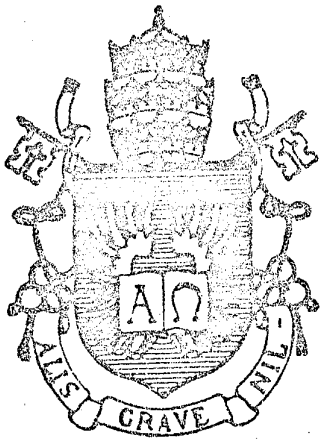

PUC

Series: Monografias em Ciência da Computação,
No.26/89

THE RIO WORKSHOP ON THE SOFTWARE PROCESS

Júlio C. S. P. Leite
Carlos J. P. Lucena

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 – CEP-22453
RIO DE JANEIRO – BRASIL

PUC/RJ - DEPARTAMENTO DE INFORMÁTICA

Série: Monografias em Ciência da Computação, Nº 26/89

Editor: Paulo Augusto Silva Veloso September, 1989

THE RIO WORKSHOP ON THE SOFTWARE PROCESS

Júlio C. S. P. Leite
Carlos J. P. Lucena

This work has been partially sponsored by FINEP

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC RIO, Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453 - Rio de Janeiro, RJ
BRASIL

Tel.: (021) 529-9386 TELEX: 31078 FAX: (021) 274-4546
BITNET: userrtlc@lncc.bitnet

The Rio Workshop on the Software Process Rio de Janeiro, Dec. 14–15 1988

Julio Cesar Sampaio do Prado Leite
Carlos José Pereira de Lucena
Departamento de Informática
Pontificia Universidade Catolica
do Rio de Janeiro

July 1989

Abstract

The Departamento de Informática of the Pontifícia Universidade Católica do Rio de Janeiro and the Instituto de Engenharia de Software of IBM Brasil organized the Rio Workshop on the Software Process, to promote a discussion about the different views of the software development process by researchers in the area. These discussions were anchored on two presentations about different approaches to the software development process. One approach came from the "formal school", represented by Dr. Thomas Maibaum of the Imperial College of Science and Technology, and the other from the "knowledge based school", represented by Dr. Allen Goldberg of the Kestrel Institute. Several researchers, with a formal background, as well as others with a more praticioner-oriented experience were present at the workshop. The questions and discussions that emerged during the two-day meeting reflect important pratical and theoretical points for the development of the software area.

1 Introduction

Software engineering emerged as a discipline in the late sixties as a response to the problems involved in developing large scale software systems. One of the first and very popular approaches to the problem has been the use of a biological growth model, mapped into what has been called the software development life cycle model [Royce 70], to explain the software development process. This approach has as one of its main characteristics the clear divison of development phases and the concept of certification of the product developed at the end of the cycle (validation taking place between products of sucessive phases).

After being in use and being discussed for some time [Kerola 81], the now classical software life cycle model started to be challenged by several researchers [Agresti, 86]. Most of the criticism directed at the traditional life cycle model have addressed its lack of flexibility and lack of formalization. Recently, initial discussions on the formalisation and automation of the software development process, [Lehman 87] and [Osterweil 87] are taking place. Different paradigms and different views of software development process are being proposed. Several research problems do still exist before the full or even partial automation of the software development process can be achieved. The workshop held at Rio de Janeiro pointed out some of those problems and some possible ways of going about solving them.

Originally designed to be a confrontation of views, the workshop ended up illustrating how a consensus is emerging with resepct to very key aspects of software development problem. The formal school acknowledges the need for using heuristics and the knowledge based school understands that formalization is a necessary next step to assure futher developments through this approach.

In the next sections we will give an overview of the presentations by Dr Maibaum and Dr Goldberg at the workshop, followed by a summary of the important points discussed. A full report is available from *Departamento de Informática – PUC* or from the *Instituto de Engenharia de Sofware – IBM Brasil* [Leite 89].

2 Maibaum's Presentation

Maibaum's presentation stressed that it is fundamental to understand the mathematical character of the idealizations that are present during program development. Through this understanding it will be possible to have a scientific basis for developing methods and tools for supporting the program development process. The idealizations are seen, basically, as a theoretic/symbolic activity, very similar to a mathematical proof. The four main components

of the idealization process are:

- Requirements,
- Architectural Design,
- Implementation, and
- Specification.

Requirements are understood as theory/hypothesis formation with experimental validation. Architectural design is the building of a high level description of the intended system. Implementation is the act of "simulating" one theory by another. Specification is a theory presentation in a chosen logic.

The main problems associated with hyphotesis formation or with requirements definition are the problems of identifying which logic to use, which language (extralogical language) to use and what axioms to use.

