Towards Information Systems for Software Producers

Aarne H. Ylä-Rotiala

Nokia Telecommunications
P.O. Box 33
FIN-02601 Espoo, FINLAND
Tel: (358) 0-511 6542
Email: aarne.yla-rotiala@ntc.nokia.com

Abstract

Software producers need a defined information base to operate on. A software production process cannot be properly defined without giving meaning to the products and raw materials of the process. The information flow is the essence of a software process, and therefore the need for well-defined software information models is evident.

A realistic software engineering information system that takes into account the relations between input and output information would be a powerful aid in software production. Software processes have been much discussed, but the attention to the information in these processes has not been adequate. A process cannot exist without an information system, implicit or explicit. If the software information is explicitly modeled, a better process with enchanced reuse and reusability will follow.

Keywords: Reuse, information system, software engineering environments, software production process.

Workshop Goals: Communication, exchange of ideas and new idea creation.

Working Groups: Reuse process models, Reuse maturity models, Tools and environments, Reuse management and economics

I would like to propose something along the lines of "Reuse in the software life cycle" or (perhaps better formulation) "How to fit reuse in actual software production". I hope this is what the "Reuse process models" working group is about.

1 Background

My background is mostly practical. How to write less code while making more money is a question that has intrigued me for a long time. I have worked as a programmer and designer (as a "software engineer") for five years, during which time I have tried to find and build reusable components. My industrial experiences include working in projects of differing size, and these experiences motivated me to begin studies aiming to a Ph.D, with a subject relating to reuse. My subject follows quite closely my position below.

2 Position

Software producers have been struggling with the problems of productivity, quality and reliability for some time now. As early as 1969 the need for mass-production was recognized. The term "software factory" has been used on several occasions. Analogies have been sought from other engineering disciplines: software reliability modeling has its origins in hardware reliability, building software is often compared to building bridges or houses, and the very term "software reuse" was borrowed from manufacturing. I claim that the traditional analogies all more or less fail in describing the process of software production and that the last of them - software reuse - is one of the worst of them. I do not claim that these analogies are worthless, I merely hold the position that in order to discuss the software production processes, their difficulties and the peculiarities of enhancing the productivity of software production one should consider the essence of software - its intangibility, zero-cost replicability, modifiability - and the actual process that creates it. Software reuse will not just happen, but it is more likely to happen if Joe Programmer's reality is taken into consideration.

Software production equals to electronic document manipulation. When one starts from the scratch, one has no previous documents to use. If, however, there is some pre-existing material to be used, and it is used, we usually claim that "reuse"'has taken place. [1] presents the idea of a maintenance process that would be based on reuse. The idea can be taken further - the whole software production could be viewed similarly. When a software house starts, its information base is usually quite small. If this information base is considered a strategic asset and carefully nurtured, the likelihood of its usefulness will grow. The benefits for a company cannot be predicted, but I would expect better producticity and larger revenues.

In the maintenance-as-reuse -paradigm there is an information system, which is available to the software engineer. The engineer has a problem description, and (s)he is to produce a set of other documents, including the program code that will solve the stated program. If the information system is to have any real value to the engineer, it should give direct support to the tasks that face him/her. The engineer has a need to find relevant pieces of information, depending on the task at hand and the dependencies and interconnections of these pieces to find out old, proven solutions. If and when these old solutions are found, the engineer can use (or "reuse" them either by modifying them or connecting them in a novel fashion. This application of old data to a new problem is what lies in the heart of reuse and is an essencial component in software production. Very seldom does one start without a single document and build everything from scratch, and if one does, the competitiveness of the resulting product is likely to be far from optimal.

The Capability Maturity Model [2] is quite concerned with processes and improving them. The repeated, defined and continuous measurement is stressed as a necessary requirement for anyone wishing to gain better performance in software production. This approach gives smaller attention to the actual process that is used in the production and virtually none to the information that is used in the process. The structure of the model itself is, however, a good example of a generic definition of an information process - the steps to take to reach level 5 just don't lead into building software, they lead into a software process. The same approach could be used when designing a software engineering environment - the required data should be made available to the engineer.

If one is to talk about software productivity and software reuse, it would be beneficial to consider the way software is currently produced - to build a model of the process that is used to create software. Naturally, the number of different processes is not known, at least not to me, but one thing remains as an invariant: software producers use tools to search, modify and connect existing documents, which they use to produce a set of new, previously unseen documents. These documents are usually in a human-understandable form and can be stored. The essential parts of a SW production process are the raw-materials (the existing documents), the tools (e.g. a compiler, or an editor) and the results (the new documents).

In manufacturing, the use of older products is true re-use, but the affix "re" somehow seems to lose meaning with software, which does not need to be reused and indeed cannot be reused. A more proper analogy would be an accountant, accounting software and an account: the accountant uses a tool (software) to handle an account (a reusable asset) to modify the account (make deposits or withdrawals). The accountant might even be able to take an existing account and use it as a template in order to create a new one, with the help of the given tool. Likewise, the software engineer uses available information, usually with the help of tools, and produces new information. The process can have, and should have, a simple description, which can naturally have an infinite number of refinements and slightly differing instantiations.

