Experience Report

Using Object-Oriented Development to Support Prototyping*

Lin Zucconi†, Gregory Mack‡, and Lloyd G. Williams††

†Lawrence Livermore National Laboratory
Livermore, CA USA

‡†Software Engineering Research
Boulder, CO USA

Abstract
This report describes a project in which semi-formal software development methods were used in parallel with a prototyping effort to develop a real-time monitor and control computer system. The prototype was used to define and clarify the requirements for the system. The semi-formal approach was used to capture and document those requirements. The resulting description of the system was then used to migrate the prototype to a first version production system. This combined approach helped to overcome many of the problems typically associated with prototyping alone.

This project was also used as a vehicle for technology transfer. Due to constraints in the local environment, the approach to technology transfer used here differed substantially from those typically described in the literature (see, e.g., [Bou189], [Pres88], [Zuc88]) due to several constraints in the local environment. The nature of this technology transfer effort and the reasons for adopting an unusual approach are also described below.

The following sections provide background information on the project and describe both the prototyping effort and the methods used to support it. The final section, presents some observations and conclusions based on this project.

1. Introduction

Prototyping can be a valuable tool in software development, particularly in cases where the requirements are uncertain or where there is a significant amount of risk associated with the implementation [Boeh84], [Boeh88]. The use of prototyping can, however, lead to several problems. These include: a less coherent design than would be obtained with a specification-based approach, poor (or nonexistent) documentation, and a tendency to deliver the prototype as the finished product as schedule pressures mount.

This report describes a project in which semi-formal software development methods were used in parallel with a prototyping effort to develop a real-time monitor and control computer system. The prototype was used to define and clarify the requirements for the system. The semi-formal approach was used to capture and document those requirements. The resulting description of the system was then used to migrate the prototype to a first version production system. This combined approach helped to overcome many of the problems typically associated with prototyping alone.

This project was also used as a vehicle for technology transfer in an initiative designed to introduce new software methods to a group at Lawrence Livermore National Laboratory. The approach to technology transfer used here differed substantially from those typically described in the literature (see, e.g., [Bou189], [Pres88], [Zuc88]) due to several constraints in the local environment. The nature of this technology transfer effort and the reasons for adopting an unusual approach are also described below.

The following sections provide background information on the project and describe both the prototyping effort and the methods used to support it. The final section, presents some observations and conclusions based on this project.

2. Background

The project described here involved development of a real-time process control system. The overall goal of the project was to develop an understanding of the requirements for the system and demonstrate its feasibility through a prototyping approach which involved both software and hardware aspects. The results of this prototyping effort were then used to produce a production version of the system.

This project also involved a significant technology transfer component. The target organization is one of many which develops software to support research efforts at Lawrence Livermore National Laboratory. This software tends to be one-of-a-kind, research applications. Project teams tend to be small (1 to 5 individuals) and, at any one time, there are many projects underway, each with a different schedule and deadline. Despite the research nature of the projects, the software systems which result tend to be long-lived and undergo significant modification and evolution over time. Because of the large amount of resources being consumed in maintenance, it was felt that a more rigorous approach to design and documentation would be valuable.

Technology transfer initiatives have traditionally involved an assessment of current capabilities and needs, followed by selection of appropriate software engineering technologies (here,
software engineering methods) and training of target personnel in the use of the new technologies (For discussions of this approach, see e.g., [Bou189], [Pres88], [Zucc88]). In this case, however, an assessment of the target organization showed that the traditional approach would be inappropriate. The small teams and staggered schedules made it difficult to schedule training classes. The traditional classroom training was, therefore, replaced with one-on-one and small group tutorials. Key individuals were identified for training as "in-house" consultants using this approach with the intention of "seeding" these individuals throughout the various projects. In addition, because there was some question as to whether the new methods would be appropriate for this environment, the project described here was chosen as a test case to provide an evaluation of the effectiveness of the new methods.

The prototyping effort was already underway when the technology transfer effort was begun and, since there were significant schedule constraints, it was decided to pursue semi-formal analysis, specification and design of the system in parallel with the existing work. Since portions of the prototype had already been coded, a significant amount of "design recovery" [Bigg89] was required to model the existing elements of the system. Once this initial information had been captured, it was possible to use the more traditional "forward" approach.