The presentation of the hypothesis (or the theory) is done via a specification. Using this representation (specification) it is possible, not only, to present the theory, but also to implement this specification in terms of another specification. This translation between theories (S(1) —> S(2)) is required to preserve the properties of S(1) in the new presentation S2. Maibaum points out that this translation is really a interpretation between theories [Maibaum 86]. In this framework, the properties of conservative extension and modularisation are important. An extension is conservative when the new theory adds no theorems to the old one. Modularization is present when it is possible to derive, algorithmically, an extension from S(1) to S(3) when extensions from S(1) to S(2) and S(2) to S(3) are knonw. This property is fundamental for structuring specifications and for implementing specifications in terms of another specificaton.

8 Goldeberg's Presentation

Goldberg's presentation focused on the experience that has been gained at the Kestrel Institute in using a knowledge intensive approach for the development of software. The main concerns of this approach are: rapid prototyping, maintenance at the specification level, maintenance based on a formal record of the design history, domain-specific knowledge, faster development cycle, and correctness by construction. The central theme of the work performed at Kestrel is transforming a formal specification into a correct and efficient code.

Notable progress has been made at Kestrel, most of it related to the engineering aspect of implementing the ideas put forward by Green and others [Balzer 83]. A language and a transformation system were defined and developed [Smith 85] and development environment was recently developed (KIDS).

Golderg gave details of one of the projects under way at Kestrel: the KBSA Performance Assistant. It's main goal is to construct an interactive development system that uses performance analysis to design efficient implementations. Different experiments have been performed, using well know programming problems, like: insertion sort, 8-queens, and topological sort, among others.

The performance assistant is basically composed of a transformer and a performance/ type analyzer. Using algorithm design tactics present in KIDS and optimizations encoded in the transformer, it has been possible in the case of the job schedule problem to achieve a linear implementation departing from prototyped specification that had, initially, an exponential complexity. Examples of optimization techniques avaialble and their degree of automation are shown below.

- Finite differencing (user identifies expression).
- Iterator inversion (user identifies expression).
- Canonicalization (automatic).
- Operation refinement (interaction in exceptional cases).
- Optimize membership test (automatic)
- Loop fusion (greedy algorithm gives near optimal results).
- Useless code elimination (automatic).
- Translation of For construct (automatic).
- Data structure selection (user annotations).

4 Important Observations

The discussions that took place during the workshop were based in Golderbg's and Malbaum's presentations and by a set of questions posed by Lucena. The questions were focused on the

role and the importance of the use of formalims for the characterization of software process and the potentiality of applying knowledge based techniques for the development of software systems. The questions were used by two working groups that after listening to the invited speakers' presentations, discussed the issues proposed and prepared a short report that was later presented by each group's chair. Following the presentation of each working group there was a session of open discussions, followed by a short conclusion statement given by each of the invited speakers.

The organization of the workshop [Leite 89], briefly sketched above, encouraged participation and an objective discussion of the topics proposed. Because of the diverse backgrounds and experiences of the participants, the workshop was able to cover many very important topics that should interest to the software engineering research community. In what follows we highlight what we consider the most important observations and issues dealt with at the workshop.

Science and Engineering The expected radical different views about the software development process did not really show up in the presentations of the two invited guests. Clearly, Maibaum's presentation reflects a theoretician's perspective while Goldberg's reflects an engineering approach to the problem. An interesting point, however, is that their views as stated are complementary.

The distiction drawn science and engineering led to two important observations. First, it is clear that software engineering needs to rely on a theoretical body of knowledge, in such a way that methods and tools (the engineering itself) can be fully developed. Second, it has been observed that computer science has experienced several situations in which theory became technology, the classical example being the area of compiler construction.

The establishment of a science of software requires precision. Precision will not come without the use of formalisms. Although this was a general consensus, various approaches to formalization were proposed. Although precision is required, there are parts of the software development process which are inherently fuzzy. The scientific aspect of software has to acknowledge it and the engineering part has to handle it. A possibility for handling these fuzzy aspects is the use of the idea of heuristics, which has been engineered with success in the area of Artificial Intelligence.

Aspects of Formalization As the discussions developed, it was observed that in talking about the formalization of the software process, it is essential that the objects dealt with by this process be also formalized. This apparentely obvious fact is nonetheless sometimes

forgotten in the rush to nail down the nature of the process of software construction. Veloso noted that, although there are no adequate formal models of the software process, descriptive models [Lehman 87] and prescritive [Balzer 83] models have been proposed.

It was a consensus that one of the roadblocks to formalization is education and better methods and tools. It was clear that the level of knowledge and mathematical trainning required to formally deal with software development needs to be increased. Maibaum noted the lack of methods for applying formalizations, most of what is available are representation schemes but not methods.

The overall opinion is that the best strategy for introducing formalism is to formalize semi-formal methods already experienced in practice. The strategy should also apply to Software Development Environments, which will evolve momentarily despite formalization, later formalisation will influence the development of new SDEs or the revison of existing ones. Von Staa noted that the introduction of formalisms follows an evoluationary approach: they are proposed, used, evaluated and revised.