An explicit information model for software engineering does not exist, but it should. Software engineers have several lower-level methods for doing their work - formal methods, object-orientation, prototyping and cleanroom software engineering are good examples of these task-oriented tools. These tools help the engineer in the task at hand, but they can not provide an environment for the whole development cycle of the software product, from the customer contact to the product delivery, or from the first representation of a product idea to the withdrawal of the product from the market. The task-specific methods should not be disturbed - if certain analysis, design, implementation etc. methods are familiar, they should naturally be used. Each of these stages should be performed in a proper information context, not with chaotic document distribution, duplication and modification. Many a coder finds a version control system useful, and there are people who consider CASE tools as enhancements to the work. In a similar fashion, a larger information system could help the work an average engineer.

The process I have described in the preceding passages is the cornerstone of my position towards software reuse. I do not see "reuse" or "reuse processes" interesting as such. What interests me are the information models actually used in software producing organizations. A data model, an information system and a process which takes all these into account is what I have in mind. A proper information system, which is easy and natural for software engineers, or even a single software engineer, will result in a boost in productivity. Stated this way, the problem may sound like a technical one, but it isn't: for an organization larger than just a few people, a consensus should be reached and the whole organization should agree on the system (though not instantaneously). Also, the basic model of information processing should be recognized. These are management problems, which may be easier to solve if the technical problems like what data to store, where and how and how to search, retrieve and reference the stored data and by which tools, have excellent and obvious solutions.

3 Comparison with other work

I'll explain here the differences and similarities between my position and PCTE[3], CMM[2] and ISO-9000, one by one, and then conclude that my opinions are very similar to those of [4] and [5]. Especially the work on Software Engineering Environments, Integrated Project Support Environments [6] and Software Repositories [7] are very close to what I am talking about.

PCTE is a standard that describes a way to share software engineering data. It has a very powerful and abstract data modeling device and a well-defined interface. The difference between PCTE and my position is that PCTE is a meta-meta-model - and is actually on a different level. PCTE is a tool to discuss an information system, not an information system as such. What is interesting with PCTE (and similar efforts, e.g. CAIS and CDIF) is that there seem to be common standards for the information systems before there is any kind of an agreement upon the contents of these systems. This peculiar order of definitions shows itself in the contents of PCTE (and others): anything can be modelled within the limits of these standards.

CMM talks about process, but almost nothing else. Process is an independent entity, and I find this questionable, since - as we all should know - a process with no defined inputs and outputs is a pretty useless one. If we take the analogy into a program, CMM's approach would be like talking about a program's efficiency and overall quality just by looking at the code and at the internal functioning of the program. What seems to be missing is the link to the outer world, the inputs and the outputs.

ISO-9000 is a general standard concerning the quality of a firm. ISO-9000 is concerned with the formal, repeatable and documented quality assurance of an organization. It gives quidelines for things like audits, formal approvals and well-defined responsibilit!es and authorities during the production. Little, if anything is said about actual production, about the stages of a process where the possible errors are being made. Some consultants claim that following ISO-9000 results in improved productivity in the field of software production, which claims may or may not be true. However, ISO-9000 is only a general framework into which an information system could be fitted. So, ISO-9000 is interesting as a starting point or as an organizational context.

Brown's text about the nature of a software engineering database is a good one and, in my opinion, a correct one. The work of e.g. Jarke et al. [4] and Rombach [5] has the same direction that I have in mind. Software producers should develop their data models and build their information systems and processes accordingly. The routine work done by engineers should increasingly happen with the information base. The difficulty of this task is described in [8], but the task should not be an impossible one.

References

[1] V. Basili, "Viewing Maintenance as Reuse-Oriented Software Development," *IEEE Software*, pp. 19–25, January 1990.

[2] M. Paulk, B. Curtis, and C. et al., "Capability Maturity Model for Software," Tech. Rep. CMU/SEI-91-TR-24, Software Engineering Institute/Carnegie Mellon University, Pittsburgh, Pennsylvania, August 1991.

[3] ECMA, "Portable Common Tool Environment (PCTE) Abstract Specification," Tech. Rep. ECMA-149, European Computer Manufacturers Association (ECMA), 1990.

[4] M. Jarke, J. Mylopoulos, J. Schmidt, and Y. Vassiliou, "Information Systems Development as Knowledge Engineering: A Review of the DAIDA Project," Tech. Rep. MIP-9010, University of Passau, Passau, Germany, 1990.

[5] H. Rombach, "A Specification Framework for SW Processes: Formal Specification and Derivation of Information Base Requirements," in *Proceedings of the 4th Intn'l Software Process Workshop*, pp. 142–147, ACM, 1988.

[6] A. Brown, *Database Support for Software Engineering.* Kogan Page, 1989.

[7] J. Mylopoulos and T. Rose, "Tutorial on Software Repositories," in *15th International Conference on Software Engineering*, IEEE, 1993.

[8] T. Biggerstaff, C. Ellis, F. Halasz, and C. Kellogg, "Information Management Challenges in the Software Design Process," Tech. Rep. STP-039-87, MCC, Austin, Texas, 1987.

4 Biography

Aarne H. Ylä-Rotiala. I received an M.Sc in CS from the University of Helsinki in 1990. I have worked as a programmer, both as an employee and as an independent contractor for five years. I have participated in several commercial projects, as a subcontractor or as a principal contractor. I have started my Ph.D studies, that were inspired by these practical experiences. Currently I am working for Nokia Telecommunications' Software Engineering Methodology Development.

5 Acknowledgements

Many thanks to my wife Tiina for proofreading this text.