The development/technology transfer team consisted of a programming/prototyping group (a total of four individuals were involved during the project), a consultant to provide training and assistance in use of the new methods, and a test engineer who was responsible for quality assurance for the production system. One of the developers was selected to use the new methods and receive tutorials in their use.

3. Prototyping

The project began with a requirements definition process (documented according to IEEE Standard IEEE 830-1984). This process was used to define constraints related to the hardware (MicroVAX II), operating environment (VaxELN real-time executive), implementation language (Ada), and the interface to the programmable logic controller (parallel I/O) which provides the interface between the computer system and the process control hardware. The requirements definition process was also used to define the process to be used to develop the production system. Since details of the machine interface (MMI) and computer system control functions were both vague and volatile, it was decided that a prototyping approach would be used to clarify these requirements. A phased approach was, therefore, chosen for implementation of the production-level software. The first phase consisted of construction and evaluation of the prototype.

The prototype was exercised by end-users over a period of about six months. This gave the developers the information they needed to complete the MMI and provided greater detail on the required functionality. Information developed via the prototyping approach included:

- Information about the MMI, including screen layouts, keyboard interaction and operational messages to be displayed in response to operator requests.
- Requirements for control functions. Exercising the prototype led the end-users to identify new requirements for control functions and data to be displayed at the operator's console. These new capabilities were added and validated by further exercising the prototype.
- Identification of potential concurrency in the control functions.

The prototyping effort produced a "Version 0" computer system which was then migrated, based on the object-oriented approach described below, to a "Version 1" system which met the critical needs of the end-users and delivered a facile user interface. The overall development effort required approximately 29 person months of effort (9 person-months for requirements investigation/definition, 1.3 for prototype development, 2.5 for model development, and 2.5 for test engineering). The prototype consisted of approximately 6,000 lines of Ada code. The production system was slightly smaller. Approximately 40 percent of the prototype code was included in the production system. Version 1 has now been operational for several months with no defects reported. Feedback from the end-users indicates that the performance and user-interface of this system are superior to those for systems developed using the previous ad hoc approach.

4. Software Development Method

The approach used on this project was a variant of object-oriented analysis, specification, and design. This approach uses the notations of real-time structured analysis and design [Ward85], [Ward86] but applies them in a way that is radically different from the traditional approach [Ward89]. Rather than proceeding via the traditional route of functional decomposition, the common real-world things (objects) in the application domain, together with the operations which they perform or which are performed on them, are identified.

This approach involves the construction of a set of models (an information model, a behavioral model and a process model) of the application domain, followed by mapping those models onto the target environment. Both the specification and the design are organized to reflect the objects in the application and/or implementation domains.

The information model, which uses an extended entity-relationship modeling technique, is used to identify application domain objects and their attributes (instance variables). The information model also describes structural relationships (e.g., association and inheritance) among these objects. The behavioral model uses state-transition diagrams to describe the state-dependent behavior of the object. The process model uses a transformation schema [Ward86] to describe the operations performed on or by the object. Together, these models provide a means of analyzing and specifying a system in terms of the static and dynamic properties of the objects which it manipulates. The models are also sufficiently detailed and formal to be executable, as demonstrated in [Webb86] and [Rei87].
In this project, the various models were used to simultaneously capture knowledge about both the objects in the application domain and the evolving prototype implementation. Models were constructed, verified against the prototype and validated through informal reviews with the end-users.

5. Interactions Between the Approaches

There was considerable interaction between the two approaches. The prototype was used as a starting point for development of the object models and the models were used as test cases for the prototype.

The model-building process led us to ask better questions of the end-users and engineers, leading to clarification and modification of the existing requirements. Inconsistencies in draft versions of the information model led to the identification of missing information. The model also provided clues to the nature of this information, making it easier to obtain. Walkthroughs of the models (using manual "execution") helped to clarify potentially inconsistent functional requirements. In some cases, this improved the development team's understanding of the system; in others, it helped identify requirements which had been previously overlooked.

The modeling process also flagged some potential problems with the design and implementation of the prototype. These included:

- Use of the information model helped identify objects from the application domain more appropriately. This, in turn, led to the definition of more accurate abstractions for these objects in the implementation domain. The result of this process was the combination of some Ada tasks to more adequately reflect these abstractions. Other, superfluous, tasks were eliminated.

- Development of the information model helped identify deficiencies (i.e., missing or inconsistent information) in the interface between the programmable logic controller and the computer system.

