PROCESS MODELS, PROCESS PROGRAMS, PROGRAMMING SUPPORT

Response To An ICSE9 Keynote Address By Lee Osterweil

M M LEHMAN

Department of Computing, Imperial College, London SW7 2BH

One way of responding to a keynote speaker is to put the expressed views into context, pointing to highlights in the address, suggesting areas where alternative viewpoints might have been presented, exposing any chinks in the armour of the otherwise solid structure erected by the speaker.

Logistics have made it impossible for this respondent to see the paper to be presented to ICSE9 by Professor L Osterweil before generating his own written response,. The above approach cannot, therefore, be taken. Instead, I raise a fundamental issue that follows from a comparison of the respective approaches to process modelling taken by Osterweil and myself. What is expressed here reflects my current understanding of his views on Process Programs and Process Programming, my reaction to what I *believe* he *will* present. I can only hope that this will not do too much violence to views to be expressed in his Proceedings paper or in the Keynote lecture itself.

To set the scene and to provide a basis and framework for discussion, let me first summarize my view of studies of the software development process in terms of my own involvement in them.

To the best of my knowledge, the first such study was a 1956 paper by Benington [BEN56]. In this, a process model with basic characteristics of that subsequently termed the 'Waterfall Model', was first presented. Current interest in the software development process makes it most appropriate that this historic paper is to re-presented at this conference. In 1968/9, totally unaware of the earlier paper, I engaged in a study whose conclusions were presented in a confidential report entitled 'The Programming Process' [LEH69]. This has now become available in the open literature [LEH85, chapter 3] and is, I believe, as relevant today as at the time it was written. It was this study and the continuing research it triggered that subsequently led my colleagues and me to the concepts of process models, evolution dynamics, program evolution and support environments.

Our earliest process models reflected the dynamics of the process [LEH85, chs. 5-9, 14, 16, 19]. By the mid 70's, at about the time that Barry Boehm [BOE76] popularized the Waterfall model first proposed by Royce [ROY70], my studies had led to a search for better understanding of the total process of software development.

This total process was seen as extending from initial verbalization of the problem to be solved or computer application to be implemented, through delivery of the product and over its subsequent evolution. The search was expressed through the development and refinement of a sequence of Process Models [LEH85 chs. 3, 7, 14, 20, 21, 2]. It was directed towards first formulating a model of an *ideal* process ('ideal' though unachievable in the sense of the 'ideal' cycle of thermodynamics). Such a model would constitute a general paradigm. A practical process would be obtained by instantiation in terms of relevant concepts, available technologies, specific implementation environments, process constraints and so on. This development of process models culminated in the LST model [LEH84] and its subsequent analysis and application as presented at the first two Process Workshops [SPW84, 86]. The importance of that model is not only in the process it depicts. It *is* a canonical model of software development and of development steps.

What has all this to do with process programs? Process programs, as described by Osterweil, are also process models. They are models constructed from linguistic elements expressed and structured in programmatic form. They are intended to define a procedure for achieving some desired end from an initial starting point and are expressed in terms of expressions in a natural or formal language. The procedure is implemented by executing the primitive actions named in the program.

A process program to describe a process that, if followed, will permit execution of some specific task in its environment, can be systematically developed, top-down, in a manner equivalent to top-down development of a procedural program. The Osterweil approach is essentially equivalent, in the context of process modelling, to the use of procedural programming (in contrast to styles such as functional, imperative and so on). Its power is defined by the properties of the language used in relation to available execution mechanisms. In fact, a process program is precisely that - a procedural program whose value depends on the constructability of a mechanism that can execute it mechanically, human intervention being restricted primarily to the provision of information. This is a view that Osterweil will not dispute; in the papers that I have seen the algorithmic nature of process programs is repeatedly stressed.

And therein lies the rub. The approach is fine, almost certainly useful, when comprehensive models of the phenomenon, the domain and the system that are the subject of the program are known and understood, when strategies and algorithms for achieving the desired ends are known *a priori*, when computational, managerial and administrative practices are fully defined. It is useless, indeed meaningless, if such phenomenological and algorithmic models do not exist [TUR86], if progress in definition (and execution) of the process is a function of the process itself.

That this is so can be illustrated by considering the power and limitations of Integrated Support Environments (IPSEs). Underlying the IPSE concept is a simple observation. One must seek to mechanize those parts of the software development process that can be defined algorithmically. Machines cannot, however, be developed to execute activities for which an *a priori* constructive definition cannot be expressed, may not even exist. So people must undertake the creative portions of the process. If Osterweil were correct one should be able to take an IPSE such as ISTAR [DOW86], embed it in an appropriate harness, provide the information required by the process at various stages and then crank the machine to produce the products that implement the initial concept. Clearly, this is not generally possible. The problem is that every stage and step of the programming process requires thought, the application of heuristic (often creative) judgement, analysis, review of earlier steps, further refinement or backtracking to redo earlier models. Using descriptive natural language to express what cannot be understood except in the context of an actual execution, is not helpful.

In terms of our current understanding and of the known properties of the programming process [LEH84], process programs are more likely to divert attention from the real problems of software engineering than to help solve them. The very existence of a programming language sets up constraints as to how a problem may be solved, severely limits human creativity. As expressed so eloquently by Arthur Koestler in his 'Act of Creation' [KOE64]:

> *"The prejudices and impurities which have become incorporated into the verbal concepts of a given 'universe of discourse' cannot be undone by any amount of discourse within the frame of reference of that universe. The rules of the game, however absurd, cannot be altered by playing that game. Among all forms of mentation, verbal thinking is the most articulate, the most complex, and the most vulnerable to infectious diseases. It is liable to absorb whispered suggestions, and to incorporate them as hidden persuaders into the code. Language can become a screen which stands between the thinker and reality. This is the reason why true creativity often starts where language ends"* .

