Software Reuse - it's TEA*– time!

Elke Hochmüller

Institut für Informatik
Universität Klagenfurt
Universitätsstr. 65–67
A–9020 Klagenfurt, AUSTRIA
Tel: ++43 463 2700 574
Email: elke@ifi.uni-klu.ac.at

Abstract

Using CASE environments and utilizing software reuse are seen as promising and mandatory means to enhance software productivity. Unfortunately, with current technology, the combination of software reuse and CASE environments is hardly supported.

On the basis of this observation, this position paper presents some criteria which can be useful in evaluation and assessment of tools and environments pretending to support software reuse.

Keywords: tool evaluation, reuse process model, institutionalizing reuse

Workshop Goals: Networking and exchanging ideas; common problem understanding; learning about reuse experiences in practice.

Working Groups: tools and environments, reuse process models, reuse management, organization and economics.

*TEA – Tool Evaluation and Assessment

1 Background

Elke Hochmüller is assistant professor at the University of Klagenfurt. She is member of Roland Mittermeir's research group which is particularly concerned with software reuse. Based on the software base approach, she developed the concept for *AUGUSTA* - a reuse-oriented software engineering environment for the development of Ada applications [Hoc95]. Furthermore, her Ph.D. thesis [Hoc92] contained some initial thoughts on ideal process models as well as organizational structures of reuse-oriented software developing companies.

In [HM95] E. Hochmüller and R. Mittermeir propose a reuse augmentation to traditional process models requiring only minimal changes to an already implemented and approved software development process (Figure 1). This reuse process model can be considered as a meta model which can be refined according to the particular needs of each enterprise. Furthermore, it is rather independent of the software development process model applied so far. Thus, a software developing enterprise need not change all its approved methods and processes, but only enrich its applied development process together with some inevitable changes in its organizational structure (e.g. software library, software library administrator). The suggested reuse process model consists of two main subprocesses which are linked together in a "ping-pong" manner. The first subprocess deals with the development of reusable components according to development for reuse (*DfR*). These compoents are used within the second subprocess which deals with (system) development with reuse (*DwR*).

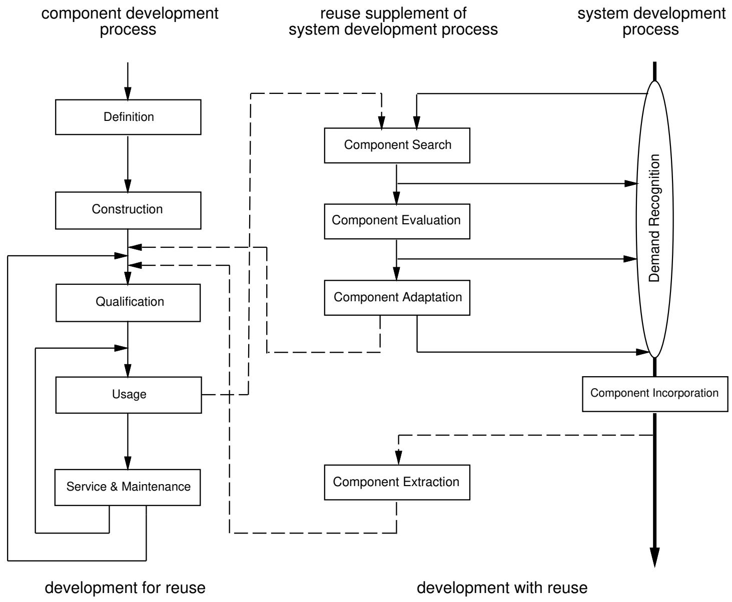

Figure 1: Reuse Process Model

2 Position

Software reuse cannot be institutionalized by a very radical change in traditional software development. The integration of software reuse into current organizational as well as process structures should take place smoothly. On the one hand, already approved software process models should not be thrown away but be enriched by planned software reuse. On the other hand, this should also be true for already well known and applied CASE tools. Thus, CASE tool vendors should focus on reuse requirements and adopt their tools accordingly. Hence, without demanding for completeness the following criteria (mainly resulting from the reuse process model previously presented) are proposed to be considered while evaluating tools regarding to their support of reuse.

1. *Component Reuse*

   Most CASE tools implement a phase driven, forward looking development model. Software development with reuse, however, is all but that linear. It follows rather a "Yoyo"-approach, where design sketches are followed by delving into the library of reusable components, evaluating some candidate components and then solidifying the high level sketches according to the already available integrable components. On the other hand, one should also consider development for reuse. This has many facets. Be it, that the development of highly generic, off-the-shelf components is promoted, or be it that the isolation of reusable components out of conventionally developed applications is supported.

2. *Central Software Library*

   A central software library containing software components of high reuse potential is an essential part of a reuse supporting CASE environment. While a common repository, which is a prerequisite for any integrated software engineering environment, will contain anything which has been produced during a particular development project, this software library should only contain components of approved quality with high potential for interproject reuse.

3. *Software Library Administrator*

   The CASE environment should support a new user role for the administration of the software library. This task could be fulfilled by a software library administrator whose competence, however, should exceed pure administrative activities. This person should particularly be responsible for the certification, classification and maintenance of software components. It is expected from a CASE tool to support the software librarian in the relevant activities.

