Specification and implementations of interactive information systems

by ANTHONY I. WASSERMAN

University of California
San Francisco, California

ABSTRACT

User Software Engineering is a methodology supported by automated tools for the development of interactive information systems. The specification process decomposes the system into user-program dialogue, database definition, and formal and informal description of system operations. Evolution of the specification is supported by tools for rapid construction of prototype versions of the system, and the resulting specification is easily transformed into the programming language PLAIN. This paper gives an overview of the USE development process, illustrating it with a development dictionary example.
INTRODUCTION

The process of developing a software system may be divided into two steps: producing an accurate specification of what the system is to do, and implementing a system that meets that specification. Other activities typically associated with the software development life cycle may be viewed as supporting one or both of these steps. For example, analysis of system requirements makes it possible to write a better specification, and architectural design leads to program structure.

In practice, these activities are affected by time and resource limitations, organizational structures, inadequate tools, poor analysis, incomplete testing, and communication difficulties, leading to many of the well-known problems of software development and evolution.

The User Software Engineering (USE) project is concerned with the process of developing an interactive information system (IIS), a particular class of software system characterized by conversational access to data. Frequently, the users of an IIS are not experts in computing, and are given a predefined set of operations to use. Examples of such systems include airline reservation systems, bibliographic searching systems, decision support systems, and text editors.

METHODOLOGY OVERVIEW

The USE methodology combines the systematic approach to software development inherent to the life cycle approach, with effective user involvement in the specification process. We view creation of an accurate specification as being more difficult than implementation of a system from the specification. This situation is especially true with an IIS, where users may have a poor concept of their needs and a limited idea of the potential capabilities of a computer system. Thus, production of a functional specification requires extensive analysis and communication.

The USE methodology combines traditional activity- and data-modeling techniques with efforts to design the user-program interface. It creates a preliminary version of the dialogue at the earliest possible stage.

The specification of an IIS is seen to consist of three parts: the user-program dialogue, the database design, and the operations (transactions) associated with various user inputs. The interaction is described in a set of augmented state transition diagrams, each of which is termed a conversation. Various user inputs may cause state transitions, including the invocation of a "subconversation" (another diagram). Actions may be associated with a transition, so that all of the operations may be attached to transitions. The database is described as a set of normalized relations.1

The operations are described both formally and informally in the USE methodology to satisfy the different audiences for the specification. The informal approach is simply a short paragraph (two or three sentences at most) of narrative text, whereas the formal approach uses a formal notation employing preconditions and postconditions in conjunction with a description of the behavior of operations similar to that developed for Alphard.2 For those operations involving database access or modification, the database operations are shown in a data manipulation language.

The following list of steps for the USE methodology shows the emphasis on development of the specification:

1. Preliminary analysis—activity and data modeling, leading to preliminary informal specifications and identification of user characteristics
2. External design—user-program dialogue
3. Creation of a prototype of the user-program dialogue with revisions as needed
4. Completion of the informal functional specification of the system operations using narrative text
5. Preliminary relational database design
6. Creation of a functional prototype system, providing at least some, and possibly all, of the system's functions
7. Formal specification of the system operations using behavioral abstraction
8. Software design at the architectural and module levels
9. Implementation in PLAIN
10. Testing and verification

There is considerable flexibility in the application of these steps, and the methodology supports variations in which some of the steps are emphasized or omitted.

In the remainder of this paper, we first describe the specification process. We then show the RAPID/USE application development tool for the rapid construction of user program dialogues and interactive systems. We then discuss the structured programming language PLAIN, which can also be used for implementation of a system. We use a simple development dictionary system as an example.

DEVELOPING THE SPECIFICATION

A major hindrance to the analysis and specification of interactive information systems is that the user and developer must reach agreement on system capabilities and operation at a very early stage, often with little understanding on the user's part. The resulting system is then, at best, only partly satisfactory, necessitating an expensive process of evolution. Many engineering disciplines build preliminary models of pro-
posed products or systems. A similar approach of prototyping a system is taken in USE. If a prototype system can assist in reaching better user understanding, then there can be significant improvements in system quality and reductions in maintenance costs. The rapid construction and modification of prototype versions of the system are important aspects of the USE methodology.

We perform an initial analysis to identify the principal data objects and the operations upon them. This information is used to define a set of “structured operations” (transactions) visible to each user class, aiding both the formal definition of system properties, which are defined as abstract operations on objects, and the design of the user–program interface.

