Tool Support for Software Engineering Education

Spiros Mancoridis, Richard C. Holt, Michael W. Godfrey

Department of Computer Science
University of Toronto
10 King's College Road
Toronto, Ontario M5S 1A4
CANADA

Abstract. Although software engineering is a well-documented area of computer science, courses in software engineering frequently do not give students enough practical experience with concepts such as software design, prototyping, programming, debugging, program understanding, software reuse, and so on. As a result, computer science graduates too often acquire a sound theoretical understanding of software engineering concepts, without practical experience using these concepts (a recurrent complaint from industry). We believe that this problem is partly due to a lack of appropriate tools for software engineering education. In this paper we present an overview of the tools we have developed for software engineering education and how they are successfully being used in the instruction of a software engineering course at the University of Toronto.

Key Words: CASE, Design, Education, Interface Viewer, Object-Oriented Turing, Program Understanding, Programming Environment, Reuse, Software Engineering, Software Landscape, Star System, Tool Integration.

1 Introduction

There is a perception in industry that universities have not been adequately preparing their computer science graduates to cope with the realities of industrial software engineering. Indeed, there has been much discussion recently of calls for educational standards, and renewed industrial interest in computer science education [10]. Many universities are now offering graduate degrees in software engineering, aimed at people already in industry.

The shortcoming of current software engineering education practice seems to be that it does not concentrate on the activities that a practitioner is likely to engage in. Traditionally, the educational emphasis has been on programming and in-the-small development, and on systems built from scratch. In practice, however, many developers work on huge legacy systems, and most of their effort is spent understanding, evolving, and reconfiguring these systems. Obviously, what is needed is a shift of emphasis in education toward in-the-large development issues, such as program understanding and evolution, and reuse.

Given these shortcomings in software engineering education, it is natural to consider what supporting tools are appropriate. A common approach has been to use a CASE tool for analysis and design together with editors and compilers for coding and building. CASE tools have several advantages: they often use an intuitive, visual approach to represent system designs; many support the object-oriented paradigm; and, being commercial products, they are usually robust and have a large user base.

However, there are several important drawbacks to the CASE approach. CASE tools can be restrictive, in that they often require the use of a particular development methodology. CASE tools are also expensive to buy, and often require ongoing technical support. The most serious problem, however, is that CASE tools are usually logically isolated from the end software product. The conceptual leap of faith between the CASE model and the actual code makes it difficult to perform activities involving both representations, such as top-down and bottom-up design, system evolution, reverse engineering, and verification.

The approach we have taken is to integrate tools for design, prototyping, editing, programming, debugging, program understanding, and reuse within a single framework. This development environment is built around the Object-Oriented Turing (OOT) language. The visual design tool is called the Software Landscape viewer; it is used to represent system designs, and supports limited querying. There is also another tool, called Star, that is used to extract Landscapes from existing software systems, generating code templates from designs, as well as generating system information that can be processed by yet other tools, such as the Interface viewer. In Section 2, we will discuss the OOT system, the Software Landscape and Interface viewers, and the Star Tool in more detail. Later, in Section 3, we will discuss how these tools are used at the University of Toronto for teaching a course in software engineering.

2 Tool Descriptions

In this section, we briefly describe the OOT programming environment. A thorough description of OOT has been presented elsewhere [8]. In this paper, we will emphasize the features of OOT that are related to large-scale software development.

2.1 OOT: A Programming Environment

Object-Oriented Turing(OOT) is an extension of the Turing language [4]. Its programming environment[1] includes a tightly-integrated set of tools for editing, high-speed compiling, linking, executing, and debugging OOT programs, as well as for browsing the Unix file system. An important aspect of the OOT environment is its consistent user-interface. To the user, OOT consists of a number of windows easily identifiable by variations in their colour, size, screen position,

[1] There are a few good surveys on programming environments [3, 12].

