Enhancing Design Methods to Support Real Design Processes

Barbara Staudt Lerner
Stanley M. Sutton, Jr
Leon J. Osterweil

CMPSCI Technical Report 98-06
January 1988

Laboratory for Advanced Software Engineering Research
Computer Science Department
University of Massachusetts

Effort sponsored by the Defense Advanced Research Projects Agency and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-97-2-0032. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon."

The views and conclusioins contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Rome Laboratory, or the U.S. Government.

Enhancing Design Methods to Support Real Design Processes

Barbara Staudt Lerner*, Stanley M. Sutton Jr., and Leon J. Osterweil
Computer Science Department
University of Massachusetts
Amherst, Massachusetts 01003
lerner@cs.umass.edu, sutton@cs.umass.edu, ljo@cs.umass.edu

Preferred track: Traceability, Integrity, and Change

Abstract

Software design methods typically focus on the activities that individual designers should perform under ideal circumstances. They rarely, if ever, address the activities that should be performed when things do not go according to plan, such as when a customer requests changes to the specification, or when early design decisions must be changed. They also rarely address issues involving coordination of multiple designers in cooperative design tasks or in competition for limited resources. We are investigating fundamental concepts required for more complete definition of design methods, developing linguistic mechanisms within a process programming language to support these concepts, and validating these through the definition of a process program that incorporates the Booch method.

1 Motivation

In recent years, numerous software design methods (for example, [1, 4, 2, 6, 3]) have been proposed to describe how successful software designers build systems so that others may learn from their experiences and use the same techniques. These methods are useful in providing an outline of activities to follow, and specifications of what well-formed finished software design products must look like. But they leave many questions unanswered about how to actually execute the method, and go about crafting the final well-formed product. As a result, we see that most tools purporting to support these methods actually only support the creation and editing of graphs constructed in the notation propounded by the method.

We have been studying the general problem of providing support for the full range of software design activities, paying special attention to such issues as the coordination of multiple designers, the handling of unplanned activities, and support for the myriad details that design entails. While the thrust of this work is quite broad, and often highly conceptual, we have found that it is quite beneficial to also reduce it to operational practice. We have done this by developing a formal process definition for a specific software design method, namely Booch Object Oriented Design [1], expressing the process definition in the JIL process programming language [5]. This process definition work has had a number of significant benefits. First, because defining a process by means of a programming language provides a precise semantics, we are able to be quite rigorous in the definition of the method. This enables us to provide details of the method that were lacking in the initial process description, and often to confirm conceptual content that may

*Voice: 413-545-3787. Fax: 413-545-1249

have been unclear. Second, a formal definition gives us a means by which we can compare the definitions of methods more precisely. This has enabled us to derive more firm and complete understandings of the relationships between several of the currently proposed software design methods. Third, a definition in terms of a process programming language offers the possibility of developing tools that provide sharp and direct support for various of the several activities that comprise the design method, rather than for only the creation and maintenance of the design notation.

On initial examination, one might feel that a programming language would be ill-suited for the definition of processes in which human creativity has an important role. Thus, our process definition project also enabled us to experiment with our hypothesis that such programming of a design process can indeed be effective provided that the programming language offers suitable features. We found that JIL offers various features that do indeed make it very appropriate as a vehicle for the definition of processes in which human creativity is essential. JIL offers a synergy between the user and program that is not typical of most programming languages. In particular, JIL provides control constructs that allow the human to be the decision maker and thereby use human intelligence to guide the program. Furthermore, JIL programs can be written to an arbitrary level of detail. The program might provide only an outline of what should be accomplished, and leave the hard work entirely to the user, as the Booch method does, or provide arbitrary amounts of detail on how to accomplish the process.

With these two mechanisms, we found it to be quite feasible to capture the semantics embodied in the Booch method. The result, however, is unsatisfactory from the point of view of providing a tool that designers could actually use. The reason is that the Booch method does not accurately capture how designers do their jobs. Fundamentally, the Booch method lacks sufficient detail to be considered an engineering process. Lack of detail is evident in the weak definitions associated with most of the activities comprising the method and also in its complete silence on some fundamental activities that are essential in software design, two of which we discuss in this paper. First, the Booch method only describes the "nominal process", what designers should do when everything is going according to plan. Second, it does not describe how multiple designers can effectively collaborate to produce a design.

2 Coping with Unpredictable Events

Our investigations have indicated that software design methods are primarily descriptions of the steps that a designer should perform, and rough postconditions that characterize the acceptance criteria for those steps. As any practicing software engineer can report, many design activities are not simply a matter of marching through from one step to the next. Errors are introduced and must be fixed, customers request changes to the requirements whose impact on the existing design must be analyzed, the design must be modified to incorporate changes, people leave the project, new people join the project. In short, design does not typically proceed in a smooth fashion, but instead has bursts, slowdowns, and even retreats on its path to completion. Any system that can effectively help the designer through this process must be able to cope with such irregular, unplanned, and possibly unpredictable events.

