AI and Software Engineering -- Managing Exploratory Programming

Richard Fikes

Price Waterhouse Technology Centre
68 Willow Road
Menlo Park, California 94025 U.S.A.

Introduction

I have been asked in this note to comment on the nature and extent to which the technologies and methodologies used to build "smart" application systems have contributed to software engineering. I will interpret the term "software engineering" broadly here as refering to the ability to create software solutions to problems, and will consider "'smart' application systems" to be what are generally referred to as "expert systems" or "knowledge-based systems".

First of all, knowledge-based systems have contributed directly to software engineering by providing a technology that expands the range of tasks which computers can effectively perform. Generally speaking, the expansion has been to simple decision tasks that were previously considered to require human intelligence. Much of the first generation rule-based and object-based technology which enabled the expansion is now being integrated into conventional programming environments so that it will be available for use when needed as a standard part of the professional system builder's repertoire.

The AI community at large has been a major contributor to software engineering by playing a pioneering role in the development of symbolic, interactive, and exploratory programming. The building of knowledge-based application systems has typically required all three of these forms of programming and has recently been particularly responsible for motivating the development of techniques for effectively managing exploratory programming activities. (See, for example, (Walters and Nielsen 1988), (Schoen and Sykes 1988), and (Fikes and Jacobstein 1989).) Since exploratory programming can be a useful methodology in a broad range of both AI and non-AI system building projects (Sheil 1983), I consider techniques for managing its use to be a significant contribution to software engineering. It is those techniques that I wish to address briefly in this note.

Managing Exploratory Programming

As we know from experience, knowledge-based system applications typically require an exploration process in order to determine the functionality, knowledge, and processing methods that are needed to perform the target task. The initial specifications from the client for such projects are often little more than aspirations (e.g., recognize the tax issues in a business situation, monitor the operation of a plant). Therefore, a major goal of the early part of a knowledge-based systems application project is to determine a realizable detailed set of functional specifications that are satisfactory to the client. The typical means for achieving that goal is to build a sucession of prototype systems, each of which is exposed to users and experts for comment and then refined to produce the next system. The challenge faced by project planners and managers is how to allow such exploration to occur and still maintain an acceptable degree of predictability in the cost, timing, and results of the project.

In the last few years, a set of guidelines has emerged from the building of knowledge-based systems for managing exploratory programming activities in the form of additions to and modifications of conventional software engineering management methodology. The following describes some of the principal points in those guidelines.

Conventional software engineering methodology holds that central to any software development is the availability of (1) a system requirements specification stating the problem to be solved and criteria for success against which the application can be designed, implemented, tested, and validated; and (2) a project plan containing an activity schedule and a description of the deliverables and costs for each activity (Sommerville 1989). What has been learned is that the use of exploratory programming does not lessen the importance of writing such a specification and plan at the beginning of a project and of maintaining those documents throughout the project, even though they are initially lacking in significant detail and change substantially as the project progresses.

The requirements specification and project plan are important because they represent a common agreement among the interested parties in a project, including the sponsor, experts, users, and developers. For exploratory programming projects, they also indicate at any given time what has been learned thus far and what questions remain open to be answered by the exploration. The project manager's job is then one of

managing the evolution of that agreement and assuring that the open questions are being effectively addressed.

The first guideline in managing the evolution of the common agreement is to make sure that all interested parties understand from the outset that the agreement is in fact going to evolve. Everyone must understand that the current agreement at any time during the exploratory phase of the project is only a best guess and that the primary purpose of the exploration is to improve those guesses and remove the uncertainties in them. Thus, the changes in the requirements specification and project plan at the end of each exploratory step are the *results* of that step. If there are no changes and no reduction of uncertainty, then nothing has been accomplished by the exploration.

The primary way to gain support for a project in which specifications and plans are expected to change is to establish clear procedures for updating the specification and plan and for informing all interested parties when those changes occur. (The manager can make clear, also, that the updating and informing procedures are *not* expected to change.) The central element of those procedures is to have frequent project reviews, including prototype demonstrations, that give all interested parties an opportunity to revise and renew their agreements. Such reviews typically produce a list of errors or problems in the prototype to be addressed in the next step such as desired changes in the user interface, missing or inadequate areas of knowledge or functionality, or errors in the knowledge or its representation.

At each project review, a detailed specific plan needs to be presented for the period up to and including the next review. An important element of that plan is a list of objectives for the step. Such a list would include the unresolved requirement specification issues to be addressed and errors or problems in the prototype that were found during the review to be fixed. These objectives provide the developers with a basis for deciding what to do next and a set of criteria for evaluating the results of the next step. At any time during a project, the plan for the current prototyping step is not expected to change substantially. Thus, an exploratory programming project can be considered to have a stepwise stable project plan such that at each review point a stable plan exists for the next step.

Summary

The technologies and methodologies used to build knowledge-based systems have expanded the range of tasks that computers can effectively perform and has contributed to the development of symbolic, interactive, and exploratory programming. In particular, experience in building knowledge-based systems has produced a set of guidelines for managing both AI and non-AI exploratory programming efforts. Those guidelines have contributed substantially to making exploratory programming a viable system development technique for use whenever there is significant uncertainty as to the functional requirements for the desired system.

References

Sommerville, I. 1989. *Software Engineering*. Reading Mass.: Addison-Wesley.

Walters, J.R., and Neilson, N.R. 1988. *Crafting Knowledge-Based Systems; Expert Systems Made (Easy) Realistic*. New York: Wiley-Interscience

Schoen, S., and Sykes, W. 1987. *Putting Artificial Intelligence to Work; Evaluating and Implementing Business Applications*. New York: John Wiley & Sons, Inc.

Fikes, R., and Jacobstein N. 1989. *Managing Expert System Projects*. Tutorial: MA5. Eleventh International Joint Conference on Artificial Intelligence. Detroit: IJCAI-89.

Shcil, B. 1983. Power Tools for Programmers. *DATAMATION* February,1983.