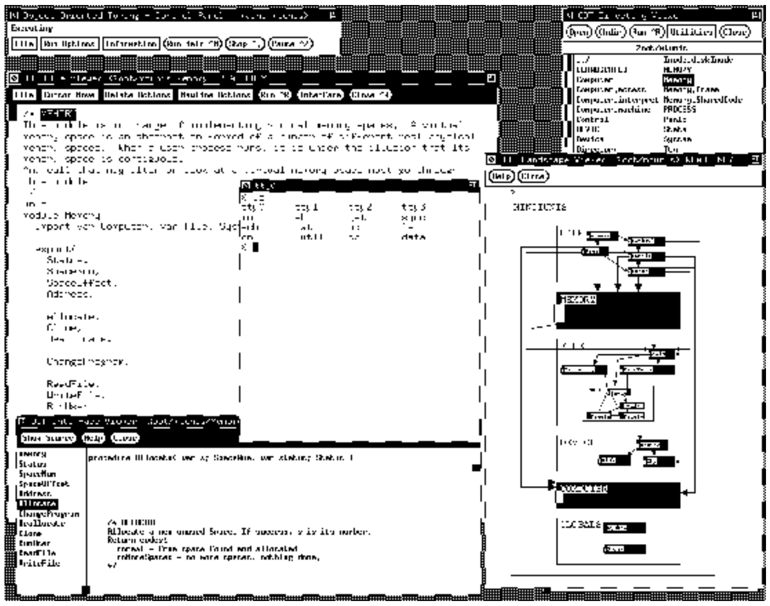

Fig. 1. OOT Snapshot. This figure is a snapshot of the OOT environment. On the top left corner resides the Control Panel. Under this is an OOT File Viewer containing source code. Under the File Viewer is the Interface Viewer which shows the interfaces of a module. The OOT Directory Viewer is at the top right corner and contains a list of the files that make up a system. Below the Directory Viewer is the Software Landscape Viewer which graphically depicts the structure of a system. The central window is a run-time window which shows the output generated by running a system.

and titles. OOT's support of a rapid edit-compile-link-debug cycle is similar to that of Borland's Turbo environments [1].

What is missing from most programming environments, such as those offered by Borland, is support for large-scale software development such as CASE tools. To support large-scale software development, the OOT environment was extended by two additional tools that provide higher-level, abstract views of OOT programs. The first tool is called the *Interface* viewer and the second the *Software Landscape* viewer [11]. These new tools are loosely integrated with the OOT environment through shared files. Figure 1 shows a snapshot of the OOT environment in execution.

Interface Viewer

The OOT Interface viewer is used to display the signatures of all exported entities (*e.g.*, functions, procedures, types, variables, constants) of an OOT unit (*i.e.*, module, class, monitor or subsystem) along with textual annotations describing them. Clicking on the name of an exported entity in the Interface viewer displays the signature of that entity as well as any comments in the source code associated with that entity. Double clicking on one of these names causes the source code of that method to pop up in an OOT editor window.

The information displayed in the Interface viewer is stored as a separate file (one interface file per OOT unit) and is interpreted by the viewer tool. These interface files are created automatically by the *Star* system described in Section 2.2. The interface information serves as a form of documentation, and is used for program understanding.

Software Landscape Viewer

The *Software Landscape* [5, 7] is a graphical notation for relating software entities, such as designs, with non-graphical entities, such as source code, in a controlled, integrated and consistent manner. A distinguishing feature of the Software Landscape is its formally-defined rules of well-formedness [11].

The term "Landscape" is intended to suggest a single visual framework within which all software development occurs; the products of software development are placed in the Landscape to be explored, understood, and used by others. These products are stored as variously related entities. Ideally, this stored information is viewed and manipulated on a large colour screen.

The Landscape is meant to aid large-scale programming. The lowest-level entities are programming language constructs such as modules and classes. Higher-level entities, such as subsystems, projects, and libraries, which have no analogue in most conventional programming languages, exist as well. The concept of the Software Landscape is related to a new stream of Software Engineering research dealing with software architectures [13].

Landscape entities are interconnected through explicitly defined relations. There are Landscape relations between language-level units, such as *imports*, *exports*, or *inherits*, as well as other relations, such as *part of* (*e.g.*, an entity is *part of* a subsystem), which is not expressible in most programming languages.

The major benefit of the Software Landscape approach lies in making a large amount of software development information easily accessible to the developer, with a uniform visual mechanism that supports browsing at various levels of granularity.

The Software Landscape viewer includes a graphical editor. It allows developers to document the architecture of OOT systems by creating and manipulating Landscapes. The viewer also provides a hyperlink from each box in the diagram to the corresponding source code.

We now describe a system for tool integration called Star, and elaborate on how it complements the OOT environment. A more detailed coverage of the Star system has been presented previously [6].