Modern programming languages use exception mechanisms so that normal code can be separated from error-handling code. This simplifies the understanding of the normal code as well as explicitly identifies the error handling code and the specific errors being handled. The same effect is desirable in describing design processes. Such a mechanism allows the separation of the nominal process from error-handling activities performed during the process. This type of mechanism would be beneficial to deal with describing errors detected at specific points in the process. For example, the Booch method describes milestones that must be satisfied before the designer should proceed to the next activity. Attempting to proceed when those

milestones are not satisfied would be considered an error in the process. Reacting to that error, which might result in iterating the activity, could be specified in an exception handler.

More general events also arise during the design process, particularly in situations where multiple designers are collaborating. One designer may require the results of another designer in order to proceed. Thus designers may want to be notified when certain significant events that are beyond their control occur. For example, a designer might want to be notified when a piece of the design on which he/she depends is modified. Other events might have broader implications. For example, a change to the requirements should be broadcast to all designers responsible for applicable sections of the design. Having a designer quit the project will likely result in reallocation of responsibilities. A complete design process should specify these general events and the appropriate reactions for them, again relying upon humans to fill in the details of the reactions.

In our development of a JIL process program to define Booch Object Oriented design we made frequent and effective use of JIL: proactive control constructs to support the specification of the nominal process, preconditions and postconditions to assist in the identification of errors in the execution of a process, the exception handling mechanism to deal with those errors, and the reaction mechanism to deal with general events. We believe that this experience strongly suggests that our approach of using a sufficiently powerful process programming language can be effective in successfully capturing the complex and elusive nature of control flow in software design. Our research into this is continuing.

3 Coordination of Multiple Designers

The Booch method focuses on the activities of an individual designer. It is quite rare, however, for designers to work alone on a project. Cooperation among designers is common; competition for limited resources is not unusual. Any method that is precise enough for designers to use as to guide their activities must address these issues.

3.1 Cooperative Concurrency Control

The design activity results in the creation of multiple design artifacts representing different pieces of the system being designed as well as different aspects of those pieces. Some artifacts, such as class diagrams, describe the static characteristics of the system; others, such as state transition diagrams, describe dynamic characteristics. The artifacts are highly interdependent. All artifacts that present different aspects of the same design element are obviously closely related. In addition, design elements depend on other elements that describe classes/objects being used or inherited from.

The interconnections between the artifacts can be used to ensure consistency between the artifacts. If one artifact is changed, consistency requirements are re-evaluated. If the change is found to be inconsistent, this will throw an exception which can be handled either by retracting the change or updating other design elements to become consistent with the change.

This process of consistency checking and change propagation becomes more complex when multiple designers are involved. At any one time, each designer is responsible for a subset of the elements being designed. Designers must interact to be sure that they are developing a consistent design. For example, if class A uses class B, class A must provide the operations required by class B. If the design of either class changes, it may affect the other. If the two classes are being designed by different designers, the designers must collaborate to ensure this consistency. Typically, designers become aware of each other's activities through traditional communication channels: meetings, electronic mail, hallway encounters, etc.

We are investigating cooperative concurrency control mechanisms to allow designers to selectively share artifacts with each other. The concurrency control mechanisms are weaker than the serializability requirements commonly found in database systems in that designers can share intermediate results, not just those that have been made visible in some project database. The intent is to allow the sharing of the actual design artifacts at the same stage of development as the communication channels allow. The benefit of this is that consistency checking across design artifacts can be done more accurately if it is based upon intermediate versions of the artifacts rather than against the previous released version which the designers know is out-of-date.

3.2 Resource Management

Design relies on the availability of appropriate resources. These resources include hardware resources, software tools, and personnel. The same design method applied to the same project can result in different activities being selected, different assignment of tasks to designers, and different scheduling decisions, particularly in the amount of concurrency that is achieved.

We are investigating mechanisms to allow us to define resource models describing general resource classes and their characteristics, environments describing the specific resource instances available to the process, resource needs of process steps, and scheduling algorithms that can intelligently assign resource instances to process steps to minimize competition for resources.

We would like to be able to write a design process program that abstracts away from the specific environment in which it is run so that it can be reused in multiple environments, can cope with insufficient or inappropriate resources, and can react to changes in the resource environment.

3.3 Agenda Management

As a design unfolds, it is important for designers to know what their responsibilities are, for managers to know who is responsible for what, and to allow responsibilities to be moved from one designer to another. Design methods do not address these issues as they focus on individual designers and assume the designer is responsible for keeping track of his/her own work.

We are developing an agenda management system that assists in these activities. Each step of a process is assigned to an agent by placing it on the agent's agenda. An agent can be human or software. An API is provided to allow software agents to interact with their agendas, while a GUI is provided for human agents. Agendas provide the ability to manage to-do lists, assess the workloads of individual designers, provide traceability of a designer's activities, find the designer responsible for a particular activity, and reallocate activities between designers.

Again, we would like to describe design processes that are independent of specific designers, yet allow the designers to collaborate effectively by tracking the responsibilities of individual designers as those assignments are made and completed.