4. *Component Certification*

   The inclusion of every software document into the software library without any checks would soon flood the library with components of little potential for reuse. Frustration of users of this library will be the consequence. To cope with this problem, a reuse library should provide clues on qualitative properties of components. The process of component certification could be a single-step activity or a rather sophisticated process depending on the specific situation of the development organization. In any case though, the certification of the candidate components should be supported by the CASE environment.

5. *Retrieval Support and Classification Scheme*

   It is required from a reuse supporting CASE environment to provide best automation of both storage and retrieval of components. In order to find specific components contained in the software library quickly and with acceptable effort, the software documents should be classified when they are stored. Therefore, an adequate classification structure is necessary.

6. *Adaptation Support*

   In case a software component which is only similar to the desired one could be identified during a search process, a CASE tool is expected to support the process of adaptation of such a component in order to fulfil precisely the specification demanded. While these modifications might encompass functional aspects as well as qualitative aspects, one might also consider those modifications, where the internals of a component would be adequate, but modifications have to be made to the interface in order to integrate it into the application under development.

7. *Testbeds and Instrumentation*

   Since component descriptions, how formal they ever might be, can naturally not cover all aspects which might be decisive for using a component in a new development project, a reuse supporting CASE tool should provide for testbeds, such that retrieved candidate components can be easily tested in isolation. In special cases, instrumentation of components for testing specific aspects might be desirable.

8. *Integration Support*

   After successful retrieval and adaptation of software components, they have to be glued together in order to achieve a complete software system. This integration of components should also be supported by the environment. With this integration support, one has usually to distinguish between prototype development and the more stringent requirements for the integration of production-quality versions.

9. *Maintenance Propagation*

   The requirement of maintenance propagation can be viewed from two perspectives:

   - *Vertical Maintenance Propagation*
     During its development process a software component passes different stages of representation (from specification document to object code). In addition to the need of storing all these different documents, a reuse supporting CASE tool should provide for automatic update of all the different representations if a particular component is changed.
   - *Horizontal Maintenance Propagation*
     A reuse supporting CASE environment should also provide the possibility to establish links between different software components of usually also different kinds of representation. These links should not only support the user in navigating between related components (which would again be another general requirement for CASE tools) but also enable the propagation of operations applied to a particular software component triggering adequate operations on related components.

10. *Reuse Success Evaluation*

    Continuous control of the reuse success can help not only the software library administrator to react in time to reorganize the contents of the library. It also assists potential users, which might build their estimates on the benefit of reusing a particular component on the past "turnover" of this component. Furthermore, management might use such statistics to reward project personnel for successful reuse or to encourage better reuse by establishing specific reuse incentives. Most necessary data can be easily collected and made available by the tool itself.

3 Comparison

A whole bunch of general requirements for CASE tools and environments has been proposed in the literature (e.g. [duP93]). Full life cycle support, integrated CASE, multiuser and teamwork support, consistency and integrity support, evaluation support, versioning, and management support are only some of the most important criteria in this connection.

However, as far as I know there have been no proposals covering specific criteria for CASE environments which focus particularly on reuse aspects in software engineering. The items presented here are far from being complete or detailed enough but are an excellent starting point for further research in this area. Regarding the nature of these requirements, they rather refer to entire CASE environments in the sense of [Bro94] than to single CASE tools. Furthermore, the scope of the evaluation criteria presented here lies beyond that of evaluation requirements tailored to reuse-specific tools (e.g. the evaluation of SoftKin [His94] as a reuse supporting tool detecting similarities at code level).

References

[Bro94] A. W. Brown. Why evaluating case environments is different from evaluating case tools. In E. Nahouraii, editor, *Proc. Third Symposium on Assessment of Quality Software Development Tools*, pages 4–13, Washington, DC, June 1994. IEEE.

[duP93] A.L. duPlessis. A method for case tool evaluation. *Information & Management*, 25(2):93–102, August 1993.

[His94] G. W. Hislop. Evaluating a software reuse tool. In E. Nahouraii, editor, *Proc. Third Symposium on Assessment of Quality Software Development Tools*, pages 184–190, Washington, DC, June 1994. IEEE.

[HM95] E. Hochmüller and R.T. Mittermeir. Improving the software development process by planned software reuse. In J. Györkös, M. Krisper, and H. C. Mayr, editors, *Proc. ReTechnologies for Information Systems (ReTIS '95)*, pages 27–39, Bled, Slovenia, June 1995. OCG.

[Hoc92] E. Hochmüller. *AUGUSTA – eine reuse-orientierte Software-Entwicklungsumgebung zur Erstellung von Ada-Applikationen.* PhD thesis, Universität Wien, May 1992.

[Hoc95] E. Hochmüller. Process support for software reuse with AUGUSTA. In *Proc. Software Systems in Engineering*, pages 173–181, Houston, January 1995. ASME.

4 Biography

Elke Hochmüller is assistant professor at the Department of Informatics, University of Klagenfurt, Austria. She received her M.S. and Ph.D. degrees from the University of Vienna in 1988 and 1992, respectively. In 1994 she left the University of Klagenfurt for half a year and developed an enterprise wide data model for an electric power supply company where she became aware of the very intricate managerial, social as well as psychological problems in practice which increased her interest in bridging the gap between research and practice. Her research interests cover various topics in the field of software engineering, particularly software reuse, process modelling, CASE, and requirements engineerig.