2.2 Star: A Mechanism for Tool Integration

Each of the tools described in Section 2.1 uses text files to store their information persistently. These text files are represented in Figure 2 as cylinders; the tools

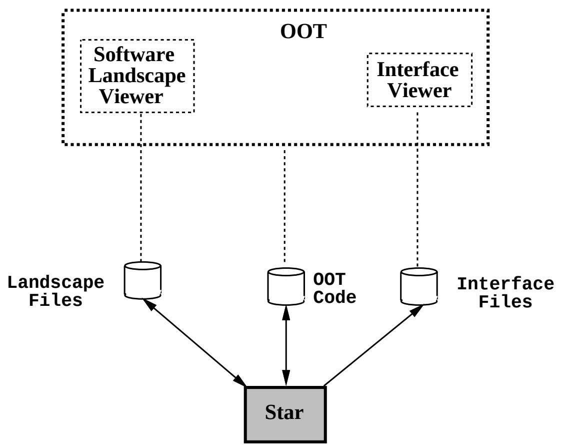

Fig. 2. Architecture of Star. At the top is the OOT environment which contains two tools for large-scale programming: the Software Landscape Viewer and the Interface Viewer. The persistent representation for each of these tools is indicated by cylindrical figures symbolizing files. The dotted lines between tools and files denote which tools use which files. The shaded box represents the Star system; which translates the persistent representation of a tool into that of another tool to achieve tool integration.

that use them are represented as rectangles.

The OOT programming environment stores source code as ASCII files, each of which may have a corresponding interface file used by the Interface viewer. Software Landscapes are stored as text files, which maintain the names, shapes, and positions of all entities in the diagram. We devised a system, called Star, that provides tool integration by translating the persistent notations of each tool to that of each other tool.

When OOT and Star are used in a bottom-up fashion (*i.e.*, Software Landscapes are extracted from the source code), the tools act as a reverse engineering system similar to the Rigi environment [9]. The generated Software Landscapes are automatically laid out using a variation of the Sugiyama algorithm [15].

When used in a top-down fashion (*i.e.*, code templates are generated from the Software Landscape design), the tools act as a CASE environment with "code-generation"[2] capabilities similar to ObjectMaker [16].

The following section elaborates our experiences gained from using OOT and Star to teach a course in software engineering.

[2] The use of the term *code generation* in CASE tool advertisements is somewhat misleading. What is meant is that code *templates* can be generated automatically from the diagrams.

3 Using the Tools in a Software Engineering Course

A fourth year University of Toronto course provides a standard coverage of software engineering concepts [14, 2].

At the beginning of the course, the students are given a set of milestones representing the phases in the life cycle of a software "product" that teams of three students in the course are required to create. The product last year was a Graphical User Interface (GUI) library written in the OOT language, targeted for use by other undergraduates. Ideally, the product would be used by other classes of students to allow them to incorporate GUI support (i.e., menus, buttons) as part of their programs.

The students were provided with basic event handling at the level of X events in OOT, and were required to design, implement, and document a class library of widgets. Each widget was to support a particular object on the screen, such as a push button or a menu.

Students were asked to start by creating a design using the Software Landscape viewer in OOT. Then, they were to "flesh-out" their designs in a top-down fashion. The goal was to provide an interesting exercise in software design, something which is too often missing in undergraduate education. The Star tool would generate code templates from their Software Landscapes initially, then they would use the OOT programming environment integrated programming facilities (editors, compiler, debugger, and so on) to implement their prototype solutions.

At the various milestones, the student teams had to deliver electronic documents detailing their product's requirements, specifications, designs, documentations, and so on. At the end of term, each team gave a demonstration of a sample program, such as a calculator, that used their own GUI library.

Over the summer, the results of the student's projects were amalgamated into a relatively polished GUI library. The Star tool was used to extract the Landscape diagram and the Interface views for this library.

This year, the students were given the new version of the GUI library. They used the Landscape and Interface views for program understanding, *i.e.*, to learn about the GUI library's structure and the functions of its parts. As an exercise, they used the library to develop a GUI interface for an existing application written in OOT.

As a major project, students in teams of three were required to create a GUI builder, which makes use of the GUI library and automatically emits code to create a user interface based on mouse selections. The teams were required to deliver documentation for that project, including a Landscape and Interface views. These were created with the aid of the Star tool.

