with "responsiveness" in the Descriptor – for example, improving the process responsiveness to software evolution. The advantage of something like this is that it leads immediately to feedback, though it may require careful explanation. One must consider software engineering processes and software systems. An appropriate descriptor might be "Improving process responsiveness: technological answers to the challenge of evolving software systems" or in short "The responsive process: technology for evolving software systems".

## A Somewhat Overdue Introduction

Wlad M Turcki  
Institute of Informatics  
University of Warsaw  
wmt@pieniny.mimuw.edu.pl

In June 1994, at the first FEAST workshop, there was a consensus that feedback phenomena play an important, although yet unrecognized, role in the software process.

It is, perhaps, not presumptuous to assume that we study the software process so that we may improve it; provided that we can agree what exactly is meant by improvement. Fortunately, whatever the precise meaning of this notion, any improvement is likely to be due to better structure, better control or both. As feedback phenomena are probably more closely related to control than to structure, we are justified in concentrating (at least initially) on the conjunction of these two notions, i.e. on feedback control in the software process.

Before going any further it is worthwhile to observe that both these terms as used in everyday conversations are richly homonymic.

Here is what Webster's New Collegiate Dictionary says about *feedback*:

**feedback:** **1** the return to the input of a part of the output of a machine, system, or process (as for producing changes in an electronic circuit that improve performance or in an automatic control device that provides self-corrective action). **2a:** the partial reversion of the effects of a process to its source or to a preceding stage. **b:** return to a point of origin of evaluative or corrective information about an action or process <the student ~ was solicited to help revise the curriculum> <we welcome ... from our readers - brickbats as well as bouquets - Johns Hopkins Mag.>; also: the information so transmitted"

The mere return of information, even if it is *evaluative or corrective* does not affect *an action or process*. To have an effect, this information must be somehow used, i.e. it must produce a change in something. The machinery by means of which the change is produced must be in place in *a point of origin* to accept the information and act on it. We call this machinery *the control element*, and the locus of the change itself (the "something" in which the change is made) *the process unit*. In this way we put a concrete, process-related "body" on the idea captured in the meaning 1 above.

During the Workshop many people mentioned "learning" in the context of feedback considerations. This being the case, the interpretation that seems most natural is that the feedback is the information returned to a *person* placed at the point of origin, is absorbed by this person and—via an act of human learning—modifies this person's future behaviour, e.g. the way this person manages whatever happens to be his/her activity domain.

This interpretation is OK for the *sociology* of the SE process. It can be a part of SE managers' education: "thou shalt pay (more) attention to the feedback you are getting"; or even more aggressively: "thou shalt seek more feedback about the actions you manage". It can be elaborated by supplying a list of sources from which the feedback is to be considered/sought (clients being very prominent on such a list!), the list may be categorized into "important", "vital", "irrelevant" etc. Suitable case studies may be conducted, yielding instances of success stories (what benefits accrue when one heeds the feedback message) and of horror/punishment (what disasters follow when the feedback information is neglected). This—no doubt—can (and will) be a useful part of education/training of SE managerial types.

But, of course, this interpretation of feedback cannot be applied to a technological view of the SE process. (Assuming we do not claim that making the "human learning process" technological is a part of the FEAST charter.) The point is that with the latter interpretation the "machinery" is all in the human brain. As yet, all attempts to emulate the nontrivial aspects of human brain activity produced very weak results; it would be audacious to propose that FEAST will succeed in 2 – 3 years where scores of researchers failed over decades.

Moreover, even if we accept that feedback is merely the "fuel for learning", we still face a dilemma: Either we explain what are the appropriate reactions to be learned, or we leave that to intuition (creativity).

In the former case, i.e. when it is ultimately known what are the recommended, beneficial, profitable reactions to a particular combination of feedback signals received, we do have explicit control machinery ("when you get too many error reports coming from the customers strengthen the quality control", "when you are too late with the delivery, cut down on the most time-consuming activity" etc.).

In the latter case (invoking intuition) such machinery is not readily apparent, but it is hard to see what advice can be given to a manager as the (necessary) second part of the admonition to pay more attention to the feedback. A rational manager will almost certainly ask: What am I supposed to do when I collect all this information fed back to me? How can I act on it? Unless we are prepared to answer: use your head (or by a similarly profound platitude), we are inextricably bound to construct control machinery.

Even if we are willing to entertain more exotic solutions, like neural networks or another AI structure learning from examples, there is no escaping some sort of rules that would associate particular feedback signals with appropriate responses. In a sense it is immaterial whether they are given explicitly, or as case stories ("A manager flooded with bug reports from the test team decided to beef-up the inspection sessions by allocating them more resources: time and qualified people. This yielded an overall improvement in process efficiency.")

Thus, we are inclined to uphold our original model, perhaps with an explanation that it does not cover (nor intends to) *all instances of feedback*. In this way we steer clear of the non-control kind of feedback and we are not leaving out much that would be of

technological consequence. Whatever other kinds of feedback (covered by the meaning **2a** and **2b** from the dictionary definition) are considered, if they are to be used for improving the software process they must be turned into an explicit control mechanism. Thus it is only as long as one is prepared to be non-technological that one is permitted to entertain notions of "feedback in general". *FEAST should firmly aim at being a technological endeavour.*

Consider now the other central notion: control.

The verb *control* has two principal (families of) meanings, to wit, once more quoting from *Webster's New Collegiate Dictionary*

**control: 1** to check, test, or verify by evidence or experiments **2a:** to exercise restraining or directing influence over:  
REGULATE **b:** to have power over: RULE

The distinction between the meaning of the verb carries over to the corresponding noun, if anything, is perhaps even stronger.

These two meanings are commonly mixed up in loose speech, whereas in application to the SE process the activities denoted by "control-1" are quite different from those denoted by "control-2". (The distinction between "control-2a" and "control-2b", although important in many contexts, is less fundamental in our considerations as—usually—one has to have power over something if one is to exercise restraining or directing influence over this something. With some hesitation we may accept that in the context of SE process "control-2a" implies "control-2b".)

The confusion is amplified (or, perhaps, generated) by the fact that a single person (or a single body, perhaps collective) often performs *both* actions, "control-1" and "control-2" with respect to a productive activity. In addition, it may happen that the same body performs "control-2" over several activities, particularly when "control-2b" is meant. Nevertheless, a precondition for any sensible approach to scientific (technological) treatment of the SE process is to *disentangle* the meanings of "control".

From now on we use "check" for "control-1" and "regulate" (or "rule", as the case may be) for "control-2". Furthermore, since ruling without regulatory powers seems a bit of constitutional nonsense, we shall not consider agents that rule but do not regulate anything.

Any SE process in place is a fairly complex structure of *activities* and *people*. To view such a structure with the intent of analysing it and, subsequently, recommending ways of improving this or that (vagueness intended!), we must establish *a method of decomposition*.

That we need to decompose the process should not require justification. That we ought to do so methodically follows from elementary considerations of economy: if we do not have a method to do the decomposition (almost) unthinkingly, the sheer volume of (repetitious!) inventions to be made each time we decompose a process is staggering. Thus, it is a *pragmatic* consideration to have a method. Moreover, the method should be preferably *recursive*, so that we can carry out the decomposition by applying the same rules repeatedly (i.e. we can, and will, reapply them repeatedly to chunks already obtained by application of the rules).

A recursive decomposition method requires that an elementary building block, an atom, be defined; otherwise, it continues *ad infinitum*. The choice of elementary building block is relatively unconstrained. That a bad choice was made one would learn later, perhaps much later, when the method breaks or the decompositions obtained are unsatisfactory for a particular purpose. When no specialized guidance is available, one should probably choose the simplest possible block and stay with it until hard evidence in favour of a better selection (or, at the very least, hard evidence for a significant deficiency of the current selection) is obtained.

There are two natural candidates for the elementary building block: people and activities. Choosing *people* would lead to a decomposition like an organizational chart of a company (for different projects executed by the same company, the "charts" need not be identical, even when the projects are concurrent). These are typical "Who's whose boss" charts, no doubt very useful for some purposes, much less so for ours.

The last statement may require a justification. Our concern is with ways to improve the process of software making, so that the product delivered to the customer is (and remains) optimal within the constraints imposed on the process and its context. Re-allocation of people could be an instrument by which thus defined process optimization is achieved. It is very doubtful, however, that signals about a suboptimality of a process can be easily (or at all!) mapped to an organizational chart. (We seriously question the ability to perform *direct* maps of the form: "Users complain about late deliveries of updates" → "Take J. Smith from Test-and-Ship and put her into Payrolls".) Thus, even if an improvement is ultimately implemented by a staff reshuffle, the ability to execute such a reshuffle is better seen as an action available to a regulator of some activity.

Therefore, in our methodical decomposition *activities* we take elementary building blocks. Of its several consequences, the most important are:

- Individuals in the company need not be uniquely identified with entities in the decomposition. Some persons will be associated with several activities, and— if the decomposition obtained is (erroneously!) interpreted as an "approximation" to the organizational chart—"vicious circles" of dependencies could be seen (J. Smith being her own boss, and, even more sinister, J. Smith being a boss of K. Doe, who in turn is J. Smith's boss). Attempts to "streamline" the decomposition to eliminate such "anomalies" must be resisted: These are not anomalies at all!
- There are two kinds of activities: such that do regulate other activities and such that do not. It must be repeated here that checking is distinct from regulating, even if in the loose speech both are rendered by "controlling". On the other hand, in a disciplined work environment all productive actions should be checked. Thus we view the checking action as inseparable from the productive action and do not represent them as two distinct entities in the decomposition.

*Comment.* There is a weak sense in which checking regulates a productive action: Often, when the check fails, the checked action, or a part of it, is repeated. As long, however, as this is just an execution of an established routine "do action until check-successful" there is no point in separating the two.

An entirely different situation arises when, being dissatisfied with the performance of the pair, we decide to *change* the routine, e.g. by installing a new kind of check. This is a regulatory action and should be viewed as a prerogative of the *regulator* who rules over the pair. *End of Comment.*

We have repeatedly used the phrase "productive action". Taken literally, it embraces all activities: after all, who would willingly tolerate "unproductive" actions! Therefore, it must be put on record that the phrase is used in a restricted sense: it denotes actions by which a part of the product is produced or some other output is generated to be consumed by another productive action. In this sense the actions of regulators (although often very productive in the broader sense) are not productive in the adopted sense: Their output merely modifies a way in which productive actions and/or ensuing checks are performed.

Thus we focus our attention on productive actions (usually, if not always) coupled with corresponding checks, and on regulating actions. From now on, for simplicity, assume that "productive action" includes the corresponding check (if any).

It is not necessarily the case that each productive action has a separate regulator. Several productive actions (for instance, code-writing actions) may be composed sequentially or in parallel into a larger unit with a common regulator. In particular, it may happen that a regulator ruling over a composite unit can change the composition of the unit. In its directing capacity, a regulator may reshuffle the productive actions within the unit, suspend and start some productive actions in the unit. (Whether we want to consider creation of a fresh action within a unit as a legitimate act of the unit's regulator is a point that needs to be further discussed.) Naturally, changing the parameters of productive actions in a unit always falls into the regulator's scope; this includes changing the numeric values of checking procedures (eg. how many caught and corrected bugs are allowed in a piece of code before it must be rewritten).