4 Scenario

Assume that a "meeting planner" system is being developed and that the design of this system is underway. Originally, the customer had not included any requirement calling for prioritization of meeting participants. However, at some point into the design, the customer adds a requirement that participants to

```
STEP Booch_Process IS
  ACTIVITY:
    Nominal_Booch_Process;    -- The nominal Booch process.

  REACTIONS:
    REACT TO Change_Request ( request r ) BY
      Collect_Estimates_of_Impact ( r );
      Reestimate_Schedule_and_Budget;
      Notify_Customer_of_Impact_on_Schedule_and_Budget;
      if ( Decide_to_Proceed ) {
        Send_Event ( Change_Order ( r ) );
      }
      else if ( Decide_to_Modify_Request ) {
        modified_r = Negotiate_Modifications_with_Customer ( r );
        Send_Event ( Change_Request ( modified_r ) );
      }
    END REACT;

END Booch_Process
```

Figure 1: A Reaction Handler for the Booch_Process

meetings be given different statuses: essential, desirable, and optional. The algorithm to decide what is the preferred time for a meeting must be changed to accommodate this prioritization of the participants.

At this point, it is no longer desirable to continue with the design. The first thing that must happen is that the designers must analyze their portion of the design to determine if they would be impacted by it and estimate the amount of effort that would be required. This analysis of the impact of a change is shown in the reaction handler presented in Figure 1.

Note that the reaction is described in very high level terms. These terms could represent steps or procedures and be further specified in JIL or they could simply represent activities requiring a human to perform them. Most likely, they would be implemented as some combination where the program would provide the data management functionality required to assist the user, such as actually sending the notification to all the designers and collecting all the results. Analysis of the design and the impact reports would most likely be done by a manager or management team.

As another example of unpredictable activities, consider how errors in the process itself are identified. Each step of a process has preconditions and postconditions associated with it. Failure of any of these throws an exception. Steps also have resources (tools, hardware, humans) required for their completion. Failure to acquire necessary resources also throws an exception. Errors in the design are often recognized when trying to evaluate postconditions of a step that define successful completion of the step. For example, Figure 2 shows a postcondition and corresponding exception handler that requires classes to provide the methods that other classes depend upon.

Again, we see separation of event handling from the nominal process, allowing each to be reasoned about and understood separately. We also see here a situation in which the designer chooses how to resolve

```
STEP Identify_Relationships_Among_Classes_and_Objects IS
  ACTIVITY:
    Nominal_Relationship_Identification_Process;

  POSTCONDITIONS:
    Needed_Methods_Provided ( classes );

  HANDLERS
    HANDLE FAILURE OF Needed_Methods_Provided ( classes ) BY
      ALTERNATE ( Add_Method,
                  Modify_Method,
                  Modify_Dependency,
                  Remove_Dependency,
                  Abort );
    END HANDLE;
END Identify_Relationships_Among_Classes_and_Objects
```

Figure 2: An Exception Handler

the problem encountered. The **alternate** operator indicates that the designer can choose from among the listed alternatives to solve the problem. Note further that the postcondition protects the process from an integrity violation regardless of how that violation occurred. The violation may have resulted from an error in the definition of the object or the relationships, or from some other cause. The combined effect of the postcondition and exception handler is to take some unanticipated integrity-violating event, the effect of which is detected when the postcondition is evaluated, and to turn it into an anticipated event, the exception thrown by the postcondition failure, which can then be handled by propagating appropriate changes to the design so as to repair the violation.

5 Conclusions

We find the problems described here to be quite common in any complicated process. It is important to be able to separate the nominal process and the exceptional processes cleanly from each other in order that they be better understood. It is inevitable that any process that requires an extended period of time, such as a design process does, has a need for mechanisms to describe these events that are often expected, yet never predictable. We believe that a design process is not fully defined unless these unpredictable cases are also planned for.

We also believe that any complex process, including design processes, typically require more than one person to be involved. It is therefore necessary to define how those people should coordinate their activities to achieve the best results in the least amount of time. Our mechanisms for selectively sharing objects under development, of reasoning about resource requirements and their impact on scheduling, and tracking what individuals are doing provides us with abilities to extend process definitions with information to coordinate multiple people while leaving the underlying nominal process unchanged.

Acknowledgments

Rodion Podorozhny is responsible for design and development of the resource management system. Eric McCall is responsible for the design and development of the agenda management system.

References

[1] G. Booch. *Object-Oriented Analysis and Design with Applications*. Benjamin Cummings, Redwood City, CA, second edition edition, 1994.

[2] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes. *Object-Oriented Development: The FUSION Method*. Prentice-Hall, Englewood Cliffs, NJ, 1994.

[3] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. *Object-Oriented Software Engineering: A Use Case Driven Approach*. ACM Press, New York, 1992.

[4] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, NJ, 1991.

[5] Stanley M. Sutton, Jr. and Leon J. Osterweil. The design of a next-generation process language. In *Proceedings of the Joint 6th European Software Engineering Conference and the 5th ACM SIGSOFT Symposium on the Foundations of Software Engineering*. Springer-Verlag, 1997. To appear.

[6] R. Wirfs-Brock, B. Wilkerson, and L. Weiner. *Designing Object-Oriented Software*. Prentice-Hall, Englewood Cliffs, NJ, 1990.