Integrating Reuse Into A Software Curriculum

Trudy Levine

Fairleigh Dickinson University
1000 River Road
Teaneck, NJ 07666
Tel: (201) 692-2020 (2261)
Email: levine@sun490.fdu.edu

Abstract

At the 5th International Conference on Software Engineering and Knowledge Engineering (SEKE'93), a workshop was held on June 18th for the purpose of studying the integration of reuse into software education. We noted the dearth of software reuse in software engineering programs and textbooks. Topics were established for further discussion and evaluation.

Keywords: software reuse, software education, reuse education

Workshop Goals: Learning; networking; advancing the state of software education and curriculum with software reuse

Working Groups: reuse education, etc.

1 Background

1.1 Integrating Reuse Into A Software Curriculum

Consider the following two major questions:

- How can software reuse improve software education?
- How can the study of software reuse be carried into industry?

Software reuse promises to aid in the production of reliable and cost efficient systems. As such, it should surely be part of a software curriculum. Even more important, perhaps, is the fact that a standardized and controlled method of studying reuse, similar to the study of expert work in the social sciences, can be a valuable tool in computer science education.

For the purpose of software education, software reuse should include reuse of code, specifications, and various types of documentation. In addition, students should be taught and encouraged to reference previously created systems and to create new software systems at least partially from previously developed components. This material should be able to be integrated into existing software engineering and computer science curriculum without major changes. In addition, the construction of previously developed components should be available for study, to aid in the development of techniques and talent.

As agreed elsewhere in the conference, as software systems grow in size and complexity there are increasing occurrences of cost overruns, time overruns, and system failures. Indeed, as more and more critical functions are controlled by software systems, the dangers of system failures are rather frightening.

Software reuse can aid system reliability in that integrated components will probably be better understood, crafted, and tested in the process of designing for reuse. There also can be considerable cost and time savings if components can be easily located and integrated. That could lead to greater reliability as well, because it is the time crunch that frequently causes developers to take short cuts in designing and testing. Of course, we do not think reuse is a software bullet, nor do we think that it can be achieved easily. We recognize the problems encountered, such as the cost of designing for reuse and the difficulties (technical, legal, social, etc.) in using components developed outside of an organization.

Designing for reuse as well as designing with reuse pose particular problems in academia. Teachers do not wish to put their efforts into learning a technology that is not yet mature and indeed that may not even be successful. Particularly in the field of computer science, there is so much new technology to learn daily that teachers must ration their time carefully. Methodologies and standardized taxonomies, templates, and metrics have not yet been developed. Appropriate textbooks and other course materials have not yet been produced. The allocation of course time in academia, via semesters of a few months, may not be appropriate for learning reuse material. And, of course, grading in academia has traditionally penalized students that "reuse" others' work. Unlike the social sciences, where students are expected to research good quality work (evaluated by experts and maintained in a library) and integrate this work into their own with appropriate footnotes, computer science courses usually have students write code "from scratch." There are exceptions with C and FORTRAN libraries, for example, but the level of reuse is small and lightly documented.

2 Position

In trying to answer the two proposed questions, our group made the following suggestions:

1. Each University should establish a component library. Standards are emerging for library interoperability, and these should be adhered to where possible.(See rig@ballston.paramax.com and ajpo.sei.cmu.edu.) Components of the library can be code, documents, specifications, etc., but standards must be defined and maintained in a specified form. For example, module headers (that include performance characteristics as well as other information specified by RIG) and test suites of a standard form must be included with all code. Multiple implementations of the same specification are very valuable, and these should be easy to obtain if a course includes assignments of components according to specifications, and the professor stores the specification with a few of the best projects (each with possibly different performance charcteristics).

   A component that is inserted into the library must be approved by a professor of an appropriate department and also by a librarian, whose job it is to guarantee that all components satisfy the prescribed format. Components can only be accessed for reading (perhaps code stored in source form, so that anyone can copy it, compile it, and execute it, but not change it in the library). The librarian only will control changes (possibly deletion) if errors are detected.