As with many SE process analytic tools, the very act of decomposing a process in a particular fashion may yield a substantial dividend, quite apart from any benefits that may accrue from applying subsequent steps. A very important kind of dividend would be the listing of regulators' admissible actions. Quite likely one will discover how *badly* defined are the regulators' prerogatives, how arbitrarily they are distributed between various regulators, and how little justification there is for allowing some regulators to do things that are just as groundlessly denied others. If this hunch proves correct, a very concrete improvement to many SE processes would be instantly available: the unification of regulators scope under similar (or even more so under identical) stimuli. Translating into "shop-floor" terms, one would advise giving similar powers to people who control similar activities. This piece of advice is of course trivial; the difference is that with "our" decomposition in hand we can flesh out the "similar" parts of the advice.

The whole idea of applying *feedback control* (after the two extensive linguistic analyses, we are entitled to use this term!) to the SE process rests on the assumption of there being a stream of similar tasks coming to productive units. Indeed, if all they do are one-of-a-kind jobs, there does not seem to be a lot to be learned from the feedback received, let alone to convert the feedback signals into rational actions of the regulators. The similarity of tasks does not imply, of course, that if the productive unit's action is writing a code to specifications given, the specifications must be always for the same function. On the other hand, feedback signals collected on a piece of code written for numeric processing functions need not be very pertinent if the next task is writing the code for a piece of a telecommunication switch.

Now, another dimension of the distinction between checking and regulating (two kinds of control, see above!) becomes apparent. Consider a stream of tasks supplied to a unit, call them  $t_1, t_2, \dots, t_n, \dots$

When task  $t_i$  is performed it is checked in the currently routine manner in the unit. The only way the regulator may influence the execution of  $t_i$  is by observing an *abnormal delay* in the unit's output or abnormal consumption of other resources. If the unit executed  $t_i$  roughly as expected, the "productive" output goes where it was meant to go and slips from the unit regulator's scope. Even if the regulator analyses this output and detects some faults in it (obviously, faults that were not considered as such by the unit's internal checks) it is too late to do anything: the cat is out of the bag! Naturally, the regulator will now react to the discovered sloppiness of the productive unit, eg. by tightening the checks, or by rearranging the productive activities within it, in a way designed to eliminate or reduce the faults. But this will have no effect before the execution of tasks  $t_{i+1}, t_{i+2}, \dots$

Yet more complicated picture presents itself when we have a hierarchy of units, ie. units embedded in units. The assumption about the stream of tasks continues to hold; now, however, the tasks have a hierarchy of their own: The big tasks for the outer unit decompose into smaller tasks for the inner units. The regulator of the outer unit sees its outputs (eg. complete systems, or versions of it). The intervals between the outer regulator's observations are likely to be much larger than those for the regulators of inner units. There is an additional degree of freedom that regulators ruling over units with internal units possess, viz. the ability to tune (or activate, or suspend) the internal units' regulators. *The little flea ...* Note however, that for a hierarchical control structure the ability of the higher level controls to act occurs only after many cycles of lower level units activity. This means that when the process decomposed into such a hierarchical structure is in its early stages, the higher levels of control simply did not start their act yet. A full realization of multilevel control's capabilities comes with the system's maturity. In other words a well-regulated process is a process that reached its steady-state!

## A Hierarchy of Feedback Loops

Tarek K Abdel-Hamid  
Naval Post Graduate School

We need to think in terms of a hierarchy of feedback loops. Here is an example:

A high level view of one key structure in the software management domain is shown in Figure 1.

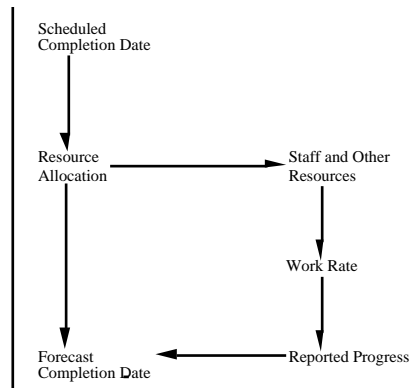


Figure 1 Resource allocation feedback loop

The figure characterizes the planning and control cycle of activities that is repeatedly traversed as a project progresses through the software development life cycle (SDLC). Resource/schedule planning on a software project is never a one time activity conducted just at the beginning of a project. More often than not, managers find that initial plans do not constitute the best course of action to actually take during the life cycle; initial plans merely show what was thought of as best when the plan was made. Since actual events on a project almost always differ from the assumed events that the plans were designed to meet, project managers must react continuously to real world events that actually occur, and not to those that might have occurred had the real world been kind enough to conform to the initial planning assumptions.

To plan and control a project in any domain, two information inputs are absolutely vital. The first is the planning information which provide the standards used for assessing the significance of what is happening. The second is accurate and timely status information. The reason why controlling software projects continues to be a difficult and frustrating process is that both these information inputs are often unreliable. For, on the one hand, it has proven to be quite difficult to accurately estimate development costs and schedules and, on the other, to measure a project's progress rate.

This raises the following interesting question: How do software managers cope with the unreliability of these key information inputs, and what are the implications (in terms of project performance) of whatever compensatory strategies they opt to use?

Because of cognitive limitations, managers often make their judgements on the basis of simple rules of thumb/heuristics. To study the cognitive heuristics that managers employ, each of the "corner" decisions of Figure 1 can be further analyzed using building block structures like the one in Figure 2.

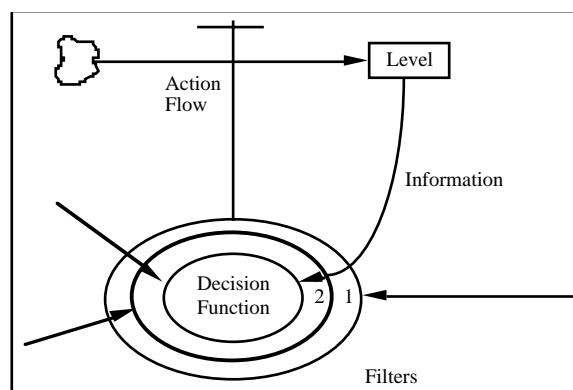


Figure 2 Detailed decision function feedback structure

One can see in the figure the standard feedback representation: decision function (e.g., staff resource allocation) → action flow (hiring) → information (available staff) → decision function (staff resource allocation).

The figure also suggests that there are additional information flows influencing the decision function (e.g., budget constraints). The two concentric circles surrounding the decision function represent organizational and cognitive filters which select or limit these information flows. The outer filter (1) represents the psychological environment provided by the organization within which managers make their judgements and decisions. For example, it represents (among other things) the influence of operating goals, rewards and incentives on information flows. The inside filter (2), represents people's cognitive limits/biases.

# The Role of Feedback in Knowledge Management

K. H. Bennett  
University of Durham

## Introduction

We are undertaking a 30 month project with ICI with the objective "to investigate the possibilities of knowledge management to aid co-operation between scientists in different ICI businesses in the area of surfactants R + T".

This might appear some way from my main research work, in software maintenance and evolution. I would certainly like to study feedback and control in maintenance processes; but as we progress with the ICI project (we are 9 months in), we are starting to realise that feedback provides an important control function- especially in terms of confidentiality and security - affecting much of what scientists do and how they relate to each other. These ideas are quite difficult to tie down, at least in a concrete form such as a "FEAST model", but that is the direction we are studying, and we want to get our new RA to work on this. So I cannot yet offer concrete models in what essentially is a CSCW problem. I'll report on our direction.

Our aim is to find a modelling method to help ICI with the knowledge management process; the project is all about using models to determine corporate strategy. This we see as preferable simply to installing technology such as Email and hoping for the best.

## Problem

We have spent 9 months trying to understand scientists' needs and how these relate to the business. In so much that we have used concrete models so far, we are using the ORCA method for requirements analysis. We have established e.g that not enough return is made from corporate knowledge, and too much corporate knowledge is personal, inaccessible and unaccountable. Underlying this is an obvious fact: scientists are not independent.

We are starting to develop (high level, process type) models of how surfactant scientists interact between businesses; and how different roles are played by different businesses (eg formulation may be in another ICI business). In Chemistry, commercial confidentiality and security provide a strong influence on scientists' interaction. We have found that there are some powerful control mechanisms; it is these we would like to understand more. The name of the project is "Tollbridge", reflecting how central this issue is.

I hope that within the next 6 months we can report specific progress, at least on very simple models.

## **Feedback in the Spiral Model**

Barry Boehm  
University of Southern California

After a short summary of the Spiral Model and the recent WinWin Spiral Model, this talk will present the basic feedback model involved in the Spiral Model. It includes feedback cycles for addressing risky combinations of objectives, constraints, and alternatives (OC&A's); infeasible combinations of OC&A's; and changes introduced in OC&A's.

As with the usual Spiral Model diagram, this feedback model is oversimplified in that it abstracts out the role of multiple concurrent spiral and feedback cycles. The talk will address such concurrency issues, and illustrate them with respect to an example: the use of the WinWin Spiral Model in the ARPA STARS program.

Based on the usage experience above, the talk will then present a proposed elaboration of the FEAST feedback model, involving the development of plans (for both the product and the project), based on assumptions about the real world, and the use of the plans to determine both project execution and expected results. Control is then defined as the revision of plans when the discrepancy between actual and expected results indicates a sufficiently large violation of real-world assumptions has occurred to require a change in plans.

The talk will then present a set of guidelines for developing plans which accommodate the need for changing (evolving) them. These Parnas-type guidelines include anticipating sources of change in real-world assumptions (about project performance, product achievability, or external process drivers); modularizing plans around anticipated sources of change in real world assumptions,; and pro-active monitoring of potentially volatile, high-impact real world assumptions.

## **A View from BNR**

Dave Homan  
Northern Telecom (Europe) Ltd

We have given some thought to the feedback model you propose. It is our view that the proposed model can only represent the conditions that exists at a certain point in time as they relate to an organisation's progress to process and organisational maturity.

On the classic scale of maturity of 1 to 5, I would suggest that the organisation the model describes is at best level 2. This of course depends on proportion of effort and resources spent of the various activities described in the model.

In a stable, mature, organisation much of the time is spent on requirements capture, getting to know who your real customers are (not unnecessarily the same person who is placing the order), verification and the integration of the product into the real world.

The stable organisation has already worked out the theories, procedures, models etc.. it requires and they become second nature to those who use them. Any change to the process is evolutionary not revolutionary. The organisation can concentrate on acquiring product knowledge and really understanding the customer needs and software simply becomes a way of interpreting those needs.

Feedback takes places at many levels and at many places our development lifecycle. We have found that one of the most important feedback's is obtained by customer involvement. Our customers are invited to design, test strategy, test specification and acceptance strategy reviews. In addition all site defects or queries are reviewed jointly with the customer and a root cause analysis report written for each defect. On completion of a project, a project debrief is held and from the results of the debrief processes may be changed.