Expressions like '**code wrong: change_code**' or '**create_design**' in the body of a Process Program do nothing to clarify the *process*. They are either trivial, a stilted form of a natural language model or their meaning is undefined and their expression merely creates an *illusion* of progress. A disciplined process is vital as computers penetrate ever more into the operation of society. To the extent that process programs express such discipline and contribute to its achievement they represent progress. They do not, however, appear to provide a fundamental contribution to the further development of a software engineering discipline.

In application domains where Software Engineering know-how is substantially complete, in Turski's words 'artificial application domains such as presented by mathematical problems' [TUR86], successful process programs can certainly be constructed. Examples include compiler construction and the numerical solution of many classes of mathematical problems. But that is because these application classes and solutions to problems in their domains can be expressed in mathematical terms. In the early years, programs in these domains will largely have been developed on an *ad hoc* basis. Such development also led gradually to clearer understanding of the application domain and hence to potential for its formalisation. It is the problem domains (the problems and possible solutions) that become well understood and formally modelled not the process for program development in general. When the former is achieved any complexity lies, at worst, in the formal representations. The process of implementation is straightforward, well defined and expressible, for example, in program form; a process program in fact. This is why, in such instances, one may create a metaprogram, a meta-compiler for example, that implements the desired system given an appropriate specification. Such a meta-program is, in fact, an IPSE with driving harness, as hypothesized above.

For applications (commonly termed 'programing-in-the-large') which provide the real challenge for software engineering as distinct from programming methodology, models of the application as a whole or of many of its parts do not, in general, exist; there is no theory of program development, there is no global and formalisable development procedure, at best there is only an abstract process model [LEH84]. For this class of applications, process programming cannot provide potential for a major breakthrough. What is first required for each instance or class of applications, what is vital if the goal of reliable, timely and cost-effective development is to be achieved, is the development of formal phenomenological models and formal procedures for transformation of those models into executable programs. Process programs are not the correct approach to this goal. Nor will they assist in the development of an engineering discipline that will facilitate the ready and reliable creation of the appropriate systems and their subsequent evolution. An individual program will, at best, create an *impression* of complete understanding. But this must, inevitably, be incomplete in critical elements. Widespread pursuit of process programming would be a diversion, developing descriptions of those parts of the process that are well understood, covering up those aspects that represent the real challenge.

For achievement of real progress, models such as those referred to in the opening sections of this response are essential; models derived by analysis, models whose development yields clarification and understanding of the activity of software development and evolution. That is the challenge for the future.

In summary, process programming is meaningful in certain restricted areas. In these, Osterweil's work is significant. Moreover, his work must, unquestionably, be commended for its originality and neatness of presentation. Nevertheless, it must, be recognized that process programs have limited value in the context of improvement of the software development and evolution process. Whatever their value and domain of application, further pursuit does not satisfy the need for intensification of the direct study of the overall process of software engineering and for developing support for that process. Process programming does not represent progress towards the goal of making the world after the computer safe to live in, a goal demanding urgent attention.

References

[BEN56] Benington HD, 'Production of Large Computer Programs', Proc. Symp. on Advanced Computer Programs for Digital Computers' sponsored by ONR, June 1956. Republished in Annals of the History of Computing, Oct. 1983, pp. 350-361

[BOE76] Boehm BW, 'Software Engineering', IEEE Trans. on Comp., vol. C-5, no. 12, Dec. 1976, pp. 1226-1241

[DOW86] Dowson M, 'ISTAR - An Integrated Project Support Environment', Proc. of the 2nd ACM SIGSOFT/SIGPLAN Software Eng. Symp. on Practical Development Support Environments, ACM SIGPLAN Notices, vol. 2, no. 1, Jan. 1987

[KOE64] Koestler A, 'The Act of Creation', 1970 edition, Pan Books Ltd, London, pp. 176-177

[LEH69] Lehman MM, 'The Programming Process', IBM Res. Rep. RC 2722, IBM Res. Div., Yorktown Heights, NY 10495, Dec. 1969, 46p. Reprinted in [LEH85] as chap. 3

[LEH84] Lehman MM, Stenning V and Turski WM, 'Another Look at Software Design Methodology', ACM Software Eng. Notes, vol. 9, no. 2, Apr. 1984, pp. 38-53

[LEH85] Lehman MM and Belady LA, 'Program Evolution - Processes of Software Change', Academic Press 1985, 539p.

[ROY70] Royce WW, 'Managing the Development of Large Software Systems: Concepts and Techniques', Proc. WestCon., Aug. 1970

[SPW84] Potts C (ed), 'Proceeding of the Software Process Workshop', Egham, Surrey, U.K., Feb. 1984. Publ. IEEE, cat. no. 84CH2044-6 Comp. Soc., Washington D.C., order no. 587, 175p.

[SPW86] Wileden JC and Dowson M (eds), Software Eng. Notes Special Issue on the International Workshop on the Software Process and Software Environments, Coto de Caza, Cal., 27-29 March 1985, vol. 11, no. 4, Aug. 1986, 74p

[TUR86] Turski WM 'And No Philosopher's Stone, Either', Information Processing 86, Proc. of the IFIP World Computer Congress, Dublin, 1986, Publ. by North-Holland, 1986, pp. 1078-1080

mml409
[liz106-response-lo]
26 January 1987