- Manual simulation using behavioral models derived from reverse-engineering of the prototype code led to discovery of a deadlock problem. This problem had previously gone undetected in exercising the prototype.

6. Assessment

Experience on this project has led to a number of observations/conclusions. It should be emphasized that the work reported here is not the result of a formal, controlled experiment. It does, however, represent careful observation of an actual software development project and technology transfer effort. It should also be noted that this project is unique in many ways. Generalization of these observations should, therefore, be done with caution. Conclusions are presented in three categories: conclusions regarding prototyping, conclusions regarding technology transfer, and conclusions regarding the use of the Ada language on this project.

6.1 Prototyping

The use of a semi-formal development technique as an adjunct to prototyping can provide significant benefits. These include:

- Development of information, behavioral and process models can improve communication between software developers, hardware engineers, and end users. Construction of models, such as those described in Section 4, can help developers gain an understanding of the application domain more quickly and assist in identifying missing information. Because they could be at least manually "executed," walkthroughs using the models could be used for validating many aspects of the prototype without having to resort to coding. Areas which are poorly understood were also explored through the use of "what if" scenarios based on the models.

- The use of a disciplined approach as an adjunct to prototyping can improve the design of the prototype. Boehm [Boeh84] has pointed out that prototypes often exhibit a design which is less coherent that those produced by more disciplined approaches. Using a more rigorous development approach in parallel with the prototyping effort, makes it possible to capture information about the evolving prototype and use this information to improve its design.

- The models described here can provide assistance in generating test plans. Significant portions of the functional test plan can be derived from the information model. The behavioral model provides information that is useful in devising path testing strategies. Process models provide specifications for primitive processes that can be used in developing unit tests.

- A semi-formal method can help establish a method and a set of criteria for migrating prototype code to a production-quality system. One problem with prototyping as an approach to software development is that schedule pressures often lead to delivery of the prototype as the finished product. Since it is possible to verify and validate the models independent of the code, they can then be used as a basis for assessing the quality of various components of the prototype. Those that meet the criteria may be used directly in the production system; those that do not should be reimplemented.

Experience on this project supports use of the Spiral Model of software development proposed by Boehm [Boeh88]. The spiral model describes software development as an iteration over four phases of activity which combine several approaches, including specification-driven and prototype-driven development. The particular combination of these activities which is to be used at a given point is determined by the need to identify and resolve uncertainty or risk in the development process. The process used here, while not identical to the Spiral Model, was similar in many respects. Prototyping was used to resolve areas of uncertainty and more traditional methods were used where applicable. Development of an enhanced version of the system is now underway. The project plan for this new
version follows the Spiral Model even more closely.

6.2 Technology Transfer

Integration of new software methods with ongoing projects can be an effective means of technology transfer in some environments. The approach used here provided more depth of training for those individuals who participated. In addition, developers gained experience using a real problem from their application domain (rather than a case study) and had assistance in overcoming the inevitable startup problems. Finally, the product provided a convincing demonstration of the effectiveness of the methods.

An approach such as this may be particularly effective in environments where local constraints make more traditional approaches impractical. There are, however, some potential drawbacks. While the training achieved using this approach tends to be more in-depth, fewer individuals are trained. This approach may, therefore, be more expensive and time-consuming than the traditional approach.

This drawback may, however, be less significant that it first appears. Application of the methods described here requires discipline and is typically difficult for novices. Technology transfer efforts have failed when new users have given-up early, producing only superficial models. These superficial models have been less useful than expected, leading to the conclusion that the methods are not useful. At a minimum, follow-up support (for example, on an initial or pilot project) should be provided as an adjunct to training in the use of new software technologies.

6.3 Use of Ada

The use of Ada as the development language enhanced the overall development effort

- Ada provided good support for prototyping. Specifications were written to support the major software structures (objects). Each developer was then free to implement the "bodies" of their respective objects more or less independently.

- The object-oriented method used here integrated well with the Ada language (see, e.g., [Booc87]). In addition, as the models were developed, it was possible to use the models to refine the original prototype code, simplifying some packages or tasks and combining others.

7. Acknowledgements

We gratefully acknowledge the support and cooperation of the following members of the Computers and Communications group at Lawrence Livermore National Laboratory: Joe Brandt, Jo Sander, Fred Holloway, Pam Viebeck, and Julio Diaz.

8. References