# An Example of Apparently Overlapping Control Loops

Manny Lehman  
Imperial College of Science, Technology and Medicine

Consider a process that transforms a software requirements statement into a validated system ready for shipping. If the basic unit defined for FEAST is valid, this process must be representable by such a structure. We know that such a process may be broken down into a number of sub-processes each of which may also be represented as a basic process unit. However for the purposes of this example it is only necessary to identify the first two and last two such units. For development of the issue to be discussed, and not because the functions chosen are considered to be especially significant, four steps in the Waterfall model are chosen. Any other set could have been chosen equally well. This brief discussion will not need to address individual sub-unit control mechanisms. It will simply reason about the nature of the control of the process as depicted, in the first instance without control loops, in figure 1.

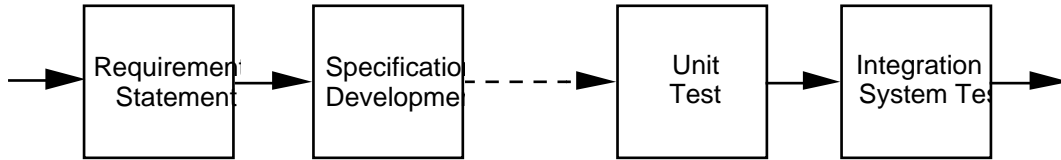


Figure 1: The Forward Path Process

Suppose that the output from the *Unit Test* sub-process includes information relating to the success of the process to that point and, in particular, details of the number and nature of failures or detected faults. Suppose further more that the data shows that a very high proportion of the faults requiring rework were due to misinterpretation by the development team of the *requirements* as stated. The *controller* receiving this information may determine that the process encompassing the activities from *Requirements Statement* to *Unit Test* must be modified by introduction of *formal* requirements representation and the use of modified *inspection* procedures in one or more of the constituent sub-processes. In terms of the basic FEAST feedback control model the process may be represented as in figure 2 where the controller must have authority over all sub-processes in the activity sequence of figure 1.

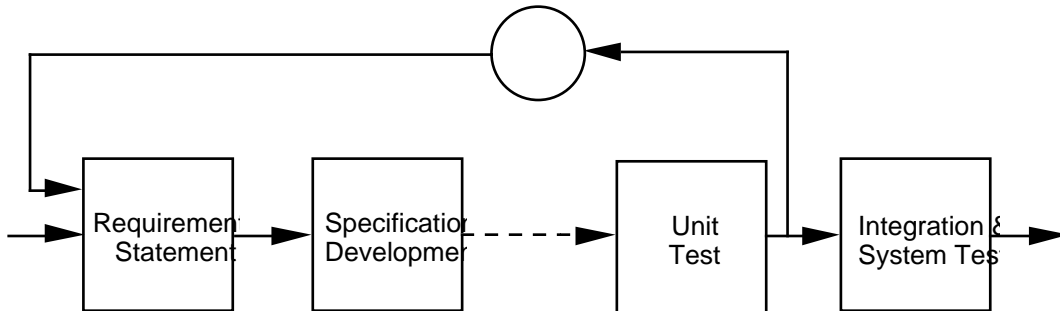


Figure 2: First Control Loop

Similarly consider a situation where *Integration and System Test* finds the functional behaviour as determined by the requirements statement acceptable but discover faults in the specification. To avoid re-encounter with similar process weaknesses in the future a controller with a control range as in figure 3 issues directives for changes to the sub-processes of that range.

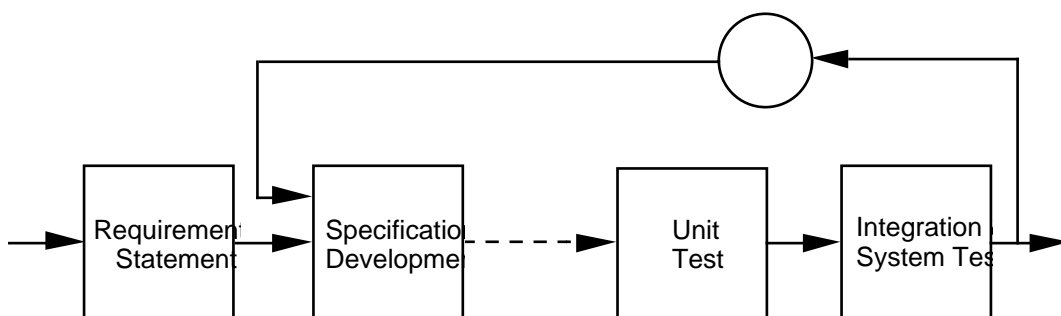


Figure 3: The Second Control Loop

These two control loops clearly overlap. Their superposition is unacceptable for at least three reasons:

1. The two proposed sets of modifications will certainly interact and must be developed jointly
2. One should never have overlapping jurisdictions since these must interfere with one another
3. Extending this simple case over the total process will lead to innumerable overlaps that cannot possibly be managed or analysed

We are thus forced to recognise that the only appropriate representation of the scenario described is that of figure 4. In this a single feedback loop and mechanism controls the entire set of sub-processes, now representable by the basic FEAST model. Whether this is appropriate or not must, of course, be determined in the context of a more extensive, and ultimately global, model.

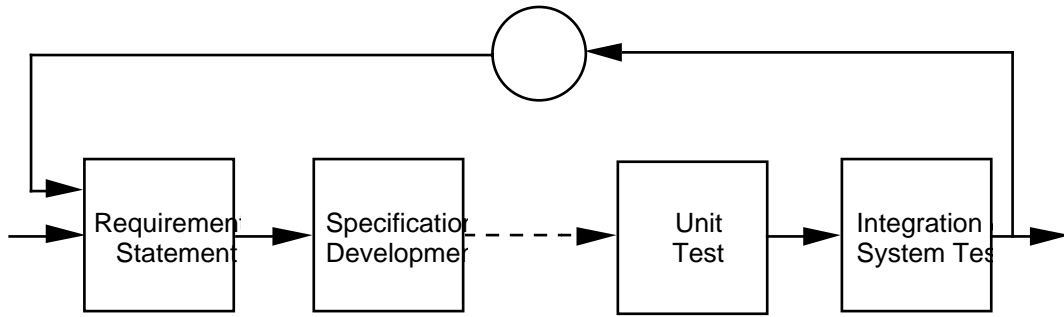


Figure 4: The Full Model

The reasoning identifying the model needs and leading to the representation of figure 4 extends to the global process. It leads to an overall feedback structure that is essentially hierarchical. This view is entirely consistent with a hierarchical management structure. Whether, in the course of the FEAST investigations, we shall encounter theoretical or practical situations which cannot be represented in this way is an open question. There is clearly no reason to abandon it unless and until a situation in which it is untenable is identified.

One final word is in order. Like so many of the others observations made and conclusions being reached in the FEAST process, the above conclusions are not new. Figure 5 reproduces a chart that has been used for more than a decade to argue the case for more attention to the front end of the development process. It makes precisely the same point.

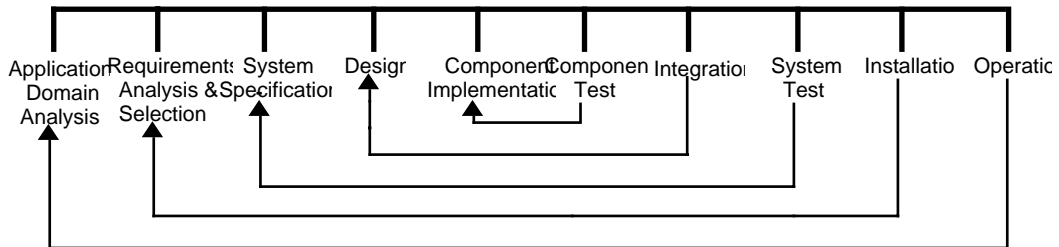


Figure 5 The Hierarchical Nature of Fault Insertion and Discovery

# MACROSCOPIC FEEDBACK SYSTEMS

Nazim H. Madhavji  
McGill University, School of Computer Science

## 1 Introduction

My starting point for discussion is that feedback is important, assuming that feedback was driven by specific goals. Thus, we make an assumption that we have feedback data, or information, which could be analysed to help us make decisions in whatever our business is (e.g., requirements engineering, controlling multiple software projects, customer satisfaction, etc.).

Traditionally, work on feedback has focused on "microscopic" intra-process and inter-process issues. By the former (microscopic but intra-process), I mean, that feedback which is focused on finegrain issues such as specific defect types, complexity of modules, elapsed time, etc., and is obtained from within a given process and is analysed and used for making improvements in the same process. Many software organisations do this on a regular basis (e.g., NASA/SEL to be specific, and CMM level 4 organisations to be general). There is strong evidence that this type of feedback pays off if carried out appropriately.

By the latter (microscopic but inter-process), I mean, that feedback which is focused on the described finegrain issues but is obtained generally during postmortem examination of one project and is analysed and used for making decisions about a subsequent project. Only some organisations do this in a methodical manner (e.g., NASA/SEL). This type of feedback generally needs a cross-project database and related technology, often called an "experience factory". We have yet to see wide-spread and solid evidence of the costs and benefits of this type of feedback, although intuitively it all sounds beneficial. Both the described types of feedback are a central aspect of a learning organisation.

However, a major challenge, according to my experience, is to be able to "convince" an organisation that such feedback systems are beneficial for "them". Unless "they" have experienced the benefits in measurable terms, it is often impossible to move these organisations.

Even worse situation than this is that of an organisation which accepts, in principle, that feedback systems are beneficial in general but that "its specific situation" has no room for "process disturbance". That is, unless the injection of the feedback system has "zero learning curve", "zero cost" and "immediate and guaranteed pay-off", the organisation will not be willing to adopt such a program.

From this, an important lesson learnt is that it is not enough to find theoretical solutions of how to inject a feedback system into an organisation, but that the solution must address the fact that many target organisations have no room whatsoever for "process disturbance". This condition calls for innovative solutions on what type of feedback and how to inject this "without pain" into the target organisation. I have yet to see such a solution. The irony of the situation is that it is those organisations that need help the most that cannot accept any help.

However, despite the described benefits of intra-process and inter-process microscopic feedback, they do not handle well "in-the-large" issues in the software process. For example, how can one visualise the quality of the "flow" of a process? How can one determine the "commonality" of a set of processes distributed across organisations? How can one know the "congruency" (i.e., fitness) of a process in a given organisation?

The basic limitation of microscopic feedback seems to be that they focus on specific points in the software process, and thereby they do not see the "wood for the trees". In other words, they are highly incremental.

Consequently, drastic changes in the process generally does not make it to the agenda of microscopic feedback systems. Thus, even though, for example, each of the several software projects in a company may be progressing well, overall the company may not be optimal. One reason for this is that there is often no synergy amongst the projects. We have found such situations in companies to be quite expensive because at the corporate level it is like chasing several "rabbits running in different directions". This is even more expensive when the "rabbits" are "wild".

What is needed to support "in-the-large" changes in the organisation is macroscopic feedback. Whereas, microscopic feedback systems are aimed at collecting point data (e.g., defect data from programs) and linking causes to process features, macroscopic feedback systems are aimed at examining components of, or even entire, processes (e.g., requirements processes) and linking them to improvement (or other such) goals. It would seem that both microscopic and macroscopic feedback strategies would be needed in an organisation in an appropriate manner. Below, I overview examples from our research that address macroscopic feedback systems.