   Beginner courses in computer science and software engineering should require the use and documentation of components from this library into assigned projects. New systems would therefore have modules of others that clearly indicate where, when and by whom they were developed. Later courses should more thoroughly integrate larger and more varied components.

   A black box approach [1] is very valuable in industry where it is comparable to engineering pluggable components. Program instantiations and transformations have equal importance, similar to the customization of hardware components. Perhaps, however, a white box approach is most applicable to an educational curriculum spread over a few years. Thus, a white box approach allows viewing of portions of code, documentations, etc. that illustrate standards and techniques for users to follow and can be incorporated into new systems as long as they are appropriately referenced. This process is similar to the analysis and integration of the research of an English major and has some relevance to art and music education.
2. Encourage faculty development in reuse technology. One of the ways might be to maintain among ourselves a list of companies that want this technology, so that we can publicize this information to our colleagues.
3. Encourage development and use of methodologies that support reuse. Object oriented programming does appear to be such a technology, although we recognize that it neither does the whole job, nor comes without a cost. Similar comments can be made about Ada programming.
4. Develop appropriate software projects, possibly covering several semesters.
5. Use existing libraries such as ASSET, CARDS, AJPO.
6. Maintain a list of references to support reuse education in particular. Below are references [1, 2, 3, 4, 5] requested by members of the group. The references where chosen for the specific purpose of reuse education, not for reuse information in general.

2.1 Group Members

Trudy Levine - FDU 1000 River Road, Teaneck, NJ 07666, levine@sun490.fdu.edu
Zeb Awan - Bell Northern Research, Canada
Armin Beer - Siemens Vienna, Austria
Frank Coyle - SMU, coyle@seas.smu.edu
Jofo Franco - Telebras Rtd Center, Brazil, franco@cpqd.ansp.br
Thomas Hemmann - GMD, hemmann@gmd.de
Masayufi Ishifawa - Hitachi Software Engineering Co., Ltd. ishi@eecs.umich.edu
Trent Jaeger - Michigan U., jaegert@eecs.umich.edu
Mike Laux - Michigan State University, laux@cps.msu.edu
Toshimi Minoura - Oregon State University, minoura@cs.orst.edu
Jeffrey Poulin - IBM FSC, poulinj@vnet.ibm.com
David Russell - Penn State Great Valley, rzn@psugv.psu.edu
Angi Voss - GMD, angi.voss@gmd.de
David Wan - National Center for High Performance Computing, c00dcw00@nchc.edu.tw

References

[1] W. Tracz, ed., *Software Reuse: Emerging Technology.* IEEE Computer Science Press, 1990. (A compilation of seminal papers).

[2] *Proceedings of the lst Reuse Education and Training Workshop*, June 18th 1993. Available for downloading from source.asset.com.

[3] G. Sindre, E. Karlsson, and T. Stalhane, "Software reuse in an Educational Perspective," in *Proceedings of the 1992 Software Engineering Education Conference*, Norwegian Institute of Technology, Springer-Verlag, 1992.

[4] T. Biggerstaff and A. Perlis, eds., *Software Reusability*, vol. 1. Addison Wesley, 1989. (A compilation of seminal papers).

[5] T. Biggerstaff and A. Perlis, eds., *Software Reusability*, vol. 2. Addison Wesley, 1989. (A compilation of seminal papers).

3 Biography

Trudy Levine has been writing a column for Ada Letters called Reusable Software Components since 1990, which has been reprinted in Crosstalk and maintained in several electronic bulletin boards. She is an Associate Professor of Computer Science at Fairleigh Dickinson and a corresponding member of RIG. She has been trying to augment reuse material in her software engineering courses over the last few years and has taught a special topics seminar course on software reuse. Her research interests also include conflict control and the Ada programming language.