A side effect of formalisation has been identified as a straight jacket; some believed it to be unavoidable, while others thought that it is necessary to enforce a discipline.

Specifications Size and Definiton Problems were raised about the definition of specification. Maffeo stressed the importance of having the specification free of implementation considerations. Maibaum believes a specification is an implementation of a more abstract specification. As usual, no conclusion was reached.

Von Staa pointed out that there is not a way for evaluating formalisms, that is, it can not be said that a formalism will work or will do better than another one. It was general consensus was that a specification should use multiple formalisms. Another consensus had to do with the size of specification. We agreed that specification size should not be considered a complexity measure.

Although the specification size is not a complexity measure, it was noticed that the use of domain knowledge allows the specification to be shorter. The capture and usage of domain knowledge will require that domains be very specialised.

Limits of the Knowledge Based Approach One of the questions posed for discussion asked about the extent to which the knowledge based approach could be successfully applied to programming in the large. Most of the participants agreed that the problem was different from programming in the small. Mendes is skeptical about it, arguing that not only do we have the difficulty of encoding knowledge but that software engineering is too young a

discipline to have real experts. Lucena made the remark that the experience reported about the use of software engineering expert systems is not very stimulating.

Veloso believes that for architectural design there is a need for something extra besides transformation rules. Maibaum believes that the transformation rules should apply, but the nature of the transformations are likely to be somewhat different.

Golderg believes that if you consider a specification to be complete, then there will be no essential difference from programming in the small, but lots of more effort should be necessary in the optimization.

There were remarks by Maibaum, Goldberg and Leite about the importance of using domain knowledge. Maibaum believes that what is holding us back from applying the transformation and knowledge based approach is the lack of domain knowledge. Goldberg stressed the point that domain knowledge is what makes specifications big, he believes that ADTs like schemes should encode domain language. Leite mentioned the Draco paradigm towards software development, where a network of domain languages is the keystone to developing software through reuse of requirements analysis.

Maibaum stressed the importance of the clear characterization of the objects in the software process, in order to make it possible the use of meta knowledge about these objects. This meta knowledge is necessary to drive the implementation of an specification. As a starting point in this area, the work of Sintzoff [Sintzoff 85] was cited.

Reusability was addressed and there was an aggreement that the notion of components is a fundamental one. The notion of reusing domains, as a result of the process of domain analysis, and reusability of design strategies (meta knowledge) were pointed out as important fields for research.

Participants Ana M. Moura, Ana M. Testoloni, Ana R. C. Rocha, Antonio C. Lirani, Arndt von Staa, Bruno Maffeo, Carlos J. P. Lucena, Eduardo T. Takahashi, Fabio N. Akhas, Fernando Manso, Geovane C. Magalhães, Gernot Ritcher, Ismael C. Ramos, Julio C.S.P. Leite, Jano M. de Sousa, Paulo Veloso, Sergio E. R. Carvalho, Sheila Veloso, Sueli B. T. Mendes, Tarcisio Pequeno, Miguel Jonathan, Antonio C. Lirani.

References

[Agresti 86] Agresti, W.: *New Paradigms for Software Development.* W. Agresti, Ed., IEEE Computer Society, Long Beach, CA. 1986.

[Balzer 83] Balzer, R., Cheatham T., Green, G.; Software Technology in the 1990's: Using a New Paradigm. *IEEE Computer*, Nov. 1983, pp 39-45.

[Kerola 81] Kerola, P. and Freeman, P.; A Comparison of Lifecycle Models. *Proceedings of the 5th International Conference on Software Engineering*, San Diego, 1981, pp.90-99.

[Lehman 87] Lehman, M.M.; Process Models, Process Programs, Programming Support. *Proceedings of the 9th International Conference on Software Engineering*, IEEE, 1987, pp. 14-16.

[Leite 89] Leite, J.C.S.P.; The Complete Report on the Rio Workshop on the Software Process. *Departamento de Informática da PUC/RJ*, Ago. 1989.

[Maibaum 86] Maibaum, T. The Role of Abstraction in Program Development, IFIP 86, North-Holland, 1986.

[Neighbors 84] Neighbors, J.; The Draco Approach to Constructing Software from Reusable Components. *IEEE Trans. on Software Engineering*, SE-10 (Sep. 1984), 564-573.

[Osterweil 87] Osterweil, L. Software Processes Are Software Too. *Proceedings of the 9th International Conference on Software Engineering*, IEEE, 1987 pp. 2-12.

[Royce 70] Royce, W.; Managing the Development of Large Software Systems: Concepts and Techniques, *Proceedings of the Westcon*, 1970.

[Sintzoff 85] Sintzoff, M., Desiderata for a Design Calculus, RR 85-13, *Unite d'Informatique of Louvain*, Sep. 1985.

[Smith 85] Smith, D.,Kotik, G., Westfold, S.;Research on Knowledge-Based Software Environments at Kestrel Institute. *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 11, Nov. 1985, pp 1278-1295.