## 2. Macroscopic Feedback in the Process Cycle

Briefly, the Process Cycle (PC) is a framework for engineering, evolving and enacting software processes [1, 2]. Through its three main sectors, the PC distinguishes the different types of tasks carried out: engineering and improvement of corporate-wide (Sector A) and project-specific (Sector B) software processes, and enacting processes (Sector C) in software projects. It interacts with the "external world" by permitting external changes and situations to be infused into the PC (in suitable forms, such as policies, laws and requirements) through the three sectors.

Internally, the PC encompasses both microscopic and macroscopic types of feedback mechanisms. The microscopic type of feedback includes the finegrain intra-process feedback between Sector C and B (i.e., from development to project-specific process management); and finegrain inter-process feedback within Sector B (i.e., from one project to the next project).

The macroscopic feedback type is categorised further into tight-loop and loose-loop. The "tight-loop" type of feedback is due to the "elicitation" of software process models in project-specific situations (i.e., feedback spans sectors C and B) [3]. Here, descriptive models of processes are elicited from Sector C and are analysed and changed in Sector B. These are qualitative models (graphically or

textually represented) and are analysed with respect to their structural and dynamic properties. Off-line experiments and case studies in Sector B may result in changes to the descriptive models, yielding prescriptive process models which are then injected back into Sector C. This type of feedback is called "tight-loop" because it is project-specific and does not deal with the case of multiple projects.

The "loose-loop" type of feedback is due to the "generalisation" of a suitable set of project-specific software process models (i.e., feedback spans sectors B and A) [4]. Here, a "set of" selected project-specific models from Sector B are analysed in Sector A for "commonality" and "variability" across the set. The generalisation process is driven by concrete requirements (or goals).

For example, one requirement could be: "to identify whether or not there is 65% or more commonality across the set of projects in the way code inspections are carried out". Thus, if there is a type of code inspection (e.g., Fagen inspection) that meets this requirement ( $\geq 65\%$ ) then the generic process model (i.e., the product of the generalisation process) will contain the "Fagen inspection component" and other types of inspections ( $< 65\%$  generic) will not be contained in the generic process model.

This, generalisation process, yields a "descriptive generic process model" which is feedback about the "state of a specific set of processes". Using such feedback as one of the inputs, the process engineer can change this descriptive generic process model into a desired "prescriptive generic process model". The aim of building the prescriptive model is to inject this model in the set of target projects, thus affecting, say, the way inspection is carried out across the set of projects.

In reality, there are other complications that need to be addressed in the elicitation and generalisation processes. For example, elicitation involves securing support from the key people involved in the project concerned in order to elicit "honest descriptive models", and injection of the "improved" process model in the live project is known to be non-trivial due to resistance to change. Similarly, generalisation involves organisation politics both in terms of which project-specific process components (e.g., Fagen inspection vs. others) are accepted in the generic process model and convincing key stakeholders across the set of projects to "accept" the prescriptive generic process model as a starting point for customisation. Clearly, this demonstrates that these macroscopic feedback systems are not centred around technical issues only, and that "people-issues" are critical for the success of such feedback systems.

While we have gained considerable experience in both "tight-loop" and "loose-loop" macroscopic feedback systems individually, as yet we have only intuitive ideas about the "synchronisation" aspects between the two systems within the same organisation. Our experience suggest that this type of research is extremely difficult to carry out at this point in time, especially due to the lack of access to appropriate resources and environments. It would thus be an important topic for discussion at the FEAST workshop how such data, resources and environments can be made accessible in order to facilitate this type of research.

### 3 Tool Support for Macroscopic Feedback

We have been investigating how to support macroscopic feedback in the Process Cycle. For the elicitation process (tight-loop feedback), we have designed and experimented with the Elicit method [3] and supporting tools. Using Elicit, we have built many descriptive models of requirements engineering processes, and plan management and dependency management processes in large organisations. For the generalisation process (loose-loop feedback), we have designed and experimented with the Generaliser method [2, 4] and supporting tools. Using Generaliser, we have built several generic process models from project-specific processes in large organisations.

In addition, we have built a congruence evaluation system (CES) [5] for evaluating the congruence (or fitness) of a process model in the context (or environment) in which it is, was, or will be running. The CES system represents the "best practice" model of a process (we have captured one for the requirements engineering process from about 30 organisations). The best practice model is called a "Contingency Model" (CM), because it identifies specific independent, dependent and contingency variables and their inter-relationships, and their impact on the success of the target process. The variables in the CM are "process" variables and "context" variables. Thus, by feeding into CES the characteristics of a specific process and the context in which the process is running, CES can evaluate the fitness of the process in the given context. This feedback is critical for making changes either to the target process or the environment in which the process is running, in order to improve the congruence of the process in its context.

### 4 Summary

In this position paper, I have distinguished traditional, microscopic feedback from macroscopic feedback. While the former type of feedback has recognised benefits, this paper describes the limitations of this approach and how they are fulfilled by the latter type of feedback. I also describe briefly some tools we are building to help support macroscopic feedback: Elicit, Generaliser and CES.

### 5 References

- [1] Madhavji N H, *The Process Cycle*, Softw. Eng. J., vol. 6, no.5, Sept. 1991, pp. 234-242
- [2] Heineman G, Botsford J, Caldiera G, Kaiser G, Kellner M and Madhavji NH, *Emerging Technologies that Support a Software Process Life Cycle*, IBM Sys. J., vol. 33, no. 3, September 1994, pp. 501-529
- [3] Madhavji N H, Hoeltje D, Hong W and Bruckhaus T, *Elicit: A Method for Eliciting Process Models*, Proc. 3rd Int. Conf. on Software Process, Reston, Virginia, October 1994
- [4] Hong W and Madhavji N H, *A Method for Generalising Software Process Models*, Technical Report, McGill University, Montreal, Canada, 1994
- [5] Perez G, El Emam K and Madhavji N H, *A System for Evaluating the Congruence of Software Process Models*, Technical Report, McGill University, Montreal, Canada, 1994

# FEAST Modelling Language

Suzanne Robertson  
The Atlantic Systems Guild Ltd.

## Introduction

This brief paper is an attempt to produce a modelling language to describe and exploit the basic feedback model proposed for the FEAST project. Fellow feasters are urged to test the model, point out deficiencies and suggest improvements. The model is composed of the components in Figure 1. If each of the components can be made measurable, it defines a modelling language that can be used to build simulation models. Here is my proposal for measuring each of the components in the model:

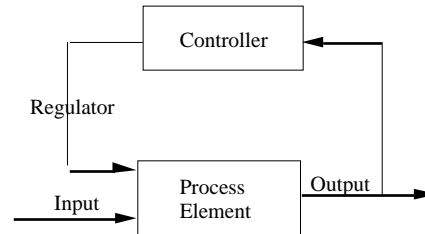


Figure 1 The feedback model proposed at FEAST Workshop 2.

## Process Element, Input and Output

The process element carries out work which is defined by the essential policy of the system. This work can be specified by:

- Definition of the Input and Output flows in the form of:
  - Domain of values for each elemental data item within the flow
  - Valid combinations of elemental data items within the flow
  - Volume
  - Frequency
  - Start/Stop Mechanism
- Specification of the processing rules in the form of:
  - Logic for transforming the input into the output
  - Algorithms

The process element is responsible for making sure that the work done is exactly as per the specification of its processing rules and the definitions of its input and output. The process element will always receive input that conforms to the definition of the input. The process element will always produce output that conforms to the definition of that output.

In other words, the process element is “controlling” the quality of its own work by checking to make sure that the work is carried out as specified. It is not important to say how this checking is carried out because different instances of a given process might be implemented very differently. The important point is that the checking requirements are inherent in the specification of the essential policy of the process element and are not the concern of the controller.

## Controller

The controller evaluates output from a process element (or a number of process elements) according to the specification of its control rules. The controller might send a regulator to the process element for the purpose of regulating the essential policy of that (or those?) process elements. The controller is specified by:

- Definition of its input, this definition is the same as the definition of the process element’s output
- Specification of the decision making processes carried out by the controller in the form of:
  - Logic for transforming the input into a regulator
  - Algorithms

## Regulator

The regulator is produced as a result of a controller carrying out its specified control rules. The regulator is specified by:

- Specification of new or changed processing rules for the process element that receives the regulator in the form of:
  - Logic for transforming the process element’s input into its output
  - Algorithms

## An Example to Test the Modelling Language

This example tests the modelling language and raises some questions.

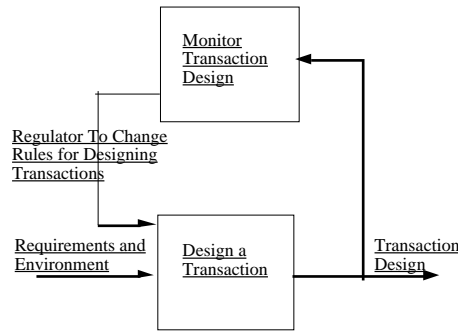


Figure 2 An example using the feedback model

In Figure 2, the process element contains rules that specify how to design a transaction for this particular environment.. When this processing takes place it receives Requirements and Environment exactly as per the input specification and it produces a Transaction Design exactly as per the output specification.

The controller, Monitor Transaction Design, carries out its specified control rules and perhaps it produces the regulator: Regulator to Change Rules for Designing Transactions. For example, this regulator might tell the process unit to change its processing rules and to design the transactions at a higher level of detail.

But has the controller got enough information to do its work? It is reasonable to suppose that Monitor Transaction Design, might need other input. This other input might be in the form of previous work done by this process element ie:other Transaction Designs. Alternatively it might be the result of work done by other process elements within the system.

Also, the controller could conceivably produce a regulator which is directed at another process element within the system. For example, suppose that monitor transaction design, discovered a fault in the transaction design that was due to work done by another process element in the system. Then the controller would send a regulator to that other process element.

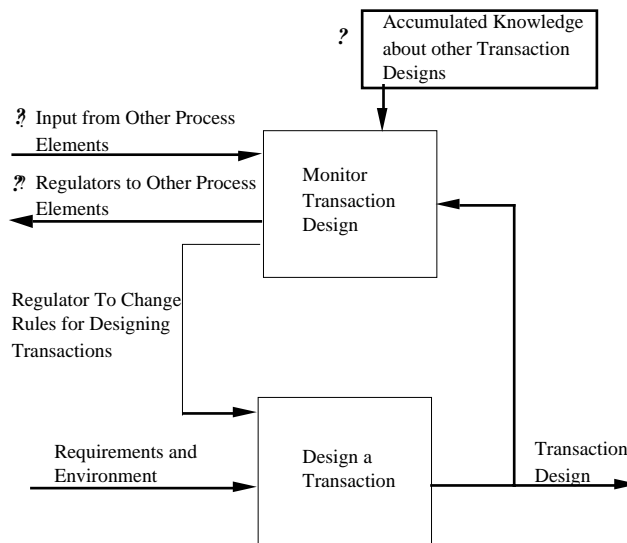


Figure 3 Some questions raised by the example

- The questions raised in Figure 3 indicates that the modelling language needs:
  - Some way of representing stored knowledge
  - Some way of representing input to a controller from a number of process elements
  - Some way of representing regulators from a controller to a number of process elements

## Conclusion

The components in the modelling language can be and must be measurable if we are to build simulation models. This excludes from the controller any decision making which has the nature of “ad hoc” or “experience and judgement”. In other words, our model excludes any human decision making that cannot be specified as a set of logical rules. However, as soon as a piece of policy, hitherto non specifiable, becomes specifiable then that policy can be added to the model. This is certainly a way of giving ourselves a definite boundary for the FEAST feedback models. The danger of focusing on deterministic policy is that we might forget that FEAST is concerned with understanding the agents that cause E-Type behaviour in systems.