Analysis also includes the identification of user characteristics, such as user skills, user motivation and intelligence, and physical workplace constraints. These characteristics, when combined with information about output needs (volume; hard copy vs. “soft” copy), are essential to the system design process in general and to the dialogue design process in particular.

DESIGN AND MODIFICATION OF USER–PROGRAM DIALOGUE

In many respects, the user–program dialogue is the most critical aspect of an IIS, since that is what the user sees. Elegant and efficient implementations are useless if the IIS is difficult to use or does not meet the user’s needs. Accordingly, our next step is to define the user interface to the system for each identified user class.

The interface can take many forms, including multiple choice (menu selection), a command language, a database query language, or natural-language-like input. In all cases, however, the normal action of the program is determined by user input, and the program may respond in a variety of ways, including results, requests for additional input, error messages, or assistance in the use of the IIS.

At this early stage, the dialogue design is far from complete. Typically, only the major operations are identified, and the options for different operations may not be fully defined. Also, the first design effort may omit some needed operations, and rarely includes more than rudimentary error handling and help facilities.

In short, the initial dialogue design is seen as a starting point for a process of gradual refinement that is achieved through partnership between the developer and the user. The dialogue is represented with USE transition diagrams, an augmented form of state transition diagrams. Initially, we used the diagrams as the sole basis for communication between the developer and user community. While that worked successfully and we were able to show the nature of the interactive interface, we sensed that the users did not really have a very good idea of how the interface would actually behave. (We observed that few people would purchase an automobile without first taking a test drive.)

Accordingly, we sought to automate the USE transition diagrams. The primary intent of such automation was to be able to encode the diagrams quickly and to generate the interface so that the prospective user could use it. Another advantage would be the ability to encode the error-handling and on-line assistance parts of the more detailed diagrams so that users could gain experience with those aspects of the dialogue. In this way, the set of diagrams can be encoded, and a running prototype produced.

There are several other reasons for building such a prototype:

1. It enables the user to evaluate the interface in practice and to suggest changes
2. It enables the developer to evaluate user performance with the interface and to modify it to minimize user errors and improve user satisfaction
3. It facilitates experimentation with alternative interfaces and modification of interfaces
4. It gives the user a more immediate sense of the proposed system and thereby encourages users to think more carefully about needed and desirable characteristics

The prototype gradually evolves into a model of a usable interface, thus yielding a formal description of one aspect of the system specification: the user–program dialogue. Since the database aspect is specified by the data-modeling activity, and by subsequent refinement into normalized relations, the only remaining aspect is the set of operations.

EXAMPLE: DEVELOPMENT DICTIONARY

Some of the concepts of the USE methodology can be seen in the effort to design a static development dictionary to support the methodology, an activity presently underway. The idea of the tool is to support definitions of data elements, data stores, data flow, and processes that are identified during the modeling of USE. In addition to the insertion, deletion, and modification of entries of these different types, the development dictionary system should support the following user queries:

1. List all the data elements contained in a given data flow or data store
2. List all the data flows that contain a given data element or lower-level data flow
3. List all the data stores containing a given data element or data flow
4. List all the processes that input or output a given data element or data flow
5. Display in alphabetic order all entries of a given type
6. Display all entries whose names contain a given input string (partial match)
7. Display all undefined entries

Finally, the system should perform certain consistency checks, such as prevention of duplicate names, and restricted deletion of entries that are part of other entries.

Part of the user–program dialogue, encoded in RAPID/USE, is shown in Figure 1. This segment shows the “main” dialogue in which the user initiates the dialogue, the subconversation dealing with the required retrievals, and the retrieval subconversation for listing the data elements contained in a given data flow or data store. (This figure comprises about 50% of the entire dialogue, but only one action.)
From the collection of the Computer History Museum (www.computerhistory.org)

The informal specification and the executable interface gives is difficult to provide realistic output messages and impossible the user a good sense of what the system will be like. How­

to program the lIS functions. From a system construction

showing the dialogue and associated actions as a set of transi­

tion diagrams, accompanied by specifications of the actions

performed. The entire lIS may be specified in this manner,

all of the actions of the system and the point at which they are

performed. The entire IIS may be specified in this manner,

including specifications of the actions and the database design. The user may review these diagrams and see the valid inputs and the actions that occur as a result of those inputs. This activity yields an informal specification of the system, along with the prototype of the user-program dialogue previously developed.