This hands-on approach, using high-level tools, based on a project that develops a useful product, is highly rewarding to the students. It provides a concrete realization of software engineering ideas such as program design which are otherwise difficult to grasp due to their already abstract nature.

These projects demonstrate how the OOT programming environment, used in conjunction with the Star integration mechanism, can assist software engineering educators to teach essential concepts such as the software life cycle, team work, delivery of reusable software, program understanding, prototyping, and top-down and bottom-up design.

4 Conclusions

In this paper we have emphasized the importance of using appropriate tools for teaching software engineering courses. We have been successfully teaching a course in software engineering at the University of Toronto using the OOT programming environment and the Star tool integration mechanism. We believe that state-of-the-art tools such as these are essential in helping students master software engineering concepts.

References

1. Bannister, H. Borland Introduces Turbo Prolog, Version 1.1. *InfoWorld 8*, 39 (September 1986).
2. Blum, B. I. *Software Engineering: A Holistic View.* Oxford University Press, New York, New York, 1992.
3. Dart, S. A., Ellison, R. J., Feiler, P. H., and Habermann, A. N. Software Development Environments. *IEEE Computer* (November 1987), 18–28.
4. Holt, R. C., and Cordy, J. R. The Turing Programming Language. *Communications of the ACM 31*, 12 (December 1988), 1410–1423.
5. Holt, R. C., Penny, D. A., and Mancoridis, S. Multicolour Programming and Metamorphic Programming: Object Oriented Programming-in-the-Large. In *Proceedings of the 1992 IBM CASCON Conference* (November 1992), pp. 43–58.
6. Mancoridis, S., Holt, R. C., and Godfrey, M. W. A Program Understanding Environment Based on the "Star" Approach to Tool Integration. In *To appear in the Proceedings of the Twenty-Second ACM Computer Science Conference* (March 1994).
7. Mancoridis, S., Holt, R. C., and Penny, D. A. A Conceptual Framework for Software Development. In *Proceedings of the Twenty-First ACM Computer Science Conference* (February 1993), pp. 74–80.
8. Mancoridis, S., Holt, R. C., and Penny, D. A. A "Curriculum-Cycle" Environment for Teaching Programming. In *Proceedings of the Twenty-Fourth ACM SIGCSE Technical Symposium on Computer Science Education* (February 1993), pp. 15–19.
9. Müller, H. A. Rigi as a Reverse Engineering Tool. Tech. Rep. Technical Report No. DCS-160-IR, University of Victoria, March 1991.
10. Müller, H. A., and Slonim, J. J. *National Workshop on Software Engineering Education.* Center for Advanced Studies IBM Canada Ltd., Toronto, Ontario, 1993.
11. Penny, D. A. *The Software Landscape: A Visual Formalism for Programming-in-the-Large.* PhD thesis, Department of Computer Science, University of Toronto, 1992.
12. Perry, D. E., and Kaiser, G. E. Models of Software Development Environments. In *Proceedings of the 10th IEEE International Conference on Software Engineering* (Singapore, 1988), pp. 60–68.

13. Shaw, M. Larger Scale Systems Require Higher-Level Abstractions. In *Proceedings of the Fifth International IEEE Computer Society Workshop on Software Specification and Design* (1989), pp. 143–146.
14. Sommerville, I. *Software Engineering, Fourth Edition.* Addison Wesley, Reading, Massachusetts, 1992.
15. Sugiyama, K., Tagawa, S., and Toda, M. Methods for Visual Understanding of Hierarchical System Structures. *IEEE Transactions on Systems, Man, and Cybernetics 11*, 2 (February 1981), 109–125.
16. Williams, T. Object-Oriented CASE Tool Lets User Tailor His Own Methods. *Computer Design* (September 1991), 122.

A Appendix: FTP Access to OOT Software Demonstration

There is an on-line demonstration version of OOT from the University of Toronto that can be accessed by anonymous FTP. The OOT environment is currently implemented on Unix platforms, such as Sun/4's, RS/6000 and SGI, plus a soon to be released version for PCs. If you have access to the Internet and Unix, you can get instructions to access the demo by executing the following:

```
%ftp 128.100.1.192
ftp> cd pub
ftp> get ootDistrib
ftp> quit
```

The ootDistrib file in your directory will now contain details on getting the demo.