## Fruits of a Initial Search for an Example

Carolyn Story

BNR Europe

In examining the impact of end user/customer attitudes it came as a surprise to me to find that in the five projects I examined they did not appear to have any major impact on the further development of the technology. So I started looking at the impact of various roles on a project, and on the choice of technologies. I have not reached any earth shattering conclusions, but will briefly report on the results obtained so far.

The first solid conclusion is the need for a good diagrammatic notation. I am having some difficulties in expressing things in terms of feedback loops and this itself may say something though it must be emphasised that the activities reported do not represent an in depth study. The notation must provide some simple, consistent, fairly rigorous, means of modelling precisely what is going on, yielding models, descriptions, etc. that can be used for communication purposes rather than to facilitate rigorous mathematical analysis. One of the main success criteria for such a modelling technique will be the ease of use and the clarity with which it shows up, previously hidden, feedback loops!

Adopting a pragmatic approach, I started by looking for an application area which would provide a rich environment for a FEAST study. Very quickly I came up with some basic questions - and many reasons why it might be difficult to apply Feast ideas! These initial impressions led to the examination of a set of projects with different classifications (upgrades, evolutionary, revolutionary, etc., a la MIT 90's). Each displayed its individual characteristics but the really influential factors impacting the use of new technology came from the software development *team* in its widest sense. The other, wider, factors/influences did not appear to exercise major influence on the technology decisions. There is clearly much to be investigated.

# A Survey and Comparison of Some Research Areas Relevant to Software Process Modeling

Terje Totland and Reidar Conradi  
University of Trondheim, Norway

## 1. Introduction

Few research areas have received so much interest from the software engineering community during the past decade as the software production process [Curtis 92]. The main objective of this increase in interest is to improve software production in terms of increased product quality, reduced costs and reduced time-to-market.

The Software Engineering community seems to agree that a fruitful way to support the software production process is through the application of process models. Extensive literature on the issue is available, and for a comprehensive "state-of-the-art", see [Promoter 94]. However, the SE community is not the only one focusing on process technology as a means to improve work. This paper presents a brief overview of other communities doing software process technology related research, and points out some main similarities and differences between the research areas.

## 2. Dimensions for Comparison of Research Areas

There are at least three dimensions that can be used for comparison of the various research areas: Business domain, intent and process elements. By business domain is meant the main business area of the organizations that are modeled. The business domain greatly influences the requirements to process technology in each area. Business areas include software production, manufacturing, insurance etc. By intent is meant the primary objectives of applying process technology to a specific business domain. The purpose of process modeling ranges from individual understanding of the process to full enactment. Last, process modeling within disparate business domains and with different intent requires focus on different aspects (process elements), such as objectives, activities, artifacts, roles, actors, tools etc.

## 3. Some Research Areas Utilizing Process Technology

Presentation of each relevant research area is very brief due to the nature of this paper.

### 3.1 Software Engineering (SE)

Software Engineering is "the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines" [Naur 69]. The field of Software Process Modeling can be considered a subdomain of Software Engineering.

Obviously, the business domain of SE is software production. The intent of applying process technology ranges from understanding and visualization of the process to partly automated enactment. Process elements include activities, artifacts, roles, and tools. Artifacts are usually not modeled in depth, i.e., there is no data modeling in the traditional sense.

Both the actual production process and the metaprocess is modeled. By metaprocess is meant the building and maintenance of a process model and its related activities. Handling the metaprocess is important in SE, as the production process changes frequently and the process models should evolve likewise. To date, emphasis on the importance of the metaprocess is particular to SE.

### 3.2 Information Systems Engineering (ISE)

Information Systems Engineering can be defined as application of a set of systematic engineering approaches to develop information systems. An information system is a system of computer components, software components, and human and organizational components that are developed, trained and assembled to fulfill the information processing requirements of a problem [Solvberg 93].

The business domain is usually information services, like banking and insurance. The intent is understanding, analysis and communication of domain knowledge in order to construct the information system. Monitoring, measuring and enactment of the process is not in focus. Process elements are as for SE, except that ISE traditionally has incorporated more of business rules and human actors into their models than SE. ISE also models the artefacts in more detail (data modeling).

### 3.3 Enterprise Modeling (EM)

Enterprise Modeling is a term that has recently emerged. The term has been given various definitions, but one representative for most applications says that "Enterprise Modeling is the process of understanding a complex social organization by constructing models" [Rumba 93]. The main focus seems to be on modeling as a process, and not on the model as the important outcome.

EM is domain free, meaning that it is not constrained to any particular business domain. Intent is mainly to gain understanding and to discuss the process. Process elements include objectives, business rules, activities, artifacts and roles.

### 3.4 Business Process Re-engineering (BPR)

Business Process Reengineering is defined as "the fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical, contemporary measures of performance, such as cost, quality, service, and speed" [Hammer 93]. BPR is thus concerned with completely rethinking how a service is provided or a product manufactured from scratch, without being constrained by current processes or organization.

As for EM, BPR is domain free. Any kind of business can be reengineered. The intent of using process technology is mainly understanding of the current process, and communication and analysis of the future process. Process elements are often just activities and roles.

### **3.5 Organizational Design (OD)**

Organizational Design is the study of organizational performance using process models and computers. Given a business problem (e.g., "How will attendance to project meetings influence the product quality?"), a process model of a hypothetical organization is built. The expected performance of this organization is then predicted based on computer simulations on the model. OD makes extensive use of organizational theory in order to build models that result in accurate predictions. Two major efforts within this area are the Virtual Design Team project at University of Stanford [Christ 93] and the Process Handbook project at MIT [Malone 92].

OD is, as BPR and EM, domain free. The main intents of building process models are understanding and simulation. This requires more formal and detailed models than within e.g., EM and BPR. Process elements are activities, artifacts, roles, actors and tools. A notable difference from SE is the often detailed modeling of human actors, and their capabilities. The similarity to BPR is prominent, however, OD seems to put much more emphasis on computer simulation than the others.

### **3.6 Workflow Management (WM)**

Workflow can be defined as "the sequence or steps used in business processes" [Marshak 91]. Marshak also requires that more than one person is involved in the process, and that the workflow consists of both sequential and parallel steps. Workflow Management is supporting and controlling the workflow.

An important objective in Workflow Management is to automatically route artifacts (documents) through a network to actors having predefined roles. The routing is done according to a set of predefined rules, and is often controlled by the state of the artifact (e.g., the price). This approach requires a relatively stable business process, as the rules are not meant to be changed on the fly.

The business domain of WM is information services, as for ISE. The intent of using process technology within WM equals the ones for SE, i.e., the full range from understanding to automation. Process elements in focus are business rules, activities, artifacts and roles. Business rules are particularly important for routing of artifacts.

### **3.7 Concurrent Engineering (CE)**

The most frequently referenced definition of Concurrent Engineering is provided in [Winner 88]: "CE is a systematic approach to the integrated concurrent design of products and their related processes, including manufacturing and support. This approach is intended to cause the developers, from the outset, to consider all elements of the product life cycle from conception through disposal, including quality, cost, schedule, and user requirements."

The main goal of CE is to run more activities in parallel with the aid of information technology and organizational restructuring. In addition, a life-cycle perspective on the product is encouraged, in order to reduce overall costs. The main benefits of successful implementation are reduced time-to-market, improved product quality and increased productivity, leading to lower costs [Carter 92].

CE is in principle domain free. The intent is to focus on understanding, communication, and analysis of the current process in order to develop a new and improved one. The main process elements are activities, roles, and artifacts.

### **3.8 Computer Integrated Manufacturing (CIM)**

Computer Integrated Manufacturing is defined by [Rembold 93] as follows: "CIM conveys the concept of a semi- or totally automated factory in which all processes leading to the manufacture of a product are integrated and controlled by computer."

The business domain of CIM is manufacturing. However, CIM focuses on processes that face replication risks, as opposed to SE, that face design risks [Bollinger 91]. The intent is simulation, planning, measurement, and automation. Main process elements are activities, artifacts and tools. A notable difference between CIM and other disciplines is the lack of focus on human actors.

## **4. Concluding Remarks**

Our brief comparison of research areas shows that all fields have much in common with Software Engineering concerning application of process technology. The intents and process elements overlap more or less, even if the business domains differ.

One main finding is that the disciplines that are most focused in their business domain (like SE and CIM) have intentions of using process technology to a further degree than the disciplines that are domain free. They may also benefit the most.

Another main finding is that all areas intend to support applications of process technology that does not require formal models, while only a few support applications that do require formal models (SE, WM, CIM). Formal treatment of models may pose extra requirements, like model completeness and consistency.

SE may learn most from other disciplines concerning how to develop process models that are intuitively understandable for human beings (as focused by EM), and in a way that facilitate worker cooperation and motivation for organizational changes (like required for BPR). SE may also use more of the organizational theory that is the foundation of especially OD.

Particular to Software Engineering is the focus on metaprocess. It is obvious that this should be present in other areas as well, as reflection on the way process technology is applied is of utmost importance for improvement.

## **Acknowledgements**

We would like to thank Guttom Sindre at IDT, NTH for his constructive critique, and valuable suggestions for improvements. Also thanks to Brian Warboys at University of Manchester and anonymous referees for helpful comments.

## References

- [Bollinger 91]T. B. Bollinger, C. McGowan: *A Critical Look at Software Capability Evaluations*, IEEE Software, pp 25 - 41, July 1991.
- [Carter 92]D. E. Carter, B. S. Baker: *Concurrent Engineering - The Product Development Environment for the 1990's*, Addison-Wesley, Reading, Massachusetts, USA, 175 pages, 1992.
- [Christ 93]T. R. Christiansen: *Modeling Efficiency and Effectiveness of Coordination in Engineering Design Teams*, PhD thesis, CIFE, University of Stanford, California, USA, 317 pages, 1993.
- [Curtis 92]B. Curtis, M. I. Kellner, J. Over: *Process Modeling*, Communications of the ACM, pp 75-90, September 1992.
- [Hammer 93]M. Hammer, J. Champy: *Re-engineering the Corporation: A Manifesto for Business Revolution*, Nicholas Brealy Publishing, London, UK, 223 pages, 1993.
- [Malone 92]T. W. Malone, K. Crowston, J. Lee, B. Pentland: *Tools for Inventing Organizations - Toward a Handbook of Organizational Processes*, working paper, Center for Coordination Science, MIT, Massachusetts, USA, 21 pages, October 1992.
- [Marshak 91]R. T. Marshak: Perspectives on Workflow, In T. E. White, L. Fischer (ed.): *New Tools for New Times - The Workflow Paradigm*, Future Strategies, Inc., Alameda, CA, USA, pp 165-176, 1994.
- [Naur 69]P. Naur, B. Randell (ed.): *Software Engineering: A Report on a Conference sponsored by the NATO Science Comm.*, NATO, 1969.
- [Promoter 94]A. Finkelstein, J. Kramer, B. A. Nuseibeh: *Software Process Modelling and Technology*, Advanced Software Development Series, Research Studies Press/Wiley & Sons, 362 pages, 1994.
- [Rembold 93]U. Rembold, B. O. Nnaji, A. Storr: *Computer Integrated Manufacturing and Engineering*, Addison-Wesley, Wokingham, UK, 640 pages, 1993.
- [Rumba 93]J. Rumbaugh: *Objects in the Constitution - Enterprise Modeling*, Journal on Object-Oriented Programming, pp 18-24, January 1993.
- [Solvberg 93]A. Solvberg, D. C. Kung: *Information Systems Engineering - An Introduction*, Springer-Verlag, Berlin, Germany, 540 pages, 1993.
- [Winner 88]R. I. Winner et al.: *The Role of Concurrent Engineering in Weapons System Acquisition*, Report R-338, Institute for Defense Analyses, Alexandria, VA, USA, 1988.