ADDITIONAL FUNCTIONALITY TO PROTOTYPES

The informal specification and the executable interface gives the user a good sense of what the system will be like. How­

ever, with only dialogue management in the prototype tool, it is difficult to provide realistic output messages and impossible to program the IIS functions. From a system construction standpoint, the goal is to have a tool that permits the rapid construction of an IIS that performs many of the IIS functions.

A key observation was that many of the operations involve database access and modification, so the desired functionality could be provided by combining dialogue management with a database management system. One of the tools used for several purposes in the methodology is the TrolllUSE relational database management system, so the linkage mechanism is designed to include routines written in the TrolllUSE data manipulation language or in any one of a variety of programming languages (PASCAL, C, FORTRAN, or PLAIN).

This tool, called RAPID/IUSE, permits a rapid implementation of the IIS specification with a notation that provides a close match to the specification method itself. Output mes-
sages are associated with nodes, and actions may be associated with arcs. The message facility is screen-oriented, so that full cursor control is available along with output.

As with the dialogue portion, the prototype can be continuously modified, gradually providing the essential functions of the system. The features desired by users in the prototype affect the specification. In short, the prototype system is used to refine the user's perceived requirements, so that less effort will be required to carry over into the modules.

The production programming language is PLAIN, a language derived from PASCAL to support both the concepts of systematic programming and the needs of interactive information systems. PLAIN provides excellent support for abstraction and modularity through an abstract data-type mechanism, parameter passing by input and output, and control over access to global and external data objects.

Most of the innovations in PLAIN support the needs of interactive information systems. PLAIN provides strings and relations as built-in data types, along with appropriate facilities for data definition and manipulation. In addition to string manipulation, strings may be compared to patterns and sets of patterns, with the ability to take action based on the result of pattern-matching and comparison operations. PLAIN provides a relational algebra-like set of operations on relations, as well as the ability to do tuple processing and to assign the result of database operations to temporary structures (markings). Finally, PLAIN provides a powerful exception-handling facility to enhance the reliability of interactive programs.

As a result, implementation of the specified IIS is straightforward in PLAIN, since the primitives of the specification method, including strings, patterns, relational databases, transactions, and pre- and postconditions, have corresponding primitives in PLAIN. Furthermore, the encoding of pre- and postconditions as assertions makes it easier to verify the correctness of the implemented system.

While one could implement the system in another programming language, the USE tools were designed to be used together, so that PLAIN provides the best possible language for transforming the specification into a running system. It can be seen that the string-handling and pattern-matching features support the construction of the user-program dialogue directly from the transition diagrams, and that the relational database design similarly can be programmed directly.

The portion of the development dictionary system shown in Figure 1 has been written in PLAIN and is shown in Figure 4. (Access to the relations is omitted, but the four relations defined in Figure 2 must be declared in an external

Figure 3—Troll/USE script for action invoked by RAPIDIUSE

[Troll script for finding components of data flows and data stores]

if exists (data_flow {$0}) then
begin
if exists (data_store {$0}) then
begin
$1 := 0; print data_flow {$0}.flow_components;
end else
begin
$1 := 1; print data_store {$0}.store_components;
end;
end;
end;
The resemblance between the two versions is apparent. Indeed, this program structure is characteristic of many interactive information systems written in PLAIN. The main program consists of a loop in which an input string is read and compared to a set of patterns, causing a multway dispatch to the appropriate procedure for the input. The procedure corresponds to a “structured operation” or transaction. One possible input terminates the program, causing exit from the loop. The declarations in the main program include an external procedure, used to name and bring into the environment of the program.

![Figure 4—PLAIN code for portion of USE development dictionary](image-url)
CONCLUSION

The USE methodology provides a series of steps to support the process of creating an IIS, from its original conception through implementation, verification, and evolution. The methodology is supported by a unified support environment, including RAPID/USE, Troll/USE, and PLAIN. In addition, other tools exist to assist with project management, including TBE, a relation editing and browsing tool, and the Integrated Development Environment, a version control and configuration management tool that guides the developer in the use of the other tools. All of these tools have been designed and developed to be used in the UNIX environment, taking advantage of many of the underlying UNIX tools. Most of the USE tools are available for a handling charge through the UCSF User Software Engineering distribution. (Commercial versions and support for the USE are provided by Interactive Development Environments, Inc., of San Francisco.) Future work will make these tools available on personal development systems, leading to a User Software Engineering machine.

REFERENCES