# Using Process Waiver Data to Improve a Design Process A Case Study of Feedback and Control Using the FEAST Model

Lawrence G. Votta and Mary L. Zajac  
AT&T Bell Laboratories

## Abstract

How do organizations use feedback to change their processes, and hence, improve their performance? A small working group is proposing that any organization under **control** almost always should be using feedback to stabilize and improve their products.

We describe the results of applying the **Feedback, Evolution, And Software Technology** (FEAST) feedback model to our work on improving the software design process of 5ESS®. The translation of the study into a FEAST model took only a couple of hours. The model concisely captures both managers and developers understanding of how the design process management team controls the definition of the software design process. However, the model needs to be extended to explicitly show the lag time between the time when the control signal is sampled and reapplied back to the design process. Several other questions are posed as important discussion points to further improve the FEAST model.

## 1 5ESS Background

Software development as practiced at AT&T's 5ESS development organization can be visualized as consisting of five major steps: feature specification, software design, coding, testing, and delivery. A feature is a unit of functionality that can be sold to customers. It is the fundamental unit tracked by project management. Features may vary from a few NCSL and no hardware to 50,000 NCSL with many complex hardware circuits developed specifically for that feature. Weider Yu et al. describe the 5ESS software development process and give examples of a feature in [3].

On a large project such as 5ESS, hundreds of features are being simultaneously developed. These features are packaged together into releases that will be delivered to customers. Packages come in a spectrum of sizes that take anywhere from three to twenty-four months to develop. Features of all different sizes and complexities are packaged together, and so the development process of features for a particular release are started and completed independently. This highly flexible approach to packaging customer features allows high feature development throughput and helps meet customer demands for customized features.

Figure 1 shows pictorially how features start and stop relative to the stage of introduction of a process change. The heavy vertical lines separate the three phases of process change; **OLD**, **MIXED**, and **NEW**. The horizontal lines with bars on either side show the duration of the feature. The two problems are managing the mixed phase when more than one version of the process is in use. The other problem is assessing whether the new process is having the desired result especially when a representative sample of features is needed for assessment.

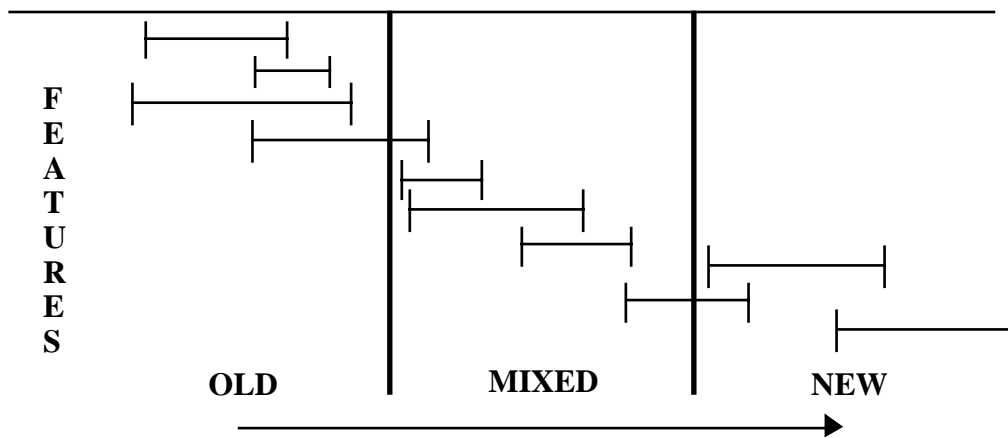


Figure 1: The Assessment Problem.

In this environment, process changes are generally introduced and made effective on a date, as opposed to by release. All features in development at the date of the process change are either grandfathered to use the old process (usually due to a concern that the change would disrupt the feature development) or switched over to the new process. In general, all features already in the stage of executing the old version of the process do not switch over to the changed process. This makes the actual date of a process change rather difficult to determine on a global level; but, allows minimum disruption for the development process.

## 2 Model Description

The FEAST project has proposed a fundamental process unit model consisting of input, output, feedback controller, and resources displayed in a simple diagram prescribed in the *FEAST Manifesto* [1]. On a sizable project like 5ESS, this model can be employed at several levels of abstraction:

- 1 the global project level,
- 2 globally at the development phase level,
- 3 the individual feature level
- 4 the individual developer level.

Figure 2 shows the 5ESS software design process cast literally into the FEAST model, an example of abstraction level 2.

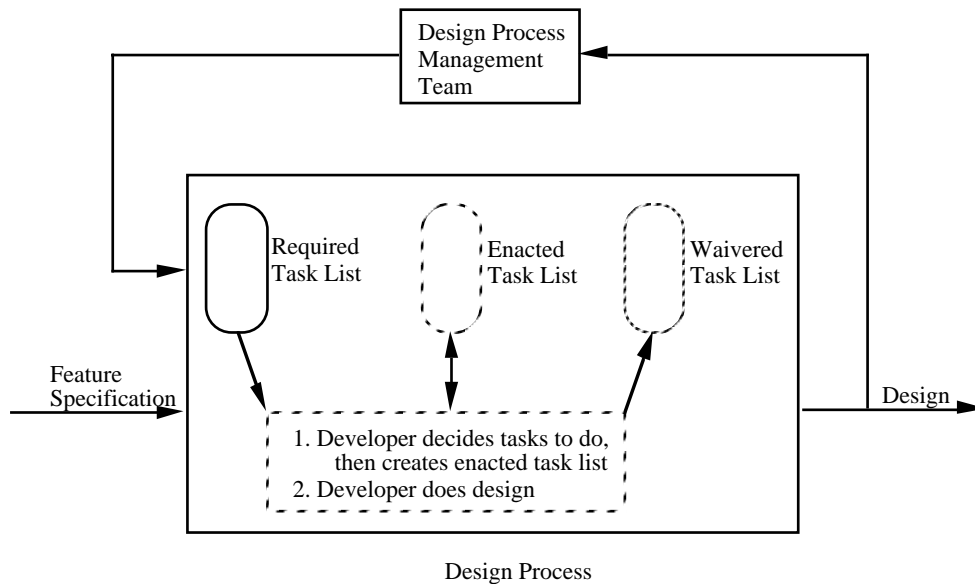


Figure 2: 5ESS Software Design Cast Into the FEAST Model.

The inputs to the design process are feature specifications and feedback from the design process management team. The output of the process is the feature design and the list of tasks that were considered not necessary for a specific reason. The decision to allow the design process to be enacted with these exceptions is yet another process that involves the manager responsible for the definition of the software design process. Finally, the resources consumed are developer time, computing resources, and other expenses incurred to successfully complete the design.

However, Figure 2 is deceptive. At this level of abstraction, there is always a lag time between the recognition in a manageable, economic way. There is another lag time before the effectiveness of the changes (that is the next burst of feedback) can be assessed in a meaningful way, since a good mix of features have to have executed the new process in order to generate representative feedback. The feedback cycle on this level of abstraction appears to be quite long, bursty, and somewhat imprecise by nature.

A proposed improved FEAST model diagram of the 5ESS software design process is shown in Figure 3. Here we have followed the prescription of the FEAST Manifesto; however, we have explicitly shown when the feedback takes effect. This is important to understand (especially in large organizations) since it defines the time scale in which any change can be observed and assessed.

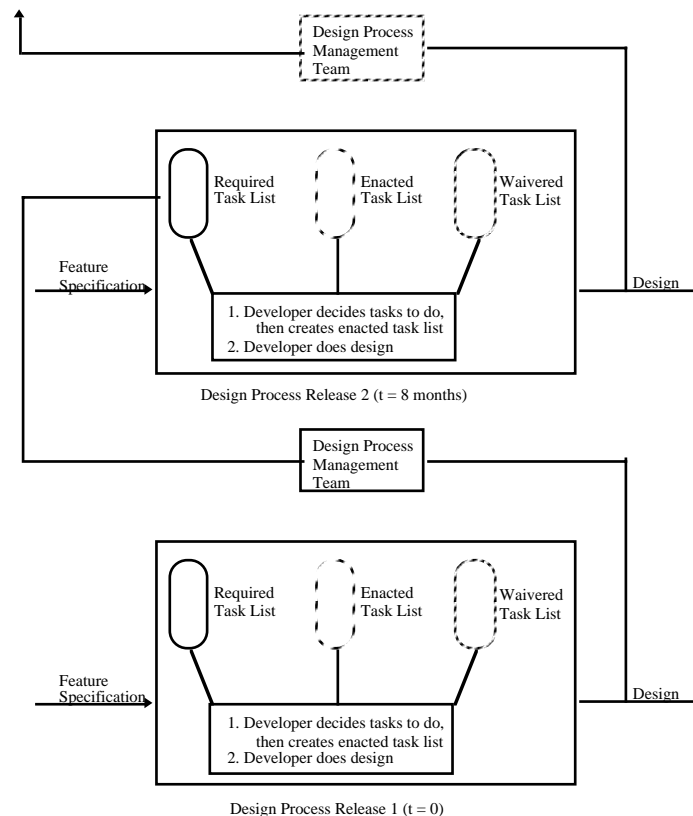


Figure 3: 5ESS Software Design Process Cast Into FEAST Model Showing Time Lag.

### 3 Discussion Questions

While doing the modeling of the 5ESS software design process, we had the following questions and discussion ideas for the next workshop.

1. The lag in time of the control signal is an important element of analyzing any system employing feedback control. The model needs to make this more explicit since it defines the time scale by which the effect of any change can be observed and accessed.
2. Understanding what to use as your control variables is extremely important and not easy to do. We offer as evidence our study of the 5ESS software design process where we recognized the importance of process enactment waiver data [2].
3. Different abstraction levels of the FEAST model result in different control characteristics such as feedback lag time and precision. We have proposed four levels in our study. The hierarchy needs to be defined and feedback/control capability understood for each.
4. In our simple model we did not include some important feedback loops in the interest of keeping the model simple. Additional loops come from other process phases – coding, test, and field performance. This complicates the feedback system and will probably lengthen the time lags mentioned above.
5. Work needs to be done on defining what constitutes useful feedback in an essentially unrepeatable system. In software development, each development project is unique (at least characterizing their similarities has been elusive), and the process is executed by people that are subject to the effects of learning, external distractions, and skill level.
6. There are many style issues revolving around audience. For instance, we would never show the feedback models to member of the 5ESS development team as shown in Figures 2 and 3. Figure 2 lacks fidelity of how it really happens, and Figure 3 does not contain the detail that would be credible to a 5ESS audience.

#### References

- [1] Lehman M M, Perry D Turski W M. *FEAST Manifesto*, FEAST Technical Report, 1995. Request from Dewayne Perry electronically at dep@research.att.com
- [2] Votta L G and Zajac M L, *A Design Process Improvement Case Study Using Process Waiver Data*, Submitted to the Fifth European Software Engineering Conference, September 1995.
- [3] Weider D. Yu, D. Smith P and Huang S T, *Software Productivity Measurements*, AT&T Technical Journal, 71:pps. 110 - 120, May/June 1990

## Introduction

Numerous presentations have shown that the FEAST concepts can be convincingly conveyed to a variety of audiences including funding agencies. While the latter all plead poverty at the present time, they were encouraging in their reception of the concepts and in requesting a proposal that would detail the proposed investigation, groups and individuals participating, a plan of action and resources required with timetable and deliverables. The text that follows constitutes the beginnings of a first draft that will form the basis for the preparation of individual proposals.

## Background

The last twenty years or so have seen an order of magnitude growth in the functionality of computer applications and, hence, in the size and complexity of the software systems implementing them. It is, nevertheless, well established that processes transforming initial statements of requirements into installed software products still suffer from serious shortcomings. The continuing problems faced in system development and maintenance have led to strong pressure for systematic improvement of what has become known as *the process*. The invention of symbolic, assembly and high level languages in the fifties initiated the search for process improvement [ran94]. Public discussion of the many problems facing industrial software development at the 1968 NATO Garmisch conference [nau69] triggered a more general effort. Since then there have been significant advances in the provision of means for such development. Major R & D investment has produced numerous concepts, methods, techniques and tools. High level languages, structured programming, abstract data types, formal methods, non-procedural programming, object orientation, CASE, support environments exemplify innovations that were each, in turn, expected to overcome the many problems that have for so long frustrated consistent, cost effective, on-time development of functionally satisfactory and reliable software. None did.

## The State of the Process

Despite the local benefit that innovation has brought to individual development steps too many problems still haunt industrial software development [gib94]. Introduction of improved methods, techniques and tools has not yielded a consistent capability for planned, on time, controlled-cost development of quality software that is functionally satisfactory when delivered. Nor has it resulted in major productivity growth, cost reduction or faster response to user needs. It has proven equally difficult to achieve major improvement in maintaining systems satisfactory as user needs and expectations change in evolving operational domains. Software *evolution* from concept to first installation and from release to release still relies on processes that are far from satisfactory [gib94].

## The Drive for Improvement

Direct interest in the process as such has existed since at least the late sixties [leh69, roy70]. Wider interest was encouraged by the first International Software Process Workshop [pot84]. Despite doubts about the specific direction being recommended [leh87], Osterweil's keynote address to the ICSE 9 conference [ost87] triggered major activity in process modelling based on programmatic concepts. Most recently the focus of interest, particularly in industry, has turned to process standards and maturation as exemplified by the ISO 9000, TickIt and Spice process related standards and the SEI CMM model. Some industrial and commercial organisations have launched their own efforts both to obtain process quality certification and through R & D improvement programs such as that described at ICSE15 [maj93]. But progress is painfully slow. This at a time when the rapid spread of computer application and its ever deeper penetration into every facet of human activity implies ever increasing dependence, individual and collective, on maintaining implementing software and related software processes satisfactory in changing operational and maintenance environments [leh91].

## Why the Slow Progress?

The limited improvement provided by innovations such as those listed above can be individually explained, innovation by innovation. The failure of formal methods, for example, to yield generally visible global impact may be related to perceptions that their successful use requires mathematical skills and that the beneficial impact of their adoption is primarily process-local. CASE tools frequently yield less benefit than expected because, being acquired one at the time, they cannot easily be used together, may not even be able to communicate directly. Integration of their outputs often requires more effort than the effort saved through their introduction. Hence, their net contribution to productivity improvement say, is small or even negative.

The extent of the failure of innovation to yield major global benefit, however, suggests an overall constraint inhibiting process improvement. Is there perhaps a common phenomenon resisting or even blocking improvement? If such can be identified a path to major improvement will have been opened. Techniques and tools to enable industry to exploit the opportunities it offers will follow.

## Information flow in the Software Process

Processes whereby software is evolved *ab initio* or enhanced and extended, involve extensive feed around of information. The flow will be both internal to and between various activities and agencies (sub-organisations). Each of the latter will have its own responsibilities and objectives. Their joint action and interaction produces information, insight and enlightenment that drives, guides and controls the process. Some flow will influence process steps that follow the information-generating activity. Other will be fed back to *control* the sub-process from whose output it was derived. The process constitutes a complex information flow network. Since output from some process elements controls their subsequent behaviour the process constitutes a multi loop feedback system [leh94]. A forthcoming publication by Lehman, Perry and Turski will discuss the role and likely impact of

feedback controls in processes whereby *E*-type<sup>1</sup> applications and software systems are evolved. For the moment it is sufficient to observe that for over four decades industrial software *process* evolution has been driven, at least in part, by organisational needs and ambitions. To ensure stability as an organisation seeks profitability and growth, feedback based *checks* and *balances* controls will have been applied<sup>2</sup>. Feedback controlled stability is a natural property of *E*-type processes.

### The Software Process as a Feedback System

A characteristic property of all feedback systems is that the stabilising effect of negative feedback reduces the global impact of changes to individual forward path elements. In the limit, externally observable system behaviour will not visibly change in response to forward path changes unless and until excessive positive feedback causes system instability. To obtain the full impact on system behaviour from forward path changes requires adjustment of feedback paths and mechanisms.

As a feedback system, the software process must be expected to display this characteristic property. Thus even if there were no other feedback controls in the processes, checks and balances alone would tend to constrain the global impact of forward path improvement of internal processes. In practice there may be many other controls inhibiting process improvement as a side effect.

### Negative Feedback as a Constraint on Process Improvement

The innovations mentioned so far all represent changes to the forward path of the software process. In general their introduction into practice was rarely, if ever, accompanied by review and adjustment of process feedback controls. It should, thus, not come as a surprise that their impact was limited. Is this not simply an instance of the stabilisation normally associated with negative feedback? For the moment, this observation is a conjecture to be verified in theory and practice. Once it has been verified one may develop means for its exploitation. Support for this conjecture is provided by other innovations, inspection, incremental release and metrics for example that are feedback based and are recognised as having made significant contributions to process effectiveness

### Exploiting Feedback

This analysis suggests that major software process improvement requires continuing review of feedback mechanisms. To date this aspect of improvement has been largely ignored. Hence, identification and adjustment of feedback mechanisms in current processes may itself be expected to produce improvement. In addition, the development of appropriate modelling and other techniques with their support tools will ensure that, in the future, relevant feedback controls will be identified and adjusted in tune with each forward path improvement to yield maximum benefit.

### Historical Note

The beginnings of Software Engineering as a separate and distinct discipline are generally associated with the 1968 NATO Garmisch Conference [nau69]. At that meeting there were numerous references to feedback and its critical role in the software process. The term strictly was used in its non-engineering connotation of information flow fed back to participants in some process or originators of some product, eg. from testers to developers or from users to manufacturers. It was recognised that such information directed to the right destination could provide guide lines, directions to be pursued, enlightenment. It was not seen as a control or process improvement mechanism. Moreover, whatever impact the concept may have had appears to have been restricted to that conference. In the Proceedings of the follow up conference a year later [bux70] the word *feedback* does not even appear in the index. Feedback was also mentioned at about the same time in a report *The Programming Process* [leh69] and discussed in more detail in subsequent papers [bel72, leh85]. This work recognised the role of feedback in determining the dynamics of the software process and the *Laws* that appeared to regulate the release process. As an example brief mention may be made of the early observation of the feedback stabilised growth characteristics of the early stages of OS/360 evolution and that of other systems [leh80] and the equally persuasive instability (chaos?) consequences of excessive positive feedback. All this is illustrated in the growth plot reproduced in figure 1 below. Its phenomenological consequences are expressed by the Laws of Software Evolution that were recognised as reflecting the consequence of feedback in the dynamics of program evolution [leh78,80,85,89].

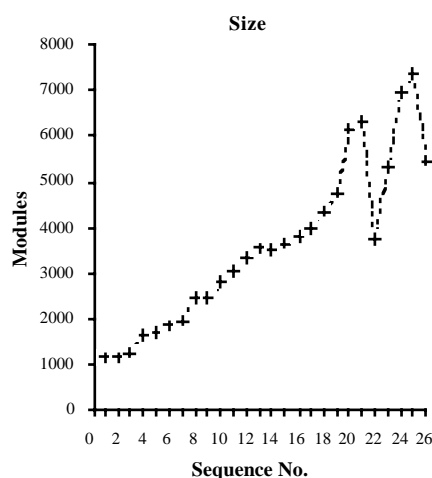


Figure 1: Feedback Stabilisation and Instability of the OS/360 Evolution Process

<sup>1</sup> *E*-type software is defined as being required to *satisfactorily* address a real world problem in contrast to *S*-type software that is required to be *correct* in the full mathematical sense with respect to a fixed specification [leh78, 94b]

<sup>2</sup> Others were developed or evolved to support the search for improved product quality and process productivity. A third process attribute, responsiveness, whilst of equal societal significance has not been so avidly pursued.

As observed with reference to the 19 data points available at the time of publication [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 budgets varying, increasing numbers of users reporting faults or desiring new capability, varying management attitudes towards system enhancement, changing release intervals and improving methods....". The last seven data points included in the plot, but not available at the time of publication, reinforce the evidence for feedback system like behaviour by showing the consequences of excessive positive feedback that resulted in the breakup of OS/360.

More recently, Abdul-Hamid and Madnick devoted a book to aspects of software process dynamics, concentrating on management aspects [abd91]. Weinberg [wei92] and Senye [sen93] provide extensive discussion of related themes. But it appears that the first published suggestion that, unless recognised and taken into account, the feedback nature of the software process is likely to constraint software process improvement did not appear till 1994. It is difficult to understand why the penny took so long to drop but then hindsight always raises that question.

### **Warning**

Feedback management and exploitation appears to have significant potential in process improvement. A word of warning must, however, be expressed. In papers at IFIP Congress '86 Brooks [bro86] and Turski [tur86], respectively, pointed out that one must expect neither a silver bullet nor a philosophers' stone to solve the software engineering problem once and for all. The FEAST conjecture is not an exception. If we can learn to exploit it, it is likely to make a significant contribution to process improvement. Its application will not overcome all problems.

### **Project FEAST**

A comprehensive investigation to confirm the conjecture, identify feedback paths and mechanisms and develop means for controlling and exploiting feedback is clearly required. The investigation must seek systematic methods of feedback based process improvement, means to support such improvement and means to assess the global impact of process element changes. The first step is the development of a conceptual and theoretical framework that includes means for the representation and evaluation of global processes.

An international project, FEAST, with these objectives is now underway. The study will examine the role of feedback control in the process, process improvement and the exploitation of advancing technology to this end. If successful it will ensure and sustain future advances in software evolution (development *ab initio*, enhancement, extension) and provide methods, tools and metrics for the effective evaluation, support and further development of process technology.

Three recent workshops have involved people from industrial, academic and research organisations in Canada, Finland, France, Norway, Poland, Portugal, UK and USA. The main focus of the discussion so far has been on the identification and definition of basic concepts, the outline definition of project issues and objectives and discussion of the initial models. This effort has been funded in part by the UK Department of Trade and Industry and in part by participants' organisations. Present support ends, effectively, in March 1995. Future progress will depend on obtaining further funding.

The envisaged multi-disciplinary project is challenging but feasible. The first practical results should be available within two years say. But in view of the difficulty of the issues under study the main body of results is likely to require 3 to 5 years to achieve. The degree of success and the rate at which it is achieved will clearly depend, at least in part, on the funding obtained. The calibre of people attracted to and participating in FEAST suggests that significant progress can be anticipated.

### **FEAST Issues**

As discussed elsewhere [leh94] the focus of project interest will be *E*-type applications [leh78] and software. Specific issues to be addressed and development to be pursued in the project include:

- Theory of software evolution, process and process evolution
- Role and nature of feedback and control in the software process
- Clarification and validation or refutation of the feedback conjecture
- Determination of the intrinsic properties of *E*-type software
- Determination of the degree to which feedback control makes major progress in process improvement difficult to achieve
- Relevance and applicability of classical feedback and systems theory and practice
- Development and/or evaluation of alternative representations
- Investigation of methods for global level modelling of software product evolution and process improvement that include human action in feedback and control mechanisms
- Model characterisation, representation, construction, quantification, evaluation and correlation with industrial processes
- Analysis, description and quantification of *E*-type process dynamics as previously recognised [bel72, leh85] and its relationship to process theory and system dynamics/control theory
- Practical implications of the dynamics and its industrial relevance
- Determination of practical methods for exploiting results of the study, how they may be evaluated and how they may be introduced into practice
- Measurement or other assessment of likely impact on global process attributes such as quality, productivity, responsiveness, determinism and on their improvement
- Development of methods and tool support for process evaluation and improvement

## Approach

The project requires a two headed approach. It must develop a theoretical framework and its support tools, importing models, methods and techniques from relevant disciplines such as stochastic control theory, system dynamics and other areas where feedback control is discussed and evaluated. Equally it must intensively follow a practice based investigation of the feedback characteristics of industrial software processes, identification of feedback mechanisms, their effect in the global process and assessment of their likely impact. These, initially two, approaches must rapidly converge to permit the construction of realistic models of feedback controlled industrial processes that can be evaluated, modified, re-evaluated and used to define, implement and, perhaps, optimise improvements to real world processes.

## Tasks

Initial tasks are underway. Alternative definitions of terms such as *feedback* and *control* have been analysed, the most appropriate adopted. Basic feedback configurations have been defined and successfully applied to the development of models of sub-process elements of both industrial and theoretical model. Other examples of sub-processes have been analysed and discussed as part of a critical examination of the viability of the initial concepts, definitions and models and to expose any limitations they may impose. An initial conceptual global feedback process model has been defined. Outputs from and information flow between its sub-organisations and activities, have been listed [per95]. **{could be included as appendix}**. This model and the listings provide the basis for a simulation model that will permit global analysis and improvement once metric attributes such as volume of flow, time delays, resource (person-time) needs have been defined and estimated. Other common standards, viewpoints and approaches have also been formulated [per94] **{Note: could be an appendix}** Such modelling and evaluation must be extended and intensified to identify feedback loops and their likely impact in the global organisation and software development, release and evolution process.

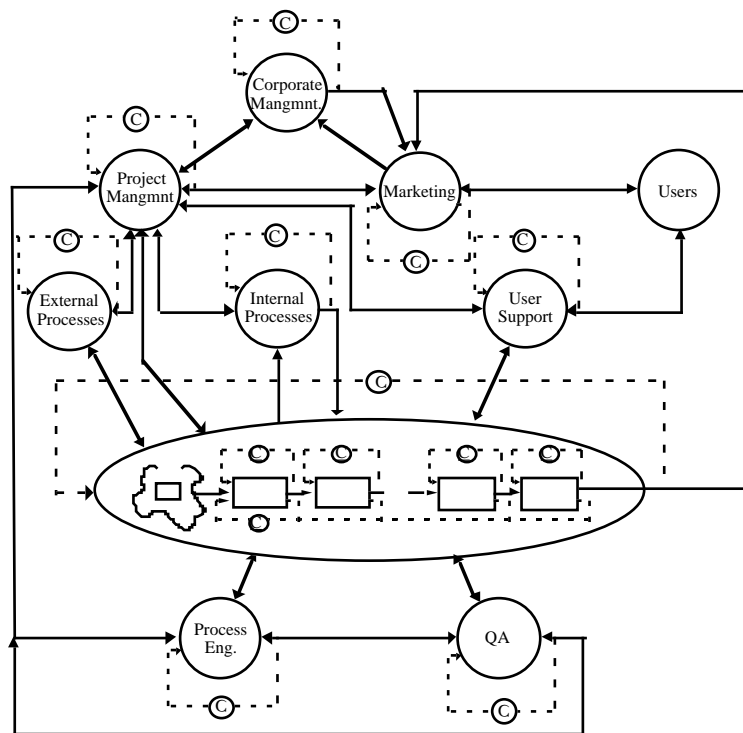


Figure 2: The Initial FEAST Global Process Model

It will be of considerable interest to examine existing process models such as the Waterfall [roy70, boe76], Spiral [boe88] and the SEI CMM models. These variously reflect technical and managerial aspects of the process but non provide the global coverage, including feedback control, pictured preliminarily in figure 2 and required under the FEAST conjecture. FEAST activity must adapt and extend these models to test the viability and applicability of the FEAST assumptions, adapting them if necessary, and to evaluate the resultant models for validity and process improvement.

It will be also necessary and is intended to explore, within the context of the software process, the application of modelling, evaluation and optimisation techniques as used in control theory and system dynamics studies to determine whether these have a role to play in the synthesis and analysis of software processes. Some prior work does exist and initial successes have been reported but it seems unlikely that these approaches will yield quickly exploitable results.

The above mentioned tasks concentrate on methods and techniques for developing and exploiting feedback control in process improvement. It is also intended to identify feedback paths and mechanisms in current industrial processes and to determine whether and to what extent these have had a constraining influence on the benefits obtained from technology and technological innovation. Success in this area will provide strong evidence for the validity of the conjecture (which we do not doubt) but, more importantly, will point the way for obtaining increased benefit from technologies already implemented and in use or waiting on the sidelines.

*To be continued and completed.*

## Organisational Involvement

Awaiting committments

## Personnel

Awaiting committments

## Action Plan

Awaiting planning activity committments

## Deliverables

Awaiting action plan

## Costs

Awaiting plan development and commitments and statements of need

## 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
- [boe76] Boehm B W, *Software Engineering*, IEEE Trans. on Comp., v. C-5, n. 12, Dec. 1976, pp. 1226-1241
- [boe88] Boehm B W, *A Spiral Model of Software Development and Enhancement*, Computer, v. 21., May 1988, pp. 61 72
- [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
- [bux70] Buxton J N and Randell B, *Software Engineering Techniques*, Report on a Conference, Sponsored by the NATO Science Committee, Rome, 1969, Scientific Affairs Division, NATO, Brussels 39, 1970, 170 p
- [gib94] Gibbs W W, *Software's Chronic Crisis*, Scientific American, Sept. 1994, pps. 72 - 80
- [leh69] Lehman MM, *The Software Process*, IBM Report no. RC 2722, Yorktown Heights, 1969, also in [leh85]
- [leh78] Lehman M M, *Laws of Program Evolution - Rules and Tools for Programming Management*, Proc. Infotech State of the Art Conf., Why Software Projects Fail, - Apr. 9 - 11 1978, pp. 11/1 - 25, also in [leh85]
- [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, also in [leh85]
- [leh85] Lehman MM and Belady L, *Program Evolution - Processes of Software Change*, Academic Press, 1985
- [leh87] Lehman M M. *Process Models, Process Programs, Programming Support - Invited Response To A Keynote Address By Lee Osterweil*, Proc. 9th Int. Conf. on Softw. Eng., Monterey, CA, 30 March 2 Apr. 1987, IEEE Comp. Soc. pub. n. 767, IEEE Cat. n. 87CH2432-3, pp. 14 - 16
- [leh89] Lehman M M, *Uncertainty in Computer Application and its Control Through the Engineering of Software*, J. of Software Maintenance: Research and Practice, v. 1, n. 1, Sept. 1989, pp. 3 - 27
- [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
- [leh94] Lehman MM, *Feedback, Evolution and Software Technology*, Preprints of the First International FEAST Workshop, Imperial College, June 1994
- [leh94b] Lehman MM, *Properties of S and E-Type Systems*, Preprints of the Second International FEAST Workshop, Imperial College, Oct. 1994
- [maj93] Major J, *Keynote Address*, ICSE15, Baltimore, 17 - 21 May 1993
- [nau69] Naur P and Randell B, *Software Engineering - Report on a Conference*, Sponsored by the NATO Science Committee, Garmisch, 1968, Scientific Affairs Division, NATO, Brussels 39, 1969, 231p
- [ost87] Osterweil L, *Software Processes are Software Too*, Proc. 9th Int. Conf. on Softw. Eng., Monterey, CA, 30 March - 2 Apr. 1987, IEEE Comp. Soc. Pub. n. 767, IEEE Cat. n. 87CH2432-3, pp. 2 - 13
- [per94] Perry D, Lehman M M and Turski W M, *FEAST Manifesto*, FEAST Technical Report, 1995. Request from Dewayne Perry electronically at dep@research.att.com
- [per94] Perry D and Lehman M M, *FEAST Model Parameters*, FEAST Technical Report, 1995, Request from Dewayne Perry electronically at dep@research.att.com
- [pot84] Potts C (ed), *Proceeding of the Software Process Workshop*, Egham, Surrey, UK, Feb. 1984. IEEE, cat. n. 84CH2044-6, Comp. Soc., Washington D.C., order n. 587, pp. 27 -35
- [ran94] Randell B, Ringland G and Wulf B, (eds.), *Software 2000 - A View of the Future, Output of a Forum Sponsored by ICL and the Commission of the European Communities*, 1994, D2D, Cavendish Rd., Stevenage, SG1 2DY, UK
- [roy70] Royce W W, *Managing the Development of Large Software Systems*, IEEE Wescon, August 1970
- [sen93] Senye P M, *The Fifth Discipline - The Art and Practice of the Learning Organisation*, Century Business, London, 1993
- [wei92] Weinberg G, *Quality Software Management*, Dorset House Publ., NY, NY, 1992
- [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